The FBUG Debugger User Guide

Version 1.12

1. Introduction

The fbug debugger is an interactive system level debugger for int**e**nt®. fbug itself is a text mode debugger, but it can be used with a graphical user interface, referred to as fbugwin. The fbug debugger has been designed to perform such tasks as debugging applications, device drivers and other pieces of int**e**nt code. At present, the fbug debugger can be run within the following environments:

- Windows NT/95/98/2000
- Linux (with X11 for fbugwin support)

The debugger can be of use in the following areas:

- Examination and modification of registers
- Single stepping through code
- Incorrect parameters *
- Identifying stack overflows*
- Setting and clearing multiple breakpoints
- Expression evaluation (VP and Native)
- Identifying misaligned loads and stores *

* These features are only available with the *developt* image.

It also supports a limited form of 'post-mortem' analysis. This occurs when a debugger is connected to a system that has crashed, and used to perform a retrospective analysis of the causes of the failure. In other words, this facility is available whilst using remote debugging when the target has the debug stub started. If it traps it is possible to attach fbug after this has happened (it is not possible to attach fbug to a local session after it has crashed, though fbug can be attached to a crash which has hung). Limited support for debugging programs written in C, C++ and the Java™ language is available.

1.1 Overview

fbug provides two discrete methods for communicating with the int**e**nt session being debugged. In either eventuality the current directory must be the int**e**nt root prior to starting. Any source files must be accessible in the host file system.

- In local debugging, the int**e**nt session must be running on the same machine as fbug. This method is the default when no method specifying option is used. Local debugging can be used to attach to a running process on Windows or Linux by using the same decimal process id as the argument.

- In remote debugging, fbug communicates with the debug stub in the int**e**nt session using the int**e**nt debug protocol. The -s option enables remote debugging using a serial port, while the -p option enables remote debugging using a specified pipe. The –r option enables remote debugging over a socket connection. The int**e**nt session with a debug stub must be started independently, in a manner that is platform dependent. The -i option can be used to send a command to start the target along the same connection as is used for remote debugging. When using remote debugging, no arguments other than option switches should be specified in the fbug command line. It is possible to use remote debugging to debug an int**e**nt session on the same Linux machine. For more information on this, please see the html documentation.

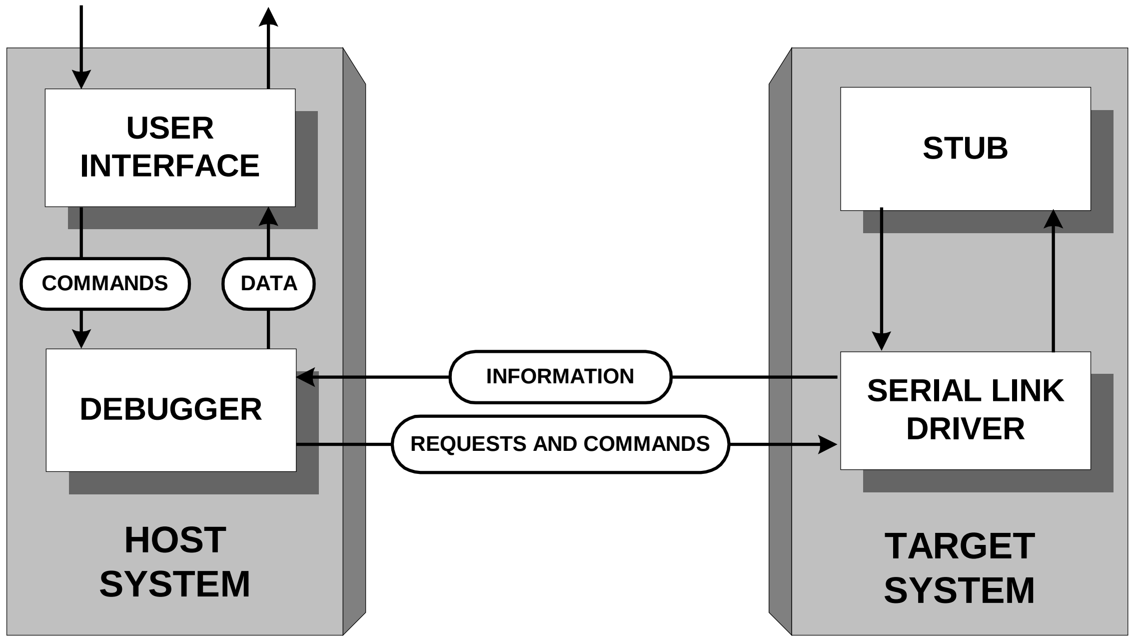

Figure 1. Example of Remote Debugging over a Serial Connection.

For either type of debugging, fbug must be run from the intent root on the host machine, with any source files accessible in the host file system - fbug cannot read intent file systems.

1.2 Invoking the fbug Debugger

The fbug debugger may be invoked either through a command line or through a desktop shortcut (on Windows). This is described in section 1.4.

1.2.1 Creating a Windows® Shortcut

Under Windows® fbug should be run by modifying the icon for starting intent (assuming that a shortcut has not already been provided by the InstallShield® process). For example, if it says:

```
d:\work\intent\sys\platform\win32\elate.exe  -Bdevelopt.img
```

This should be changed to:

```
d:\work\intent\sys\platform\win32\fbug.exe
d:\work\intent\sys\platform\win32\elate.exe -Bdevelopt.img
```

Before starting, the current directory should be the intent root. This is achieved by setting the "Start In" part of the shortcut.

The above example assumes local debugging, although a shortcut for debugging a remote target can be created in a similar example (see below).

```
d:\work\intent\sys\platform\win32\fbug -s com1,115200 -T
d:\work\intent\sys\platform\myplatform\myimage.sym
```

Note the use of the *developt* image to make use of the checking translator. The checking translator results in tools that are much larger than if the normal translator is used, as a result of the extra debug code that is inserted into the tool, and performs checking of potential stack overflows, parameter mismatches and unaligned memory access.

Invoking fbug will cause a window to appear on the screen upon which fbug commands can be entered.

1.2.2 Command Line Invocation

Command line invocation for the fbug debugger is as follows:

```
sys/platform/linux/fbug [<fbug_options>] sys/platform/linux/elate
[<elate_options>]
```

In this case the first <arg> is the name of the intent driver program, and subsequent <arg>s specify options and arguments to pass to the driver program.

Under Windows®, the following should be entered at a command prompt:

```
sys\platform\win32\fbug [<fbugwin_options>]
C:/intent/sys/platform/win32/elate <elate_command>
```

(assuming C:/intent is the root directory).

1.3 Entering fbug

fbug is entered, showing the current execution position, when one of the following happens in the target:

- Execution is interrupted by the user (see below);
- A breakpoint (or watchpoint) is hit;
- An exception is encountered. *

* This also applies under remote debugging when intent starts up or shuts down. This requires the –u option on the debug stub built into the target image,

Execution can be interrupted by pressing Ctrl-C in fbug, except when using local debugging on Linux, in which case a SIGINT must be sent to the target intent session. When intent is running on a terminal or an xterm, this can be done by pressing Ctrl-], otherwise the Linux kill or the killall command can be used. . This causes the intent system to be stopped in either case and fbug entered. It prints a brief description of the problem (e.g. a hard coded breakpoint), a register dump, the name of the tool where the problem occurred and the offset into that tool, and the source line where the problem occurred (only if the tool was assembled with the -g option). If source is not available a disassembly of the instruction which caused the problem can be displayed.

When using local debugging on Windows, pressing Ctrl-C may not immediately interrupt execution if the intent session is idle. In this case, move the mouse over the intent window.

When using remote debugging, pressing Ctrl-C will not interrupt execution unless the -t option was specified when starting the debug stub, or if the whole intent session has crashed in a way which stops the debug stub responding to communication from fbug. In these cases, fbug will exit after a timeout of a few seconds. When execution has stopped and fbug is responding to input, fbug has a selected process and a selected stack level. All expression evaluation and CPU register evaluation is performed in the context of the selected process and the selected stack level. Each time execution stops, the current process becomes the selected process, and the top stack level in that process becomes the selected stack level.

1.4 Invoking the fbugwin Graphical User Interface

To start up fbugwin and a particular intent image together, the full path of the intent executable, and the name of the image file should be passed as parameters to the command sys/platform/win32/fbugwin.

When fbugwin is started, it displays a window containing a main client area (which displays disassembly and source code when the target is running), and optionally some other windows, displaying various information gathered from the target (see information areas). If the "-g" option is specified on the command-line, intent is started immediately.

fbugwin provides facilities for starting, stopping and interrupting the target, examining registers and memory read from the target, changing settings, evaluating expressions, etc. It should be noted that most of the facilities provided by fbugwin can only be used when the target is "stopped", i.e. when it has crashed, hit a breakpoint or been interrupted. fbugwin may start one of a selection of intent sessions whose parameters have previously been set up (for example, *develop*, *developt*, a target board connected to using a serial line, another machine on the Internet etc), exit the intent session, and then restart and continue debugging (without quitting fbugwin, and possibly using a different target setup).

Before fbugwin can be used for debugging, a "target" must be configured through the target menu. See section 7 for more information upon this.

Under Windows, a shortcut should be created whose target is set as follows:

```
"C:\intent\sys\platform\win32\fbugwin.exe [options]"
```

The "Start in" directory for the shortcut should be the root directory (C:\intent).

Under Linux, fbugwin should be started from the intent root directory using a command such as

```
"sys/platform/linux/ix86/fbugwin [options]"
```

1.4.1 fbug Command Line Options

-d	Create a file fbug.log logging various internal workings of fbug.
-h	Output a help message describing these options. No debugging session is started.
-r *<hostname>[,<port>]*	Enable remote debugging with the serial debug protocol using a TCP socket to <hostname> on port or service <port> (default 1421). With this option, no <arg>s should be specified.
-i *<filename>*	For remote debugging only, send command(s) to the target when starting up. The contents of file <filename> are sent to the target, with any lf or cr/lf sequences converted to cr (as if the lines of the file had been typed on a terminal). If <filename> is -, then input is read from stdin and sent to the target, until either an eof is received on stdin, or the target first sends an intent debug protocol packet. Because fbug also outputs any non-protocol data it receives, this allows fbug to act as a primitive terminal until intent is started on the target.
-s *<name>[,<baud>]*	Enable remote debugging with the intent serial debug protocol using the serial port <name>, for example /dev/ttyS0 on Linux or com1 on Windows. The serial port is set up as follows:

	• no handshaking; • 8 bits, 1 stop bit, no parity; • baud rate of <baud>, defaulting to 19200. With this option, no <arg>s should be specified.
-p *<inname>[,<outname>]*	Enable remote debugging with the int**e**nt serial debug protocol using a file or device. <inname> is the filename for receiving data from the target; if <outname> is specified it is the filename for sending data to the target, otherwise <inname> is used for both directions. This option does no tty setup so cannot be used on a serial/tty or pseudo tty device. It is typically used with pipes to allow debugging of an int**e**nt session on the same machine using the int**e**nt serial debug protocol. With this option, no <arg>s should be specified.
-t *<symfile>*	Load sysgen symbol file. This avoids the need for fbug to read the tool list from the target at startup, as long as the debug stub sends an "int**e**nt starting" message (which is enabled by the debug stub's -u option).
-j	Pass possible Java™ exceptions on, without stopping in fbug. Depending on the CPU and platform, a Java™ NullPointerException and/or DivideByZeroException may be implemented as a hardware CPU exception. The -j option causes fbug to pass such exceptions on to the application rather than trapping them itself.
-T *<symfile>*	Load sysgen symbol file, and assume that this reflects complete tool list even if no "int**e**nt starting" message is seen. This avoids the need for fbug to read the tool list from the target at startup, but means that it will not know about tools which had already been dynamically loaded or dynamic atoms which had already been added by the time fbug started. Thus the option is used to stop fbug reading the tool list when no -u option was specified to the debug stub, or when fbug is attached to an already running target.
-x	Redirect fbug input/output. Note that the xfbug script (sys/platform/linux/xfbug.html), performs all the complicated redirection. On Linux with local debugging only, this option causes fbug to use handles 3, 4 and 5 for its input, output and errors, instead of the normal 0, 1 and 2. int**e**nt's input/output continues to use 0, 1 and 2. If used in conjunction with piping in shell commands, this option can be used to cause fbug to appear in a different terminal from int**e**nt.
-noguess	Disable guessing in stack traces. When fbug displays a process's call stack, it uses a heuristic to fill in the stack levels for tools that don't have debug information. Although this is generally useful, it can give false entries in the stack trace and can upset some targets, which don't like being asked to read arbitrary areas of memory. This option disables the heuristic, so that no stack trace will be shown for tools that don't have debug information.

1.4.2 Fbugwin Specific Options

-r *<intent root directory>*	This option sets the int**e**nt root directory to the specified directory. This is used for the following purposes: • In local debugging, this sets the root of the int**e**nt tree, used to find the int**e**nt executable to run. • In both local and remote debugging, this is used as the default path to search when looking for source files (note that fbugwin cannot

	access files within intent filesystems).
-g [<targetname>]	By default on startup, fbugwin does not start intent, it simply initialises and waits for the user to tell it to do so. This option specifies that intent should be started immediately when fbugwin starts up. If no target is specified, then the default target will be started. Other targets can be started using the unique name specified on configuration of the target. All subsequent parameters are passed through to fbug without further interpretation. This allows fbug and intent options (fbugwin makes no distinction) using the same option letters as specified here to be passed through.
-l [<language>]	Specifies the language setting for the fbugwin user interface. This overrides the value oft the LANG environment variable, which controls the default language setting.
-z	Start in an iconified state. fbugwin will deiconify when the target traps.
-h	Display a usage message and exit.
--	All subsequent parameters are passed through to fbug without further interpretation. This allows fbug and Elate options (fbugwin makes no distinction) using the same option letters as specified here to be passed through.

1.5 Remote Debugging

An example command line for remote debugging would be as follows:

```
fbug -s com1,115200 -T myimage.sym
```

This would be amended as follows, in order to make use of fbugwin:

```
fbugwin -s com1,115200 -T myimage.sym
```

This would debug over com1 at 115200 baud where myimage.sym is the sym file created by sysbuild (using its -t option).

1.6 Generating Debuggable Tools

A block of debug material exists for each intent tool. This block is an optional part of the tool structure, and plays no part in the ordinary running of the system. Indeed, intent supplies a special utility for stripping the debug block from each tool, so as to render the system more compact. A tool that is to provide debug information, however, must be generated with this block attached.

It is customary for debuggable tools to be generated before the debugger is invoked. However, this is not invariably the case.

The command used to invoke the assembler is "asm". If the tools being assembled need to be debuggable, the option "-g" should be added to this command. Also note that it may be helpful to use the –v option to provide the user with information about the tools generated. Using –g will in turn allow disassembly of a tool with the dis –s command, which shows the source that goes with the corresponding VP byte codes. For example:

```
$asm –g demo/example/hello

$dis –s demo/example/hello.00
```

Note that when assembling the .asm extension is optional, but when disassembling the .00 extension is required, as there may be versions of the tool with different extensions, e.g. hello.00 hello.15. Note the disassembler can be used to disassemble tools with any supported extension:

```
$dis –s demo/example/hello.15
```

If the VP tools are being compiled from C or C++, the option "-g" must be added to the command "vpcc" when invoking the compiler. In such cases, the compiler will inform the assembler that the tools are to be generated in a debuggable form.

To obtain debug info for tools in the image the sysgen -s option should not be used (alternatively the -nos sysbuild option can be used), instead using the sysgen -m option (the default in sysbuild) to provide a map file, and then using the fbug -t/-T option to load it. In the case of dynamically loaded tools, the only option is to allow them to contain debug info by not using the translator flags (in the .jtrans and .trans lines in the sys file) which instruct the translators to strip off debug info.

2. Local Debugging Architecture

fbug is essentially a hosted debugger. This means it relies on services provided by an underlying operating system to perform certain tasks. These tasks include:

- Creating a new process
- Waiting for a debug event
- Looking at debuggee’s memory
- Looking at debuggee’s registers
- Starting debuggee running

In local debugging, the int**e**nt session must be running on the same machine as fbug, and fbug uses the host OS's native debugging facilities.

3. The fbug Interface

As has previously been stated in this document, when fbug is entered and execution interrupted, it will display disassembly or source code. fbug has a full screen textual interface. Initially, as in the example below, only one window will be visible, however the interface normally consists of multiple 'windows,' separated by horizontal lines. Each window has a title line above it, except the topmost window, which is always the main source and disassembly window. The bottom line of the screen shows either a message or the current menu. If there is a message and fbug is in a state where it is responding to input (i.e. the target has stopped), then pressing any key replaces the message by the current menu, in addition to any other action the key has.

Within a window, the cursor keys move the cursor, scrolling when necessary. Certain windows are able to scroll sideways as well as up and down. The Window/Switch command (shortcut F6) switches the cursor into the next window down, or back to the top window if already on the bottom window. The Window/Close command (shortcut F4) closes the current window. The Window/Resize command is used to resize windows. On Linux, when executing fbug and the intent session in the same terminal, the fbug screen will be corrupted by any output from the intent session. When fbug has been re-entered, its screen can be refreshed by pressing ^L or ^R.

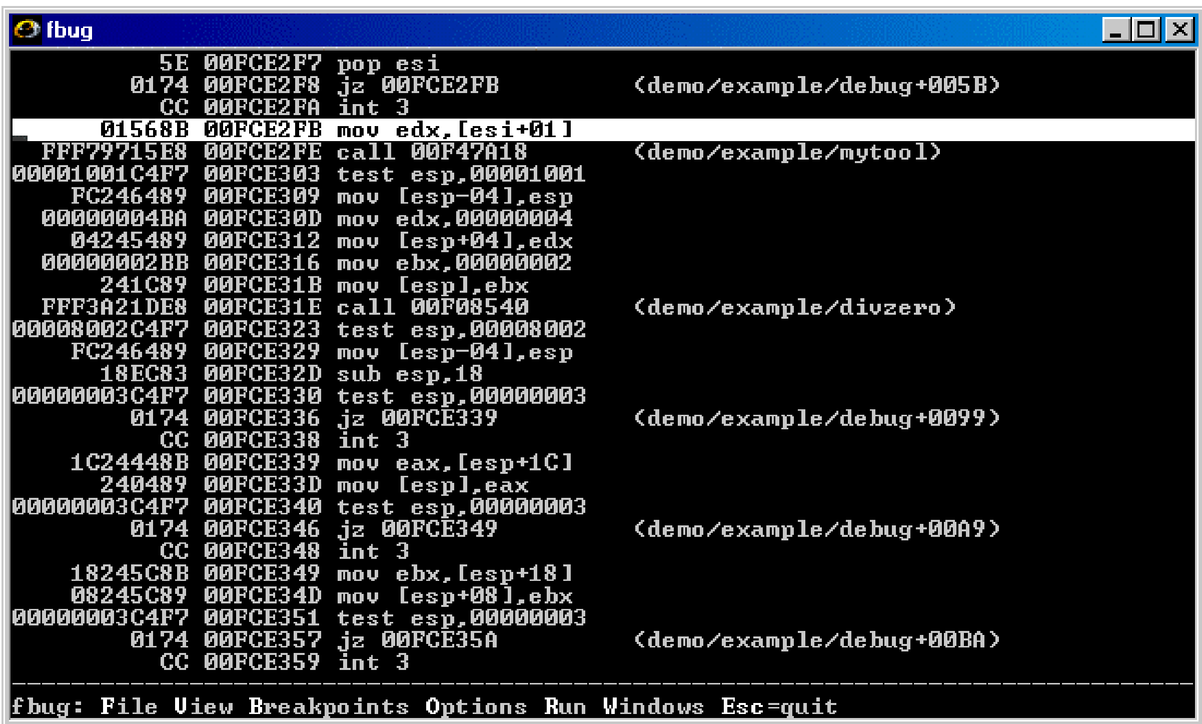

Figure 2: Window showing the initial menu.

Figure 2 shows the initial menu with which a user will be presented, assuming that no message is being displayed. At present, there are six initial menus that the programmer may make use of. These are as follows:

- File
- View
- Breakpoints
- Run
- Windows
- Options

Any of these menus can be selected by pressing the key corresponding to the option's letter (which should be displayed on the screen in highlighted text).

From the initial menu, a further set of options is available:

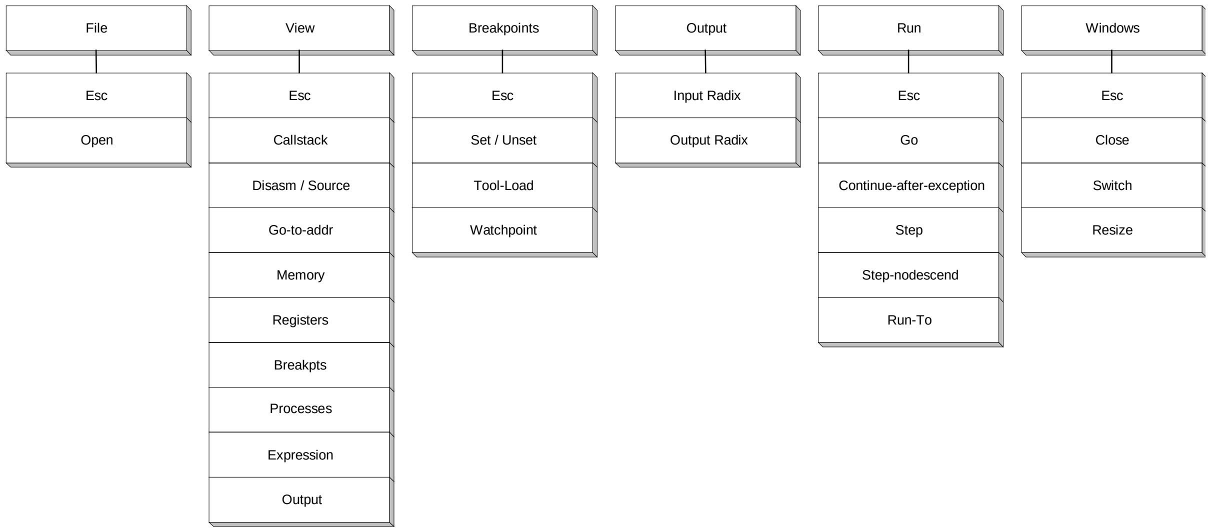

Figure 3. Text Debugger Options

3.1.1 Shortcut keys

Many commonly used commands have shortcut keys, which can be used from any menu:

- F2: View/Registers: view CPU registers
- F3: View/Disasm-source: switch list window between source and disassembly
- F4: Window/Close: close current window
- F5: Run/Go: continue execution
- F6: Window/Switch: move cursor to next window
- F7: Run/RunTo: continue execution up to cursor position
- F8: Run/Step: step statement/instruction
- F9: Breakpoint/Set-unset: set or unset breakpoint
- F10: Run/step-Nodescend: step over statement/instruction
- F11: View/Expression: evaluate an expression.

3.1.2 List window

The topmost window, which is present when fbug is entered for the first time, is the list window. If source is available, this either shows source, or disassembly interspersed with source lines may be displayed. If not, only disassembly is available. It is not possible to close the list window. The current instruction (in the selected process and stack level) is shown in inverse video; any source line or disassembly line with a breakpoint set is shown highlighted.

The View/Disasm-source command (shortcut F3) allows the user to toggle between source code and a mixture of disassembly and source (if available). The View/Go-to-address command moves to a disassembly at the requested address. The Breakpoint/Set-unset command (shortcut F9) sets a breakpoint at the cursor position, or removes breakpoints if there are any.

When intent traps or hits a breakpoint or stops executing and goes into fbug for any other reason the current location within in the program is shown in inverse video. On Windows, most of the fbug text is shown as grey on black, so that the line of source or disassembly is shown as black on grey.

It is then possible to change the selected stack level by moving around in the callstack window and hitting enter. This then shows where in the program the user was located at that stack level. For example, if stack level 0, the topmost one in the callstack window is the current location, then stack level 1 is where execution will return to at the next ret instruction, stack level 2 is where execution will return to at the next ret after that, etc. It is possible to see the chain of qcalls, goss or ncalls that led to the current point of execution, and where each of these calls is located.

Where a stack level's execution point is not in a tool with debug info, fbug uses a "guessing" algorithm to find the next one down (higher numbered). This finds it, but may result in some additional false levels appearing in between.

Similarly, it is possible to change the selected process using the Processes window, in order to see where execution was in other processes.

If either the selected process or selected stack level is changed, the list will then move to show where the user was at that stack level in that process, with that process and that stack level's PC (only available if showing disassembly rather than source) in inverse video as above.

3.1.3 Memory Window

The View/Memory command causes a memory window to be created, displaying memory at the requested address. If the requested address is variable, then the actual address of the memory changes when execution stops if the value of the address expression changes. The data is shown as separate bytes in hexadecimal. Note that bytes are shown in VP order; when the target has a big-endian CPU, bytes within a 4 byte word are reversed.

It is not possible to change the contents of a memory location within a memory window. Instead, use the View/Expression command to evaluate an expression that assigns a value to memory. Multiple memory windows can exist. Within a memory window, it is possible to scroll through the whole address range.

3.1.4 Registers Window

The View/Regs command (shortcut F2) creates a registers window, or closes it if it already exists. The registers window shows the values of all the CPU's native registers. Integer registers are shown in hexadecimal. It is not possible to change a register within the registers window. Instead, use the View/Expression command to evaluate an expression that assigns a value to a register. There can only be one registers window.

3.1.5 Callstack window

The View/Callstack command creates a callstack window, unless one already exists. This shows the call stack in the selectedprocess with the most recently executed level, level 0, at the top. The selected stack level is shown in inverse video. Moving the cursor to another stack level and hitting <enter> causes that stack level to become the selected one, showing the corresponding location in the list window. There can only be one callstack window.

3.1.6 Process window

The View/Processes command creates a Processes window. This shows the int**e**nt process, with the current one at the top. The selected process is shown in inverse video. Moving the cursor to another process and hitting <enter> causes that process to become the selected one, showing the corresponding location in the list window and its callstack in the callstack window if any. There can only be one processes window.

3.1.7 Breakpoints window

The View/Breakpoints command creates a breakpoints window, which shows all the currently set breakpoints. Within this window, the Breakpoint/Set-unset command adds or deletes a breakpoint at the cursor position, and hitting <enter> moves the list window to the position of the breakpoint. There can only be one breakpoints window.

3.1.8 Query window

The query window contains expressions to monitor and be updated each time fbug is re-entered (at the end of a step, or at a breakpoint, or at an exception). The View/Expression command allows an expression to be added to the query window by hitting <enter> after it has been evaluated. New expressions are added at the top of the window and it may be necessary to scroll down to see other expressions.

Certain types of expression represent a "container" which contains further items, for example a pointer (containing the item it points to), a structure or object (containing its members), or an array (containing its elements). Such an expression in the query window can be expanded, showing its contained items, by pressing <enter> with the cursor on that line. Pressing <enter> again contracts it, hiding its contained items. There can only be one query window.

3.1.9 Output Window

The output window contains text output by the intent session in these ways:

- ktrace output on Windows local debugging, when intent is configured to send ktrace output to OutputDebugString (using the driver's -t option);
- Any other text sent to OutputDebugString on Windows local debugging, for example the DLL relocation messages at startup on NT 4;
- ktrace output on remote debugging, when the debug stub's -k option is used;
- Any non-debug-protocol text sent to the port being used by the debug stub on remote debugging, including ktrace output when the debug stub's -k option is not being used and the PII debug stub (which uses *sys/pii/odata*) is being used.

The output window is created the first time output is received. It can be closed in the normal way by the Window/Close command, and (re-) opened by the View/Output command.

fbug has a buffer to remember more lines of output than can be shown in the window, and the text can be scrolled. However, when the target is running and output is received, the window moves back to showing the most recent text.

3.2 Menus and commands

The current menu is shown on the bottom line of the screen (unless a message is being shown there). Commands and other menus are accessed by pressing letter keys; <alt> should not be used with the letter.

- Quit

The quit command prompts for confirmation, and then quits fbug. In local debugging, this also kills the intent session being debugged.

3.2.1 File Menu

- File/Open

This command prompts for a filename and opens the requested file in the list window. It is then possible to set breakpoints in that source file, even if the tool has not yet been loaded.

3.2.2 View Menu

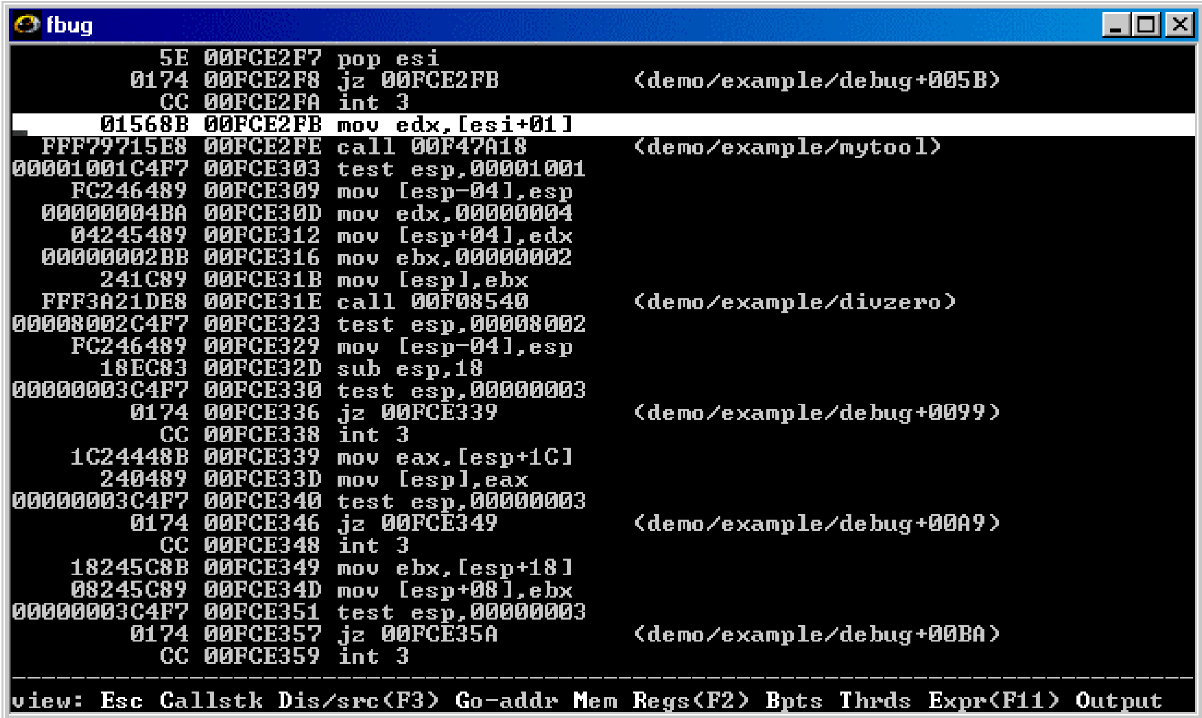

Figure 4. The View Menu, beneath list window shoring disassembly

- View/Callstack

This command creates a callstack window, or moves the cursor to it if it already exists.

This window shows the call stack in the selected process, with the most recently executed level, level 0, at the top. The selected stack level is shown in inverse video. Moving the cursor to another stack level and hitting <enter> causes that stack level to become the selected one, showing the corresponding location in the list window. There can only be one callstack window.

- View/Disasm-source

This command switches the list window between source mode and disassembly mode, as long as the mode being switched to is available. Source mode is not available if the disassembly being displayed is outside any tool or is in a tool with no source line debug information. Similarly, disassembly mode is not available if the source file being displayed is not used by any currently loaded tool with source line debug information.

- View/Go-to-address

This command prompts for an address expression, and then switches the list window to disassembly mode and moves the cursor to the address given by the expression.

- View/Memory

This command prompts for an address expression, and then creates a new memory window displaying memory at that address. The expression must not contain an assignment. If the expression is variable (i.e. it contains a variable, register or memory load), then it is re-evaluated, changing which area of memory is displayed, when fbug is re-entered after executing some target code, or when the selected process or stack level is changed. The data is shown in the form of separate bytes in hexadecimal. Note that bytes are shown in VP order; when the target has a big-endian CPU, bytes within a 4 byte word are reversed. It is not possible to change memory within a memory window. Instead, use the View/Expression command to evaluate an expression that assigns a value to memory. Multiple memory windows can exist. Within a memory window, it is possible to scroll through the whole address range.

The default input radix (which can be set from the options menu) governs how memory addresses are interpreted (e.g. as decimal or hex values).

- View/Registers

This command creates (or closes if it already exists) the registers window, which displays the CPU registers. This shows the values of all the CPU's registers. Integer registers are shown in hexadecimal. It is not possible to change a register within the registers window. Instead, use the View/Expression command to evaluate an expression that assigns a value to a register.

There can only be one registers window.

- View/Breakpoints

This command creates (or moves the cursor to if it already exists) the breakpoints window, which shows the currently set breakpoints and allows you to remove breakpoints. This shows all the currently set breakpoints. Within this window, the Breakpoint/Set-unset command deletes the breakpoint at the cursor position, and hitting <enter> moves the list window to the position of the breakpoint. There can only be one breakpoints window.

- View/Processes

This command creates a processes window, or moves the cursor to it if it already exists. This shows the intent processes, with the current one at the top. The selected process is shown in inverse video. Moving the cursor to another process and hitting <enter> causes that process to become the selected one, showing its PC location in the list window and its callstack in the callstack window if any.
There can only be one processes window.

- View/Expression

This command prompts for an expression to evaluate, and then shows its value at the bottom of the screen. By evaluating an expression containing an assignment, this command can be used to set a variable, register or memory location to a value.

An integer value is output in a format determined by the default output radix, which is initially decimal but can be set using the Options/Output-radix command. This can be overridden using a format specifier, consisting of a '/' character and a single letter at the start of the text. The valid format specifiers are:

- `/d`: decimal;
- `/x`: hexadecimal;
- `/o`: octal;
- `/t`: binary (base **t**wo);
- `/c`: character;
- `/a`: address (i.e. as if it is a pointer).

For example, to evaluate the sum of the registers eax and ecx, and then display the result in binary, enter the expression '/t eax+ecx.'

In address format, if the address falls within a tool, the tool name and offset are also displayed. In decimal, hexadecimal and octal formats, if the integer value is the value of an atom, the atom name is also displayed.

The default output radix or format specifier affects only integer values; pointers are always output in address format and floating point values are always output in decimal floating point format. After the expression text has been entered, fbug displays the result. If the expression does not contain an assignment, fbug then waits for a keypress:

- <enter>: the expression is added to the query window;
- <esc>: the expression is not added to the query window;
- Other: the expression is not added to the query window, and the keypress is processed in the normal way.

- View/Output

This command (re-)opens the output window, or moves the cursor to it if it already exists.

3.2.3 The Breakpoints Menu

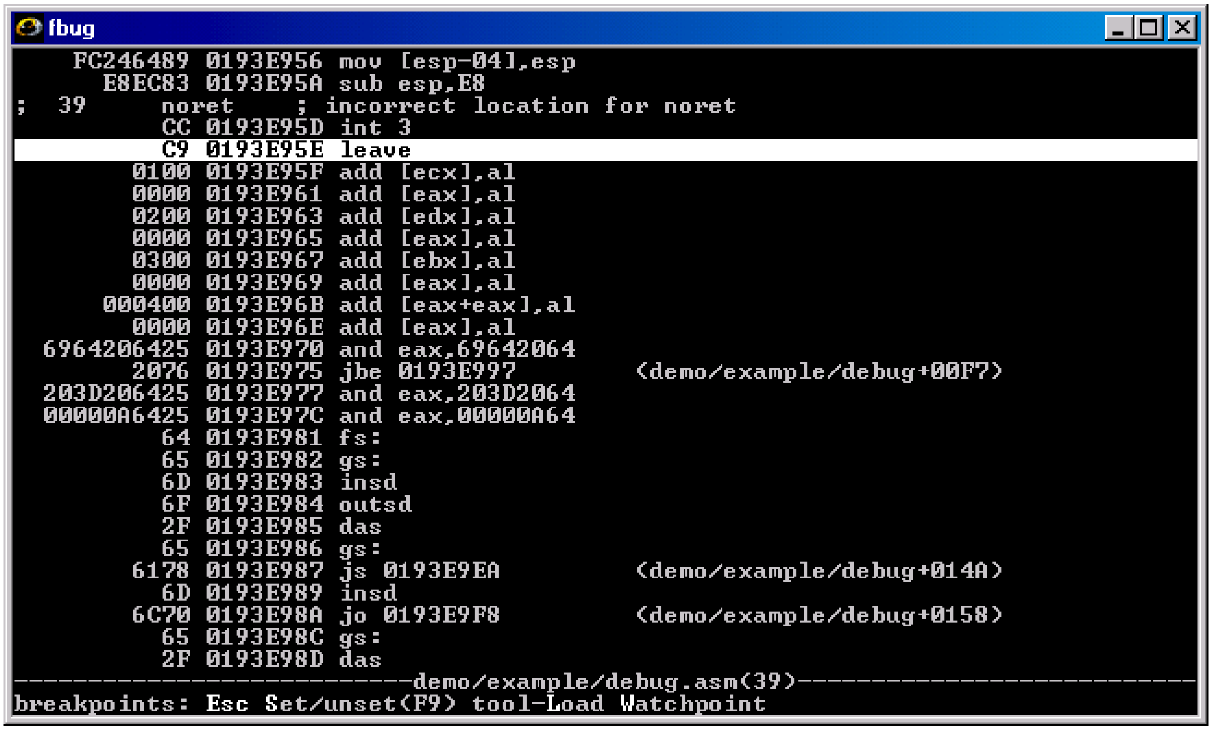

Figure 5. The Breakpoint Menu

- Breakpoint/Set-unset

If the cursor is in the list window, this command sets a breakpoint at the cursor location, or removes a set breakpoint at the cursor location.

It is allowed to set a breakpoint in a source file, which is not referenced by any currently loaded tool. A breakpoint will be hit at the loading or unloading of the tool, and on loading a breakpoint will be set at the corresponding instruction in code. When remote debugging is in use, the toolname is extrapolated from the name of the source file. If this is unlikely to locate the correct tool, a tool load breakpoint should be set manually via the breakpoints menu.

Setting a breakpoint in a source file when the relevant tool is loaded also causes a breakpoint to be set which will be hit if the tool is unloaded. It does this so it knows it should remove the breakpoint when the tool is unloaded. Note that having any such breakpoints set causes tool loading to slow down.

- Breakpoint/tool-Load

This command prompts for a tool name, and then sets a tool load breakpoint. This causes execution to stop when a tool of that name is loaded or unloaded. Having any such breakpoints set causes tool loading to slow down.

- Breakpoint/Watchpoint

This command prompts for an expression, and then sets a watchpoint. If the text entered starts with an equals sign '=', the watchpoint causes execution to stop when the value of the expression (without the initial '=') changes. Otherwise, the expression is expected to be a condition and execution stops when the value of the expression changes to non-zero. Watchpoints can only be set where the hardware and the debug stub (for remote debugging) support them. Currently they work only on the ix86 CPUs; watchpoints with a total of up to four memory loads can be supported.

See also: View Menu/Breakpoints

3.2.4 The Run Menu

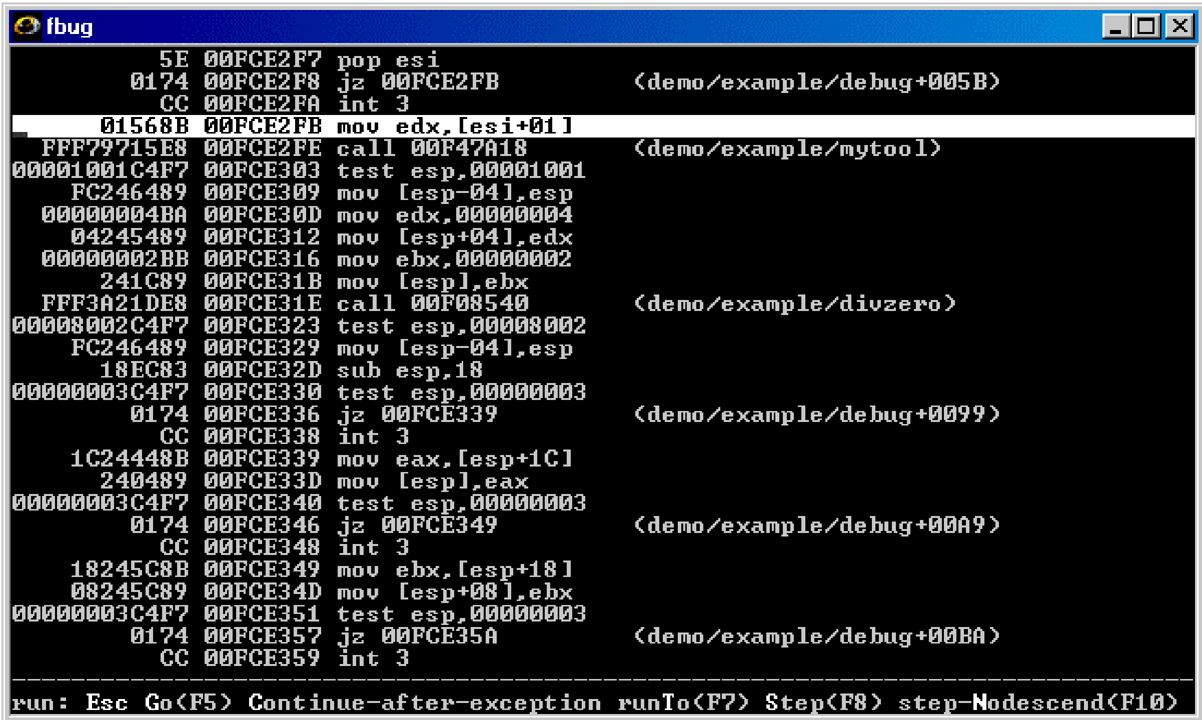

Figure 6. The Run Menu

- Run/Go

This command causes the target to continue executing, if possible.

- Run/Continue-after-exception

This command causes the target to continue executing, but if fbug was entered due to an exception, this exception is passed on to intent for it to process.

- Run/Run-To

If the cursor is in the list window, the callstack window or the processes window, this command causes the target to continue executing up to the cursor position (up to the stack level or process that the cursor is on, if in one of those windows). Execution may stop for another reason, for example hitting a breakpoint, before reaching the cursor position.

- Run/Step

If the list window is currently in source mode, this command causes a source statement to be stepped, otherwise it causes a machine instruction to be stepped. If the statement or instruction calls a subroutine, execution stops at the start of that subroutine.

- Run/step-Nodescend

If the list window is currently in source mode, this command causes a source statement to be stepped, otherwise it causes a machine instruction to be stepped. If the statement or instruction calls a subroutine, execution continues until the subroutine call returns and the next statement/instruction in the original subroutine is hit.

3.2.5 The Window Menu

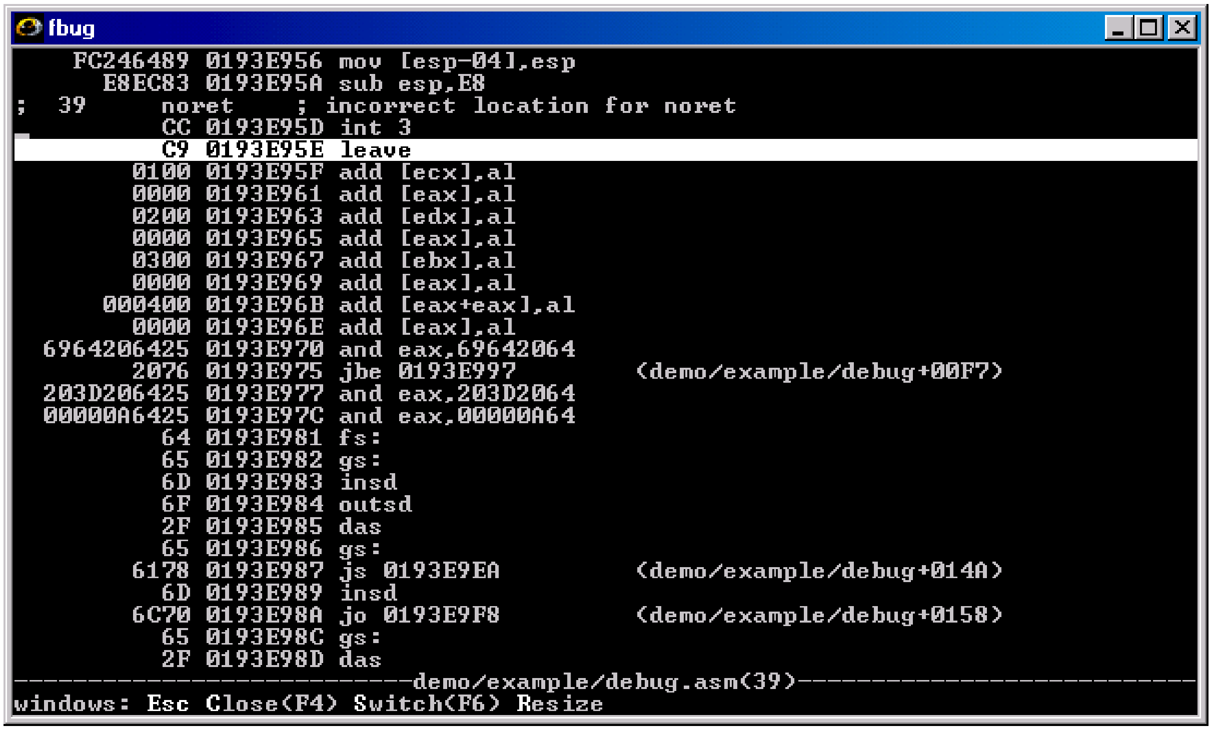

Figure 7. The Window Menu

- Window/Close

This command (shortcut F4) closes the window where the cursor is. It is not possible to close the list window.

- Window/Switch

This command (shortcut F6) moves the cursor to the next window down, or to the top window if already on the bottom window.

- Window/Resize

This command causes resize mode to be entered, in which the boundaries between windows can be moved. The boundary currently being moved is shown in inverse video, and it is moved using the up and down arrow keys. The F6 key changes which boundary is being moved and any other key changes out of resize mode.

3.2.6 Options

- Options/Input-radix

This command allows the default input radix (see Expressions below) to be changed.

- Options/Output-radix

This command allows the default output radix (see View/Expression command) to be changed.

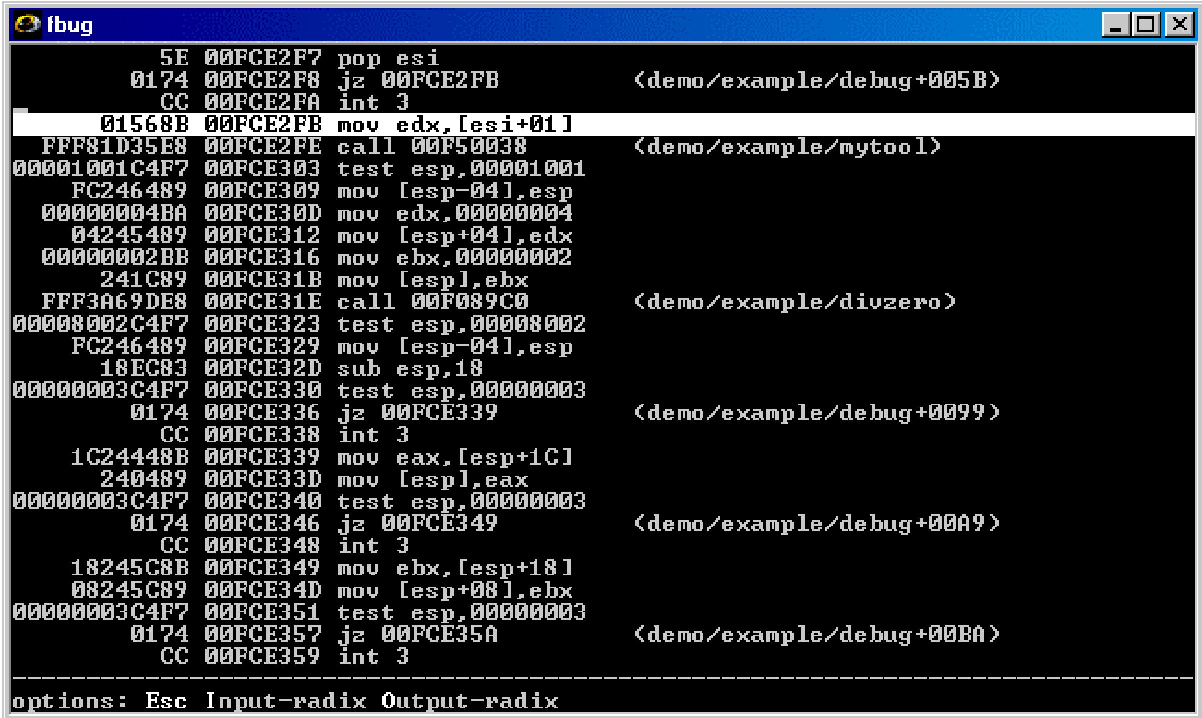

Figure 8. The Options Menu

4. Expressions

In fbug, an expression is interpreted in much the same way as an expression in C or Java, in the context of the PC in the selected stack level and process. A variable, register or memory location can be changed by evaluating an expression which contains an assignment operator such as '='. Restrictions to this are as follows:

- C and Java variables cannot currently be accessed by fbug.
- Float and double VP registers may give incorrect values on any ix86 based target (including local debugging on Linux and Windows).
- It is not possible to set the value of a VP register.

4.1.1 Radix specification

The radix of a number can be specified either in the C/Java way (0x for hexadecimal, 0 for octal) or the VP assembler way ($ for hexadecimal, $$o for octal, $$b for binary). If a number does not have a prefix specifying its radix, the default input radix is used. This is initially decimal, but can be changed by the Options/Input-radix command. The $$d prefix may be used for a decimal number when the default input radix is not decimal. If the default input radix is hexadecimal, and an un-prefixed hexadecimal number is used which starts with a letter, then it is interpreted as a number only if there is no symbol of that name. A floating point number is always decimal.

4.1.2 Tool, atom and register names

Prefixing a name in an expression with a '#' character has two effects:

- In addition to the characters which can normally be in a name, the characters '!', '$', '_' and '/' can also be used. This allows a tool or atom name to be specified. The value of a tool name is the address of the start of its code. Tool and atom names with other characters in (for example '+') can be specified by putting the tool/atom name in double quotes after the '#' character.

- The name is not matched by any high-level language variable or type. This allows a CPU or VP register or atom that has the same name as a variable to be accessed.

4.1.3 Primitive type names

If the context PC is in a C or Java program with debug information, then fbug understands the primitive type names used by C or Java respectively. Otherwise, it understands only the following unambiguous types:

- (unsigned) byte: 8 bit integer
- (unsigned) int:: 32 bit integer
- float: 32 bit floating point value
- void: no type, only used for pointers.
- (unsigned) short: 16 bit integer
- (unsigned) long: 64 bit integer
- double: 64 bit floating point value

Note that char is ambiguous because it has different meanings in C and Java.

4.1.4 Assembler-style memory access

To access a value in memory, either the C style or an assembler style syntax may be used. For example, the 32 bit integer at address 0x12345678 can be found using either of these expressions:

```
*(int*)0x12345678
[0x12345678].i
```

The suffix on the assembler style load must be one of:

- .b: unsigned byte
- .i: unsigned integer
- .f: float point
- .p: pointer
- .s: unsigned short
- .l: unsigned long
- .d: double

If the suffix is absent, .i is assumed.

5. The fbug Graphical User Interface

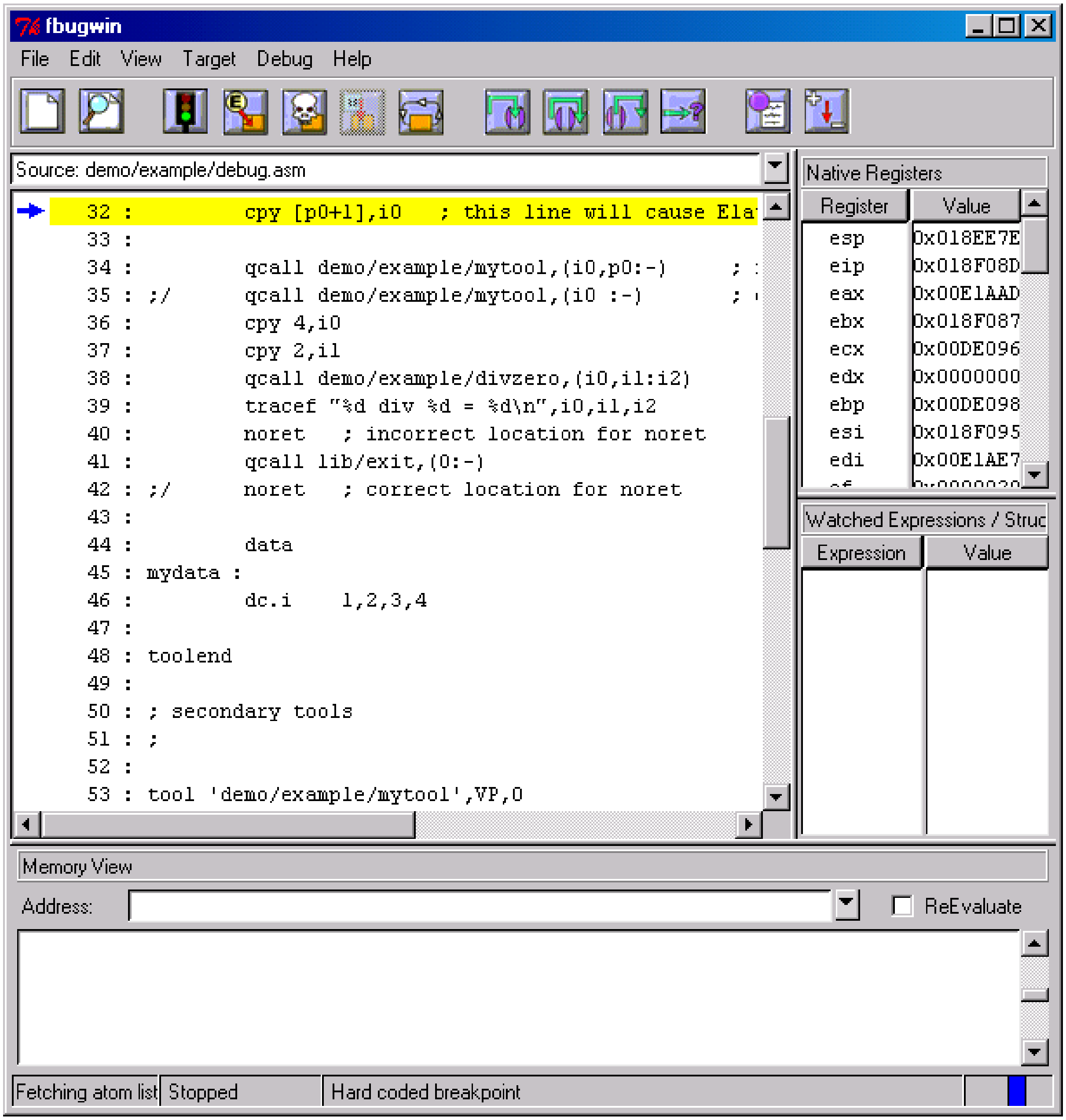

Figure 9. Fbugwin

The GUI presents the user with six initial menus – File, Edit, View, Target, Debug, and Help. The toolbar simply consists of a set of graphical shortcuts to functionality that is also available using the menu system. The status-bar at the bottom left of the main fbugwin window shows a short piece of text describing the purpose of each button when the mouse is held over the button.

5.1.1 File Menu

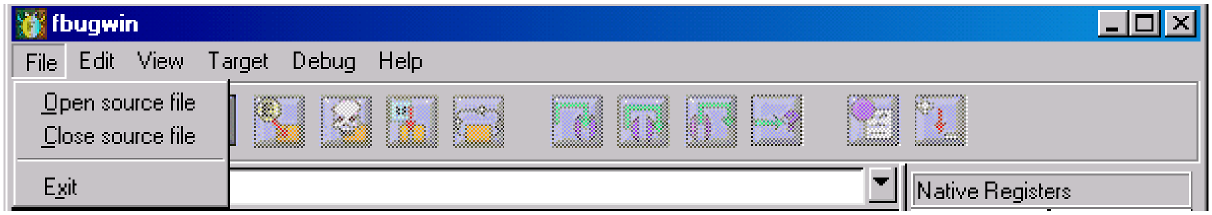

Figure 10. The File Menu

The file menu contains three shortcuts – Open, Close and Exit.

Open	Open source file, displaying it in main Source/Disassembly Window.
Close	If in *Source Mode*, close current source file. If in *Disassembly Mode*, clear main Source / Disassembly Window, causing redisplay on next operation.
Exit	Exit fbugwin, aborting debug session if it is running.

5.1.2 Edit Menu

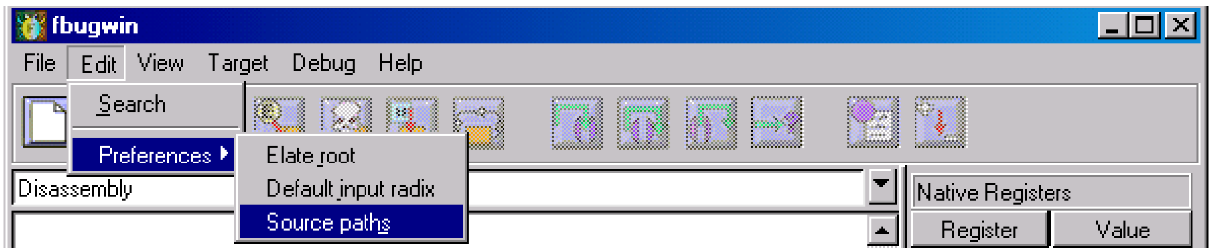

Figure 11. The Edit Menu.

Search	This menu item displays a dialog box into which the user may enter the details of a search to be performed of the current source file (searching is currently not implemented in *Disassembly Mode*). Options which may be set include the search text, the search direction, flags indicating whether the search is to be a regular-expression search and whether it is to be case-sensitive or not. The regular expressions are as defined by POSIX 1003.2, and can be basic, advanced or extended regular expressions.
Preferences	This displays a sub-menu allowing the user to set preferences. Currently the only preferences which can be set here are the option to reset the root directory and the input radix. In addition, it is possible add and remove source path list entries.

5.1.3 View Menu

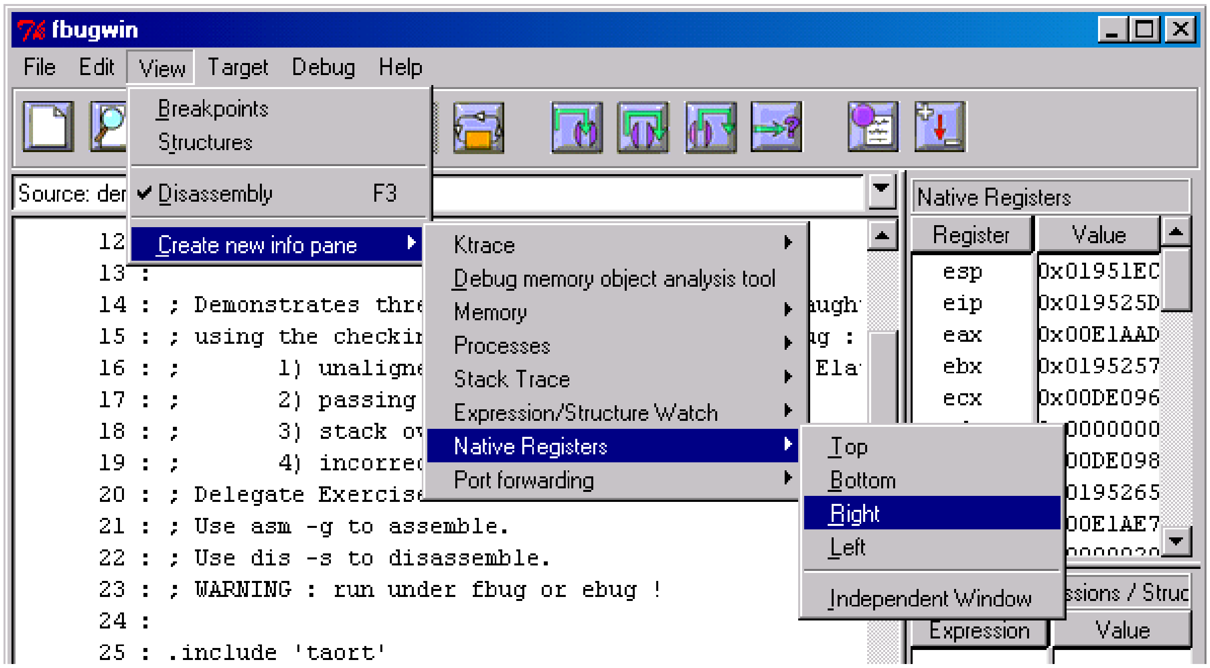

Figure 12. The View Menu

Breakpoints	This option displays a dialog box, allowing the user to clear, view and modify breakpoints. Breakpoints can be added to the list using the Set breakpoint at address or Set breakpoint at tool entry options from the Debug menu. It is also possible to toggle breakpoints from the Debug Menu. Extant breakpoints can be assigned a count and / or an expression using this dialog. If a count is assigned, this indicates the number of times the breakpoint can be hit before trapping into the debugger. If an expression is assigned, the expression will be evaluated whenever the breakpoint is hit, but will only trap into the debugger if the expression is true.
Structures	This option displays a dialog box which provides information about the structures available, and allows the user to add and remove structures. The *View* button on the dialog box also allows the user to examine the contents of structures at a known address. The *New* button provides the facility to define new structures. Definitions should be entered in the same format as the output of the asm -S command.
Disassembly	This is a flag controlling whether to use *Disassembly Mode* or *Source Mode*. In *Source Mode*, source is only displayed if debug information is available from the target. The window list above the Source / Disassembly window allows switching between the difference source and disassembly views available.
Create New Info Pane	This option displays a drop down menu, enabling the user to add the following information windows*, either as part of the main fbug window or as separate windows.

* These are as follows:

- Ktrace
- Memory
- Stacktrace
- Native Registers
- Debug Memory Object analysis tool
- Processes
- Expression / Structure Watch
- Port Forwarding

5.1.4 Target Menu

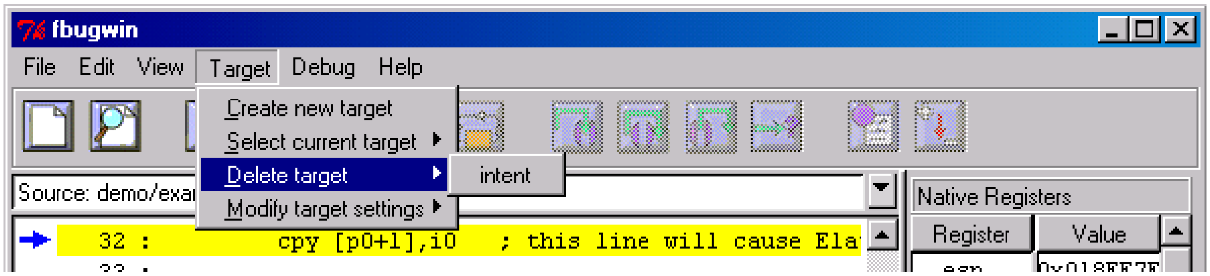

Figure 13. The Target Menu

Create new target	This menu item displays a series of dialog boxes which one can use to specify the parameters for a new target.
Select current target	Displays a sub-menu consisting of all existing target configurations, and allows the user to select which one to use. If this is changed while the target is running, the change will not take effect until the target is restarted.
Delete target	Displays a sub-menu consisting of all existing target configurations, and allows the user to select one to be deleted.
Target / Modify target settings	This option displays a sub-menu consisting of all existing target configurations, and allows the user to select one and modify any of its parameters.

5.1.5 Debug Menu

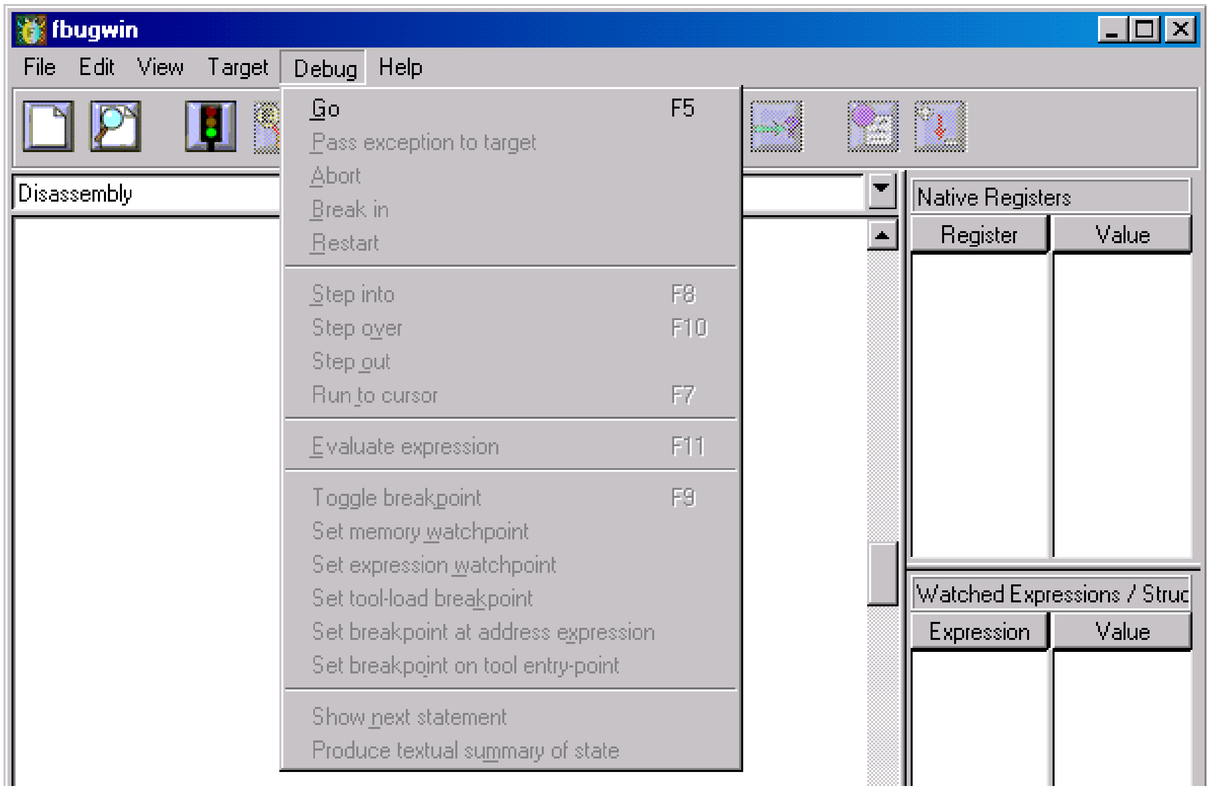

Figure 14. The Debug Menu

Debug / Go	This option starts the debug session if it is not already running, and otherwise allows the target to continue.
Debug / Pass exception to target	This option allows the target to continue, but if an exception occurred passes it to the target for processing by intent exception handlers.
Debug / Abort	This option aborts the debug session, and terminates the target intent session if it is a local process on Linux or Windows.
Debug / Break in	This option interrupts the target intent session.
Debug / Restart	This option restarts the target intent session if possible.
Debug / Step into	This option causes the target to "step into" the current instruction or source-line, depending on whether the debugger is in *Source Mode* or *Disassembly Mode*.
Debug / Step over	This option causes the target to "step over" the current instruction or source-line, depending on whether the debugger is in *Source Mode* or *Disassembly*. Thus if the current instruction or source-line is a call, the debugger will continue until the instruction or source-line immediately after the call, rather than stopping on the destination instruction.
Debug / Step out	This option causes the target to "step out" of the current function. This sets a breakpoint on the

	instruction immediately after the current stack frame, runs until this breakpoint is hit, and then removes the breakpoint.
Debug / Run to cursor	This option sets a temporary breakpoint at the cursor position, runs until a breakpoint is hit, and then removes the breakpoint set above.
Debug / Show next statement	This option shows the instruction which would be executed next if the target were allowed to run. If the target has crashed, the line which was being executed is shown. The relevant line is indicated by a small, blue arrow to the left of the line in the Source / Disassembly Window.
Debug / Evaluate expression	This option displays a dialog box into which the user can enter an fbug expression for evaluation. The result of the expression is presented to the user in the chosen format.
Debug / Set memory watchpoint	Displays a dialog box into which the user may enter an address (or an expression evaluating to an address), a size (byte, short or word), and a flag (break on change in boolean value / break on any change). A watchpoint expression is constructed from these parameters and is set. Execution will be interrupted if the value of the specified memory location changes in the way specified by the user.
Debug / Toggle breakpoint	Toggle (i.e. set and unset) a breakpoint at the current cursor location.
Debug / Set watchpoint on expression	Displays a dialog box into which the user may enter a Watchpoint Expression. A watchpoint is set on this expression, and execution will be interrupted if the value changes in the way specified by the watchpoint.
Debug / Set tool-load breakpoint	The user is prompted for a tool name, and a breakpoint is set on the loading of the specified tool. In addition, a small dialog box pops up informing you if the breakpoint is hit.
Debug / Set breakpoint at address expression	The user is prompted for an address expression (toolnames can be specified by prefixing them with a "#" character). A breakpoint is set at the resulting address.
Debug / Set breakpoint at tool entry-point	The user is prompted for a tool name. If the specified tool is present in memory, a breakpoint is set at its entry-point. Otherwise, a breakpoint is set on the loading of the tool, and when the tool-load breakpoint is hit, a breakpoint is automatically set at the entry-point of the tool.
Debug / Produce Textual Summary of State	This menu item produces a textual summary of the target machine's state, suitable for sending in an email or recording for later analysis. When the menu item is selected, a dialog box appears, in which the user can select which parts of the target machine's state should be recorded, including native registers, expressions, code areas, memory areas, stack information and process information.

5.2 The fbug Windowing Options

5.2.1 Window Location

The location of the various fbugwin windows can be selected from the View menu, or from a context menu within the relevant window. There are currently five options:

Top	The corresponding window is docked at the top of the fbugwin main window.
Bottom	The corresponding window is docked at the bottom of the fbugwin main window.
Right	The corresponding window is docked at the right of the fbugwin main window.
Left	The corresponding window is docked at the left of the fbugwin main window.
Independent window	The corresponding window is displayed in a separate top-level window that can be moved and resized independently of the main fbugwin window. If this window is closed, the state reverts to "Not displayed".
Not updating (available only from context menu)	The corresponding window is no longer updated. In this mode, the information contained in the window is not refreshed each time the target stops. Thus, the execution speed of the debugger may be slightly faster. The words 'update disabled' are displayed in the title of the window, and updating can be re-enabled from the context menu.

5.2.2 Source / Disassembly Window

This main window displays source and / or disassembly information gathered from the target. It is updated whenever the target stops to show the code at the current program-counter location.

If the "currently-selected process" (see Process Information) or the current stack level is changed the code at the corresponding location is displayed in this window.

The window consists of several parts: a section on the left showing the presence of breakpoints (if there are any in the currently displayed area) as small red circles and the current program-counter (if it is in the currently displayed area), and a section on the right showing either the source file or the disassembly. A small blue arrow indicates the current line of source code or disassembly.
If source information is available, then this window can display either the source code for the current tool, or the native instructions at the specified addresses interspersed with source lines. If source information is not available, then only the disassembly into native instructions will be shown. Immediately above the window is a window list, which can be used to switch between the source or disassembly views currently available.

There are two modes of operation, *Source mode* and *Disassembly mode*, which govern which view is displayed by default. If source information is not available, disassembly will be displayed. Toggling between the modes is achieved by checking the "Disassembly" item in the View menu.

Context menus can be brought up by clicking the right mouse button over different areas of the window. Clicking on a line of source code in *Source mode* brings up Menu A. In *Disassembly mode*, clicking on a line of disassembly brings up Menu B, or clicking on a toolname (displayed in green) Menu C.

Source / Disassembly window Context menu A

- Search file
- Goto line
- Toggle breakpoint at this line
- Modify breakpoint settings
- Repeat last search
- Close file
- Run to this line

Source / Disassembly window Context menu B

- Disassemble at address
- Toggle breakpoint at this address
- Disassemble at tool
- Run to this address

- Modify breakpoint settings

Source / Disassembly window Context menu C

- Disassemble at address
- Toggle breakpoint at this address
- Modify breakpoint settings
- View tool "toolname" at offset 0xXX (only present if available)
- Disassemble at tool
- Run to this address
- View tool "toolname"

Source/Disassembly Context / Search file	This menu item displays a dialog box into which the user may enter the details of a search to be performed of the current search file. Options which may be set include the search text, the search direction, and flags indicating whether the search is to be a regular-expression search and whether it is to be case-sensitive.
Source/Disassembly Context / Repeat last search	This option performs a search with the criteria specified for the previous search. This search is begun from the line over the which the mouse was clicked, *not* from the line on which the previous search ended.
Source/Disassembly Context / Goto line	This option prompts the user for a line number, and then moves the cursor to the specified line.
Source/Disassembly Context / Close file	This option closes the current source file.
Source/Disassembly Context / Toggle breakpoint at this line	This option toggles (i.e. sets and unsets) a breakpoint on the line of code indicated.
Source/Disassembly Context / Run to this line	This option sets a temporary breakpoint at the source line specified, runs until a breakpoint is hit, and then removes the breakpoint set above.
Source/Disassembly Context / Modify breakpoint settings	If the line selected has a breakpoint set, then this option allows you to alter the conditions under which the breakpoint traps into the debugger. Breakpoints can be assigned a count and / or an expression. If a count is assigned, this indicates the number of times the breakpoint can be hit before trapping into the debugger. If an expression is assigned, the expression will be evaluated whenever the breakpoint is hit, but will only trap into the debugger if the expression is true.
Source/Disassembly Context / Disassemble at address …	The user is prompted for a memory address, and the disassembly into native code of the data at that address is displayed.
Source/Disassembly Context / Disassemble at tool ...	The user is prompted to enter a toolname which occurrs within the current context, and the disassembly of the tool into native code is displayed. If source information is available, then lines of source code will be interleaved with the disassembly.
Source/Disassembly Context / Toggle breakpoint at this address	This option toggles (i.e. sets or unsets) a breakpoint on the line of code indicated.
Source/Disassembly Context / Run to this address	This option sets a temporary breakpoint at the address specified, runs until a breakpoint is hit, and then removes the breakpoint.
Source/Disassembly Context / View tool "toolname"	This option displays the disassembly of the tool toolname, interspersed with source code if available.
Source/Disassembly Context / View tool "toolname" (offset 0xXX)	Displays the disassembly of the tool toolname at offset 0xXX, interspersed with source code if available.

5.2.3 Memory Information

This window displays the contents of memory at a specified address on the target. The "Address" text entry field allows the user to enter an address-expression (an fbug expression which evaluates to an address). The memory at this address is then displayed.

The window contains a checkbutton labeled "Reevaluate". If this is selected when an address expression is entered, the expression is re-evaluated each time the target stops, and the memory window updated to display the contents of the memory at the resulting address (which may be different to the previous displayed address). If the "Reevaluate" checkbutton is not selected, the address-expression is evaluated once when it is entered, and the memory at the resulting address is displayed. When the target stops subsequently the displayed address does not change. When the target stops, the contents of the memory window are updated, and any differences from the previous values are highlighted in red.

If the right mouse button is clicked in the memory window, a popup menu is displayed, presenting the user with the options to search the target memory or alter the display.

Memory Information Window Context Menu

- Search target memory
- View memory as 16-bit words
- Limit width to powers of 2
- View memory as 32-bit words
- View memory as 8-bit bytes and chars

Memory Information Context / Search target memory	This option displays a dialog box which allows the user to specify a search string, the address from which to begin the search and the maximum search length in bytes. The search string can be a textual string, and may involve any of the standard characters which use the backslash '\' as an escape character, or can be a byte value to search for directly in memory. A hex value can be indicated by preceding the search value with '\x', octal by preceding it with '\o'. It should be noted that searching for '\abcdefgh' and '\ab\cd\ef\gh' will produce different results. The latter will search for the 4 bytes occurring the order they are written, whereas the former will search for the little-endian word 'ghefcdab'.
Memory Information Context / View memory as 32-bit words	This option displays the target memory as a series of 32-bit words.
Memory Information Context / View memory as 16-bit words	This option displays the target memory as a series of 16-bit words.
Memory Information Context / View memory as 8-bit bytes and chars	This option displays the target memory as a series of bytes on the left-hand side of the window, and interprets the data as ASCII characters (where possible) on the right hand side.
Memory Information Context / Limit width to powers of 2	Checking this item ensures that the number of columns of memory displayed is the largest power of 2 that will fit into the window. If it is not checked, then the largest number of columns possible will be displayed.

5.2.4 Process Information

This window displays information about any currently running processes on the target system, including its PID, GP, Priority, Wake count, Suspend count and State, and main tool. The PID of the "currently-selected process" is displayed in blue. The "currently-selected process" can be changed by clicking on a PID.

Context menus can be brought up by clicking the right mouse button on various areas of the window: clicking on a column header displays Menu A, clicking on a process, Menu B.

Process Information Context Menu A

- Select displayed fields
- Disable field "fieldname"
- Restore default field layout

Process Information Context Menu B

- Show properties of process "0xXX"

Process Information Context / Select displayed fields	This option provides a list of checkboxes that allow the user to select which information is displayed about the processes. The fields available include the process ID, priority, suspend count, current program-counter, flag indicating whether the process was pre-empted, main tool of the process, globals pointer, wake count, state, current stack-pointer and parent process ID.
Process Information Context / Restore default field layout	This option returns all the fields displaying information about the processes to their default widths.
Process Information Context / Disable field "fieldname"	This option stops displaying the field "fieldname", and resizes the other fields appropriately.
Process Information Context / Show properties of process "0xXX"	This option displays detailed information about process 0xXX, including the process' ID, globals pointer, priority, wake count, suspend count, state, current program counter, current stack pointer, whether or not the process was pre-empted, the PID of the parent, and the main tool.

5.2.5 Native Register Information

This window displays the current values of the target's native registers. It is updated whenever the target stops, and any changes from the previous values are highlighted in red.

If the right mouse button is clicked in this window, a context menu A is displayed, allowing the user to select which native registers should be displayed. A right-click over a register name or value brings up menu B.

Native Register Information Context Menu A

- Select displayed registers

Native Register Information Context Menu B

- Display as hexadecimal
- Display as binary
- Display as signed decimal
- Restore original format
- Select displayed registers
- Display as octal
- Display as character
- Display as unsigned decimal
- Modify/Display value of "register"

Native Register Context / Select displayed registers	This option provides a list of native registers, from which the user may select which ones they wish to be displayed.

Native Register Context / Display as hexadecimal	This option displays the contents of the selected register as hexadecimal value.
Native Register Context / Display as octal	This option displays the contents of the selected register as octal.
Native Register Context / Display as binary	This option displays the contents of the selected register as binary.
Native Register Context / Display as character	This option displays the contents of the selected register as a character.
Native Register Context / Display as signed decimal	This option displays the contents of the selected register as a signed decimal value.
Native Register Context /	This option displays the contents of the selected register an unsigned

Display as unsigned decimal	decimal value.
Native Register Context / Restore original format	This option returns to displaying the contents of the selected register in the default format.
Native Register Context / Modify/Display value of "register"	This option displays a dialog box which allows the user to change the contents of the native register "register

5.2.6 Watched Expressions

This window displays the current value of the "watched expressions". These are expressions specified by the user, whose values are updated whenever the target stops. Differences from the previous values of the expressions are highlighted in red.

If the right mouse button is clicked in the Expression section of the window, context Menu A is displayed; clicking on a watched expression brings up Menu B, and the value of a watched expression Menu C.

Watched Expressions Window Context Menu A

- Add new watch expression/structure
- View kernel data area
- View VP registers

Watched Expressions Window Context Menu B

- Remove watch expression/structure "name"
- View VP registers
- Add new watch expression/structure
- View kernel data area

Watched Expressions Window Context Menu C

- Display as hexadecimal
- Display as binary
- Display as signed decimal
- Restore original format
- Display as octal
- Display as character
- Display as unsigned decimal
- Modify/display value of "name"

Watched Expressions Context / Add new watch expression/structure	This option displays a dialog box which allows the user to enter a new expression to be 'watched'.
Watched Expressions Context / View VP registers	This option adds the VP registers to the list of watched expressions, displaying them as an unexpanded list. If the toggle button is not present to expand the list, then the contents of the registers is not available. It should be noted that VP registers added in this way cannot be handled individually. Individual VP registers can be added as watched expressions using the Add new watch expression/structure menuitem.
Watched Expressions Context / View kernel data area	This option adds all the fields of the Kernel data area to the list of watched expressions, displaying them as an unexpanded list.
Watched Expressions Context / Remove watch expression/structure "name"	Removes the selected expression or structure, “name” from the list.
Watched Expression Context / Display as hexadecimal	This option displays the selected value as a hexadecimal.
Watched Expression Context / Display as octal	This option displays the selected value as octal.
Watched Expression Context / Display as binary	This option displays the selected value as binary.

Watched Expression Context / Display as character	This option displays the selected value as a character.
Watched Expression Context / Display as signed decimal	This option displays the selected value as a signed decimal.
Watched Expression Context / Display as unsigned decimal	This option displays the selected value as an unsigned decimal.
Watched Expression Context / Restore to original format	This option returns to displaying the selected value in its original format.
Watched Expression Context / Modify/display value of "name"	This option displays a dialog box which allows the user to alter the value of the corresponding expression, “name.” Currently, the values of VP registers may not be altered.

5.2.7 Ktrace Information

The Ktrace Information window simply displays any re-directed ktrace information. To redirect from ktrace.log, consult the relevant documentation for your platform. If the right mouse button is clicked over the ktrace window, the following context menu is displayed:

Ktrace Information Context Menu

- Set number of scrollback lines
- Set output file
- Set ktrace filter regexp
- Clear contents

Ktrace Context / Set number of scrollback lines	The ktrace window buffers lines of ktrace, and presents them as a scrollable display. This can be used to set the number of lines which are buffered at any time. The default setting is 500 lines.
Ktrace Context / Set ktrace filter regexp	This option displays a dialog box which allows the user to enter a regular expression. The ktrace window will then display only lines of ktrace which contain a matching expression. If an expression is set it is displayed in the title bar of the window. This feature uses regular expressions as defined by POSIX 1003.2, and handles basic, advanced and extended regular expressions. This option can also be used to remove a regular expression filter, by leaving the text area of the dialog box blank. It should be noted that the filter expression remains set until it is explicitly removed. If a filter expression is set at the time fbugwin is closed down it will be saved as a preference, and will be set next time fbugwin is opened.
Ktrace Context / Set output file	This option displays a dialog box which allows the user to enter a filename. The ktrace output is then both displayed in the ktrace window and written to the specified file. The output filename will remain set until explicitly cleared. Each time the target is restarted, or allowed to conitinue, ktrace is appended to the end of the output file. If fbugwin is closed and re-opened, the output file is cleared.
Ktrace Context / Clear contents	This option clears the contents of the ktrace information window. It does *not* clear any other preferences, such as regular expression filters.

5.2.8 Stacktrace Information

This window displays the stacktrace information for the currently selected process.

The "top", or 0th, level of the stack trace shows the stack pointer and program counter at the point at which execution stopped, also the name of the tool which was being executed, and the offset reached within that tool. The previous level (1) indicates the toolname previously executed, the offset at which the current tool was called, and corresponding stack pointer and program counter. The remainder of the call stack displays information in a similar format.

It should be borne in mind that the call stack is calculated by fbug and may contain 'phantom' entries, which cannot be algorithmically determined to be false.

Right-clicking on any of the column headers displays context Menu A; clicking on any line of stacktrace other than the top one displays Menu B.

Stacktrace Information Context Menu A

- Select displayed fields
- Disable field "name"
- Restore default field layout

Stacktrace Information Context Menu B

- Run to stack level X
- Read all stack entries

Stacktrace Context / Select displayed fields	This option displays a dialog box which allows the user to select the fields which are to display information. Available fields include: • Num (the stack level) • Stack pointer

	• Program counter • Details (eg. toolname if known)
Stacktrace Context / Restore original field layout	This option resizes the displayed fields to return them to their default widths.
Stacktrace Context / Disable field "name"	This option removes the field "name" from the display. (If "name" is the only field currently displayed, this has no effect).
Stacktrace Context / Run to stack level X	This option sets a breakpoint at the instruction at stack level X (ie the one following the call), runs until a breakpoint is hit, and then removes the breakpoint which was set.
Stacktrace Context / Read all stack entries	If there are more stack entries than can be displayed in the window, then only the number required to fill the window are read in. Scrolling down causes the remaining entries to be read in as they are needed, with the scrollbar resizing itself as more entries become available. Alternatively, this option allows all remaining stack entries to be read in at once.

5.2.9 Port Forwarding

When Port Forwarding is enabled the specified port (either a serial port or a socket connection) is made available to another machine over a socket connection. This allows debugging of remote targets from machines other than those directly connected to the target.

The Port Forwarding window provides information on the current parameters set for forwarding, and enables you to change them (via the *Change Settings* button). The *Start* button will remain greyed out until a serial port, or client host and port, has been specified.

5.2.10 Debug Memory Object Analysis Tool

When using the Debug Memory Object, the sys/kn/mem/util/showmem tool becomes available. This outputs information about memory blocks, in a textual form.

The Debug Memory Object Analysis Tool allows this information to be presented in a graphical format, and is always opened in a separate window. Right-clicking in the window brings up menu A which enables the user to select a file from which to read in memory information.

Once the file has been read the user is prompted to 'select a section' and right-clicking again will bring up context menu B. The Select section, PID and mode option allows the user to choose the information which is to be displayed.

The data is displayed as a tree diagram, with nodes whose stacktraces share a common tool sharing a parent node. Each node provides information about the size and number of allocated blocks it represents, and the tools which performed the allocation. Further information about the memory blocks can be obtained by expanding the nodes, either on an individual basis or by using the Expand all terminal nodes option on menu B.

Debug Analysis Context Menu A

- Write results to a file
- Disable filtering of stacktrace entries
- Analyse debug memory object data from file

Debug Analysis Context Menu B

- Write results to a file
- Disable filtering of stacktrace entries
- Select section, pid and mode
- Collapse all terminal nodes
- Analyse debug memory object data from file
- Discard current data
- Expand all terminal nodes

Debug Analysis Context / Write results to a file	This option allows the user to save a textual representation of the data to a specified file. This is an easier to read format than the data

	obtained directly from *sys/kn/mem/util/showmem*
Debug Analysis Context / Analyse debug memory object data from file	The user is prompted to enter a filename from which to read data. The file specified should contain information in the format output by *sys/kn/mem/util/showmem*, and should not contain anything else.
Debug Analysis Context / Disable filtering of stacktrace entries	The stacktrace filtering is intended to remove spurious entries from the stacktrace; this option can be used to turn it off if it is not required.
Debug Analysis Context / Discard current data	This option discards all data previously read in.
Debug Analysis Context / Select section, pid and node	This option displays a dialog box, allowing the user to choose which information is to be displayed. Each time *sys/kn/mem/util/showmem* is called, the output is written as a separate 'section'. Thus if *sys/kn/mem/util/showmem* has been called three times, the results will appear as sections 0,1 and 2 respectively. A graph can be drawn for an individual process, by selecting the PID of the required process, or for all processes (select 'All PIDs'). The mode simply indicates whether the graph should include any information about allocated blocks only, free blocks only, or all blocks.
Debug Analysis Context / Expand all terminal nodes	Each of the terminal nodes can be expanded individually to give more information about the blocks they represent, including the memory address. This option expands all terminal nodes.
Debug Analysis Context / Collapse all terminal nodes	If terminal nodes have been expanded to display more information, this option collapses all currently-expanded terminal nodes.

6. Common Problems in Debugging

The tutorials in the following section test fbug by demonstrating how to trap five common programming errors, as exemplified by the demonstration program *demo/example/debug.asm*.

The problems covered by the tutorial can only be detected through a checking translator, for example the pentiumt translator available for the ix86. Debugging using a checking translator can help to avoid the more obscure problems that such faults may cause on target hardware. Specifically:

1.Non-aligned Memory Access

This is detected by the pentiumt translator emitting special code to detect the situation and hit a hard coded breakpoint. When not using pentiumt, ix86 platforms will not detect this. Nonetheless, it will cause a crash on the majority of other CPUs.

2. Stack overflow:

This is detected by the pentiumt translator emitting special code to detect the situation and hit a hard coded breakpoint. Other translators will carry on using the overflowed stack, which will cause memory corruption that can be hard to track down.

3. Incorrect parameters:

This is detected by the pentiumt translator emitting special code to detect the situation and hit a hard coded breakpoint. Other translators simply carry on, which may cause corruption of registers that may be hard to track down.

4. Divide by Zero

It is a requirement of the VP specification that divide by 0 is detected. On ix86 cpus this is done by using the hardware exception, which fbug detects. On many other cpus, there is no divide instruction and as such the divide by zero is detected by the emulation tool, which will not be picked up by fbug.

5. Incorrect use of the noret instruction:

This is detected by the pentiumt translator emitting special code to detect the situation and hit a hard coded breakpoint followed by a leave instruction. Other translators continue on into whatever code was following, which may cause erratic behaviour to occur or crash immediately.

The following section also demonstrates setting of breakpoints.

7. Using the Debugger Graphical Interface

This tutorial is mainly intended for use with fbugwin, although brief notes are included describing use of fbug.. In order to usefbugwin, it is first necessary to go through the process of creating a new target. A "target" is simply a named set of parameters for the debugger, which define how the debugger should communicate with the ‘debuggee.’ A new target may be selected through the target menu, through a series of dialogboxes. The first of these allows the selection of the name of the new target (which must be unique), and the target type, which is one of: Local, Remote (serial), Remote (socket) and Remote (pipe). For these purposes, Local should be selected.

The second dialog box depends on the target type which was just selected. For a "Local" target, the user is prompted to enter the executable name, image file and any applicable flags for the executable. It also allows the user to select the name of a symbol-file to use. Alternatively, the user can select not to use a symbol-file at all. Finally, additional fbug options may be entered.

As such for Windows®, this will be of the form:

Executable name:

```
C:/intent/sys/platform/win32/elate.exe
```

Image file:

```
C:/intent/sys/platform/win32/developt.img
```

For an intent graphics window the following may be specified.

Flags:

```
-i0 –c’dev/ave/dsk/tiles/ -sdev/ave/dsk/runapp.scr’
```

For Linux® it will be of the form:

```
sys/platform/linux/elate
```

```
sys/platform/linux/ix86/developt.img
```

For a "Remote (pipe)" target, the user is prompted to enter the names of the input and output pipes. If the output pipe is not specified, the input pipe is used for both input and output. For a "Remote (serial)", the user is prompted to enter the name of the serial port to use (such as "COM1" and so on for Windows, or "/dev/ttyS0" on Linux), and the baud rate. For a "Remote (socket)" target, the user is prompted to enter a hostname, and a port.

Once the process of creating a new target has been finished, a button on the toolbar (currently containing a picture of traffic lights) can be used to start the intent session (assuming this has not already been done through a command line option).

fbug users will need to specify the image name at the command line.

7.1 Non-aligned memory access

At the intent shell prompt type:

```
$ asm –vg demo/example/debug
```

This assembles *debug.asm* in verbose mode with debugging information included. *Debug.asm* contains a number of common errors that we are going to use the debugger to find and correct. Now run the code:

```
$ demo/example/debug
```

Initially, *demo/example/debug.asm* may be within the intent filesystem which cannot be read by fbug. Either specify the filename when prompted by fbugwin or (if using fbug) touch the file before running:

```
touch demo/example/debug.asm
```

At this point fbugwin should report a hard coded breakpoint and display the line that is causing the problem:

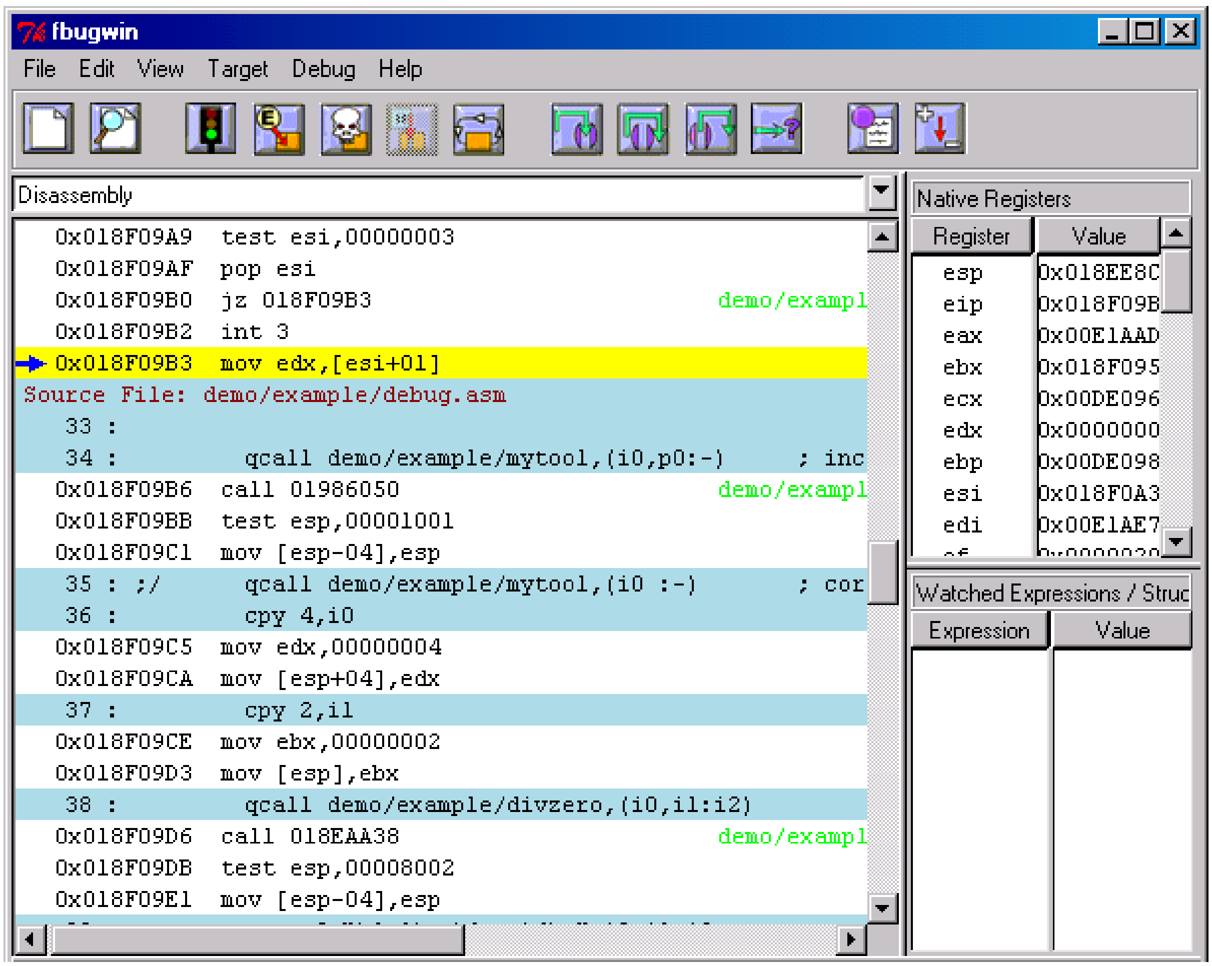

Figure 15. Non-aligned Memory Access

The line of interest is highlighted above by the presence of an arrow – in the text user interface the entire line will simply be highlighted. Note that fbug displays the VP source, although the above screenshot of fbugwin has the interface displaying both source and disassembly. To correct the problem, comment out the offending line. If unaligned memory access is required then the *cpy.ni* instruction should be used (note the 'n').

7.2 Incorrect parameters passed

Reassemble the source code after modifying it, while remembering to use the –g option to add debugging information to the tool. Run the tool again. This time fbugwin will display the point at which a problem is reached, which should be line 54:

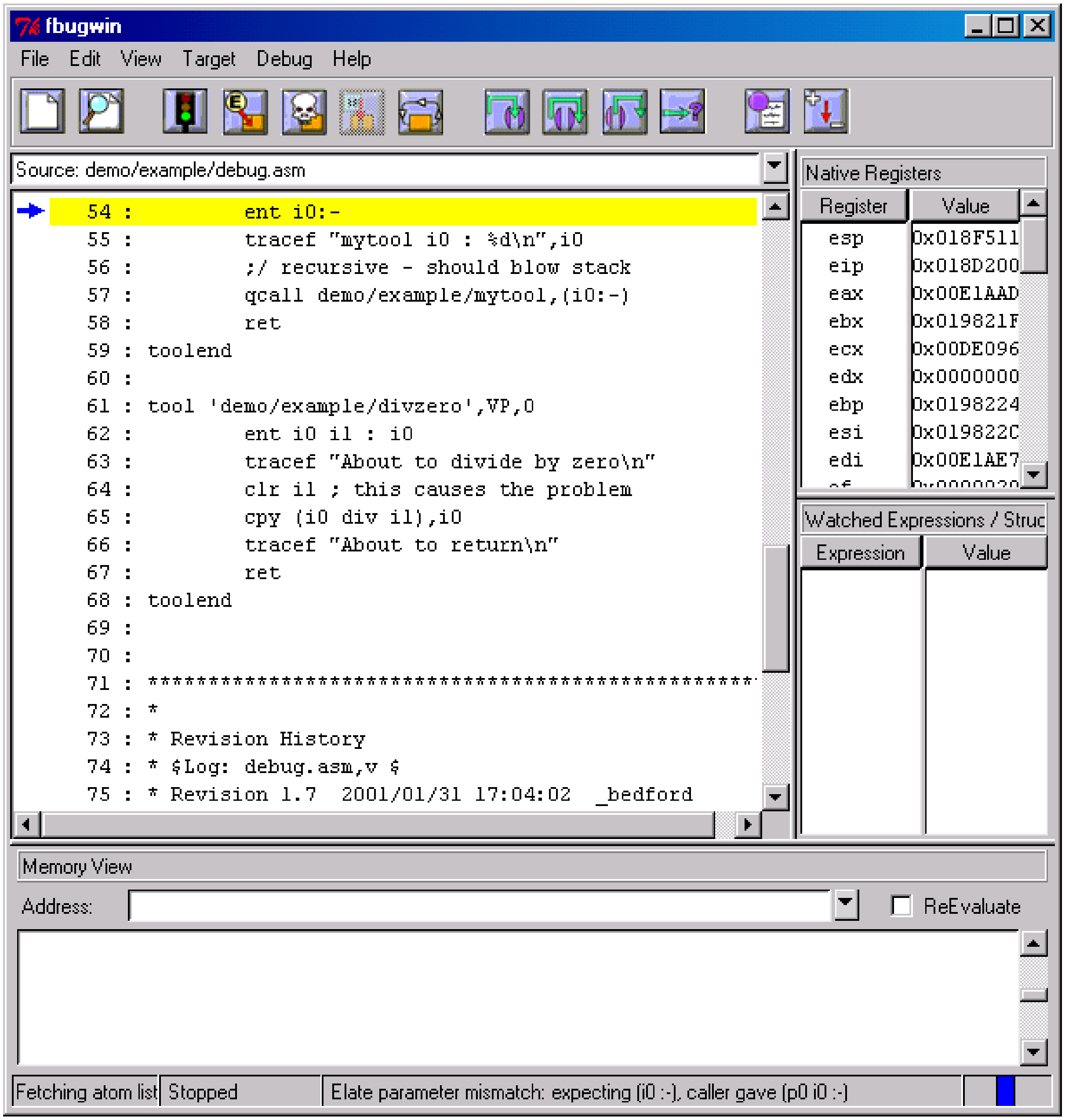

Figure 16. Incorrect Parameters

As such, it is necessary to establish the calling line location and what subroutine is being called.

A pop-up menu box displays information, if available, concerning the cause of the problem. An Elate® parameter mismatch: expecting (i0:-), caller gave (p0:-)" is reported. Since we can see from the source/disassembly window that execution stopped at line 54, we know that *demo/example/mytool* was called with the wrong parameters. If a stacktrace window is opened, clicking on the previous stack level (i.e. Num=1) displays in the source/disassembly window the line from which *demo/example/mytool* was called (line 34). As we can see from that the wrong parameters are passed in.

In the case of debugging with fbug; in the stack trace window (which can be accessed by going through (V)iew & (C)allstack), move to the line of interest, and hit enter. The list window then moves the source or disassembly to the location in question. The source should display a comment to the effect that that particular line of source code has the incorrect parameters.

Correct the incorrect line in the source code – simply comment it out and replace it with line number 35- and then reassemble.

7.3 Stack overflow

The next problem that should become apparent is a stack overflow. The tool *debug.asm* deliberately simulates this by making *demo/example/mytool* recursive. Of course, this quickly uses up the 8K of stack allocated to the main tool (*demo/example/debug*) and thereby causes fbugwin to be entered into.

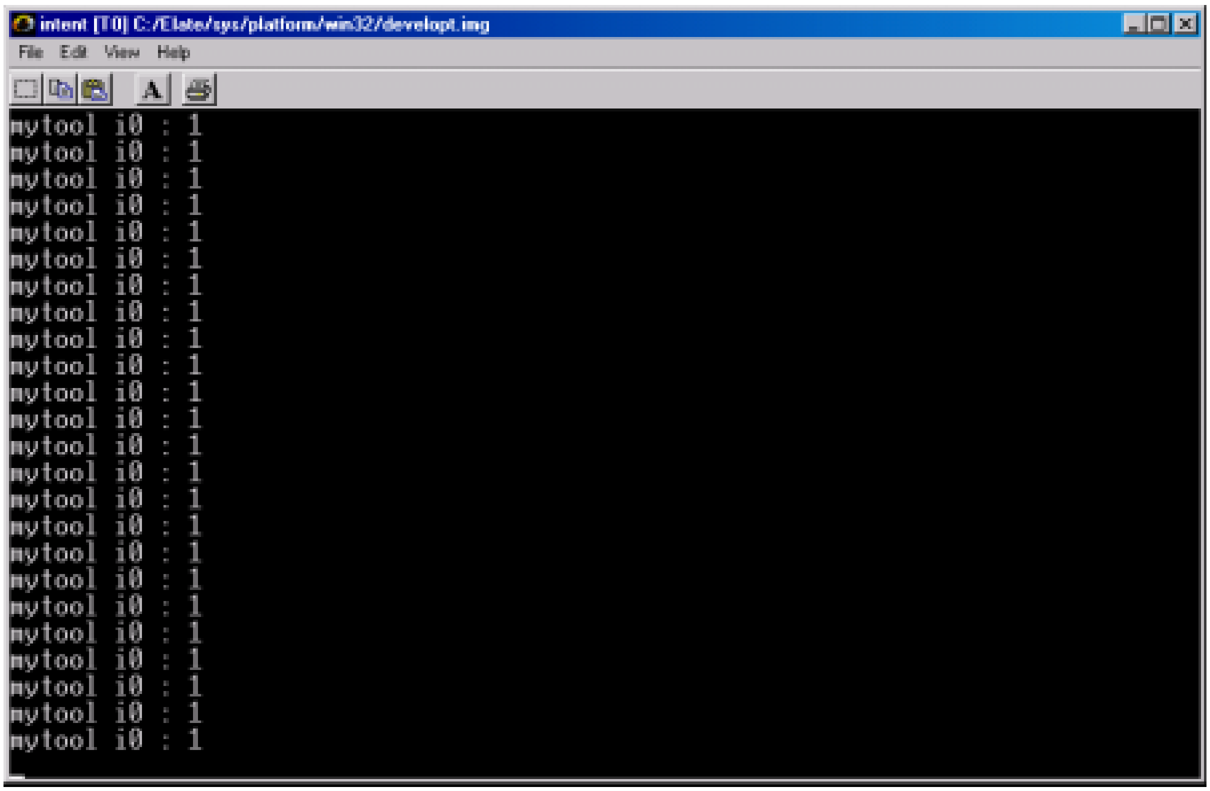

*Figure 17. in**te**nt SessionStack Overflow*

The cause of a stack overflow is best located by examining the stack trace, looking in particular for 'loops' in the stack trace or large jumps in SP. In this case we find that *demo/example/mytool* has been called repeatedly, thereby forming a loop. Clicking on one of these lines of trace brings up the relevant code in the source/disassembly window, from which it can be seen that *demo/example/mytool* has been repeatedly calling itself.

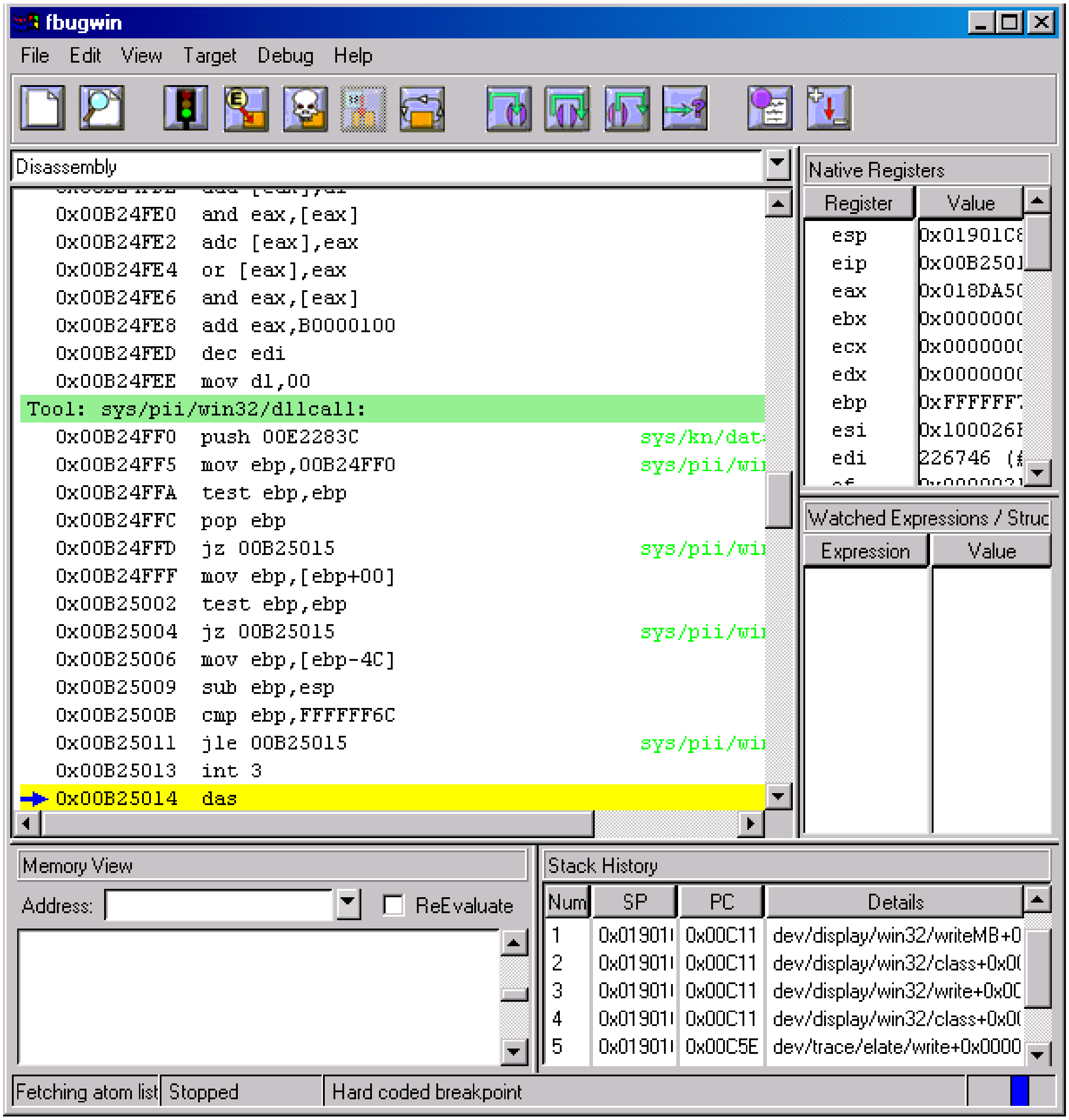

Figure 18. The Stack Window (displayed in the bottom right-hand corner)

In the case of using fbug, selecting callstack under the view menu again will show that the tool *demo/example/mytool* has been called repeatedly. Selecting one such line (with the source display being engaged rather than disassembly) should show the offending line of source code, namely line 57.

To rectify this issue, either comment out this line of code, thereby preventing it from calling itself, or rewrite the tool in such a manner as to stop the recursion at another point in the code.

7.4 Divide by zero

After fixing the recursive call problem, reassemble and run again. This time, fbugwin (or fbug) traps a new problem and informs us that an exception has occurred. The disassembly is as follows:

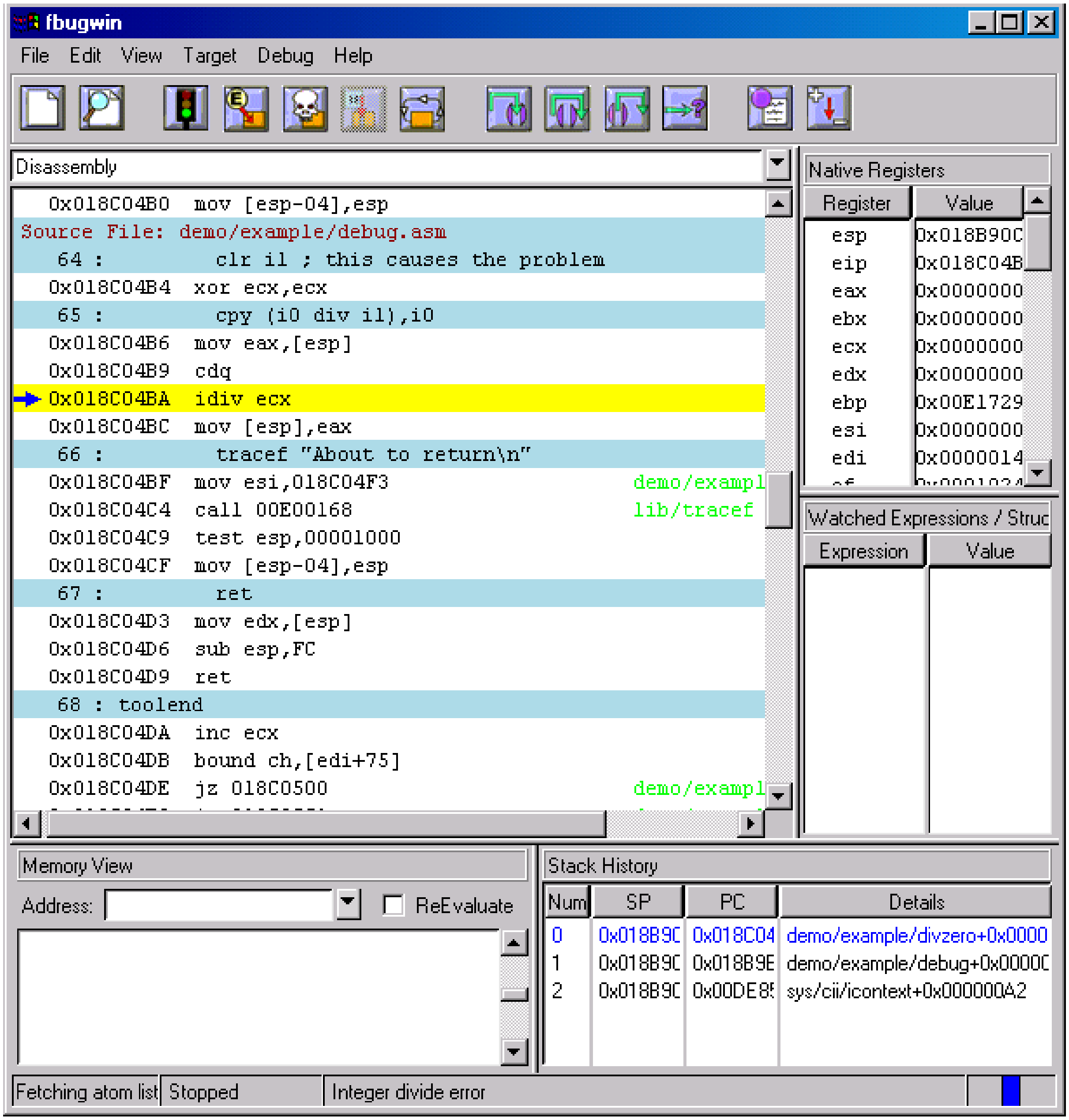

Figure 19. Divide by Zero

The status bar indicates that and ‘integer divide error’ has taken place and the source/disassembly window shows that the error has taken place on line 65, where i0 is divided by i1. Entering i1 as a ‘watch expression’ in the Watch Expression / Structures window confirms that i1 is indeed equal to zero. The highlighted line of disassembly also shows that ecx is the native register currently equivalent to i1.

In the case of fbug, the contents of *ecx* can be ascertained by typing in (V)iew (E)xpression, and then entering ecx on the command line.

Leave this error un-rectified for the next task.

7.5 Setting a breakpoint

To demonstrate this we are going to add a breakpoint to *debug.asm* using fbugwin (there are other ways, such as calling *sys/cii/breakpt* or the shell command *dbgbp*).

To set a breakpoint at the current cursor location access the Debug menu and Toggle Breakpoint (right-clicking on the relevant line and using Toggle Breakpoint from the context menu also works). To confirm where the breakpoint has been set, look at View Breakpoints.

This still leaves the original problem within the code, which was that the value of ecx was 0. Arbitrary expressions can be evaluated using the "Debug/Evaluate Expression" menu item. This includes the ability to assign values to registers. As such the following should be entered at the evaluate expression dialog.

```
ecx = 2
```

The special case of modifying register values can also be done using a context-menu in the native register window. If the user right-clicks on the register value that they want to modify, they should obtain a menu, containing (in this case) "modify value of register ecx." Finally, Go, also under the Debug menu can then resume execution, as does pressing the ‘traffic lights’ button.

To rectify this problem for the next task, comment out this line:

```
clr i1
```

which is commented as the source of the error in the code.

If using fbug, go to (B)reakpoint (S)et. Following this (V)iew (B)reakpoint will list the breakpoints that have been set. The register can then be modified by entering ecx = 2 under (V)iew (E)xpression and then typing in (R)un and (G)o in order to continue execution of our program. fbug should then hit the breakpoint which has just been set.

7.6 Incorrect use of noret instruction

The *noret* instruction is used to mark a point in the code that should never be reached. If execution does reach a *noret* instruction, the result is undefined. If running under fbug or fbugwin then this error will be trapped.

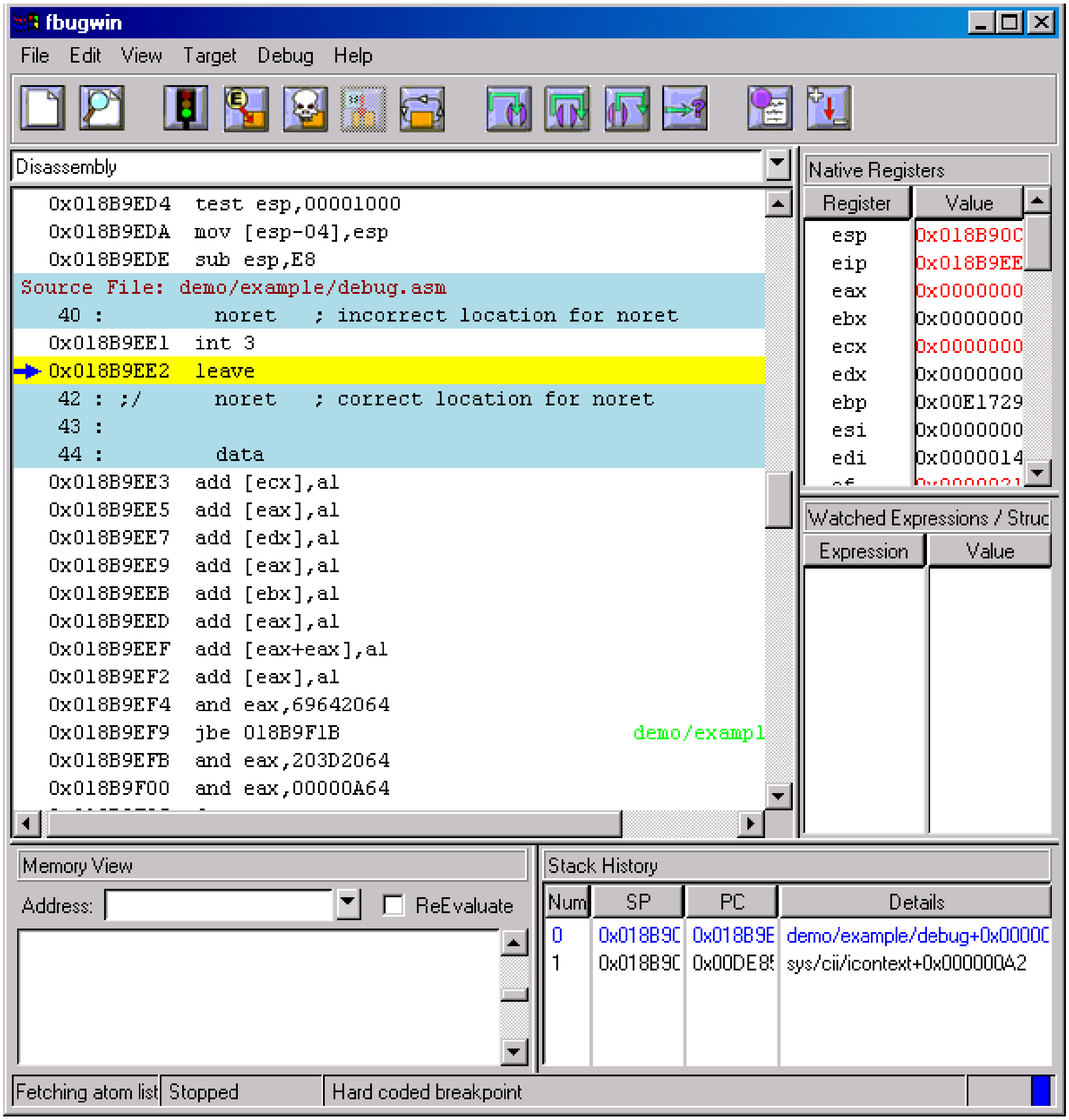

Figure 20. Noret

Omitting this instruction after a call, which is known not to return, does not alter the semantics of the code, but may make the translated code less optimised. To correct the error, comment out the noret instruction at line 40 of the source code and remove the comments prefacing line 42. Once this and the other errors have been commented out, *demo/example/debug.asm* should run normally.

8. Notes on Remote Debugging

The int**e**nt system being tested, otherwise known as the target system, runs upon one processor. The host system, which houses the int**e**nt debugger, runs upon another. These two processors are joined by any of the connections previously listed. After the debugger has been invoked, it communicates with a 'debug stub' on the target system via this link. Through this means, the debugger is able to 'interrogate' the target system.

The target system may be any recent version of int**e**nt for which a debug stub has been written. At the current time, debug stubs exist for the majority of supported platforms. Debugging cannot occur unless the appropriate debug stub device is loaded. Aside from the debugger program resident upon the host, and the debug stub contained in the target, no files are essential to the debugging process. It is even unnecessary for the target to contain a filesystem. However, source files are required on the host if the user wishes source information to be displayed by either fbug or fbugwin.

8.1 The Debug Stub

The int**e**nt debug stubs - the target based part of the int**e**nt debugger - are standard int**e**nt devices. There are separate debug stubs for each supported platform, and on each platform there are separate debug devices for each means of communicating with the debugger. A debug stub acts as an agent for the target int**e**nt session. It provides the debugger with the most basic debugging facilities:

- suspending and resuming execution
- examination and modification of CPU registers
- examination and modification of memory
- insertion and removal of breakpoints

Some of these facilities may not be available on all platforms. In particular, the exact behaviour of breakpoints is extremely platform dependent.

Some debug stub devices can be written to. The devices that support this will send the data written to the debugger host, as trace data. This feature is useful with embedded targets that have only one spare serial port that must suffice for both *ktraces* and debugging. Only one debug stub can be mounted at a time. It should always be mounted at */device/dbg.*

8.2 Loading the Debug Stub

In order for debugging to take place, the appropriate debug stub must be loaded onto the target system. This can take place before or after the debugger is invoked.

It is possible to load the debug stub by using either a sysgen directive or a "*devstart*" command. The *"devstart"* command is designed to be entered at the keyboard during run time. However, the target may be a system upon which no shell is running, and to which no keyboard is connected., in which case the sysgen directive may be used to ensure that the debug stub is started automatically when the system boots.

The sysgen directive will usually take the following form:

```
.obj (new tool=<_new filename>", mountstr="<mountname>", cmdline="debug
<textual parameters>")
```

The "*devstart*" command takes the following form:

```
$devstart <mountname> <pathname> <options>
```

8.3 Baud rate

By default, serial communication will take place at 19200 baud. However, it is possible to alter the speed to suit the platform. For example, it may be necessary to speed up the baud rate, e.g. 115200.

Where it is desired to alter the baud rate, fbug may be invoked with the –s option, as documented above. It is not possible to adjust the baud rate after the debugger has been invoked. When creating a target on fbugwin, the user is asked to specify a baud rate. This can be altered between intent sessions.

9. Other Utilities

Other useful utilities are available for debugging and analysis of code. These include:

- Data Flow Analyser (DFA)
- Test Coverage Analyser (TCA)

9.1 DFA

DFA is a data flow analysis utility which can:

- Check conformance with the VP2 specification
- Analyse register data flow and detect anomalies
- Emit modified tools containing selected optimisations

DFA currently supports VP tools in small tool format conforming to the VP specification.

9.1.1 Using DFA

Note that detailed information about various DFA command line options and operation can be found in *app/dfa/dfa.html*. DFA is designed to be run from the shell. It will accept either a single toolname or filename as input or will accept a series of arguments from stdin. DFA output is directed to stdout.

Usage:

```
dfa [-options] toolname[.00]
```

Certain tests are performed whenever DFA runs:

- Data flow analysis tests
- VP conformance checks

Other tests or optimisations are only performed when the relevant options are selected (see the -*r*, -*d*, -*b* and -*z* options).

Additionally, the options selected determine the level of detail and the format of the output (see the -*f*, -*m*, -*q*, -*t*, -*v* and -*s* options).

If it is intended to emit (write) a modified tool, the -w option must be selected; the modifications made to the tool then reflect the other options selected. -*b* makes a backup of a file before modifying it.
For example, if you want to find out where zaps can be added to a tool, use the -*z* option; if you want the zaps to be added automatically, use the -*zw* option.

As there are a large number of reports generated by DFA it is a good idea to start analysis using the -*fs* or -*ms* combination. If processing a large number of files use the -*qt* option. This allows the most significant problems to be pinpointed quickly. Effort spent chasing minor anomalies is usually wasted if there are also significant or fatal anomalies, since correcting the latter will nearly always change the former.

9.2 TCA

TCA is a test coverage analysis utility that measures the structural coverage achieved when testing VP tools. TCA currently supports VP tools in small tool format conforming to the VP specification.

9.2.1 Using TCA

For detailed information about using TCA please refer to *app/dfa/tca.html*.

The measurement of test coverage takes place in three stages:

- Instrumenting the object of the test
- Running the test
- Recording and analysis of the test coverage results

Usage:

```
tca [-options] toolname[.00]
```

9.2.2 Instrumenting The Object Tool(s)

Instrumenting the tool means modifying the tool binary; the modified tool is functionally identical to the input tool, but also has code inserted, which at each entry point and block entry, records the occurrence of execution of that code. When the modified tool is used in place of the original tool, the test coverage data is gathered.

TCA instruments tools using the *-i* switch. It will accept either a single toolname or filename as input or will accept a series of arguments from stdin. There is no output at this stage unless there are error messages; if a tool has already been instrumented, this will be reported to stdout.

The instrumentation can either be standard (*-i* switch) or instrument entry points only (*-ie* switch). Specifying the *-b* switch when instrumenting a tool causes the original to be backed up.

Instrumenting a tool also creates and initialises a test coverage data (.tcd) file. This file is maintained for the life of that version of the tool.

Note that instrumented tools are not intended for inclusion in any deliverable build; they are only intended for the purposes of test coverage measurement.

9.2.3 Running The Test

The test is run in the normal way. An instrumented version of a tool should behave identically to the original; if it does not, this normally indicates that the tool is performing some illegal operations within VP. As the tests are run, a number of named data areas are created and maintained in the int**e**nt system:

- For each instrumented tool, a named data area is created which contains the test coverage data for that tool. The first time the instrumented tool is called, the named data area is created and initialised. Each subsequent call to the tool, and each time any code within the tool is exercised, the relevant parts of the named data area are updated.

- There is a single master named data area with a well known name that contains a list header. All of the extant tool ndas are maintained on a standard int**e**nt doubly linked list. As instrumented tools are called and their ndas created, the ndas are added to the list. Whilst there are any tool ndas in existence, the master nda also exists.

Note that the data maintained in named data areas whilst tests are being run is only transient, and only applies to that series of tests. The data must be committed to file to become permanent.

Whilst running the tests, typing tca -l in the shell will return a list to stdout of all tools for which named data areas currently exist (if any). This provides a means of discovering which tools are being exercised at any instant.

If a tool is updated and re-instrumented whilst tests are going on, this event will be detected the next time the tool is called, and the named data area re-initialised. Therefore there is no need to quit tca before substituting an updated version of a tool.

9.3 More Utilities

9.3.1 The Debug Device Driver

The device driver call-tracing driver is a standard device driver object. It is intended to sit between the user and a real device driver. It simply passes calls through to the underlying driver, optionally printing tracing information as it does so. It is principally intended as a debugging aid, and since the tracing can be turned on and off at run time it should be useful to those without access to source code.

It provides methods that allow it to be used as a filesystem device, character device, block device, pointer device, or keyboard device.

It is possible to run several call-tracing drivers simultaneously to trace different underlying devices. The tracing output from the drivers is guaranteed to only break at a line end, so while lines of output from two drivers may be interleaved, a line from one driver will never be broken in two by output from the other. For more information see *dev/dbg/elate/user.html*

9.3.2 The Debug Memory Allocator

This allocator keeps track of extra information about allocated memory blocks in order to facilitate tracking of memory leaks. The Debug memory allocator is a subclass of memory allocation object. It is designed this way to enable it to run on top of a variety of different underlying memory objects with a minimum of configuration effort.

The allocator keeps a list of currently allocated blocks, together with the PID of the process that allocated them, and a quantity of stack information at the time of allocation. For more information please see *sys/kn/mem/debug/user.html.*

9.3.3 Exception Flight Recorder

The Exception Flight Recorder is a set of linked programs that provide low-level details of hardware and software exceptions to aid in debugging. There is a user-level interface to start the recorder and configure it to report the desired information. For more information please see *sys/dbg/flightrec/flightrec.html*