


[TRACE32 Online Help](#)

[TRACE32 Directory](#)

[TRACE32 Index](#)

TRACE32 Training	
Training Simulator and Demo Software	1
Introduction	2
About the Demo	2
The User Interface	4
Display and Modify Memory	7
Debug the Program	10
How to Set Breakpoints	17
Breakpoints	17
Display and Modify HLL Variables	22
Format HLL-Variables	25
Exit TRACE32	27

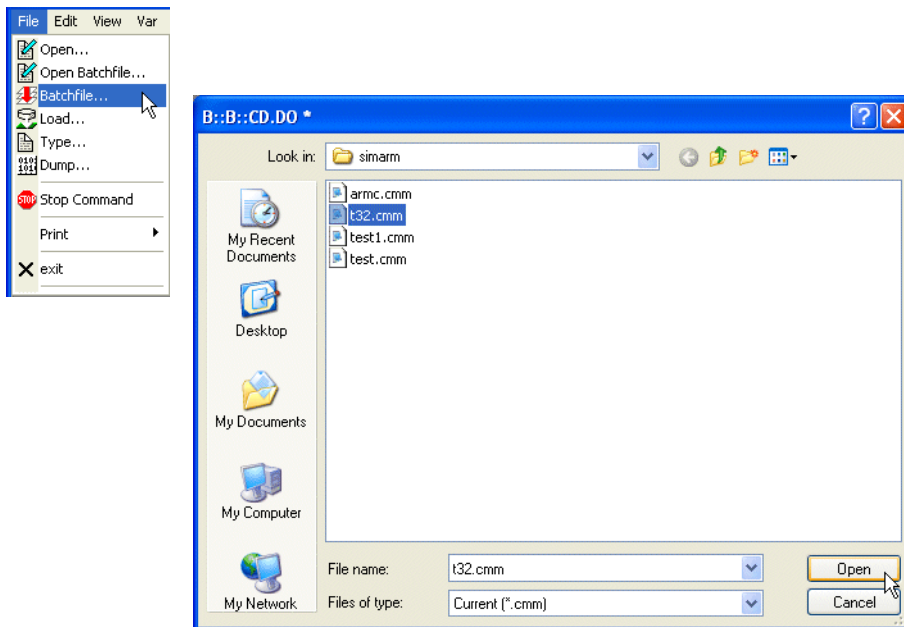
Introduction

About the Demo

How to use this tutorial:

The tutorial contains a guided debug session. It uses a simple C-program example to show you the most important debug features.

When the simulator is started a script file that sets up a debug session is automatically executed. The script file can also be started by using the **Batchfile** command in the **File** menu.



You should perform a number of exercises as you read this tutorial. We **recommend to go completely through all chapters**, since besides the tour (written in normal text format) there are very helpful remarks (*written in italics*) which will not be repeated in other chapters.

How long does it take?

30 minutes.

The User Interface

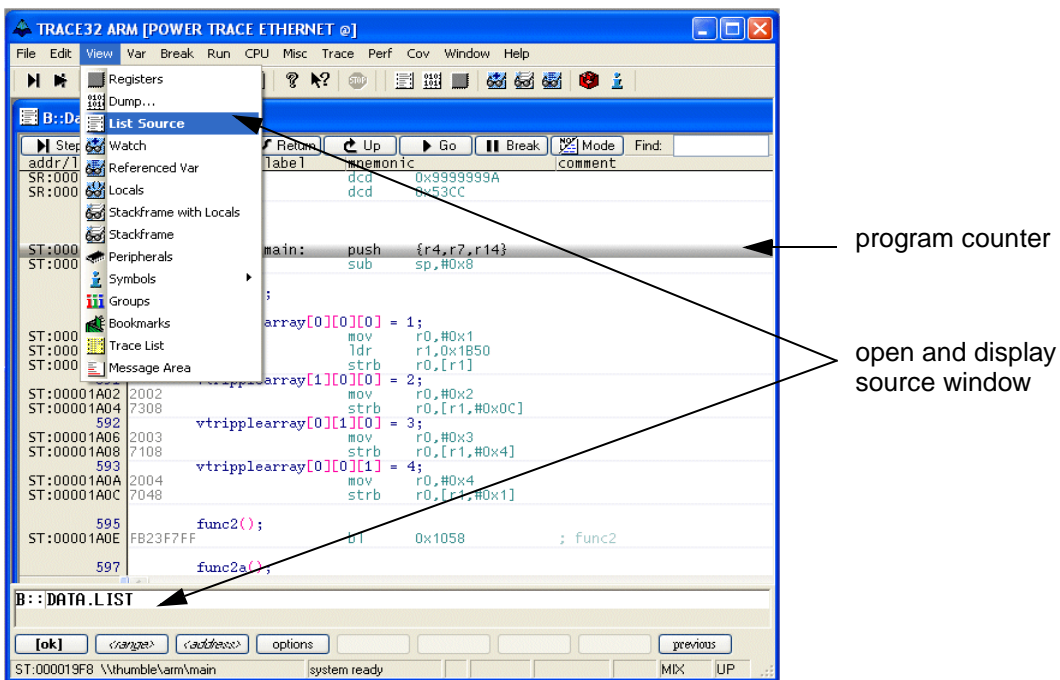
TRACE32 uses several windows to display the information the user requests. One of the most important aspects of the application is the code view. The command **Data.List** will display the disassembled/decompiled code in an own window. According to our startup script "start.cmm" in the section **Building Batch Jobs** we loaded the application and instructed TRACE32 to display the source code in a **Data.List** window .

```
; set window position and display Data.List window
```

```
WinPOS 0% 0% 100% 100%
```

```
Data.List
```

If you do not see the source code (in the Data.List window) because you have used a different startup script, you may open it by the command "**Data.List**" in the command line or the menu item "**List Source**" in the **View** menu. Both methods are displayed in the screenshot below.



The gray bar indicates the current position of the program counter. Right now it is located on the main address symbol. This is because it was instructed in the startup script, as follows:

```
; set cpu register
```

```
Register.Set pc main
```

The most functions of TRACE32 can alternatively be selected in three ways:

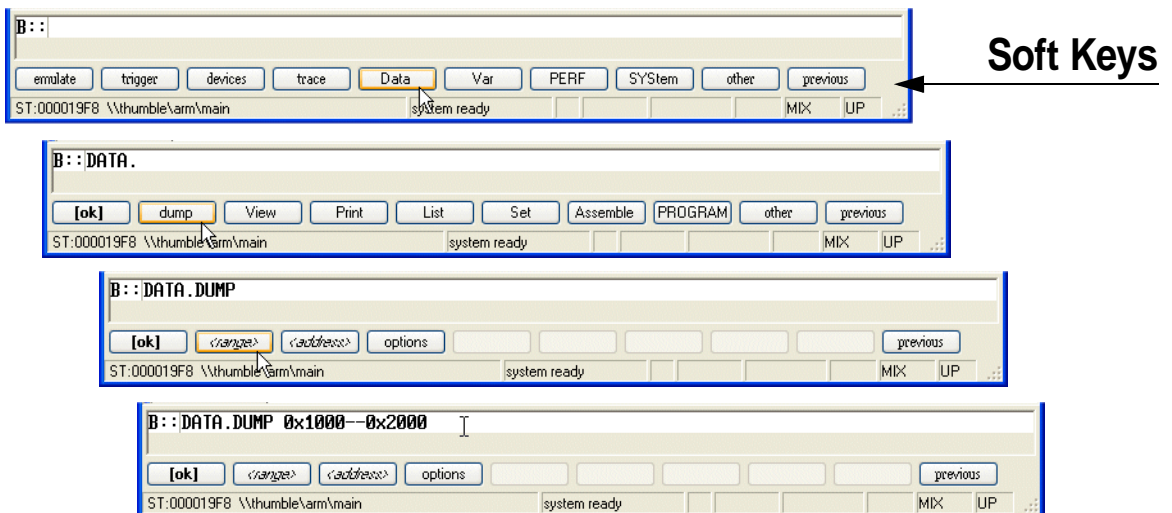
1. from a pulldown menu
2. from a button in the main tool bar
3. from the command line.

Please refer to the previous screenshots and remember it, even if we mostly use the command line in the following chapters.

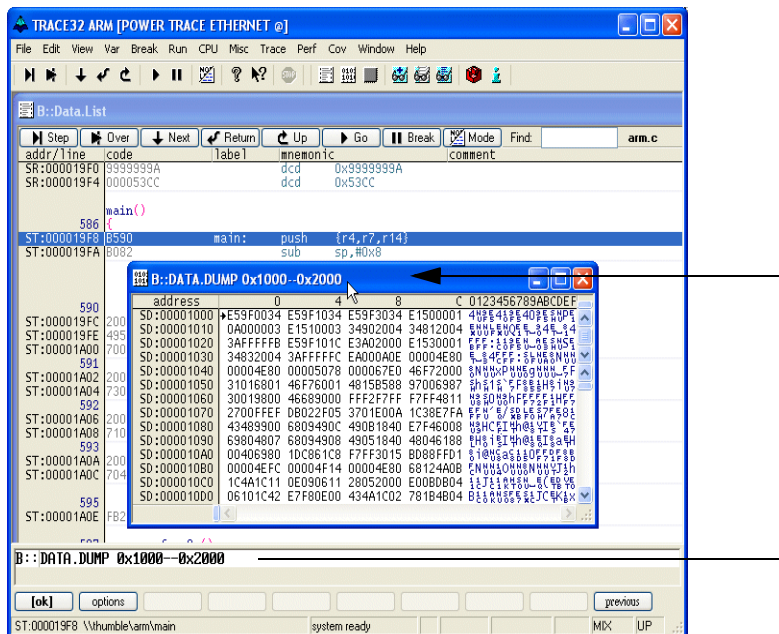
The TRACE32 commands are not case sensitive. In the TRACE32 books we use upper case letters for the characters that are significant for the command entry. E.g. `Register.view` can be shortened by `r` or `R`. Another example which shows the typical TRACE32 command structure `<command family>.<subcommand>` is `Data.List` that can be shortened to `d.l` no matter whether it's lower case or upper case.

A good hint is to look at the soft keys. They provide a guided command entry. All possible commands and parameters are displayed. Instead of writing to the command line you can assemble the correct command by clicking on the soft keys.

Example: Assembly of the `Data.dump` command by using the softkeys.



Regardless of the method a command was initiated (button, menu or in the command line), TRACE32 displays the command that was executed in the window title.



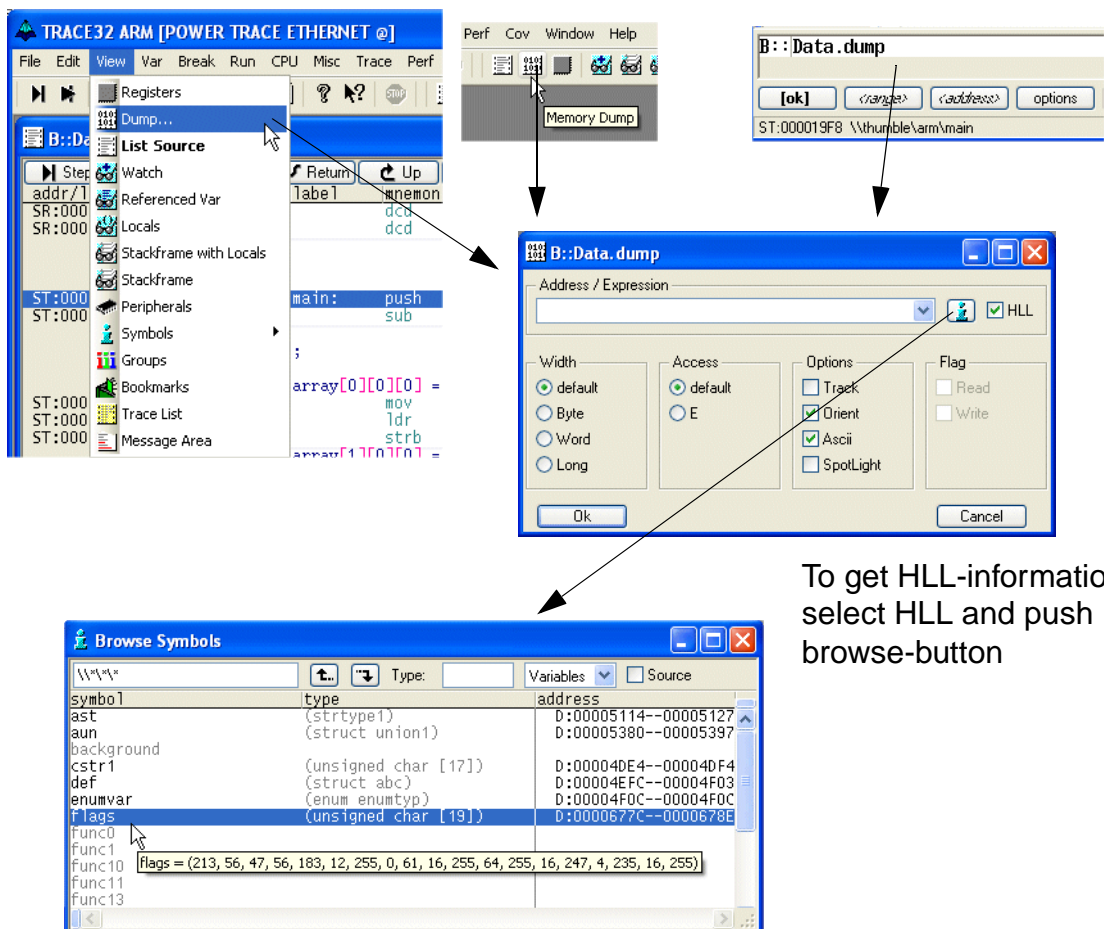
Working with the menu, softkey or buttons in the tool bar may be helpful in figuring out the command .

Display and Modify Memory

To inspect an address range in the memory use the **Data.dump** window as seen in the previous section. Type in the command `data.dump <address range>` or select

- **Select Dump...** from the **View** menu.
- Push the icon in the toolbar.
- Type in the command **Data.dump**.

Next a dialog will be opened. Fill in data item to display in the following dialog box. Use the **Browse** button to browse through the symbol data base. Select a label by a double-click and then confirm by pushing **OK**. If using the command in the command line, you may also specify an adress or symbol directly.



To get HLL-information select HLL and push browse-button

The data will be displayed in an own window. It's title reflects the calling method an parameter.

In the following screenshot it was called by a command in the command line. .



address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
SD:00006770	47	62	57	E4	A0	70	99	82	6D	09	3D	90	D5	38	2F	38	GbWEA40p32mH=SD8/8
SD:00006780	87	0C	FF	00	3D	10	FF	40	FF	10	F7	04	EB	10	FF	40	8FFH=1F@F1FEE1F@
SD:00006790	BF	60	FB	00	EF	A8	FF	80	FF	04	FB	00	FD	11	FF	01	7FFU0F0F07TB0F@
SD:000067A0	67	B6	BB	30	E9	81	DF	73	BD	58	FD	E2	FA	02	C7	18	F8FNEAF8FFNF1FS
SD:000067B0	FD	82	FF	00	FE	68	57	00	97	44	FB	52	3F	BC	EE	C4	F8FNEAF8FFNF1FS
SD:000067C0	FF	24	7F	00	FF	10	BD	00	FE	00	DF	00	EE	D0	FF	02	F8FNEAF8FFNF1FS
SD:000067D0	BE	00	FF	00	FB	00	FF	00	FD	01	EF	00	FF	20	FF	20	F8FNEAF8FFNF1FS
SD:000067E0	2E	14	18	54	BB	00	BF	32	F6	E0	E3	80	5B	EC	9F	04	F8FNEAF8FFNF1FS
SD:000067F0	9B	33	D9	80	BD	20	7F	10	96	9A	8F	31	27	54	FF	CC	F8FNEAF8FFNF1FS

Memory Class + Address

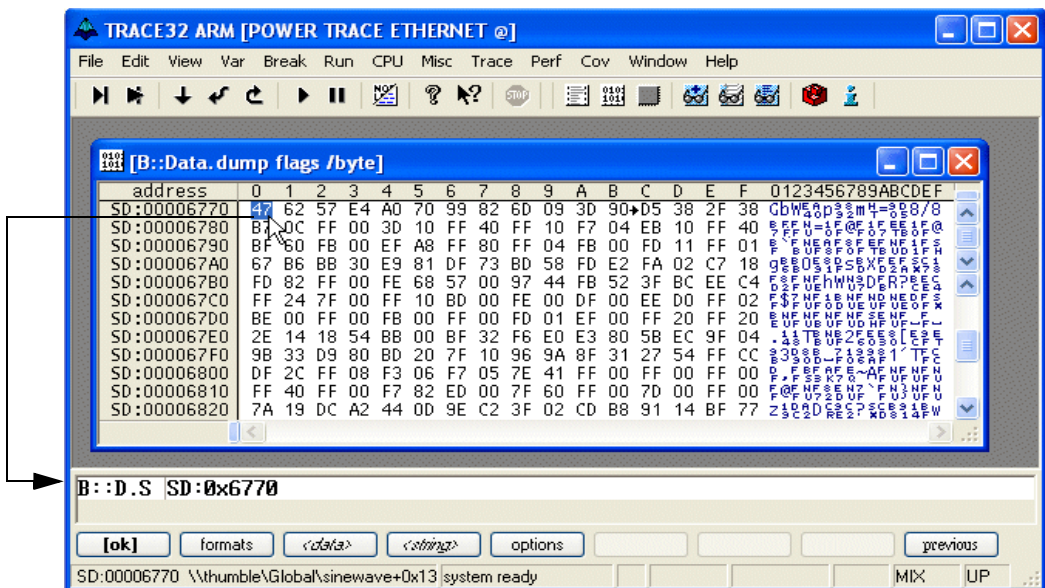
Hex-Value

ASCII-Value

There are different display options and ways to define an address range:

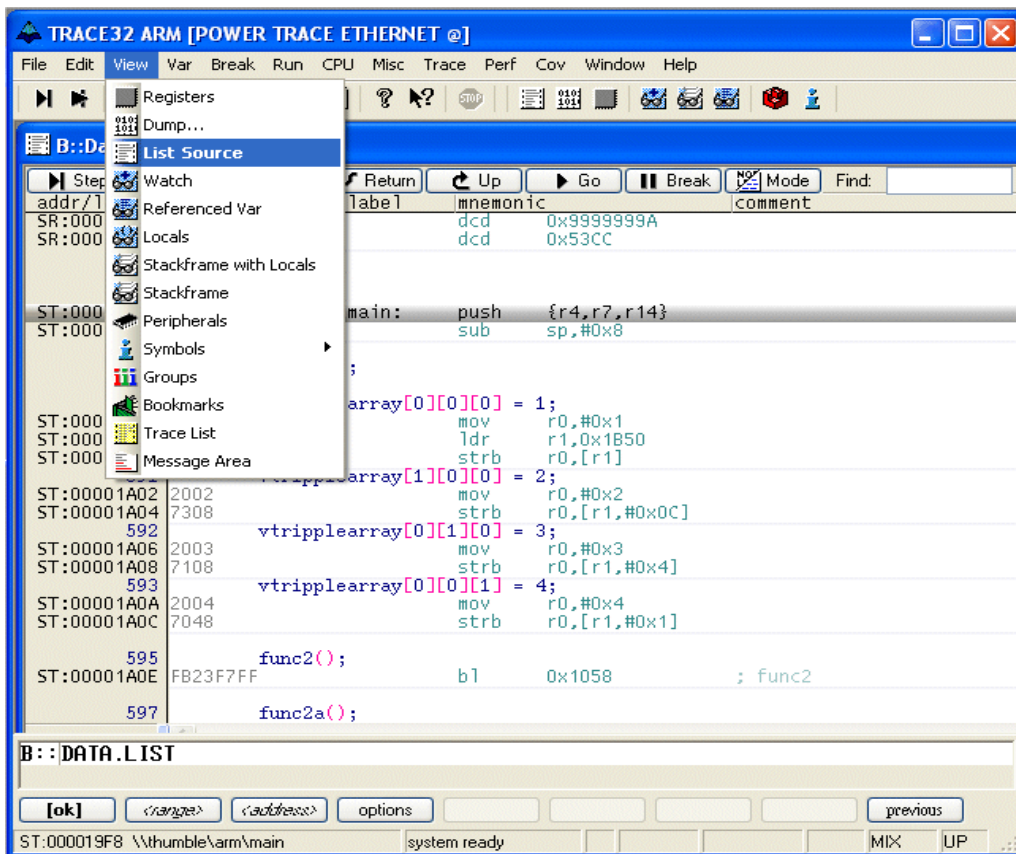
- <start address>--<end address>
- <start address>++<offset>

The value at a memory address can be modified by a double click. A **Data.Set** command for the selected address is displayed in the command line. Enter the new value and confirm it with return..



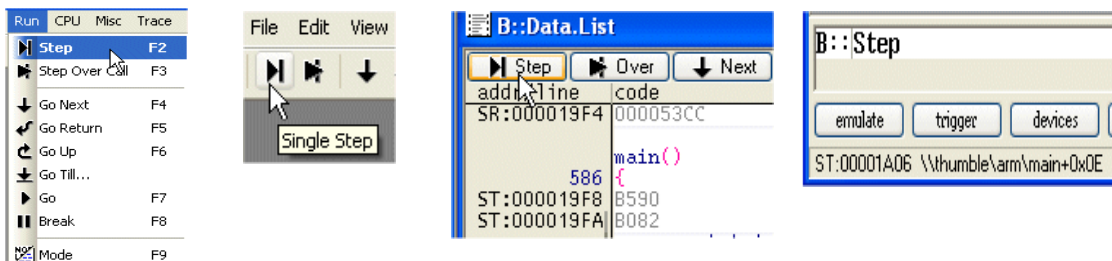
Debug the Program

Open the **Data.List** window by using **List Source** in the **View** menu. The program listing around the program counter is displayed.



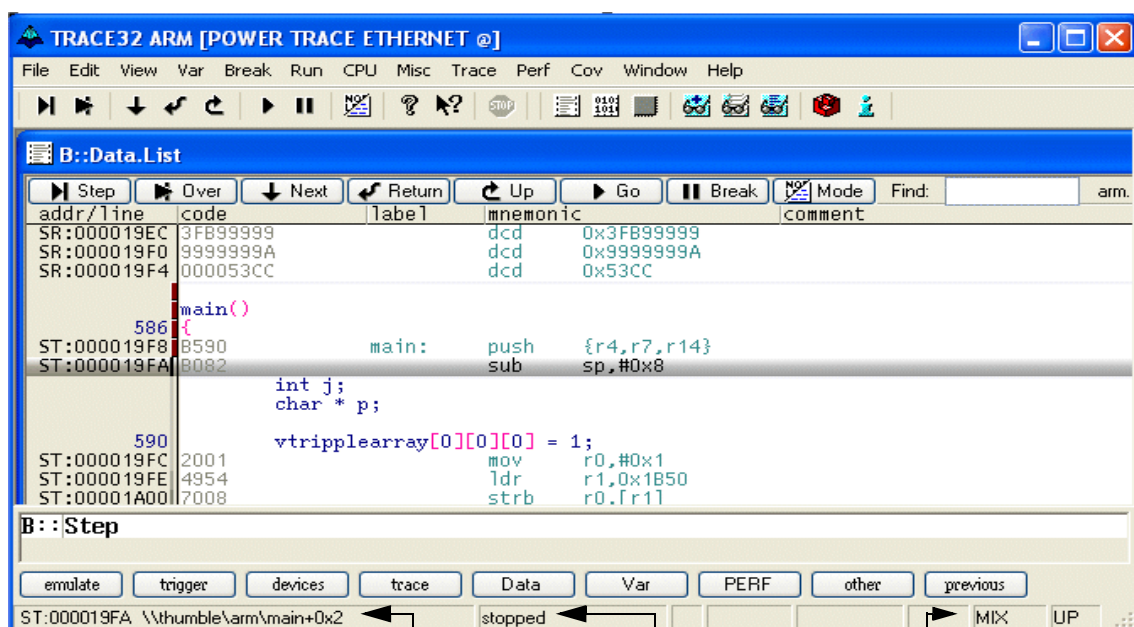
Single step through the program by clicking one of the following:

- the **Step** menu option in the **Run** menu
- **<F2>** (as noted in the step menu option in the **Run** menu)
- the **Step** button in the toolbar
- the **Step** button in the **Data.List** window
- the **Step** command entered in the command line



Now have a look at the state line: The address of the current cursor position (gray bar in active window) is displayed there.

The next field displays the state of the debugger: **stopped** means your application program is stopped. You can now for example inspect or change memory. .

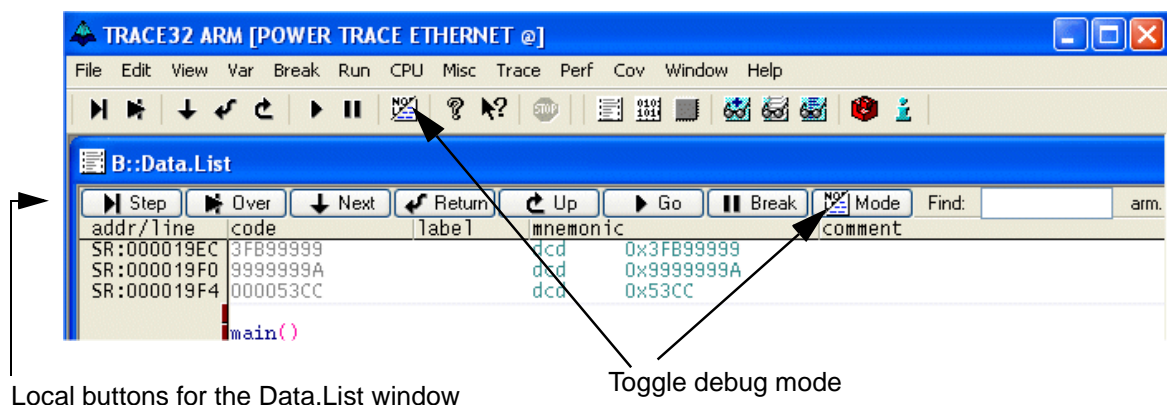


(symbolic) address at the cursor position

state of debugger

debug mode

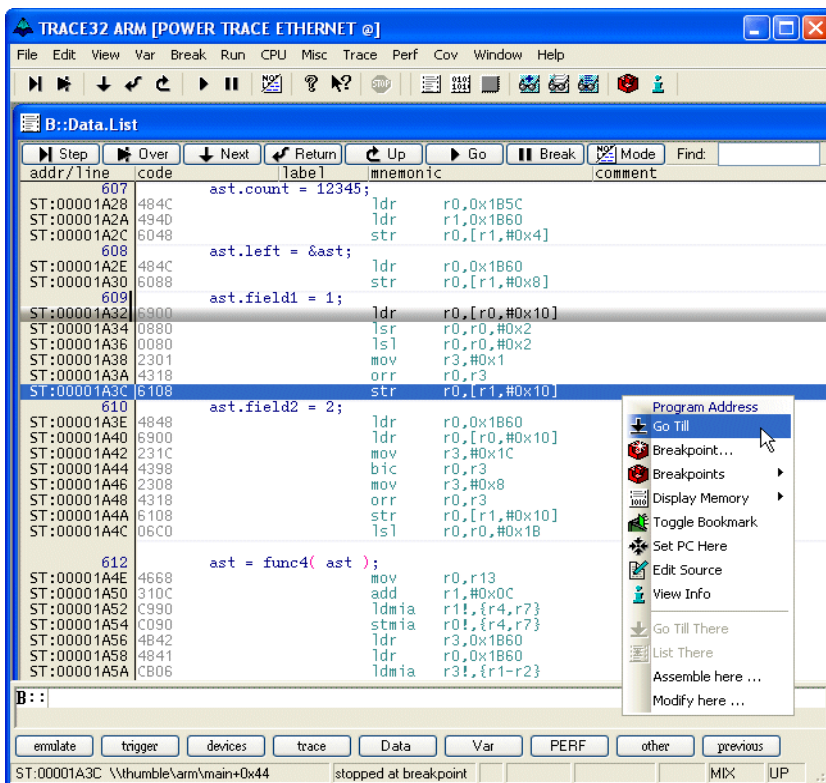
Depending on the startup script which was used, the display of the code will be either HLL (High Level Language) only or a MIXed mode with HLL and it's corresponding assembler mnemonic. The debug mode field in the state line indicates, which mode is used. Pushing the mode button on top of the Data List windows will change the mnemonic to HLL and vice versa. The state line always shows the debug mode HLL or MIX.



Local buttons for the Data.List window

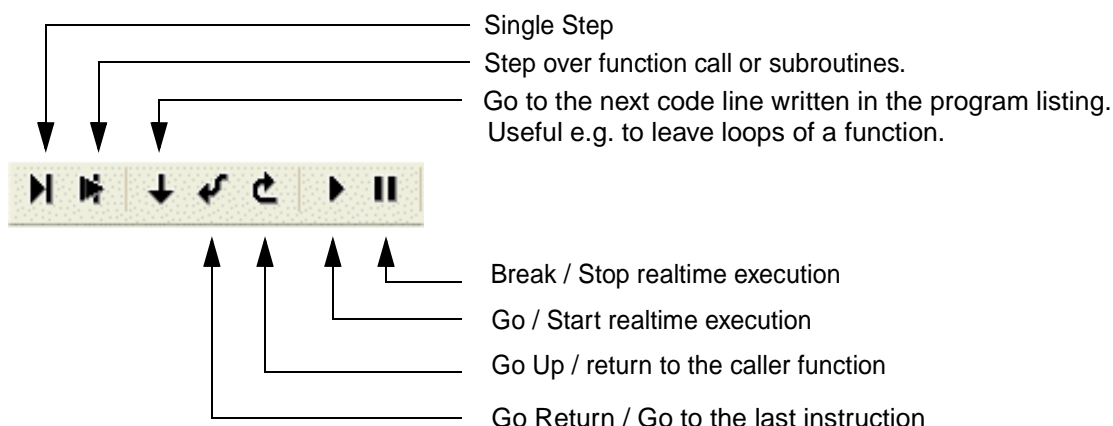
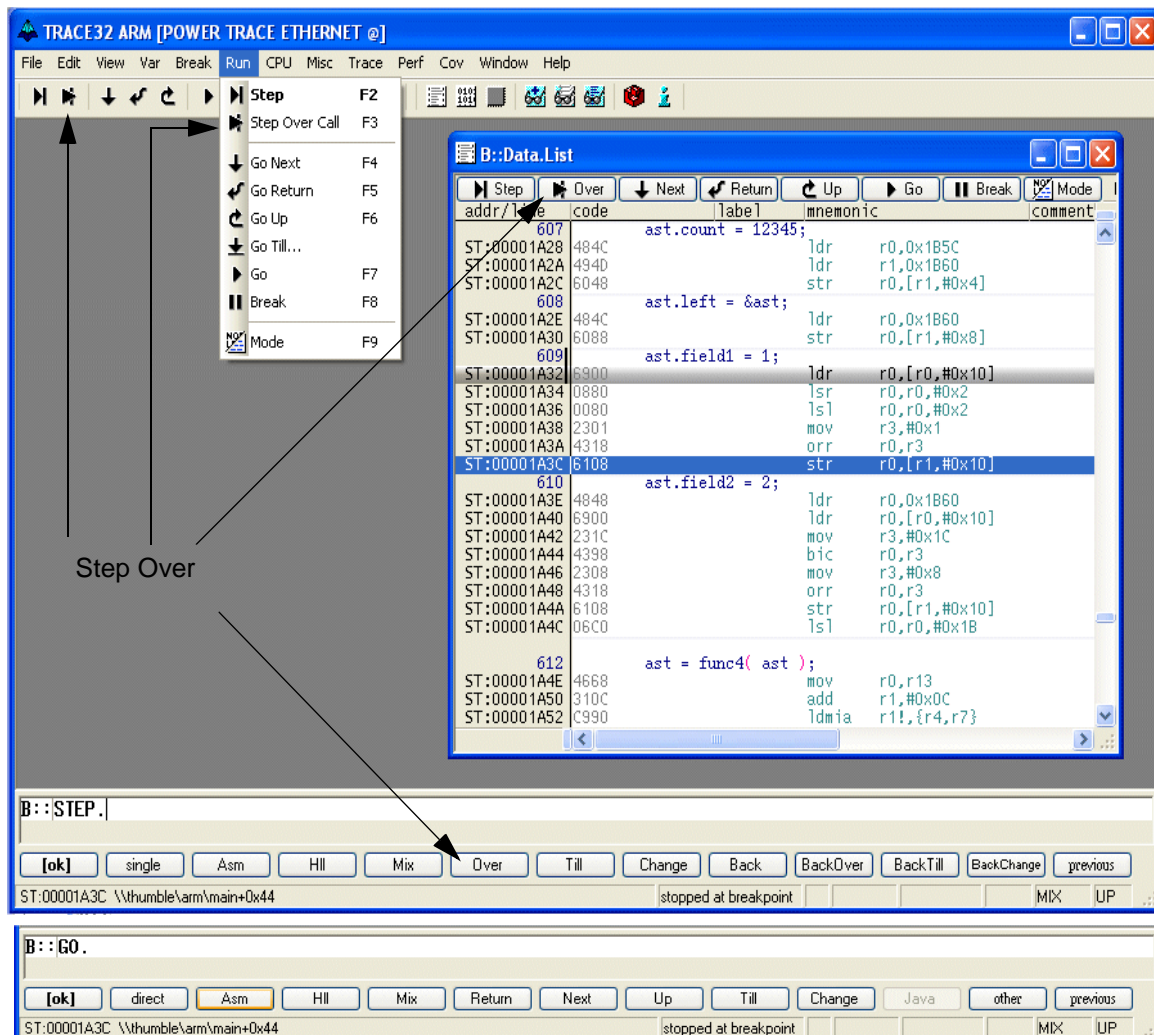
Toggle debug mode

Toggle the debug mode to **HLL** and do another **Step**. The step you did was a high level language step to the next HLL line. If you switch to MIX mode and push the step key, it will execute one assembler line.

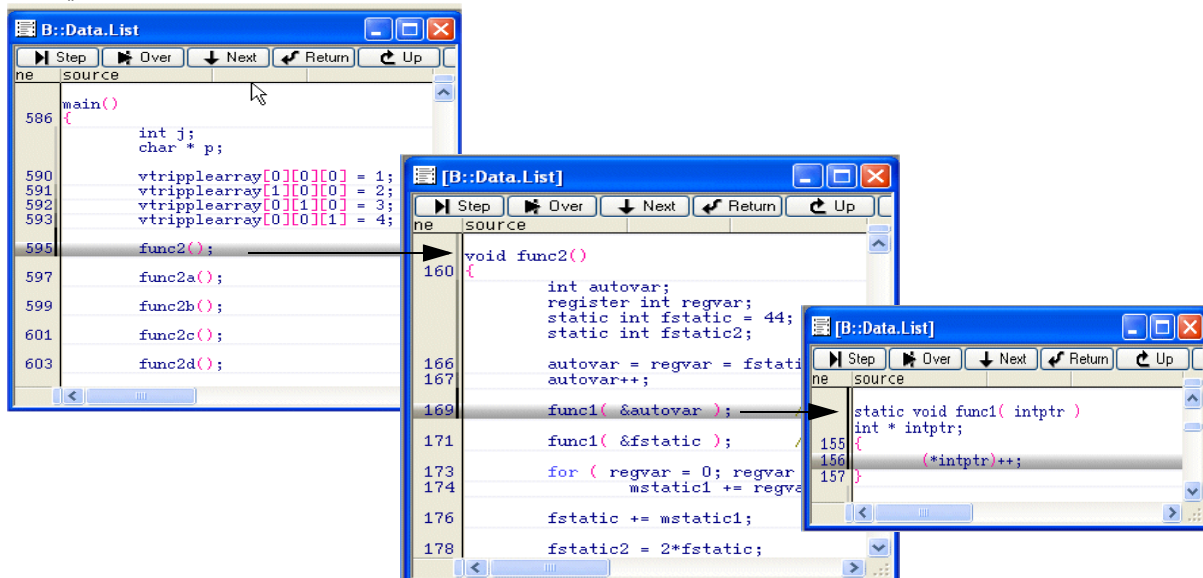


Select a code line and press the right mouse button. If you select **Go Till** the program execution is started. It stops when the program reaches the selected code line.

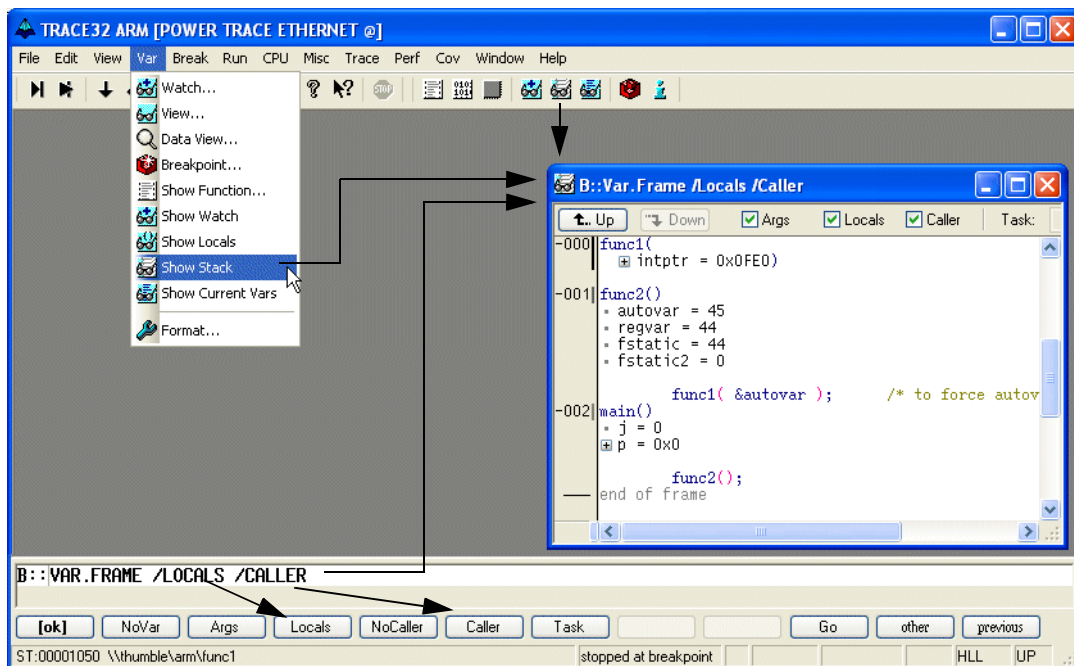
Single Stepping is one of the basic debugging commands. Look at the **Run** pulldown menu, at the local buttons of the **Data.List** window or at the main tool bar for the other debug commands.



For the following example, assume we have a nesting of functions where main calls func2() and func2 calls func1().



The **Var.Frame** window displays the function nesting for your application program. With the option LOCAL the local variables of each function are displayed. When the option CALLER is set, a few lines from the C-code are displayed to indicate where the function was called. The following screen corresponds with the nesting and calling sequence as mentioned above..

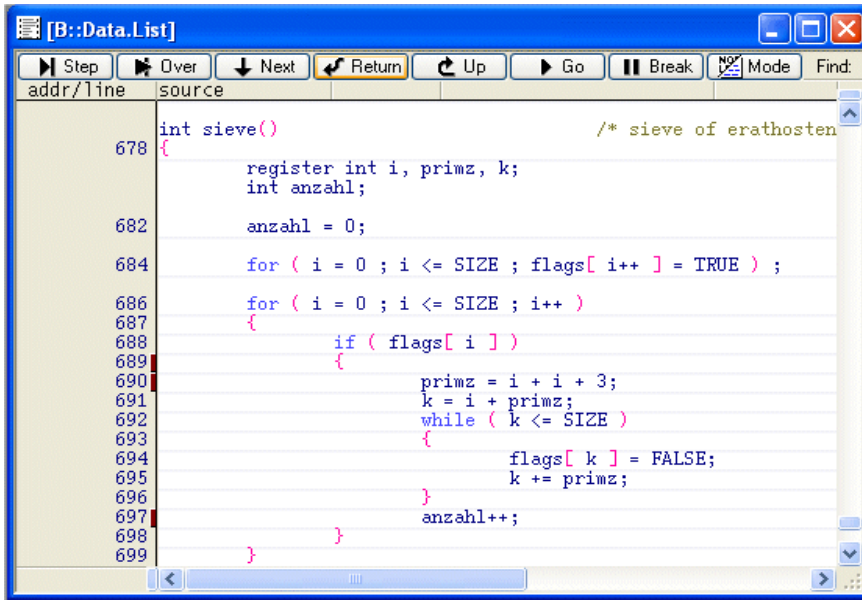


TRACE32-ICD provides also more complex debug control commands. You can run or step until an expression changes or becomes true. Example: *Var.Step.Till j>9* single steps the program until the variable *j* becomes greater than 9.

How to Set Breakpoints

Breakpoints

Back to the program. Doubleclick on the code line where you want to set a program breakpoint. Be aware to click the white space in the code line, and not the code literal. All code lines to which a program breakpoint is set are marked with a small red bar.



Use **List** from the **Breakpoint** menu to display the information about all breakpoints.

The format of the Break List is as follows

4. Column 1: hex address of breakpoint.
5. Column 2: type of the breakpoint.
6. Column 3: indicates how breakpoint is realized - as SOFTWARE, ONCHIP or DISABLED. A gray bar in Data.List indicates a DISABLED breakpoint
7. Column 4: code line of breakpoint. E.g. func2\6 means HLL-line 6 in func2. func2\13+0x8 means HLL-line 13 in func2 plus 8 bytes (useful in display mode MIX only).

The image shows two screenshots of the TRACE32 ARM software interface, demonstrating the Break List and Data List windows.

Top Screenshot: The main window displays the source code of a function `func2`. The `B::Break.List` window is open, showing a list of breakpoints. The `B::Data.List` window is also open, showing the source code with the breakpoint locations highlighted.

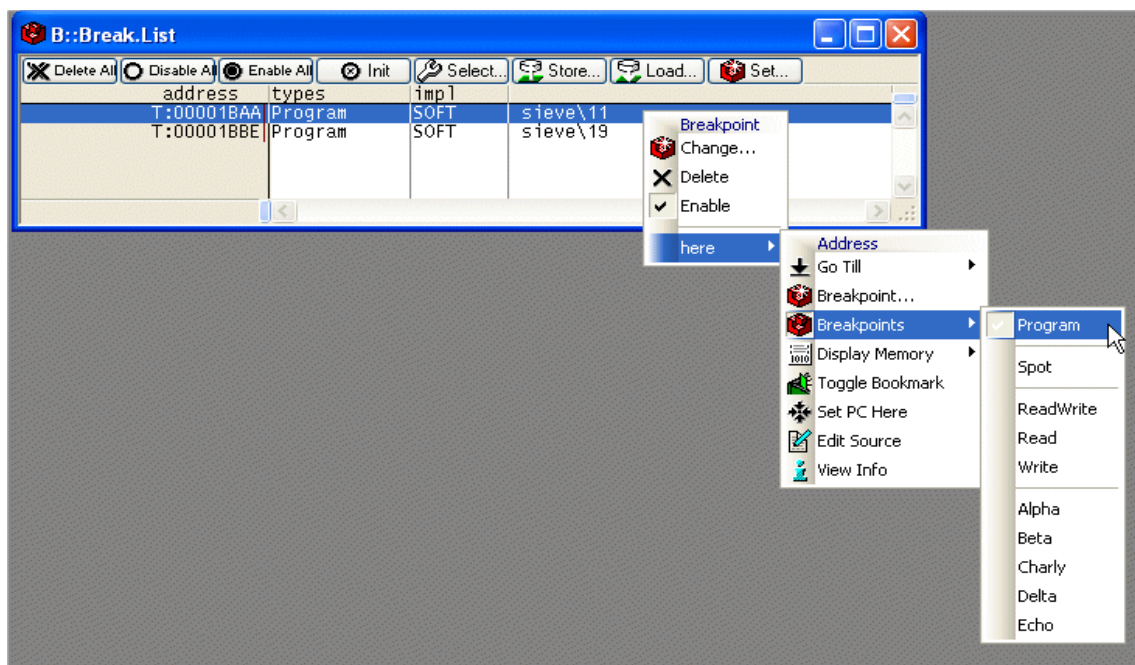
address	types	impl	func2
T:00001058	Program	SOFT	func2
T:0000105A	Program	DISABLE	func2\6
T:00001060	Program	SOFT	func2\7
T:00001066	Program	DISABLE	func2\9
T:0000106C	Program	SOFT	func2\11

Bottom Screenshot: The main window displays the assembly code of the function `func2`. The `B::Break.List` window is open, showing a list of breakpoints. The `B::Data.List` window is also open, showing the assembly code with the breakpoint locations highlighted.

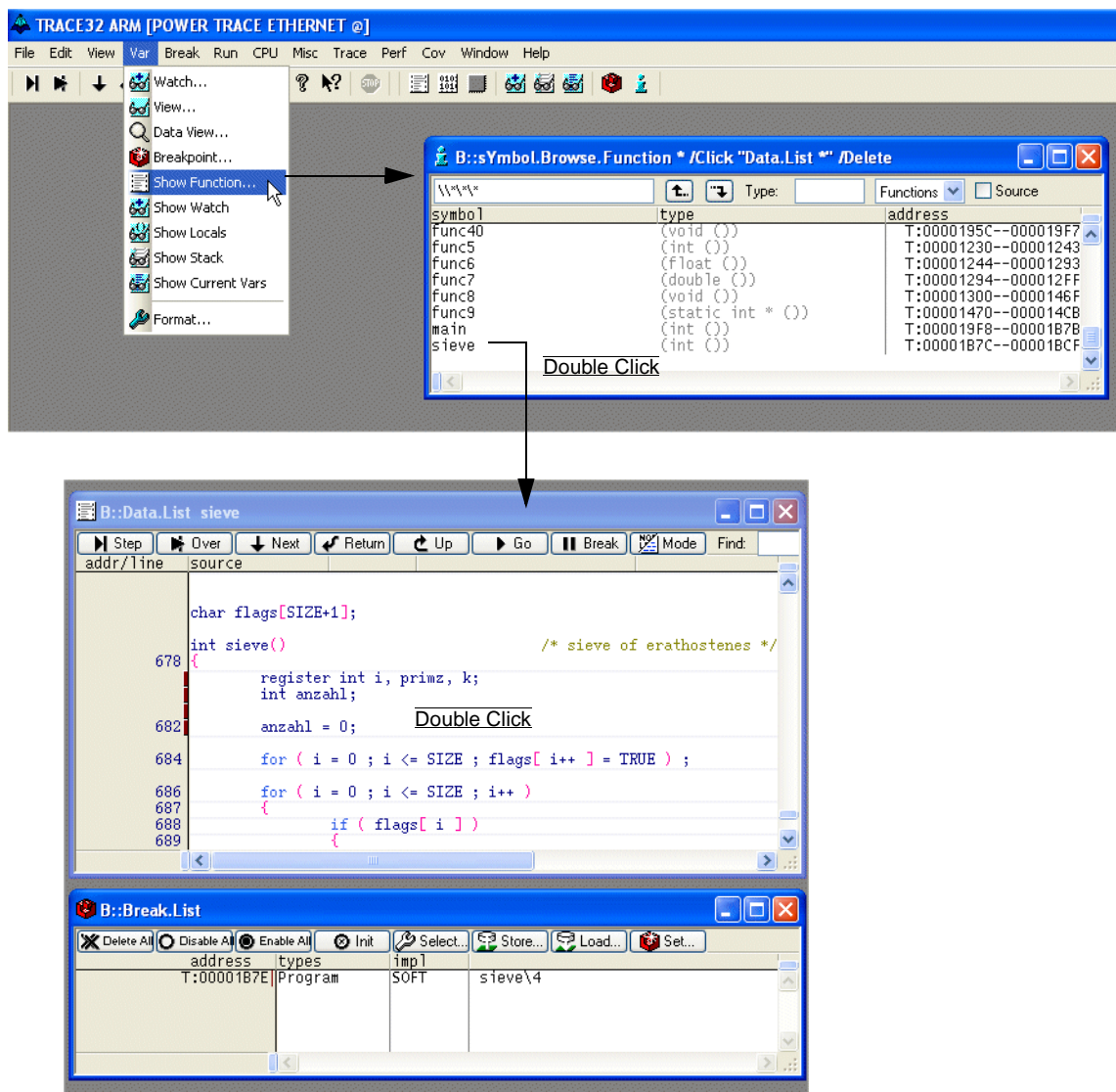
address	types	impl	func2
T:00001072	Program	SOFT	func2\13
T:0000107A	Program	SOFT	func2\13+0x8

Start the program execution with **Go**. If the program does not reach your breakpoint, you can stop the program execution with **Break**.

You can remove the breakpoint by another doubleclick to the marked line or by toggling the breakpoint in the **Break.List** window.

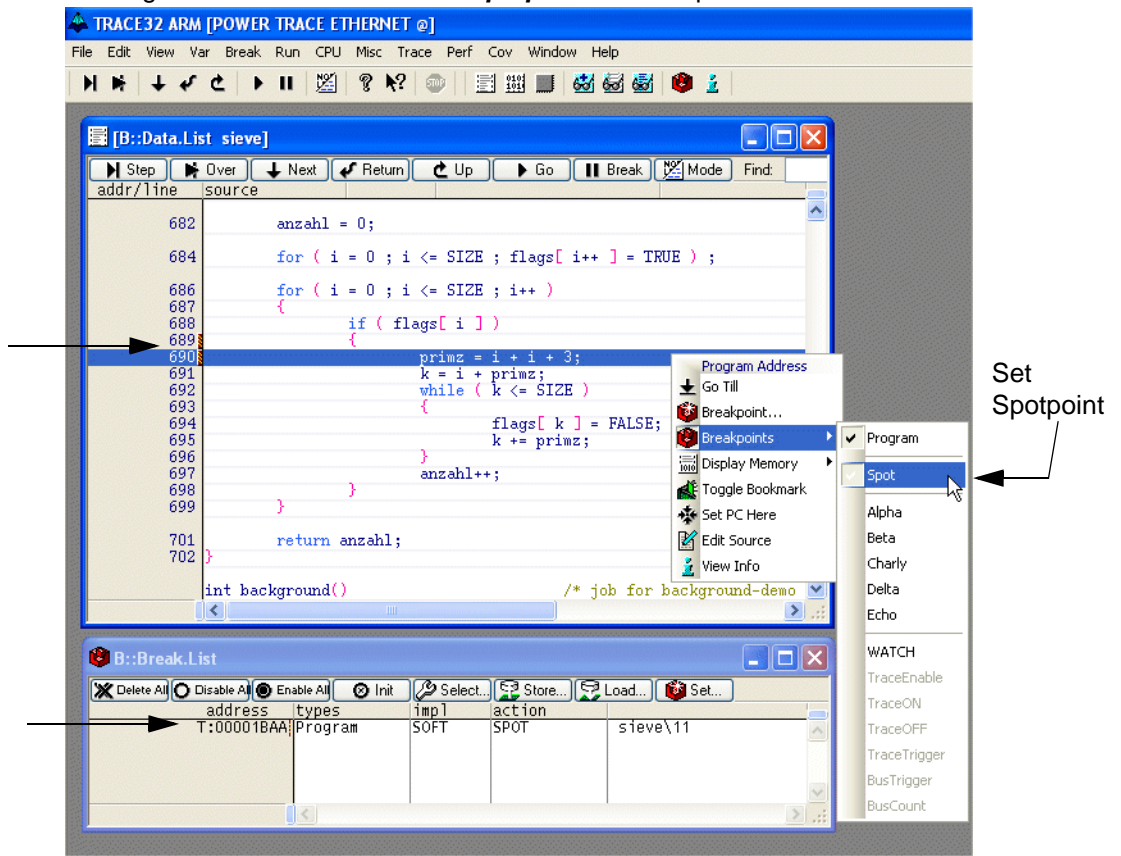


To set a program breakpoint to a code line, that is not displayed yet, select **Show Function...** in the **Var** menu. Double click the function to display it and then set the breakpoint by a doubleclick to the code line.

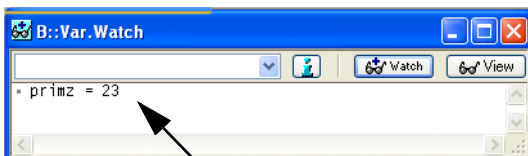
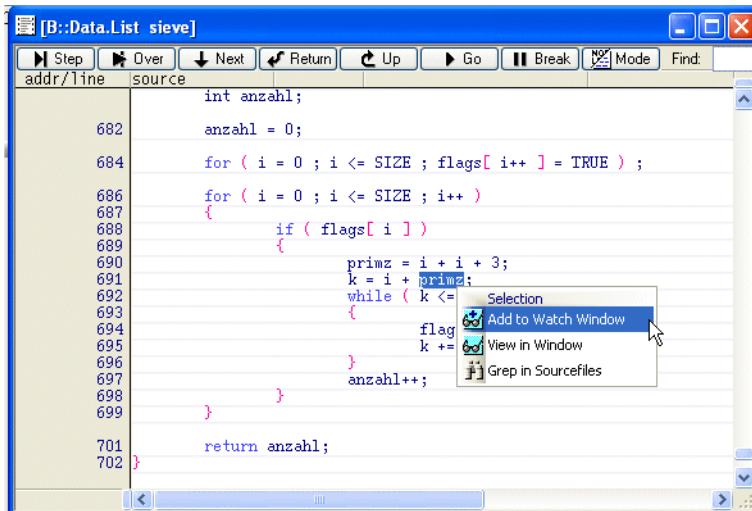


The second breakpoint type is a spot breakpoint. A spot breakpoint is a watchpoint, that stops the program execution for a short time to update all displayed information and then restarts the program execution.

To set a spot breakpoint, select the code line where it makes sense, that the displayed information is updated. Press the right mouse button and select **Spotpoint** from the pulldown menu.



To watch for example all changes on the variable `primz`, select the variable by the mouse, press the right mouse button and apply **Add to Watch Window** from the pulldown menu.

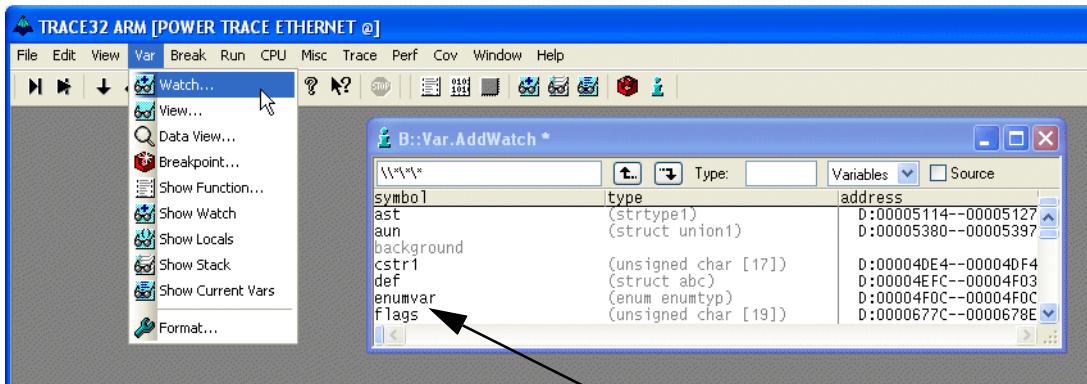


If you now start the program execution with **Go**, you can watch the changes on the variable `primz`.

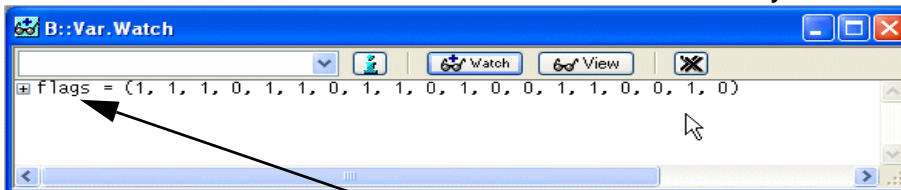
8. Assign a breakpoint in `func2a` and `func2b`.
Check breakpoints by doing a `Break.List`.

Display and Modify HLL Variables

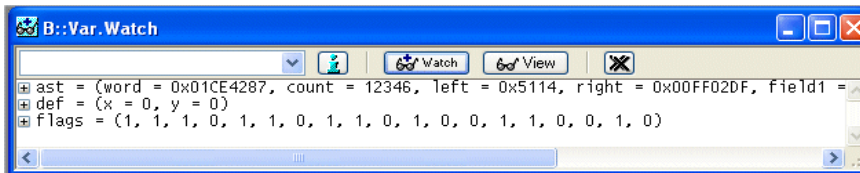
To display HLL variables use the **Watch** command from the **Var** pulldown menu.



Select the variable by a double click from the symbol database

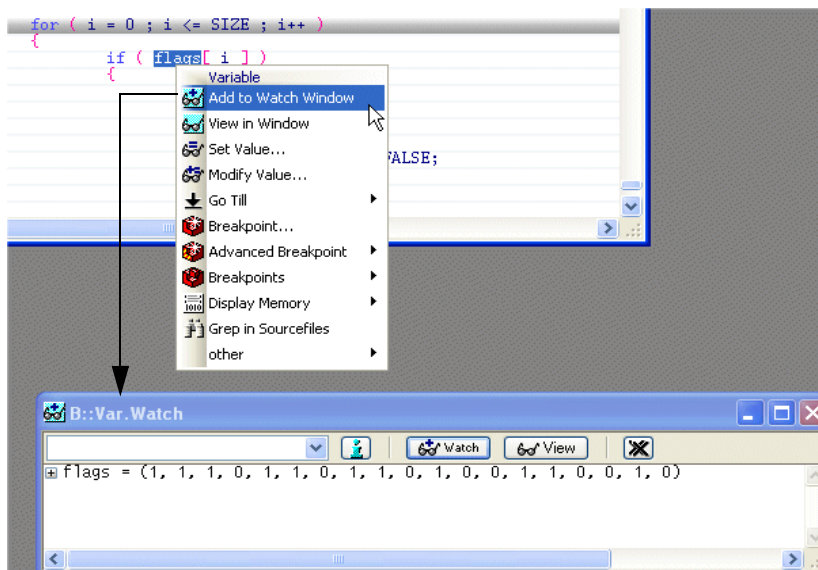


The selected variable is displayed at the top of the **Watch** Window



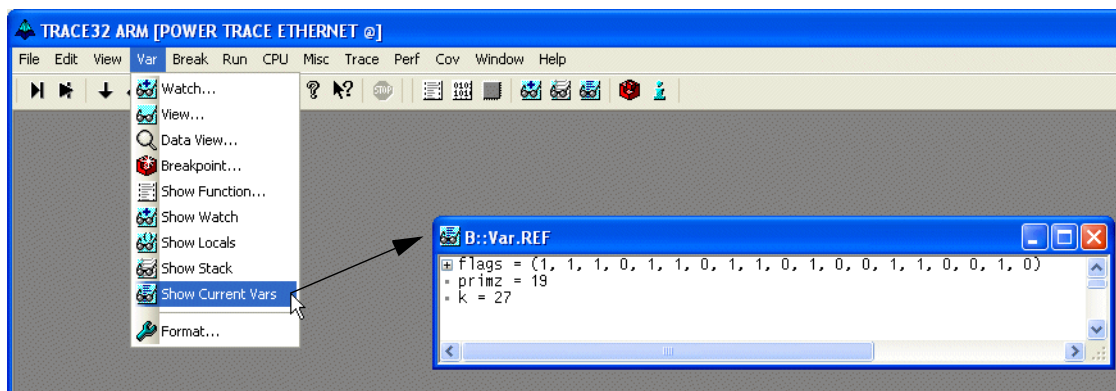
Every time you use **Watch** from the **Var** menu, the new variable is added to the top of the window. Resize the window to see all entries in the **Watch** Window.

A quicker way to look at a variable is to mark the variable in the **Data.List** window by the cursor and to press the right mouse button. From the **Var pulldown** menu select **Add to Watch Window**.



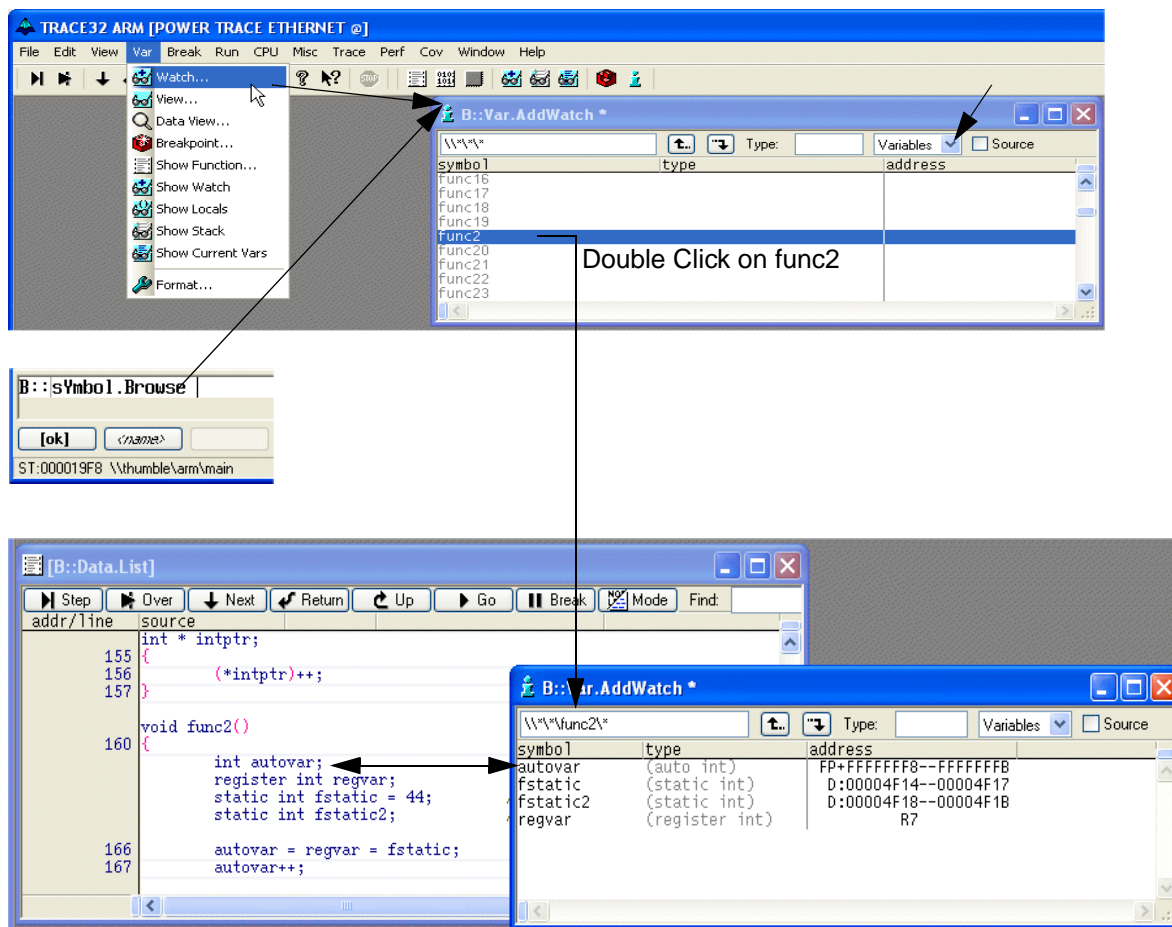
If you want to display a more complex structure or an array in a separate window use **View...** in the **Var** pulldown menu.

If you just want to watch all variables accessed by the current program context use **Show Current Vars** from the **Var** pulldown menu and execute a few single steps.

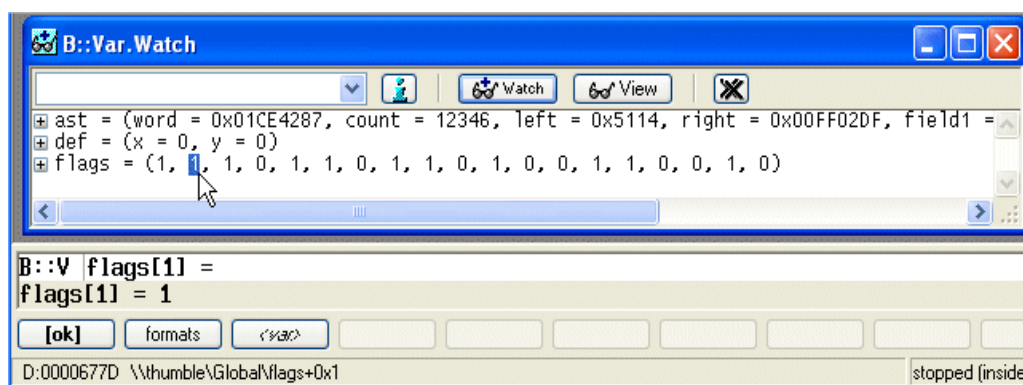


In most windows a context sensitive menu can be used with the right mouse button. If you select a variable you get access to the Var pulldown menu, that provides all features for displaying and modifying variables.

You want to inspect a variable and you are not sure about the spelling open the symbol browser to display all symbols stored in the internal symbol database.

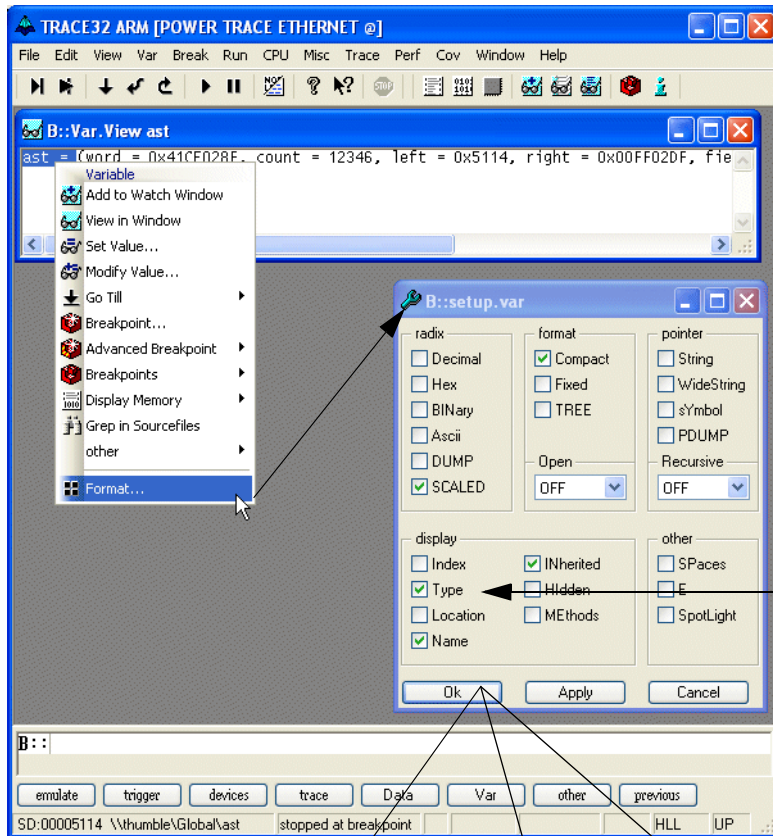


If you want to modify a variable value, double click to the value. The appropriate **Var.set** command will be displayed in the command line. Enter the new value and confirm with return.

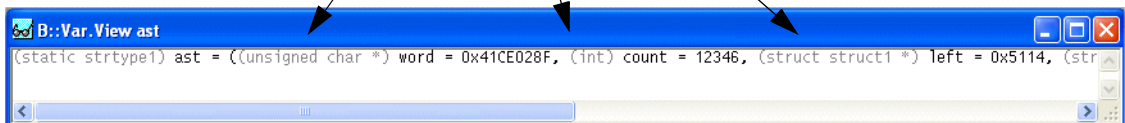


Format HLL-Variables

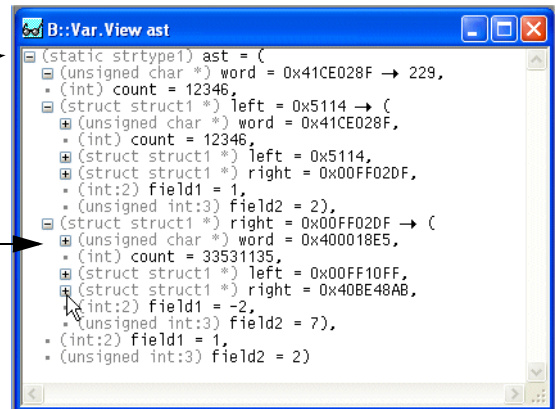
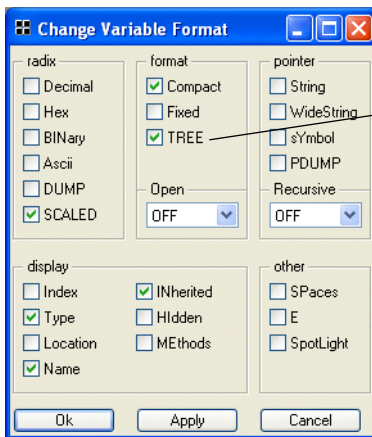
To adapt the display of a variable to your need, select the variable name, press the right mouse and select **Format...** from the **Var pulldown** menu.



Select **Type** to display the variable with the complete type information



If you display more complex HLL structures, select **TREE** in the **Format** field of the **Change Variable Format** dialog box. This formatting allows to select the display for each member of the structure by clicking on + or -.



Besides that, To exit from TRACE32-ICD use ***Exit*** in the ***File*** menu.