

TRACE 32

In-Circuit Debugger

Quick Installation and Tutorial



Copyright (C) 1989-1998 Lauterbach Datentechnik GmbH

All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system or translated into any language in any form by any means without the written permission of Lauterbach Datentechnik GmbH.

TRACE32 is a registered trademark of Lauterbach Datentechnik GmbH.

Lauterbach Datentechnik GmbH
Fichtenstrasse 27
D-85649 Hofolding
Germany

telephone: ++49 8104/8943-50
facsimile: ++49 8104/8943-49
e-mail: support@lauterbach.com

TRACE32: In-Circuit Debugger (ICD)
August 1998

Table of Contents

TRACE32-ICD Preface _____ Page PRE-1

What is an In-Circuit Debugger?	PRE-1
About this Guide	PRE-2
Where to Look for More Information	PRE-4

Quick Installation for TRACE32-ICD _____ Page INST-1

About the Installation Guide	INST-1
Product Overview	INST-2
System Concepts	INST-4
Host Interfaces	INST-4
The Debug Module	INST-11
Hardware Installation	INST-12
Installation with ISA Interface Card (PODPC)	INST-12
Installation with Parallel Interface Card (PODPAR)	INST-15
Installation with Ethernet Interface (PODETH)	INST-18
Software Installation	INST-20
PODPC or PODPAR on MS-WINDOWS (PC)	INST-21
PODETH on MS-WINDOWS (PC)	INST-26
PODETH on SunOS, Solaris (SUN)	INST-29
PODETH on HP-UX 10.X (HP-9000)	INST-31
Troubleshooting	INST-33

Tutorial for TRACE32-ICD _____ Page TUT-1

About the Tutorial	TUT-1
Start TRACE32-ICD	TUT-2
Setup the Application Properties	TUT-3
The Online Help	TUT-4
Setup the Debug Environment	TUT-7
Batch Jobs	TUT-11
The User Interface	TUT-13
How to Debug the Program	TUT-15
How to Display and Modify CPU Registers	TUT-25
How to Display and Modify the Special Function Registers	TUT-27
How to Display and Modify Memory	TUT-29
How to Set Breakpoints	TUT-31
Software Breakpoints	TUT-31
Breakpoints in ROM, Flash, EEPROM	TUT-35
Breakpoints on Data Accesses	TUT-36
Onchip Breakpoints (Overview)	TUT-38
Display and Modify HLL Variables	TUT-39
Format HLL-Variables	TUT-42
Exit TRACE32	TUT-44

TRACE32-ICD Preface

What is an In-Circuit Debugger?

Most CPUs provide an onchip debug system implemented in the CPU. Typical examples are the BDM interface from Motorola, the JTAG interface for the ARM7 or the JTAG interface for the PowerPC family. The debug interface usually requires a few CPU pins that are used for the communication between the onchip debug system and a third party development tool. The onchip debug system provides the following basic features:

- Read/write memory
- Read/write CPU register
- Single step and real time execution
- Hardware breakpoints and trigger features (not supported by all CPUs)

The In-Circuit Debugger TRACE32-ICD uses these basic features of the onchip debug system to provide a powerful debug tool that offers:

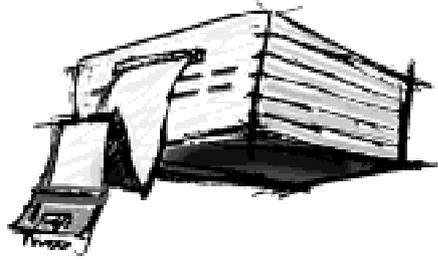
- Easy high-level and assembler debugging
- Display of internal and external peripherals on a logical level
- Onchip break and trigger support
- RTOS awareness
- Flash programming
- Powerful script language
- Multiprocessor debugging

TRACE32-ICD is part of the TRACE32 microprocessor development tool product line. This product line also includes a trace extension for some ICD-Debuggers and the full featured In-Circuit Emulators TRACE32-ICE for all CISCs and TRACE32-FIRE for the fast RISC processors.

In-Circuit Debugger
TRACE32-ICD



In-Circuit Emulator
TRACE32-ICE/FIRE



All TRACE32 development tools use a common user interface. So the look and feel stays the same no matter what system you are using.

About this Guide

The purpose of this guide is:

- To get your In-Circuit Debugger installed.
(See [Quick Installation for TRACE32-ICD](#)).
- To help you to write the proper start up sequence.
(See *Tutorial for TRACE32-ICD*).
- To make you familiar with the basic functions of TRACE32-ICD.
(See *Tutorial for TRACE32-ICD*).

Since an In-Circuit Debugger always needs a working target system, we also offer Simulators for some major CPU families, that allow you to get familiar with our debugging tools even before your target hardware is available.

Simulator:

A simulator can run without a target processor. The work of the CPU is simulated by software. So real time debugging is not possible.

You can download our simulators from our web page <http://www.lauterbach.com> or from our CD **TRACE32 Product Information** by using the link **Download Demo Software** in the main menu.

Where to Look for More Information

The common TRACE32 user interface contains a detailed Online-Help, that offers the most current description of all debug features.

If you have installed the TRACE32 software [The Online Help](#) on how to start the Online Help.

The Online Help can also be called without installing the TRACE32 software:

- On WINDOWS systems simply call the help file `\DOS\MAN.HLP` on the CD.
- Or use the PDF file `TRACE32-MANUAL.PDF` in the PDF directory of the CD to open the Online help in PDF format.

The following books contain the most relevant information for the use of TRACE32-ICD:

ICD Targets



Since the **Quick Installation for TRACE32-ICD** and the **Tutorial for TRACE32-ICD** are generic for all supported CPUs, we strongly recommend to read the ICD Target Guide for your CPU. This guide describes all CPU specific settings and features.

ICD Debugger User's Guide

This book provides information about the basic concepts for the In-Circuit Debuggers and the other PODBUS devices (e.g. ICD Trace, ICD Risc Trace, EPROM Simulator).

Reference

In the Reference book you can find an alphabetical list of all debug commands for the TRACE32 development tools and a detailed description of their syntax and function. All commands not available for TRACE32-ICD are marked with (E – TRACE32-ICE only) or (F – TRACE32-FIRE only).

The following books contain information for advanced users.

Installation Guide	General Installation Guide for all TRACE32 development tools.
Operation System User´s Guide	Gives an overview of the features for the TRACE32 Operating System e.g. windows management, file management, printer etc.
Operation System Reference	Provides a detailed description of the TRACE32 Operating System e.g. windows management, file management, printer etc.
Practice User´s Guide	Introduction into the batch language PRACTICE.
Practice Reference	Alphabetical list of all PRACTICE commands.

Quick Installation for TRACE32-ICD

About the Installation Guide

This Installation Guide describes the basic installation and configuration for all TRACE32-ICD In-Circuit Debuggers that are implemented using their onchip debug interface. Typical examples for onchip debug interfaces are BDM, JTAG, ONCE ...

For the installation and configuration of the ROM monitors or for special system configurations (e.g. additional devices, multiprocessor debugging etc.) refer to the [Installation Guide](#).

Product Overview

A complete TRACE32-ICD configuration consists of:

- **A PODBUS interface to the host (PODPC, PODPAR or PODETH)**

- **A Debug Module**

Lauterbach offers two different debug modules: the Standard Debug Module and the Power Debug Module. Both modules can be used all CPUs that are supported by TRACE32-ICD.

Some CPUs like the PPC603 or the MPC740/750 need the Power Debug Module, since their complex debug interface requires the transfer of huge amounts of data. To guarantee a fast data transfer rate the Power Debug Module has its own RISC controller.

- **A Debug Cable**

The Debug Cable connects the Debug Module to the debug interface on your target. For the connector pinout of your debug interface refer to the ***ICD Target Guide*** for your CPU.

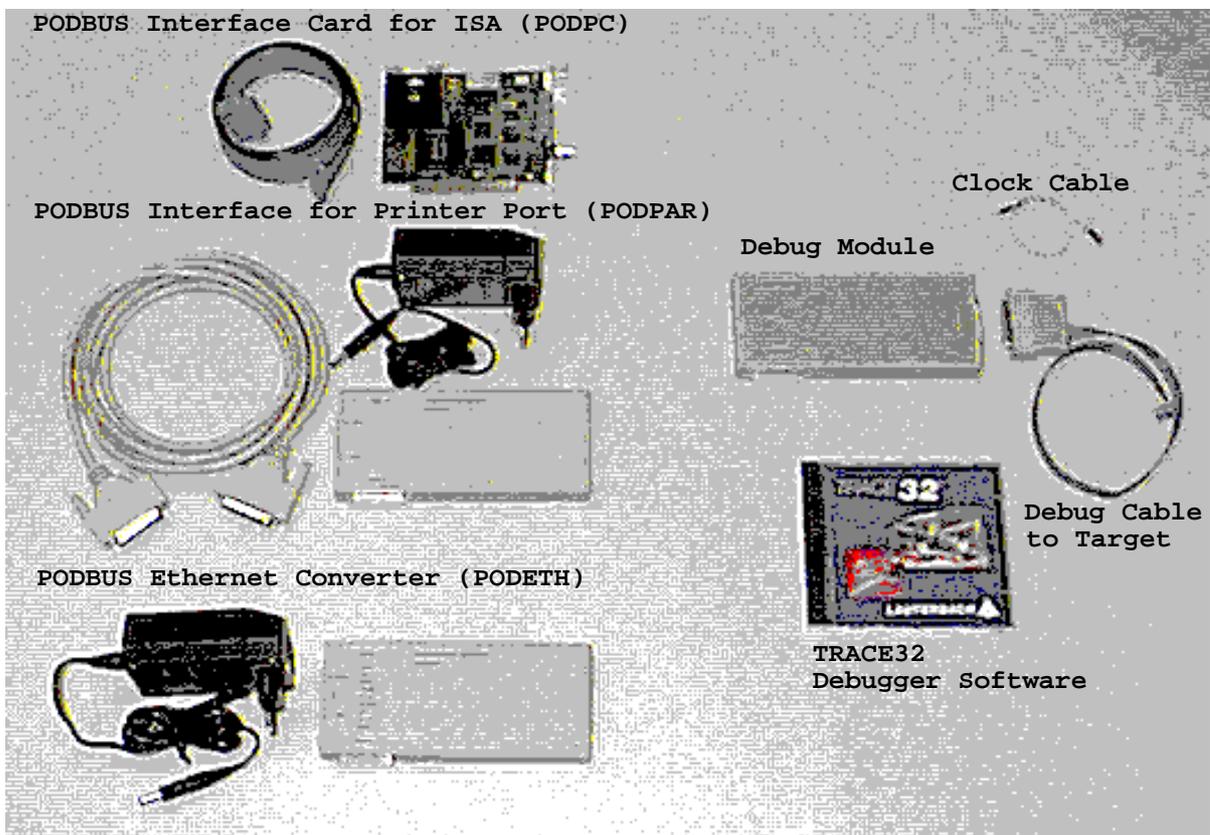
- **A Clock Cable**

By default TRACE32-ICD uses a fixed clock to run the debug interface. A clock range from 100KHz up to 5 MHz can be used.

We also provide a clock cable to allow you the use of the divided CPU clock as clock for the debug interface. The relation between the CPU clock and the debug interface clock is specific for your CPU. Refer to your **ICD Target Guide** for detailed information. The use of the divided CPU clock has the following advantages:

- The max. speed for the debug interface can be used. However we recommend 10MHz as the max. speed.
- The clock for the debug interface is automatically adapted if the CPU clock is changed by your application program.

- **The TRACE32 debugger software**



System Concepts

The TRACE32 Debugger Software runs on a host (PC or workstation). The host communicates with the target via the Debug Module.

- For the fast communication between the host and the debug module a special bus interface is used: PODBUS (Processor Oriented Device Bus).

An advantage of the PODBUS is that your debug environment can be easily expanded by additional PODBUS devices, to provide a powerful tool configuration. Additional PODBUS devices are ICD Trace, ICD RISC Trace, EPROM Simulator, Stimuli Generator etc.

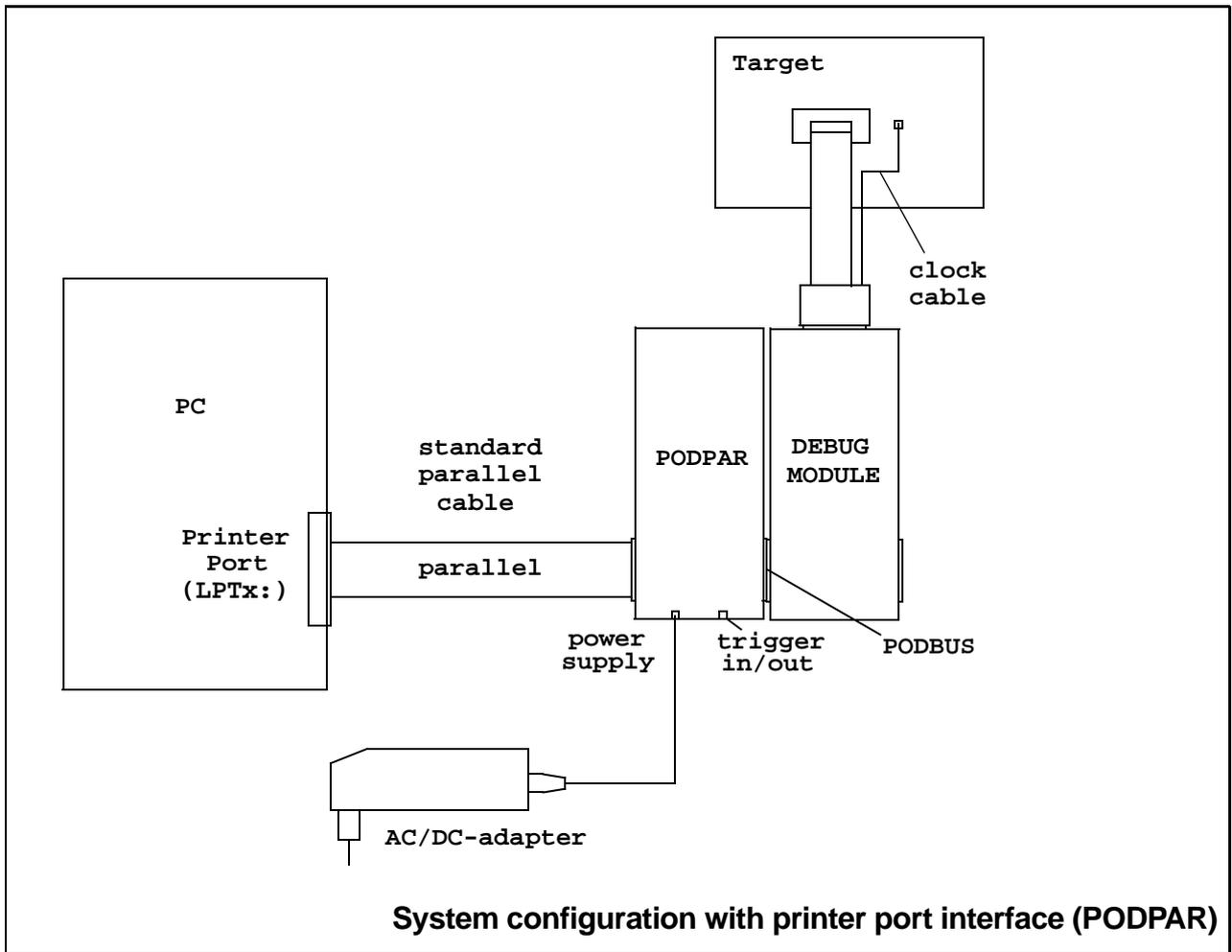
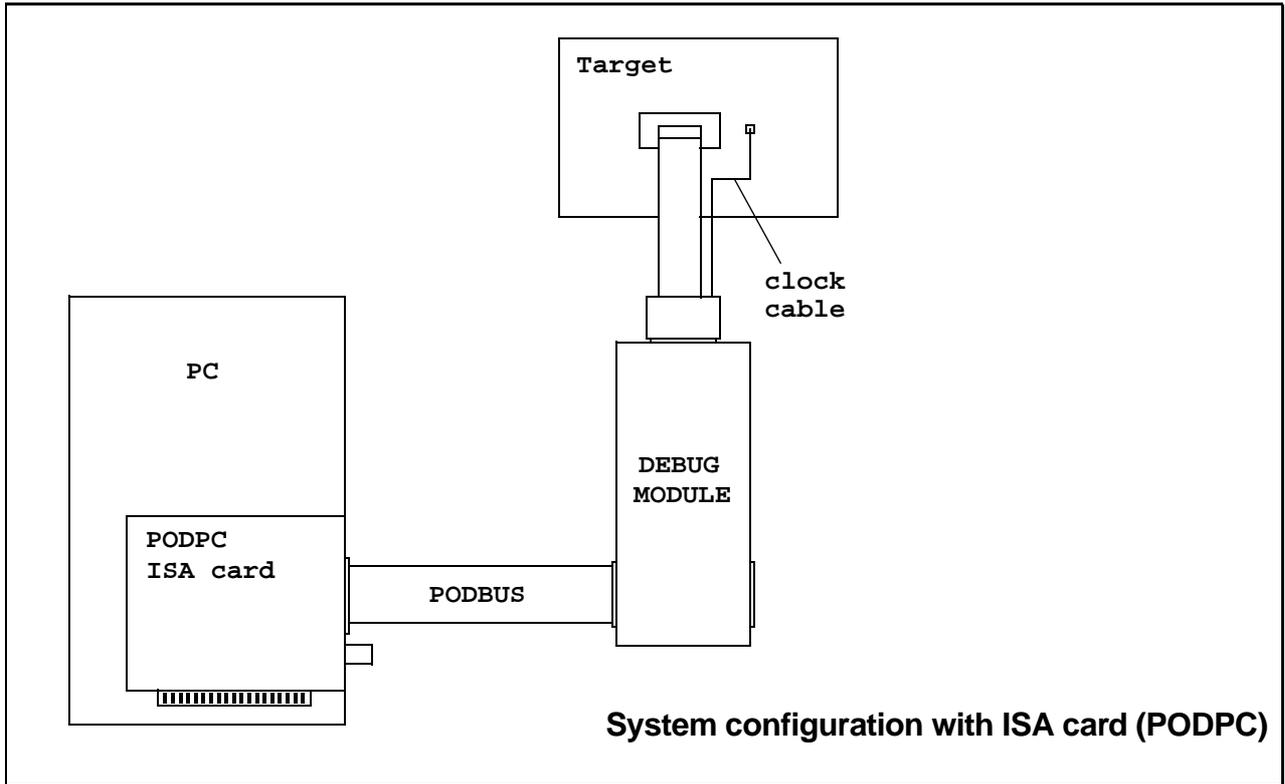
- The connection between the target and the debug module is done by a processor specific debug interface cable.

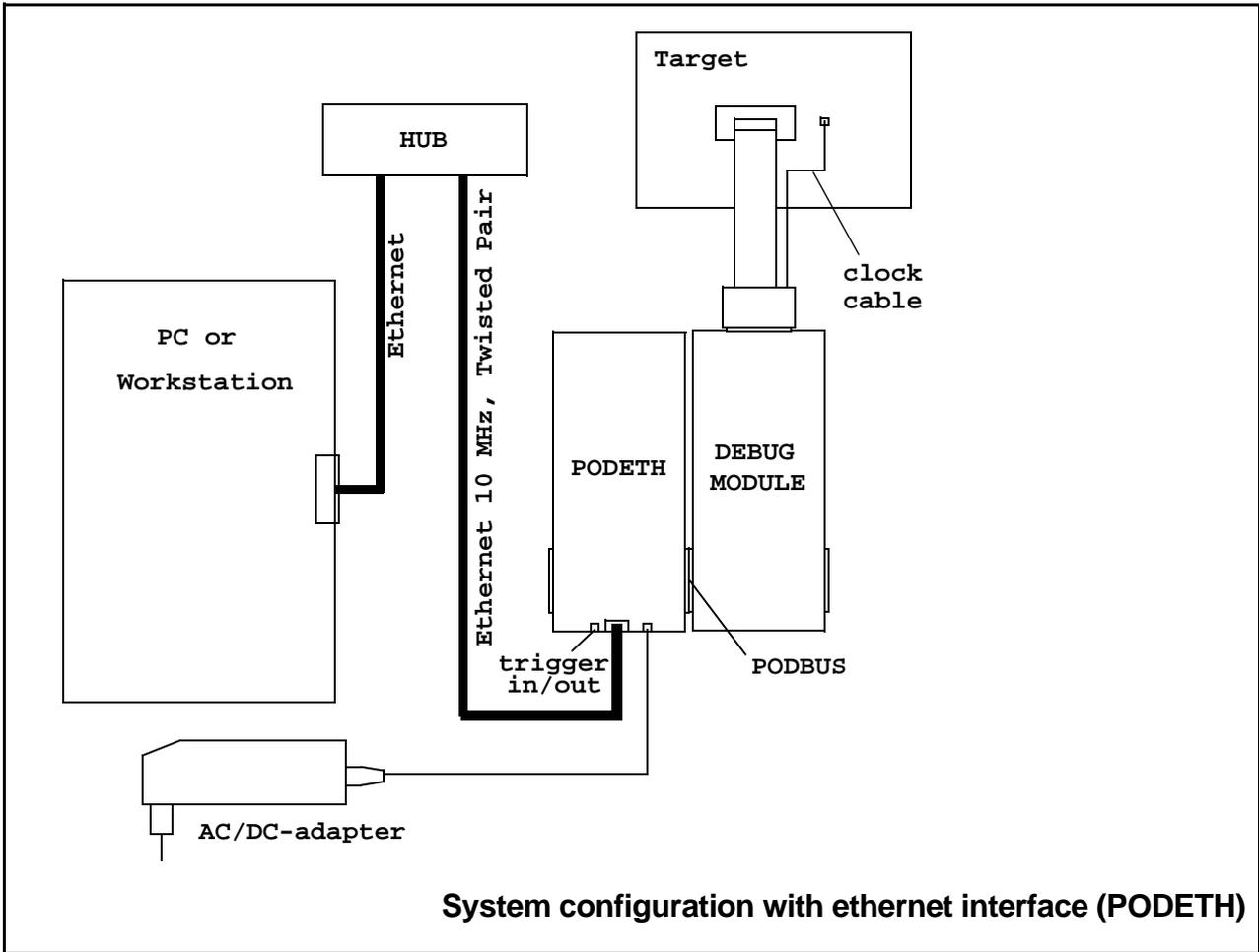


Host Interfaces

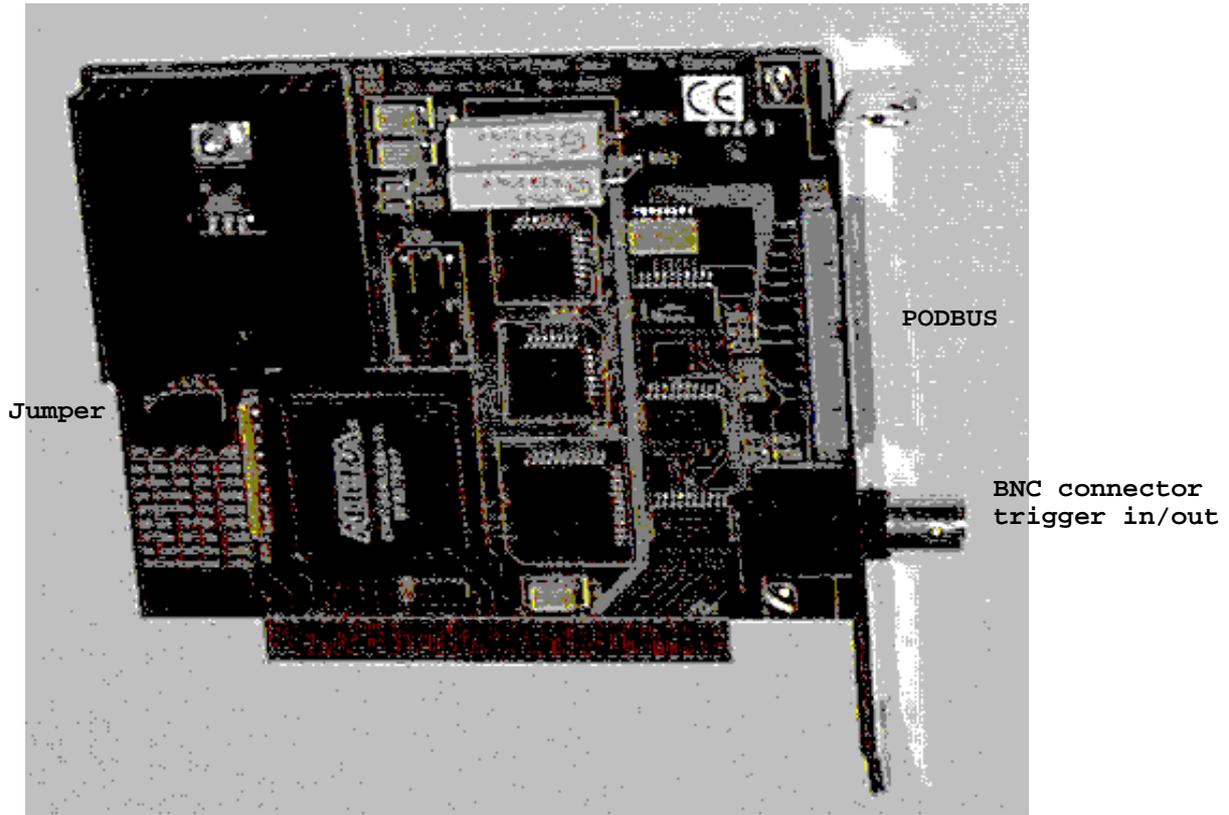
Three different interfaces are available to connect the host system to the PODBUS:

- **PODPC interface card** for ISA bus
- **PODPAR converter** for printer port (LPTx:)
- **PODETH converter** to Ethernet (Twisted Pair)





PODPC Interface Card for ISA bus



When a PODPC interface card is used no additional power supply is required for additional PODBUS devices.

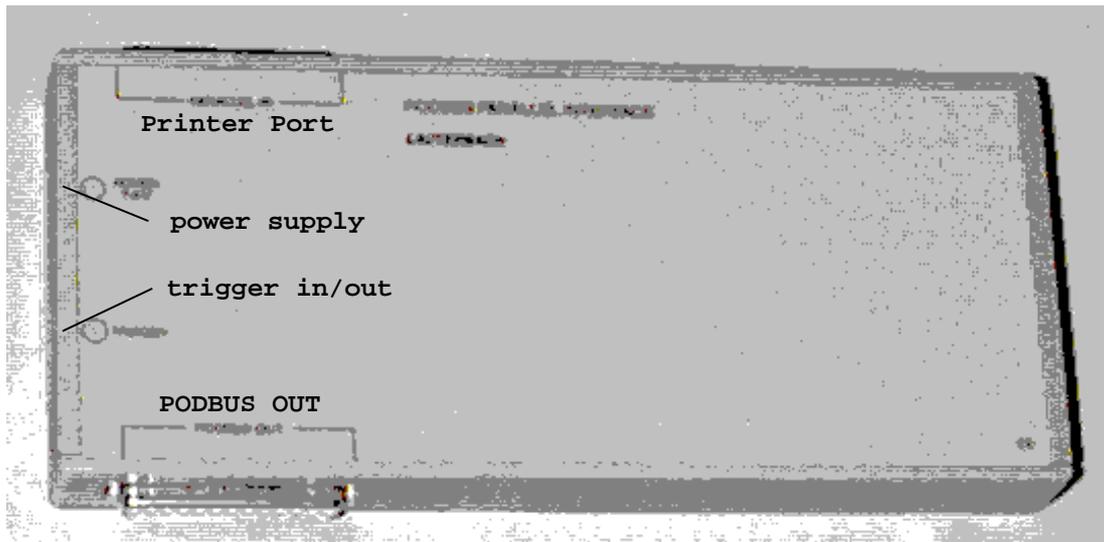
Address select jumpers on the ISA card:

JP0	JP1	JP2	Address (hex)	Address (dec)
on	on	on	350	848
off	on	on	250	592
on	off	on	260	608
off	off	on	280	640
on	on	off	300	768
off	on	off	330	816
on	off	off	340	832
off	off	off	390	912

JP3 must always be on.

The BNC connector is used to input/output a trigger signal to/from the Debug Module. For more information on this Trigger feature refer to [Trigger](#) in the *ICD Users Guide*.

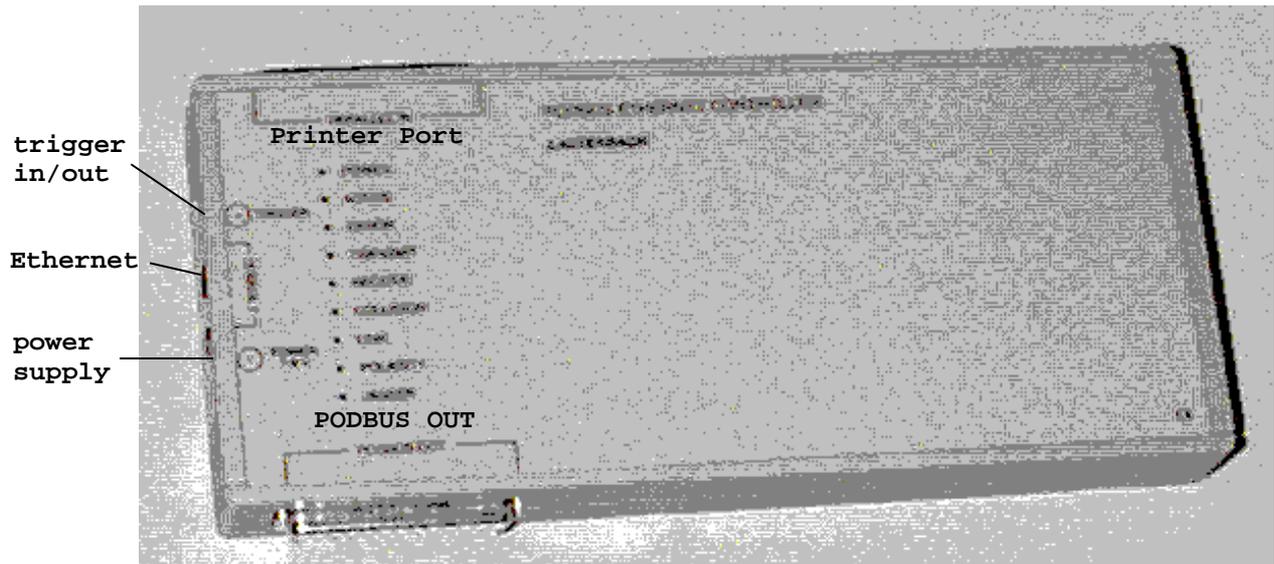
PODPAR Converter for Printer Port (LPTx:)



External power supply is required for the PODPAR (Plug: 7-9V; inner negative, outer positive). Please use only the enclosed AC/DC-Adapter.

The BNC connector is used to input/output a trigger signal to/from the Debug Module. For more information on this Trigger feature refer to [Trigger](#) in the *ICD Users Guide*.

PODETH Converter to Ethernet (Twisted Pair)



The PODBUS controller for ethernet contains both, an ethernet connector for twisted pair and a printer port interface, to provide a flexible host connection.

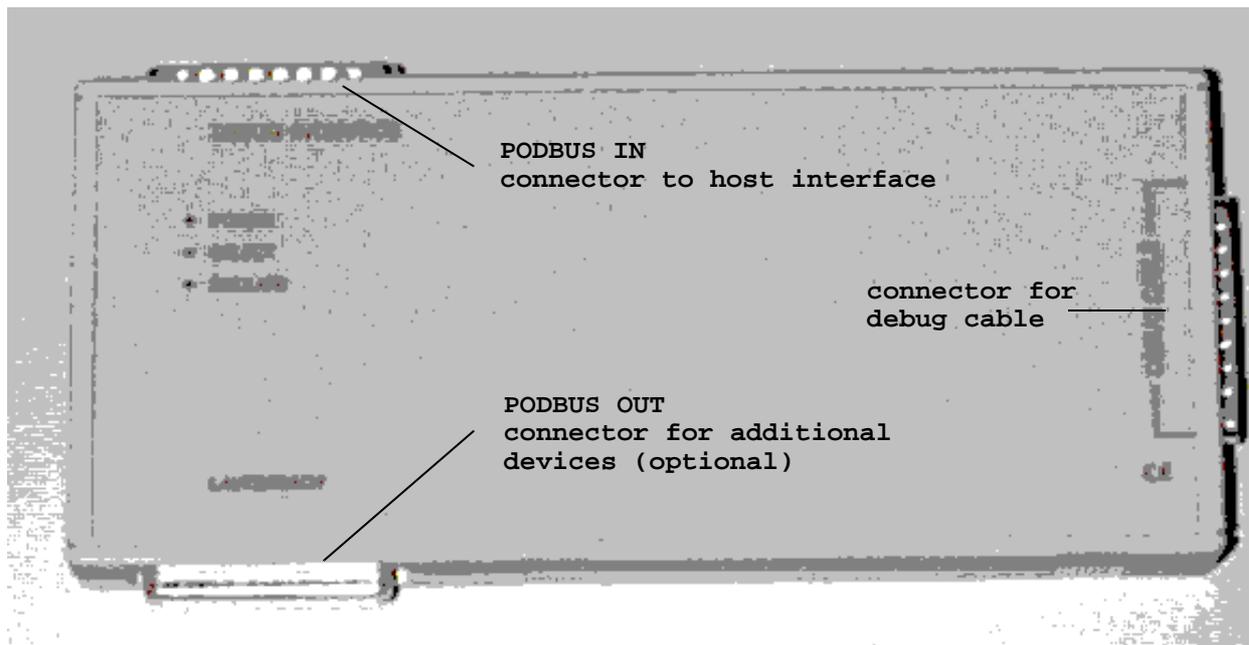
External power supply is required for the PODETH (Plug: 7-9V; inner negative, outer positive). Please use only the enclosed AC/DC-Adapter.

The BNC connector is used to input/output a trigger signal to/from the Debug Module. For more information on this Trigger feature refer to [Trigger](#) in the *ICD Users Guide*.

The LEDs transmit, receive, collision, link, polarity, jabber indicate the status of the integrated Ethernet transceiver MC68160.

LEDs	
POWER	External power is supplied.
ACTIVE	On if device is active, flashes if device is not used.
ERROR	On if device is active, flashes if device is not used. In error case an error code will be pulsed. See the <i>ICD Debugger User Guide</i> for further information.
TRANSMIT	Transmit activity.
RECEIVE	Receive activity.
COLLISION	Collision activity.
LINK	Twisted pair link integrity.
POLARITY	Twisted Pair Polarity Error (receiver inputs TPRX+, TPRX- are reversed).
JABBER	Twisted Pair Jabber condition detected.

The Debug Module



Some processor types like MPC740/750, PPC603 ... require a Power Debug Module. But there is no difference in using this module.

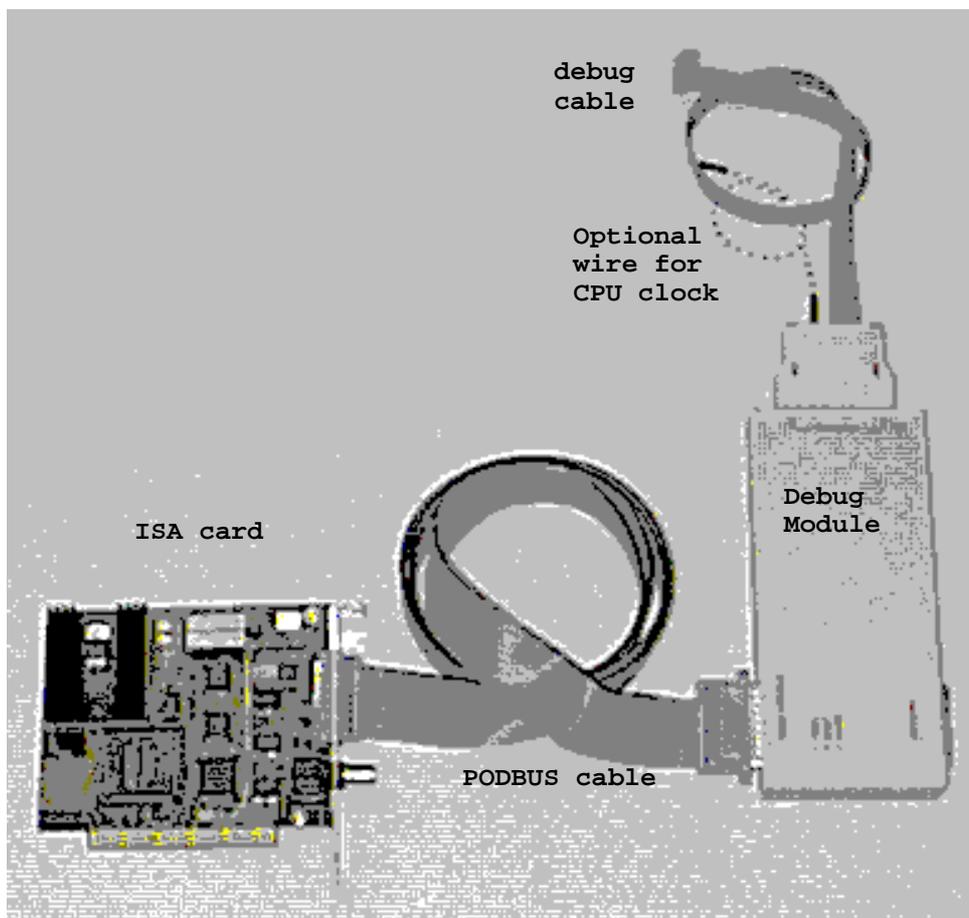
LEDs	
POWER	Power is applied to the debug module. Power will be supplied via the PODBUS. Therefore power is on when the host interface is on.
SELECT	(Optionally) It is possible to connect more than one device to the PODBUS. Each device can be independently controlled by the debugger software. The LED is on when this device is currently controlled.
EMULATE	The user program is running on the target.

Hardware Installation

Choose one of the following installations depending on your host interface:

- **ISA Interface Card** (PODPC)
- **Parallel Interface Card** (PODPAR)
- **Ethernet Interface** (PODETH)

Installation with ISA Interface Card (PODPC)



Do not connect or disconnect anything while power is on!

To configure your TRACE32-ICD with the PODPC interface card:

1. Install the ISA card (PODPC) on your PC.

Insert the ISA card (PODPC) in a free ISA slot of your PC. The default I/O address of the ISA interface card is set to 350h. If this address is already in use it can be changed by setting the jumpers on the card. Address select jumpers on the ISA card:

JP0	JP1	JP2	Address (hex)	Address (dec)
on	on	on	350	848
off	on	on	250	592
on	off	on	260	608
off	off	on	280	640
on	on	off	300	768
off	on	off	330	816
on	off	off	340	832
off	off	off	390	912

JP3 must always be on.

If you change the jumper setting a modification in the TRACE32 configuration file `config.t32` is also required.

Example: If the jumpers settings are JP0=OFF, JP1=ON, JP2=OFF, JP3=ON the file `config.t32` must have the following contents:

```
PBI=  
ADDRESS=816 (the address must be entered in decimal format)
```

For more information on the configuration file see [PODPC or PODPAR on MS-WINDOWS \(PC\)](#).

The PODPC requires no additional power supply.

2. **Connect the Debug Module with PODBus cable to the ISA card.**
3. **Connect the Debug Module to the target using the debug cable.**

A list of the supported processors and the pinout of the dedicated debug interface connector (BDM, JTAG, OCDS, COP, ONCE, ...) can be found in the *ICD Target Guide* for your CPU family.

4. Connect the clock cable (optional).

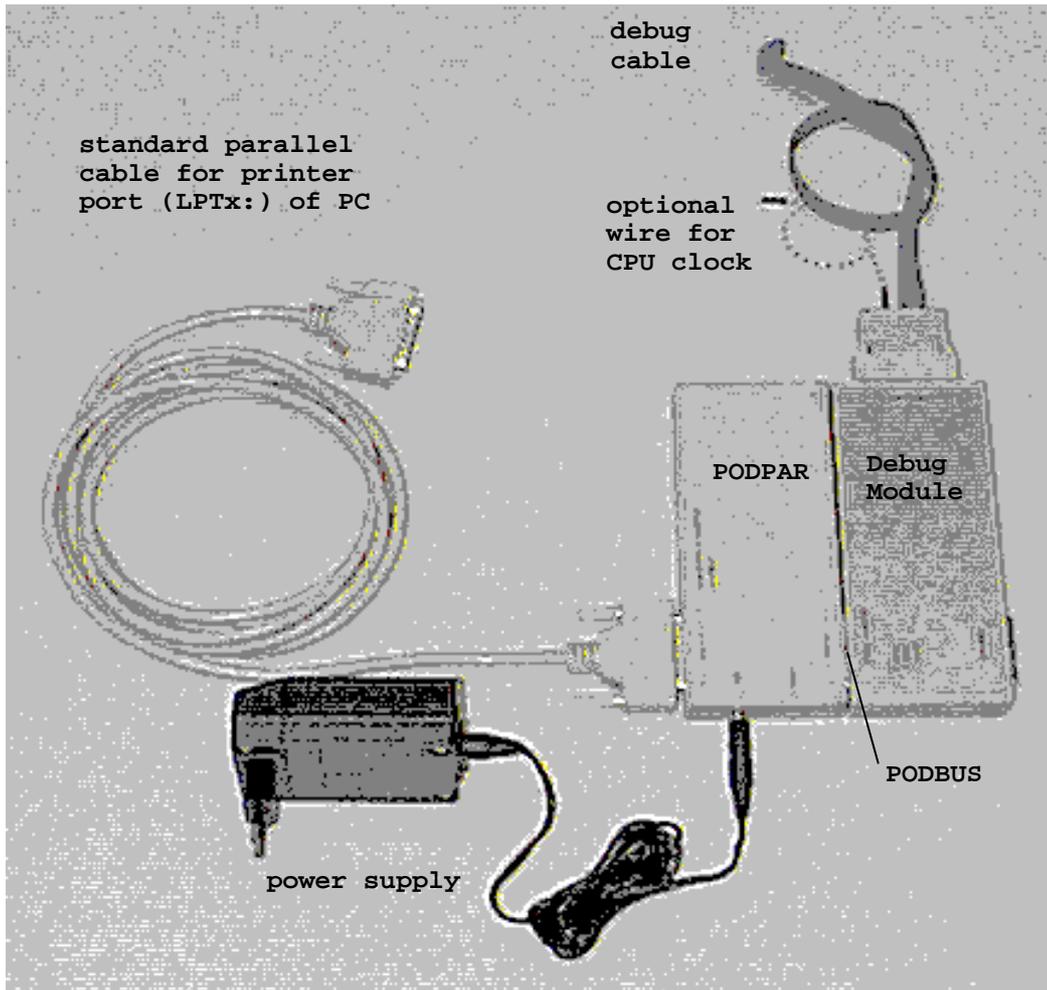
By default TRACE32-ICD uses a fixed clock to run the debug interface. A clock range from 100KHz up to 5 MHz can be used.

We also provide a clock cable to allow you the use of the divided CPU clock as clock for the debug interface. The relation between the CPU clock and the debug interface clock is specific for your CPU. Refer to your **ICD Target Guide** for detailed information. The use of the divided CPU clock has the following advantages:

- The max. speed for the debug interface can be used. However we recommend 10MHz as the max. speed.
- The clock for the debug interface is automatically adapted if the CPU clock is changed by your application program.

The setting for the debug clock are done during the [Setup of the Debug Environment](#).

Installation with Parallel Interface Card (PODPAR)



Do not connect or disconnect anything while power is on!

To configure your TRACE32-ICD with the PODPAR interface card:

1. **Connect the Printer Port Interface PODPAR with enclosed cable to the printer port (LPTx:) of the PC.**

Normally no changes on your PC are needed. The printer port interface supports standard and EPP parallel ports (improved speed). At common PC's the parallel port mode can be selected in the BIOS. The BIOS must be switched to EPP, EPP V1.7 or EPP V1.9. Some PC's require special tools to modify the parallel port mode. Please ask your PC manufacturer.

During the software installation you will be asked which printer port you want to use and if EPP mode is supported. The entry in the `config.t32` file will be generated automatically.

For standard configuration the file `config.t32` must have the following contents:

```
PBI=  
LPT1                (or optional LPT2)
```

If the PC supports EPP mode the following `config.t32` entries are required:

```
PBI=  
LPT1                (or optional LPT2)  
EPP
```

For more information on the configuration file see [PODPC or PODPAR on MS-WINDOWS \(PC\)](#).

2. **Connect the power supply to the PODBUS printer port interface.**

(Plug: 7-9V; inner negative, outer positive). Please use only the enclosed AC/DC-adapter.
3. **Connect the Debug Module via the PODBUS connector to the PODPAR box.**
4. **Connect the Debug Module to the target by the debug cable.**

A list of the supported processors and the pinout of the dedicated debug interface connector (BDM, JTAG, OCDS, COP, ONCE, ...) can be found in the *ICD Target Guide* for your CPU family.

5. Connect the clock cable (optional).

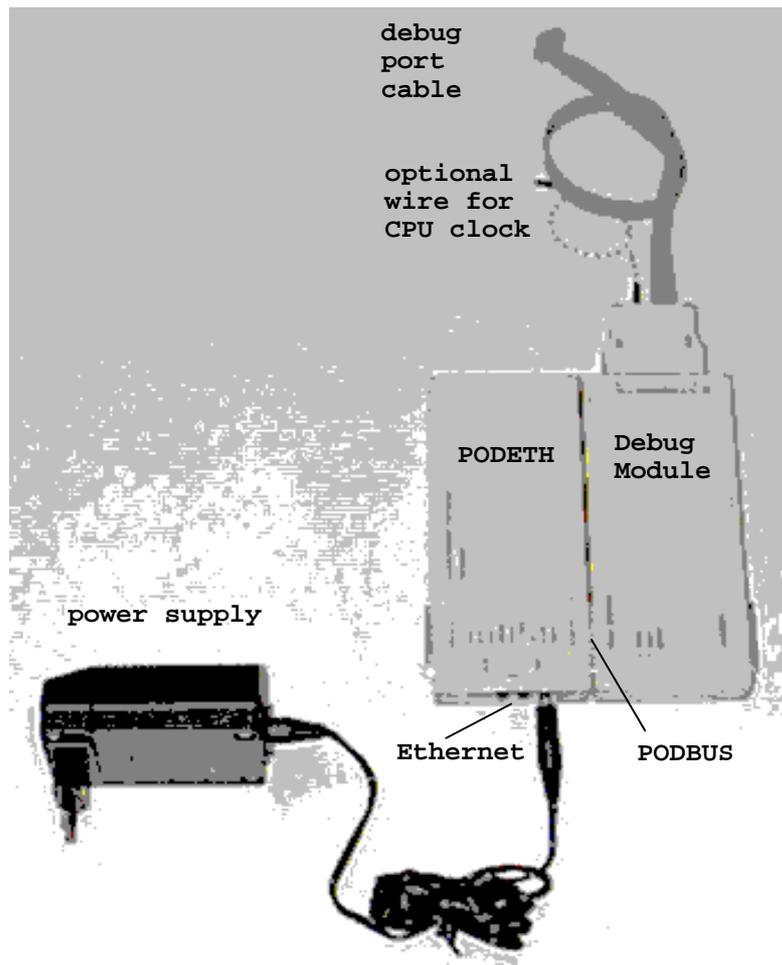
By default TRACE32-ICD uses a fixed clock to run the debug interface. A clock range from 100KHz up to 5 MHz can be used.

We also provide a clock cable to allow you the use of the divided CPU clock as clock for the debug interface. The relation between the CPU clock and the debug interface clock is specific for your CPU. Refer to your *ICD Target Guide* for detailed information. The use of the divided CPU clock has the following advantages:

- The max. speed for the debug interface can be used. However we recommend 10MHz as the max. speed.
- The clock for the debug interface is automatically adapted if the CPU clock is changed by your application program.

The setting for the debug clock are done during the [Setup of the Debug Environment](#).

Installation with Ethernet Interface (PODETH)



PODETH provides an Ethernet interface (twisted pair, 10 Base T) for the Debug Module.



Do not connect or disconnect anything while power is on!

To configure your TRACE32-ICD with the PODETH interface card:

1. **Connect the Ethernet Interface to your Ethernet (cable not enclosed).**
2. **Connect the power supply to the PODBUS Ethernet Interface.**

(Plug: 7-9V; inner negative, outer positive). Please use only the enclosed AC/DC-adaptor.

3. **Connect the Debug Module via the PODBUS connector to the PODETH box.**
4. **Connect the Debug Module to the target by the debug cable.**

A list of the supported processors and the pinout of the dedicated debug interface connector (BDM, JTAG, OCDS, COP, ONCE, ...) can be found in the **ICD Targets** for your CPU family.

5. **Connect the clock cable (optional).**

By default TRACE32-ICD uses a fixed clock to run the debug interface. A clock range from 100KHz up to 5 MHz can be used.

We also provide a clock cable to allow you the use of the divided CPU clock as clock for the debug interface. The relation between the CPU clock and the debug interface clock is specific for your CPU. Refer to your **ICD Target Guide** for detailed information. The use of the divided CPU clock has the following advantages:

- The max. speed for the debug interface can be used. However we recommend 10MHz as the max. speed.
- The clock for the debug interface is automatically adapted if the CPU clock is changed by your application program.

The setting for the debug clock are done during the [Setup of the Debug Environment](#).

Software Installation

In this section the installation for the TRACE32-ICD software is described.

The installation depends on the operating system, the host and the host interface. Therefore select the corresponding chapters according to your needs:

- [PODPC or PODPAR on MS-WINDOWS](#) (PC)
- [PODETH on MS-WINDOWS](#) (PC)
- [PODETH on SunOS, Solaris](#) (SUN)
- [PODETH on HP-UX 10.X](#) (HP-9000)

Installation on Windows 3.x:

1. Start `\bin\win32s\disk1\setup.exe` on the CD to install WIN32s.
2. Then start `install.bat` in the root directory of the CD-ROM and follow the instructions of the installation program.

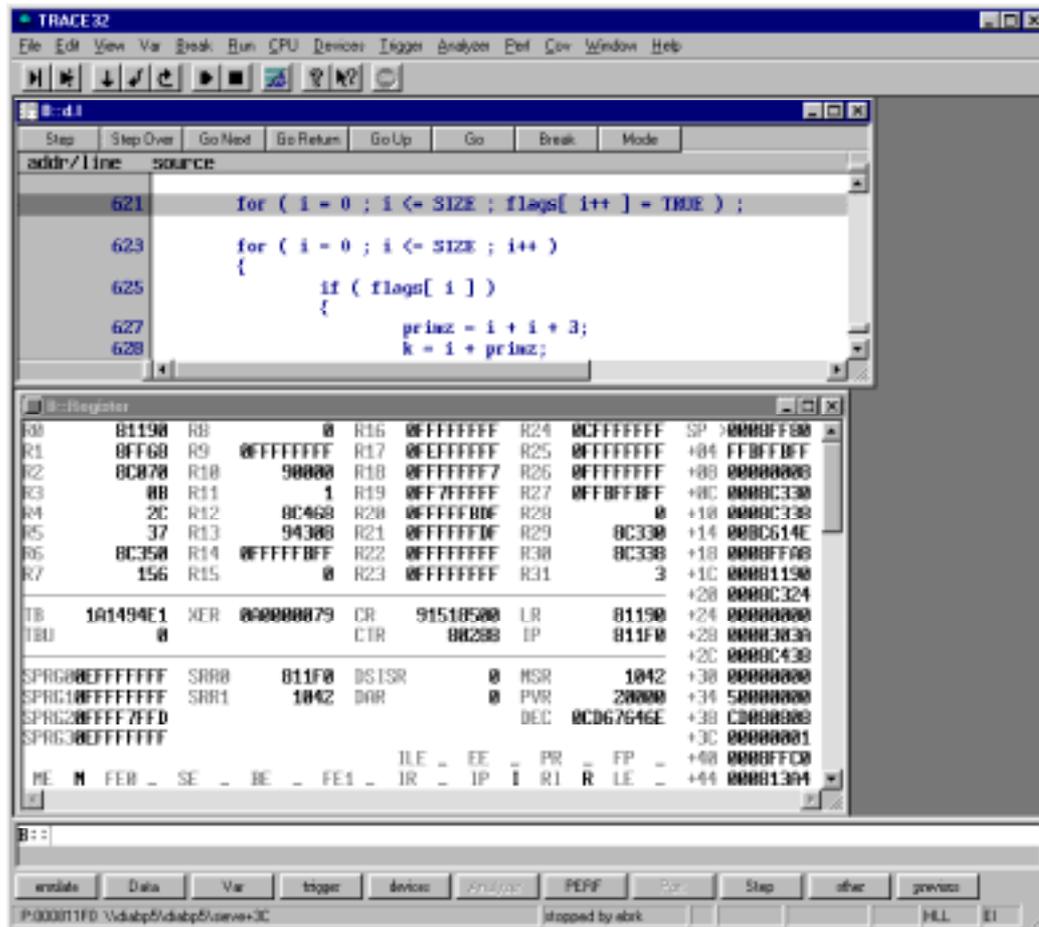
Installation on Windows 3.x (WIN32s), Windows 95, Windows 98, Windows NT:

1. Start `setup.bat` in the root directory of the CD, if it does not start automatically. Follow the instructions of the installation program.

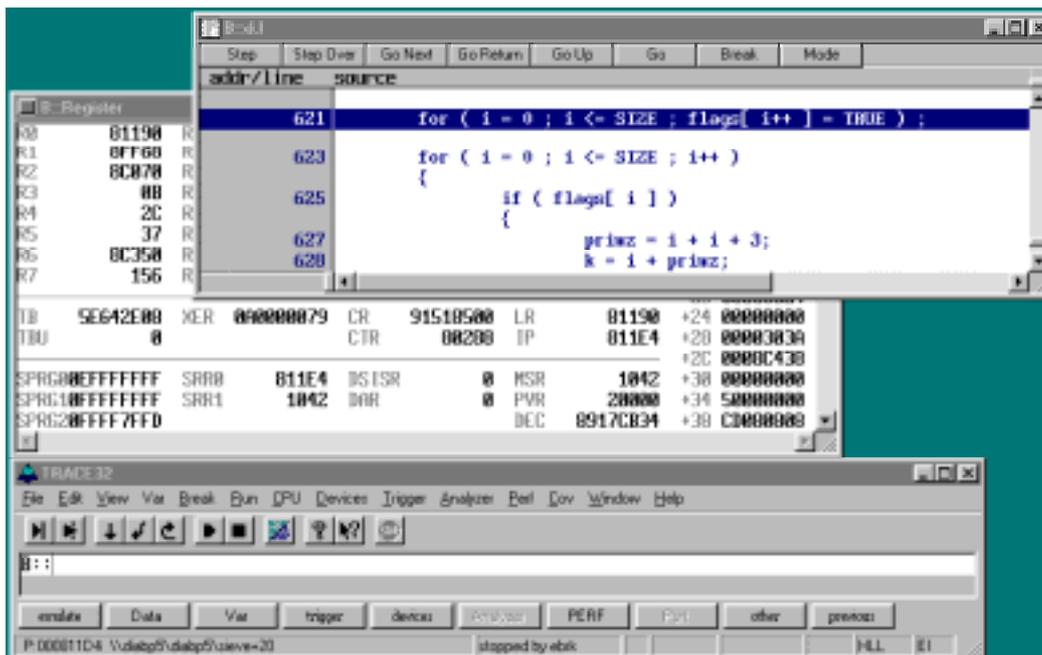
Some hints to answer the questions of the installation program:

- Select product type **ICD/FIRE BDM/JTAG/ROM-Monitor with PC**.
- In the ICE/BDM-ROM list select **ICD <CPU>** e.g. ICD 68K.
- Select **ICD/FIRE BDM/JTAG/ROM-Monitor with PODPC** if you are using the ISA card (PODPC) for your host connection.
- Select **ICD/FIRE BDM/JTAG/ROM-Monitor with PODPAR** if you are using the printer port for your host connection. Then select EPP mode if supported by your PC, standard mode otherwise. At last select the used printer port LPTx:.

- Select Multiple document interface (MDI) or Multiple window interface (MWI).



When the TRACE32 debugger software is started, an application specific window is displayed. All windows which display debugger information are placed within this application window (MDI).



When the TRACE32 debugger software is started, a command window is displayed, that contains the main menu bar, the main tool bar and the command line. All windows which display debugger information are placed freely on the screen (MWI).

- Select “remote control access not active”

The installation program generates a **TRACE32** folder where you can start the debugger software (and the Online Help) from.

If you prefer to start the program by clicking on an icon on the desktop, use the Explorer, search for the startfile **T32M<CPU>.EXE** (e.g t32m68.exe for the CPU32 from Motorola) and pull it onto your desktop.

The Configuration File for PODPC on MS-WINDOWS

The configuration file is used to make some basic settings for the cooperation between TRACE32-ICD and the host system. The entries are automatically made by the installation program.

By default the software assumes that the configuration file is in the system directory and is named `config.t32`. Refer to [Setup the Application Properties](#) in *Tutorial for TRACE32-ICD* if you want to change the name or location of the configuration file.

Standard settings in the configuration file:

```
;Environment variables
OS=
ID=T32
TMP=C:\temp           ;temp directory for TRACE32
SYS=d:\t32\isa       ;system directory for TRACE32

; Printer settings
PRINTER=WINDOWS      ;all standard windows printer can be
                    ;used from the TRACE32 user interface
```

Settings in the configuration file if the jumper settings are changed during the hardware installation:

```
PBI=
ADDRESS=816           ;the address must be entered in
                    ;decimal format

;Environment variables
OS=
ID=T32
TMP=C:\temp           ;temp directory for TRACE32
SYS=d:\t32\isa       ;system directory for TRACE32

; Printer settings
PRINTER=WINDOWS      ;all standard windows printer can be
                    ;used from the TRACE32 user interface
```

For more advanced setting refer to the [Installation Guide](#).

The Configuration File for PODPAR on MS-WINDOWS

The configuration file is used to make some basic settings for the cooperation between TRACE32-ICD and the host system. The entries are automatically made by the installation program.

By default the software assumes that the configuration file is in the system directory and is named `config.t32`. Refer to [Setup the Application Properties](#) in *Tutorial for TRACE32-ICD* if you want to change the name or location of the configuration file.

Standard settings in the configuration file:

```
PBI=                ;or optional LPT2
LPT1

; Environment variables
OS=
ID=T32
TMP=C:\temp        ;temp directory for TRACE32
SYS=d:\t32\isa     ;system directory for TRACE32

; Printer settings
PRINTER=WINDOWS    ;all standard windows printer can be
                   ;used from the TRACE32 user interface
```

Settings in the configuration file if EPP mode should be used:

```
PBI=
LPT1                ;or optional LPT2
EPP

;Environment variables
OS=
ID=T32
TMP=C:\temp        ;temp directory for TRACE32
SYS=d:\t32\isa     ;system directory for TRACE32

; Printer settings
PRINTER=WINDOWS    ;all standard windows printer can be
                   ;used from the TRACE32 user interface
```

For more advanced setting refer to the [Installation Guide](#).

Installation of the TRACE32 Debugger Software

Installation on Windows 3.x:

1. Start `\bin\win32s\disk1\setup.exe` on the CD to install WIN32s.
2. Then start `install.bat` in the root directory of the CD-ROM and follow the instructions of the installation program.

Installation on Windows 3.x (WIN32s), Windows 95, Windows 98, Windows NT:

1. Start `setup.bat` in the root directory of the CD, if it does not start automatically. Follow the instructions of the installation program.

Some hints to answer the questions of the installation program:

- Select product type **ICD/FIRE BDM/JTAG/ROM-Monitor with Ethernet**
- In the ICE/BDM-ROM list select **ICD <CPU>** e.g. ICD 68K
- Select **Multiple document interface (MDI)**
- Select **Remote control access not active**

The installation program generates a **TRACE32** folder where you can start the debugger software (and the Online Help) from.

If you prefer to start the program by clicking on an icon on the desktop, use the Explorer, search for the startfile `t32w95.exe` and pull it onto your desktop.

The Configuration File for PODETH on MS-WINDOWS

The configuration file is used to make some basic settings for the cooperation between TRACE32-ICD and the host system. The entries are automatically made by the installation program.

By default the software assumes that the configuration file is in the system directory and is named `config.t32`. Refer to [Setup the Application Properties](#) in *Tutorial for TRACE32-ICD* if you want to change the name or location of the configuration file.

Standard settings in the configuration file:

```
LINK=NET
NODE=t32
PACKLEN=1024

; Environment variables
OS=
ID=T32
TMP=C:\temp           ;temp directory for TRACE32
SYS=d:\t32\isa       ;system directory for TRACE32

; Printer settings
PRINTER=WINDOWS      ;all standard windows printer can be
                    ;used from the TRACE32 user interface
```

For more advanced setting refer to the [Installation Guide](#).

Preparations for the Ethernet Interface

The Ethernet connection requires the driver program `t32w95.exe` with networking capabilities. This program requires that a WINSOCK compatible TCP/IP provider is installed. First a new node must be created for TRACE32. The Ethernet address of the system is placed on the bottom side of the system. The following line must be added to the file HOSTS:

```
192.9.200.5          t32
```

Note, that the INTERNET address given here is an example only. Contact your network administrator for a new INTERNET address for TRACE32-ICD. When a RARP server is used, the Ethernet address of the system must be entered in the file ETHERS:

```
0:c0:8a:0:0:0      t32
```

The INTERNET address is requested by a RARP protocol by TRACE32-ICD. If no RARP server is running, the address for the first connect must be set in the host table. After the first successful connect the INTERNET address is stored in nonvolatile memory within TRACE32-ICD. The following command sets the host translation table:

```
arp -s t32 0-c0-8a-0-0-0
```

NOTE: Windows95 has a bug, that may cause the arp command to fail, when the arp cache is empty. In this case 'ping' another host before executing the arp command, this will fill the arp cache.

Installation of the TRACE 32 Debugger Software

In the following example the directory `/home/t32` is used as the system directory.

The system directory is created with the following command:

```
mkdir /home/t32      (or similar)
```

The files are extracted from the CD to the system directory with the following commands:

```
mount -F hsfs -o ro /dev/dsk/c0t6d0s2 /cdrom/trace32 (or similar )
cd /home/t32
mkdir suns
cp -r /cdrom/trace32/unix/* .
cp -r /cdrom/trace32/bin/suns .
mv suns/config.t32 .
chmod -R u+w *
```

In the login script (`.cshrc` in the home directory for the C-shell) the following lines must be added:

```
setenv T32SYS /home/t32
setenv T32TMP /tmp
setenv T32ID T32
```

Prepare and install the fonts:

```
cd /home/t32/fonts
mkfontdir .
xset +fp /home/t32/fonts
xset fp rehash
```

Verify that you have write permission to the system directory and the `boot.t32` file and prepare the configuration file:

```
cd /home/t32
vi config.t32
...
LINK=NET
NODE=t32
```

Copy the executable file to a directory in the PATH, or include it in the \$PATH variable:

```
cp t32cde /usr/bin
```

Preparations for the Ethernet Interface

Before installation a new node must be created. The Ethernet address of the system is placed on the bottom side of the system. The following line must be added to the file /etc/hosts:

```
192.9.200.5    t32
```

Note, that the INTERNET address given here is an example only. Contact your network administrator for a new INTERNET address for TRACE32. The Ethernet address of the system must be entered in the file /etc/ethers:

```
0:c0:8a:0:0:0    t32
```

The INTERNET address is requested by a RARP protocol by TRACE32. If no RARP server is running, the address for the first connect must be set in the host table. After the first successful connect the INTERNET address is stored in nonvolatile memory within TRACE32. The following command sets the host translation table:

```
arp -s t32 0:c0:8a:0:0:0
```

This command must be executed immediately before the first startup of the emulator. It is not required for future startups because the INTERNET address is stored in the emulator. The arp cache table should be checked just before the first startup with the command 'arp -a'.

The net driver must be activated. The node name can be changed, when not identical to 't32'.

Configuration Command:

LINK=NET

NODE=<nodename>

Node name of TRACE32 (default: t32)

POOL=<nodename>,...

Define a set of nodes, which are scanned for connection.

Installation of the TRACE 32 Debugger Software

In the following example the directory `/home/t32` is used as the system directory and the connection is made by Ethernet.

The system directory is created by the following commands:

```
mkdir /home/t32
cd /home/t32
mkdir hp700
```

Extracting the files from CD (the device name is only an example):

```
pfs_mountd &
pfsd &
pfs_mount -t iso9660 -x unix /dev/dsk/c0t2d0 /cdrom
cp -r /cdrom/unix/* /home/t32
cp -r /cdrom/bin/hp700 /home/t32/hp700
mv hp700/config.t32 /home/t32
```

In the login script (`.cshrc` in the home directory) the following lines must be added:

```
setenv T32SYS /home/t32
setenv T32TMP /tmp
setenv T32ID T32
```

Prepare and install the fonts:

```
cd /home/t32/fonts
/usr/bin/X11/mkfontdir
xset +fp /home/t32/fonts
xset fp rehash
```

Verify that you have write permission to the system directory and the `boot.t32` file and prepare the configuration file:

```
cd /home/t32
vi config.t32
...
LINK=NET
NODE=t32
PACKLEN=1024
```

Copy the executable file to a directory in the PATH, or include it in the \$PATH variable:

```
cp t32cde /usr/bin
```

Preparations for the Ethernet Interface

For the adaptation to Ethernet a new node must be created. The Ethernet address of the system is placed on the bottom side of the system. The following line must be added to the file /etc/hosts:

```
192.9.200.5    t32
```

Note, that the INTERNET address given here is an example only. Contact the network administrator for a new INTERNET address for TRACE32.

The INTERNET address is not available to the TRACE32 system. Therefore it can't response to ARP requests. For the first connection, the Ethernet address of the system must be entered in the host table by the following command:

```
arp -s t32 0:c0:8a:0:0:0
```

The net driver must be activated. The node name can be changed, when not identical to 't32'.

On HP-9000/300 and HP-9000/400 and also some HP-9000/700 workstations the packet size must be limited to 1024 bytes. Use the command **PACKLEN=1024** for these type of workstations!

Configuration Command:

LINK=NET

NODE=<nodename> (default: t32)

PACKLEN=<psize> (default:1472)

Troubleshooting

If you can not solve your problem with the following hints contact our support line:

telephone: ++49 8104/8943-50

facsimile: ++49 8104/8943-49

e-mail: support@lauterbach.com

System doesn't response to ping on Ethernet

Internet address already setup in system, or arp used?

When arp is used, it must be used on the same workstation short before.

Ethernet address correct?

System on the correct subnet?

Cables and transceiver o.k.?

Ethernet software in host (PC) configured correctly?

xset +fp fontpath gives error 'bad value ...'

Does the font directory exist?

Does the fonts.dir file exist (created by mkfontdir)?

Is the directory seen under the same name by the X-server?

Have all directories that lead to the font directory read and execute permissions for everybody?

Executable program does not start or gives fatal error

When transferring between different OS-systems, files copied in binary mode? Access rights to file in directory o.k.?

Configuration file contents o.k.?

Executable program displays 'FATAL ERROR selecting device-driver ...'

Using configuration file for MS-DOS for the WINDOWS-Driver?

WINDOWS and workstation drivers cannot load new drivers.

Environment variable 'T32CONFIG' and/or 'T32SYS' correctly set?

Executable program displays 'error reading config.t32:'

Configuration file contents o.k.?

Commands in file in uppercase?

Blanks inserted/not inserted?

Device specific commands placed after device header?

Device configuration blocks separated by empty lines?

Environment variable 'T32CONFIG' and/or 'T32SYS' correctly set?

Executable program stops without message, but with window opened

Access rights to directory o.k.?

On UNIX host, try with 'NOLOCK' feature.

When using the RS232 interface: Is a login process active on the tty?

Program stops with message 'font xxxx not found'

Do fonts appear in the 'xlsfonts' command?

Can one font (e.g. t32-lsys-16) be displayed by 'xfd -fn t32-lsys-16'?

Fonts added to X-Windows FONTPATH?

Fonts converted, when required, and .bdf files removed?

Command to generate font directory executed with correct parameters?

Fonts installed on the X-Windows server, not client?

If using an X-Terminal, use the conversion programs for the X-Terminal?

Executable program displays 'boot.t32 not found'

Access rights to directory o.k.?

Read and write access to boot.t32 (write required on UNIX without NOLOCK)? Configuration file contents o.k.?

Environment variable 'T32SYS' correctly set?

Executable program stops after displaying 'error reading boot.t32'

When transferring between different OS-systems, files copied in binary mode? Access rights granted?

Try again after switching off the TRACE32 system?

Executable program stops after displaying 'booting...' or 'finished.'

When transferring between different OS-systems, files copied in binary mode? Packet size set correctly on Ethernet, handshake set when required?

Bootloader stops with message 'fatal error ...'

When transferring between different OS-systems, files copied in binary mode? Mixing different versions of the software, e.g. MCC.T32 and MCCxxx.t32?

Bootloader displays 'cannot save image...'

Write access right on system directory?

Disk full?

Existing read-only file?

Software crashes or stops after booting is finished

Boot image file maybe destroyed, remove all boot0x.t32 files?

Connection of modules o.k., connector bend?

Software doesn't work stable

Boot image file maybe destroyed, remove all boot0x.t32 files?

Connection of modules o.k., connector bend?

Check connection of Fibre Optic, Ethernet or Parallel interface.

On Ethernet try with smaller packet size and/or handshake.

Emulation system doesn't work correctly

Check Emulation Probe Manual in 'Targets' part of the manual.

Parallel Port not working stable

Check that the port is on the correct mode. Choose either EPP 1.9 or compatible mode.

The mode selection can usually be done in the BIOS setup (can be activated during booting).

Tutorial for TRACE32-ICD

About the Tutorial

What is it about?

This is a tutorial for all **In-Circuit Debuggers (TRACE32-ICD)**, that are implemented using their onchip debug interface. Typical examples for onchip debug interfaces are BDM, JTAG, ONCE ...

Preconditions:

The tutorial assumes that the **development system is already installed**. You should have basic knowledge of the C-programming language in order to be able to follow the example code found in this tutorial. In addition working with WINDOWS is assumed as known. Also some knowledge of the used processor and assembler/compiler is necessary to get your debug system running.

Purpose of this tutorial:

The purpose of this tutorial is **to get your debug system running**, to write the batch job that does the necessary startup procedure and **to make you familiar with the main features** of the In-Circuit Debugger.

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. You should perform a number 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.

Where can I get more information:

The common TRACE32 user interface contains a detailed Online-Help, that offers the most current description of all debug features. Refer to [The Online Help](#) on how to start the Online Help system.

How long does it take?

60 minutes.

Start TRACE32-ICD

If your debug tools are installed in accordance to **Quick Installation for TRACE32-ICD**, power up your debug system first and then the target.

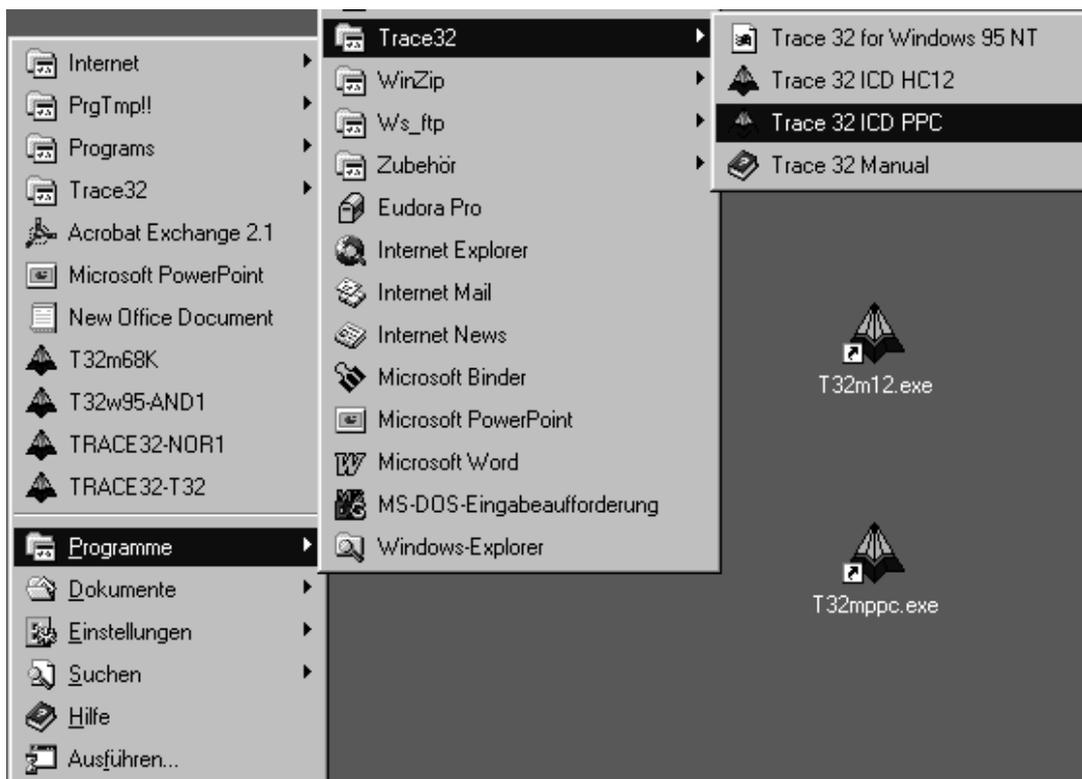


To work with the In-Circuit Debugger (ICD) a working target system is required!



**Take care of the proper sequence on powering up/down:
Power Up: debugger - target
Power down: target - debugger**

To start the debugger software on your host open the **TRACE32** folder in the start menu and start the TRACE32 user interface. If you have generated an icon on your desktop, double click there. In the example below the software for two processor families (PowerPC and 68HC12) is installed.



Setup the Application Properties

The property window of your application allows same basic setting for the debugger.

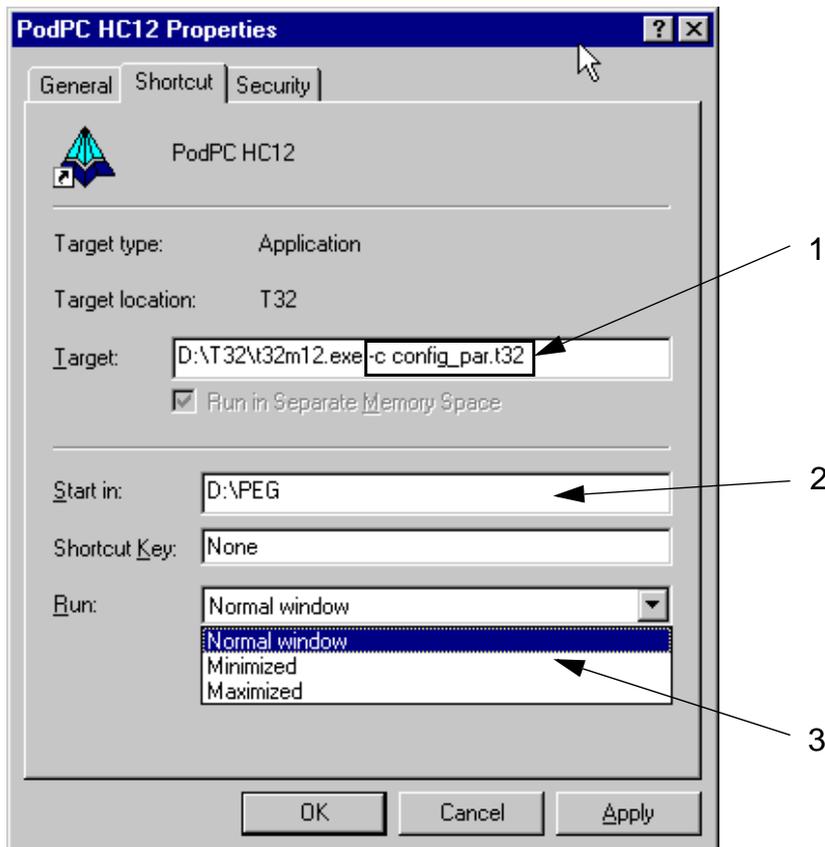
1. Definition of an user specific configuration file.

By default the configuration file `config.t32` in the system directory is used. The option `-c` allows you to define your own location and name for the configuration file. For more information on the configuration file refer to ***Quick Installation for TRACE32-ICD***

2. Definition of a working directory.

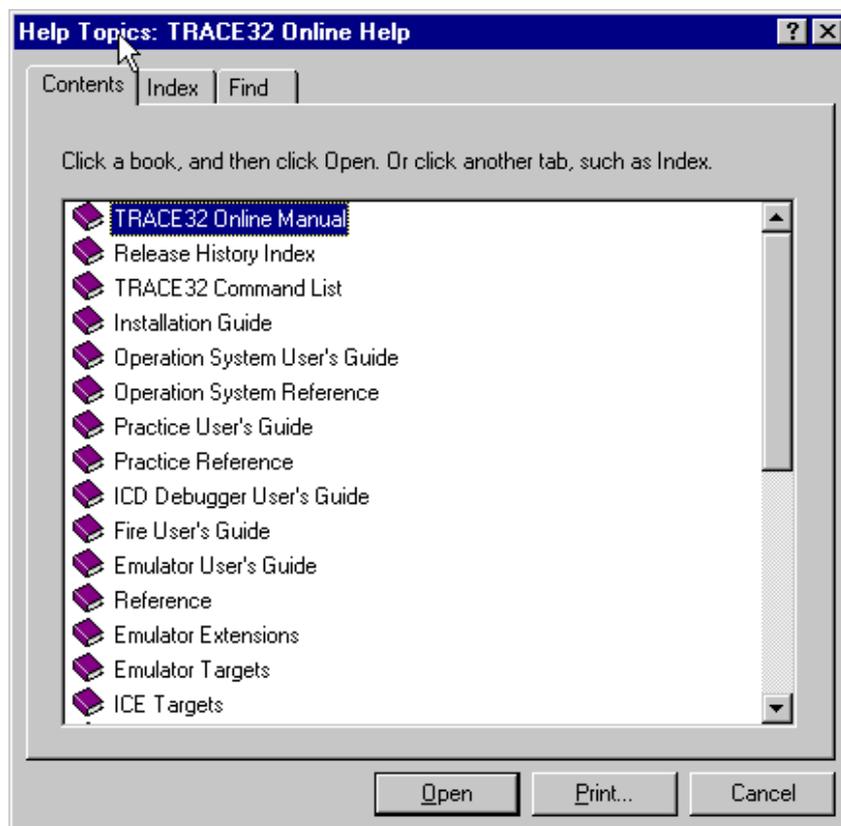
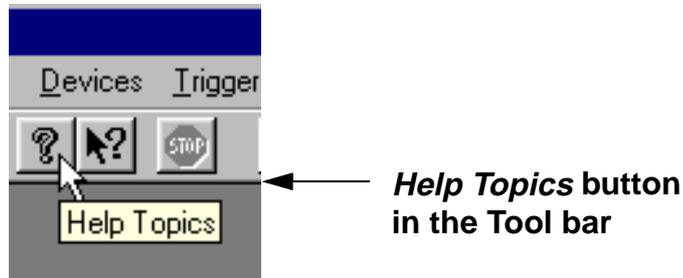
It is recommended not to work in the system directory.

3. Definition of the start-up size of the application window.



The Online Help

The most current version of the TRACE32 books is available in the online help. Use the **Help Topics** button to get access to the TRACE32 online help.

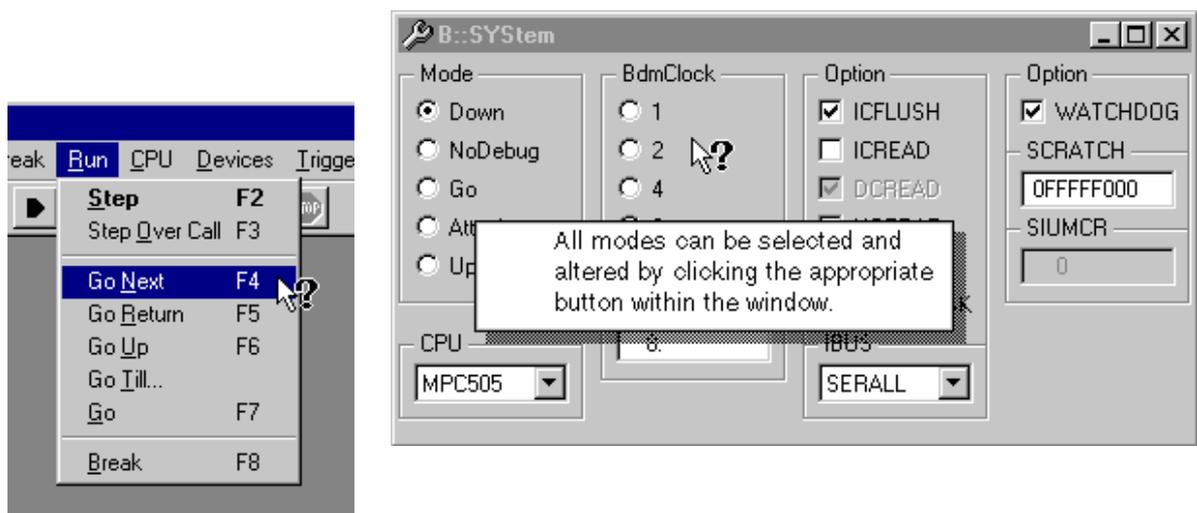


The online help provides also a powerful context sensitive help.

Use the **Help Context** button on the tool bar to activate the context sensitive help



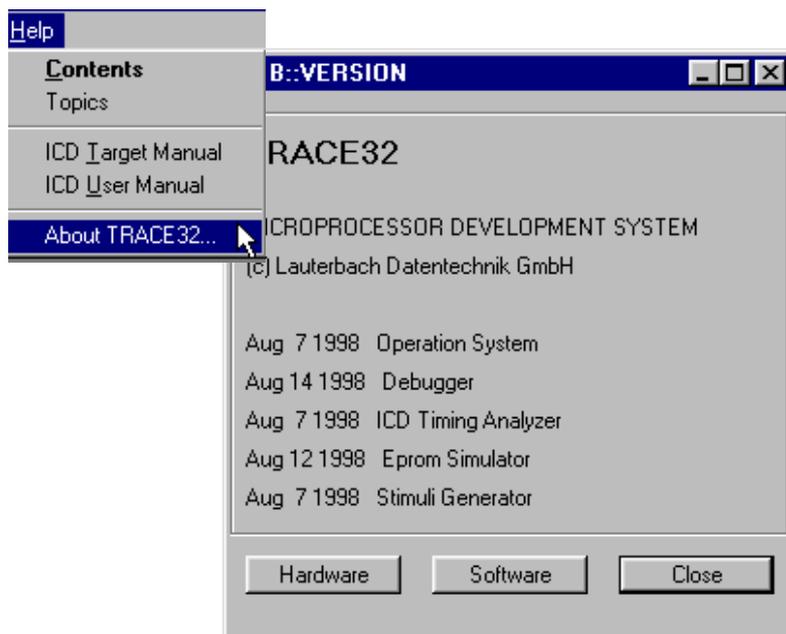
Click first on the **Help Context** button and then move the quotation mark cursor on the object you are interested in. A popup window is opened to display information about this object.



Like the command **Go** with a temporary breakpoint set to the next assembler command or next HLL line. This command can be used to overstep a subroutine call instruction or to leave a loop. See also the command **Step.Over**.

See also [Go.Next Continue program](#)

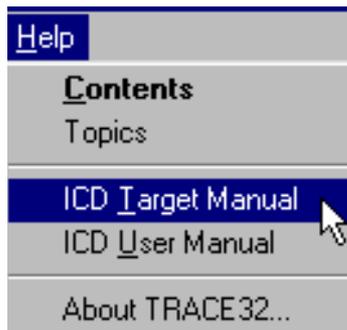
The **About TRACE32...** command in the **Help** menu provides version information for all parts of TRACE32-ICD.



Setup the Debug Environment

In order to set-up your debugger, you need some knowledge about your CPU and about your target configuration. To be able to download your program including all symbol and debug information you also need some knowledge about your compiler.

A basic start-up procedure and the CPU specific setting for the ICD-Debuggers are described in the *ICD Target Manual*.



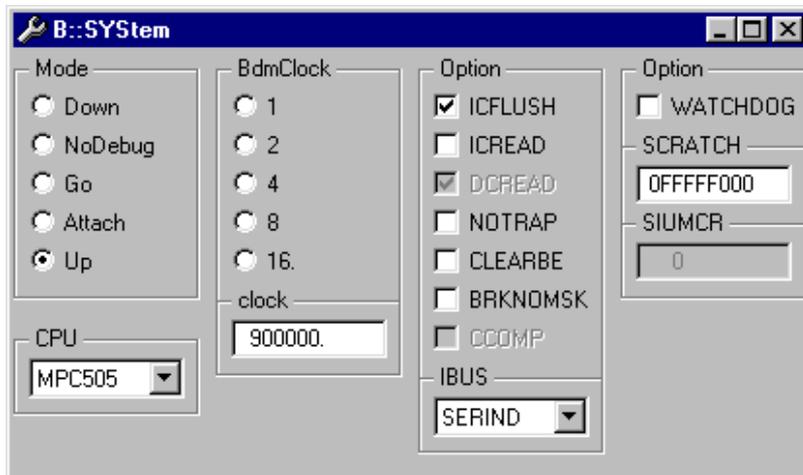
ICD Target Manual gives quick access to the settings and additional features for your CPU

Look at the *Reference* book for a detailed description of all generic commands.

A typical start-up procedure consists of:

1. The CPU specific settings.

The **SYStem** Window provides all CPU specific settings. Use **System Settings...** in the **CPU** menu to open this window.



Inform the debugger about the CPU type on your target, if an automatic detection of the CPU is not possible. Select the correct CPU type from the pull down menu in the field **CPU**.

(Command: **SYStem.CPU** <CPU type>)

Set the system options in the option field corresponding to your target configuration and application program.

(Command: **SYStem.Option** <option>)

*On some processor types the special function registers can be moved. The command (**SYStem.Option** **BASE**) is used to inform the debugger about the new base address (address of the first SFR). If the debugger does not have the correct base address no or wrong data will be displayed and **FLASH** programming might not be possible.*

Set the transfer clock from the debug interface to your target. By default TRACE32-ICD uses a fixed clock to run the debug interface. A clock range from 100KHz up to 5 MHz can be used. (Command: `system.BdmClock`)

We also provide a clock cable to allow you the use of the divided CPU clock as clock for the debug interface. The relation between the CPU clock and the debug interface clock is specific for your CPU. Refer to your *ICD Target Manual* for detailed information. The use of the divided CPU clock has the following advantages:

- The max. speed for the debug interface can be used. However we recommend 10MHz as the max. speed.
- The clock for the debug interface is automatically adapted if the CPU clock is changed by your application program.

2. Enter the debug mode.

Select the **Up** button in the **Mode** field to restart the CPU with debug mode enable. (Command: `system.Up`)

The user interface establishes the communication to the target's microprocessor. After this command you should be able to access the registers.

3. Do the target specific settings.

The CPU is active now and you can initialize the CPU by writing to the special function registers using the **Data.Set** command. E. g. some CPU need to set the chip selects in order to access memory.

4. Load your application.

Load your application by using the command **Data.LOAD.<option> <file_name>**.

The option required for your compiler can be found in the *ICD Target Manual* in the section **Compiler**.

If the file should be loaded to an Eeprom, the memory class EEPROM must be used to generate the required programming sequence. Example: d.load.b epromdata EEPROM:

*For flash programming refer to the **FLASH** command group in the **Reference** book.*

To display the source code the compiled program must be equipped with debug information (frequently: compiler option “debug”). Then TRACE32 can load the compiler output formats directly.

5. Initialize program counter and stackpointer: **Register.Set**

Many compilers add these settings in the startup code to the user program automatically.

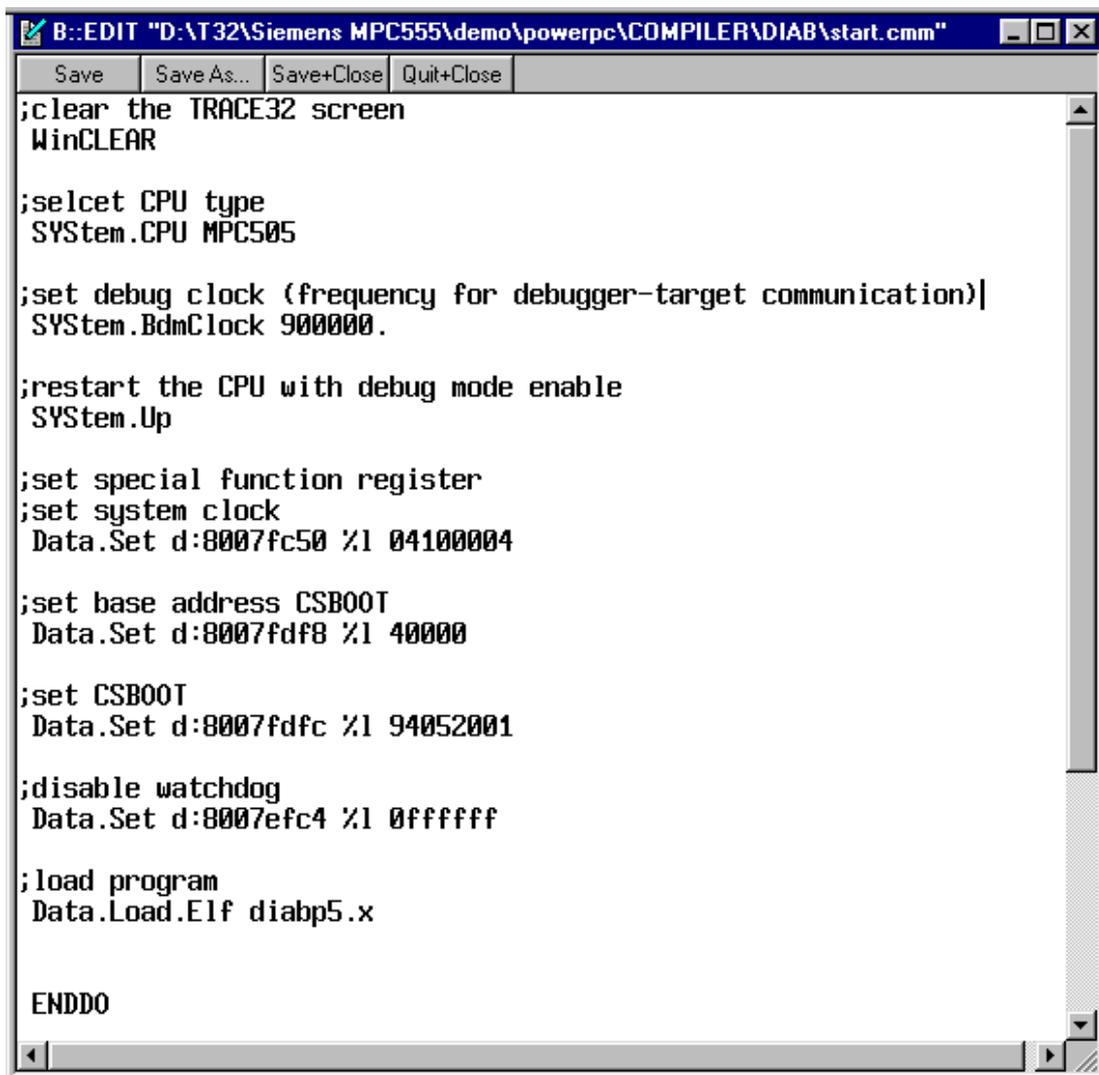
It is recommended to write a batch job to set up the debugger to guarantee a proper start-up sequence.

Batch Jobs

Create a new batch file **start.cmm** in your working directory by using the command **PEDIT start.cmm**.

*TRACE32 has its own command language for batch jobs. It is called PRACTICE and it is very powerful (see the **PRACTICE User's Guide** and **PRACTICE Reference** for more information). All commands of the TRACE32 development tools, commands for program flow, conditional commands and I/O commands are allowed. The default extension for batch files is ".cmm".*

*Also debugging of a PRACTICE program is supported. Look at the description in the **PRACTICE User's Guide** and **PRACTICE Reference** (commands: **PLIST**, **PEDIT**, **PBREAK**).*



```
B::EDIT "D:\T32\Siemens MPC555\demo\powerpc\COMPILER\DIAB\start.cmm"
Save Save As... Save+Close Quit+Close
;clear the TRACE32 screen
WinCLEAR

;set CPU type
SYStem.CPU MPC505

;set debug clock (frequency for debugger-target communication)
SYStem.BdmClock 900000.

;restart the CPU with debug mode enable
SYStem.Up

;set special function register
;set system clock
Data.Set d:8007fc50 %1 04100004

;set base address CSBOOT
Data.Set d:8007fdf8 %1 40000

;set CSBOOT
Data.Set d:8007fdfc %1 94052001

;disable watchdog
Data.Set d:8007efc4 %1 0ffffff

;load program
Data.Load.Elf diabp5.x

ENDDO
```

Enter the required commands and finish the batchjob by **ENDDO** and click the **Save** button. The picture above shows a startup procedure for the PowerPC505.

Start the startup procedure by using **Batchfile...** in the **File** pulldown menu.

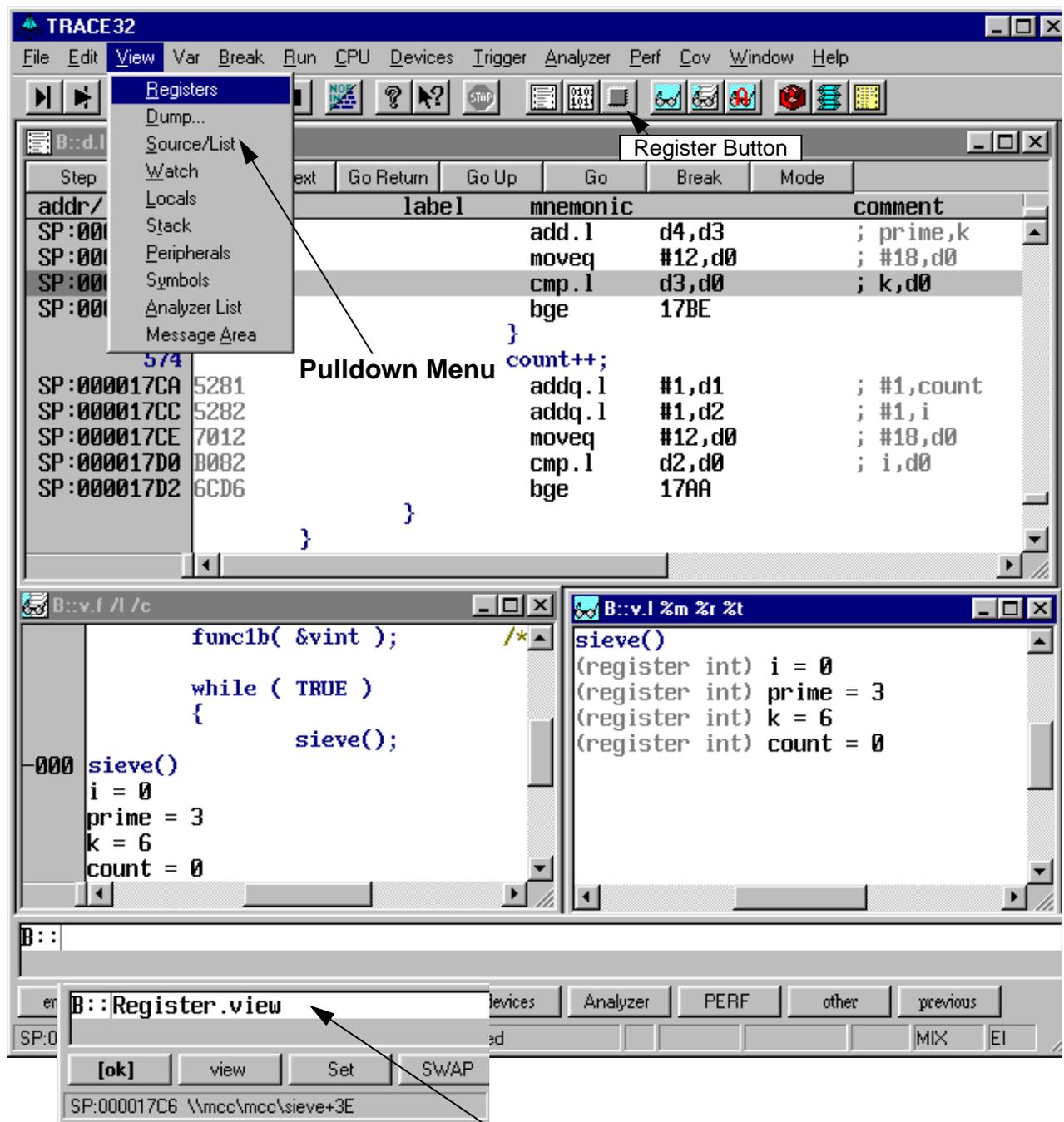


To continue our tour take one of the example files you can find in the TRACE32 system directory under

`\demo\
e.g. \demo\powerpc\compiler\Diab\Diabc.cmm.`

or use your own batch file, if you have already prepared one.

The User Interface



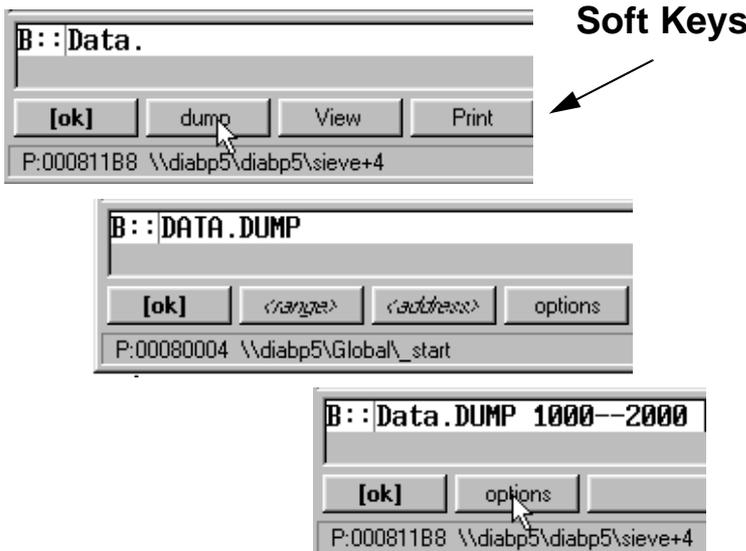
Open a window to display the CPU registers. You can alternatively select **Registers** from the **View** pulldown menu, push the **Register** button or enter `Register.view` at the prompt `B::` in the command line.

Most features can alternatively be selected from a pulldown menu, from a button in the main tool bar or from the command line. Please remember this even if we use just one way 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. Another example which shows the typical TRACE32 command structure <command family>.<subcommand> is Data.List that can be shortened to d.l.

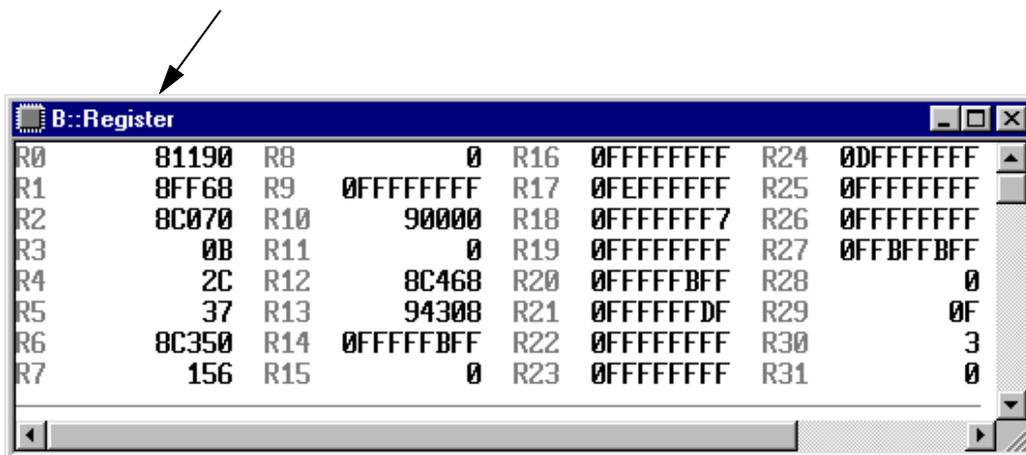
A good hint is to look at the soft keys. They provide a guided command entry. All possible commands and parameters are displayed. Instead of writing to the command line you can assemble the correct command by clicking on the soft keys.

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



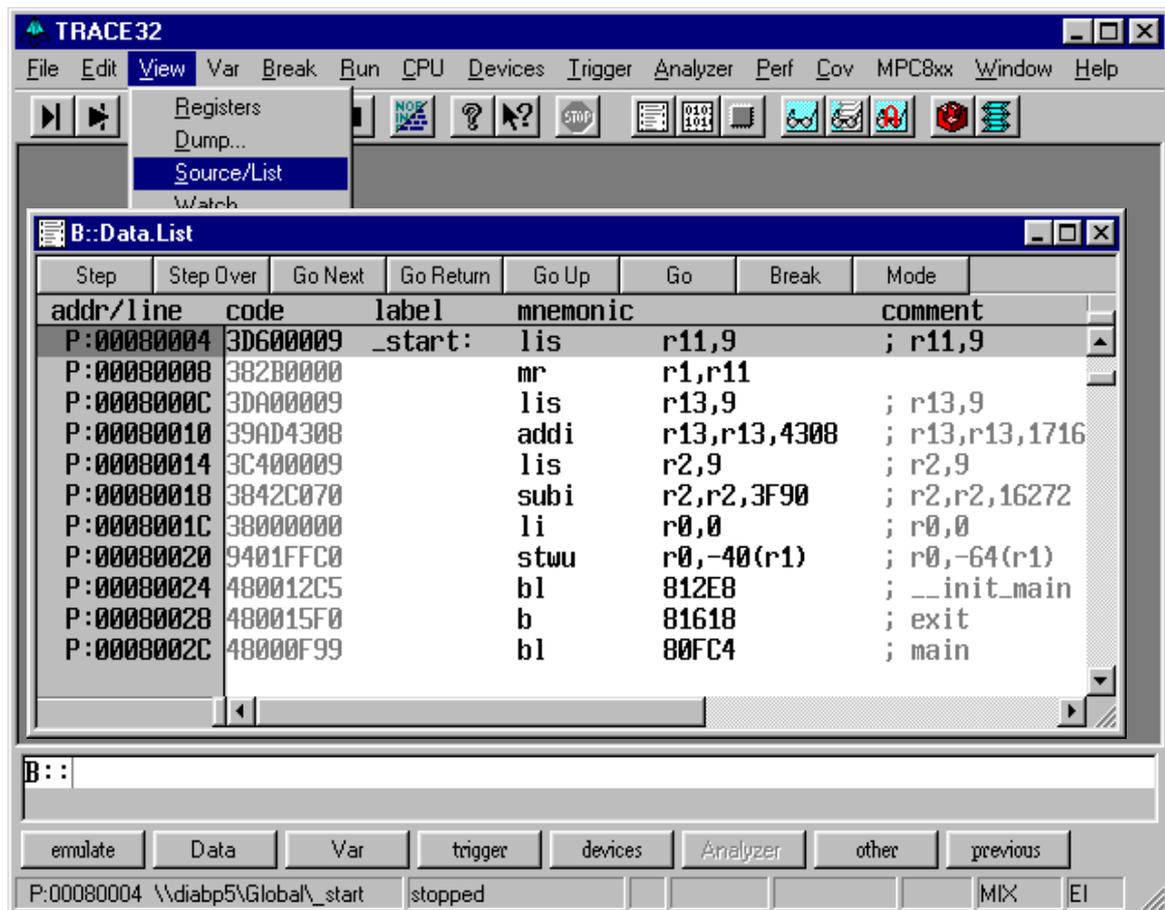
More detailed information about the TRACE32 user interface can be found in the **Operating System User's Guide**.

In the window header the TRACE32 command that was executed to open the window is displayed.

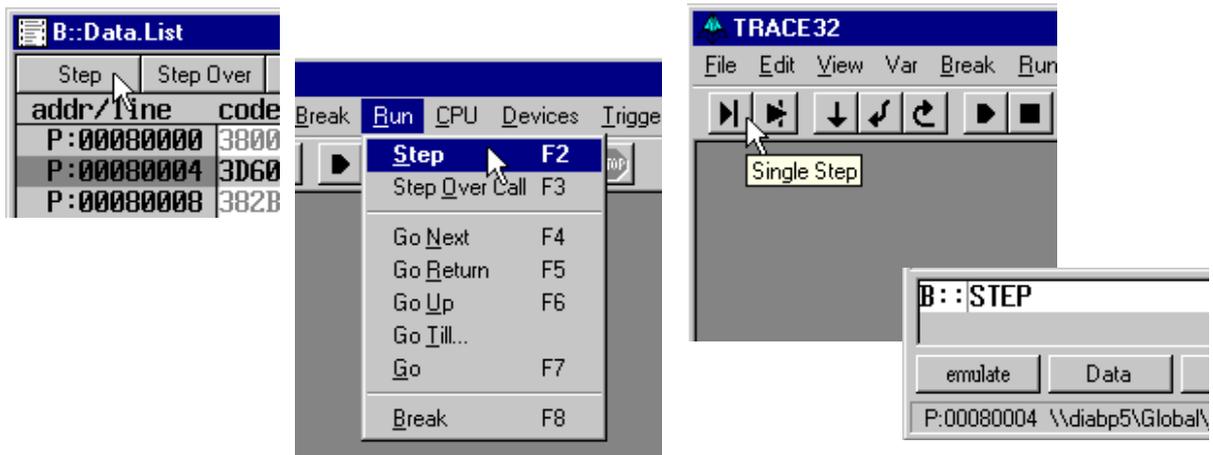


How to Debug the Program

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



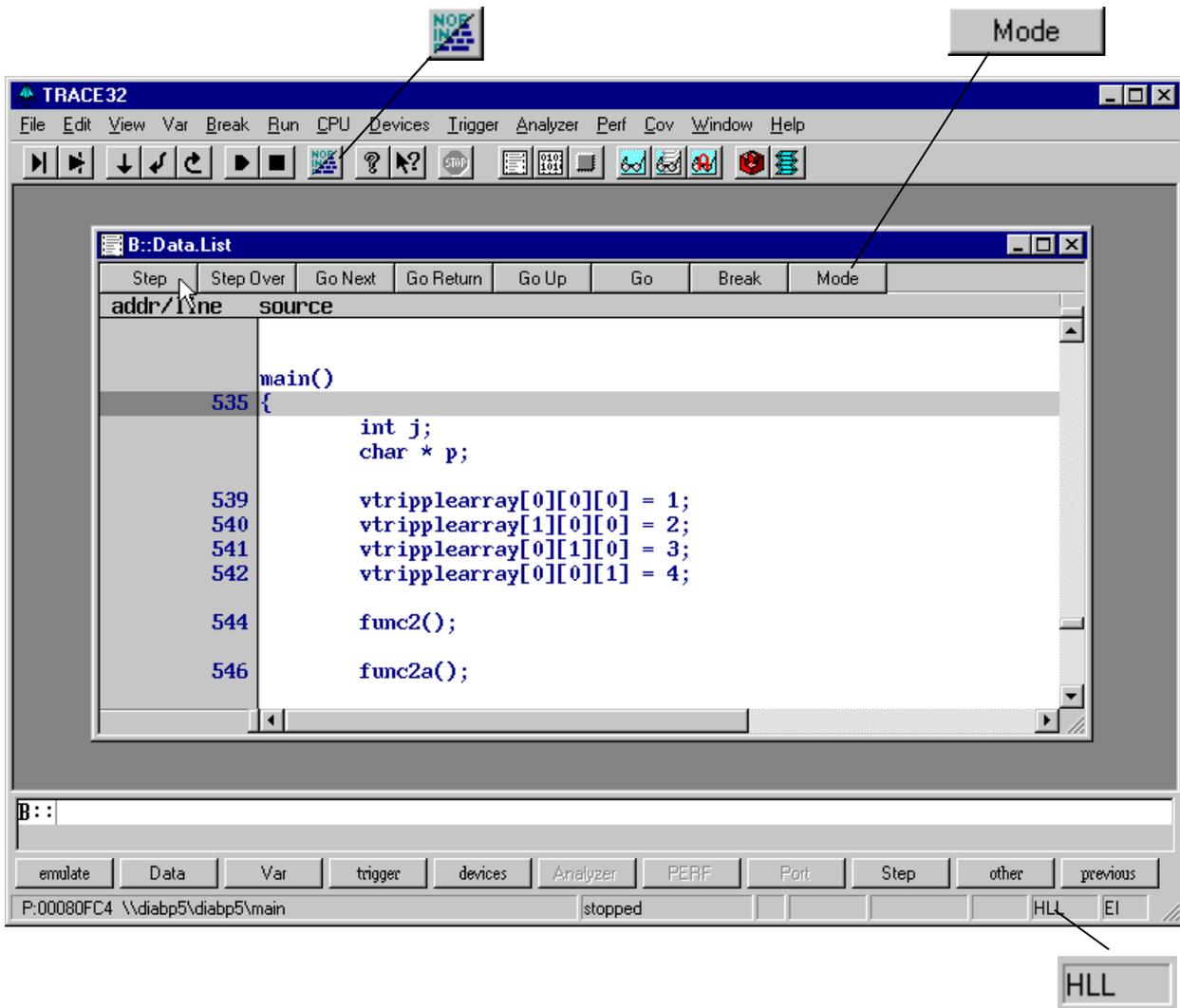
Single step through the program by clicking the **Step** button in the **Data.List** window:



Remember: Mostly there are many possibilities to get the same result. Here you can alternatively use the pulldown menu, the accelerator F2, the speed bar or the command line.

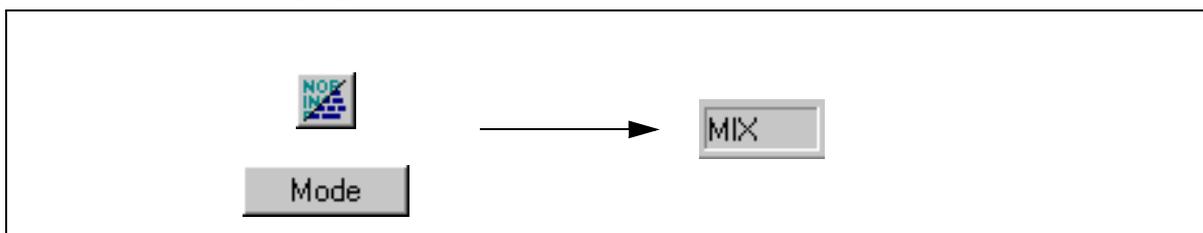
Toggle the debug mode from **MIXed** to **HLL** by a click on the **Mode** button in the **Data.List** window. The state line shows the debug mode HLL: High Level Language. Then do another **Step**.

Button to toggle between MIXed and HLL mode



The step you did was a high level language step to the next HLL line.

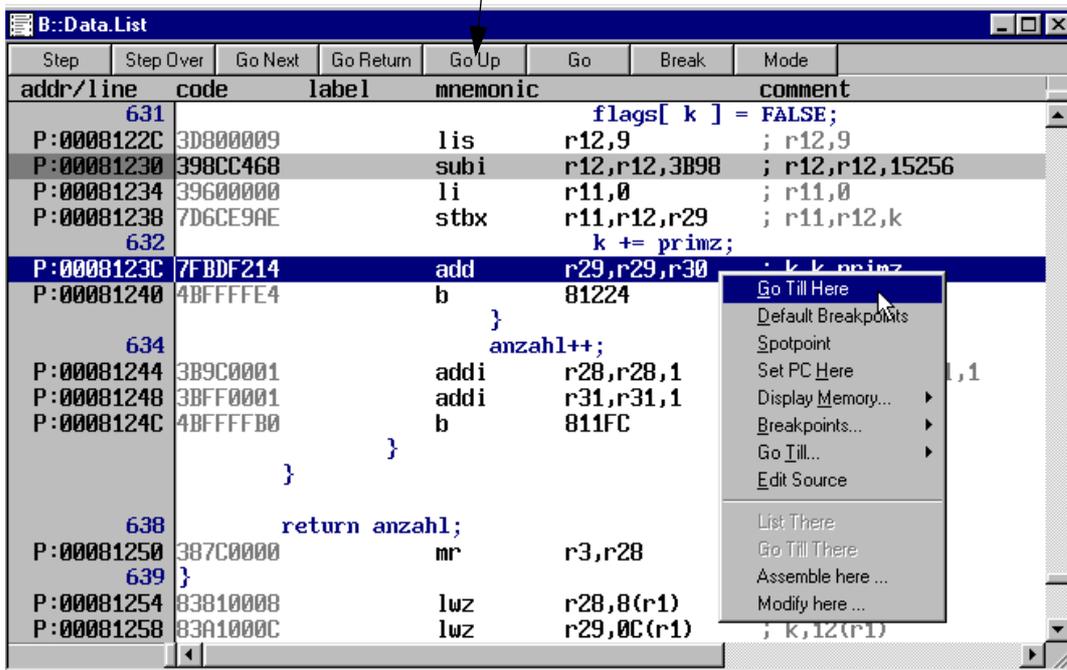
If you toggle back to MIXed mode, both the C-source and the corresponding assembler commands will be displayed.



If you single step while an interrupt is pending, the next step will enter the interrupt routine. If you want to single step your program without entering an interrupt routine use:

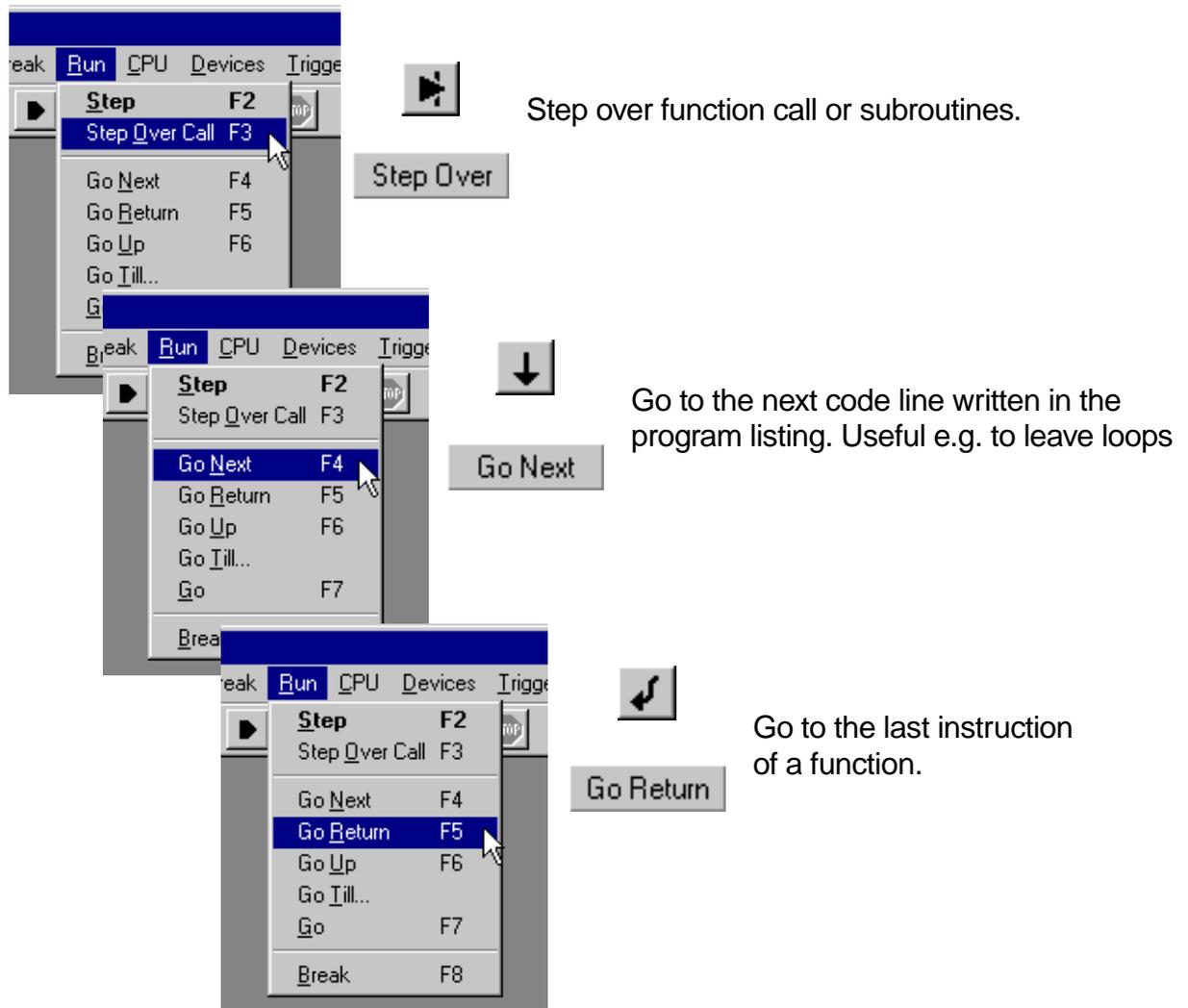
- **SETUP.IMASKASM ON** to disable the interrupts during single stepping on assembler or mixed level.
- **SETUP.IMASKHLL ON** to disable the interrupts during single stepping on HLL level.

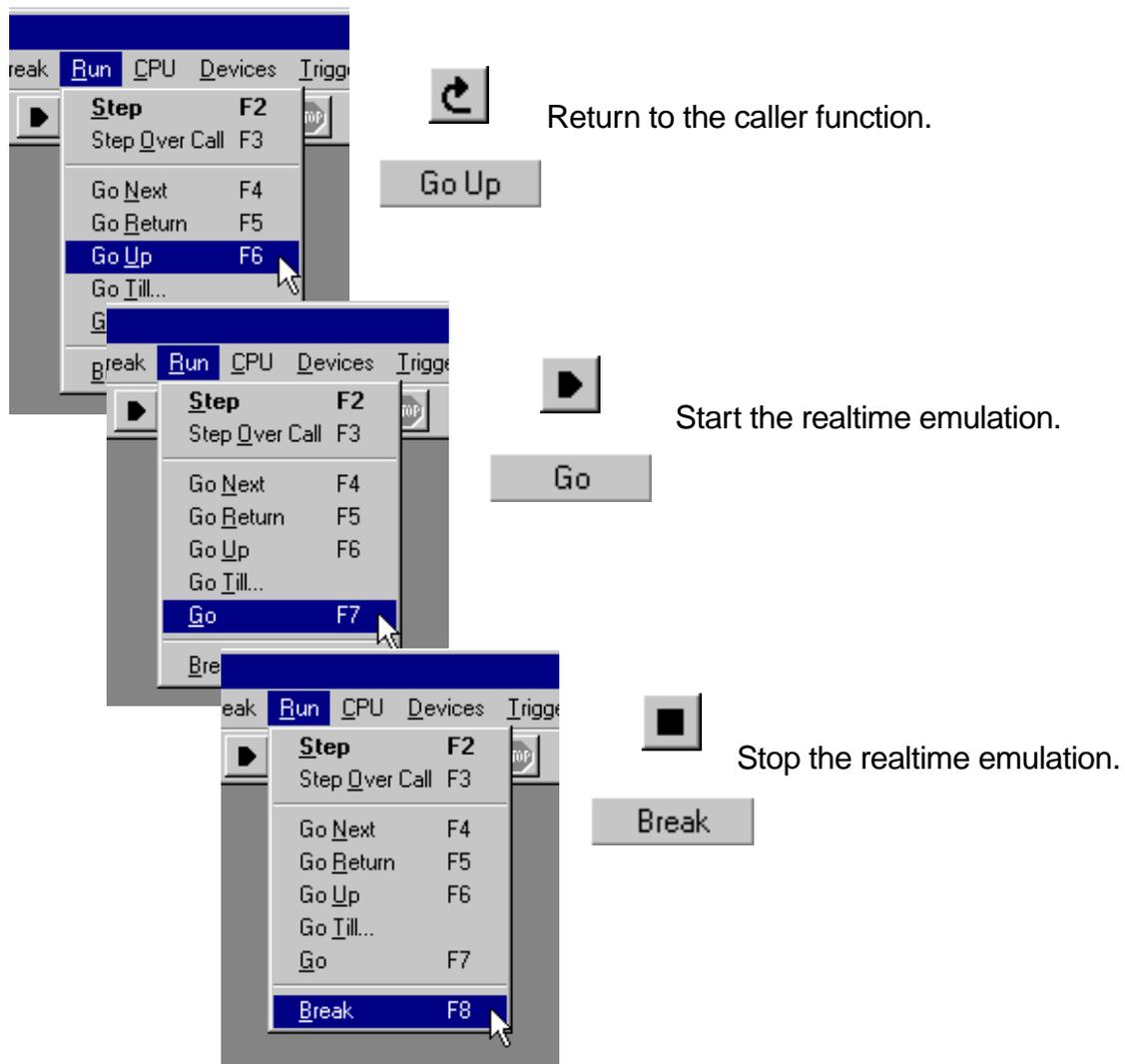
Local buttons for the Data.List window



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

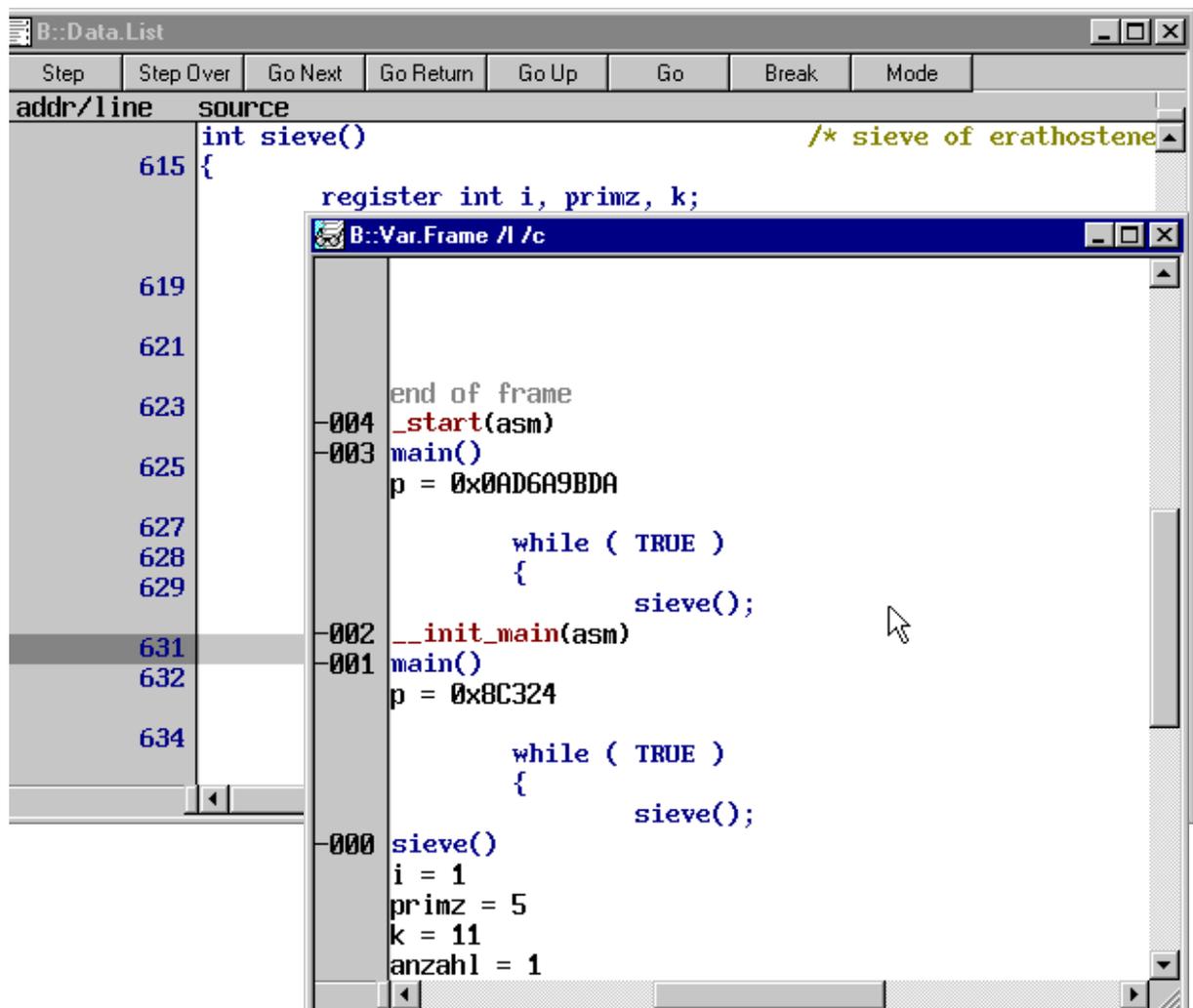
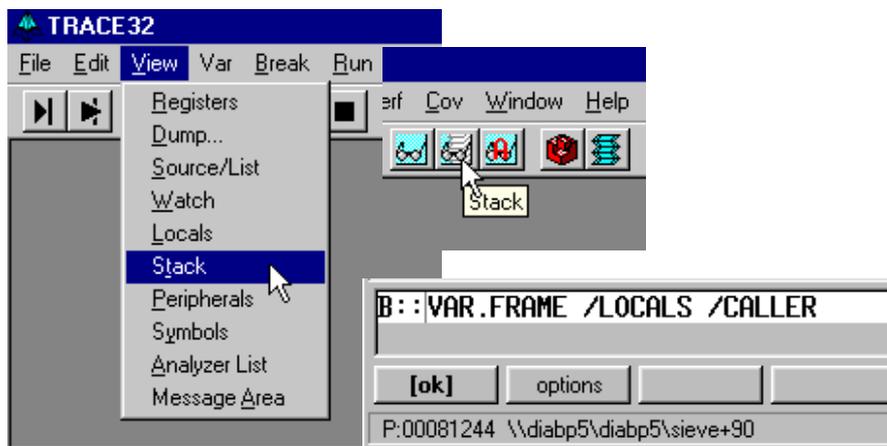
Single Stepping is one of the basic debugging commands. Look at the **Run** pulldown menu, at the local buttons of the **Data.List** window or at the main tool bar for the other debug commands.





The commands Go Next, Go Return and Go Up are only available if the program is running in RAM or if the CPU provides onchip breakpoints.

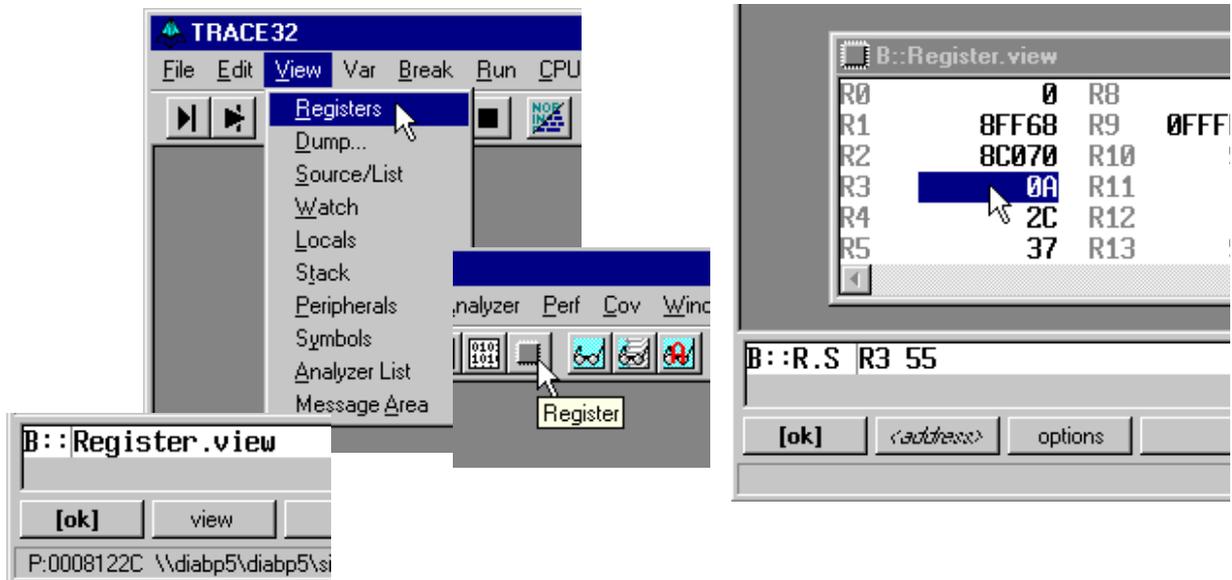
The **Var.Frame** window displays the function nesting for your application program. With the option LOCAL the local variables of each function are displayed. When the option CALLER is set, a few lines from the C-code are displayed to indicate where the function was called.



*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. More detailed information can be found in the **Reference** book at the description of the commands [Step.Change](#), [Step.Till](#), [Go.Change](#), [Go.Till](#), [Var.Step.Change](#), [Var.Step.Till](#), [Var.Go.Change](#) and [Var.Go.Till](#).*

How to Display and Modify CPU Registers

We want to inspect the CPU registers now.



Try to change a register value by double-clicking on the value you want to change. The **Register.Set** command for the selected register is displayed in the command line. You only have to enter the new value and confirm it with return (see right picture above).

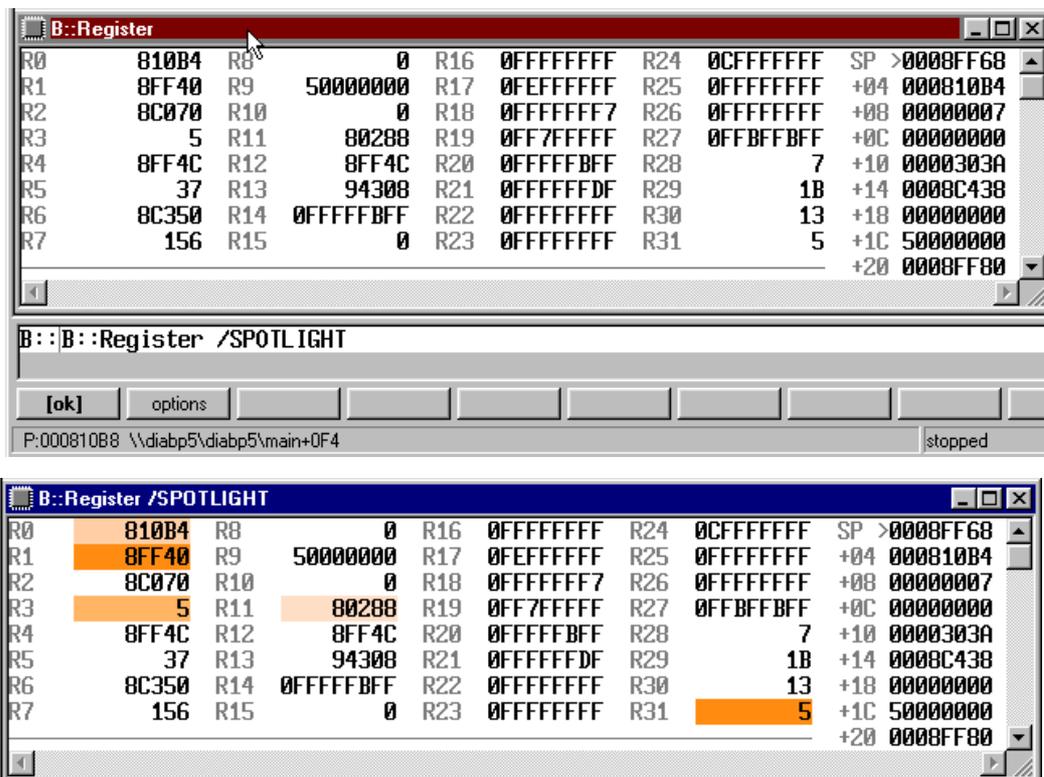
If the registers changed by the last steps should be marked in the **Register** window:

1. Click to the window header with the right mouse button.

The command, that was used to open the window is displayed in the command line for modification. The window header becomes red.

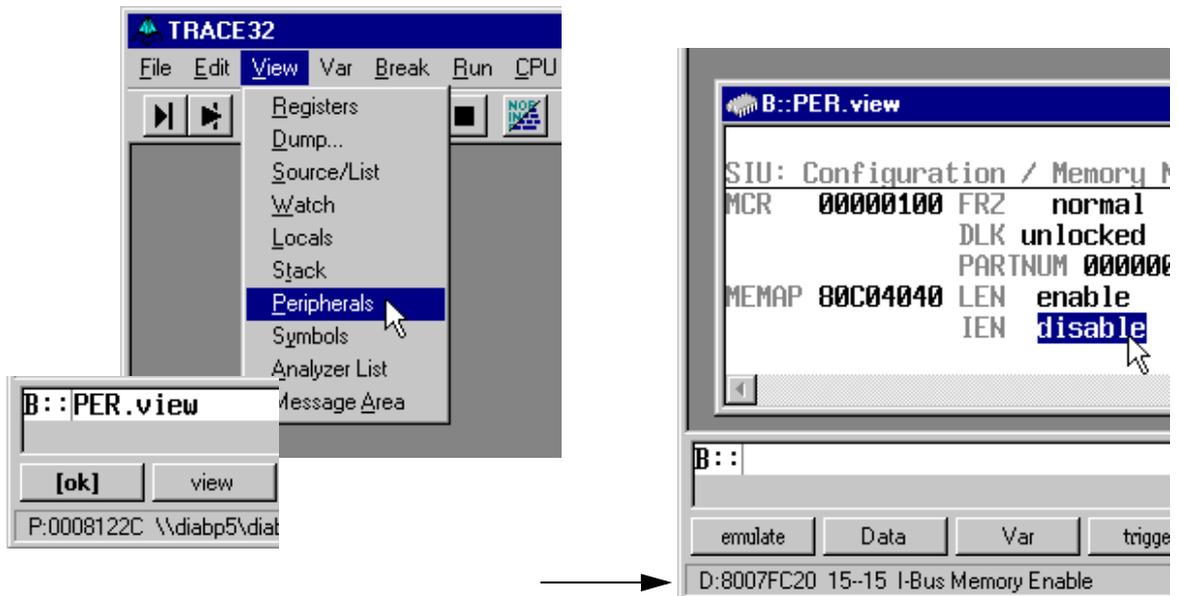
2. Set the option **/SpotLight** and confirm the modification with return.
3. Execute a few single steps.

The registers changed by the last step are marked in dark red. The registers changed in by the step before the last step are marked a little bit lighter etc. This works up to a level of 4 step.



How to Display and Modify the Special Function Registers

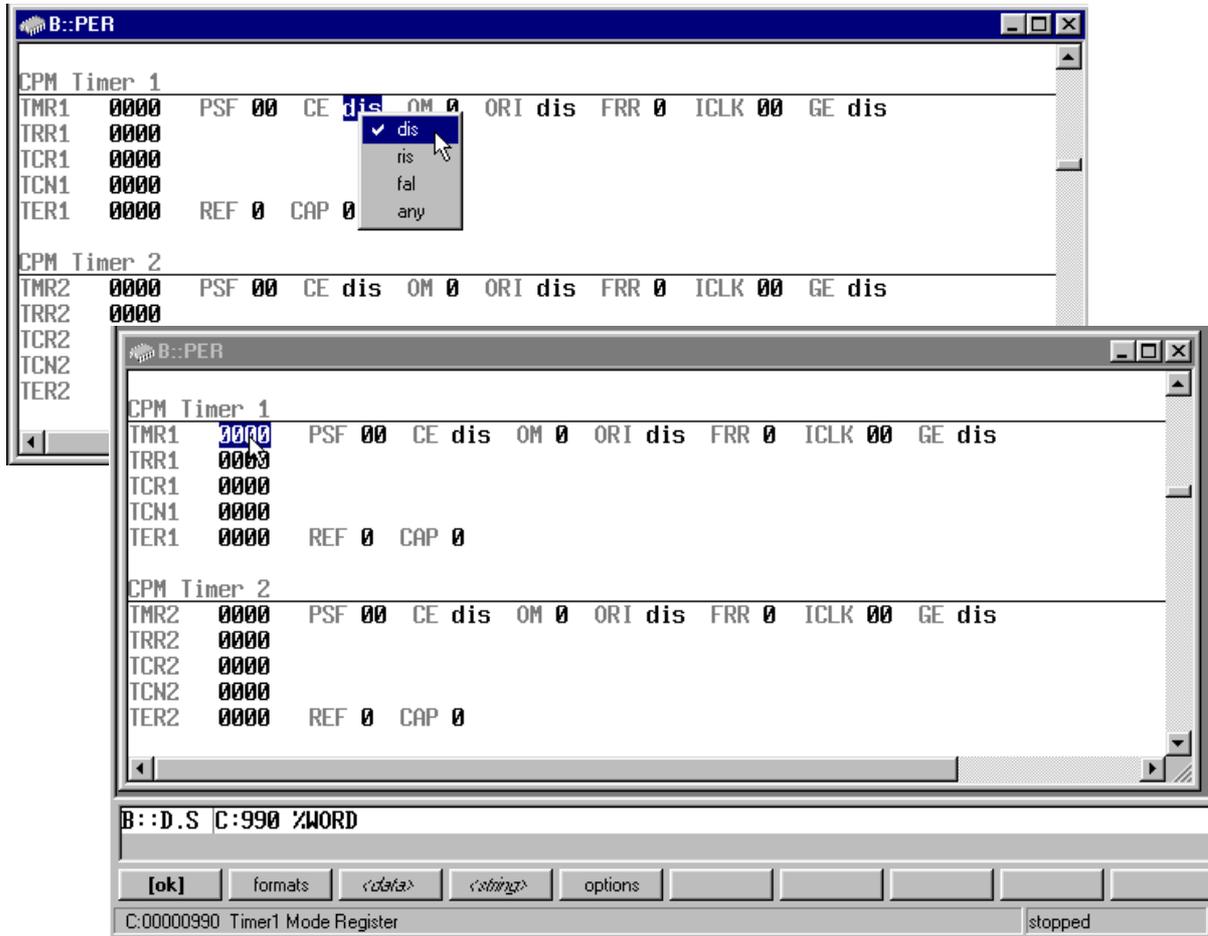
Open the **Peripherals** window to display your CPU's special function register:



If you select the register contents the address, bit position and complete name of the special function register is displayed in the state line.

You can modify the contents of a special function register:

- By pressing the right mouse button and selecting one of the predefined logical value from the pulldown menu.
- By a double-click to numeric values. A **Data.Set** command to change the register contents is displayed in the command line. Enter the new value and confirm it with return.



How to Display and Modify Memory

To inspect an address range in the memory use the **Data.dump** window.

Select **Dump...** from the **View** menu. The **Dump Memory** dialog box is opened.

- Use the **Browse** button to browse through the symbol data base. Select a label by a double-click and then confirm by pushing **OK**.
- Or enter the address directly in the **Address** field and push **OK** to open the **Data.dump** window.

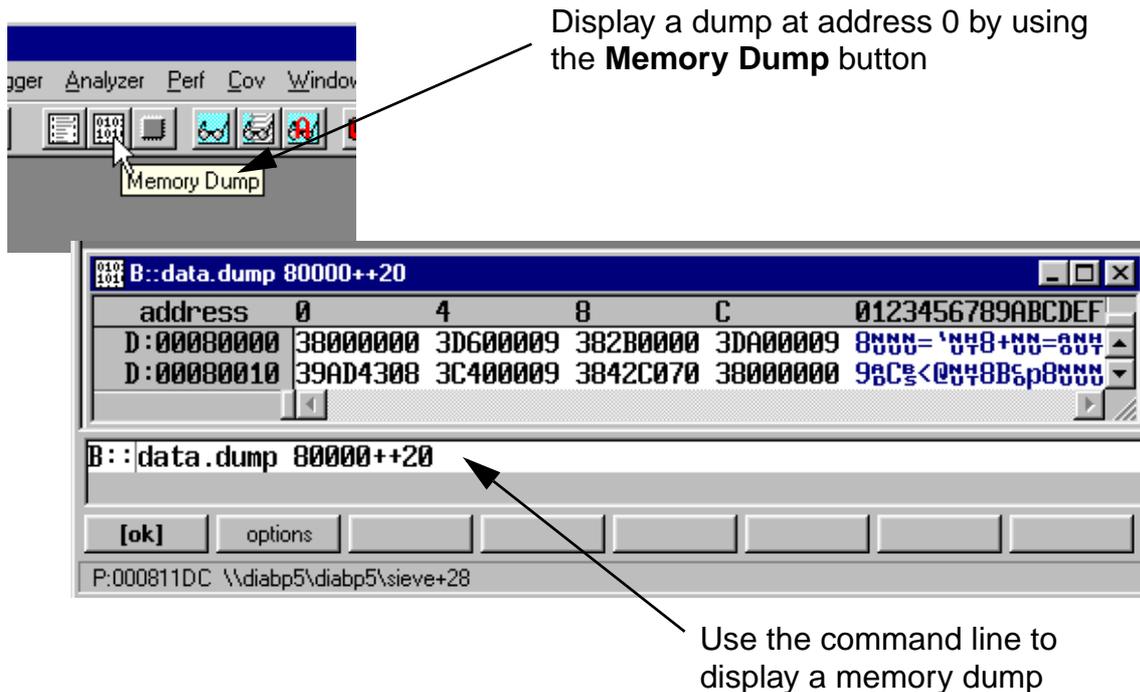
The screenshot illustrates the steps to display memory in TRACE32. It shows the **View** menu with **Dump...** selected, the **Dump Memory** dialog box, a symbol browser window listing symbols like **def**, **enumvar**, **flags**, and **func0**, and the **Data.dump** window showing a memory dump table.

The **Data.dump** window displays the following table:

address	0	2	4	6	8	A	C	E	0123456789ABCDEF
D:0008C460	0000	0000	0000	0000	0101	0100	0101	0001	NNNNNNNNNNSSSSSSNS
D:0008C470	0100	0100	0001	0100	0001	0000	0000	0000	SSSSSSSSSSNNNNNNNN
D:0008C480	0000	0000	0000	0000	0000	0000	0000	0000	NNNNNNNNNNNNNNNNNN
D:0008C490	0000	0000	0000	0000	0000	0000	0000	0000	NNNNNNNNNNNNNNNNNN
D:0008C4A0	0000	0000	0000	0000	0000	0000	0000	0000	NNNNNNNNNNNNNNNNNN
D:0008C4B0	0000	0000	0000	0000	0000	0000	0000	0000	NNNNNNNNNNNNNNNNNN

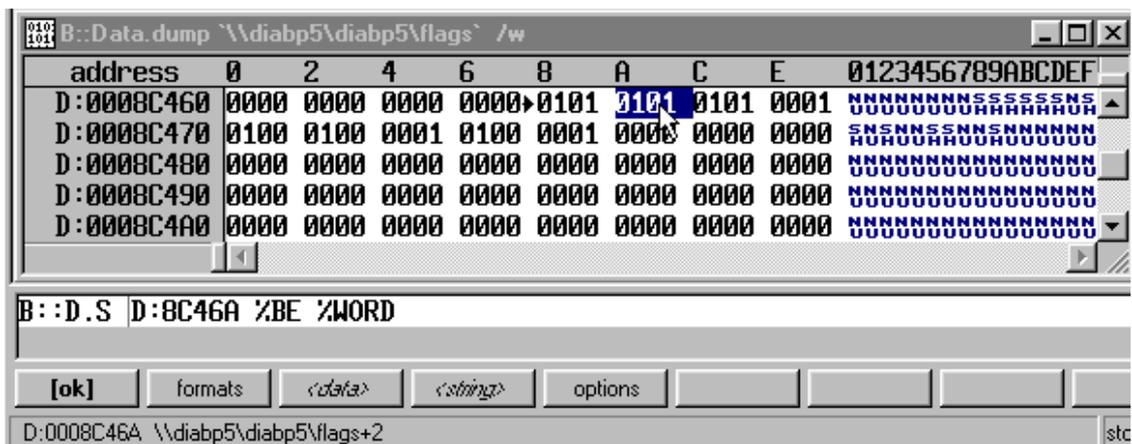
Arrows point to the following labels below the table:

- Memory Class + Address (pointing to the first column)
- Value (pointing to the data columns)
- ASCII (pointing to the hex dump column)



There are two different ways to define an address range:

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



Again, 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.

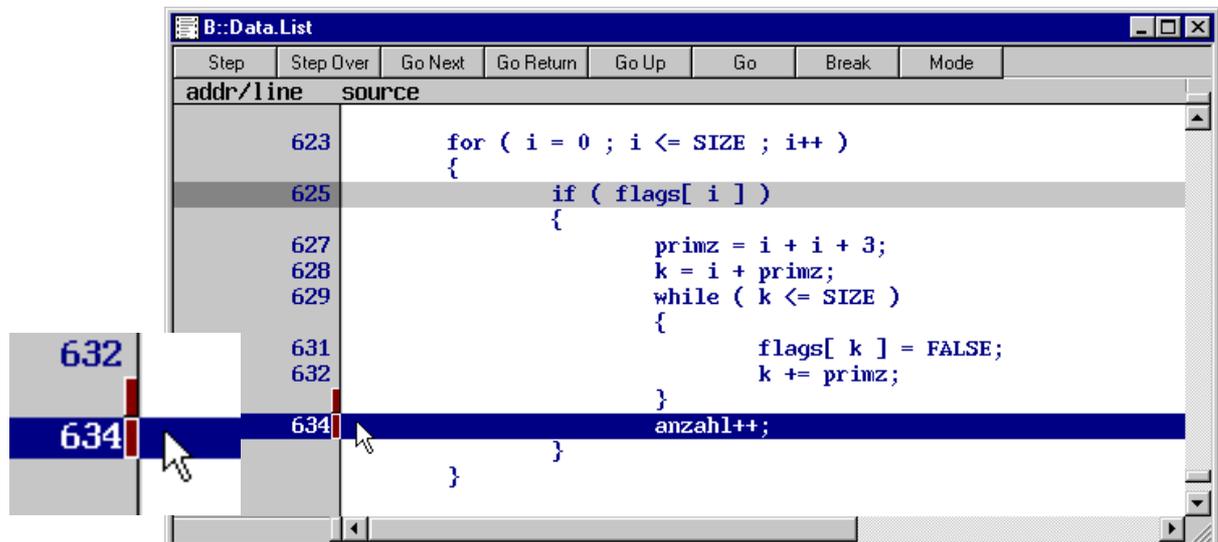
How to Set Breakpoints

Software Breakpoints

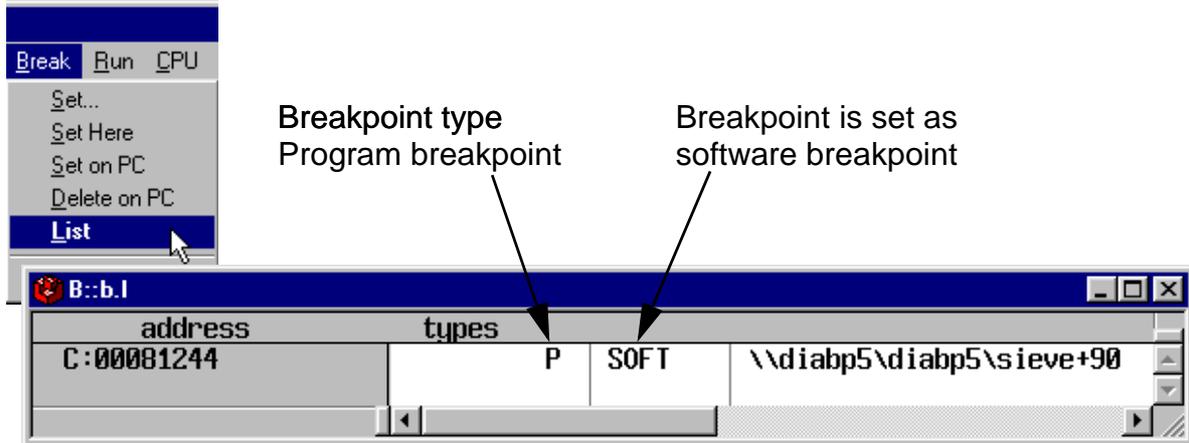
The ICD debuggers use software breakpoints by default. When a software breakpoint is set to an instruction the code at this address is replaced by a special instruction e.g. TRAP, that stops the realtime execution and returns the control to the onchip debug system. **This method requires RAM at the break positions!** If you run your program out of RAM the number of software breakpoints is unlimited.

If your program does not run in RAM, refer to [Breakpoints in ROM, FLASH or EEPROM](#).

Back to the program. Doubleclick on the code line where you want to set a program breakpoint. All code lines to which a program breakpoint is set are marked with a small red bar.

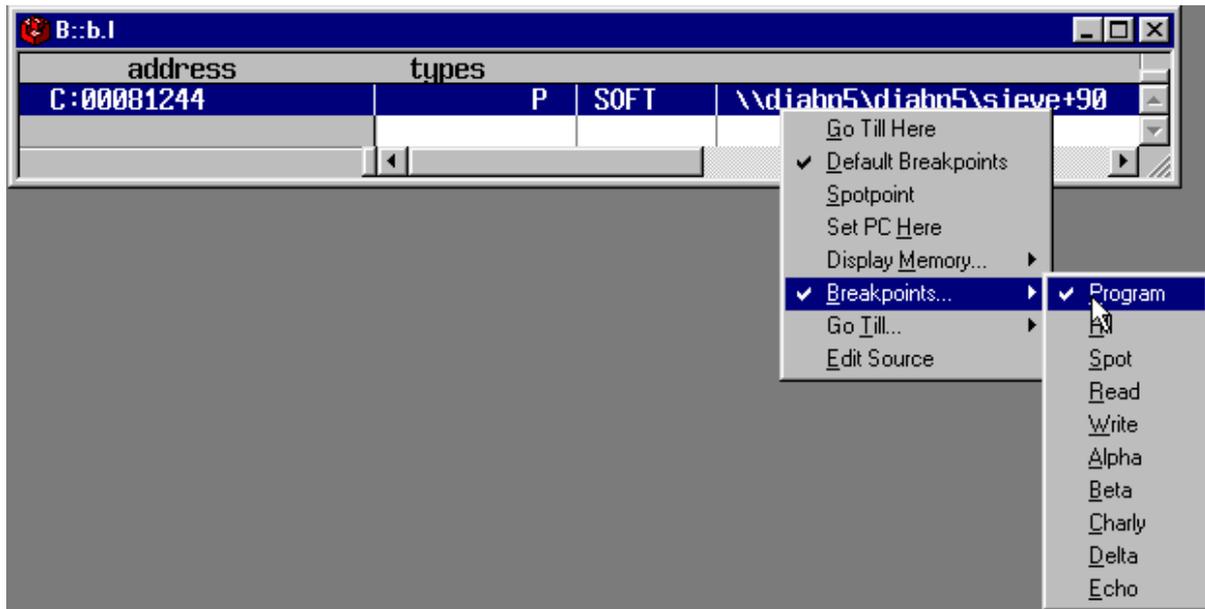


Use **List** from the **Breakpoint** menu to display the information about all set breakpoints.

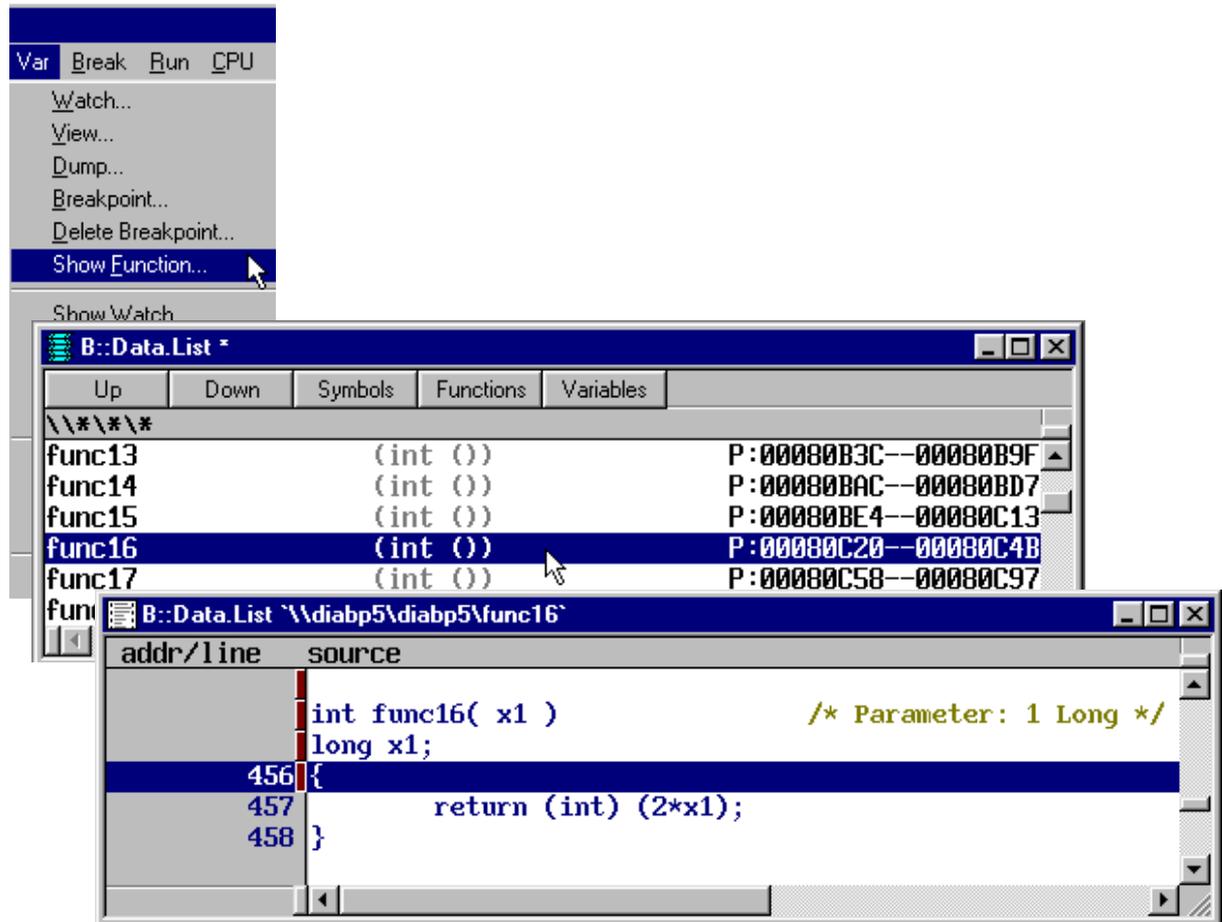


Start the program execution with **Go**. If the program does not reach your breakpoint, you can stop the program execution with **Break**.

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

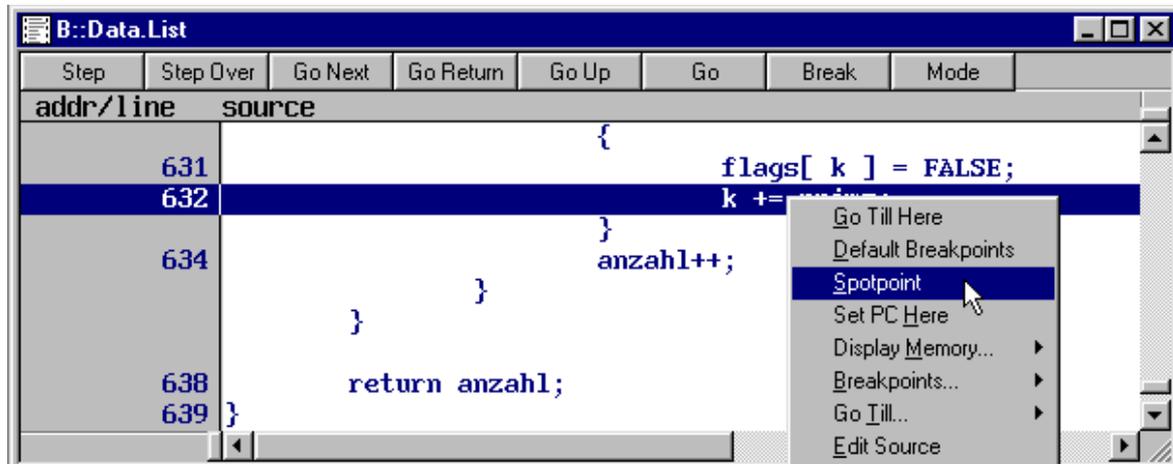


To set a program breakpoint to a code line, that is not displayed select **Show Function...** in the **Var** menu. Doubleclick to the function to display it and then set the breakpoint by a doubleclick to the code line.

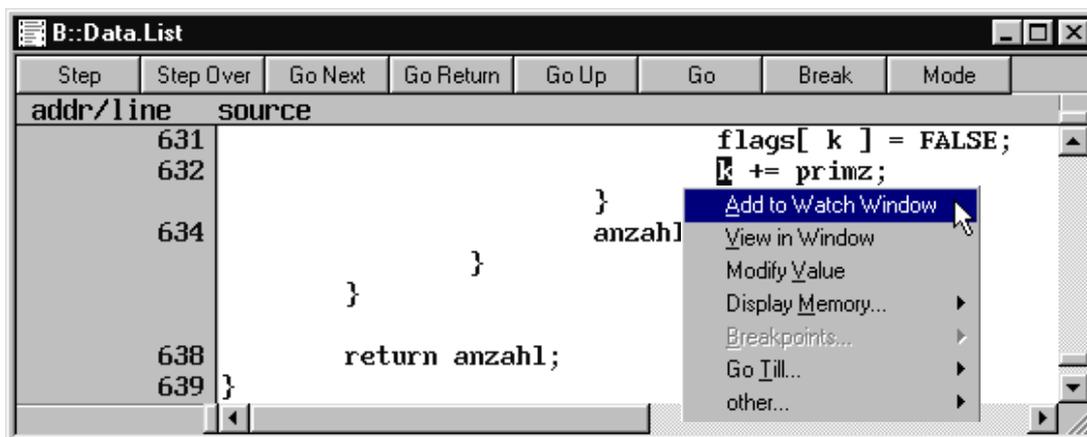


The second breakpoint type, that is available when software breakpoints are used, is a spot breakpoint. A spot breakpoint is a watchpoint, that stops the program execution for a short time to update all displayed information and then restarts the program execution.

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 k, 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 k.

Breakpoints in ROM, Flash, EEPROM

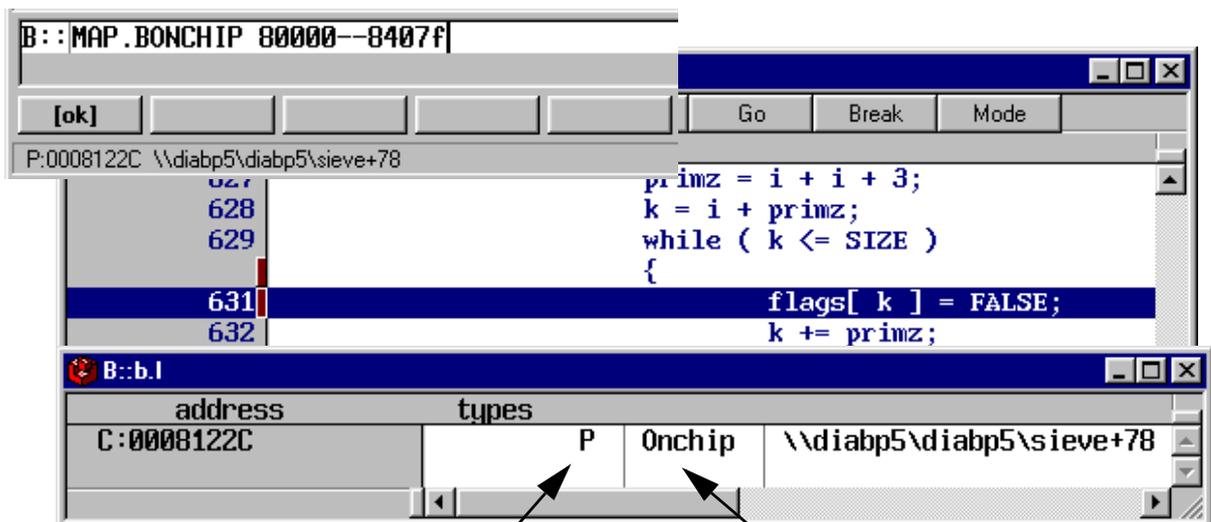
Most processor types (not 6833x and 6834x) provide a small number of onchip breakpoints. These breakpoints are used by TRACE32-ICD to set program or spot breakpoints even if the program doesn't run in RAM. For information on the available onchip breakpoints for your CPU refer to the [Onchip Breakpoints \(Overview\)](#)



Since the debugger uses software breakpoints by default, you must inform the debugger that the onchip breakpoints should be used!

MAP.BOnchip <address_range>

The command **MAP.BOnchip** indicates, that whenever a program or spot breakpoint is set within the specified address range, the debugger should use an onchip breakpoint.



Breakpoint type
Program breakpoint

Breakpoint is set as
onchip breakpoint

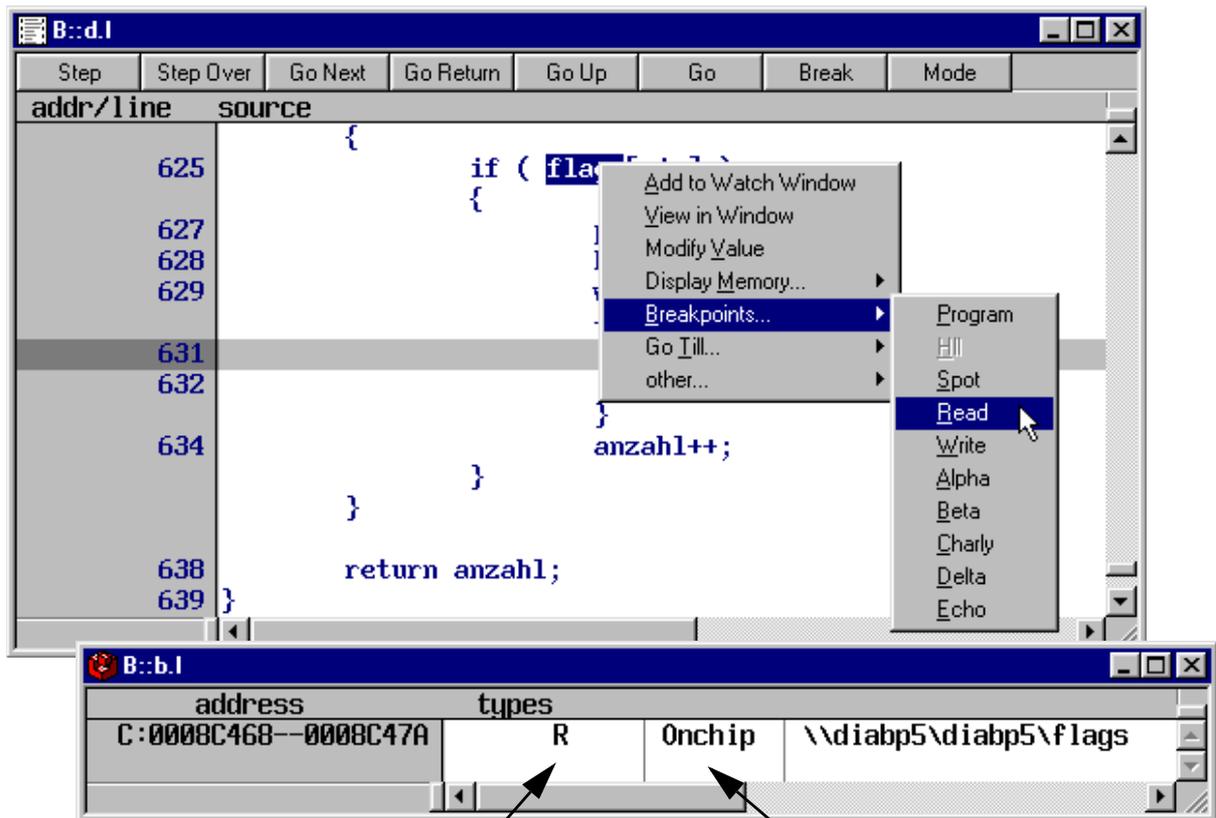
When the number of available onchip breakpoints is exceeded, the following error message is displayed:



Breakpoints on Data Accesses

For most CPUs the provided onchip breakpoints can also be used by TRACE32-ICD to stop the program execution when a read or write access occurs to a specific address location. For information on the available onchip breakpoints for your CPU refer to the [Onchip Breakpoint Overview](#).

To stop the program execution on a read access to a variable, select the variable with the cursor, press the write mouse button and select **Read** from the **Breakpoint...** pulldown.



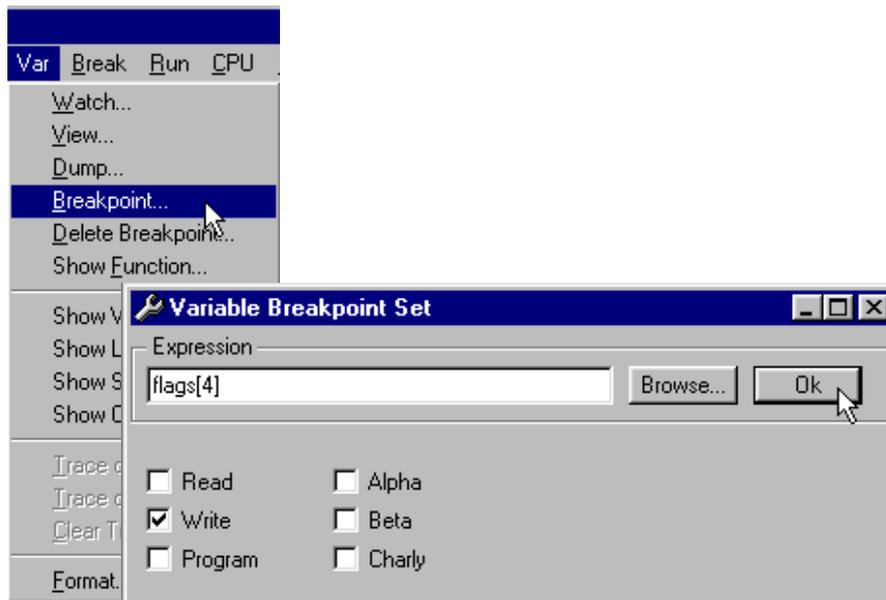
Breakpoint type
Read breakpoint

Breakpoint is set as
onchip breakpoint

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

To stop the program execution on a write access to a variable, you can also use **Breakpoints...** from the **Var** pulldown.

- **Browse** through the symbol data base to find the variable, select it by a double click, select **Write** in the **Variable Breakpoint Set** dialog box. Push **OK** to set the breakpoint.
- Or enter the variable name or the hll expression (e.g. to indicate only an element of an array) in the **Expression** field of the **Variable Breakpoint Set** dialog box. Select **Write** and push **OK** to set the breakpoint.



Breakpoint type
Write breakpoint

Breakpoint is set as
onchip breakpoint

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

Most CPU also provide more complex break- and trigger configurations. The support for these features by the TRACE32-ICD user interface varies for each CPU. For more information refer to the **ICD Target Manual**.

Onchip Breakpoints (Overview)

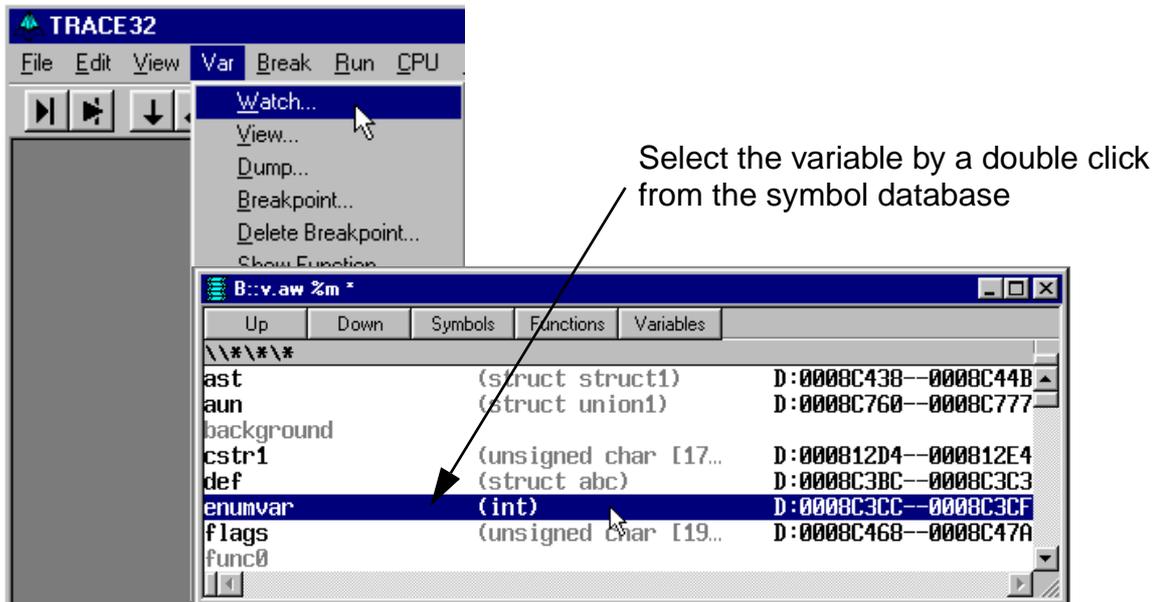
The following list give an overview of the usage of the onchip breakpoints by TRACE32-ICD:

- **CPU family**
- **Onchip breakpoints:** Total amount of available onchip breakpoints.
- **Instruction breakpoints:** Number of onchip breakpoints that can be used for program and spot breakpoints
- **Data breakpoints:** Number of onchip breakpoints that can be used as read or write breakpoints

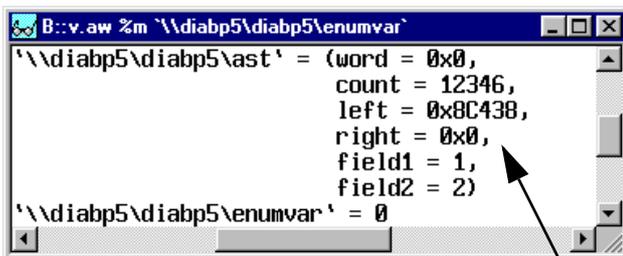
CPU family	Onchip Breakpoints	Instruction Breakpoints	Data Breakpoints
68k 6833x 6834x 68360	— — 1	— — 1	— — 1
68HC12 68HC12B/D all others	2 —	2 —	2 —
68HC16	—	—	—
ColdFire	2	1	1
PPC400			
MPC500/800	4 Instruction 2 Data	4	2
PPC600	1 (No onchip breakpoint, if software breakpoints are used)	1	—
PPC740/750	2 (Reduced to 1 if software breakpoints are used)	2/1	1
DSP56K 56k/56300 56100	2 1	2 1	2 1
ARM7	2 (Reduced to 1 if software breakpoints are used)	2/1	2/1

Display and Modify HLL Variables

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



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**.

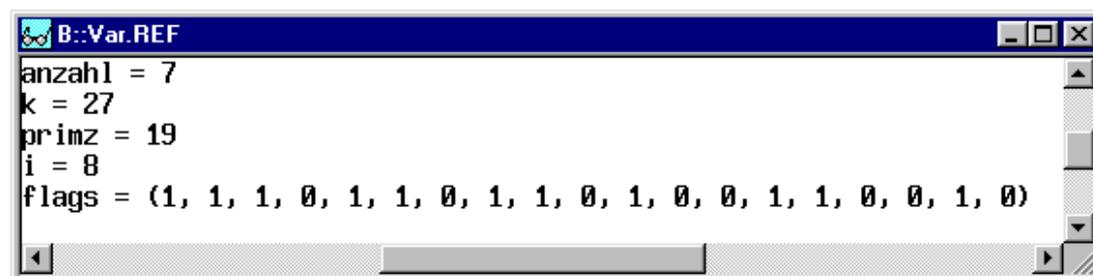
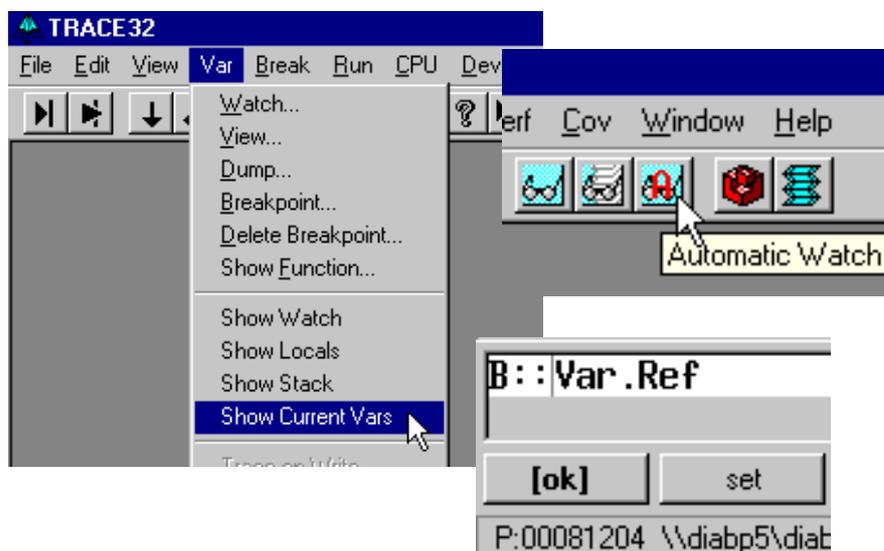
```

while ( k <= SIZE )
{
    flags[ k ] = FALSE;
    k +=
}
anzahl++;

```

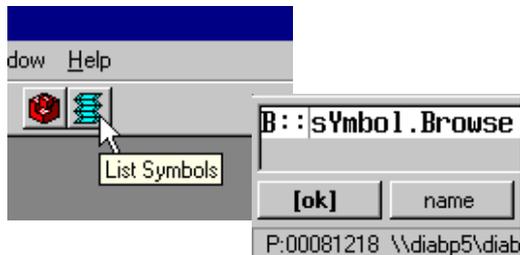
If you want to display a more complex structure or an array in a separate window use **View...** in the **Var** pulldown menu.

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 view single steps.

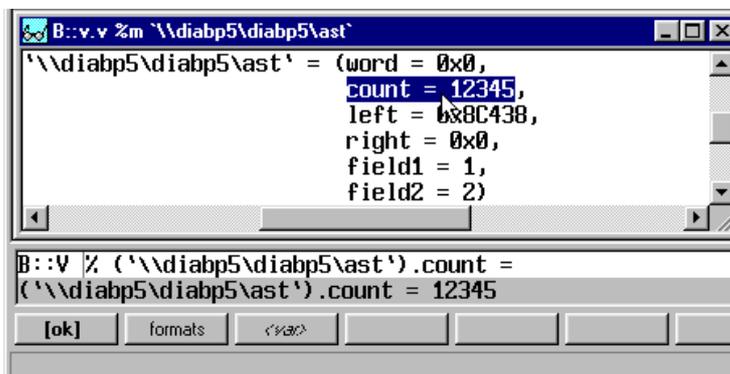


In most windows a context sensitive menu can be used with the right mouse button. If you select a variable you get access to the *Var* pulldown menu, that provides all features for displaying and modifying variables.

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.

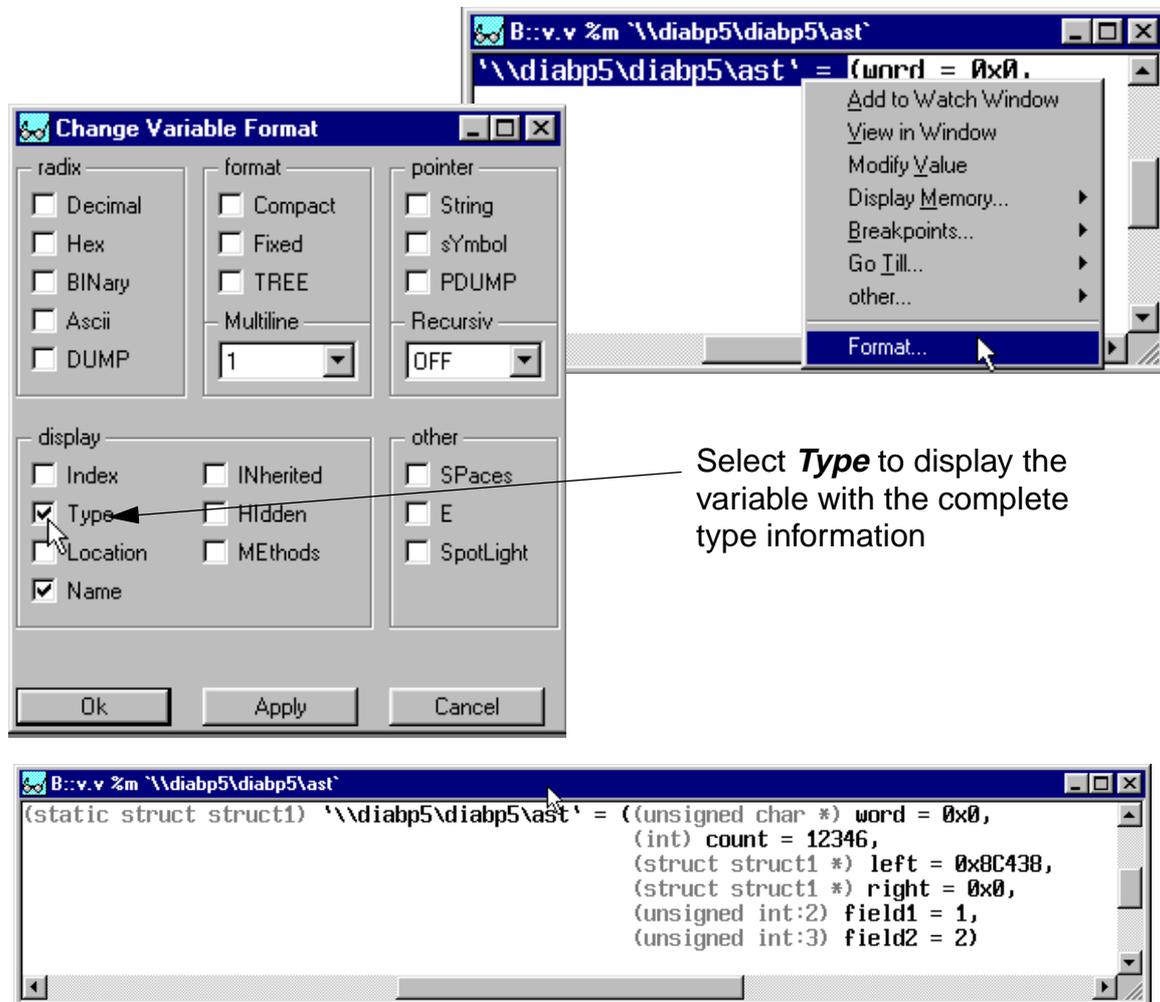


If you want to modify a variable value, double click to the value. The appropriate **Var.set** command will be displayed in the command line. Enter the new value and confirm with return.

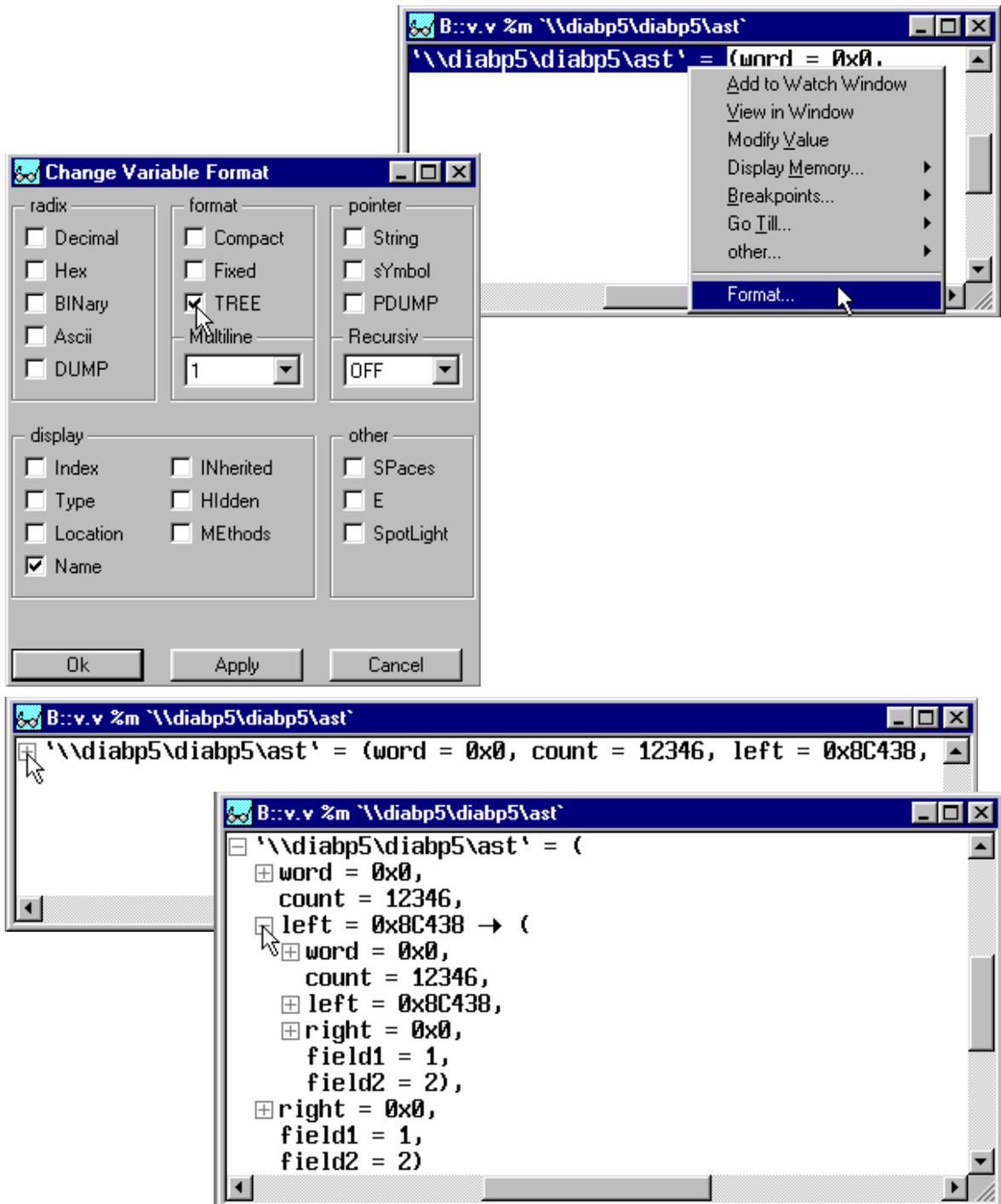


Format HLL-Variables

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

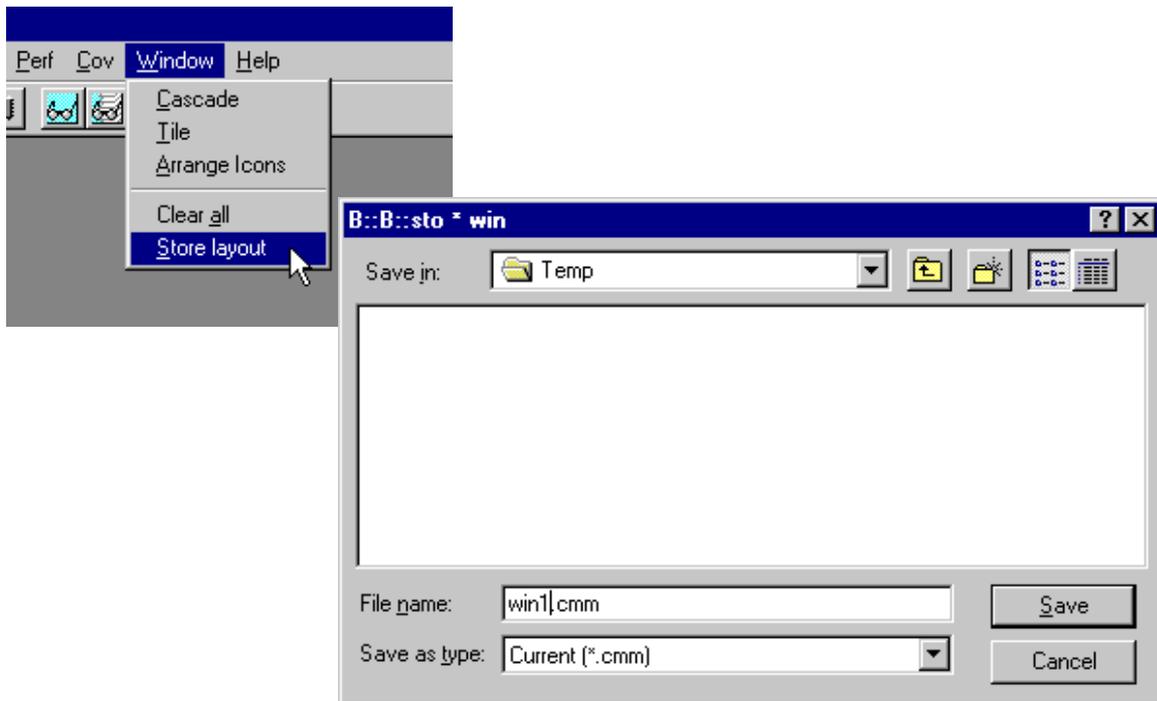


If you display more complex HLL structures, select **TREE** in the **Format** field of the **Change Variable Format** dialog box. This formatting allows to select the display for each member of the structure by clicking on + or -.

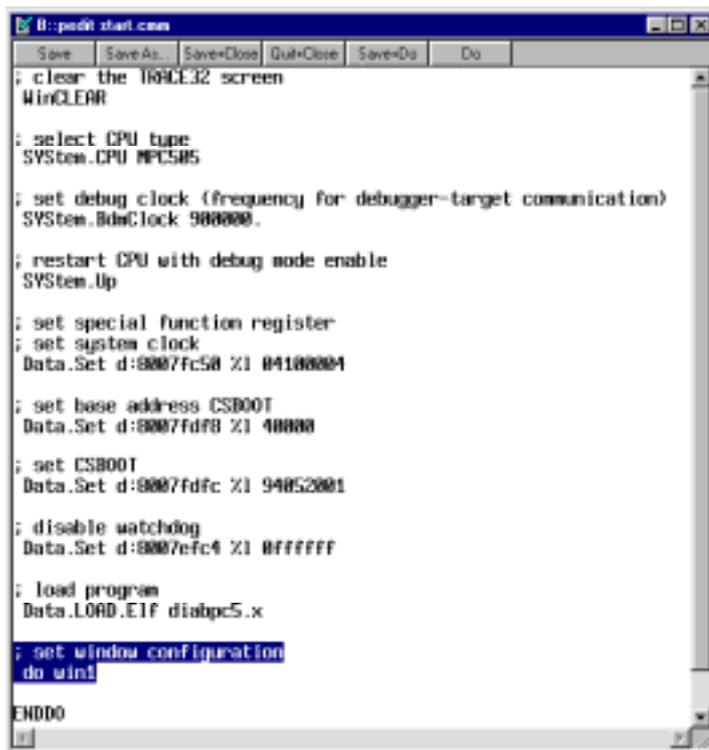


Exit TRACE32

To save the window configuration for your TRACE32-ICD use **Store layout** from the **Window** menu. Store layout generates a PRACTICE file, that includes all commands to reactivate your complete window configuration automatically. Enter a filename for the PRACTICE file into the **File name** field of the **Store** dialog box and push **Save** to generate the PRACTICE file.



Since PRACTICE support a modular program structure, you can enter a call for the automatic window configuration into your start-up file.



```
Save Save As... Save+Close Quit+Close Save+Do Do
; clear the TRACE32 screen
WinCLEAR

; select CPU type
SYSTEM.CPU MPC585

; set debug clock (frequency for debugger-target communication)
SYSTEM.DebugClock 900000.

; restart CPU with debug mode enable
SYSTEM.Up

; set special function register
; set system clock
Data.Set d:8007fc58 %I 04100004

; set base address CSB00T
Data.Set d:8007fd98 %I 40000

; set CSB00T
Data.Set d:8007fd9c %I 94052001

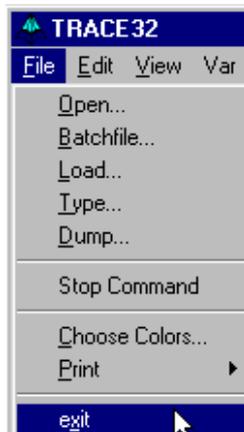
; disable watchdog
Data.Set d:8007efc4 %I 0FFFFFFF

; load program
Data.LOAD.EIF diatbpc5.x

; set window configuration
do win!

ENDDO
```

To exit from TRACE32-ICD by using **Exit** in the **File** menu.



	<p>Take care of the proper sequence on powering up/down: Power Up: debugger - target Power down: target - debugger</p>
---	---