LAUTERBACH
DEVELOPMENT TOOLS

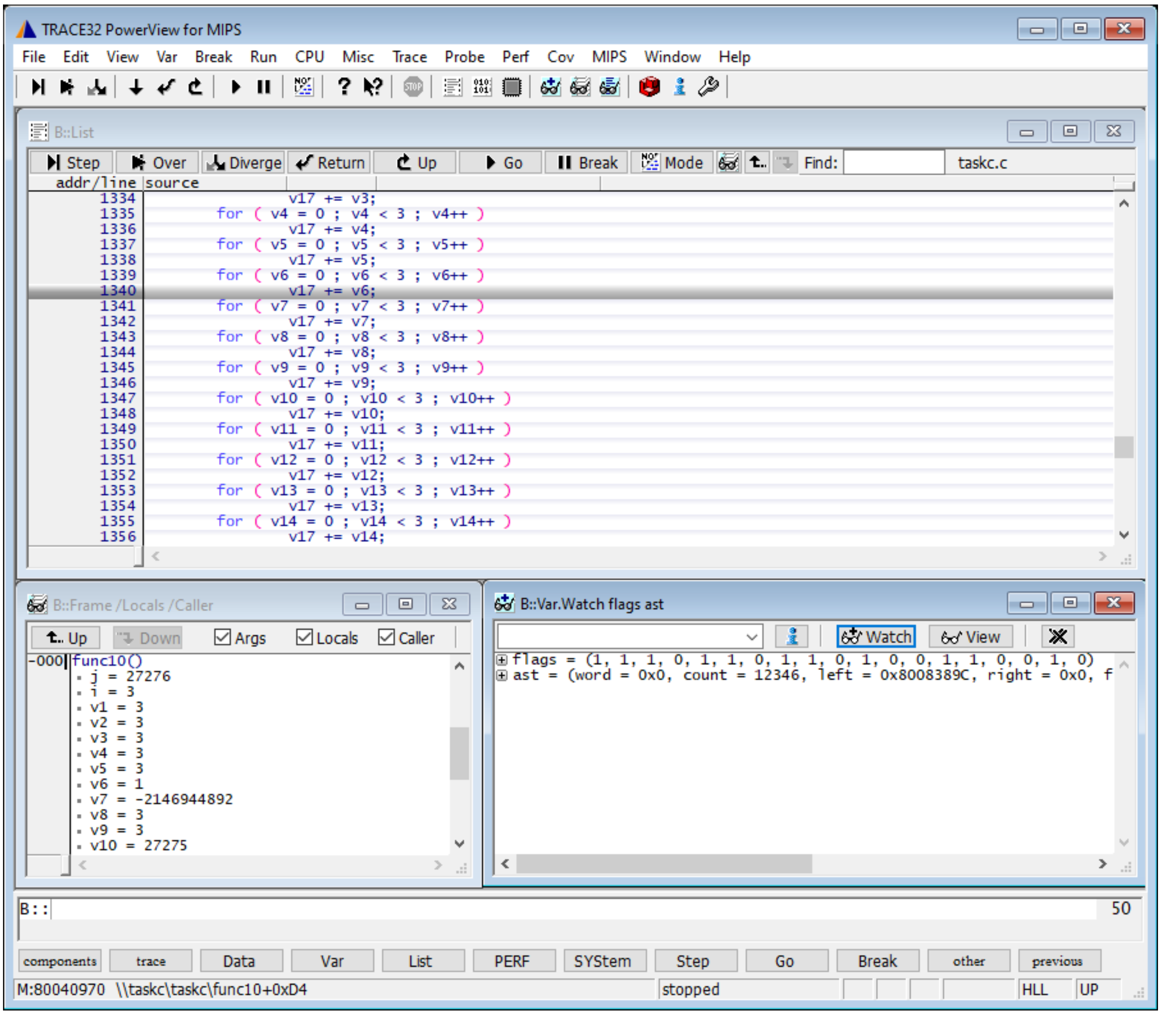
Simulator for MIPS

MANUAL

Simulator for MIPS

Simulator for MIPS

Version 06-Jun-2024


TRACE32 PowerView for MIPS
File Edit View Var Break Run CPU Misc Trace Probe Perf Cov MIPS Window Help
B::List
Step Over Diverge Return Up Go Break Mode Find: taskc.c
B::Frame /Locals /Caller
Up Down Args Locals Caller
B::Var.Watch flags ast
Watch View
components trace Data Var List PERF SYStem Step Go Break other previous

```
addr/line source
                v17 += v3;
          for ( v4 = 0 ; v4 < 3 ; v4++ )
                v17 += v4;
          for ( v5 = 0 ; v5 < 3 ; v5++ )
                v17 += v5;
          for ( v6 = 0 ; v6 < 3 ; v6++ )
                v17 += v6;
          for ( v7 = 0 ; v7 < 3 ; v7++ )
                v17 += v7;
          for ( v8 = 0 ; v8 < 3 ; v8++ )
                v17 += v8;
          for ( v9 = 0 ; v9 < 3 ; v9++ )
                v17 += v9;
          for ( v10 = 0 ; v10 < 3 ; v10++ )
                v17 += v10;
          for ( v11 = 0 ; v11 < 3 ; v11++ )
                v17 += v11;
          for ( v12 = 0 ; v12 < 3 ; v12++ )
                v17 += v12;
          for ( v13 = 0 ; v13 < 3 ; v13++ )
                v17 += v13;
          for ( v14 = 0 ; v14 < 3 ; v14++ )
                v17 += v14;
```

```
-000|func10()
    | j = 27276
    | i = 3
    | v1 = 3
    | v2 = 3
    | v3 = 3
    | v4 = 3
    | v5 = 3
    | v6 = 1
    | v7 = -2146944892
    | v8 = 3
    | v9 = 3
    | v10 = 27275
```

```
⊞ flags = (1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0)
⊞ ast = (word = 0x0, count = 12346, left = 0x8008389C, right = 0x0, f
```

```
B::                                                            50
M:80040970  \\taskc\taskc\func10+0xD4        stopped        HLL  UP
```

All general commands are described in the **“PowerView Command Reference”** (ide_ref.pdf) and **“General Commands Reference”**.

TRACE32 Simulator License

[build 68859 - DVD 02/2016]

The extensive use of the TRACE32 Instruction Set Simulator requires a *TRACE32 Simulator License*.

For more information, see **www.lauterbach.com/sim_license.html**.

Quick Start of the Simulator

To start the simulator, proceed as follows:

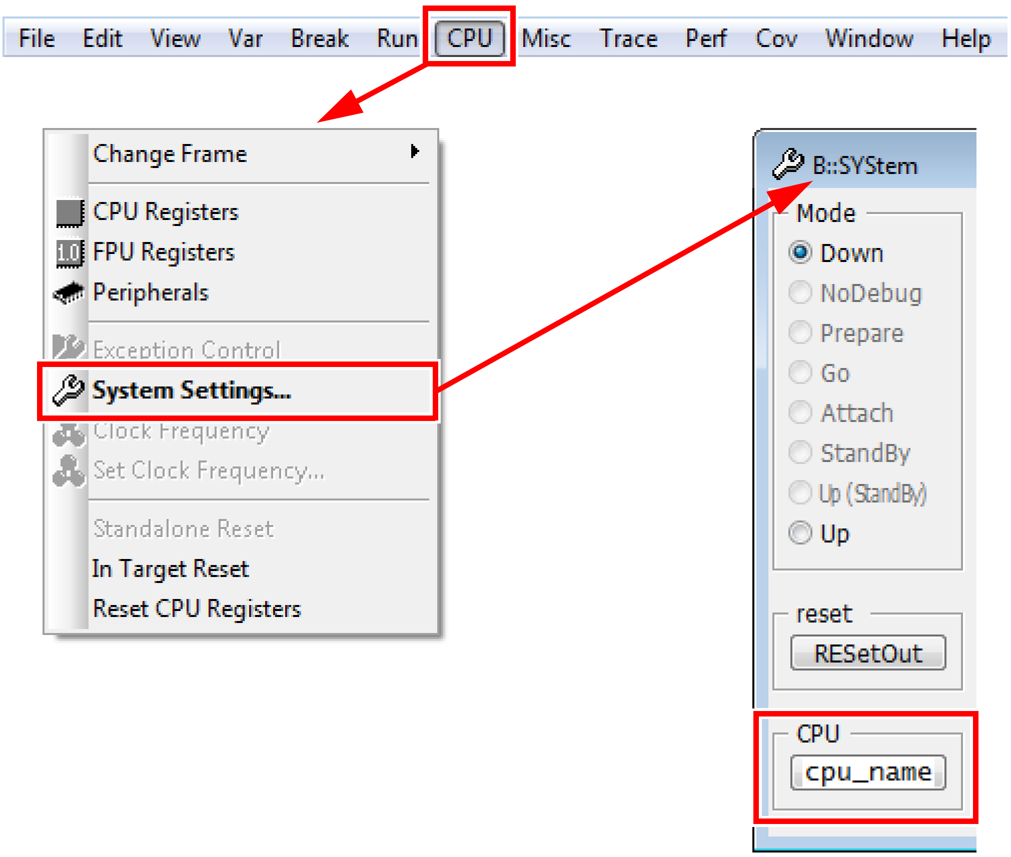
1. Select the device prompt for the Simulator and reset the system.

```
B::

RESet
```

The device prompt `B::` is normally already selected in the TRACE32 command line. If this is not the case, enter `B::` to set the correct device prompt. The **RESet** command is only necessary if you do not start directly after booting TRACE32.

2. Specify the CPU specific settings.



```
SYStem.CPU <cpu_name>
```

The default values of all other options are set in such a way that it should be possible to work without modification. Please consider that this is probably not the best configuration for your target.

3. Enter debug mode.

```
SYStem.Up
```

This command resets the CPU and enters debug mode. After this command is executed it is possible to access memory and registers.

4. Load the program.

```
Data.LOAD.<file_format> <file>      ; load program and symbols
```

See the **Data.LOAD** command reference for a list of supported file formats. If uncertain about the required format, try **Data.LOAD.auto**.

A detailed description of the **Data.LOAD** command and all available options is given in the reference guide.

5. Start-up example

A typical start sequence is shown below. This sequence can be written to a PRACTICE script file (*.cmm, ASCII format) and executed with the command **DO** *<file>*.

```
B::                                 ; Select the ICD device prompt

WinCLEAR                            ; Clear all windows

SYStem.CPU <cpu_name>               ; Select CPU type

SYStem.Up                           ; Reset the target and enter
                                    ; debug mode

Data.LOAD.<file_format> <file>      ; Load the application

Register.Set pc main                ; Set the PC to function main

PER.view                            ; Show clearly arranged
                                    ; peripherals in window      *)

List.Mix                            ; Open source code window    *)

Register.view /SpotLight            ; Open register window       *)

Frame.view /Locals /Caller          ; Open the stack frame with
                                    ; local variables            *)

Var.Watch %Spotlight flags ast      ; Open watch window for
                                    ; variables                  *)
```

*) These commands open windows on the screen. The window position can be specified with the **WinPOS** command.

Peripheral Simulation

For more information, see **“API for TRACE32 Instruction Set Simulator”** (simulator_api.pdf).

Troubleshooting

No information available.

FAQ

Please refer to https://support.lauterbach.com/kb.

Memory Classes

The following MIPS specific memory classes are available.

Memory Class	Description
P	Program Memory
D	Data Memory
CP0	Coprocessor 0 Register
CP1	Coprocessor 1 Register (if implemented)
CP2	Coprocessor 2 Register (if implemented)
CP3	Coprocessor 3 Register (if implemented)
DBG	Debug Memory Class (gives additional information)
E	Emulation Memory, Pseudo Dualport Access to Memory (see **SYStem.CpuAccess**)
VM	Virtual Memory (memory on the debug system)

To access a memory class, write the class in front of the address.

Examples:

’Data.dump CP0:0--3’ displays the register 0 (Index), 1 (Random), 2 (EntryLo0), 3 (EntryLo1) of the System Control Coprocessor (=CP0).

The register number can have values between 0 and 31. The value of “select” must be multiplied by 32 and added to the register number. “Data.dump CP0:0x30--0x30” displays the Config1 register (register number: 0x10; select: 0x01). Select is 0 for the registers mentioned above.

ICD-MIPS64: For the memory classes CPx and DBG are only 64-bit (QUAD) write accesses possible.

Belated Trace Analysis

The following commands are required for a belated trace analysis:

TCB.Version *<number>*	Inform the TRACE32 Instruction Set Simulator which trace cell version was used to record the loaded trace information.
TCB.SourceSizeBits *<number>*	Inform the TRACE32 Instruction Set Simulator how much bits were used in the loaded trace information to identify the source core.
TCB.ThreadSizeBits *<number>*	Inform the TRACE32 Instruction Set Simulator how much bits were used in the loaded trace information to identify the source thread context.
TCB.InsCompSizeBits *<number>*	Inform the TRACE32 Instruction Set Simulator how much bits were used for instruction completion information.
TCB.FCR ON | OFF	Inform the TRACE32 Instruction Set Simulator about existence of optional function call - return bit.
TCB.IM ON | OFF	Inform the TRACE32 Instruction Set Simulator about existence of optional Instruction cache miss bit.
TCB.LSM ON | OFF	Inform the TRACE32 Instruction Set Simulator about existence of optional data cache load store miss bit.
TCB.Type *<number>*	Inform the TRACE32 Instruction Set Simulator on the used Trace Control Block Type.

MIPS specific SYStem Commands

SYStem.CONFIG — Configure debugger according to target topology

The **SYStem.CONFIG** commands have no effect on the simulator. They are only provided to allow the user to run PRACTICE scripts written for the debugger within the simulator without modifications.

SYStem.CPU — Select the used CPU

Format:	**SYStem.CPU** *<cpu>*
<cpu>:	**AUTO** **MIPS4K** **RC32334** **F731940** **MIPS5K**

SYStem.LOCK — Lock and tristate the debug port

Format:	**SYStem.LOCK** [**ON** | **OFF**]

Default: OFF.

If the system is locked, no access to the debug port will be performed by the debugger. While locked, the debug connector of the debugger is tristated. The main intention of the **SYStem.LOCK** command is to give debug access to another tool. The command has no effect for the simulator.

SYStem.MemAccess — Select run-time memory access method

Format:	**SYStem.MemAccess Enable | StopAndGo | Denied** **SYStem.ACCESS** (deprecated)

Enable **CPU** (deprecated)	Memory access during program execution to target is enabled.
Denied	Memory access during program execution to target is disabled.
StopAndGo	Temporarily halts the core(s) to perform the memory access. Each stop takes some time depending on the speed of the JTAG port, the number of the assigned cores, and the operations that should be performed.

SYStem.Option.OVERLAY — Enable overlay support

Format:	**SYStem.Option.OVERLAY** [**ON** | **OFF** | **WithOVS**]

Default: OFF.

ON	Activates the overlay extension and extends the address scheme of the debugger with a 16 bit virtual overlay ID. Addresses therefore have the format *<overlay_id>***:***<address>*. This enables the debugger to handle overlaid program memory.
OFF	Disables support for code overlays.
WithOVS	Like option **ON**, but also enables support for software breakpoints. This means that TRACE32 writes software breakpoint opcodes to both, the *execution area* (for active overlays) and the *storage area*. This way, it is possible to set breakpoints into inactive overlays. Upon activation of the overlay, the target's runtime mechanisms copies the breakpoint opcodes to the execution area. For using this option, the storage area must be readable and writable for the debugger.

Example:

```
SYStem.Option.OVERLAY ON
Data.List 0x2:0x11c4                   ; Data.List <overlay_id>:<address>
```

SYStem.Option.MMUSPACES Separate address spaces by space IDs

Format:	**SYStem.Option.MMUSPACES** [**ON** | **OFF**] **SYStem.Option.MMUspaces** [**ON** | **OFF**] (deprecated) **SYStem.Option.MMU** [**ON** | **OFF**] (deprecated)

Default: OFF.

Enables the use of space IDs for logical addresses to support **multiple** address spaces.

For an explanation of the TRACE32 concept of address spaces (zone spaces, MMU spaces, and machine spaces), see **"TRACE32 Concepts"** (trace32_concepts.pdf).

NOTE:

SYStem.Option.MMUSPACES should not be set to **ON** if only one translation table is used on the target.

If a debug session requires space IDs, you must observe the following sequence of steps:

1. Activate **SYStem.Option.MMUSPACES**.

2. Load the symbols with **Data.LOAD**.

Otherwise, the internal symbol database of TRACE32 may become inconsistent.

Examples:

```
;Dump logical address 0xC00208A belonging to memory space with
;space ID 0x012A:
Data.dump D:0x012A:0xC00208A

;Dump logical address 0xC00208A belonging to memory space with
;space ID 0x0203:
Data.dump D:0x0203:0xC00208A
```

SYStem.Mode Establish the communication with the target

Format:	**SYStem.Mode** <*mode*> **SYStem.Down** **(alias for SYStem.Mode Down)** **SYStem.Up** **(alias for SYStem.Mode Up)**
<*mode*>:	**Down** **Up**

Down	(Disables the debugger and keeps the CPU in reset. (default)
Up	Resets the target and sets the CPU to debug mode. After the execution of this command the CPU is stopped and all registers are set to the default level.

SYStem.Option.Address32 Use 32-bit addresses

Format:	**SYStem.Option.Address32** [**ON** | **OFF**]

Default: OFF.

This option is functionable for 64bit architectures only, not for 32bit architectures.

Enable Address32 if you want to work with 32bit addresses on a 64bit MIPS CPU. If enabled, TRACE32 accepts and displays only 32bit addresses. Internally, they are sign-extended to 64bit addresses before they are used on the CPU. This results in a mapping as follows:

Address used in TRACE32	**Mapped to address on 64bit CPU**
`0x0000 0000 - 0x7FFF FFFF`	`0x0000 0000 0000 0000 - 0x0000 0000 7FFF FFFF`
`0x8000 0000 - 0xFFFF FFFF`	`0xFFFF FFFF 8000 0000 - 0xFFFF FFFF FFFF FFFF`

As a result, with Address32 ON, only the 32bit Compatibility Address Spaces 0x0000 0000 0000 0000 - 0x0000 0000 7FFF FFFF and 0xFFFF FFFF 8000 0000 - 0xFFFF FFFF FFFF FFFF can be accessed.
This option is helpful if you debug a 32bit Linux kernel on a 64bit MIPS CPU.
Careful: if 64bit addresses are used in TRACE32 with Address32 ON, bits 32-63 will truncated. Turn this option off if you need to access real 64bit addresses.

SYStem.Option.DisMode

Define disassembler mode

Format:	**SYStem.Option.DisMode** <*mode*>
<*mode*>:	**AUTO** **ACCESS** **MIPS32** **MIPS16** **MICROMIPS** **NANOMIPS** **MIPSR6**

This command specifies the selected disassembler.

Default: AUTO.

AUTO	Automatic selection of disassembler mode. The information provided by the compiler output format is used for the disassembler selection. If no information is available it has the same behavior as ACCESS. (default)
ACCESS	Disassembler mode will be selected by entered access class.
MIPS32	The MIPS32 disassembler is used.
MIPS16	The MIPS16 disassembler is used.
MICROMIPS	The microMIPS disassembler is used.
NANOMIPS	The nanoMIPS disassembler is used.
MIPSR6	The MIPS R6 disassembler is used.

SYStem.Option.Endianness Define endianness of target memory

Format: **SYStem.Option.Endianness** [**AUTO** | **Little** | **Big**]

Default: AUTO.

This option selects the byte ordering mechanism. If it is set to AUTO, the kernel mode endianness will be detected and selected.

SYStem.Option.IMASKASM Disable interrupts while ASM single stepping

Format: **SYStem.Option.IMASKASM** [**ON** | **OFF**]

Default: OFF.

If enabled, the interrupt mask bits of the CPU will be set during assembler single-step operations. The interrupt routine is not executed during single-step operations. After single step the interrupt mask bits are restored to the value before the step.

SYStem.Option.IMASKHLL Disable interrupts while HLL single stepping

Format: **SYStem.Option.IMASKHLL** [**ON** | **OFF**]

Default: OFF.

If enabled, the interrupt mask bits of the cpu will be set during HLL single-step operations. The interrupt routine is not executed during single-step operations. After single step the interrupt mask bits are restored to the value before the step.

SYStem.RESetOut CPU reset command

The command asserts nRESET on the JTAG connector in the TRACE32 In-Circuit Debugger (ICD) but is ignored by the TRACE32 Instruction Set Simulator. However, the command is allowed in the simulator so that you can run scripts which have actually been made for the debugger. For more information about the effect in the debugger, refer to your **Processor Architecture Manual** (debugger_<*arch*>.pdf).

MMU.DUMP

Page wise display of MMU translation table

Format:	**MMU.DUMP** *<table>* [*<range>* | *<address>* | *<range>* *<root>* | *<address>* *<root>*] **MMU.***<table>***.dump** (deprecated)
<table>:	**PageTable** **KernelPageTable** **TaskPageTable** *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0** *<cpu_specific_tables>*

Displays the contents of the CPU specific MMU translation table.

- If called without parameters, the complete table will be displayed.
- If the command is called with either an address range or an explicit address, table entries will only be displayed if their **logical** address matches with the given parameter.

<root>	The *<root>* argument can be used to specify a page table base address deviating from the default page table base address. This allows to display a page table located anywhere in memory.
<range> *<address>*	Limit the address range displayed to either an address range or to addresses larger or equal to *<address>*. For most table types, the arguments *<range>* or *<address>* can also be used to select the translation table of a specific process if a space ID is given.
PageTable	Displays the entries of an MMU translation table. • if *<range>* or *<address>* have a space ID: displays the translation table of the specified process • else, this command displays the table the CPU currently uses for MMU translation.

KernelPageTable	Displays the MMU translation table of the kernel. If specified with the **MMU.FORMAT** command, this command reads the MMU translation table of the kernel and displays its table entries.
TaskPageTable *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**	Displays the MMU translation table entries of the given process. Specify one of the **TaskPageTable** arguments to choose the process you want. In MMU-based operating systems, each process uses its own MMU translation table. This command reads the table of the specified process, and displays its table entries. • For information about the first three parameters, see **"What to know about the Task Parameters"** (general_ref_t.pdf). • See also the appropriate **OS Awareness Manuals**.

CPU specific tables for MMU.DUMP

TLB	Displays the contents of the Translation Lookaside Buffer.

MMU.FORMAT Define MMU table structure

[Examples]

Format 1: **MMU.FORMAT** *<format>* [*<base_address>* [*<logical_kernel_address_range>* *<physical_kernel_address>*]]

Format 2: **MMU.FORMAT** *<format>* [*<base_address>* [*<base_address_highrange>* [*<logical_kernel_address_range>* *<physical_kernel_address>*]]]
MIPS64 only

Default *<format>*: STD.

Defines the information needed for the page table walks, which are performed by TRACE32 for debugger address translation, page table dumps, or page table scans.

Format 1 is the normal, CPU-architecture independent command syntax. This format does *not* require the additional input parameter *<base_address_highrange>* of format 2.

Format 2: For MIPS64, there are four **MMU.FORMAT** *<format>* keywords which require the additional input parameter *<base_address_highrange>*. These keywords are **LINUX64**, **LINUX64RIXI**, **LINUX64HTLB**, and **LINUX64HTLBP16**.

<format>

<format> is to be replaced with a CPU architecture specific keyword which defines the structure of the MMU page tables used by the kernel. By default, TRACE32 assumes that the MMU format is **STD**, unless *you* specify the **MMU.FORMAT** *<format>* explicitly.

The table below indicates if a *<format>* requires the additional parameter *<base_address_highrange>*.

<format>	Description
EXTENSION	Table walk performed by a TRACE32 extension that a) was developed by the customer and b) defines table walk callback functions.
LINUX32	Linux 32-bit, page size 4kB
LINUX32P16	Linux 32-bit, page size 16kB
LINUX32P16R2	Linux 32-bit, page size 16kB, used on MIPS32 R2 or R6 (internally identical to format LINUX32P16R41)
LINUX32P16R2	Deprecated: internally identical to format LINUX32P16R41
LINUX32R4K	Linux 32-bit, page size 4kB, like LINUX32 but different page flags
LINUX32RIXI	Linux 32-bit with RI/XI bits
LINUX64	Linux 64-bit with 64-bit PTEs, page size 4kB. Separate page table for high address range can be specified with optional extra parameter *<base_address_highrange>*.
LINUX64HTLB	Linux 64-bit with 64-bit PTEs, page size 4kB for huge TLB. Uses separate sub table for addresses > 0xFFFFFFFFC0000000.
LINUX64HTLBP16	Linux 64-bit like LINUX64HTLB but pag esize 16kB.
LINUX64P16	Linux 64-bit with 64-bit PTEs, page size 16kB. Depth 3 levels.
LINUX64P64	Linux 64-bit with 64-bit PTEs, page size 64kB. Depth 3 levels.
LINUX64P64LT	Linux 64-bit with 64-bit PTEs, page size 64kB. Depth 2 levels with large level 1 table (used for BROADCOM(R) XLP SDK 3.7.10 and alike)
LINUX64RIXI	Linux 64-bit with 64-bit PTEs with RI/XI bits, page size 4kB. Separate page table for high address range can be specified with optional extra parameter *<base_address_highrange>*.
LINUXBIG	Linux 32-bit with 64-bit PTEs on MIPS32
LINUXBIG64	Linux 32-bit with 64-bit PTEs on MIPS64
STD	Standard format defined by the CPU
WINCE6	Format used by Windows CE6

<base_address>

<base_address> defines the start address of the default page table which is usually the kernel page table. The kernel page table contains translations for mapped address ranges owned by the kernel.

The debugger address translation uses the default page table if no process specific page table (task page table) is available to translate an address.

<base_address> can be left empty by typing a comma or set to zero if there is no default page table available in the system.

<base_address_highrange>

Using *<base_address_highrange>*, you can specify a second page table responsible for the translation of addresses >= 0xFFFFFFFF00000000. Then, two page tables are in use:

- Addresses in range 0x0--0xFFFFFFFEFFFFFFFF will be translated with the page table defined by the argument *<base_address>*.
- Addresses in range 0xFFFFFFFF00000000--0xFFFFFFFFFFFFFFFF will be translated with the page table defined by the argument *<base_address_highrange>*.

<logical_kernel_address_range> and <physical_kernel_address> for the Default Translation

The arguments *<logical_kernel_address_range>* and *<physical_kernel_address>* define a linear logical-to-physical address translation for the kernel addresses, called *kernel translation* or *default translation*. This translation should cover all statically mapped logical address ranges of kernel code or kernel data.

For the *<physical_kernel_address>* you just need to specify the start address.

NOTE:	If no kernel translation is specified for a given memory access, TRACE32 tries to use static address translations defined by the command **TRANSlation.Create**. The kernel translation is shown in the **TRANSlation.List** window.

Examples

NOTE:	A backslash \ is used as a line continuation character in PRACTICE script files (*.cmm). No white space permitted after the backslash.

Examples of Format 1:

```
;           <format>   <base_address>    <logical_range>      <phys_range>
MMU.FORMAT LINUX       swapper_pg_dir

MMU.FORMAT LINUX       swapper_pg_dir \
                        0xC000000000000000--0xc00000007FFFFFFF 0x20000000
```

Examples of Format 2 with <base_address_highrange>:

```
;           <format>    <base_address>  <base_address_highrange>
MMU.FORMAT LINUX64     swapper_pg_dir    module_pg_dir

MMU.FORMAT LINUX64     swapper_pg_dir    module_pg_dir \
                         0xC000000000000000--0xc00000007FFFFFFF 0x20000000
;                                   <logical_range>              <phys_range>
```

Examples of Format 2 without <base_address_highrange>:

In this example, not only the *<base_address_highrange>* is omitted but also all remaining parameters.

```
;           <format>    <base_address>  <base_address_highrange>
MMU.FORMAT LINUX64     swapper_pg_dir
```

If you need all parameters of Format 2 except for *<base_address_highrange>*, then use two commas to specify an empty input parameter.

```
;           <format>    <base_address>  <base_address_highrange>
MMU.FORMAT LINUX64     swapper_pg_dir


MMU.FORMAT LINUX64     swapper_pg_dir              ,, \
                        0xC000000000000000--0xC00000007FFFFFFF 0x20000000
;                                   <logical_range>              <phys_range>
```

Format:	**MMU.List** *<table>* [*<range>* | *<address>* | *<range>* *<root>* | *<address>* *<root>*] **MMU.***<table>***.List** (deprecated)
<table>:	**PageTable** **KernelPageTable** **TaskPageTable** *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**

Lists the address translation of the CPU-specific MMU table.

- If called without address or range parameters, the complete table will be displayed.
- If called without a table specifier, this command shows the debugger-internal translation table. See **TRANSlation.List**.
- If the command is called with either an address range or an explicit address, table entries will only be displayed if their **logical** address matches with the given parameter.

<root>	The *<root>* argument can be used to specify a page table base address deviating from the default page table base address. This allows to display a page table located anywhere in memory.
PageTable	Lists the current MMU translation of the CPU. This command reads all tables the CPU currently uses for MMU translation and lists the address translation.
KernelPageTable	Lists the MMU translation table of the kernel. If specified with the **MMU.FORMAT** command, this command reads the MMU translation table of the kernel and lists its address translation.
TaskPageTable *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**	Lists the MMU translation of the given process. Specify one of the **TaskPageTable** arguments to choose the process you want. In MMU-based operating systems, each process uses its own MMU translation table. This command reads the table of the specified process, and lists its address translation. • For information about the first three parameters, see **“What to know about the Task Parameters”** (general_ref_t.pdf). • See also the appropriate **OS Awareness Manuals**.

Format:	**MMU.SCAN** *<table>* [*<range>* *<address>*] **MMU.***<table>***.SCAN** (deprecated)
<table>:	**PageTable** **KernelPageTable** **TaskPageTable** *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0** **ALL** *<cpu_specific_tables>*

Loads the CPU-specific MMU translation table from the CPU to the debugger-internal static translation table.

- If called without parameters, the complete page table will be loaded. The list of static address translations can be viewed with **TRANSlation.List**.
- If the command is called with either an address range or an explicit address, page table entries will only be loaded if their **logical** address matches with the given parameter.

Use this command to make the translation information available for the debugger even when the program execution is running and the debugger has no access to the page tables and TLBs. This is required for the real-time memory access. Use the command **TRANSlation.ON** to enable the debugger-internal MMU table.

PageTable	Loads the entries of an MMU translation table and copies the address translation into the debugger-internal static translation table. • if *<range>* or *<address>* have a space ID: loads the translation table of the specified process • else, this command loads the table the CPU currently uses for MMU translation.
KernelPageTable	Loads the MMU translation table of the kernel. If specified with the **MMU.FORMAT** command, this command reads the table of the kernel and copies its address translation into the debugger-internal static translation table.

TaskPageTable *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>*:**0x0**	Loads the MMU address translation of the given process. Specify one of the **TaskPageTable** arguments to choose the process you want. In MMU-based operating systems, each process uses its own MMU translation table. This command reads the table of the specified process, and copies its address translation into the debugger-internal static translation table. • For information about the first three parameters, see **"What to know about the Task Parameters"** (general_ref_t.pdf). • See also the appropriate **OS Awareness Manual**.
ALL	Loads all known MMU address translations. This command reads the OS kernel MMU table and the MMU tables of all processes and copies the complete address translation into the debugger-internal static translation table. See also the appropriate **OS Awareness Manual**.

CPU specific Tables in MMU.SCAN <table>

TLB	Loads the translation table from the CPU to the debugger-internal translation table.

TrOnchip Commands

TrOnchip.state — Display on-chip trigger window

Format:	**TrOnchip.state**

Opens the **TrOnchip.state** window.

TrOnchip.RESet — Set on-chip trigger to default state

Format:	**TrOnchip.RESet**

Sets the TrOnchip settings and trigger module to the default settings.