# [RISC-V] [HW-Designer] Instruction/data cache handling in TRACE32

2026-10-07 - Comments (0) - RISC-V

# Instruction- and Data-Caches in RISC-V

Unfortunately, the RISC-V debug specification does not define an explicit, standardized way to handle instruction- or data caches. Nevertheless, in some scenarios a debugger and/or the debug IP of the device under test need to be able to trigger certain cache operations, such as cache flushes.

**Example #1:**

Let's assume an SoC with a single RISC-V core with instruction cache. The core's instruction memory contains a RISC-V instruction 'A'. The RISC-V core gets halted some instructions before the execution of 'A'. The core has not executed 'A' yet, however it has already loaded 'A' into its instruction cache, and into its instruction pipeline. Now the debugger sets a software breakpoint onto 'A', which means it (temporarily) replaces 'A' with an 'EBREAK' instruction. After that, the debugger lets the core resume again. In order to achieve that the core will execute the new 'EBREAK' instruction instead of the originally cached 'A' instruction, the debugger needs to flush the core's instruction cache and instruction pipeline before letting the CPU resume.

**Example #2:**

Let's assume an SoC with multiple RISC-V cores with data caches. Each core is halted. Each core has already cached the memory data 'X' of an address 'B' in their respective data cache. Now, the debugger uses the direct system bus access to write new data 'Y' into the data memory at address 'B'. When the debugger resumes the cores now, it needs to make sure that each core's data cache does not contain the old/outdated data 'X' anymore. That is why the debugger needs to flush each core's data cache before letting the cores resume, or before reading any memory content from the perspective of one of the cores.

# Cache Handling via FENCE/FENCE.I

In order to solve the above example scenarios and other scenarios, the debugger uses the **program buffer** execution of the **'FENCE'** and/or **'FENCE.I'** instructions. According to the RISC-V ISA specification, these instructions must ensure that a cache and/or pipeline flush (or an equivalent chip-specific operation) will happen, if applicable to the respective chip and if needed in the respective scenario.

The following points show example scenarios in which the debugger usually uses the FENCE and FENCE.I instructions. Of course, the described behavior may not always be applicable, and may also change in future debugger versions.

- **Resume core:** Before the debugger lets a core resume, it will execute both a 'FENCE' and 'FENCE.I' instruction via program buffer.
- **Read memory via core:** Before the debugger reads memory from perspective of a core (via program buffer or 'access memory' abstract command), it will execute a 'FENCE' instruction via program buffer.
- **Write memory via core:** After the debugger has written memory from perspective of a core (via program buffer or 'access memory' abstract command), it will execute a 'FENCE' instruction via program buffer.

The above actions by the debugger are only possible if the hardware's debug IP does have a program buffer with sufficient size, and if the FENCE and FENCE.I instructions are supported by the respective core. In that case, the debugger will execute the described steps independent of whether the respective core does have any caches or not.

# Cache Handling via the “Base Cache Management Operation” (CMO) ISA extension

The optional “Base Cache Management Operation” (CMO) ISA extensions (Zicbom, Zicboz, Zicbop) define standardized mechanisms to operate on caches of a RISC-V core. Information about this ISA extension can for example be found here: https://github.com/riscv/riscv-CMOs

As of today, the Lauterbach RISC-V debugger’s disassembler does support this extension. However, the debugger does not use the instructions of this extension to execute cache flushes via execution in the program buffer (to cover the scenarios mentioned above) just yet. However, we are considering to do so in the future. If you have a core that supports this extension and want the debugger to perform cache flushes with it, please contact support@lauterbach.com

# Cache Handling without Program Buffer

If the hardware's debug IP does however *not* have any program buffer, or if it does have a program buffer of insufficient size, or if FENCE/FENCE.I/CMO ISA extensions are not supported by the core, then none of the solutions mentioned above are possible. This is for example the case, if RISC-V debug IP only does have the ‘access memory’ abstract command for memory access from perspective of the core.

In this case, debugger must assume that the debug IP will automatically handle all respective cache-related scenarios itself. For example, the debug IP could automatically issue a cache flush and/or ensure cache coherency, whenever an ‘access memory’ abstract command is executed.

- Tags
- Hardware Designer
- RISC-V