Chapter 4: Developing RISC-V

Contents

Introduction

This chapter gets into the technical details of the RISC-V Instruction Set Architecture (ISA) and how it differs from other ISAs. We will cover some of the rules and regulations that govern RISC-V International, as well as how the ISA extensions and standards are developed.

By the end of this chapter, you should be able to:

- Describe the process used to develop the RISC-V ISA and extensions;
- Differentiate between the RISC-V Base ISAs, Extensions, and Standards;
- Understand the basics of the Unprivileged and Privileged specifications.

RISC-V Instruction Set Architecture Primer

Defining an Instruction Set Architecture (ISA)

An instruction set architecture (ISA) is an abstract model of a computer. It is also referred to as architecture or computer architecture. A realization of an ISA, such as a central processing unit (CPU), is called an implementation. Some ISAs you may have heard of include x86, ARM, MIPS, PowerPC, or SPARC. All of these ISAs require a license to implement them. On the other hand, the RISC-V ISA is provided under open-source licenses that do not require fees to use.

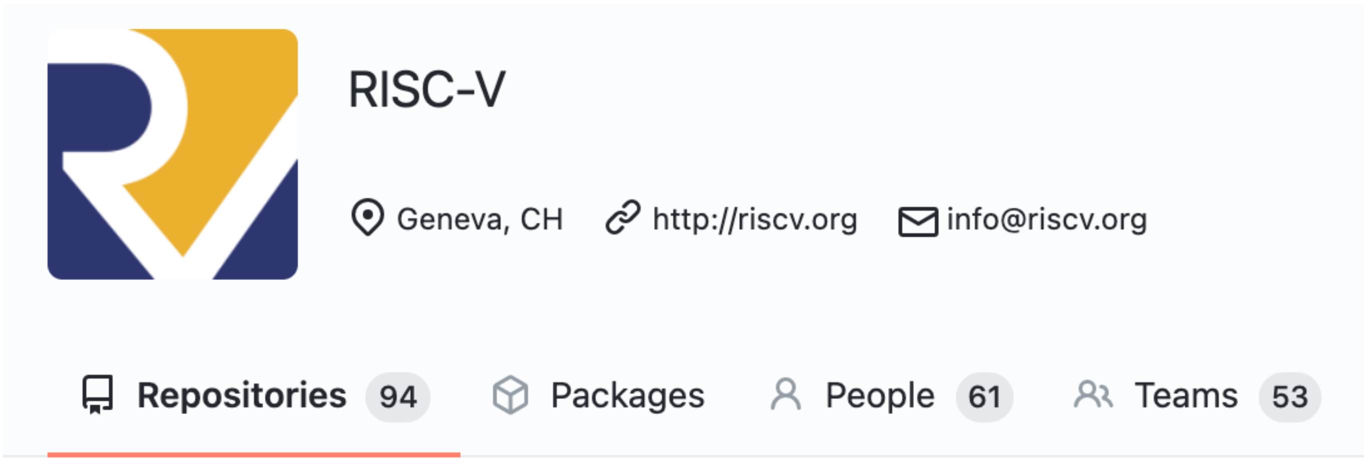
How the RISC-V ISA Is Different

The most notable difference between RISC-V and other ISAs is that RISC-V is developed by a member organization that is completely free to join and licenses its ISA with permissive open source licenses. This means that anyone can contribute to the specifications, and no one company or group of companies can drive the direction of the standards.

RISC-V International is governed by its Board of Directors. The Board is composed of members elected to represent all classes of membership to ensure we offer a strategic voice at all levels. In addition, the Technical Steering Committee (TSC) provides leadership to our technical initiatives in setting long term strategy, forming tactical committees and work groups, and approving technical deliverables for ratification or release.

Collaborative Development Model

A RISC-V Specification starts its life as a Task Group approved by the Technical Steering Committee (TSC). Once a Task Group has an approved charter, they begin work publicly on GitHub by writing their documents in AsciiDoc format. These repositories on GitHub can only receive pull requests from RISC-V International members, however the work is done publicly and transparently. For groups who choose to take minutes, those minutes from the Task Group meetings are published publicly as well. The public is free to submit issues to the GitHub repository in order to give early feedback on any specification. Non-ISA specifications and standards (e.g. processor trace, architectural tests, software overlay) are developed in a similar fashion.



Picture 1. RISC-V on GitHub.

RISC-V Specifications live on GitHub and are housed alongside dozens of software projects. See a list of ratified specifications[1] and the links to their GitHub repositories.

Creating and Curating Open Specifications

The process of writing the specifications is usually led by a Hardware Architect at one of the RISC-V International member organizations. They may not write the actual text, but they act as the chair to the Task Group overseeing the specification's development. It can take anywhere from several months to more than a year for the group to complete a specification. We will talk about the lifecycle of an extension later in this chapter.

What makes this development process open hinges on three key facts:

1. The Task Group mailing list is publicly visible.
2. The specification document is publicly visible and comments can be left.
3. There is a public mailing list where anyone can send email. (isa-dev@groups.riscv.org)

Using this methodology, even non-members can participate in the development of any specification or standard by asking questions, making suggestions, or simply following along. Furthermore, during the ratification process, there is a 45 day window where all specification work must be frozen and the specification published publicly for review. Anyone is welcome to comment at this time and all issues will be brought to resolution before the vote for ratification happens.

While becoming a member of RISC-V International is the easiest way to contribute to open specifications, it is not the only way. Anyway can contribute by interacting with the Task Groups in public forums like the mailing list and GitHub.

RISC-V Extensions Lifecycle

Each RISC-V extension goes through several stages on its way to ratification. In this section we will briefly review each stage known as a “milestone”.

1. **Plan.**

   The task group develops a final charter and sets some timeline estimates.

2. **Development.**

   The groups releases several versions considered unstable.

3. **Freeze.**

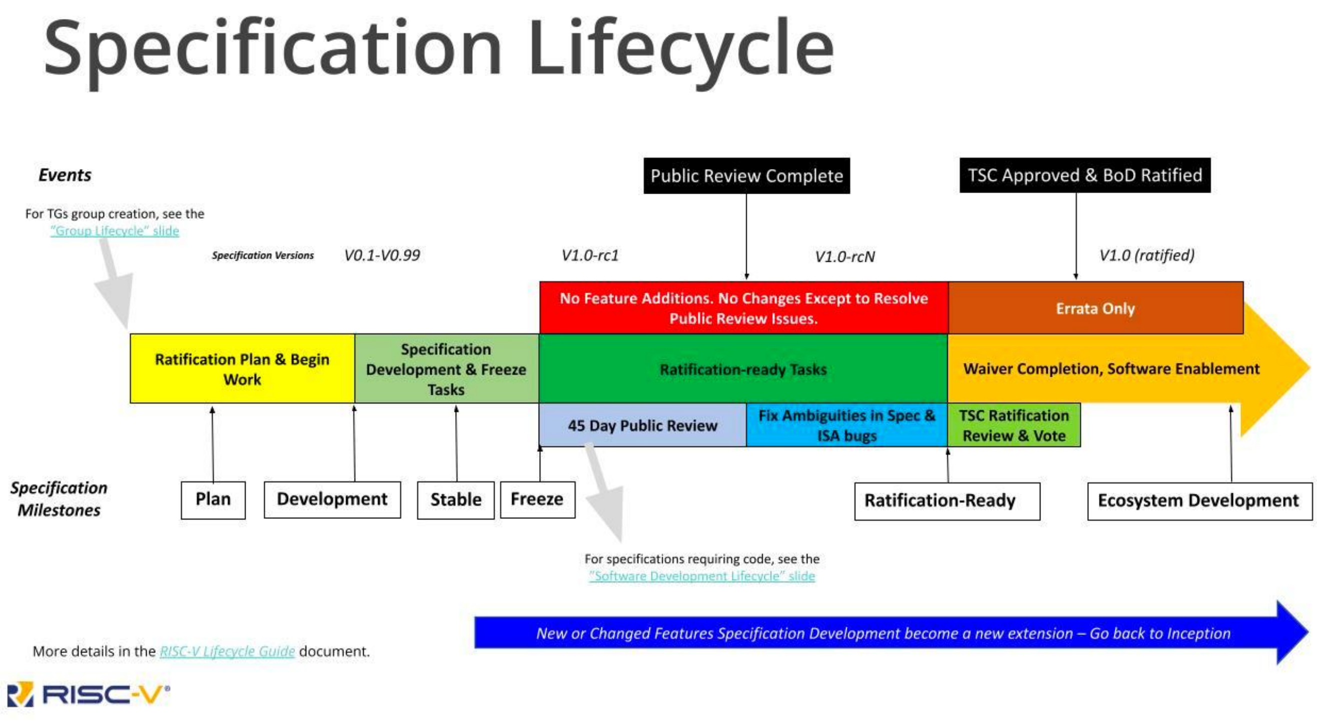
   The group produces a complete final draft of the specification with no major unknowns and no expected changes (only to fix issues but no new features).

4. **Ratification Ready.**

   The draft specification is sent out for public review, any public comments or questions are addressed, and the Technical Steering Committee is made aware that a vote is required.

5. **Ecosystem Development.**

   The extension is ratified and supported as part of the RISC-V ISA. Many tasks remain for the community to work on. As an example, the Vector specification was ratified, however adding auto-vectorization to compilers is part of that specification’s Ecosystem Development task list.


Specification Lifecycle
Events
For TGs group creation, see the "Group Lifecycle" slide
Public Review Complete
TSC Approved & BoD Ratified
Specification Versions
V0.1-V0.99
V1.0-rc1
V1.0-rcN
V1.0 (ratified)
No Feature Additions. No Changes Except to Resolve Public Review Issues.
Errata Only
Ratification Plan & Begin Work
Specification Development & Freeze Tasks
Ratification-ready Tasks
Waiver Completion, Software Enablement
45 Day Public Review
Fix Ambiguities in Spec & ISA bugs
TSC Ratification Review & Vote
Specification Milestones
Plan
Development
Stable
Freeze
Ratification-Ready
Ecosystem Development
For specifications requiring code, see the "Software Development Lifecycle" slide
New or Changed Features Specification Development become a new extension – Go back to Inception
More details in the RISC-V Lifecycle Guide document.
RISC-V

Picture 2. Specification life cycle.

П Once an extension has been ratified it is added to either the Unprivileged or Privileged Specification. Occasionally a specification is created as part of a separate document, with the debug specification being the most common example. However, this is a rare case and usually indicates that the extension is not part of the ISA, but rather a "standard" or "non-ISA specification". We will now review the Unprivileged and Privileged Specification in greater detail.

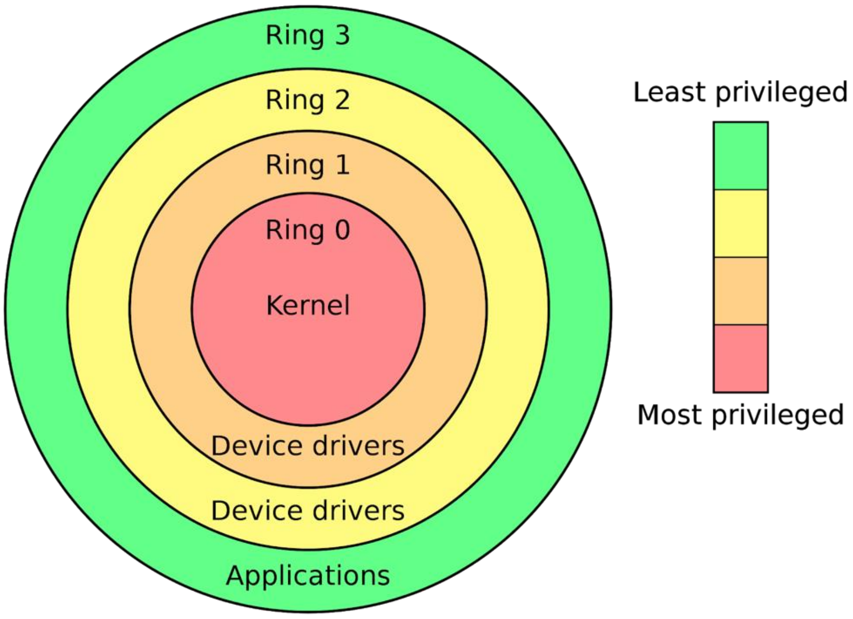
The Unprivileged Specification

Organizing the Specifications

The RISC-V ISA is broken up into two parts:

– Volume 1, Unprivileged Specification;
– Volume 2, Privileged Specification.

To understand why the specification is broken up into two different parts, we must first understand a bit about computer architecture and security. Historically, processors used hierarchical protection domains, often called protection rings, to protect data and code from malicious actors.



Picture 3. By Hertzsprung at English Wikipedia, CC BY-SA 3.0

The most privileged code runs in "Ring 0" and has access to the entire system. The processor will decide which privileges to grant executing code based on the privilege level. As an example, accessing memory by physical address may be restricted to "Ring 0" such that other rings must reference the virtual address space. Typically the processor can run in only one of the privilege modes at a time and there are special instructions to move between modes. All of these details can change from system to system, however they must follow the rules set out in the specification documents of a given architecture.

RISC-V currently has three privilege levels: User Mode (U-mode), Supervisor Mode (S-mode), and Machine Mode (M-mode). One can think of these as “Ring 2”, “Ring 1”, and “Ring 0” respectively. Other modes like a hypervisor mode (H-mode) will likely be added in the near future. Much like in the figure above, U-mode is for user processes, S-mode is for kernel and/or device drivers, and M-mode is used for bootloader and/or firmware. Each privilege level has access to specific Control and Status Registers (CSRs), and higher privilege levels can access the CSRs of those less privileged levels.

Inside the Unprivileged Specification

Simply put, the unprivileged specification details items that are not related to machine mode (M-mode) or to Supervisor Mode (S-mode). The unprivileged specification includes the base ISA as well as extensions to that base like integer (I), float (F), double (D), compressed instructions (C), and many more.

The base instruction sets describe the instruction format, basic integer instructions, load and store instructions, and other fundamental details of the ISA. We break these up into several bases:

– RV32I – Integer 32 bit;
– RV32E – Reduced RV32I for embedded purposes;
– RV64I – Integer 64 bit;
– RV128I – Integer 128 bit.

All these “Base ISA’s” either reduce or extend off the RV32I base instruction set. As an example, RV64I widens the integer registers and the supported user address space to 64 bits. This means that the LOAD and STORE instructions work a bit differently than in RV32I and the unprivileged specification contains the chapter explaining these differences.

Base ISA Extensions

The unprivileged specification also contains the descriptions of the extensions to these base ISAs. Again, any extension that does note require M-mode to operate can be described in the unprivileged specification.

Each extension to the base ISA is developed and maintained by a Task Group:

- Crypto Task Group working on cryptographic extensions which can move many complex cryptographic algorithms into hardware, improving reliability and speed;
- Extension Task Group working on bit manipulation extensions which can speed up many common mathematical tasks;
- Vector (V) Extension Task Group working on vector instructions which are at the heart of many graphical processing computations.

Once ratified, these extensions are added to the unprivileged specification. The following are some of the ratified extensions that you might see in a RISC-V processor.

“M” Standard Extension

Chapter 7 of the Unprivileged Specification [1] describes how integer multiplication and division should be accomplished. It describes how each of the multiplication instructions (MUL, MULH, MULHU, MULHU, MULW) will behave, which registers are used for the multiplier and multiplicand, and where the result will be stored. It does the same for division since functionally one can view division as simply the inverse of multiplication. It may seem odd to you that this extension is not required. However, for many embedded processors, multiplication can be done in software if it is not required very often or even at all. Removing this logic from a processor will save money on development, keeping the end product cost lower.

“F” Standard Extension

Chapter 11 describes how we add single-precision floating-point computational instructions that are compliant with the IEEE 754-2008 arithmetic standard. There are many resources available covering the details of floating-point arithmetic in computing. It is enough to understand that this chapter describes how this process is implemented in RISC-V, and is complimented by Chapter 12 (the D extension) which describes double-precision floating-point computational instructions. Lastly, Chapter 13 covers the Q standard extension for 128-bit quad-precision binary floating-point instructions. All three of these conform to IEEE standards. Again, many embedded applications do not require floating point logic, and hence this extension is not part of the Base ISAs.

“C” Standard Extension

Chapter 16 describes the compressed instruction-set extension which reduces static and dynamic code size by adding short 16-bit instruction encodings for common operations. Typically, 50%–60% of the

RISC-V instructions in a program can be replaced with RVC instructions, resulting in a 25%–30% code-size reduction. The C extension is compatible with all other standard instruction extensions. The C extension allows 16-bit instructions to be freely intermixed with 32-bit instructions, with the latter able to start on any 16-bit boundary. As such, with the addition of the C extension to any system, no instructions can raise instruction-address-misaligned exceptions.

This covers most of the currently ratified extensions in the unprivileged specification. However, it is important to note that many extensions are included in the specification in a "draft" or "frozen" stage. As we discussed in the section on "RISC-V Extension Lifecycle", these specifications are not yet ratified and any implementation should avoid using them in production.

The Privileged Specification

Overview

As its name suggests, the privileged specification contains descriptions of the RISC-V ISA which operate in Machine Mode (M-mode) or Supervisor Mode (S-mode). These modes have elevated privileges and are therefore described in a completely separate document from the base ISA and standard extensions. This specification also contains additional functionality required for running rich operating systems like Linux.

The first part of each chapter of the privileged specification details the Control and Status Registers (CSRs) which are only accessible from M-mode and S-mode. We will not cover these details here, but will rather focus on other details specific to these two modes.

Machine-Level (M-Mode) ISA, Version 1.11

This chapter describes the machine-level features available in machine-mode (M-mode). M-mode is used for low-level access to a hardware platform and is the first mode entered at reset, when the processor finishes initializing and is ready to execute code. M-mode can also be used to implement features that are too difficult or expensive to implement in hardware directly. A good example of this would be a watchdog timer implemented in low level software (firmware) which helps the system recover from faults. We will cover three important features of M-mode described in the specification: non-maskable interrupts, physical memory attributes, and physical memory protection.

Non-Maskable Interrupts (NMI)

Non-maskable interrupts (NMI) are only used for hardware error conditions. When fired, they cause an immediate jump to an NMI handler running in M-mode, regardless of how that hardware thread has its interrupt enable bit set. In other words, that interrupt will be serviced without a way to block the service in configuration. Each NMI will have a “mcause” register associated with it. This allows implementations to decide how they wish to handle these interrupts and allows them to define many possible causes. NMIs do not reset processor state which enables diagnosis, reporting, and possible containment of the hardware error.

Physical Memory Attributes (PMA)

The physical memory map for a system includes address ranges like: memory regions, memory-mapped control registers, and empty holes in the address space. Some memory regions might not support reads, writes, or execution; some might not support subword or subblock accesses; some might not support atomic operations; and some might not support cache coherence or might have different memory models. In RISC-V systems, these properties and capabilities of each region of the machine’s physical address space are termed physical memory attributes (PMA).

The PMAs of some memory regions are fixed at chip design time—for example, for an on-chip ROM. Others are fixed at board design time, depending, for example, on which other chips are connected to off-chip buses. Some devices might be configurable at run time to support different uses that imply different PMAs—for example, an on-chip scratchpad RAM might be cached privately by one core in one end-application, or accessed as a shared non-cached memory in another end-application. Most systems will require that at least some PMAs are dynamically checked in hardware later in the execution pipeline after the physical address is known, as some operations will not be supported at all physical memory addresses, and some operations require knowing the current setting of a configurable PMA attribute.

For RISC-V, we separate out specification and checking of PMAs into a separate hardware structure, the “PMA checker”. In many cases, the attributes are known at system design time for each physical address region, and can be hardwired into the PMA checker. Where the attributes are run-time configurable, platform-specific memory-mapped control registers can be provided to specify these attributes at a granularity appropriate to each region on the platform (e.g., for an on-chip static random-access memory (SRAM) that can be flexibly divided between cacheable and uncacheable uses).

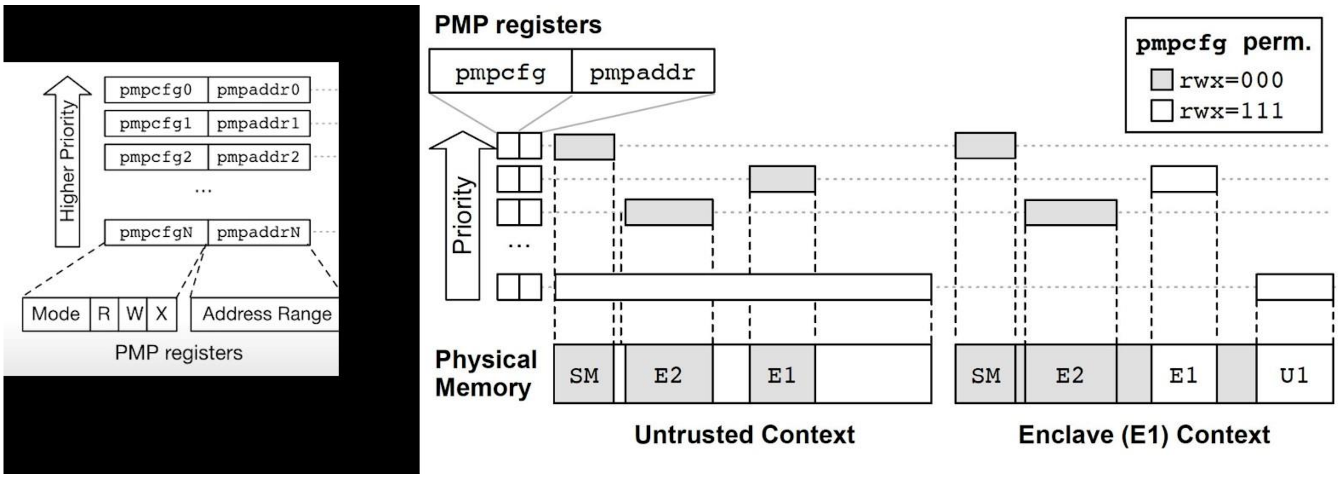
The details of PMAs could easily take up an entire chapter of this course. We will not cover memory-ordering PMAs, idempotency PMAs, coherence PMAs, or cacheability PMAs. The details of PMAs are described in detail in section 3.5 of the Privileged Specification [1]. Advanced users may want to review this section.

Physical Memory Protection (PMP)

A common feature of most modern processors is some way of performing secure remote computation or a "trusted execution environment". Examples of this technology include Intel Software Guard Extensions (SGX), AMD Secure Encrypted Virtualization (SEV), and Arm TrustZone. While the RISC-V ISA does not provide an end-to-end solution for Trusted Execution Environments, the physical memory protection (PMP) capabilities are a solid foundation on which one might construct such a system.

RISC-V PMP limits the physical addresses accessible by software running on a hart (hardware thread). An optional PMP unit provides per-hart machine-mode control registers to allow physical memory access privileges (read, write, execute) to be specified for each physical memory region. The PMP values are checked in parallel with the PMA checks we covered in the last section. The granularity of PMP access control settings are platform-specific and within a platform may vary by physical memory region, but the standard PMP encoding supports regions as small as four bytes. Certain regions' privileges can be hardwired—for example, some regions might only ever be visible in machine mode but in no lower-privilege layers.

PMP entries are described by an 8-bit configuration register and one 32 (or 64) bit address register. Up to 16 PMP entries are supported. If any PMP entries are implemented, then all PMP CSRs must be implemented, but any PMP CSR fields may be hardwired to zero. PMP CSRs are only accessible to M-mode.



Picture 4. By Lee, D., Kohlbrenner, D., Shinde, S., Song, D., & Asanovic, K. (2019). Keystone: A Framework for Architecting TEEs. CoRR, vol. abs/1907.10119

Here we see an example of how one might set up two different contexts, one untrusted and one with access to "Enclave E1". In this example an application is run in user context U1. That application only has access to its own memory and the memory inside enclave E1. The memory inside enclave E2 and that located in the "security monitor" (SM) are not available to the user application. In this way, data confidentiality is assured simply by allowing the security monitor (running in M-mode) to change the PMP settings allowing or denying access to memory regions based on the PMP configurations.

Supervisor-Level (S-Mode) ISA, Version 1.11

This chapter describes the RISC-V supervisor-level architecture, which contains a common core that is used with various supervisor-level address translation and protection schemes. Supervisor mode is deliberately restricted in terms of interactions with underlying physical hardware, such as physical memory and device interrupts, to support clean virtualization. In this spirit, certain supervisor-level facilities, including requests for timer and interprocessor interrupts, are provided by implementation-specific mechanisms. In some systems, a supervisor execution environment (SEE) provides these facilities in a manner specified by a supervisor binary interface (SBI). Other systems supply these facilities directly, through some other implementation-defined mechanism.

RISC-V supports Page-Based 32-bit, 39-bit, and 48-bit virtual memory addressing. The supervisor (S-Mode) memory-management fence instruction (SFENCE.VMA) is used to synchronize updates to in-memory memory-management data structures with current execution. Executing this instruction guarantees that any previous stores already visible to the current RISC-V hart (hardware thread) are ordered before all subsequent implicit references from that hart to the memory-management data structures.

Virtual Memory is a concept which takes several months of graduate level education to grasp and is beyond the scope of this course. It is enough for this course that you understand that RISC-V supports Page-Based virtual memory of several widths, and that there is a special S-Mode instruction used for synchronizing updates between hardware threads.

Non-ISA Specifications

Task Groups can also work on software or standards that are not part of the ISA. For example, the following groups work on projects that do not lead to specifications being written, but rather standards that encourage communities to develop their products around a common framework:

- Debug Task Group working on external debugging support and standards;
- Compliance Task Group working on RISC-V ISA compliance tests and frameworks.
- Configuration Structure Task Group working on how to represent the configuration structure of a given hardware implementation both in a human-readable format, as well as a binary format.

List of sources

1. Specifications: [Electronic resource] // RISC-V. URL: https://riscv.org/technical/specifications/.