RVA23 Profile

Version 0.4, March 21, 2024: This document has been prepared for internal review by RISC-V International. Changes prior to ratification remain likely.

Table of Contents

Preamble

This document is in the development state.

Do not use for implementations. Assume everything can change.

RVA Profiles Rationale

RISC-V was designed to provide a highly modular and extensible instruction set and includes a large and growing set of standard extensions, where each standard extension is a bundle of instruction-set features. This is no different than other industry ISAs that continue to add new ISA features. Unlike other ISAs, however, RISC-V has a broad set of contributors and implementers, and also allows users to add their own custom extensions. For some deep embedded markets, highly customized processor configurations are desirable for efficiency, and all software is compiled, ported, and/or developed in-house by the same organization for that specific processor configuration. However, for other markets that expect a substantial fraction of software to be delivered to end-customers in binary form, compatibility across multiple implementations from different RISC-V vendors is required.

The RVIA ISA extension ratification process ensures that all processor vendors have agreed to the specification of a standard extension if present. However, by themselves, the ISA extension specifications do not guarantee that a certain set of standard extensions will be present in all implementations.

The primary goal of the RVA profiles is to align processor vendors targeting binary software markets, so software can rely on the existence of a certain set of ISA features in a particular generation of RISC-V implementations.

Alignment is not only for compatibility, but also to ensure RISC-V is competitive in these markets. The binary app markets are also generally those with the most competitive performance requirements (e.g., mobile, client, server). RVIA cannot mandate the ISA features that a RISC-V binary software ecosystem should use, as each ecosystem will typically select the lowest-common denominator they empirically observe in the deployed devices in their target markets. But RVIA can align hardware vendors to support a common set of features in each generation through the RVA profiles. Without proactive alignment through RVA profiles, RISC-V will be uncompetitive, as even if a particular vendor implements a certain feature, if other vendors do not, then binary distributions will not generally use that feature and all implementations will suffer. While certain features may be discoverable, and alternate code provided in case of presence/absence of a feature, the added cost to support such options is only justified for certain limited cases, and binary app markets will not support a wide range of optional features, particularly for the nascent RISC-V binary app ecosystems.

To maintain alignment and increase RISC-V competitiveness over time, the mandatory set of extensions must increase over time in successive generations of RVA profile. (RVA profiles may eventually have to deprecate previously mandatory instructions, but that is unlikely in the near future.) Note that the RISC-V ISA will continue to evolve, regardless of whether a given software ecosystem settles on a certain generation of profile as the baseline for their ecosystem for many years or even decades. There are many existing binary software ecosystems, which will migrate to RISC-V and evolve at different rates, and more new ones will doubtless be created over the hopefully long lifetime of RISC-V. High-performance application processors require considerable investment, and no single binary app ecosystem can justify the development costs of these processors, especially for RISC-V in its early stage of adoption.

While the heart of the profile is the set of mandatory extensions, there are several kinds of optional

extension that serve important roles in the profile.

The first kind are *localized options*, whose presence or use necessarily differs along geo-political and/or jurisdictional boundaries, with crypto being the obvious example. These will always be optional. At least for crypto, discovery has been found to be perfectly acceptable to handle this optionality on other architectures, as the use of the extensions is well contained in certain libraries.

The second kind of optional extension is a *development option*, which represents a new ISA extension in an early part of its lifecycle but which is intended to become mandatory in a later generation of the RVA profile. Processor vendors and software toolchain providers will have varying development schedules, and providing an optional phase in a new extension's lifecycle provides some flexibility while maintaining overall alignment, and is particularly appropriate when hardware or software development for the extension is complex. Denoting an extension as a *development option* signals to the community that development should be prioritized for such extensions as they will become mandatory.

The third kind of optional extension are *expansion options*, which are those that may have a large implementation cost but are not always needed in a particular platform, and which can be readily handled by discovery. These are also intended to remain available as expansion options in future versions of the profile. Several supervisor-mode extensions fall into this category, e.g., Sv57, which has a notable PPA impact over Sv48 and is not needed on smaller platforms. Some unprivileged extensions that may fall into this category are possible future matrix extensions. These have large implementation costs, and use of matrix instructions can be readily supported with discovery and alternate math libraries.

The fourth kind of optional extensions are *transitory options*, where it is not clear if the extension will change to a mandatory, localized, or expansion option, or be possibly dropped over time. Cryptography provides some examples where earlier cyphers have been broken and are now deprecated. RVIA used this mechanism to enable scalar crypto until vector crypto was ready. Software security features may also be in this category, with examples of deprecated security features occuring in other architectures. As another example, the recent avalanche of new numeric datatypes for AI/ML may eventually subside with a few survivors actually being used longer term. Denoting an option as transitory signals to the community that this extension may be removed in a future profile, though the time scale may span many years.

Except for the localized options, it could be argued that other three kinds of option could be left out of profiles. Binary distributions of applications willing to invest in discovery can use an optional extension, and customers compiling their own applications can take advantage of the feature on a particular implementation, even when that system is mostly running binary distributions that ignore the new extension. However, there is value in providing guidance to align hardware vendors and software developers around what extensions are worth implementing and worth discovering, by designating only a few important features as profile options and limiting their granularity.

RVA23 Profiles

The RVA23 profiles are intended to align implementations of RISC-V 64-bit application processors to allow binary software ecosystems to rely on a a large set of guaranteed extensions and a small number of discoverable coarse-grain options. It is explicitly a non-goal of RVA23 to allow more hardware implementation flexibility by supporting only a minimal set of features and a large number of fine-grain extensions.

Only user-mode (RVA23U64) and supervisor-mode (RVA23S64) profiles are specified in this family.

RVA23U64 Profile

The RVA23U64 profile specifies the ISA features available to user-mode execution environments in 64-bit applications processors. This is the most important profile within the application processor family in terms of the amount of software that targets this profile.

RVA23U64 Mandatory Base

RV64I is the mandatory base ISA for RVA23U64 and is little-endian. As per the unprivileged architecture specification, the `ecall` instruction causes a requested trap to the execution environment.

RVA23U64 Mandatory Extensions

The following mandatory extensions were present in RVA22U64.

- **M** Integer multiplication and division.
- **A** Atomic instructions.
- **F** Single-precision floating-point instructions.
- **D** Double-precision floating-point instructions.
- **C** Compressed Instructions.
- **Zicsr** CSR instructions. These are implied by presence of F.
- **Zicntr** Base counters and timers.
- **Zihpm** Hardware performance counters.
- **Ziccif** Main memory regions with both the cacheability and coherence PMAs must support instruction fetch, and any instruction fetches of naturally aligned power-of-2 sizes up to min(ILEN,XLEN) (i.e., 32 bits for RVA23) are atomic.
- **Ziccrse** Main memory regions with both the cacheability and coherence PMAs must support RsrvEventual.
- **Ziccamoa** Main memory regions with both the cacheability and coherence PMAs must support AMOArithmetic.
- **Zicclsm** Misaligned loads and stores to main memory regions with both the cacheability and coherence PMAs must be supported.

- **Za64rs** Reservation sets are contiguous, naturally aligned, and a maximum of 64 bytes.
- **Zihintpause** Pause instruction.
- **Zba** Address computation.
- **Zbb** Basic bit manipulation.
- **Zbs** Single-bit instructions.
- **Zic64b** Cache blocks must be 64 bytes in size, naturally aligned in the address space.
- **Zicbom** Cache-Block Management Operations.
- **Zicbop** Cache-Block Prefetch Operations.
- **Zicboz** Cache-Block Zero Operations.
- **Zfhmin** Half-Precision Floating-point transfer and convert.
- **Zkt** Data-independent execution time.

The following mandatory extensions are new in RVA23U64:

- **V** Vector Extension.

V was optional in RVA22U64.

- **Zvfhmin** Vector FP16 conversion instructions.
- **Zvbb** Vector bit-manipulation instructions.
- **Zvkt** Vector data-independent execution time.
- **Zihintntl** Non-temporal locality hints.
- **Zicond** Conditional Zeroing instructions.
- **Zimop** Maybe Operations.
- **Zcmop** Compressed Maybe Operations.
- **Zcb** Additional 16b compressed instructions.
- **Zfa** Additional scalar FP instructions.
- **Zawrs** Wait on reservation set.
- **Supm** Pointer masking, with the execution environment providing a means to select PMLEN=0 and PMLEN=7 at minimum.

RVA23U64 Optional Extensions

RVA23U64 has eleven profile options (Zvkng, Zvksg, Zacas, Zvbc, Zfh, Zbc, Zvfh, Zfbfmin, Zvfbfmin, Zvfbfwma, Zama16b).

Localized Options

The following localized options are new in RVA23U64:

- **Zvkng** Vector Crypto NIST Algorithms including GHASH.

- **Zvksg** Vector Crypto ShangMi Algorithms including GHASH.

The scalar crypto extensions Zkn and Zks that were options in RVA22 are not options in RVA23. The goal is for both hardware and software vendors to move to use vector crypto, as vectors are now mandatory and vector crypto is substantially faster than scalar crypto.

We have included only the Zvkng/Zvksg options with GHASH to standardize on a higher performance crypto alternative. Zvbc is listed as a development option for use in other algorithms, and will become mandatory. Scalar Zbc is now listed as an expansion option, i.e., it will probably not become mandatory.

Development Options

The following are new development options intended to become mandatory in RVA24U64:

- **Zabha** Byte and Halfword Atomic Memory Operations
- **Zacas** Compare-and-swap
- **Ziccamoc** Main memory regions with both the cacheability and coherence PMAs must provide AMOCASQ level PMA support.
- **Zvbc** Vector carryless multiply.
- **Zama16b** Misaligned loads, stores, and AMOs to main memory regions that do not cross a naturally aligned 16-byte boundary are atomic.

Expansion Options

The following expansion options were also present in RVA22U64:

- **Zfh** Scalar Half-Precision Floating-Point (FP16).

The following are new expansion options in RVA23U64:

- **Zbc** Scalar carryless multiply.
- **Zvfh** Vector half-precision floating-point (FP16).
- **Zfbfmin** Scalar BF16 FP conversions.
- **Zvfbfmin** Vector BF16 FP conversions.
- **Zvfbfwma** Vector BF16 widening mul-add.

Transitory Options

There are no transitory options in RVA23U64.

Scalar crypto is no longer an option in RVA23U64, though the Zbc extension has now been exposed as an expansion option.

RVA23U64 Recommendations

Implementations are strongly recommended to raise illegal-instruction exceptions on attempts to execute unimplemented opcodes.

RVA23S64 Profile

The RVA23S64 profile specifies the ISA features available to a supervisor-mode execution environment in 64-bit applications processors. RVA23S64 is based on privileged architecture version 1.13.

Priv 1.13 is still being defined.

RVA23S64 Mandatory Base

RV64I is the mandatory base ISA for RVA23S64 and is little-endian. The `ecall` instruction operates as per the unprivileged architecture specification. An `ecall` in user mode causes a contained trap to supervisor mode. An `ecall` in supervisor mode causes a requested trap to the execution environment.

RVA23S64 Mandatory Extensions

The following unprivileged extensions are mandatory:

- The RVA23S64 mandatory unprivileged extensions include all the mandatory unprivileged extensions in RVA23U64.
- **Zifencei** Instruction-Fetch Fence.

Zifencei is mandated as it is the only standard way to support instruction-cache coherence in RVA23 application processors. A new instruction-cache coherence mechanism is under development (tentatively named Zjid) which might be added as an option in the future.

The following privileged extensions are mandatory:

- **Ss1p13** Supervisor Architecture version 1.13.

Ss1p13 supersedes Ss1p12 but is not yet ratified.

The following privileged extensions were also mandatory in RVA22S64:

- **Svbare** The `satp` mode Bare must be supported.
- **Sv39** Page-Based 39-bit Virtual-Memory System.
- **Svade** Page-fault exceptions are raised when a page is accessed when A bit is clear, or written when D bit is clear.
- **Ssccptr** Main memory regions with both the cacheability and coherence PMAs must support hardware page-table reads.

- **Sstvecd** `stvec.MODE` must be capable of holding the value 0 (Direct). When `stvec.MODE=Direct`, `stvec.BASE` must be capable of holding any valid four-byte-aligned address.
- **Sstvala** stval must be written with the faulting virtual address for load, store, and instruction page-fault, access-fault, and misaligned exceptions, and for breakpoint exceptions other than those caused by execution of the EBREAK or C.EBREAK instructions. For illegal-instruction exceptions, stval must be written with the faulting instruction.
- **Sscounterenw** For any hpmcounter that is not read-only zero, the corresponding bit in scounteren must be writable.
- **Svpbmt** Page-Based Memory Types
- **Svinval** Fine-Grained Address-Translation Cache Invalidation

The following are new mandatory extensions:

- **Svnapot** NAPOT Translation Contiguity

Svnapot was optional in RVA22.

- **Sstc** supervisor-mode timer interrupts.

Sstc was optional in RVA22.

- **Sscofpmf** Count Overflow and Mode-Based Filtering.
- **Ssnpm** Pointer masking, with `senvcfg.PME` and `henvcfg.PME` supporting, at minimum, settings PMLEN=0 and PMLEN=7.
- **Ssu64xl** `sstatus.UXL` must be capable of holding the value 2 (i.e., UXLEN=64 must be supported).

Ssu64xl was optional in RVA22.

- **H** The hypervisor extension.

The hypervisor was optional in RVA22.

The following extensions were required when the hypervisor was implemented in RVA22.

- **Ssstateen** Supervisor-mode view of the state-enable extension. The supervisor-mode (`sstateen0-3`) and hypervisor-mode (`hstateen0-3`) state-enable registers must be provided.
- **Shcounterenw** For any `hpmcounter` that is not read-only zero, the corresponding bit in `hcounteren` must be writable.
- **Shvstvala** `vstval` must be written in all cases described above for `stval`.
- **Shtvala** `htval` must be written with the faulting guest physical address in all circumstances permitted by the ISA.
- **Shvstvecd** `vstvec.MODE` must be capable of holding the value 0 (Direct). When `vstvec.MODE`=Direct, `vstvec.BASE` must be capable of holding any valid four-byte-aligned address.

- **Shvsatpa** All translation modes supported in `satp` must be supported in `vsatp`.
- **Shgatpa** For each supported virtual memory scheme SvNN supported in `satp`, the corresponding hgatp SvNNx4 mode must be supported. The `hgatp` mode Bare must also be supported.

RVA23S64 Optional Extensions

RVA23S64 has ten unprivileged options (Zvkng, Zvksg, Zacas, Zvbc, Zfh, Zbc, Zvfh, Zfbfmin, Zvfbfmin, Zvfbfwma) from RVA23U64, and seven privileged options (Sv48, Sv57, Svadu, Zkr, Sdext, Ssstrict, Svvptc).

Localized Options

There are no privileged localized options in RVA23S64.

Development Options

There are no privileged development options in RVA23S64.

Expansion Options

The following privileged expansion options were present in RVA22S64:

- **Sv48** Page-Based 48-bit Virtual-Memory System.
- **Sv57** Page-Based 57-bit Virtual-Memory System.
- **Zkr** Entropy CSR.

The following are new privileged expansion options in RVA23S64

- **Svadu** Hardware A/D bit updates.
- **Sdext** Debug triggers
- **Ssstrict** No non-conforming extensions are present. Attempts to execute unimplemented opcodes or access unimplemented CSRs in the standard or reserved encoding spaces raises an illegal instruction exception that results in a contained trap to the supervisor-mode trap handler.

  

  Ssstrict does not prescribe behavior for the custom encoding spaces or CSRs.

- **Svvptc** Transitions from invalid to valid PTEs will be visible in bounded time without an explicit SFENCE.
- **Sspm** Supervisor-mode pointer masking, with the supervisor execution environment providing a means to select PMLEN=0 and PMLEN=7 at minimum.

Transitory Options

There are no privileged transitory options in RVA23S64.

RVA23S64 Recommendations

- Implementations are strongly recommended to raise illegal-instruction exceptions when attempting to execute unimplemented opcodes or access unimplemented CSRs.

Glossary of ISA Extensions

The following unprivileged ISA extensions are defined in Volume I of the RISC-V Instruction Set Manual.

- M Extension for Integer Multiplication and Division
- A Extension for Atomic Memory Operations
- F Extension for Single-Precision Floating-Point
- D Extension for Double-Precision Floating-Point
- Q Extension for Quad-Precision Floating-Point
- C Extension for Compressed Instructions
- Zifencei Instruction-Fetch Synchronization Extension
- Zicsr Extension for Control and Status Register Access
- Zicntr Extension for Basic Performance Counters
- Zihpm Extension for Hardware Performance Counters
- Zihintpause Pause Hint Extension
- Zfh Extension for Half-Precision Floating-Point
- Zfhmin Minimal Extension for Half-Precision Floating-Point
- Zfinx Extension for Single-Precision Floating-Point in x-registers
- Zdinx Extension for Double-Precision Floating-Point in x-registers
- Zhinx Extension for Half-Precision Floating-Point in x-registers
- Zhinxmin Minimal Extension for Half-Precision Floating-Point in x-registers

The following privileged ISA extensions are defined in Volume II of the RISC-V Instruction Set Manual.

- Sv32 Page-based Virtual Memory Extension, 32-bit
- Sv39 Page-based Virtual Memory Extension, 39-bit
- Sv48 Page-based Virtual Memory Extension, 48-bit
- Sv57 Page-based Virtual Memory Extension, 57-bit
- Svpbmt, Page-Based Memory Types
- Svnapot, NAPOT Translation Contiguity
- Svinval, Fine-Grained Address-Translation Cache Invalidation
- Hypervisor Extension
- Sm1p11, Machine Architecture v1.11
- Sm1p12, Machine Architecture v1.12
- Ss1p11, Supervisor Architecture v1.11

- Ss1p12, Supervisor Architecture v1.12
- Ss1p13, Supervisor Architecture v1.13

The following extensions have not yet been incorporated into the RISC-V Instruction Set Manual; the hyperlinks lead to their separate specifications.

- Zba Address Computation Extension
- Zbb Bit Manipulation Extension
- Zbc Carryless Multiplication Extension
- Zbs Single-Bit Manipulation Extension
- Zbkb Extension for Bit Manipulation for Cryptography
- Zbkc Extension for Carryless Multiplication for Cryptography
- Zbkx Crossbar Permutation Extension
- Zk Standard Scalar Cryptography Extension
- Zkn NIST Cryptography Extension
- Zknd AES Decryption Extension
- Zkne AES Encryption Extension
- Zknh SHA2 Hashing Extension
- Zkr Entropy Source Extension
- Zks ShangMi Cryptography Extension
- Zksed SM4 Block Cypher Extension
- Zksh SM3 Hashing Extension
- Zkt Extension for Data-Independent Execution Latency
- V Extension for Vector Computation
- Zve32x Extension for Embedded Vector Computation (32-bit integer)
- Zve32f Extension for Embedded Vector Computation (32-bit integer, 32-bit FP)
- Zve32d Extension for Embedded Vector Computation (32-bit integer, 64-bit FP)
- Zve64x Extension for Embedded Vector Computation (64-bit integer)
- Zve64f Extension for Embedded Vector Computation (64-bit integer, 32-bit FP)
- Zve64d Extension for Embedded Vector Computation (64-bit integer, 64-bit FP)
- Zicbom Extension for Cache-Block Management
- Zicbop Extension for Cache-Block Prefetching
- Zicboz Extension for Cache-Block Zeroing
- Sstc Extension for Supervisor-mode Timer Interrupts
- Sscofpmf Extension for Count Overflow and Mode-Based Filtering
- Smstateen Extension for State-enable

- Svvptc Eliding Memory-management Fences on setting PTE valid
- Zacas Extension for Atomic Compare-and-Swap (CAS) instructions
- Zabha Extension for Byte and Halfword Atomic Memory Operations
- **Ziccif**: Main memory supports instruction fetch with atomicity requirement
- **Ziccrse**: Main memory supports forward progress on LR/SC sequences
- **Ziccamoa**: Main memory supports all atomics in A
- **Ziccamoc** Main memory supports atomics in Zacas
- **Zicclsm**: Main memory supports misaligned loads/stores
- **Zama16b**: Misaligned loads, stores, and AMOs to main memory regions that do not cross a naturally aligned 16-byte boundary are atomic.
- **Za64rs**: Reservation set size of at most 64 bytes
- **Za128rs**: Reservation set size of at most 128 bytes
- **Zic64b**: Cache block size isf 64 bytes
- **Svbare**: Bare mode virtual-memory translation supported
- **Svade**: Raise exceptions on improper A/D bits
- **Ssccptr**: Main memory supports page table reads
- **Sscounterenw**: Support writeable enables for any supported counter
- **Sstvecd**: `stvec` supports Direct mode
- **Sstvala**: `stval` provides all needed values
- **Ssu64xl**: UXLEN=64 must be supported
- **Ssstateen**: Supervisor-mode view of the state-enable extension
- **Shcounterenw**: Support writeable enables for any supported counter
- **Shvstvala**: `vstval` provides all needed values
- **Shtvala**: `htval` provides all needed values
- **Shvstvecd**: `vstvec` supports Direct mode
- **Shvsatpa**: `vsatp` supports all modes supported by `satp`
- **Shgatpa**: SvNNx4 mode supported for all modes supported by `satp`, as well as Bare