RISC-V Profiles

Version 1.0, April 2, 2023: This document is in Ratified state.

Table of Contents

Preamble

This document is in the Ratified state

No changes are allowed. Any desired or needed changes can be the subject of a follow-on new extension. Ratified extensions are never revised

Changes since Public Review version 0.8

- Clarified that profile name can be used as ISA base string
- Renamed Ssptead to Svade
- Fixed Ssu64xl to make supporting UXL=64 mandatory
- Added section listing new extension names in profiles document
- Added new extension name Sscounterenw
- Removed outdated text on Zicntr/Zihpm ratification plan

Chapter 1. Introduction

RISC-V was designed to provide a highly modular and extensible instruction set, and includes a large and growing set of standard extensions. In addition, users may add their own custom extensions. This flexibility can be used to highly optimize a specialized design by including only the exact set of ISA features required for an application, but the same flexibility also leads to a combinatorial explosion in possible ISA choices. Profiles specify a much smaller common set of ISA choices that capture the most value for most users, and which thereby enable the software community to focus resources on building a rich software ecosystem with application and operating system portability across different implementations.

Another pragmatic concern is the long and unwieldy ISA strings required to encode common sets of extensions, which will continue to grow as new extensions are defined.

Each profile is built on a standard base ISA plus a set of mandatory ISA extensions, and provides a small set of standard ISA options to extend the mandatory components. Profiles provide a convenient shorthand for describing the ISA portions of hardware and software platforms, and also guide the development of common software toolchains shared by different platforms that use the same profile. The intent is that the software ecosystem focus on supporting the profiles' mandatory base and standard options, instead of attempting to support every possible combination of individual extensions. Similarly, hardware vendors should aim to structure their offerings around standard profiles to increase the likelihood their designs will have mainstream software support.

Profiles are not intended to prohibit the use of combinations of individual ISA extensions or the addition of custom extensions, which can continue to be used for more specialized applications albeit without the expectation of widespread software support or portability between hardware platforms.

As RISC-V evolves over time, the set of ISA features will grow, and new platforms will be added that may need different profiles. To manage this evolution, RISC-V is adopting a model of regular annual releases of new ISA profiles, following an ISA roadmap managed by the RISC-V Technical Steering Committee. The architecture profiles will also be used for branding and to advertise compatibility with the RISC-V standard.

This document describes the general structure of RISC-V architecture profiles and also the specifics of the first few profiles: RVI20 is a generic RISC-V unprivileged software profile, and RVA20 and RVA22 are architecture profiles for application processors.

Chapter 2. Profiles versus Platforms

Profiles only describe ISA features, not a complete execution environment.

A *software platform* is a specification for an execution environment, in which software targeted for that software platform can run.

A *hardware platform* is a specification for a hardware system (which can be viewed as a physical realization of an execution environment).

Both software and hardware platforms include specifications for many features beyond details of the ISA used by RISC-V harts in the platform (e.g., boot process, calling convention, behavior of environment calls, discovery mechanism, presence of certain memory-mapped hardware devices, etc.). Architecture profiles factor out ISA-specific definitions from platform definitions to allow ISA profiles to be reused across different platforms, and to be used by tools (e.g., compilers) that are common across many different platforms.

A platform can add additional constraints on top of those in a profile. For example, mandating an extension that is a standard option in the underlying profile, or constraining some implementation-specific parameter in the profile to lie within a certain range.

A platform cannot remove mandates or reduce other requirements in a profile.

A new profile should be proposed if existing profiles do not match the needs of a new platform.

Chapter 3. Components of a Profile

3.1. Profile Family

Every profile is a member of a *profile family*. A profile family is a set of profiles that share the same base ISA but which vary in highest-supported privilege mode. The initial two types of family are:

- generic unprivileged instructions (I)
- application processors running rich operating systems (A)

> More profile families may be added over time.

A profile family may be updated no more than annually, and the release calendar year is treated as part of the profile family name.

Each profile family is described in more detail below.

3.2. Profile Privilege Mode

RISC-V has a layered architecture supporting multiple privilege modes, and most RISC-V platforms support more than one privilege mode. Software is usually written assuming a particular privilege mode during execution. For example, application code is written assuming it will be run in user mode, and kernel code is written assuming it will be run in supervisor mode.

> Software can be run in a mode different than the one for which it was written. For example, privileged code using privileged ISA features can be run in a user-mode execution environment, but will then cause traps into the enclosing execution environment when privileged instructions are executed. This behavior might be exploited, for example, to emulate a privileged execution environment using a user-mode execution environment.

The profile for a privilege mode describes the ISA features for an execution environment that has the eponymous privilege mode as the most-privileged mode available, but also includes all supported lower-privilege modes. In general, available instructions vary by privilege mode, and the behavior of RISC-V instructions can depend on the current privilege mode. For example, an S-mode profile includes U-mode as well as S-mode and describes the behavior of instructions when running in different modes in an S-mode execution environment, such as how an `ecall` instruction in U-mode causes a contained trap into an S-mode handler whereas an `ecall` in S-mode causes a requested trap out to the execution environment.

A profile may specify that certain conditions will cause a requested trap (such as an `ecall` made in the highest-supported privilege mode) or fatal trap to the enclosing execution environment. The profile does not specify the behavior of the enclosing execution environment in handling requested or fatal traps.

> In particular, a profile does not specify the set of ECALLs available in the outer

execution environment. This should be documented in the appropriate binary interface to the outer execution environment (e.g., Linux user ABI, or RISC-V SEE).

In general, a profile can be implemented by an execution environment using any hardware or software technique that provides compatible functionality, including pure software emulation.

A profile does not specify any invisible traps.

In particular, a profile does not constrain how invisible traps to a more-privileged mode can be used to emulate profile features.

A more-privileged profile can always support running software to implement a less-privileged profile from the same profile family. For example, a platform supporting the S-mode profile can run a supervisor-mode operating system that provides user-mode execution environments supporting the U-mode profile.

Instructions in a U-mode profile, which are all executed in user mode, have potentially different behaviors than instructions executed in user mode in an S-mode profile. For this reason, a U-mode profile cannot be considered a subset of an S-mode profile.

3.3. Profile ISA Features

An architecture profile has a mandatory ratified base instruction set (RV32I or RV64I for the current profiles). The profile also includes ratified ISA extensions placed into two categories:

1. Mandatory
2. Optional

As the name implies, *Mandatory ISA extensions* are a required part of the profile. Implementations of the profile must provide these. The combination of the profile base ISA plus the mandatory ISA extensions are termed the profile *mandates*, and software using the profile can assume these always exist.

The *Optional* category (also known as *options*) contains extensions that may be added as options, and which are expected to be generally supported as options by the software ecosystem for this profile.

The level of "support" for an Optional extension will likely vary greatly among different software components supporting a profile. Users would expect that software claiming compatibility with a profile would make use of any available supported options, but as a bare minimum software should not report errors or warnings when supported options are present in a system.

An optional extension may comprise many individually named and ratified extensions but a profile option requires all constituent extensions are present. In particular, unless explicitly listed as a profile option, individual extensions are not by themselves a profile option even when required as part of a profile option. For example, the Zbkb extension is not by itself a profile option even though it is a required component of the Zkn option.

Profile optional extensions are intended to capture the granularity at which the broad software ecosystem is expected to cope with combinations of extensions.

All components of a ratified profile must themselves have been ratified.

Platforms may provide a discovery mechanism to determine what optional extensions are present.

Extensions that are not explicitly listed in the mandatory or optional categories are termed *non-profile* extensions, and are not considered parts of the profile. Some non-profile extensions can be added to an implementation without conflicting with the mandatory or optional components of a profile. In this case, the implementation is still compatible with the profile even though additional non-profile extensions are present. Other non-profile extensions added to an implementation might alter or conflict with the behavior of the mandatory or optional extensions in a profile, in which case the implementation would not be compatible with the profile.

Extensions that are released after a given profile is released are by definition non-profile extensions. For example, mandatory or optional profile extensions for a new profile might be prototyped as non-profile extensions on an earlier profile.

3.4. Profile Naming Convention

A profile name is a string comprised of, in order:

1. Prefix **RV** for RISC-V.
2. A specific profile family name string. Initially a single letter (**I**, **M**, or **A**), but later profiles may have longer family name strings.
3. A numeric string giving the first complete calendar year for which the profile is ratified, represented as number of years after year 2000, i.e., **20** for profiles built on specifications ratified during 2019. The year string will be longer than two digits in the next century.
4. A privilege mode (**U**, **S**, **M**). Hypervisor support is treated as an option.
5. A base ISA XLEN specifier (**32**, **64**).

The initial profiles based on specifications ratified in 2019 are:

- RVI20U32 basic unprivileged instructions for RV32I
- RVI20U64 basic unprivileged instructions for RV64I
- RVA20U64, RVA20S64 64-bit application-processor profiles

Profile names are embeddable into RISC-V ISA naming strings. This implies that there will be no standard ISA extension with a name that matches the profile naming convention. This allows tools that process the RISC-V ISA naming string to parse and/or process a combined string.

Chapter 4. RVI20 Profiles

The RVI20 profiles document the initial set of unprivileged instructions. These provide a generic target for software toolchains and represent the minimum level of compatibility with RISC-V ratified standards. The two profiles RVI20U32 and RVI20U64 correspond to the RV32I and RV64I base ISAs respectively.

These are designed as *unprivileged* profiles as opposed to *user-mode* profiles. Code using this profile can run in any privilege mode, and so requested and fatal traps may be horizontal traps into an execution environment running in the same privilege mode.

4.1. RVI20U32

RVI20U32 specifies the ISA features available to generic unprivileged execution environments.

4.1.1. RVI20U32 Mandatory Base

RV32I is the mandatory base ISA for RVI20U32, and is little-endian.

As per the unprivileged architecture specification, the `ecall` instruction causes a requested trap to the execution environment.

Misaligned loads and stores might not be supported.

The `fence.tso` instruction is mandatory.

The `fence.tso` instruction was incorrectly described as optional in the 2019 ratified specifications. However, `fence.tso` is encoded within the standard `fence` encoding such that implementations must treat it as a simple global fence if they do not natively support TSO-ordering optimizations. As software can always assume without any penalty that `fence.tso` is being exploited by a hardware implementation, there is no advantage to making the instruction an option. Later versions of the unprivileged ISA specifications correctly indicate that `fence.tso` is mandatory.

4.1.2. RVI20U32 Mandatory Extensions

There are no mandatory extensions for RVI20U32.

4.1.3. RVI20U32 Optional Extensions

- **M** Integer multiplication and division.
- **A** Atomic instructions.
- **F** Single-precision floating-point instructions.
- **D** Double-precision floating-point instructions.

The rationale to not include Q as an optional extension is that quad-precision floating-point is unlikely to be implemented in hardware, and so we do not require or expect software to expend effort optimizing use of Q instructions in case they are present.

- **C** Compressed Instructions.
- **Zifencei** Instruction-fetch fence instruction.
- Misaligned loads and stores may be supported.
- **Zicntr** Basic counters.

The Zicsr extension is not supported independent of the Zicntr or F extensions.

- **Zihpm** Hardware performance counters.

4.2. RVI20U64

RVI20U64 specifies the ISA features available to generic unprivileged execution environments.

4.2.1. RVI20U64 Mandatory Base

RV64I is the mandatory base ISA for RVI20U64, and is little-endian.

As per the unprivileged architecture specification, the `ecall` instruction causes a requested trap to the execution environment.

Misaligned loads and stores might not be supported.

The `fence.tso` instruction is mandatory.

The `fence.tso` instruction was incorrectly described as optional in the 2019 ratified specifications. However, `fence.tso` is encoded within the standard `fence` encoding such that implementations must treat it as a simple global fence if they do not natively support TSO-ordering optimizations. As software can always assume without any penalty that `fence.tso` is being exploited by a hardware implementation, there is no advantage to making the instruction a profile option. Later versions of the unprivileged ISA specifications correctly indicate that `fence.tso` is mandatory.

4.2.2. RVI20U64 Mandatory Extensions

There are no mandatory extensions for RVI20U64.

4.2.3. RVI20U64 Optional Extensions

- **M** Integer multiplication and division.
- **A** Atomic instructions.

- **F** Single-precision floating-point instructions.
- **D** Double-precision floating-point instructions.

> The rationale to not include Q as a profile option is that quad-precision floating-point is unlikely to be implemented in hardware, and so we do not require or expect software to expend effort optimizing use of Q instructions in case they are present.

- **C** Compressed Instructions.
- **Zifencei** Instruction-fetch fence instruction.
- Misaligned loads and stores may be supported.
- **Zicntr** Basic counters.

> The Zicsr extension is not supported independent of the Zicntr or F extensions.

- **Zihpm** Hardware performance counters.

Chapter 5. RVA20 Profiles

The RVA20 profiles are intended to be used for 64-bit application processors running rich OS stacks. Only user-mode (RVA20U64) and supervisor-mode (RVA20S64) profiles are specified in this family.

> There is no machine-mode profile currently defined for application processor families. A machine-mode profile for application processors would only be used in specifying platforms for portable machine-mode software. Given the relatively low volume of portable M-mode software in this domain, the wide variety of potential M-mode code, and the very specific needs of each type of M-mode software, we are not specifying individual M-mode ISA requirements in the A-family profiles.

> Only XLEN=64 application processor profiles are currently defined. It would be possible to also define very similar XLEN=32 variants.

5.1. RVA20U64 Profile

The RVA20U64 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.

RVA20U64 has one optional extension (Zihpm).

5.1.1. RVA20U64 Mandatory Base

RV64I is the mandatory base ISA for RVA20U64, and is little-endian.

As per the unprivileged architecture specification, the `ecall` instruction causes a requested trap to the execution environment.

The `fence.tso` instruction is mandatory.

> The `fence.tso` instruction was incorrectly described as optional in the 2019 ratified specifications. However, `fence.tso` is encoded within the standard `fence` encoding such that implementations must treat it as a simple global fence if they do not natively support TSO-ordering optimizations. As software can always assume without any penalty that `fence.tso` is being exploited by a hardware implementation, there is no advantage to making the instruction a profile option. Later versions of the unprivileged ISA specifications correctly indicate that `fence.tso` is mandatory.

5.1.2. RVA20U64 Mandatory Extensions

- **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 Zicntr or F.
- **Zicntr** Basic 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 RVA20) are atomic.

  Ziccif is a new extension name capturing this feature. The fetch atomicity requirement facilitates runtime patching of aligned instructions.

- **Ziccrse** Main memory regions with both the cacheability and coherence PMAs must support RsrvEventual.

  Ziccrse is a new extension name capturing this feature.

- **Ziccamoa** Main memory regions with both the cacheability and coherence PMAs must support AMOArithmetic.

  Ziccamoa is a new extension name capturing this feature.

- **Za128rs** Reservation sets must be contiguous, naturally aligned, and at most 128 bytes in size.

  Za128rs is a new extension name capturing this feature. The minimum reservation set size is effectively determined by the size of atomic accesses in the A extension.

- **Zicclsm** Misaligned loads and stores to main memory regions with both the cacheability and coherence PMAs must be supported.

  This introduces a new extension name for this feature. This requires misaligned support for all regular load and store instructions (including scalar and vector) but not AMOs or other specialized forms of memory access. Even though mandated, misaligned loads and stores might execute extremely slowly. Standard software distributions should assume their existence only for correctness, not for performance.

5.1.3. RVA20U64 Optional Extensions

- **Zihpm** Hardware performance counters.

  Hardware performance counters are a supported option in RVA20. The number of counters is platform-specific.

  The rationale to not make Q an optional extension is that quad-precision floating-point is unlikely to be implemented in hardware, and so we do not require or expect A-profile software to expend effort optimizing use of Q instructions in case they are present.

Zifencei is not classed as a supported option in the user-mode profile because it is not sufficient by itself to produce the desired effect in a multiprogrammed multiprocessor environment without OS support, and so the instruction cache flush should always be performed using an OS call rather than using the `fence.i` instruction. `fence.i` semantics can be expensive to implement for some hardware memory hierarchy designs, and so alternative non-standard instruction-cache coherence mechanisms can be used behind the OS abstraction. A separate extension is being developed for more general and efficient instruction cache coherence.

The execution environment must provide a means to synchronize writes to instruction memory with instruction fetches, the implementation of which likely relies on the Zifencei extension. For example, RISC-V Linux supplies the `__riscv_flush_icache` system call and a corresponding vDSO call.

5.1.4. RVA20U64 Recommendations

Recommendations are not strictly mandated but are included to guide implementers making design choices.

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

5.2. RVA20S64 Profile

The RVA20S64 profile specifies the ISA features available to a supervisor-mode execution environment in 64-bit applications processors. RVA20S64 is based on privileged architecture version 1.11.

RVA20S64 has one unprivileged option (Zihpm) and one privileged option (Sv48).

5.2.1. RVA20S64 Mandatory Base

RV64I is the mandatory base ISA for RVA20S64, 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.

5.2.2. RVA20S64 Mandatory Extensions

The following unprivileged extensions are mandatory:

- The RVA20S64 mandatory unprivileged extensions include all the mandatory unprivileged extensions in RVA20U64.

- **Zifencei** Instruction-Fetch Fence.

  > Zifencei is mandated as it is the only standard way to support instruction-cache coherence in RVA20 application processors. A new instruction-cache coherence mechanism is under development which might be added as an option in the future.

The following privileged extensions are mandatory:

- **Ss1p11** Privileged Architecture version 1.11.
- **Svbare** The `satp` mode Bare must be supported.

  > This is a new extension name for this feature.

- **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.

  > This is a new extension name for this feature.

- **Ssccptr** Main memory regions with both the cacheability and coherence PMAs must support hardware page-table reads.

  > This is a new extension name for this feature.

- **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.

  > This is a new extension name for this feature.

- **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.

  > This is a new extension name for this feature.

5.2.3. RVA20S64 Optional Extensions

RVA20S64 has one unprivileged option.

- **Zihpm** Hardware performance counters.

  > The number of counters is platform-specific.

RVA20S64 has the following privileged options:

- **Sv48** Page-Based 48-bit Virtual-Memory System.
- **Ssu64xl** `sstatus.UXL` must be capable of holding the value 2 (i.e., UXLEN=64 must be supported).

  This is a new extension name for this feature.

Chapter 6. RVA22 Profiles

The RVA22 profiles are intended to be used for 64-bit application processors running rich OS stacks. Only user-mode (RVA22U64) and supervisor-mode (RVA22S64) profiles are specified in this family.

6.1. RVA22U64 Profile

The RVA22U64 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.

6.1.1. RVA22U64 Mandatory Base

RV64I is the mandatory base ISA for RVA22U64, including mandatory `fence.tso`, and is little-endian.

> Later versions of the RV64I unprivileged ISA specification ratified in 2021 made clear that `fence.tso` is mandatory.

As per the unprivileged architecture specification, the `ecall` instruction causes a requested trap to the execution environment.

6.1.2. RVA22U64 Mandatory Extensions

The following mandatory extensions were present in RVA20U64.

- **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 RVA22) are atomic.
- **Ziccrse** Main memory regions with both the cacheability and coherence PMAs must support RsrvEventual.

  > Ziccrse is a new extension name capturing this feature.

- **Ziccamoa** Main memory regions with both the cacheability and coherence PMAs must support AMOArithmetic.

Ziccamoa is a new extension name capturing this feature.

- **Zicclsm** Misaligned loads and stores to main memory regions with both the cacheability and coherence PMAs must be supported.

This is a new extension name for this feature. Even though mandated, misaligned loads and stores might execute extremely slowly. Standard software distributions should assume their existence only for correctness, not for performance.

The following mandatory feature was further restricted in RVA22U64:

- **Za64rs** Reservation sets are contiguous, naturally aligned, and a maximum of 64 bytes.

This is a new extension name capturing this feature. The maximum reservation size has been reduced to match the required cache block size. The minimum reservation size is effectively set by the instructions in the mandatory A extension.

The following mandatory extensions are new for RVA22U64.

- **Zihintpause** Pause instruction.

While the `pause` instruction is a HINT can be implemented as a NOP and hence trivially supported by hardware implementers, its inclusion in the mandatory extension list signifies that software should use the instruction whenever it would make sense and that implementors are expected to exploit this information to optimize hardware execution.

- **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.

This is a new extension name for this feature. While the general RISC-V specifications are agnostic to cache block size, selecting a common cache block size simplifies the specification and use of the following cache-block extensions within the application processor profile. Software does not have to query a discovery mechanism and/or provide dynamic dispatch to the appropriate code. We choose 64 bytes at it is effectively an industry standard. Implementations may use longer cache blocks to reduce tag cost provided they use 64-byte sub-blocks to remain compatible. Implementations may use shorter cache blocks provided they sequence cache operations across the multiple cache blocks comprising a 64-byte block to remain compatible.

- **Zicbom** Cache-Block Management Operations.
- **Zicbop** Cache-Block Prefetch Operations.

As with other HINTS, the inclusion of prefetches in the mandatory set of extensions indicates that software should generate these instructions where they are expected to be useful, and hardware is expected to exploit that information.

- **Zicboz** Cache-Block Zero Operations.
- **Zfhmin** Half-Precision Floating-point transfer and convert.

Zfhmin is a small extension that adds support to load/store and convert IEEE FP16 numbers to and from IEEE FP32 format. The hardware cost for this extension is low, and mandating the extension avoids adding an option to the profile.

- **Zkt** Data-independent execution time.

Zkt requires a certain subset of integer instructions execute with data-independent latency. Mandating this feature enables portable libraries for safe basic cryptographic operations. It is expected that application processors will naturally have this property and so implementation cost is low, if not zero, in most systems that would support RVA22.

6.1.3. RVA22U64 Optional Extensions

RVA22U64 has four profile options (Zfh, V, Zkn, Zks):

- **Zfh** Half-Precision Floating-Point.

A future profile might mandate Zfh.

- **V** Vector Extension.

The smaller vector extensions (Zve32f, Zve32x, Zve64d, Zve64f, Zve64x) are not provided as separately supported profile options. The full V extension is specified as the only supported profile option.

A future profile might mandate V.

- **Zkn** Scalar Crypto NIST Algorithms.
- **Zks** Scalar Crypto ShangMi Algorithms.

The scalar crypto extensions are expected to be superseded by vector crypto standards in future profiles, and the scalar extensions may be removed as supported options once vector crypto is present.

The smaller component scalar crypto extensions (Zbc, Zbkb, Zbkc, Zbkx, Zknd, Zkne, Zknh, Zksed, Zksh) are not provided as separate options in the profile. Profile implementers should provide all of the instructions in a given algorithm suite as part of the Zkn or Zks supported options.

Access to the entropy source (Zkr) in a system is usually carefully controlled. While the design supports unprivileged access to the entropy source, this is unlikely to be commonly used in an application processor, and so Zkr was not added as a profile option. This also means the roll-up Zk was not added as a profile option.

The Zfinx, Zdinx, Zhinx, Zhinxmin extensions are incompatible with the profile mandates to support the F and D extensions.

6.1.4. RVA22U64 Recommendations

Recommendations are not strictly mandated but are included to guide implementers making design choices.

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

6.2. RVA22S64 Profile

The RVA22S64 profile specifies the ISA features available to a supervisor-mode execution environment in 64-bit applications processors. RVA22S64 is based on privileged architecture version 1.12.

6.2.1. RVA22S64 Mandatory Base

RV64I is the mandatory base ISA for RVA22S64, including mandatory `fence.tso`, and is little-endian.

Later versions of the RV64I unprivileged ISA specification ratified in 2021 made clear that `fence.tso` is mandatory.

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.

6.2.2. RVA22S64 Mandatory Extensions

The following unprivileged extensions are mandatory:

- The RVA22S64 mandatory unprivileged extensions include all the mandatory unprivileged extensions in RVA22U64.
- **Zifencei** Instruction-Fetch Fence.

Zifencei is mandated as it is the only standard way to support instruction-cache coherence in RVA22 application processors. A new instruction-cache coherence mechanism is under development which might be added as an option in the future.

The following privileged extensions are mandatory:

- **Ss1p12** Privileged Architecture version 1.12.

  ℹ Ss1p12 supersedes Ss1p11.

- **Svbare** The `satp` mode Bare must be supported.

  ℹ This is a new extension name for this feature.

- **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.

  ℹ This is new extension name capturing this feature.

- **Svpbmt** Page-Based Memory Types
- **Svinval** Fine-Grained Address-Translation Cache Invalidation

6.2.3. RVA22S64 Optional Extensions

RVA22S64 has four unprivileged options (Zfh, V, Zkn, Zks) from RVA22U64, and eight privileged options (Sv48, Sv57, Svnapot, Ssu64xl, Sstc, Sscofpmf, Zkr, H).

The privileged optional extensions are:

- **Sv48** Page-Based 48-bit Virtual-Memory System.
- **Sv57** Page-Based 57-bit Virtual-Memory System.
- **Svnapot** NAPOT Translation Contiguity

  ℹ It is expected that Svnapot will be mandatory in the next profile release.

- **Ssu64xl** `sstatus.UXL` must be capable of holding the value 2 (i.e., UXLEN=64 must be supported).

  ℹ This is a new extension name for this feature.

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

Sstc was not made mandatory in RVA22S64 as it is a more disruptive change affecting system-level architecture, and will take longer for implementations to adopt. It is expected to be made mandatory in the next profile release.

- **Sscofpmf** Count Overflow and Mode-Based Filtering.

Platforms may choose to mandate the presence of Sscofpmf.

- **Zkr** Entropy CSR.

Technically, Zk is also a privileged-mode option capturing that Zkr, Zkn, and Zkt are all implemented. However, the Zk rollup is less descriptive than specifying the individual extensions explicitly.

- **H** The hypervisor extension.

When the hypervisor extension is implemented, the following are also mandatory:

- **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.

The Smstateen extension specification is an M-mode extension as it includes M-mode features, but the supervisor-mode visible components of the extension are named as the Ssstateen extension. Only Ssstateen is mandated in the RVA22S64 profile when the hypervisor extension is implemented. These registers are not mandated or supported options without the hypervisor extension, as there are no RVA22S64 supported options with relevant state to control in the absence of the hypervisor extension.

- **Shcounterenw** For any `hpmcounter` that is not read-only zero, the corresponding bit in `hcounteren` must be writable.

This is a new extension name for this feature.

- **Shvstvala** `vstval` must be written in all cases described above for `stval`.

This is a new extension name for this feature.

- **Shtvala** `htval` must be written with the faulting guest physical address in all circumstances permitted by the ISA.

This is a new extension name for this feature.

- **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.

This is a new extension name for this feature.

- **Shvsatpa** All translation modes supported in `satp` must be supported in `vsatp`.

  > ℹ This is a new extension name for this feature.

- **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.

  > ℹ This is a new extension name for this feature.

6.2.4. RVA22S64 Recommendations

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

Chapter 7. New ISA Extensions

This profile specification introduces the following new extension names for existing features, but none require new features:

- **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
- **Zicclsm**: Main memory supports misaligned loads/stores
- **Za64rs**: Reservation set size of 64 bytes
- **Za128rs**: Reservation set size of 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

Chapter 8. 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

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