[RISC-V International Logo] | *risc-v_logo.png*

RVA22 Profile Release

2023-04-03

Table of Contents

Licensing and Acknowledgements

TODO: revmark

Copyright and license information

Acknowledgements

Contributors to the RVA22 Profile (in alphabetical order) include:

- Krste Asanovic <krste@sifive.com> (SiFive)

We express our gratitude to everyone that contributed to, reviewed or improved this specification through their comments and questions.

Chapter 1. RISC-V Profiles

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.

1.1. 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.

1.2. Components of a Profile

1.2.1. 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.

1.2.2. 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.

1.2.3. 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.

Chapter 2. RVA Profile Class

The RVA profile class targets application processors for markets requiring a high-degree of binary compatibility between compliant implementations.

2.1. RVA Description

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.

2.2. RVA Naming Scheme

The profile class name is RVA (RISC-V Apps processor). A profile release name is an integer (currently 2 digits, could grow in the future). A full profile name is comprised of, in order:

- Prefix **RVA** for RISC-V Applications
- Profile release
- Privilege mode:
  - **U** Unprivileged (available to any privilege mode, **U** is **not** User-mode)
  - **S** Supervisor mode (note that Hypervisor support is treated as an option)
  - **M** Machine mode
- A base ISA XLEN specifier (**32**, **64**)

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.

2.3. RVA Profile Releases

The following profile releases are defined in this profile class:

Name
RVA20

State
ratified

Ratification date
2023-04-03

Name
RVA22

State
ratified

Ratification date
2023-04-03

2.4. Extension Presence

The RVA Profile Class references 42 extensions.

Table 1. Status

Extension	RVA20U64	RVA20S64	RVA22U64	RVA22S64
A	mandatory	-	mandatory	-
C	mandatory	-	mandatory	-
D	mandatory	-	mandatory	-
F	mandatory	-	mandatory	-
H	-	-	-	optional
I	mandatory	-	mandatory	-
M	mandatory	-	mandatory	-
S	-	mandatory	-	mandatory
Ssccptr	-	mandatory	-	mandatory
Sscofpmf	-	-	-	optional
Sscounterenw	-	-	-	mandatory
Sstc	-	-	-	optional
Sstvala	-	mandatory	-	mandatory
Sstvecd	-	mandatory	-	mandatory
Sv39	-	mandatory	-	mandatory
Sv48	-	optional	-	optional
Sv57	-	-	-	optional
Svade	-	mandatory	-	mandatory
Svbare	-	mandatory	-	mandatory
Svinval	-	-	-	mandatory
Svnapot	-	-	-	optional
Svpbmt	-	-	-	mandatory
U	mandatory	-	mandatory	-
V	-	-	optional	-
Za128rs	mandatory	-	mandatory	-
Zba	-	-	mandatory	-
Zbb	-	-	mandatory	-
Zbs	-	-	mandatory	-
Zfhmin	-	-	mandatory	-
Zic64b	-	-	mandatory	-
Zicbom	-	-	mandatory	-
Zicbop	-	-	mandatory	-
Zicboz	-	-	mandatory	-

Ziccamoa	mandatory	-	mandatory	-
Ziccif	mandatory	-	mandatory	-
Zicclsm	mandatory	-	mandatory	-
Ziccrse	mandatory	-	mandatory	-
Zicntr	mandatory	-	mandatory	-
Zifencei	-	mandatory	-	mandatory
Zihintpause	-	-	mandatory	-
Zihpm	optional	-	mandatory	-
Zkt	-	-	mandatory	-

Chapter 3. RVA22 Profile Release

This profile release targets 64-bit application processors for markets requiring a high-degree of binary compatibility between compliant implementations.

RVA22 has 102 associated implementation-defined parameters across all its defined profiles.

3.1. RVA22 Description

This profile release is intended to be used for 64-bit application processors running rich OS stacks. Only user-mode and supervisor-mode profiles are specified in this release.

There is no machine-mode profile currently defined for this release. 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 this release.

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

3.2. 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 application processors in terms of the amount of software that targets this profile.

3.2.1. Mandatory Extensions

The RVA22U64 Profile has 24 mandatory extensions.

- **Zihpm** Programmable hardware performance counters

  Version ~> 2.0

  The number of counters is platform-specific.

- **Zihintpause** PAUSE instruction

  Version ~> 2.0

  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 generation instructions

  Version ~> 1.0

- **Zbb** Basic bit manipulation

  Version ~> 1.0

- **Zbs** Single-bit instructions

  Version ~> 1.0

- **Zic64b** 64-byte cache blocks

  Version ~> 1.0

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 instructions

  Version ~> 1.0

- **Zicbop** Cache block prefetch

  Version ~> 1.0

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 instruction

  Version ~> 1.0

- **Zfhmin** Minimal half-precision Floating-point

  Version ~> 1.0

Zfhmin is a small extension that adds support to load/store and convert IEEE 754 half-precision numbers to and from the IEEE 754 single-precision format. The hardware cost for this extension is low, and mandating the extension

avoids adding an option to the profile.

- **Zkt** Data-independent execution latency

  Version ~> 1.0

  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.

- **A** Atomic instructions

  Version ~> 2.1

- **C** Compressed instructions

  Version ~> 2.2

- **D** Double-precision floating-point

  Version ~> 2.2

  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.

- **F** Single-precision floating-point

  Version ~> 2.2

- **M** Integer multiply and divide instructions

  Version ~> 2.0

- **U** User-level privilege mode

  Version ~> 2.0

- **Zicntr** Architectural performance counters

  Version = 2.0

- **Ziccif** Main memory fetch requirement for RVA profiles

  Version ~> 1.0

  Ziccif is a profile-defined extension introduced with RVA20. The fetch atomicity

requirement facilitates runtime patching of aligned instructions.

- **Ziccrse** Main memory reservability requirement for RVA profiles

  Version ~> 1.0

  Ziccrse is a profile-defined extension introduced with RVA20.

- **Ziccamoa** Main memory atomicity requirement for RVA profiles

  Version ~> 1.0

  Ziccamo is a profile-defined extension introduced with RVA20.

- **Za128rs** Reservation set requirement for RVA profiles

  Version ~> 1.0

  Za128rs is a profile-defined extension introduced with RVA20. The minimum reservation set size is effectively determined by the size of atomic accesses in the A extension.

- **Zicclsm** Main memory misaligned requirement for RVA profiles

  Version ~> 1.0

  Zicclsm is a profile-defined extension introduced with RVA20. 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.

- **I** Base integer ISA (RV32I or RV64I)

  Version ~> 2.1

  RVI is the mandatory base ISA for RVA, 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.

3.2.2. Optional Extensions

The RVA22U64 Profile has 4 optional extensions.

- **Zfh**

  Version ~> 1.0

  A future profile might mandate V.

- **V** Variable-length vector

  Version ~> 1.0

  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**

  Version ~> 1.0

- **Zks**

  Version ~> 1.0

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.

3.2.3. 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.

3.2.4. Implementation-dependencies

RVA22U64 has 24 associated implementation-defined parameters.

CACHE_BLOCK_SIZE

The observable size of a cache block, in bytes

FORCE_UPGRADE_CBO_INVAL_TO_FLUSH

When true, an implementation prohibits setting menvcfg.CBIE == 11 such that all cbo.inval instructions either trap (when menvcfg.CBIE == '00') or flush (when menvcfg.CBIE == '01').

When false, an implementation allows a true INVAL operation for cbo.inval, and thus supports the setting menvcfg.CBIE == 11.

CACHE_BLOCK_SIZE

The observable size of a cache block, in bytes

CACHE_BLOCK_SIZE

The observable size of a cache block, in bytes

HW_MSTATUS_VS_DIRTY_UPDATE

Indicates whether or not hardware will write to mstatus.VS

Values are:

never	Hardware never writes mstatus.VS
precise	Hardware writes mstatus.VS to the Dirty (3) state precisely when V registers are modified
imprecise	Hardware writes mstatus.VS imprecisely. This will result in a call to unpredictable() on any attempt to read mstatus or write vector state.

MSTATUS_VS_LEGAL_VALUES

The set of values that mstatus.VS will accept from a software write.

MUTABLE_MISA_V

Indicates whether or not the V extension can be disabled with the misa.V bit.

LRSC_FAIL_ON_NON_EXACT_LRSC

Whether or not a Store Conditional fails if its physical address and size do not exactly match the physical address and size of the last Load Reserved in program order (independent of whether or not the SC is in the current reservation set)

LRSC_FAIL_ON_VA_SYNONYM

Whether or not an `sc.l`/sc.d will fail if its VA does not match the VA of the prior `lr.l`/lr.d, even if the physical address of the SC and LR are the same

LRSC_MISALIGNED_BEHAVIOR

What to do when an LR/SC address is misaligned and MISALIGNED_AMO == false.

- 'always raise misaligned exception': self-explainitory
- 'always raise access fault': self-explainitory
- 'custom': Custom behavior; misaligned LR/SC may sometimes raise a misaligned exception and sometimes raise a access fault. Will lead to an 'unpredictable' call on any misaligned LR/SC access

LRSC_RESERVATION_STRATEGY

Strategy used to handle reservation sets.

- "reserve naturally-aligned 64-byte region": Always reserve the 64-byte block containing the LR/SC address
- "reserve naturally-aligned 128-byte region": Always reserve the 128-byte block containing the LR/SC address
- "reserve exactly enough to cover the access": Always reserve exactly the LR/SC access, and no more
- "custom": Custom behavior, leading to an 'unpredictable' call on any LR/SC

MISALIGNED_AMO

whether or not the implementation supports misaligned atomics in main memory

MUTABLE_MISA_A

When the A extensions is supported, indicates whether or not the extension can be disabled in the misa.A bit.

MUTABLE_MISA_C

Indicates whether or not the C extension can be disabled with the misa.C bit.

MUTABLE_MISA_D

Indicates whether or not the D extension can be disabled with the misa.D bit.

HW_MSTATUS_FS_DIRTY_UPDATE

Indicates whether or not hardware will write to mstatus.FS

Values are:

never	Hardware never writes mstatus.FS
precise	Hardware writes mstatus.FS to the Dirty (3) state precisely when F registers are modified
imprecise	Hardware writes mstatus.FS imprecisely. This will result in a call to unpredictable() on any attempt to read mstatus or write FP state.

MSTATUS_FS_LEGAL_VALUES

The set of values that mstatus.FS will accept from a software write.

MUTABLE_MISA_F

Indicates whether or not the F extension can be disabled with the misa.F bit.

MUTABLE_MISA_M

Indicates whether or not the M extension can be disabled with the misa.M bit.

MUTABLE_MISA_U

Indicates whether or not the U extension can be disabled with the misa.U bit.

TRAP_ON_ECALL_FROM_U

Whether or not an ECALL-from-U-mode causes a synchronous exception.

The spec states that implementations may handle ECALLs transparently without raising a trap, in which case the EEI must provide a builtin.

UXLEN

Set of XLENs supported in U-mode. Can be one of:

- 32: SXLEN is always 32
- 64: SXLEN is always 64
- 3264: SXLEN can be changed (via mstatus.UXL) between 32 and 64

U_MODE_ENDIANESS

Endianess of data in U-mode. Can be one of:

- little: M-mode data is always little endian
- big: M-mode data is always big endian
- dynamic: M-mode data can be either little or big endian, depending on the CSR field mstatus.UBE

TIME_CSR_IMPLEMENTED

Whether or not a real hardware time CSR exists. Implementations can either provide a real CSR or emulate access at M-mode.

Possible values:

true

time/timeh exists, and accessing it will not cause an IllegalInstruction trap

false

time/timeh does not exist. Accessing the CSR will cause an IllegalInstruction trap or enter an unpredictable state, depending on TRAP_ON_UNIMPLEMENTED_CSR. Privileged software may emulate the time CSR, or may pass the exception to a lower level.

3.3. 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.

3.3.1. Mandatory Extensions

The RVA22S64 Profile has 16 mandatory extensions.

- **S** Supervisor mode

  Version ~> 1.12

- **Sscounterenw** Supervisor counter enable

  Version ~> 1.0

  Sstvala is a new extension name introduced with RVA22.

- **Svpbmt** Page-based memory types

  Version ~> 1.0

- **Svinval** Fine-grained address-translation cache invalidation

  Version ~> 1.0

- **Ssstateen**

  Version ~> 1.0

  Ssstateen is a new extension name introduced with RVA22.

- **Shvstvala**

Version ~> 1.0

Shvstvala is a new extension name introduced with RVA22.

- **Shtvala**

  Version ~> 1.0

  Shtvala is a new extension name introduced with RVA22.

- **Shvstvecd**

  Version ~> 1.0

  Shvstvecd is a new extension name introduced with RVA22.

- **Shgatpa**

  Version ~> 1.0

  Shgatpa is a new extension name introduced with RVA22.

- **Zifencei** Instruction fence

  Version ~> 2.0

  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.

- **Svbare** Bare virtual addressing

  Version ~> 1.0

  Svbare is a new extension name introduced with RVA20.

- **Sv39** 39-bit virtual address translation (3 level)

  Version ~> 1.11

- **Svade** Exception on PTE A/D Bits

  Version ~> 1.0

  Svbare is a new extension name introduced with RVA20.

  It is subsequently defined in more detail with the ratification of Svadu.

- **Ssccptr** Cacheable and coherent main memory page table reads

Version ~> 1.0

Ssccptr is a new extension name introduced with RVA20.

- **Sstvecd** Direct exception vectoring

  Version ~> 1.0

Sstvecd is a new extension name introduced with RVA20.

- **Sstvala** stval requirements for RVA profiles

  Version ~> 1.0

Sstvala is a new extension name introduced with RVA20.

3.3.2. Optional Extensions

The RVA22S64 Profile has 8 optional extensions.

- **Sv57** 57-bit virtual address translation (5 level)

  Version ~> 1.12

- **Svnapot** Naturally-aligned Power of Two Translation Contiguity

  Version ~> 1.0

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

- **Sstc** Superivisor mode timer interrupts

  Version ~> 1.0

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** Counter Overflow and Privilege Mode Filtering

  Version ~> 1.0

Platforms may choose to mandate the presence of Sscofpmf.

- **Zkr**

  Version ~> 1.0

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** Hypervisor

  Version ~> 1.0

  The following extensions become mandatory when H is implemented:

  - Ssstateen
  - Shcounterenw
  - Shvstvala
  - Shtvala
  - Shvstvecd
  - Shgatpa

- **Sv48** 48-bit virtual address translation (4 level)

  Version ~> 1.11

- **Ssu64xl**

  Version ~> 1.0

  Ssu64xl is a new extension name introduced with RVA20.

3.3.3. 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.

3.3.4. Implementation-dependencies

RVA22S64 has 78 associated implementation-defined parameters.

ASID_WIDTH
: Number of implemented ASID bits. Maximum is 16 for XLEN==64, and 9 for XLEN==32

MSTATUS_FS_LEGAL_VALUES
: The set of values that mstatus.FS will accept from a software write.

MSTATUS_FS_WRITEABLE
: When S is enabled but F is not, mstatus.FS is optionally writeable.

This parameter only has an effect when both S and F mode are disabled.

MSTATUS_TVM_IMPLEMENTED

Whether or not mstatus.TVM is implemented.

When not implemented mstatus.TVM will be read-only-zero.

MSTATUS_VS_LEGAL_VALUES

The set of values that mstatus.VS will accept from a software write.

MSTATUS_VS_WRITEABLE

When S is enabled but V is not, mstatus.VS is optionally writeable.

This parameter only has an effect when both S and V mode are disabled.

MUTABLE_MISA_S

Indicates whether or not the S extension can be disabled with the misa.S bit.

REPORT_ENCODING_IN_STVAL_ON_ILLEGAL_INSTRUCTION

When true, stval is written with the encoding of an instruction that causes an `IllegalInstruction` exception.

When false stval is written with 0 when an `IllegalInstruction` exception occurs.

REPORT_VA_IN_STVAL_ON_BREAKPOINT

When true, stval is written with the virtual PC of the EBREAK instruction (same information as mepc).

When false, stval is written with 0 on an EBREAK instruction.

Regardless, stval is always written with a virtual PC when an external breakpoint is generated

REPORT_VA_IN_STVAL_ON_INSTRUCTION_ACCESS_FAULT

When true, stval is written with the virtual PC of an instructino when fetch causes an `InstructionAccessFault`.

WHen false, stval is written with 0 when an instruction fetch causes an `InstructionAccessFault`.

REPORT_VA_IN_STVAL_ON_INSTRUCTION_MISALIGNED

When true, stval is written with the virtual PC when an instruction fetch is misaligned.

When false, stval is written with 0 when an instruction fetch is misaligned.

Note that when IALIGN=16 (i.e., when the C or one of the `Zc*` extensions are implemented), it is impossible to generate a misaligned fetch, and so this parameter has no effect.

REPORT_VA_IN_STVAL_ON_INSTRUCTION_PAGE_FAULT

When true, stval is written with the virtual PC of an instructino when fetch causes an `InstructionPageFault`.

WHen false, stval is written with 0 when an instruction fetch causes an `InstructionPageFault`.

REPORT_VA_IN_STVAL_ON_LOAD_ACCESS_FAULT

When true, stval is written with the virtual address of a load when it causes a `LoadAccessFault`.

WHen false, stval is written with 0 when a load causes a `LoadAccessFault`.

REPORT_VA_IN_STVAL_ON_LOAD_MISALIGNED

When true, stval is written with the virtual address of a load instruction when the address is misaligned and MISALIGNED_LDST is false.

When false, stval is written with 0 when a load address is misaligned and MISALIGNED_LDST is false.

REPORT_VA_IN_STVAL_ON_LOAD_PAGE_FAULT

When true, stval is written with the virtual address of a load when it causes a `LoadPageFault`.

WHen false, stval is written with 0 when a load causes a `LoadPageFault`.

REPORT_VA_IN_STVAL_ON_STORE_AMO_ACCESS_FAULT

When true, stval is written with the virtual address of a store when it causes a `StoreAmoAccessFault`.

WHen false, stval is written with 0 when a store causes a `StoreAmoAccessFault`.

REPORT_VA_IN_STVAL_ON_STORE_AMO_MISALIGNED

When true, stval is written with the virtual address of a store instruction when the address is misaligned and MISALIGNED_LDST is false.

When false, stval is written with 0 when a store address is misaligned and MISALIGNED_LDST is false.

REPORT_VA_IN_STVAL_ON_STORE_AMO_PAGE_FAULT

When true, stval is written with the virtual address of a store when it causes a `StoreAmoPageFault`.

WHen false, stval is written with 0 when a store causes a `StoreAmoPageFault`.

SATP_MODE_BARE

Whether or not satp.MODE == Bare is supported.

SCOUNTENABLE_EN

Indicates which counters can delegated via scounteren

An unimplemented counter cannot be specified, i.e., if HPM_COUNTER_EN[3] is false, it would be illegal to set SCOUNTENABLE_EN[3] to true.

SCOUNTENABLE_EN[0:2] must all be false if Zicntr is not implemented. SCOUNTENABLE_EN[3:31] must all be false if Zihpm is not implemented.

STVAL_WIDTH

The number of implemented bits in stval.

Must be greater than or equal to *max*(PHYS_ADDR_WIDTH, VA_SIZE)

STVEC_MODE_DIRECT

Whether or not stvec.MODE supports Direct (0).

STVEC_MODE_VECTORED

Whether or not stvec.MODE supports Vectored (1).

SV_MODE_BARE

Whether or not writing mode=Bare is supported in the satp register.

SXLEN

Set of XLENs supported in S-mode. Can be one of:

- 32: SXLEN is always 32
- 64: SXLEN is always 64
- 3264: SXLEN can be changed (via mstatus.SXL) between 32 and 64

S_MODE_ENDIANESS

Endianess of data in S-mode. Can be one of:

- little: M-mode data is always little endian
- big: M-mode data is always big endian
- dynamic: M-mode data can be either little or big endian, depending on the CSR field mstatus.SBE

TRAP_ON_ECALL_FROM_S

Whether or not an ECALL-from-S-mode causes a synchronous exception.

The spec states that implementations may handle ECALLs transparently without raising a trap, in which case the EEI must provide a builtin.

TRAP_ON_SFENCE_VMA_WHEN_SATP_MODE_IS_READ_ONLY

For implementations that make satp.MODE read-only zero (always Bare, *i.e.*, no virtual translation is implemented), attempts to execute an SFENCE.VMA instruction might raise an illegal-instruction exception.

TRAP_ON_SFENCE_VMA_WHEN_SATP_MODE_IS_READ_ONLY indicates whether or not that exception occurs.

TRAP_ON_SFENCE_VMA_WHEN_SATP_MODE_IS_READ_ONLY has no effect when some virtual translation mode is supported.

GSTAGE_MODE_BARE

Whether or not writing mode=Bare is supported in the hgatp register.

HCOUNTENABLE_EN

Indicates which counters can delegated via hcounteren

An unimplemented counter cannot be specified, i.e., if HPM_COUNTER_EN[3] is false, it would be illegal to set HCOUNTENABLE_EN[3] to true.

HCOUNTENABLE_EN[0:2] must all be false if Zicntr is not implemented. HCOUNTENABLE_EN[3:31] must all be false if Zihpm is not implemented.

IGNORE_INVALID_VSATP_MODE_WRITES_WHEN_V_EQ_ZERO

Whether writes from M-mode, U-mode, or S-mode to vsatp with an illegal mode setting are ignored (as they are with satp), or if they are treated as WARL, leading to undpredictable behavior.

MUTABLE_MISA_H

Indicates whether or not the H extension can be disabled with the misa.H bit.

NUM_EXTERNAL_GUEST_INTERRUPTS

Number of supported virtualized guest interrupts

Corresponds to the `GEILEN` parameter in the RVI specs

REPORT_ENCODING_IN_VSTVAL_ON_ILLEGAL_INSTRUCTION

When true, vstval is written with the encoding of an instruction that causes an `IllegalInstruction` exception.

When false vstval is written with 0 when an `IllegalInstruction` exception occurs.

REPORT_GPA_IN_TVAL_ON_INSTRUCTION_GUEST_PAGE_FAULT

Whether or not GPA >> 2 is written into htval/mtval2 when an instruction guest page fault occurs.

If false, 0 will be written into htval/mtval2 on an instruction guest page fault.

REPORT_GPA_IN_TVAL_ON_INTERMEDIATE_GUEST_PAGE_FAULT

Whether or not GPA >> 2 is written into htval/mtval2 when a guest page fault occurs while walking a VS-mode page table.

If false, 0 will be written into htval/mtval2 on an intermediate guest page fault.

REPORT_GPA_IN_TVAL_ON_LOAD_GUEST_PAGE_FAULT

Whether or not GPA >> 2 is written into htval/mtval2 when a load guest page fault occurs.

If false, 0 will be written into htval/mtval2 on a load guest page fault.

REPORT_GPA_IN_TVAL_ON_STORE_AMO_GUEST_PAGE_FAULT

Whether or not GPA >> 2 is written into htval/mtval2 when a store/amo guest page fault occurs.

If false, 0 will be written into htval/mtval2 on a store/amo guest page fault.

REPORT_VA_IN_VSTVAL_ON_BREAKPOINT

When true, vstval is written with the virtual PC of the EBREAK instruction (same information as mepc).

When false, vstval is written with 0 on an EBREAK instruction.

Regardless, vstval is always written with a virtual PC when an external breakpoint is generated

REPORT_VA_IN_VSTVAL_ON_INSTRUCTION_ACCESS_FAULT

When true, vstval is written with the virtual PC of an instructino when fetch causes an `InstructionAccessFault`.

WHen false, vstval is written with 0 when an instruction fetch causes an `InstructionAccessFault`.

REPORT_VA_IN_VSTVAL_ON_INSTRUCTION_MISALIGNED

When true, vstval is written with the virtual PC when an instruction fetch is misaligned.

When false, vstval is written with 0 when an instruction fetch is misaligned.

Note that when IALIGN=16 (i.e., when the C or one of the `Zc*` extensions are implemented), it is impossible to generate a misaligned fetch, and so this parameter has no effect.

REPORT_VA_IN_VSTVAL_ON_INSTRUCTION_PAGE_FAULT

When true, vstval is written with the virtual PC of an instructino when fetch causes an `InstructionPageFault`.

WHen false, vstval is written with 0 when an instruction fetch causes an `InstructionPageFault`.

REPORT_VA_IN_VSTVAL_ON_LOAD_ACCESS_FAULT

When true, vstval is written with the virtual address of a load when it causes a `LoadAccessFault`.

WHen false, vstval is written with 0 when a load causes a `LoadAccessFault`.

REPORT_VA_IN_VSTVAL_ON_LOAD_MISALIGNED

When true, vstval is written with the virtual address of a load instruction when the address is misaligned and MISALIGNED_LDST is false.

When false, vstval is written with 0 when a load address is misaligned and MISALIGNED_LDST is false.

REPORT_VA_IN_VSTVAL_ON_LOAD_PAGE_FAULT

When true, vstval is written with the virtual address of a load when it causes a `LoadPageFault`.

WHen false, vstval is written with 0 when a load causes a `LoadPageFault`.

REPORT_VA_IN_VSTVAL_ON_STORE_AMO_ACCESS_FAULT

When true, vstval is written with the virtual address of a store when it causes a `StoreAmoAccessFault`.

WHen false, vstval is written with 0 when a store causes a `StoreAmoAccessFault`.

REPORT_VA_IN_VSTVAL_ON_STORE_AMO_MISALIGNED

When true, vstval is written with the virtual address of a store instruction when the address is misaligned and MISALIGNED_LDST is false.

When false, vstval is written with 0 when a store address is misaligned and MISALIGNED_LDST is false.

REPORT_VA_IN_VSTVAL_ON_STORE_AMO_PAGE_FAULT

When true, vstval is written with the virtual address of a store when it causes a `StoreAmoPageFault`.

WHen false, vstval is written with 0 when a store causes a `StoreAmoPageFault`.

SV32X4_TRANSLATION

Whether or not Sv32x4 translation mode is supported.

SV32_VSMODE_TRANSLATION

Whether or not Sv32 translation is supported in first-stage (VS-stage) translation.

SV39X4_TRANSLATION

Whether or not Sv39x4 translation mode is supported.

SV39_VSMODE_TRANSLATION

Whether or not Sv39 translation is supported in first-stage (VS-stage) translation.

SV48X4_TRANSLATION

Whether or not Sv48x4 translation mode is supported.

SV48_VSMODE_TRANSLATION

Whether or not Sv48 translation is supported in first-stage (VS-stage) translation.

SV57X4_TRANSLATION

Whether or not Sv57x4 translation mode is supported.

SV57_VSMODE_TRANSLATION

Whether or not Sv57 translation is supported in first-stage (VS-stage) translation.

TINST_VALUE_ON_BREAKPOINT

Value written into htinst/mtinst on a Breakpoint exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_FINAL_INSTRUCTION_GUEST_PAGE_FAULT

Value to write into htval/mtval2 when there is a guest page fault on a final translation.

Possible values: * "always zero": Always write the value zero * "always pseudoinstruction": Always write the pseudoinstruction

TINST_VALUE_ON_FINAL_LOAD_GUEST_PAGE_FAULT

Value to write into htval/mtval2 when there is a guest page fault on a final translation.

Possible values: * "always zero": Always write the value zero * "always pseudoinstruction": Always write the pseudoinstruction * "always transformed standard instruction": Always write the transformation of the standard instruction encoding * "custom": A custom value, which will cause an UNPREDICTABLE event.

TINST_VALUE_ON_FINAL_STORE_AMO_GUEST_PAGE_FAULT

Value to write into htval/mtval2 when there is a guest page fault on a final translation.

Possible values: * "always zero": Always write the value zero * "always pseudoinstruction": Always write the pseudoinstruction * "always transformed standard instruction": Always write the transformation of the standard instruction encoding * "custom": A custom value, which will cause an UNPREDICTABLE event.

TINST_VALUE_ON_INSTRUCTION_ADDRESS_MISALIGNED

Value written into htinst/mtinst when there is an instruction address misaligned exception.

Possible values: * "always zero": Always write the value zero * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_LOAD_ACCESS_FAULT

Value written into htinst/mtinst on an AccessFault exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "always transformed standard instruction": Always write a transformed standard instruction as defined by H * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_LOAD_ADDRESS_MISALIGNED

Value written into htinst/mtinst on a VirtualInstruction exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "always transformed standard instruction": Always write a transformed standard instruction as defined by H * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_LOAD_PAGE_FAULT

Value written into htinst/mtinst on a LoadPageFault exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "always transformed standard instruction": Always write a transformed standard instruction as defined by H * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_MCALL

Value written into htinst/mtinst on a MCall exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_SCALL

Value written into htinst/mtinst on a SCall exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_STORE_AMO_ACCESS_FAULT

Value written into htinst/mtinst on an AccessFault exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "always transformed standard instruction": Always write a transformed standard instruction as defined by H * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_STORE_AMO_ADDRESS_MISALIGNED

Value written into htinst/mtinst on a VirtualInstruction exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "always transformed standard instruction": Always write a transformed standard instruction as defined by H * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_STORE_AMO_PAGE_FAULT

Value written into htinst/mtinst on a StoreAmoPageFault exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "always transformed standard instruction": Always write a transformed standard instruction as defined by H * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_UCALL

Value written into htinst/mtinst on a UCall exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_VIRTUAL_INSTRUCTION

Value written into htinst/mtinst on a VirtualInstruction exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_VSCALL

Value written into htinst/mtinst on a VSCall exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "custom": Write a custom value, which resuls in UNPREDICTABLE

TRAP_ON_ECALL_FROM_VS

Whether or not an ECALL-from-VS-mode causes a synchronous exception.

The spec states that implementations may handle ECALLs transparently without raising a trap, in which case the EEI must provide a builtin.

VMID_WIDTH

Number of bits supported in hgatp.VMID (i.e., the supported width of a virtual machine ID).

VSXLEN

Set of XLENs supported in VS-mode. Can be one of:

- 32: VSXLEN is always 32
- 64: VSXLEN is always 64
- 3264: VSXLEN can be changed (via hstatus.VSXL) between 32 and 64

VS_MODE_ENDIANESS

Endianess of data in VS-mode. Can be one of:

- little: M-mode data is always little endian
- big: M-mode data is always big endian
- dynamic: M-mode data can be either little or big endian, depending on the CSR field hstatus.VSBE

VUXLEN

Set of XLENs supported in VU-mode. Can be one of:

- 32: VUXLEN is always 32
- 64: VUXLEN is always 64
- 3264: VUXLEN can be changed (via vsstatus.UXL) between 32 and 64

VU_MODE_ENDIANESS

Endianess of data in VU-mode. Can be one of:

- little: M-mode data is always little endian
- big: M-mode data is always big endian
- dynamic: M-mode data can be either little or big endian, depending on the CSR field vsstatus.UBE

Appendix A: Extension Details

A.1. A Extension

Atomic instructions

Table 2. Status

Profile	v2.1.0
RVA22U64	mandatory
RVA22S64	-

2.1.0

Ratification date

2019-12

A.1.1. Synopsis

The atomic-instruction extension, named A, contains instructions that atomically read-modify-write memory to support synchronization between multiple RISC-V harts running in the same memory space. The two forms of atomic instruction provided are load-reserved/store-conditional instructions and atomic fetch-and-op memory instructions. Both types of atomic instruction support various memory consistency orderings including unordered, acquire, release, and sequentially consistent semantics. These instructions allow RISC-V to support the RCsc memory consistency model. cite:[Gharachorloo90memoryconsistency]

After much debate, the language community and architecture community appear to have finally settled on release consistency as the standard memory consistency model and so the RISC-V atomic support is built around this model.

The A extension comprises instructions provided by the Zaamo and Zalrsc extensions.

A.1.2. Specifying Ordering of Atomic Instructions

The base RISC-V ISA has a relaxed memory model, with the `FENCE` instruction used to impose additional ordering constraints. The address space is divided by the execution environment into memory and I/O domains, and the `FENCE` instruction provides options to order accesses to one or both of these two address domains.

To provide more efficient support for release consistency cite:[Gharachorloo90memoryconsistency], each atomic instruction has two bits, *aq* and *rl*, used to specify additional memory ordering constraints as viewed by other RISC-V harts. The bits order accesses to one of the two address domains, memory or I/O, depending on which address domain the atomic instruction is accessing. No ordering constraint is implied to accesses to the other domain, and a FENCE instruction should be used to order across both domains.

If both bits are clear, no additional ordering constraints are imposed on the atomic memory operation. If only the *aq* bit is set, the atomic memory operation is treated as an *acquire* access, i.e., no following memory operations on this RISC-V hart can be observed to take place before the acquire memory operation. If only the *rl* bit is set, the atomic memory operation is treated as a

release access, i.e., the release memory operation cannot be observed to take place before any earlier memory operations on this RISC-V hart. If both the *aq* and *rl* bits are set, the atomic memory operation is *sequentially consistent* and cannot be observed to happen before any earlier memory operations or after any later memory operations in the same RISC-V hart and to the same address domain.

A.1.3. Instructions

The following instructions are added by this extension:

amoadd.d	**Atomic fetch-and-add doubleword**
amoadd.w	**Atomic fetch-and-add word**
amoand.d	**Atomic fetch-and-and doubleword**
amoand.w	**Atomic fetch-and-and word**
amomax.d	**Atomic MAX doubleword**
amomax.w	**Atomic MAX word**
amomaxu.d	**Atomic MAX unsigned doubleword**
amomaxu.w	**Atomic MAX unsigned word**
amomin.d	**Atomic MIN doubleword**
amomin.w	**Atomic MIN word**
amominu.d	**Atomic MIN unsigned doubleword**
amominu.w	**Atomic MIN unsigned word**
amoor.d	**Atomic fetch-and-or doubleword**
amoor.w	**Atomic fetch-and-or word**
amoswap.d	**Atomic SWAP doubleword**
amoswap.w	**Atomic SWAP word**
amoxor.d	**Atomic fetch-and-xor doubleword**
amoxor.w	**Atomic fetch-and-xor word**
lr.d	**Load reserved doubleword**
lr.w	**Load reserved word**
sc.d	**Store conditional doubleword**
sc.w	**Store conditional word**

A.1.4. Parameters

This extension has the following implementation options:

LRSC_FAIL_ON_NON_EXACT_LRSC

Whether or not a Store Conditional fails if its physical address and size do not exactly match the physical address and size of the last Load Reserved in program order (independent of whether

or not the SC is in the current reservation set)

LRSC_FAIL_ON_VA_SYNONYM

Whether or not an `sc.l`/sc.d will fail if its VA does not match the VA of the prior `lr.l`/lr.d, even if the physical address of the SC and LR are the same

LRSC_MISALIGNED_BEHAVIOR

What to do when an LR/SC address is misaligned and MISALIGNED_AMO == false.

- 'always raise misaligned exception': self-explainitory
- 'always raise access fault': self-explainitory
- 'custom': Custom behavior; misaligned LR/SC may sometimes raise a misaligned exception and sometimes raise a access fault. Will lead to an 'unpredictable' call on any misaligned LR/SC access

LRSC_RESERVATION_STRATEGY

Strategy used to handle reservation sets.

- "reserve naturally-aligned 64-byte region": Always reserve the 64-byte block containing the LR/SC address
- "reserve naturally-aligned 128-byte region": Always reserve the 128-byte block containing the LR/SC address
- "reserve exactly enough to cover the access": Always reserve exactly the LR/SC access, and no more
- "custom": Custom behavior, leading to an 'unpredictable' call on any LR/SC

MISALIGNED_AMO

whether or not the implementation supports misaligned atomics in main memory

MUTABLE_MISA_A

When the A extensions is supported, indicates whether or not the extension can be disabled in the misa.A bit.

A.2. C Extension

Compressed instructions

Table 3. Status

Profile	v2.2.0
RVA22U64	mandatory
RVA22S64	-

2.2.0

Ratification date

2019-12

A.2.1. Synopsis

The C extension reduces static and dynamic code size by adding short 16-bit instruction encodings for common operations. The C extension can be added to any of the base ISAs (RV32, RV64, RV128), and we use the generic term "RVC" to cover any of these. 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.

A.2.2. Overview

RVC uses a simple compression scheme that offers shorter 16-bit versions of common 32-bit RISC-V instructions when:

- the immediate or address offset is small, or
- one of the registers is the zero register (`x0`), the ABI link register (`x1`), or the ABI stack pointer (`x2`), or
- the destination register and the first source register are identical, or
- the registers used are the 8 most popular ones.

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 now able to start on any 16-bit boundary, i.e., IALIGN=16. With the addition of the C extension, no instructions can raise instruction-address-misaligned exceptions.

Removing the 32-bit alignment constraint on the original 32-bit instructions allows significantly greater code density.

The compressed instruction encodings are mostly common across RV32C, RV64C, and RV128C, but as shown in Table 34, a few opcodes are used for different purposes depending on base ISA. For example, the wider address-space RV64C and RV128C variants require additional opcodes to compress loads and stores of 64-bit integer values, while RV32C uses the same opcodes to compress loads and stores of single-precision floating-point values. Similarly, RV128C requires additional opcodes to capture loads and stores of 128-bit integer values, while these same opcodes are used for

loads and stores of double-precision floating-point values in RV32C and RV64C. If the C extension is implemented, the appropriate compressed floating-point load and store instructions must be provided whenever the relevant standard floating-point extension (F and/or D) is also implemented. In addition, RV32C includes a compressed jump and link instruction to compress short-range subroutine calls, where the same opcode is used to compress ADDIW for RV64C and RV128C.

Double-precision loads and stores are a significant fraction of static and dynamic instructions, hence the motivation to include them in the RV32C and RV64C encoding.

Although single-precision loads and stores are not a significant source of static or dynamic compression for benchmarks compiled for the currently supported ABIs, for microcontrollers that only provide hardware single-precision floating-point units and have an ABI that only supports single-precision floating-point numbers, the single-precision loads and stores will be used at least as frequently as double-precision loads and stores in the measured benchmarks. Hence, the motivation to provide compressed support for these in RV32C.

Short-range subroutine calls are more likely in small binaries for microcontrollers, hence the motivation to include these in RV32C.

Although reusing opcodes for different purposes for different base ISAs adds some complexity to documentation, the impact on implementation complexity is small even for designs that support multiple base ISAs. The compressed floating-point load and store variants use the same instruction format with the same register specifiers as the wider integer loads and stores.

RVC was designed under the constraint that each RVC instruction expands into a single 32-bit instruction in either the base ISA (RV32I/E, RV64I/E, or RV128I) or the F and D standard extensions where present. Adopting this constraint has two main benefits:

- Hardware designs can simply expand RVC instructions during decode, simplifying verification and minimizing modifications to existing microarchitectures.
- Compilers can be unaware of the RVC extension and leave code compression to the assembler and linker, although a compression-aware compiler will generally be able to produce better results.

We felt the multiple complexity reductions of a simple one-one mapping between C and base IFD instructions far outweighed the potential gains of a slightly denser encoding that added additional instructions only supported in the C extension, or that allowed encoding of multiple IFD instructions in one C instruction.

It is important to note that the C extension is not designed to be a stand-alone ISA, and is meant to be used alongside a base ISA.

Variable-length instruction sets have long been used to improve code density. For example, the IBM Stretch cite:[stretch], developed in the late 1950s, had an ISA

with 32-bit and 64-bit instructions, where some of the 32-bit instructions were compressed versions of the full 64-bit instructions. Stretch also employed the concept of limiting the set of registers that were addressable in some of the shorter instruction formats, with short branch instructions that could only refer to one of the index registers. The later IBM 360 architecture cite:[ibm360] supported a simple variable-length instruction encoding with 16-bit, 32-bit, or 48-bit instruction formats.

In 1963, CDC introduced the Cray-designed CDC 6600 cite:[cdc6600], a precursor to RISC architectures, that introduced a register-rich load-store architecture with instructions of two lengths, 15-bits and 30-bits. The later Cray-1 design used a very similar instruction format, with 16-bit and 32-bit instruction lengths.

The initial RISC ISAs from the 1980s all picked performance over code size, which was reasonable for a workstation environment, but not for embedded systems. Hence, both ARM and MIPS subsequently made versions of the ISAs that offered smaller code size by offering an alternative 16-bit wide instruction set instead of the standard 32-bit wide instructions. The compressed RISC ISAs reduced code size relative to their starting points by about 25-30%, yielding code that was significantly smaller than 80x86. This result surprised some, as their intuition was that the variable-length CISC ISA should be smaller than RISC ISAs that offered only 16-bit and 32-bit formats.

Since the original RISC ISAs did not leave sufficient opcode space free to include these unplanned compressed instructions, they were instead developed as complete new ISAs. This meant compilers needed different code generators for the separate compressed ISAs. The first compressed RISC ISA extensions (e.g., ARM Thumb and MIPS16) used only a fixed 16-bit instruction size, which gave good reductions in static code size but caused an increase in dynamic instruction count, which led to lower performance compared to the original fixed-width 32-bit instruction size. This led to the development of a second generation of compressed RISC ISA designs with mixed 16-bit and 32-bit instruction lengths (e.g., ARM Thumb2, microMIPS, PowerPC VLE), so that performance was similar to pure 32-bit instructions but with significant code size savings. Unfortunately, these different generations of compressed ISAs are incompatible with each other and with the original uncompressed ISA, leading to significant complexity in documentation, implementations, and software tools support.

Of the commonly used 64-bit ISAs, only PowerPC and microMIPS currently supports a compressed instruction format. It is surprising that the most popular 64-bit ISA for mobile platforms (ARM v8) does not include a compressed instruction format given that static code size and dynamic instruction fetch bandwidth are important metrics. Although static code size is not a major concern in larger systems, instruction fetch bandwidth can be a major bottleneck in servers running commercial workloads, which often have a large instruction working set.

Benefiting from 25 years of hindsight, RISC-V was designed to support compressed instructions from the outset, leaving enough opcode space for RVC to be added as a

simple extension on top of the base ISA (along with many other extensions). The philosophy of RVC is to reduce code size for embedded applications *and* to improve performance and energy-efficiency for all applications due to fewer misses in the instruction cache. Waterman shows that RVC fetches 25%-30% fewer instruction bits, which reduces instruction cache misses by 20%-25%, or roughly the same performance impact as doubling the instruction cache size. cite:[waterman-ms]

A.2.3. Compressed Instruction Formats

Table 4 shows the nine compressed instruction formats. CR, CI, and CSS can use any of the 32 RVI registers, but CIW, CL, CS, CA, and CB are limited to just 8 of them. Table 5 lists these popular registers, which correspond to registers `x8` to `x15`. Note that there is a separate version of load and store instructions that use the stack pointer as the base address register, since saving to and restoring from the stack are so prevalent, and that they use the CI and CSS formats to allow access to all 32 data registers. CIW supplies an 8-bit immediate for the ADDI4SPN instruction.

The RISC-V ABI was changed to make the frequently used registers map to registers 'x8-x15'. This simplifies the decompression decoder by having a contiguous naturally aligned set of register numbers, and is also compatible with the RV32E and RV64E base ISAs, which only have 16 integer registers.

Compressed register-based floating-point loads and stores also use the CL and CS formats respectively, with the eight registers mapping to `f8` to `f15`.

The standard RISC-V calling convention maps the most frequently used floating-point registers to registers `f8` to `f15`, which allows the same register decompression decoding as for integer register numbers.

The formats were designed to keep bits for the two register source specifiers in the same place in all instructions, while the destination register field can move. When the full 5-bit destination register specifier is present, it is in the same place as in the 32-bit RISC-V encoding. Where immediates are sign-extended, the sign extension is always from bit 12. Immediate fields have been scrambled, as in the base specification, to reduce the number of immediate muxes required.

The immediate fields are scrambled in the instruction formats instead of in sequential order so that as many bits as possible are in the same position in every instruction, thereby simplifying implementations.

For many RVC instructions, zero-valued immediates are disallowed and `x0` is not a valid 5-bit register specifier. These restrictions free up encoding space for other instructions requiring fewer operand bits.

Table 4. Compressed 16-bit RVC instruction formats

Format	Meaning	15 14 13 12		11 10 9 8 7	6 5 4 3 2		1 0
CR	Register	funct4		rd/rs1	rs2		op
CI	Immediate	funct3	imm	rd/rs1	imm		op
CSS	Stack-relative Store	funct3	imm		rs2		op
CIW	Wide Immediate	funct3	imm			rd'	op
CL	Load	funct3	imm	rs1'	imm	rd'	op
CS	Store	funct3	imm	rs1'	imm	rs2'	op
CA	Arithmetic	funct6		rd'/rs1'	funct2	rs2'	op
CB	Branch/Arithmetic	funct3	offset	rd'/rs1'	offset		op
CJ	Jump	funct3	jump target				op

Table 5. Registers specified by the three-bit rs1', rs2', and rd' fields of the CIW, CL, CS, CA, and CB formats.

RVC Register Number	000	001	010	011	100	101	110	111
Integer Register Number	x8	x9	x10	x11	x12	x13	x14	x15
Integer Register ABI Name	s0	s1	a0	a1	a2	a3	a4	a5
Floating-Point Register Number	f8	f9	f10	f11	f12	f13	f14	f15
Floating-Point Register ABI Name	fs0	fs1	fa0	fa1	fa2	fa3	fa4	fa5

A.2.4. Instructions

The following instructions are added by this extension:

c.add	**Add**
c.addi	**Add a sign-extended non-zero immediate**
c.addi16sp	**Add a sign-extended non-zero immediate**
c.addi4spn	**Add a zero-extended non-zero immediate, scaled by 4, to the stack pointer**
c.addiw	**Add a sign-extended non-zero immediate**
c.addw	**Add word**
c.and	**And**
c.andi	**And immediate**
c.beqz	**Branch if Equal Zero**
c.bnez	**Branch if NOT Equal Zero**
c.ebreak	**Breakpoint exception.**
c.j	**Jump**
c.jal	**Jump and Link**

c.jalr	**Jump and Link Register.**
c.jr	**Jump Register**
c.ld	**Load double**
c.ldsp	**Load doubleword from stack pointer**
c.li	**Load the sign-extended 6-bit immediate**
c.lq	**Load quadruple**
c.lqsp	**Load quadruple word from stack pointer**
c.lui	**Load the non-zero 6-bit immediate field into bits 17-12 of the destination register**
c.lw	**Load word**
c.lwsp	**Load word from stack pointer**
c.mv	**Move Register**
c.nop	**Non-operation**
c.or	**Or**
c.sd	**Store double**
c.sdsp	**Store doubleword to stack**
c.slli	**Shift left logical immediate**
c.sq	**Store quadruple**
c.sqsp	**Store quadruple word to stack**
c.srai	**Shift right arithmetical immediate**
c.srli	**Shift right logical immediate**
c.sub	**Subtract**
c.subw	**Subtract word**
c.sw	**Store word**
c.swsp	**Store word to stack**
c.xor	**Exclusive Or**

A.2.5. Parameters

This extension has the following implementation options:

MUTABLE_MISA_C

Indicates whether or not the C extension can be disabled with the misa.C bit.

A.3. D Extension

Double-precision floating-point

Table 6. Status

Profile	v2.2.0
RVA22U64	mandatory
RVA22S64	-

2.2.0

Ratification date

2019-12

Changes

- Define NaN-boxing scheme, changed definition of FMAX and FMIN

A.3.1. Synopsis

The D extension adds double-precision floating-point computational instructions compliant with the IEEE 754-2008 arithmetic standard. The D extension depends on the base single-precision instruction subset F.

A.3.2. D Register State

The D extension widens the 32 floating-point registers, `f0-f31`, to 64 bits (FLEN=64 in Table 8. The `f` registers can now hold either 32-bit or 64-bit floating-point values as described below in Section A.3.3.

FLEN can be 32, 64, or 128 depending on which of the F, D, and Q extensions are supported. There can be up to four different floating-point precisions supported, including H, F, D, and Q.

A.3.3. NaN Boxing of Narrower Values

When multiple floating-point precisions are supported, then valid values of narrower *n*-bit types, *n*<FLEN, are represented in the lower *n* bits of an FLEN-bit NaN value, in a process termed NaN-boxing. The upper bits of a valid NaN-boxed value must be all 1s. Valid NaN-boxed *n*-bit values therefore appear as negative quiet NaNs (qNaNs) when viewed as any wider *m*-bit value, $n < m \leq$ FLEN. Any operation that writes a narrower result to an 'f' register must write all 1s to the uppermost FLEN-*n* bits to yield a legal NaN-boxedvalue.

Software might not know the current type of data stored in a floating-point register but has to be able to save and restore the register values, hence the result of using wider operations to transfer narrower values has to be defined. A

common case is for callee-saved registers, but a standard convention is also desirable for features including varargs, user-level threading libraries, virtual machine migration, and debugging.

Floating-point *n*-bit transfer operations move external values held in IEEE standard formats into and out of the f registers, and comprise floating-point loads and stores (FL*n*/FS*n*) and floating-point move instructions (FMV.*n*.X/FMV.X.*n*). A narrower *n*-bit transfer, *n*<FLEN, into the f registers will create a valid NaN-boxed value. A narrower *n*-bit transfer out of the floating-point registers will transfer the lower *n* bits of the register ignoring the upper FLEN-*n* bits.

Apart from transfer operations described in the previous paragraph, all other floating-point operations on narrower *n*-bit operations, *n*<FLEN, check if the input operands are correctly NaN-boxed, i.e., all upper FLEN-*n* bits are 1. If so, the *n* least-significant bits of the input are used as the input value, otherwise the input value is treated as an *n*-bit canonical NaN.

Earlier versions of this document did not define the behavior of feeding the results of narrower or wider operands into an operation, except to require that wider saves and restores would preserve the value of a narrower operand. The new definition removes this implementation-specific behavior, while still accommodating both non-recoded and recoded implementations of the floating-point unit. The new definition also helps catch software errors by propagating NaNs if values are used incorrectly.

Non-recoded implementations unpack and pack the operands to IEEE standard format on the input and output of every floating-point operation. The NaN-boxing cost to a non-recoded implementation is primarily in checking if the upper bits of a narrower operation represent a legal NaN-boxed value, and in writing all 1s to the upper bits of a result.

Recoded implementations use a more convenient internal format to represent floating-point values, with an added exponent bit to allow all values to be held normalized. The cost to the recoded implementation is primarily the extra tagging needed to track the internal types and sign bits, but this can be done without adding new state bits by recoding NaNs internally in the exponent field. Small modifications are needed to the pipelines used to transfer values in and out of the recoded format, but the datapath and latency costs are minimal. The recoding process has to handle shifting of input subnormal values for wide operands in any case, and extracting the NaN-boxed value is a similar process to normalization except for skipping over leading-1 bits instead of skipping over leading-0 bits, allowing the datapath muxing to be shared.

A.3.4. Instructions

The following instructions are added by this extension:

fadd.d	**No synopsis available.**
fclass.d	**No synopsis available.**

fcvt.d.h	**No synopsis available.**
fcvt.d.l	**No synopsis available.**
fcvt.d.lu	**No synopsis available.**
fcvt.d.s	**No synopsis available.**
fcvt.d.w	**No synopsis available.**
fcvt.d.wu	**No synopsis available.**
fcvt.h.d	**No synopsis available.**
fcvt.l.d	**No synopsis available.**
fcvt.lu.d	**No synopsis available.**
fcvt.s.d	**No synopsis available.**
fcvt.w.d	**No synopsis available.**
fcvt.wu.d	**No synopsis available.**
fcvtmod.w.d	**No synopsis available.**
fdiv.d	**No synopsis available.**
feq.d	**No synopsis available.**
fld	**No synopsis available.**
fle.d	**No synopsis available.**
fleq.d	**No synopsis available.**
fli.d	**No synopsis available.**
flt.d	**No synopsis available.**
fltq.d	**No synopsis available.**
fmadd.d	**No synopsis available.**
fmax.d	**No synopsis available.**
fmaxm.d	**No synopsis available.**
fmin.d	**No synopsis available.**
fminm.d	**No synopsis available.**
fmsub.d	**No synopsis available.**
fmul.d	**No synopsis available.**
fmv.d.x	**No synopsis available.**
fmv.x.d	**No synopsis available.**
fmvh.x.d	**No synopsis available.**
fmvp.d.x	**No synopsis available.**
fnmadd.d	**No synopsis available.**
fnmsub.d	**No synopsis available.**

fround.d	**No synopsis available.**
froundnx.d	**No synopsis available.**
fsd	**No synopsis available.**
fsgnj.d	**No synopsis available.**
fsgnjn.d	**No synopsis available.**
fsgnjx.d	**No synopsis available.**
fsqrt.d	**No synopsis available.**
fsub.d	**No synopsis available.**

A.3.5. Parameters

This extension has the following implementation options:

MUTABLE_MISA_D

Indicates whether or not the D extension can be disabled with the misa.D bit.

A.4. F Extension

Single-precision floating-point

Table 7. Status

Profile	v2.2.0
RVA22U64	mandatory
RVA22S64	-

2.2.0

Ratification date

2019-12

Changes

- Define NaN-boxing scheme, changed definition of FMAX and FMIN

A.4.1. Synopsis

This chapter describes the standard instruction-set extension for single-precision floating-point, which is named "F" and adds single-precision floating-point computational instructions compliant with the IEEE 754-2008 arithmetic standard cite:[ieee754-2008]. The F extension depends on the "Zicsr" extension for control and status register access.

A.4.2. F Register State

The F extension adds 32 floating-point registers, `f0-f31`, each 32 bits wide, and a floating-point control and status register fcsr, which contains the operating mode and exception status of the floating-point unit. This additional state is shown in Table 8. We use the term FLEN to describe the width of the floating-point registers in the RISC-V ISA, and FLEN=32 for the F single-precision floating-point extension. Most floating-point instructions operate on values in the floating-point register file. Floating-point load and store instructions transfer floating-point values between registers and memory. Instructions to transfer values to and from the integer register file are also provided.

We considered a unified register file for both integer and floating-point values as this simplifies software register allocation and calling conventions, and reduces total user state. However, a split organization increases the total number of registers accessible with a given instruction width, simplifies provision of enough regfile ports for wide superscalar issue, supports decoupled floating-point-unit architectures, and simplifies use of internal floating-point encoding techniques. Compiler support and calling conventions for split register file architectures are well understood, and using dirty bits on floating-point register file state can reduce context-switch overhead.

Table 8. RISC-V standard F extension single-precision floating-point state

FLEN-1	**0**
f0	
f1	
f2	
f3	
f4	
f5	
f6	
f7	
f8	
f9	
f10	
f11	
f12	
f13	
f14	
f15	
f16	
f17	
f18	
f19	
f20	
f21	
f22	
f23	
f24	
f25	
f26	
f27	
f28	
f29	
f30	
f31	
FLEN	
31	0
fcsr	
32	

Floating-Point Control and Status Register

The floating-point control and status register, fcsr, is a RISC-V control and status register (CSR). It is a 32-bit read/write register that selects the dynamic rounding mode for floating-point arithmetic operations and holds the accrued exception flags, as shown in Floating-Point Control and Status Register.

Floating-point control and status register

Unresolved directive in RVA22.adoc - include::images/wavedrom/float-csr.adoc[]

The fcsr register can be read and written with the FRCSR and FSCSR instructions, which are assembler pseudoinstructions built on the underlying CSR access instructions. FRCSR reads fcsr by copying it into integer register *rd*. FSCSR swaps the value in fcsr by copying the original value into integer register *rd*, and then writing a new value obtained from integer register *rs1* into fcsr.

The fields within the fcsr can also be accessed individually through different CSR addresses, and separate assembler pseudoinstructions are defined for these accesses. The FRRM instruction reads the Rounding Mode field `frm` (fcsr bits 7—5) and copies it into the least-significant three bits of integer register *rd*, with zero in all other bits. FSRM swaps the value in `frm` by copying the original value into integer register *rd*, and then writing a new value obtained from the three least-significant bits of integer register *rs1* into `frm`. FRFLAGS and FSFLAGS are defined analogously for the Accrued Exception Flags field `fflags` (fcsr bits 4—0).

Bits 31—8 of the fcsr are reserved for other standard extensions. If these extensions are not present, implementations shall ignore writes to these bits and supply a zero value when read. Standard software should preserve the contents of these bits.

Floating-point operations use either a static rounding mode encoded in the instruction, or a dynamic rounding mode held in `frm`. Rounding modes are encoded as shown in Table 9. A value of 111 in the instruction's *rm* field selects the dynamic rounding mode held in `frm`. The behavior of floating-point instructions that depend on rounding mode when executed with a reserved rounding mode is *reserved*, including both static reserved rounding modes (101-110) and dynamic reserved rounding modes (101-111). Some instructions, including widening conversions, have the *rm* field but are nevertheless mathematically unaffected by the rounding mode; software should set their *rm* field to RNE (000) but implementations must treat the *rm* field as usual (in particular, with regard to decoding legal vs. reserved encodings).

Table 9. Rounding mode encoding.

Rounding Mode	Mnemonic	Meaning
000	RNE	Round to Nearest, ties to Even
001	RTZ	Round towards Zero
010	RDN	Round Down (towards `-\infty`)
011	RUP	Round Up (towards `+\infty`)
100	RMM	Round to Nearest, ties to Max Magnitude
101		*Reserved for future use.*

Rounding Mode	Mnemonic	Meaning
110		*Reserved for future use.*
111	DYN	In instruction's *rm* field, selects dynamic rounding mode; In Rounding Mode register, *reserved.*

The C99 language standard effectively mandates the provision of a dynamic rounding mode register. In typical implementations, writes to the dynamic rounding mode CSR state will serialize the pipeline. Static rounding modes are used to implement specialized arithmetic operations that often have to switch frequently between different rounding modes.

The ratified version of the F spec mandated that an illegal-instruction exception was raised when an instruction was executed with a reserved dynamic rounding mode. This has been weakened to reserved, which matches the behavior of static rounding-mode instructions. Raising an illegal-instruction exception is still valid behavior when encountering a reserved encoding, so implementations compatible with the ratified spec are compatible with the weakened spec.

The accrued exception flags indicate the exception conditions that have arisen on any floating-point arithmetic instruction since the field was last reset by software, as shown in Table 10. The base RISC-V ISA does not support generating a trap on the setting of a floating-point exception flag.

Table 10. Accrued exception flag encoding.

Flag Mnemonic	Flag Meaning
NV	Invalid Operation
DZ	Divide by Zero
OF	Overflow
UF	Underflow
NX	Inexact

As allowed by the standard, we do not support traps on floating-point exceptions in the F extension, but instead require explicit checks of the flags in software. We considered adding branches controlled directly by the contents of the floating-point accrued exception flags, but ultimately chose to omit these instructions to keep the ISA simple.

A.4.3. NaN Generation and Propagation

Except when otherwise stated, if the result of a floating-point operation is NaN, it is the canonical NaN. The canonical NaN has a positive sign and all significand bits clear except the MSB, a.k.a. the quiet bit. For single-precision floating-point, this corresponds to the pattern `0x7fc00000`.

We considered propagating NaN payloads, as is recommended by the standard, but this decision would have increased hardware cost. Moreover, since this feature is optional in the standard, it cannot be used in portable code.

Implementors are free to provide a NaN payload propagation scheme as a nonstandard extension enabled by a nonstandard operating mode. However, the canonical NaN scheme described above must always be supported and should be the default mode.

We require implementations to return the standard-mandated default values in the case of exceptional conditions, without any further intervention on the part of user-level software (unlike the Alpha ISA floating-point trap barriers). We believe full hardware handling of exceptional cases will become more common, and so wish to avoid complicating the user-level ISA to optimize other approaches. Implementations can always trap to machine-mode software handlers to provide exceptional default values.

A.4.4. Subnormal Arithmetic

Operations on subnormal numbers are handled in accordance with the IEEE 754-2008 standard.

In the parlance of the IEEE standard, tininess is detected after rounding.

Detecting tininess after rounding results in fewer spurious underflow signals.

A.4.5. Instructions

The following instructions are added by this extension:

fadd.s	**No synopsis available.**
fclass.s	**Single-precision floating-point classify.**
fcvt.l.s	**No synopsis available.**
fcvt.lu.s	**No synopsis available.**
fcvt.s.l	**No synopsis available.**
fcvt.s.lu	**No synopsis available.**
fcvt.s.w	**Convert signed 32-bit integer to single-precision float**
fcvt.s.wu	**No synopsis available.**
fcvt.w.s	**Convert single-precision float to integer word to signed 32-bit integer.**
fcvt.wu.s	**No synopsis available.**
fdiv.s	**No synopsis available.**
feq.s	**Single-precision floating-point equal**

fle.s	**Single-precision floating-point less than or equal**
flt.s	**Single-precision floating-point less than**
flw	**Single-precision floating-point load**
fmadd.s	**No synopsis available.**
fmax.s	**No synopsis available.**
fmin.s	**No synopsis available.**
fmsub.s	**No synopsis available.**
fmul.s	**No synopsis available.**
fmv.h.x	**Half-precision floating-point move from integer**
fmv.w.x	**Single-precision floating-point move from integer**
fmv.x.w	**Move single-precision value from floating-point to integer register**
fnmadd.s	**No synopsis available.**
fnmsub.s	**No synopsis available.**
fsgnj.s	**Single-precision sign inject**
fsgnjn.s	**Single-precision sign inject negate**
fsgnjx.s	**Single-precision sign inject exclusive or**
fsqrt.s	**No synopsis available.**
fsub.s	**No synopsis available.**
fsw	**Single-precision floating-point store**

A.4.6. Parameters

This extension has the following implementation options:

HW_MSTATUS_FS_DIRTY_UPDATE

Indicates whether or not hardware will write to mstatus.FS

Values are:

never	Hardware never writes mstatus.FS
precise	Hardware writes mstatus.FS to the Dirty (3) state precisely when F registers are modified
imprecise	Hardware writes mstatus.FS imprecisely. This will result in a call to unpredictable() on any attempt to read mstatus or write FP state.

MSTATUS_FS_LEGAL_VALUES

The set of values that mstatus.FS will accept from a software write.

MUTABLE_MISA_F

Indicates whether or not the F extension can be disabled with the misa.F bit.

A.5. H Extension

Hypervisor

Table 11. Status

Profile	v1.0.0
RVA22U64	-
RVA22S64	optional

1.0.0

Ratification date

2019-12

A.5.1. Synopsis

This chapter describes the RISC-V hypervisor extension, which virtualizes the supervisor-level architecture to support the efficient hosting of guest operating systems atop a type-1 or type-2 hypervisor. The hypervisor extension changes supervisor mode into *hypervisor-extended supervisor mode* (HS-mode, or *hypervisor mode* for short), where a hypervisor or a hosting-capable operating system runs. The hypervisor extension also adds another stage of address translation, from *guest physical addresses* to supervisor physical addresses, to virtualize the memory and memory-mapped I/O subsystems for a guest operating system. HS-mode acts the same as S-mode, but with additional instructions and CSRs that control the new stage of address translation and support hosting a guest OS in virtual S-mode (VS-mode). Regular S-mode operating systems can execute without modification either in HS-mode or as VS-mode guests.

In HS-mode, an OS or hypervisor interacts with the machine through the same SBI as an OS normally does from S-mode. An HS-mode hypervisor is expected to implement the SBI for its VS-mode guest.

The hypervisor extension depends on an "I" base integer ISA with 32 x registers (RV32I or RV64I), not RV32E or RV64E, which have only 16 x registers. CSR mtval must not be read-only zero, and standard page-based address translation must be supported, either Sv32 for RV32, or a minimum of Sv39 for RV64.

The hypervisor extension is enabled by setting bit 7 in the misa CSR, which corresponds to the letter H. RISC-V harts that implement the hypervisor extension are encouraged not to hardwire misa[7], so that the extension may be disabled.

The baseline privileged architecture is designed to simplify the use of classic virtualization techniques, where a guest OS is run at user-level, as the few privileged instructions can be easily detected and trapped. The hypervisor extension improves virtualization performance by reducing the frequency of these traps.

The hypervisor extension has been designed to be efficiently emulable on platforms that do not implement the extension, by running the hypervisor in S-

> mode and trapping into M-mode for hypervisor CSR accesses and to maintain shadow page tables. The majority of CSR accesses for type-2 hypervisors are valid S-mode accesses so need not be trapped. Hypervisors can support nested virtualization analogously.

A.5.2. Privilege Modes

The current *virtualization mode*, denoted V, indicates whether the hart is currently executing in a guest. When V=1, the hart is either in virtual S-mode (VS-mode), or in virtual U-mode (VU-mode) atop a guest OS running in VS-mode. When V=0, the hart is either in M-mode, in HS-mode, or in U-mode atop an OS running in HS-mode. The virtualization mode also indicates whether two-stage address translation is active (V=1) or inactive (V=0). Table 12 lists the possible privilege modes of a RISC-V hart with the hypervisor extension.

Table 12. Privilege modes with the hypervisor extension.

Virtualization Mode (V)	Nominal Privilege	Abbreviation	Name	Two-Stage Translation
0 0 0	U S M	U-mode HS-mode M-mode	User mode Hypervisor-extended supervisor mode Machine mode	Off Off Off
1 1	U S	VU-mode VS-mode	Virtual user mode Virtual supervisor mode	On On

For privilege modes U and VU, the *nominal privilege mode* is U, and for privilege modes HS and VS, the nominal privilege mode is S.

HS-mode is more privileged than VS-mode, and VS-mode is more privileged than VU-mode. VS-mode interrupts are globally disabled when executing in U-mode.

This description does not consider the possibility of U-mode or VU-mode interrupts and will be revised if an extension for user-level interrupts is adopted.

A.5.3. Instructions

The following instructions are added by this extension:

hfence.gvma	**No synopsis available.**
hfence.vvma	**No synopsis available.**
hlv.b	**No synopsis available.**
hlv.bu	**No synopsis available.**
hlv.d	**No synopsis available.**
hlv.h	**No synopsis available.**
hlv.hu	**No synopsis available.**
hlv.w	**No synopsis available.**
hlv.wu	**No synopsis available.**
hlvx.hu	**No synopsis available.**
hlvx.wu	**No synopsis available.**
hsv.b	**No synopsis available.**
hsv.d	**No synopsis available.**
hsv.h	**No synopsis available.**
hsv.w	**No synopsis available.**

A.5.4. Parameters

This extension has the following implementation options:

GSTAGE_MODE_BARE

Whether or not writing mode=Bare is supported in the hgatp register.

HCOUNTENABLE_EN

Indicates which counters can delegated via hcounteren

An unimplemented counter cannot be specified, i.e., if HPM_COUNTER_EN[3] is false, it would be illegal to set HCOUNTENABLE_EN[3] to true.

HCOUNTENABLE_EN[0:2] must all be false if Zicntr is not implemented. HCOUNTENABLE_EN[3:31] must all be false if Zihpm is not implemented.

IGNORE_INVALID_VSATP_MODE_WRITES_WHEN_V_EQ_ZERO

Whether writes from M-mode, U-mode, or S-mode to vsatp with an illegal mode setting are ignored (as they are with satp), or if they are treated as WARL, leading to undpredictable behavior.

MUTABLE_MISA_H

Indicates whether or not the H extension can be disabled with the misa.H bit.

NUM_EXTERNAL_GUEST_INTERRUPTS

Number of supported virtualized guest interrupts

Corresponds to the `GEILEN` parameter in the RVI specs

REPORT_ENCODING_IN_VSTVAL_ON_ILLEGAL_INSTRUCTION

When true, vstval is written with the encoding of an instruction that causes an `IllegalInstruction` exception.

When false vstval is written with 0 when an `IllegalInstruction` exception occurs.

REPORT_GPA_IN_TVAL_ON_INSTRUCTION_GUEST_PAGE_FAULT

Whether or not GPA >> 2 is written into htval/mtval2 when an instruction guest page fault occurs.

If false, 0 will be written into htval/mtval2 on an instruction guest page fault.

REPORT_GPA_IN_TVAL_ON_INTERMEDIATE_GUEST_PAGE_FAULT

Whether or not GPA >> 2 is written into htval/mtval2 when a guest page fault occurs while walking a VS-mode page table.

If false, 0 will be written into htval/mtval2 on an intermediate guest page fault.

REPORT_GPA_IN_TVAL_ON_LOAD_GUEST_PAGE_FAULT

Whether or not GPA >> 2 is written into htval/mtval2 when a load guest page fault occurs.

If false, 0 will be written into htval/mtval2 on a load guest page fault.

REPORT_GPA_IN_TVAL_ON_STORE_AMO_GUEST_PAGE_FAULT

Whether or not GPA >> 2 is written into htval/mtval2 when a store/amo guest page fault occurs.

If false, 0 will be written into htval/mtval2 on a store/amo guest page fault.

REPORT_VA_IN_VSTVAL_ON_BREAKPOINT

When true, vstval is written with the virtual PC of the EBREAK instruction (same information as mepc).

When false, vstval is written with 0 on an EBREAK instruction.

Regardless, vstval is always written with a virtual PC when an external breakpoint is generated

REPORT_VA_IN_VSTVAL_ON_INSTRUCTION_ACCESS_FAULT

When true, vstval is written with the virtual PC of an instructino when fetch causes an `InstructionAccessFault`.

WHen false, vstval is written with 0 when an instruction fetch causes an `InstructionAccessFault`.

REPORT_VA_IN_VSTVAL_ON_INSTRUCTION_MISALIGNED

When true, vstval is written with the virtual PC when an instruction fetch is misaligned.

When false, vstval is written with 0 when an instruction fetch is misaligned.

Note that when IALIGN=16 (i.e., when the C or one of the `Zc*` extensions are implemented), it is impossible to generate a misaligned fetch, and so this parameter has no effect.

REPORT_VA_IN_VSTVAL_ON_INSTRUCTION_PAGE_FAULT

When true, vstval is written with the virtual PC of an instructino when fetch causes an `InstructionPageFault`.

WHen false, vstval is written with 0 when an instruction fetch causes an `InstructionPageFault`.

REPORT_VA_IN_VSTVAL_ON_LOAD_ACCESS_FAULT

When true, vstval is written with the virtual address of a load when it causes a `LoadAccessFault`.

WHen false, vstval is written with 0 when a load causes a `LoadAccessFault`.

REPORT_VA_IN_VSTVAL_ON_LOAD_MISALIGNED

When true, vstval is written with the virtual address of a load instruction when the address is misaligned and MISALIGNED_LDST is false.

When false, vstval is written with 0 when a load address is misaligned and MISALIGNED_LDST is false.

REPORT_VA_IN_VSTVAL_ON_LOAD_PAGE_FAULT

When true, vstval is written with the virtual address of a load when it causes a `LoadPageFault`.

WHen false, vstval is written with 0 when a load causes a LoadPageFault.

REPORT_VA_IN_VSTVAL_ON_STORE_AMO_ACCESS_FAULT

When true, vstval is written with the virtual address of a store when it causes a StoreAmoAccessFault.

WHen false, vstval is written with 0 when a store causes a StoreAmoAccessFault.

REPORT_VA_IN_VSTVAL_ON_STORE_AMO_MISALIGNED

When true, vstval is written with the virtual address of a store instruction when the address is misaligned and MISALIGNED_LDST is false.

When false, vstval is written with 0 when a store address is misaligned and MISALIGNED_LDST is false.

REPORT_VA_IN_VSTVAL_ON_STORE_AMO_PAGE_FAULT

When true, vstval is written with the virtual address of a store when it causes a StoreAmoPageFault.

WHen false, vstval is written with 0 when a store causes a StoreAmoPageFault.

SV32X4_TRANSLATION

Whether or not Sv32x4 translation mode is supported.

SV32_VSMODE_TRANSLATION

Whether or not Sv32 translation is supported in first-stage (VS-stage) translation.

SV39X4_TRANSLATION

Whether or not Sv39x4 translation mode is supported.

SV39_VSMODE_TRANSLATION

Whether or not Sv39 translation is supported in first-stage (VS-stage) translation.

SV48X4_TRANSLATION

Whether or not Sv48x4 translation mode is supported.

SV48_VSMODE_TRANSLATION

Whether or not Sv48 translation is supported in first-stage (VS-stage) translation.

SV57X4_TRANSLATION

Whether or not Sv57x4 translation mode is supported.

SV57_VSMODE_TRANSLATION

Whether or not Sv57 translation is supported in first-stage (VS-stage) translation.

TINST_VALUE_ON_BREAKPOINT

Value written into htinst/mtinst on a Breakpoint exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "custom": Write a custom value,

which resuls in UNPREDICTABLE

TINST_VALUE_ON_FINAL_INSTRUCTION_GUEST_PAGE_FAULT

Value to write into htval/mtval2 when there is a guest page fault on a final translation.

Possible values: * "always zero": Always write the value zero * "always pseudoinstruction": Always write the pseudoinstruction

TINST_VALUE_ON_FINAL_LOAD_GUEST_PAGE_FAULT

Value to write into htval/mtval2 when there is a guest page fault on a final translation.

Possible values: * "always zero": Always write the value zero * "always pseudoinstruction": Always write the pseudoinstruction * "always transformed standard instruction": Always write the transformation of the standard instruction encoding * "custom": A custom value, which will cause an UNPREDICTABLE event.

TINST_VALUE_ON_FINAL_STORE_AMO_GUEST_PAGE_FAULT

Value to write into htval/mtval2 when there is a guest page fault on a final translation.

Possible values: * "always zero": Always write the value zero * "always pseudoinstruction": Always write the pseudoinstruction * "always transformed standard instruction": Always write the transformation of the standard instruction encoding * "custom": A custom value, which will cause an UNPREDICTABLE event.

TINST_VALUE_ON_INSTRUCTION_ADDRESS_MISALIGNED

Value written into htinst/mtinst when there is an instruction address misaligned exception.

Possible values: * "always zero": Always write the value zero * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_LOAD_ACCESS_FAULT

Value written into htinst/mtinst on an AccessFault exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "always transformed standard instruction": Always write a transformed standard instruction as defined by H * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_LOAD_ADDRESS_MISALIGNED

Value written into htinst/mtinst on a VirtualInstruction exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "always transformed standard instruction": Always write a transformed standard instruction as defined by H * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_LOAD_PAGE_FAULT

Value written into htinst/mtinst on a LoadPageFault exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "always transformed standard instruction": Always write a transformed standard instruction as defined by H * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_MCALL

Value written into htinst/mtinst on a MCall exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_SCALL

Value written into htinst/mtinst on a SCall exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_STORE_AMO_ACCESS_FAULT

Value written into htinst/mtinst on an AccessFault exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "always transformed standard instruction": Always write a transformed standard instruction as defined by H * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_STORE_AMO_ADDRESS_MISALIGNED

Value written into htinst/mtinst on a VirtualInstruction exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "always transformed standard instruction": Always write a transformed standard instruction as defined by H * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_STORE_AMO_PAGE_FAULT

Value written into htinst/mtinst on a StoreAmoPageFault exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "always transformed standard instruction": Always write a transformed standard instruction as defined by H * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_UCALL

Value written into htinst/mtinst on a UCall exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_VIRTUAL_INSTRUCTION

Value written into htinst/mtinst on a VirtualInstruction exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "custom": Write a custom value, which resuls in UNPREDICTABLE

TINST_VALUE_ON_VSCALL

Value written into htinst/mtinst on a VSCall exception from VU/VS-mode.

Possible values: * "always zero": Always write the value zero * "custom": Write a custom value, which resuls in UNPREDICTABLE

TRAP_ON_ECALL_FROM_VS

Whether or not an ECALL-from-VS-mode causes a synchronous exception.

The spec states that implementations may handle ECALLs transparently without raising a trap, in which case the EEI must provide a builtin.

VMID_WIDTH

Number of bits supported in hgatp.VMID (i.e., the supported width of a virtual machine ID).

VSXLEN

Set of XLENs supported in VS-mode. Can be one of:

- 32: VSXLEN is always 32
- 64: VSXLEN is always 64
- 3264: VSXLEN can be changed (via hstatus.VSXL) between 32 and 64

VS_MODE_ENDIANESS

Endianess of data in VS-mode. Can be one of:

- little: M-mode data is always little endian
- big: M-mode data is always big endian
- dynamic: M-mode data can be either little or big endian, depending on the CSR field hstatus.VSBE

VUXLEN

Set of XLENs supported in VU-mode. Can be one of:

- 32: VUXLEN is always 32
- 64: VUXLEN is always 64
- 3264: VUXLEN can be changed (via vsstatus.UXL) between 32 and 64

VU_MODE_ENDIANESS

Endianess of data in VU-mode. Can be one of:

- little: M-mode data is always little endian
- big: M-mode data is always big endian
- dynamic: M-mode data can be either little or big endian, depending on the CSR field vsstatus.UBE

A.6. I Extension

Base integer ISA (RV32I or RV64I)

Table 13. Status

Profile	v2.1.0
RVA22U64	mandatory
RVA22S64	-

2.1.0

Ratification date

2019-06

Changes

- ratified RVWMO memory model and exclusion of FENCE.I, counters, and CSR instructions that were in previous base ISA

A.6.1. Synopsis

Base integer instructions — TODO

A.6.2. Instructions

The following instructions are added by this extension:

add	**Integer add**
addi	**Add immediate**
addiw	**Add immediate word**
addw	**Add word**
and	**And**
andi	**And immediate**
auipc	**Add upper immediate to pc**
beq	**Branch if equal**
bge	**Branch if greater than or equal**
bgeu	**Branch if greater than or equal unsigned**
blt	**Branch if less than**
bltu	**Branch if less than unsigned**
bne	**Branch if not equal**
ebreak	**Breakpoint exception**
ecall	**Environment call**

fence	**Memory ordering fence**
jal	**Jump and link**
jalr	**Jump and link register**
lb	**Load byte**
lbu	**Load byte unsigned**
ld	**Load doubleword**
lh	**Load halfword**
lhu	**Load halfword unsigned**
lui	**Load upper immediate**
lw	**Load word**
lwu	**Load word unsigned**
or	**Or**
ori	**Or immediate**
sb	**Store byte**
sd	**Store doubleword**
sh	**Store halfword**
sll	**Shift left logical**
slli	**Shift left logical immediate**
slliw	**Shift left logical immediate word**
sllw	**Shift left logical word**
slt	**Set on less than**
slti	**Set on less than immediate**
sltiu	**Set on less than immediate unsigned**
sltu	**Set on less than unsigned**
sra	**Shift right arithmetic**
srai	**Shift right arithmetic immediate**
sraiw	**Shift right arithmetic immediate word**
sraw	**Shift right arithmetic word**
srl	**Shift right logical**
srli	**Shift right logical immediate**
srliw	**Shift right logical immediate word**
srlw	**Shift right logical word**
sub	**Subtract**
subw	**Subtract word**

sw	**Store word**
xor	**Exclusive Or**
xori	**Exclusive Or immediate**

A.7. M Extension

Integer multiply and divide instructions

Table 14. Status

Profile	v2.0.0
RVA22U64	mandatory
RVA22S64	-

2.0.0

Ratification date

2019-12

A.7.1. Synopsis

This chapter describes the standard integer multiplication and division instruction extension, which is named M and contains instructions that multiply or divide values held in two integer registers.

We separate integer multiply and divide out from the base to simplify low-end implementations, or for applications where integer multiply and divide operations are either infrequent or better handled in attached accelerators.

A.7.2. Instructions

The following instructions are added by this extension:

div	**Signed division**
divu	**Unsigned division**
divuw	**Unsigned 32-bit division**
divw	**Signed 32-bit division**
mul	**Signed multiply**
mulh	**Signed multiply high**
mulhsu	**Signed/unsigned multiply high**
mulhu	**Unsigned multiply high**
mulw	**Signed 32-bit multiply**
rem	**Signed remainder**
remu	**Unsigned remainder**
remuw	**Unsigned 32-bit remainder**
remw	**Signed 32-bit remainder**

A.7.3. Parameters

This extension has the following implementation options:

MUTABLE_MISA_M

Indicates whether or not the M extension can be disabled with the misa.M bit.

A.8. S Extension

Supervisor mode

Table 15. Status

Profile	v1.12.0
RVA22U64	-
RVA22S64	mandatory

1.12.0

Ratification date

2021-12

A.8.1. Synopsis

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.

A.8.2. Instructions

The following instructions are added by this extension:

sfence.vma	**Supervisor memory-management fence**
sret	**Supervisor Exception Return**

A.8.3. Parameters

This extension has the following implementation options:

ASID_WIDTH

Number of implemented ASID bits. Maximum is 16 for XLEN==64, and 9 for XLEN==32

MSTATUS_FS_LEGAL_VALUES

The set of values that mstatus.FS will accept from a software write.

MSTATUS_FS_WRITEABLE

When S is enabled but F is not, mstatus.FS is optionally writeable.

This parameter only has an effect when both S and F mode are disabled.

MSTATUS_TVM_IMPLEMENTED

Whether or not mstatus.TVM is implemented.

When not implemented mstatus.TVM will be read-only-zero.

MSTATUS_VS_LEGAL_VALUES

The set of values that mstatus.VS will accept from a software write.

MSTATUS_VS_WRITEABLE

When S is enabled but V is not, mstatus.VS is optionally writeable.

This parameter only has an effect when both S and V mode are disabled.

MUTABLE_MISA_S

Indicates whether or not the S extension can be disabled with the misa.S bit.

REPORT_ENCODING_IN_STVAL_ON_ILLEGAL_INSTRUCTION

When true, stval is written with the encoding of an instruction that causes an `IllegalInstruction` exception.

When false stval is written with 0 when an `IllegalInstruction` exception occurs.

REPORT_VA_IN_STVAL_ON_BREAKPOINT

When true, stval is written with the virtual PC of the EBREAK instruction (same information as mepc).

When false, stval is written with 0 on an EBREAK instruction.

Regardless, stval is always written with a virtual PC when an external breakpoint is generated

REPORT_VA_IN_STVAL_ON_INSTRUCTION_ACCESS_FAULT

When true, stval is written with the virtual PC of an instructino when fetch causes an `InstructionAccessFault`.

WHen false, stval is written with 0 when an instruction fetch causes an `InstructionAccessFault`.

REPORT_VA_IN_STVAL_ON_INSTRUCTION_MISALIGNED

When true, stval is written with the virtual PC when an instruction fetch is misaligned.

When false, stval is written with 0 when an instruction fetch is misaligned.

Note that when IALIGN=16 (i.e., when the C or one of the `Zc*` extensions are implemented), it is impossible to generate a misaligned fetch, and so this parameter has no effect.

REPORT_VA_IN_STVAL_ON_INSTRUCTION_PAGE_FAULT

When true, stval is written with the virtual PC of an instructino when fetch causes an `InstructionPageFault`.

WHen false, stval is written with 0 when an instruction fetch causes an `InstructionPageFault`.

REPORT_VA_IN_STVAL_ON_LOAD_ACCESS_FAULT

When true, stval is written with the virtual address of a load when it causes a `LoadAccessFault`.

WHen false, stval is written with 0 when a load causes a `LoadAccessFault`.

REPORT_VA_IN_STVAL_ON_LOAD_MISALIGNED

When true, stval is written with the virtual address of a load instruction when the address is misaligned and MISALIGNED_LDST is false.

When false, stval is written with 0 when a load address is misaligned and MISALIGNED_LDST is false.

REPORT_VA_IN_STVAL_ON_LOAD_PAGE_FAULT

When true, stval is written with the virtual address of a load when it causes a `LoadPageFault`.

WHen false, stval is written with 0 when a load causes a `LoadPageFault`.

REPORT_VA_IN_STVAL_ON_STORE_AMO_ACCESS_FAULT

When true, stval is written with the virtual address of a store when it causes a `StoreAmoAccessFault`.

WHen false, stval is written with 0 when a store causes a `StoreAmoAccessFault`.

REPORT_VA_IN_STVAL_ON_STORE_AMO_MISALIGNED

When true, stval is written with the virtual address of a store instruction when the address is misaligned and MISALIGNED_LDST is false.

When false, stval is written with 0 when a store address is misaligned and MISALIGNED_LDST is false.

REPORT_VA_IN_STVAL_ON_STORE_AMO_PAGE_FAULT

When true, stval is written with the virtual address of a store when it causes a `StoreAmoPageFault`.

WHen false, stval is written with 0 when a store causes a `StoreAmoPageFault`.

SATP_MODE_BARE

Whether or not satp.MODE == Bare is supported.

SCOUNTENABLE_EN

Indicates which counters can delegated via scounteren

An unimplemented counter cannot be specified, i.e., if HPM_COUNTER_EN[3] is false, it would be illegal to set SCOUNTENABLE_EN[3] to true.

SCOUNTENABLE_EN[0:2] must all be false if Zicntr is not implemented. SCOUNTENABLE_EN[3:31] must all be false if Zihpm is not implemented.

STVAL_WIDTH

The number of implemented bits in stval.

Must be greater than or equal to *max*(PHYS_ADDR_WIDTH, VA_SIZE)

STVEC_MODE_DIRECT

Whether or not stvec.MODE supports Direct (0).

STVEC_MODE_VECTORED

Whether or not stvec.MODE supports Vectored (1).

SV_MODE_BARE

Whether or not writing mode=Bare is supported in the satp register.

SXLEN

Set of XLENs supported in S-mode. Can be one of:

- 32: SXLEN is always 32
- 64: SXLEN is always 64
- 3264: SXLEN can be changed (via mstatus.SXL) between 32 and 64

S_MODE_ENDIANESS

Endianess of data in S-mode. Can be one of:

- little: M-mode data is always little endian
- big: M-mode data is always big endian
- dynamic: M-mode data can be either little or big endian, depending on the CSR field mstatus.SBE

TRAP_ON_ECALL_FROM_S

Whether or not an ECALL-from-S-mode causes a synchronous exception.

The spec states that implementations may handle ECALLs transparently without raising a trap, in which case the EEI must provide a builtin.

TRAP_ON_SFENCE_VMA_WHEN_SATP_MODE_IS_READ_ONLY

For implementations that make satp.MODE read-only zero (always Bare, *i.e.*, no virtual translation is implemented), attempts to execute an SFENCE.VMA instruction might raise an illegal-instruction exception.

TRAP_ON_SFENCE_VMA_WHEN_SATP_MODE_IS_READ_ONLY indicates whether or not that exception occurs.

TRAP_ON_SFENCE_VMA_WHEN_SATP_MODE_IS_READ_ONLY has no effect when some virtual translation mode is supported.

A.9. Ssccptr Extension

Cacheable and coherent main memory page table reads

Table 16. Status

Profile	v1.0.0
RVA22U64	-
RVA22S64	mandatory

1.0.0

Ratification date

Ratification document

github.com/riscv/riscv-profiles/releases/tag/v1.0

A.9.1. Synopsis

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

This extension was ratified with the RVA20 profiles.

A.10. Sscofpmf Extension

Counter Overflow and Privilege Mode Filtering

Table 17. Status

Profile	v1.0.0
RVA22U64	-
RVA22S64	optional

1.0.0

Ratification date

2023-08

Ratification document

drive.google.com/file/d/1KcjgbLM5L1ZKY8934aJl8aQwGlMz6Cbo/view?usp=drive_link

A.10.1. Synopsis

Counter Overflow and Privilege Mode Filtering

A.11. Sscounterenw Extension

Supervisor counter enable

Table 18. Status

Profile	v1.0.0
RVA22U64	-
RVA22S64	mandatory

1.0.0

Ratification date

2023-08

Ratification document

drive.google.com/file/d/1KcjgbLM5L1ZKY8934aJl8aQwGlMz6Cbo/view?usp=drive_link

A.11.1. Synopsis

For any hpmcounter that is not read-only zero, the corresponding bit in scounteren must be writable.

This extension was ratified with the RVA22 profiles.

A.12. Sstc Extension

Superivisor mode timer interrupts

Table 19. Status

Profile	v0.9.0
RVA22U64	-
RVA22S64	-

0.9.0

Ratification date

Ratification document

drive.google.com/file/d/1m84Re2yK8m_vbW7TspvevCDR82MOBaSX/view?usp=drive_link

A.12.1. Synopsis

Superivisor mode timer interrupts

A.13. Sstvala Extension

stval requirements for RVA profiles

Table 20. Status

Profile	v1.0.0
RVA22U64	-
RVA22S64	mandatory

1.0.0

Ratification date

Ratification document

github.com/riscv/riscv-profiles/releases/tag/v1.0

A.13.1. Synopsis

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 virtual-instruction and illegal-instruction exceptions, stval must be written with the faulting instruction.

This extension was ratified with the RVA20 profiles.

A.14. Sstvecd Extension

Direct exception vectoring

Table 21. Status

Profile	v1.0.0
RVA22U64	-
RVA22S64	mandatory

1.0.0

Ratification date

Ratification document

github.com/riscv/riscv-profiles/releases/tag/v1.0

A.14.1. Synopsis

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.

A.15. Sv39 Extension

39-bit virtual address translation (3 level)

Table 22. Status

Profile	v1.12.0
RVA22U64	-
RVA22S64	mandatory

1.12.0

Ratification date

unknown

Ratification document

github.com/riscv/riscv-isa-manual/releases/download/Priv-v1.12/riscv-privileged-20211203.pdf

A.15.1. Synopsis

39-bit virtual address translation (3 level)

A.16. Sv48 Extension

48-bit virtual address translation (4 level)

Table 23. Status

Profile	v1.12.0
RVA22U64	-
RVA22S64	optional

1.12.0

Ratification date

unknown

Ratification document

github.com/riscv/riscv-isa-manual/releases/download/Priv-v1.12/riscv-privileged-20211203.pdf

A.16.1. Synopsis

48-bit virtual address translation (4 level)

A.17. Sv57 Extension

57-bit virtual address translation (5 level)

Table 24. Status

Profile	v1.12.0
RVA22U64	-
RVA22S64	optional

1.12.0

Ratification date

unknown

Ratification document

github.com/riscv/riscv-isa-manual/releases/download/Priv-v1.12/riscv-privileged-20211203.pdf

A.17.1. Synopsis

57-bit virtual address translation (5 level)

A.18. Svade Extension

Exception on PTE A/D Bits

Table 25. Status

Profile	v1.0.0
RVA22U64	-
RVA22S64	mandatory

1.0.0

Ratification date

2023-11

Ratification document

github.com/riscvarchive/riscv-svadu/releases/download/v1.0/riscv-svadu.pdf

A.18.1. Synopsis

The Svade extension indicates that hardware does **not** update the A/D bits of a page table during a page walk. Rather, encountering a PTE with the A bit clear or the D bit clear when an operation is a write will cause a Page Fault.

A.19. Svbare Extension

Bare virtual addressing

Table 26. Status

Profile	**v1.0.0**
RVA22U64	-
RVA22S64	mandatory

1.0.0

Ratification date

==== Synopsis

This extension mandates that the satp mode Bare must be supported.

This extension was ratified as part of the RVA22 profile.

A.20. Svinval Extension

Fine-grained address-translation cache invalidation

Table 27. Status

Profile	v1.0.0
RVA22U64	-
RVA22S64	mandatory

1.0.0

Ratification date

2021-11

A.20.1. Synopsis

The Svinval extension splits sfence.vma, `hfence.vma`, and hfence.gvma instructions into finer-grained invalidation and ordering operations that can be more efficiently batched or pipelined on certain classes of high-performance implementation.

The sinval.vma instruction invalidates any address-translation cache entries that an sfence.vma instruction with the same values of rs1 and rs2 would invalidate. However, unlike sfence.vma, sinval.vma instructions are only ordered with respect to sfence.vma, sfence.w.inval, and sfence.inval.ir instructions as defined below.

The sfence.w.inval instruction guarantees that any previous stores already visible to the current RISC-V hart are ordered before subsequent sinval.vma instructions executed by the same hart. The sfence.inval.ir instruction guarantees that any previous sinval.vma instructions executed by the current hart are ordered before subsequent implicit references by that hart to the memory-management data structures.

When executed in order (but not necessarily consecutively) by a single hart, the sequence sfence.w.inval, sinval.vma, and sfence.inval.ir has the same effect as a hypothetical sfence.vma instruction in which:

- the values of *rs1* and *rs2* for the sfence.vma are the same as those used in the sinval.vma,
- reads and writes prior to the sfence.w.inval are considered to be those prior to the sfence.vma, and
- reads and writes following the sfence.inval.ir are considered to be those subsequent to the sfence.vma.

If the hypervisor extension is implemented, the Svinval extension also provides two additional instructions: hinval.vvma and hinval.gvma. These have the same semantics as sinval.vma, except that they combine with sfence.w.inval and sfence.inval.ir to replace hfence.vvma and hfence.gvma, respectively, instead of sfence.vma. In addition, hinval.gvma uses VMIDs instead of ASIDs.

sinval.vma, hinval.vvma, and hinval.gvma require the same permissions and raise the same exceptions as sfence.vma, hfence.vvma, and hfence.gvma, respectively. In particular, an attempt to

execute any of these instructions in U-mode always raises an IllegalInstruction exception, and an attempt to execute sinval.vma or hinval.gvma in S-mode or HS-mode when mstatus.TVM=1 also raises an IllegalInstruction exception. An attempt to execute hinval.vvma or hinval.gvma in VS-mode or VU-mode, or to execute sinval.vma in VU-mode, raises a `VirtualInstruction` exception. When hstatus.VTVM=1, an attempt to execute sinval.vma in VS-mode also raises a `VirtualInstruction` exception.

Attempting to execute sfence.w.inval or sfence.inval.ir in U-mode raises an `IllegalInstruction` exception. Doing so in VU-mode raises a `VirtualInstruction` exception. sfence.w.inval and sfence.inval.ir are unaffected by the mstatus.TVM and hstatus.VTVM fields and hence are always permitted in S-mode and VS-mode.

sfence.w.inval and sfence.inval.ir instructions do not need to be trapped when mstatus.TVM=1 or when hstatus.VTVM=1, as they only have ordering effects but no visible side effects. Trapping of the sinval.vma instruction is sufficient to enable emulation of the intended overall TLB maintenance functionality.

In typical usage, software will invalidate a range of virtual addresses in the address-translation caches by executing an sfence.w.inval instruction, executing a series of sinval.vma, hinval.vvma, or hinval.gvma instructions to the addresses (and optionally ASIDs or VMIDs) in question, and then executing an sfence.inval.ir instruction.

High-performance implementations will be able to pipeline the address-translation cache invalidation operations, and will defer any pipeline stalls or other memory ordering enforcement until an sfence.w.inval, sfence.inval.ir, sfence.vma, hfence.gvma, or hfence.vvma instruction is executed.

Simpler implementations may implement sinval.vma, hinval.vvma, and hinval.gvma identically to sfence.vma, hfence.vvma, and hfence.gvma, respectively, while implementing sfence.w.inval and sfence.inval.ir instructions as no-ops.

A.20.2. Instructions

The following instructions are added by this extension:

sfence.inval.ir	**Order implicit page table reads after invalidation**
sfence.w.inval	**Order writes before sfence**
sinval.vma	**Invalidate cached address translations**

A.21. Svnapot Extension

Naturally-aligned Power of Two Translation Contiguity

Table 28. Status

Profile	v1.0.0
RVA22U64	-
RVA22S64	optional

1.0.0

Ratification date

2021-11

A.21.1. Synopsis

In Sv39, Sv48, and Sv57, when a PTE has N=1, the PTE represents a translation that is part of a range of contiguous virtual-to-physical translations with the same values for PTE bits 5-0. Such ranges must be of a naturally aligned power-of-2 (NAPOT) granularity larger than the base page size.

The Svnapot extension depends on Sv39.

Table 29. Page table entry encodings when pte.N=1

i	*pte.ppn*[*i*]	Description	*pte.napot_bits*
0	`x xxxx xxx1`	*Reserved*	-
0	`x xxxx xx1x`	*Reserved*	-
0	`x xxxx x1xx`	*Reserved*	-
0	`x xxxx 1000`	64 KiB contiguous region	4
0	`x xxxx 0xxx`	*Reserved*	-
≥1	`x xxxx xxxx`	*Reserved*	-

NAPOT PTEs behave identically to non-NAPOT PTEs within the address-translation algorithm in [sv32algorithm], except that:

- If the encoding in *pte* is valid according to Table 29, then instead of returning the original value of *pte*, implicit reads of a NAPOT PTE return a copy of *pte* in which *pte.ppn*[*i*][*pte.napot_bits*-1:0] is replaced by *vpn*[*i*][*pte.napot_bits*-1:0]. If the encoding in *pte* is reserved according to Table 29, then a page-fault exception must be raised.
- Implicit reads of NAPOT page table entries may create address-translation cache entries mapping *a* + *j**PTESIZE to a copy of *pte* in which *pte.ppn*[*i*][*pte.napot_bits*-1:0] is replaced by *vpn[i][pte.napot_bits*-1:0], for any or all *j* such that *j* >> *napot_bits* = *vpn*[*i*] >> *napot_bits*, all for the address space identified in *satp* as loaded by step 1.

The motivation for a NAPOT PTE is that it can be cached in a TLB as one or more entries representing the contiguous region as if it were a single (large) page covered by a single translation. This compaction can help relieve TLB pressure in some scenarios. The encoding is designed to fit within the pre-existing Sv39, Sv48,

and Sv57 PTE formats so as not to disrupt existing implementations or designs that choose not to implement the scheme. It is also designed so as not to complicate the definition of the address-translation algorithm.

The address translation cache abstraction captures the behavior that would result from the creation of a single TLB entry covering the entire NAPOT region. It is also designed to be consistent with implementations that support NAPOT PTEs by splitting the NAPOT region into TLB entries covering any smaller power-of-two region sizes. For example, a 64 KiB NAPOT PTE might trigger the creation of 16 standard 4 KiB TLB entries, all with contents generated from the NAPOT PTE (even if the PTEs for the other 4 KiB regions have different contents).

In typical usage scenarios, NAPOT PTEs in the same region will have the same attributes, same PPNs, and same values for bits 5-0. RSW remains reserved for supervisor software control. It is the responsibility of the OS and/or hypervisor to configure the page tables in such a way that there are no inconsistencies between NAPOT PTEs and other NAPOT or non-NAPOT PTEs that overlap the same address range. If an update needs to be made, the OS generally should first mark all of the PTEs invalid, then issue SFENCE.VMA instruction(s) covering all 4 KiB regions within the range (either via a single SFENCE.VMA with rs1=<code>x0</code>, or with multiple SFENCE.VMA instructions with rs1≠`x0`), then update the PTE(s), as described in [sfence.vma], unless any inconsistencies are known to be benign. If any inconsistencies do exist, then the effect is the same as when SFENCE.VMA is used incorrectly: one of the translations will be chosen, but the choice is unpredictable.

If an implementation chooses to use a NAPOT PTE (or cached version thereof), it might not consult the PTE directly specified by the algorithm in [sv32algorithm] at all. Therefore, the D and A bits may not be identical across all mappings of the same address range even in typical use cases The operating system must query all NAPOT aliases of a page to determine whether that page has been accessed and/or is dirty. If the OS manually sets the A and/or D bits for a page, it is recommended that the OS also set the A and/or D bits for other NAPOT aliases as appropriate in order to avoid unnecessary traps.

Just as with normal PTEs, TLBs are permitted to cache NAPOT PTEs whose V (Valid) bit is clear.

Depending on need, the NAPOT scheme may be extended to other intermediate page sizes and/or to other levels of the page table in the future. The encoding is designed to accommodate other NAPOT sizes should that need arise. For example:

—

i	*pte.ppn*[*i*]	Description	*pte.napot_bits*
0	x xxxx xxx1	8 KiB contiguous region	1
0	x xxxx xx10	16 KiB contiguous region	2
0	x xxxx x100	32 KiB contiguous region	3
0	x xxxx 1000	64 KiB contiguous region	4
0	x xxx1 0000	128 KiB contiguous region	5
…	…	…	…
1	x xxxx xxx1	4 MiB contiguous region	1
1	x xxxx xx10	8 MiB contiguous region	2
…	…	…	…

In such a case, an implementation may or may not support all options. The discoverability mechanism for this extension would be extended to allow system software to determine which sizes are supported.

Other sizes may remain deliberately excluded, so that PPN bits not being used to indicate a valid NAPOT region size (e.g., the least-significant bit of *pte.ppn*[*i*]) may be repurposed for other uses in the future.

However, in case finer-grained intermediate page size support proves not to be useful, we have chosen to standardize only 64 KiB support as a first step.

A.22. Svpbmt Extension

Page-based memory types

Table 30. Status

Profile	v1.0.0
RVA22U64	-
RVA22S64	mandatory

1.0.0

Ratification date

==== Synopsis

This extension mandates that the satp mode Bare must be supported.

This extension was ratified as part of the RVA22 profile.

A.23. U Extension

User-level privilege mode

Table 31. Status

Profile	v1.12.0
RVA22U64	-
RVA22S64	-

1.12.0

Ratification date

2019-12

A.23.1. Synopsis

User-level privilege mode

A.23.2. Parameters

This extension has the following implementation options:

MUTABLE_MISA_U

Indicates whether or not the U extension can be disabled with the misa.U bit.

TRAP_ON_ECALL_FROM_U

Whether or not an ECALL-from-U-mode causes a synchronous exception.

The spec states that implementations may handle ECALLs transparently without raising a trap, in which case the EEI must provide a builtin.

UXLEN

Set of XLENs supported in U-mode. Can be one of:

- 32: SXLEN is always 32
- 64: SXLEN is always 64
- 3264: SXLEN can be changed (via mstatus.UXL) between 32 and 64

U_MODE_ENDIANESS

Endianess of data in U-mode. Can be one of:

- little: M-mode data is always little endian
- big: M-mode data is always big endian
- dynamic: M-mode data can be either little or big endian, depending on the CSR field mstatus.UBE

A.24. V Extension

Variable-length vector

Table 32. Status

Profile	v1.0.0
RVA22U64	optional
RVA22S64	-

1.0.0

Ratification date

==== Synopsis

TODO

A.24.1. Instructions

The following instructions are added by this extension:

vaadd.vv	**No synopsis available.**
vaadd.vx	**No synopsis available.**
vaaddu.vv	**No synopsis available.**
vaaddu.vx	**No synopsis available.**
vadc.vim	**No synopsis available.**
vadc.vvm	**No synopsis available.**
vadc.vxm	**No synopsis available.**
vadd.vi	**No synopsis available.**
vadd.vv	**No synopsis available.**
vadd.vx	**No synopsis available.**
vand.vi	**No synopsis available.**
vand.vv	**No synopsis available.**
vand.vx	**No synopsis available.**
vasub.vv	**No synopsis available.**
vasub.vx	**No synopsis available.**
vasubu.vv	**No synopsis available.**
vasubu.vx	**No synopsis available.**
vcompress.vm	**No synopsis available.**
vcpop.m	**No synopsis available.**
vdiv.vv	**No synopsis available.**

vdiv.vx	**No synopsis available.**
vdivu.vv	**No synopsis available.**
vdivu.vx	**No synopsis available.**
vfadd.vf	**No synopsis available.**
vfadd.vv	**No synopsis available.**
vfclass.v	**No synopsis available.**
vfcvt.f.x.v	**No synopsis available.**
vfcvt.f.xu.v	**No synopsis available.**
vfcvt.rtz.x.f.v	**No synopsis available.**
vfcvt.rtz.xu.f.v	**No synopsis available.**
vfcvt.x.f.v	**No synopsis available.**
vfcvt.xu.f.v	**No synopsis available.**
vfdiv.vf	**No synopsis available.**
vfdiv.vv	**No synopsis available.**
vfirst.m	**No synopsis available.**
vfmacc.vf	**No synopsis available.**
vfmacc.vv	**No synopsis available.**
vfmadd.vf	**No synopsis available.**
vfmadd.vv	**No synopsis available.**
vfmax.vf	**No synopsis available.**
vfmax.vv	**No synopsis available.**
vfmerge.vfm	**No synopsis available.**
vfmin.vf	**No synopsis available.**
vfmin.vv	**No synopsis available.**
vfmsac.vf	**No synopsis available.**
vfmsac.vv	**No synopsis available.**
vfmsub.vf	**No synopsis available.**
vfmsub.vv	**No synopsis available.**
vfmul.vf	**No synopsis available.**
vfmul.vv	**No synopsis available.**
vfmv.f.s	**No synopsis available.**
vfmv.s.f	**No synopsis available.**
vfmv.v.f	**No synopsis available.**
vfncvt.f.f.w	**No synopsis available.**

vfncvt.f.x.w	**No synopsis available.**
vfncvt.f.xu.w	**No synopsis available.**
vfncvt.rod.f.f.w	**No synopsis available.**
vfncvt.rtz.x.f.w	**No synopsis available.**
vfncvt.rtz.xu.f.w	**No synopsis available.**
vfncvt.x.f.w	**No synopsis available.**
vfncvt.xu.f.w	**No synopsis available.**
vfnmacc.vf	**No synopsis available.**
vfnmacc.vv	**No synopsis available.**
vfnmadd.vf	**No synopsis available.**
vfnmadd.vv	**No synopsis available.**
vfnmsac.vf	**No synopsis available.**
vfnmsac.vv	**No synopsis available.**
vfnmsub.vf	**No synopsis available.**
vfnmsub.vv	**No synopsis available.**
vfrdiv.vf	**No synopsis available.**
vfrec7.v	**No synopsis available.**
vfredmax.vs	**No synopsis available.**
vfredmin.vs	**No synopsis available.**
vfredosum.vs	**No synopsis available.**
vfredusum.vs	**No synopsis available.**
vfrsqrt7.v	**No synopsis available.**
vfrsub.vf	**No synopsis available.**
vfsgnj.vf	**No synopsis available.**
vfsgnj.vv	**No synopsis available.**
vfsgnjn.vf	**No synopsis available.**
vfsgnjn.vv	**No synopsis available.**
vfsgnjx.vf	**No synopsis available.**
vfsgnjx.vv	**No synopsis available.**
vfslide1down.vf	**No synopsis available.**
vfslide1up.vf	**No synopsis available.**
vfsqrt.v	**No synopsis available.**
vfsub.vf	**No synopsis available.**
vfsub.vv	**No synopsis available.**

vfwadd.vf	**No synopsis available.**
vfwadd.vv	**No synopsis available.**
vfwadd.wf	**No synopsis available.**
vfwadd.wv	**No synopsis available.**
vfwcvt.f.f.v	**No synopsis available.**
vfwcvt.f.x.v	**No synopsis available.**
vfwcvt.f.xu.v	**No synopsis available.**
vfwcvt.rtz.x.f.v	**No synopsis available.**
vfwcvt.rtz.xu.f.v	**No synopsis available.**
vfwcvt.x.f.v	**No synopsis available.**
vfwcvt.xu.f.v	**No synopsis available.**
vfwmacc.vf	**No synopsis available.**
vfwmacc.vv	**No synopsis available.**
vfwmsac.vf	**No synopsis available.**
vfwmsac.vv	**No synopsis available.**
vfwmul.vf	**No synopsis available.**
vfwmul.vv	**No synopsis available.**
vfwnmacc.vf	**No synopsis available.**
vfwnmacc.vv	**No synopsis available.**
vfwnmsac.vf	**No synopsis available.**
vfwnmsac.vv	**No synopsis available.**
vfwredosum.vs	**No synopsis available.**
vfwredusum.vs	**No synopsis available.**
vfwsub.vf	**No synopsis available.**
vfwsub.vv	**No synopsis available.**
vfwsub.wf	**No synopsis available.**
vfwsub.wv	**No synopsis available.**
vid.v	**No synopsis available.**
viota.m	**No synopsis available.**
vl1re16.v	**No synopsis available.**
vl1re32.v	**No synopsis available.**
vl1re64.v	**No synopsis available.**
vl1re8.v	**No synopsis available.**
vl2re16.v	**No synopsis available.**

vl2re32.v	**No synopsis available.**
vl2re64.v	**No synopsis available.**
vl2re8.v	**No synopsis available.**
vl4re16.v	**No synopsis available.**
vl4re32.v	**No synopsis available.**
vl4re64.v	**No synopsis available.**
vl4re8.v	**No synopsis available.**
vl8re16.v	**No synopsis available.**
vl8re32.v	**No synopsis available.**
vl8re64.v	**No synopsis available.**
vl8re8.v	**No synopsis available.**
vle16.v	**No synopsis available.**
vle16ff.v	**No synopsis available.**
vle32.v	**No synopsis available.**
vle32ff.v	**No synopsis available.**
vle64.v	**No synopsis available.**
vle64ff.v	**No synopsis available.**
vle8.v	**No synopsis available.**
vle8ff.v	**No synopsis available.**
vlm.v	**No synopsis available.**
vloxei16.v	**No synopsis available.**
vloxei32.v	**No synopsis available.**
vloxei64.v	**No synopsis available.**
vloxei8.v	**No synopsis available.**
vloxseg2ei16.v	**No synopsis available.**
vloxseg2ei32.v	**No synopsis available.**
vloxseg2ei64.v	**No synopsis available.**
vloxseg2ei8.v	**No synopsis available.**
vloxseg3ei16.v	**No synopsis available.**
vloxseg3ei32.v	**No synopsis available.**
vloxseg3ei64.v	**No synopsis available.**
vloxseg3ei8.v	**No synopsis available.**
vloxseg4ei16.v	**No synopsis available.**
vloxseg4ei32.v	**No synopsis available.**

vloxseg4ei64.v	**No synopsis available.**
vloxseg4ei8.v	**No synopsis available.**
vloxseg5ei16.v	**No synopsis available.**
vloxseg5ei32.v	**No synopsis available.**
vloxseg5ei64.v	**No synopsis available.**
vloxseg5ei8.v	**No synopsis available.**
vloxseg6ei16.v	**No synopsis available.**
vloxseg6ei32.v	**No synopsis available.**
vloxseg6ei64.v	**No synopsis available.**
vloxseg6ei8.v	**No synopsis available.**
vloxseg7ei16.v	**No synopsis available.**
vloxseg7ei32.v	**No synopsis available.**
vloxseg7ei64.v	**No synopsis available.**
vloxseg7ei8.v	**No synopsis available.**
vloxseg8ei16.v	**No synopsis available.**
vloxseg8ei32.v	**No synopsis available.**
vloxseg8ei64.v	**No synopsis available.**
vloxseg8ei8.v	**No synopsis available.**
vlse16.v	**No synopsis available.**
vlse32.v	**No synopsis available.**
vlse64.v	**No synopsis available.**
vlse8.v	**No synopsis available.**
vlseg2e16.v	**No synopsis available.**
vlseg2e16ff.v	**No synopsis available.**
vlseg2e32.v	**No synopsis available.**
vlseg2e32ff.v	**No synopsis available.**
vlseg2e64.v	**No synopsis available.**
vlseg2e64ff.v	**No synopsis available.**
vlseg2e8.v	**No synopsis available.**
vlseg2e8ff.v	**No synopsis available.**
vlseg3e16.v	**No synopsis available.**
vlseg3e16ff.v	**No synopsis available.**
vlseg3e32.v	**No synopsis available.**
vlseg3e32ff.v	**No synopsis available.**

vlseg3e64.v	**No synopsis available.**
vlseg3e64ff.v	**No synopsis available.**
vlseg3e8.v	**No synopsis available.**
vlseg3e8ff.v	**No synopsis available.**
vlseg4e16.v	**No synopsis available.**
vlseg4e16ff.v	**No synopsis available.**
vlseg4e32.v	**No synopsis available.**
vlseg4e32ff.v	**No synopsis available.**
vlseg4e64.v	**No synopsis available.**
vlseg4e64ff.v	**No synopsis available.**
vlseg4e8.v	**No synopsis available.**
vlseg4e8ff.v	**No synopsis available.**
vlseg5e16.v	**No synopsis available.**
vlseg5e16ff.v	**No synopsis available.**
vlseg5e32.v	**No synopsis available.**
vlseg5e32ff.v	**No synopsis available.**
vlseg5e64.v	**No synopsis available.**
vlseg5e64ff.v	**No synopsis available.**
vlseg5e8.v	**No synopsis available.**
vlseg5e8ff.v	**No synopsis available.**
vlseg6e16.v	**No synopsis available.**
vlseg6e16ff.v	**No synopsis available.**
vlseg6e32.v	**No synopsis available.**
vlseg6e32ff.v	**No synopsis available.**
vlseg6e64.v	**No synopsis available.**
vlseg6e64ff.v	**No synopsis available.**
vlseg6e8.v	**No synopsis available.**
vlseg6e8ff.v	**No synopsis available.**
vlseg7e16.v	**No synopsis available.**
vlseg7e16ff.v	**No synopsis available.**
vlseg7e32.v	**No synopsis available.**
vlseg7e32ff.v	**No synopsis available.**
vlseg7e64.v	**No synopsis available.**
vlseg7e64ff.v	**No synopsis available.**

vlseg7e8.v	**No synopsis available.**
vlseg7e8ff.v	**No synopsis available.**
vlseg8e16.v	**No synopsis available.**
vlseg8e16ff.v	**No synopsis available.**
vlseg8e32.v	**No synopsis available.**
vlseg8e32ff.v	**No synopsis available.**
vlseg8e64.v	**No synopsis available.**
vlseg8e64ff.v	**No synopsis available.**
vlseg8e8.v	**No synopsis available.**
vlseg8e8ff.v	**No synopsis available.**
vlsseg2e16.v	**No synopsis available.**
vlsseg2e32.v	**No synopsis available.**
vlsseg2e64.v	**No synopsis available.**
vlsseg2e8.v	**No synopsis available.**
vlsseg3e16.v	**No synopsis available.**
vlsseg3e32.v	**No synopsis available.**
vlsseg3e64.v	**No synopsis available.**
vlsseg3e8.v	**No synopsis available.**
vlsseg4e16.v	**No synopsis available.**
vlsseg4e32.v	**No synopsis available.**
vlsseg4e64.v	**No synopsis available.**
vlsseg4e8.v	**No synopsis available.**
vlsseg5e16.v	**No synopsis available.**
vlsseg5e32.v	**No synopsis available.**
vlsseg5e64.v	**No synopsis available.**
vlsseg5e8.v	**No synopsis available.**
vlsseg6e16.v	**No synopsis available.**
vlsseg6e32.v	**No synopsis available.**
vlsseg6e64.v	**No synopsis available.**
vlsseg6e8.v	**No synopsis available.**
vlsseg7e16.v	**No synopsis available.**
vlsseg7e32.v	**No synopsis available.**
vlsseg7e64.v	**No synopsis available.**
vlsseg7e8.v	**No synopsis available.**

vlsseg8e16.v	**No synopsis available.**
vlsseg8e32.v	**No synopsis available.**
vlsseg8e64.v	**No synopsis available.**
vlsseg8e8.v	**No synopsis available.**
vluxei16.v	**No synopsis available.**
vluxei32.v	**No synopsis available.**
vluxei64.v	**No synopsis available.**
vluxei8.v	**No synopsis available.**
vluxseg2ei16.v	**No synopsis available.**
vluxseg2ei32.v	**No synopsis available.**
vluxseg2ei64.v	**No synopsis available.**
vluxseg2ei8.v	**No synopsis available.**
vluxseg3ei16.v	**No synopsis available.**
vluxseg3ei32.v	**No synopsis available.**
vluxseg3ei64.v	**No synopsis available.**
vluxseg3ei8.v	**No synopsis available.**
vluxseg4ei16.v	**No synopsis available.**
vluxseg4ei32.v	**No synopsis available.**
vluxseg4ei64.v	**No synopsis available.**
vluxseg4ei8.v	**No synopsis available.**
vluxseg5ei16.v	**No synopsis available.**
vluxseg5ei32.v	**No synopsis available.**
vluxseg5ei64.v	**No synopsis available.**
vluxseg5ei8.v	**No synopsis available.**
vluxseg6ei16.v	**No synopsis available.**
vluxseg6ei32.v	**No synopsis available.**
vluxseg6ei64.v	**No synopsis available.**
vluxseg6ei8.v	**No synopsis available.**
vluxseg7ei16.v	**No synopsis available.**
vluxseg7ei32.v	**No synopsis available.**
vluxseg7ei64.v	**No synopsis available.**
vluxseg7ei8.v	**No synopsis available.**
vluxseg8ei16.v	**No synopsis available.**
vluxseg8ei32.v	**No synopsis available.**

vluxseg8ei64.v	**No synopsis available.**
vluxseg8ei8.v	**No synopsis available.**
vmacc.vv	**No synopsis available.**
vmacc.vx	**No synopsis available.**
vmadc.vi	**No synopsis available.**
vmadc.vim	**No synopsis available.**
vmadc.vv	**No synopsis available.**
vmadc.vvm	**No synopsis available.**
vmadc.vx	**No synopsis available.**
vmadc.vxm	**No synopsis available.**
vmadd.vv	**No synopsis available.**
vmadd.vx	**No synopsis available.**
vmand.mm	**No synopsis available.**
vmandn.mm	**No synopsis available.**
vmax.vv	**No synopsis available.**
vmax.vx	**No synopsis available.**
vmaxu.vv	**No synopsis available.**
vmaxu.vx	**No synopsis available.**
vmerge.vim	**No synopsis available.**
vmerge.vvm	**No synopsis available.**
vmerge.vxm	**No synopsis available.**
vmfeq.vf	**No synopsis available.**
vmfeq.vv	**No synopsis available.**
vmfge.vf	**No synopsis available.**
vmfgt.vf	**No synopsis available.**
vmfle.vf	**No synopsis available.**
vmfle.vv	**No synopsis available.**
vmflt.vf	**No synopsis available.**
vmflt.vv	**No synopsis available.**
vmfne.vf	**No synopsis available.**
vmfne.vv	**No synopsis available.**
vmin.vv	**No synopsis available.**
vmin.vx	**No synopsis available.**
vminu.vv	**No synopsis available.**

vminu.vx	**No synopsis available.**
vmnand.mm	**No synopsis available.**
vmnor.mm	**No synopsis available.**
vmor.mm	**No synopsis available.**
vmorn.mm	**No synopsis available.**
vmsbc.vv	**No synopsis available.**
vmsbc.vvm	**No synopsis available.**
vmsbc.vx	**No synopsis available.**
vmsbc.vxm	**No synopsis available.**
vmsbf.m	**No synopsis available.**
vmseq.vi	**No synopsis available.**
vmseq.vv	**No synopsis available.**
vmseq.vx	**No synopsis available.**
vmsgt.vi	**No synopsis available.**
vmsgt.vx	**No synopsis available.**
vmsgtu.vi	**No synopsis available.**
vmsgtu.vx	**No synopsis available.**
vmsif.m	**No synopsis available.**
vmsle.vi	**No synopsis available.**
vmsle.vv	**No synopsis available.**
vmsle.vx	**No synopsis available.**
vmsleu.vi	**No synopsis available.**
vmsleu.vv	**No synopsis available.**
vmsleu.vx	**No synopsis available.**
vmslt.vv	**No synopsis available.**
vmslt.vx	**No synopsis available.**
vmsltu.vv	**No synopsis available.**
vmsltu.vx	**No synopsis available.**
vmsne.vi	**No synopsis available.**
vmsne.vv	**No synopsis available.**
vmsne.vx	**No synopsis available.**
vmsof.m	**No synopsis available.**
vmul.vv	**No synopsis available.**
vmul.vx	**No synopsis available.**

vmulh.vv	**No synopsis available.**
vmulh.vx	**No synopsis available.**
vmulhsu.vv	**No synopsis available.**
vmulhsu.vx	**No synopsis available.**
vmulhu.vv	**No synopsis available.**
vmulhu.vx	**No synopsis available.**
vmv.s.x	**No synopsis available.**
vmv.v.i	**No synopsis available.**
vmv.v.v	**No synopsis available.**
vmv.v.x	**No synopsis available.**
vmv.x.s	**No synopsis available.**
vmv1r.v	**No synopsis available.**
vmv2r.v	**No synopsis available.**
vmv4r.v	**No synopsis available.**
vmv8r.v	**No synopsis available.**
vmxnor.mm	**No synopsis available.**
vmxor.mm	**No synopsis available.**
vnclip.wi	**No synopsis available.**
vnclip.wv	**No synopsis available.**
vnclip.wx	**No synopsis available.**
vnclipu.wi	**No synopsis available.**
vnclipu.wv	**No synopsis available.**
vnclipu.wx	**No synopsis available.**
vnmsac.vv	**No synopsis available.**
vnmsac.vx	**No synopsis available.**
vnmsub.vv	**No synopsis available.**
vnmsub.vx	**No synopsis available.**
vnsra.wi	**No synopsis available.**
vnsra.wv	**No synopsis available.**
vnsra.wx	**No synopsis available.**
vnsrl.wi	**No synopsis available.**
vnsrl.wv	**No synopsis available.**
vnsrl.wx	**No synopsis available.**
vor.vi	**No synopsis available.**

vor.vv	**No synopsis available.**
vor.vx	**No synopsis available.**
vredand.vs	**No synopsis available.**
vredmax.vs	**No synopsis available.**
vredmaxu.vs	**No synopsis available.**
vredmin.vs	**No synopsis available.**
vredminu.vs	**No synopsis available.**
vredor.vs	**No synopsis available.**
vredsum.vs	**No synopsis available.**
vredxor.vs	**No synopsis available.**
vrem.vv	**No synopsis available.**
vrem.vx	**No synopsis available.**
vremu.vv	**No synopsis available.**
vremu.vx	**No synopsis available.**
vrgather.vi	**No synopsis available.**
vrgather.vv	**No synopsis available.**
vrgather.vx	**No synopsis available.**
vrgatherei16.vv	**No synopsis available.**
vrsub.vi	**No synopsis available.**
vrsub.vx	**No synopsis available.**
vs1r.v	**No synopsis available.**
vs2r.v	**No synopsis available.**
vs4r.v	**No synopsis available.**
vs8r.v	**No synopsis available.**
vsadd.vi	**No synopsis available.**
vsadd.vv	**No synopsis available.**
vsadd.vx	**No synopsis available.**
vsaddu.vi	**No synopsis available.**
vsaddu.vv	**No synopsis available.**
vsaddu.vx	**No synopsis available.**
vsbc.vvm	**No synopsis available.**
vsbc.vxm	**No synopsis available.**
vse16.v	**No synopsis available.**
vse32.v	**No synopsis available.**

vse64.v	**No synopsis available.**
vse8.v	**No synopsis available.**
vsetivli	**No synopsis available.**
vsetvl	**No synopsis available.**
vsetvli	**No synopsis available.**
vsext.vf2	**No synopsis available.**
vsext.vf4	**No synopsis available.**
vsext.vf8	**No synopsis available.**
vslide1down.vx	**No synopsis available.**
vslide1up.vx	**No synopsis available.**
vslidedown.vi	**No synopsis available.**
vslidedown.vx	**No synopsis available.**
vslideup.vi	**No synopsis available.**
vslideup.vx	**No synopsis available.**
vsll.vi	**No synopsis available.**
vsll.vv	**No synopsis available.**
vsll.vx	**No synopsis available.**
vsm.v	**No synopsis available.**
vsmul.vv	**No synopsis available.**
vsmul.vx	**No synopsis available.**
vsoxei16.v	**No synopsis available.**
vsoxei32.v	**No synopsis available.**
vsoxei64.v	**No synopsis available.**
vsoxei8.v	**No synopsis available.**
vsoxseg2ei16.v	**No synopsis available.**
vsoxseg2ei32.v	**No synopsis available.**
vsoxseg2ei64.v	**No synopsis available.**
vsoxseg2ei8.v	**No synopsis available.**
vsoxseg3ei16.v	**No synopsis available.**
vsoxseg3ei32.v	**No synopsis available.**
vsoxseg3ei64.v	**No synopsis available.**
vsoxseg3ei8.v	**No synopsis available.**
vsoxseg4ei16.v	**No synopsis available.**
vsoxseg4ei32.v	**No synopsis available.**

vsoxseg4ei64.v	**No synopsis available.**
vsoxseg4ei8.v	**No synopsis available.**
vsoxseg5ei16.v	**No synopsis available.**
vsoxseg5ei32.v	**No synopsis available.**
vsoxseg5ei64.v	**No synopsis available.**
vsoxseg5ei8.v	**No synopsis available.**
vsoxseg6ei16.v	**No synopsis available.**
vsoxseg6ei32.v	**No synopsis available.**
vsoxseg6ei64.v	**No synopsis available.**
vsoxseg6ei8.v	**No synopsis available.**
vsoxseg7ei16.v	**No synopsis available.**
vsoxseg7ei32.v	**No synopsis available.**
vsoxseg7ei64.v	**No synopsis available.**
vsoxseg7ei8.v	**No synopsis available.**
vsoxseg8ei16.v	**No synopsis available.**
vsoxseg8ei32.v	**No synopsis available.**
vsoxseg8ei64.v	**No synopsis available.**
vsoxseg8ei8.v	**No synopsis available.**
vsra.vi	**No synopsis available.**
vsra.vv	**No synopsis available.**
vsra.vx	**No synopsis available.**
vsrl.vi	**No synopsis available.**
vsrl.vv	**No synopsis available.**
vsrl.vx	**No synopsis available.**
vsse16.v	**No synopsis available.**
vsse32.v	**No synopsis available.**
vsse64.v	**No synopsis available.**
vsse8.v	**No synopsis available.**
vsseg2e16.v	**No synopsis available.**
vsseg2e32.v	**No synopsis available.**
vsseg2e64.v	**No synopsis available.**
vsseg2e8.v	**No synopsis available.**
vsseg3e16.v	**No synopsis available.**
vsseg3e32.v	**No synopsis available.**

vsseg3e64.v	**No synopsis available.**
vsseg3e8.v	**No synopsis available.**
vsseg4e16.v	**No synopsis available.**
vsseg4e32.v	**No synopsis available.**
vsseg4e64.v	**No synopsis available.**
vsseg4e8.v	**No synopsis available.**
vsseg5e16.v	**No synopsis available.**
vsseg5e32.v	**No synopsis available.**
vsseg5e64.v	**No synopsis available.**
vsseg5e8.v	**No synopsis available.**
vsseg6e16.v	**No synopsis available.**
vsseg6e32.v	**No synopsis available.**
vsseg6e64.v	**No synopsis available.**
vsseg6e8.v	**No synopsis available.**
vsseg7e16.v	**No synopsis available.**
vsseg7e32.v	**No synopsis available.**
vsseg7e64.v	**No synopsis available.**
vsseg7e8.v	**No synopsis available.**
vsseg8e16.v	**No synopsis available.**
vsseg8e32.v	**No synopsis available.**
vsseg8e64.v	**No synopsis available.**
vsseg8e8.v	**No synopsis available.**
vssra.vi	**No synopsis available.**
vssra.vv	**No synopsis available.**
vssra.vx	**No synopsis available.**
vssrl.vi	**No synopsis available.**
vssrl.vv	**No synopsis available.**
vssrl.vx	**No synopsis available.**
vssseg2e16.v	**No synopsis available.**
vssseg2e32.v	**No synopsis available.**
vssseg2e64.v	**No synopsis available.**
vssseg2e8.v	**No synopsis available.**
vssseg3e16.v	**No synopsis available.**
vssseg3e32.v	**No synopsis available.**

vssseg3e64.v	**No synopsis available.**
vssseg3e8.v	**No synopsis available.**
vssseg4e16.v	**No synopsis available.**
vssseg4e32.v	**No synopsis available.**
vssseg4e64.v	**No synopsis available.**
vssseg4e8.v	**No synopsis available.**
vssseg5e16.v	**No synopsis available.**
vssseg5e32.v	**No synopsis available.**
vssseg5e64.v	**No synopsis available.**
vssseg5e8.v	**No synopsis available.**
vssseg6e16.v	**No synopsis available.**
vssseg6e32.v	**No synopsis available.**
vssseg6e64.v	**No synopsis available.**
vssseg6e8.v	**No synopsis available.**
vssseg7e16.v	**No synopsis available.**
vssseg7e32.v	**No synopsis available.**
vssseg7e64.v	**No synopsis available.**
vssseg7e8.v	**No synopsis available.**
vssseg8e16.v	**No synopsis available.**
vssseg8e32.v	**No synopsis available.**
vssseg8e64.v	**No synopsis available.**
vssseg8e8.v	**No synopsis available.**
vssub.vv	**No synopsis available.**
vssub.vx	**No synopsis available.**
vssubu.vv	**No synopsis available.**
vssubu.vx	**No synopsis available.**
vsub.vv	**No synopsis available.**
vsub.vx	**No synopsis available.**
vsuxei16.v	**No synopsis available.**
vsuxei32.v	**No synopsis available.**
vsuxei64.v	**No synopsis available.**
vsuxei8.v	**No synopsis available.**
vsuxseg2ei16.v	**No synopsis available.**
vsuxseg2ei32.v	**No synopsis available.**

vsuxseg2ei64.v	**No synopsis available.**
vsuxseg2ei8.v	**No synopsis available.**
vsuxseg3ei16.v	**No synopsis available.**
vsuxseg3ei32.v	**No synopsis available.**
vsuxseg3ei64.v	**No synopsis available.**
vsuxseg3ei8.v	**No synopsis available.**
vsuxseg4ei16.v	**No synopsis available.**
vsuxseg4ei32.v	**No synopsis available.**
vsuxseg4ei64.v	**No synopsis available.**
vsuxseg4ei8.v	**No synopsis available.**
vsuxseg5ei16.v	**No synopsis available.**
vsuxseg5ei32.v	**No synopsis available.**
vsuxseg5ei64.v	**No synopsis available.**
vsuxseg5ei8.v	**No synopsis available.**
vsuxseg6ei16.v	**No synopsis available.**
vsuxseg6ei32.v	**No synopsis available.**
vsuxseg6ei64.v	**No synopsis available.**
vsuxseg6ei8.v	**No synopsis available.**
vsuxseg7ei16.v	**No synopsis available.**
vsuxseg7ei32.v	**No synopsis available.**
vsuxseg7ei64.v	**No synopsis available.**
vsuxseg7ei8.v	**No synopsis available.**
vsuxseg8ei16.v	**No synopsis available.**
vsuxseg8ei32.v	**No synopsis available.**
vsuxseg8ei64.v	**No synopsis available.**
vsuxseg8ei8.v	**No synopsis available.**
vwadd.vv	**No synopsis available.**
vwadd.vx	**No synopsis available.**
vwadd.wv	**No synopsis available.**
vwadd.wx	**No synopsis available.**
vwaddu.vv	**No synopsis available.**
vwaddu.vx	**No synopsis available.**
vwaddu.wv	**No synopsis available.**
vwaddu.wx	**No synopsis available.**

vwmacc.vv	**No synopsis available.**
vwmacc.vx	**No synopsis available.**
vwmaccsu.vv	**No synopsis available.**
vwmaccsu.vx	**No synopsis available.**
vwmaccu.vv	**No synopsis available.**
vwmaccu.vx	**No synopsis available.**
vwmaccus.vx	**No synopsis available.**
vwmul.vv	**No synopsis available.**
vwmul.vx	**No synopsis available.**
vwmulsu.vv	**No synopsis available.**
vwmulsu.vx	**No synopsis available.**
vwmulu.vv	**No synopsis available.**
vwmulu.vx	**No synopsis available.**
vwredsum.vs	**No synopsis available.**
vwredsumu.vs	**No synopsis available.**
vwsub.vv	**No synopsis available.**
vwsub.vx	**No synopsis available.**
vwsub.wv	**No synopsis available.**
vwsub.wx	**No synopsis available.**
vwsubu.vv	**No synopsis available.**
vwsubu.vx	**No synopsis available.**
vwsubu.wv	**No synopsis available.**
vwsubu.wx	**No synopsis available.**
vxor.vi	**No synopsis available.**
vxor.vv	**No synopsis available.**
vxor.vx	**No synopsis available.**
vzext.vf2	**No synopsis available.**
vzext.vf4	**No synopsis available.**
vzext.vf8	**No synopsis available.**

A.24.2. Parameters

This extension has the following implementation options:

HW_MSTATUS_VS_DIRTY_UPDATE

Indicates whether or not hardware will write to mstatus.VS

Values are:

never	Hardware never writes mstatus.VS
precise	Hardware writes mstatus.VS to the Dirty (3) state precisely when V registers are modified
imprecise	Hardware writes mstatus.VS imprecisely. This will result in a call to unpredictable() on any attempt to read mstatus or write vector state.

MSTATUS_VS_LEGAL_VALUES

The set of values that mstatus.VS will accept from a software write.

MUTABLE_MISA_V

Indicates whether or not the V extension can be disabled with the misa.V bit.

A.25. Za128rs Extension

Reservation set requirement for RVA profiles

Table 33. Status

Profile	v1.0.0
RVA22U64	mandatory
RVA22S64	-

1.0.0

Ratification date

==== Synopsis

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

This extension was ratified as part of the RVA20 profile.

The minimum reservation set size is effectively determined by the size of atomic accesses in the A extension.

A.26. Zba Extension

Address generation instructions

Table 34. Status

Profile	v1.0.0
RVA22U64	mandatory
RVA22S64	-

1.0.0

Ratification date

2021-06

A.26.1. Synopsis

The Zba instructions can be used to accelerate the generation of addresses that index into arrays of basic types (halfword, word, doubleword) using both unsigned word-sized and XLEN-sized indices: a shifted index is added to a base address.

The shift and add instructions do a left shift of 1, 2, or 3 because these are commonly found in real-world code and because they can be implemented with a minimal amount of additional hardware beyond that of the simple adder. This avoids lengthening the critical path in implementations.

While the shift and add instructions are limited to a maximum left shift of 3, the slli instruction (from the base ISA) can be used to perform similar shifts for indexing into arrays of wider elements. The slli.uw — added in this extension — can be used when the index is to be interpreted as an unsigned word.

A.26.2. Instructions

The following instructions are added by this extension:

add.uw	**Add unsigned word**
sh1add	**Shift left by 1 and add**
sh1add.uw	**Shift unsigend word left by 1 and add**
sh2add	**Shift left by 2 and add**
sh2add.uw	**Shift unsigend word left by 2 and add**
sh3add	**Shift left by 3 and add**
sh3add.uw	**Shift unsigend word left by 3 and add**
slli.uw	**Shift left unsigned word (Immediate)**

A.27. Zbb Extension

Basic bit manipulation

Table 35. Status

Profile	v1.0.0
RVA22U64	mandatory
RVA22S64	-

1.0.0

Ratification date

2021-06

A.27.1. Synopsis

Basic bit manipulation

A.27.2. Instructions

The following instructions are added by this extension:

andn	**AND with inverted operand**
clz	**Count leading zero bits**
clzw	**Count leading zero bits in word**
cpop	**Count set bits**
cpopw	**Count set bits in word**
ctz	**Count trailing zero bits**
ctzw	**Count trailing zero bits in word**
max	**Maximum**
maxu	**Unsigned maximum**
min	**Minimum**
minu	**Unsigned minumum**
orc.b	**Bitware OR-combine, byte granule**
orn	**OR with inverted operand**
rev8	**Byte-reverse register (RV64 encoding)**
rol	**Rotate left (Register)**
rolw	**Rotate left word (Register)**
ror	**Rotate right (Register)**
rori	**Rotate right (Immediate)**

roriw	**Rotate right word (Immediate)**
rorw	**Rotate right word (Register)**
sext.b	**Sign-extend byte**
sext.h	**Sign-extend halfword**
xnor	**Exclusive NOR**
zext.h	**Zero-extend halfword**

A.28. Zbs Extension

Single-bit instructions

Table 36. Status

Profile	v1.0.0
RVA22U64	mandatory
RVA22S64	-

1.0.0

Ratification date

2021-06

Ratification document

drive.google.com/drive/u/0/folders/1_wqb-rXOVkGa6rqmugN3kwCftWDf1daU

A.28.1. Synopsis

The single-bit instructions provide a mechanism to set, clear, invert, or extract a single bit in a register. The bit is specified by its index

A.28.2. Instructions

The following instructions are added by this extension:

bclr	**Single-Bit clear (Register)**
bclri	**Single-Bit clear (Immediate)**
bext	**Single-Bit extract (Register)**
bexti	**Single-Bit extract (Immediate)**
binv	**Single-Bit invert (Register)**
binvi	**Single-Bit invert (Immediate)**
bset	**Single-Bit set (Register)**
bseti	**Single-Bit set (Immediate)**

A.29. Zfhmin Extension

Minimal half-precision Floating-point

Table 37. Status

Profile	v1.0.0
RVA22U64	mandatory
RVA22S64	-

1.0.0

Ratification date

2021-11

A.29.1. Synopsis

Zfhmin provides minimal support for 16-bit half-precision binary floating-point instructions. The Zfhmin extension is a subset of the `Zfh` extension, consisting only of data transfer and conversion instructions. Like `Zfh`, the Zfhmin extension depends on the single-precision floating-point extension, F. The expectation is that Zfhmin software primarily uses the half-precision format for storage, performing most computation in higher precision.

The Zfhmin extension includes the following instructions from the `Zfh` extension: flh, fsh, fmv.x.h, fmv.h.x, fcvt.s.h, and fcvt.h.s. If the D extension is present, the fcvt.d.h and fcvt.h.d instructions are also included. If the `Q` extension is present, the fcvt.q.h and fcvt.h.q instructions are additionally included.

Zfhmin does not include the fsgnj.h instruction, because it suffices to instead use the fsgnj.s instruction to move half-precision values between floating-point registers.

Half-precision addition, subtraction, multiplication, division, and square-root operations can be faithfully emulated by converting the half-precision operands to single-precision, performing the operation using single-precision arithmetic, then converting back to half-precision. cite:[roux:hal-01091186] Performing half-precision fused multiply-addition using this method incurs a 1-ulp error on some inputs for the RNE and RMM rounding modes.

Conversion from 8- or 16-bit integers to half-precision can be emulated by first converting to single-precision, then converting to half-precision. Conversion from 32-bit integer can be emulated by first converting to double-precision. If the D extension is not present and a 1-ulp error under RNE or RMM is tolerable, 32-bit integers can be first converted to single-precision instead. The same remark applies to conversions from 64-bit integers without the `Q` extension.

A.29.2. Instructions

The following instructions are added by this extension:

fcvt.h.s	**Convert half-precision float to a single-precision float**
fcvt.s.h	**Convert single-precision float to a half-precision float**
flh	**Half-precision floating-point load**
fmv.x.h	**Move half-precision value from floating-point to integer register**
fsh	**Half-precision floating-point store**

A.30. Zic64b Extension

64-byte cache blocks

Table 38. Status

Profile	v1.0.0
RVA22U64	mandatory
RVA22S64	-

1.0.0

Ratification date

Ratification document

github.com/riscv/riscv-profiles/releases/tag/v1.0

A.30.1. Synopsis

Cache blocks must be 64 bytes in size, naturally aligned in the address space.

This extension was ratified with the RVA20 profiles.

A.31. Zicbom Extension

Cache block management instructions

Table 39. Status

Profile	v1.0.1.pre.b34ea8a
RVA22U64	mandatory
RVA22S64	-

1.0.1.pre.b34ea8a

Ratification date

2022-05

A.31.1. Synopsis

Cache block management instructions

A.31.2. Instructions

The following instructions are added by this extension:

cbo.clean	**Cache Block Clean**
cbo.flush	**Cache Block Flush**
cbo.inval	**Cache Block Invalidate**

A.31.3. Parameters

This extension has the following implementation options:

CACHE_BLOCK_SIZE

The observable size of a cache block, in bytes

FORCE_UPGRADE_CBO_INVAL_TO_FLUSH

When true, an implementation prohibits setting menvcfg.CBIE == 11 such that all cbo.inval instructions either trap (when menvcfg.CBIE == '00') or flush (when menvcfg.CBIE == '01').

When false, an implementation allows a true INVAL operation for cbo.inval, and thus supports the setting menvcfg.CBIE == 11.

A.32. Zicbop Extension

Cache block prefetch

Table 40. Status

Profile	v1.0.1.pre.b34ea8a
RVA22U64	mandatory
RVA22S64	-

1.0.1.pre.b34ea8a

Ratification date

2022-05

A.32.1. Synopsis

Cache block prefetch instruction

A.32.2. Parameters

This extension has the following implementation options:

CACHE_BLOCK_SIZE

The observable size of a cache block, in bytes

A.33. Zicboz Extension

Cache block zero instruction

Table 41. Status

Profile	v1.0.1.pre.b34ea8a
RVA22U64	mandatory
RVA22S64	-

1.0.1.pre.b34ea8a

Ratification date

2022-05

A.33.1. Synopsis

Cache block zero instruction

A.33.2. Instructions

The following instructions are added by this extension:

cbo.zero	**Cache Block Zero**

A.33.3. Parameters

This extension has the following implementation options:

CACHE_BLOCK_SIZE

The observable size of a cache block, in bytes

A.34. Ziccamoa Extension

Main memory atomicity requirement for RVA profiles

Table 42. Status

Profile	v1.0.0
RVA22U64	mandatory
RVA22S64	-

1.0.0

Ratification date

==== Synopsis

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

This extension was ratified as part of the RVA20 profile.

A.35. Ziccif Extension

Main memory fetch requirement for RVA profiles

Table 43. Status

Profile	v1.0.0
RVA22U64	mandatory
RVA22S64	-

1.0.0

Ratification date

==== Synopsis

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.

This extension was ratified as part of the RVA20 profile.

A.36. Zicclsm Extension

Main memory misaligned requirement for RVA profiles

Table 44. Status

Profile	v1.0.0
RVA22U64	mandatory
RVA22S64	-

1.0.0

Ratification date

==== Synopsis

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

This extension was ratified as part of the RVA20 profile.

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.

A.37. Ziccrse Extension

Main memory reservability requirement for RVA profiles

Table 45. Status

Profile	**v1.0.0**
RVA22U64	mandatory
RVA22S64	-

1.0.0

Ratification date

==== Synopsis

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

This extension was ratified as part of the RVA20 profile.

A.38. Zicntr Extension

Architectural performance counters

Table 46. Status

Profile	v2.0.0
RVA22U64	mandatory
RVA22S64	-

2.0.0

Ratification date

2019-12

A.38.1. Synopsis

Architectural performance counters

A.38.2. Parameters

This extension has the following implementation options:

TIME_CSR_IMPLEMENTED

Whether or not a real hardware time CSR exists. Implementations can either provide a real CSR or emulate access at M-mode.

Possible values:

true

time/timeh exists, and accessing it will not cause an IllegalInstruction trap

false

time/timeh does not exist. Accessing the CSR will cause an IllegalInstruction trap or enter an unpredictable state, depending on TRAP_ON_UNIMPLEMENTED_CSR. Privileged software may emulate the time CSR, or may pass the exception to a lower level.

A.39. Zifencei Extension

Instruction fence

Table 47. Status

Profile	v2.0.0
RVA22U64	-
RVA22S64	mandatory

2.0.0

Ratification date

==== Synopsis

This chapter defines the "Zifencei" extension, which includes the FENCE.I instruction that provides explicit synchronization between writes to instruction memory and instruction fetches on the same hart. Currently, this instruction is the only standard mechanism to ensure that stores visible to a hart will also be visible to its instruction fetches.

We considered but did not include a "store instruction word" instruction as in cite:[majc]. JIT compilers may generate a large trace of instructions before a single FENCE.I, and amortize any instruction cache snooping/invalidation overhead by writing translated instructions to memory regions that are known not to reside in the I-cache.

The FENCE.I instruction was designed to support a wide variety of implementations. A simple implementation can flush the local instruction cache and the instruction pipeline when the FENCE.I is executed. A more complex implementation might snoop the instruction (data) cache on every data (instruction) cache miss, or use an inclusive unified private L2 cache to invalidate lines from the primary instruction cache when they are being written by a local store instruction. If instruction and data caches are kept coherent in this way, or if the memory system consists of only uncached RAMs, then just the fetch pipeline needs to be flushed at a FENCE.I.

The FENCE.I instruction was previously part of the base I instruction set. Two main issues are driving moving this out of the mandatory base, although at time of writing it is still the only standard method for maintaining instruction-fetch coherence.

First, it has been recognized that on some systems, FENCE.I will be expensive to implement and alternate mechanisms are being discussed in the memory model task group. In particular, for designs that have an incoherent instruction cache and an incoherent data cache, or where the instruction cache refill does not snoop a coherent data cache, both caches must be completely flushed when a FENCE.I instruction is encountered. This problem is exacerbated when there are multiple

levels of I and D cache in front of a unified cache or outer memory system.

Second, the instruction is not powerful enough to make available at user level in a Unix-like operating system environment. The FENCE.I only synchronizes the local hart, and the OS can reschedule the user hart to a different physical hart after the FENCE.I. This would require the OS to execute an additional FENCE.I as part of every context migration. For this reason, the standard Linux ABI has removed FENCE.I from user-level and now requires a system call to maintain instruction-fetch coherence, which allows the OS to minimize the number of FENCE.I executions required on current systems and provides forward-compatibility with future improved instruction-fetch coherence mechanisms.

Future approaches to instruction-fetch coherence under discussion include providing more restricted versions of FENCE.I that only target a given address specified in *rs1*, and/or allowing software to use an ABI that relies on machine-mode cache-maintenance operations.

A.39.1. Instructions

The following instructions are added by this extension:

fence.i	**Instruction fence**

A.40. Zihintpause Extension

PAUSE instruction

Table 48. Status

Profile	v2.0.0
RVA22U64	mandatory
RVA22S64	-

2.0.0

Ratification date

==== Synopsis

The PAUSE instruction is a HINT that indicates the current hart's rate of instruction retirement should be temporarily reduced or paused. The duration of its effect must be bounded and may be zero.

Software can use the PAUSE instruction to reduce energy consumption while executing spin-wait code sequences. Multithreaded cores might temporarily relinquish execution resources to other harts when PAUSE is executed. It is recommended that a PAUSE instruction generally be included in the code sequence for a spin-wait loop.

A future extension might add primitives similar to the x86 MONITOR/MWAIT instructions, which provide a more efficient mechanism to wait on writes to a specific memory location. However, these instructions would not supplant PAUSE. PAUSE is more appropriate when polling for non-memory events, when polling for multiple events, or when software does not know precisely what events it is polling for.

The duration of a PAUSE instruction's effect may vary significantly within and among implementations. In typical implementations this duration should be much less than the time to perform a context switch, probably more on the rough order of an on-chip cache miss latency or a cacheless access to main memory.

A series of PAUSE instructions can be used to create a cumulative delay loosely proportional to the number of PAUSE instructions. In spin-wait loops in portable code, however, only one PAUSE instruction should be used before re-evaluating loop conditions, else the hart might stall longer than optimal on some implementations, degrading system performance.

PAUSE is encoded as a FENCE instruction with *pred*=`W`, *succ*=`0`, *fm*=`0`, *rd*=`x0`, and *rs1*=`x0`.

PAUSE is encoded as a hint within the FENCE opcode because some implementations are expected to deliberately stall the PAUSE instruction until outstanding memory transactions have completed. Because the successor set is null, however, PAUSE does not *mandate* any particular memory ordering—hence,

it truly is a HINT.

Like other FENCE instructions, PAUSE cannot be used within LR/SC sequences without voiding the forward-progress guarantee.

The choice of a predecessor set of W is arbitrary, since the successor set is null. Other HINTs similar to PAUSE might be encoded with other predecessor sets.

A.41. Zihpm Extension

Programmable hardware performance counters

Table 49. Status

Profile	v2.0.0
RVA22U64	mandatory
RVA22S64	-

2.0.0

Ratification date

unknown

A.41.1. Synopsis

Programmable hardware performance counters

A.42. Zkt Extension

Data-independent execution latency

Table 50. Status

Profile	v1.0.0	v1.0.1
RVA22U64	mandatory	mandatory
RVA22S64	-	-

1.0.0

Ratification date

2021-11

1.0.1

Ratification date

Changes

- Fix typos to show that c.srli, c.srai, and c.slli are Zkt instructions in RV64.

A.42.1. Synopsis

The Zkt extension attests that the machine has data-independent execution time for a safe subset of instructions. This property is commonly called "constant-time" although should not be taken with that literal meaning.

All currently proposed cryptographic instructions (scalar `K` extension) are on this list, together with a set of relevant supporting instructions from I, M, C, and B extensions.

Failure to prevent leakage of sensitive parameters via the direct timing channel is considered a serious security vulnerability and will typically result in a CERT CVE security advisory.

Scope and Goal

An "ISA contract" is made between a programmer and the RISC-V implementation that Zkt instructions do not leak information about processed secret data (plaintext, keying information, or other "sensitive security parameters" — FIPS 140-3 term) through differences in execution latency. Zkt does *not* define a set of instructions available in the core; it just restricts the behaviour of certain instructions if those are implemented.

Currently, the scope of this document is within scalar RV32/RV64 processors. Vector cryptography instructions (and appropriate vector support instructions) will be added later, as will other security-related functions that wish to assert leakage-free execution latency properties.

Loads, stores, conditional branches are excluded, along with a set of instructions that are rarely necessary to process secret data. Also excluded are instructions for which workarounds exist in standard cryptographic middleware due to the limitations of other ISA processors.

The stated goal is that OpenSSL, BoringSSL (Android), the Linux Kernel, and similar trusted software will not have directly observable timing side channels when compiled and running on a Zkt-enabled RISC-V target. The Zkt extension explicitly states many of the common latency assumptions made by cryptography developers.

Vendors do not have to implement all of the list's instructions to be Zkt compliant; however, if they claim to have Zkt and implement any of the listed instructions, it must have data-independent latency.

For example, many simple RV32I and RV64I cores (without Multiply, Compressed, Bitmanip, or Cryptographic extensions) are technically compliant with Zkt. A constant-time AES can be implemented on them using "bit-slice" techniques, but it will be excruciatingly slow when compared to implementation with AES instructions. There are no guarantees that even a bit-sliced cipher implementation (largely based on boolean logic instructions) is secure on a core without Zkt attestation.

Out-of-order implementations adhering to Zkt are still free to fuse, crack, change or even ignore sequences of instructions, so long as the optimisations are applied deterministically, and not based on operand data. The guiding principle should be that no information about the data being operated on should be leaked based on the execution latency.

It is left to future extensions or other techniques to tackle the problem of data-independent execution in implementations which advanced out-of-order capabilities which use value prediction, or which are otherwise data-dependent.

Note to software developers

Programming techniques can only mitigate leakage directly caused by arithmetic, caches, and branches. Other ISAs have had micro-architectural issues such as Spectre, Meltdown, Speculative Store Bypass, Rogue System Register Read, Lazy FP State Restore, Bounds Check Bypass Store, TLBleed, and L1TF/Foreshadow, etc. See e.g. NSA Hardware and Firmware Security Guidance

It is not within the remit of this proposal to mitigate these *micro-architectural* leakages.

Background

- Timing attacks are much more powerful than was realised before the 2010s, which has led to a significant mitigation effort in current cryptographic code-bases.
- Cryptography developers use static and dynamic security testing tools to trace the handling of secret information and detect occasions where it influences a branch or is used for a table lookup.
- Architectural testing for Zkt can be pragmatic and semi-formal; *security by design* against basic timing attacks can usually be achieved via conscious implementation (of relevant iterative multi-cycle instructions or instructions composed of micro-ops) in way that avoids data-dependent latency.
- Laboratory testing may utilize statistical timing attack leakage analysis techniques such as those

described in ISO/IEC 17825 cite:[IS16].

- Binary executables should not contain secrets in the instruction encodings (Kerckhoffs's principle), so instruction timing may leak information about immediates, ordering of input registers, etc. There may be an exception to this in systems where a binary loader modifies the executable for purposes of relocation — and it is desirable to keep the execution location (PC) secret. This is why instructions such as LUI, AUIPC, and ADDI are on the list.
- The rules used by audit tools are relatively simple to understand. Very briefly; we call the plaintext, secret keys, expanded keys, nonces, and other such variables "secrets". A secret variable (arithmetically) modifying any other variable/register turns that into a secret too. If a secret ends up in address calculation affecting a load or store, that is a violation. If a secret affects a branch's condition, that is also a violation. A secret variable location or register becomes a non-secret via specific zeroization/sanitisation or by being declared ciphertext (or otherwise no-longer-secret information). In essence, secrets can only "touch" instructions on the Zkt list while they are secrets.

Specific Instruction Rationale

- HINT instruction forms (typically encodings with `rd=x0`) are excluded from the data-independent time requirement.
- Floating point (F, D, Q, L extensions) are currently excluded from the constant-time requirement as they have very few applications in standardised cryptography. We may consider adding floating point add, sub, multiply as a constant time requirement for some floating point extension in case a specific algorithm (such as the PQC Signature algorithm Falcon) becomes critical.
- Cryptographers typically assume division to be variable-time (while multiplication is constant time) and implement their Montgomery reduction routines with that assumption.
- Zicsr, Zifencei are excluded.
- Some instructions are on the list simply because we see no harm in including them in testing scope.

Programming Information

For background information on secure programming "models", see:

- Thomas Pornin: *"Why Constant-Time Crypto?"* (A great introduction to timing assumptions.) www.bearssl.org/constanttime.html
- Jean-Philippe Aumasson: *"Guidelines for low-level cryptography software."* (A list of recommendations.) github.com/veorq/cryptocoding
- Peter Schwabe: *"Timing Attacks and Countermeasures."* (Lecture slides — nice references.) summerschool-croatia.cs.ru.nl/2016/slides/PeterSchwabe.pdf
- Adam Langley: *"ctgrind."* (This is from 2010 but is still relevant.) www.imperialviolet.org/2010/04/01/ctgrind.html
- Kris Kwiatkowski: *"Constant-time code verification with Memory Sanitizer."* www.amongbytes.com/post/20210709-testing-constant-time/

- For early examples of timing attack vulnerabilities, see www.kb.cert.org/vuls/id/997481 and related academic papers.

Zkt listings

The following instructions are included in the Zkt subset They are listed here grouped by their original parent extension.

Note to implementers

You do not need to implement all of these instructions to implement Zkt. Rather, every one of these instructions that the core does implement must adhere to the requirements of Zkt.

RVI (Base Instruction Set)

Only basic arithmetic and `slt*` (for carry computations) are included. The data-independent timing requirement does not apply to HINT instruction encoding forms of these instructions.

RV32	RV64	Mnemonic	Instruction
✓	✓	lui *rd*, *imm*	'lui'
✓	✓	auipc *rd*, *imm*	'auipc'
✓	✓	addi *rd*, *rs1*, *imm*	'addi'
✓	✓	slti *rd*, *rs1*, *imm*	'slti'
✓	✓	sltiu *rd*, *rs1*, *imm*	'sltiu'
✓	✓	xori *rd*, *rs1*, *imm*	'xori'
✓	✓	ori *rd*, *rs1*, *imm*	'ori'
✓	✓	andi *rd*, *rs1*, *imm*	'andi'
✓	✓	slli *rd*, *rs1*, *imm*	'slli'
✓	✓	srli *rd*, *rs1*, *imm*	'srli'
✓	✓	srai *rd*, *rs1*, *imm*	'srai'
✓	✓	add *rd*, *rs1*, *rs2*	'add'
✓	✓	sub *rd*, *rs1*, *rs2*	'sub'
✓	✓	sll *rd*, *rs1*, *rs2*	'sll'
✓	✓	slt *rd*, *rs1*, *rs2*	'slt'
✓	✓	sltu *rd*, *rs1*, *rs2*	'sltu'
✓	✓	xor *rd*, *rs1*, *rs2*	'xor'
✓	✓	srl *rd*, *rs1*, *rs2*	'srl'
✓	✓	sra *rd*, *rs1*, *rs2*	'sra'
✓	✓	or *rd*, *rs1*, *rs2*	'or'
✓	✓	and *rd*, *rs1*, *rs2*	'and'

RV32	RV64	Mnemonic	Instruction
	✓	addiw *rd*, *rs1*, *imm*	'addiw'
	✓	slliw *rd*, *rs1*, *imm*	'slliw'
	✓	srliw *rd*, *rs1*, *imm*	'srliw'
	✓	sraiw *rd*, *rs1*, *imm*	'sraiw'
	✓	addw *rd*, *rs1*, *rs2*	'addw'
	✓	subw *rd*, *rs1*, *rs2*	'subw'
	✓	sllw *rd*, *rs1*, *rs2*	'sllw'
	✓	srlw *rd*, *rs1*, *rs2*	'srlw'
	✓	sraw *rd*, *rs1*, *rs2*	'sraw'

RVM (Multiply)

Multiplication is included; division and remaindering excluded.

RV32	RV64	Mnemonic	Instruction
✓	✓	mul *rd*, *rs1*, *rs2*	'mul'
✓	✓	mulh *rd*, *rs1*, *rs2*	'mulh'
✓	✓	mulhsu *rd*, *rs1*, *rs2*	'mulhsu'
✓	✓	mulhu *rd*, *rs1*, *rs2*	'mulhu'
	✓	mulw *rd*, *rs1*, *rs2*	'mulw'

RVC (Compressed)

Same criteria as in RVI. Organised by quadrants.

RV32	RV64	Mnemonic	Instruction
✓	✓	c.nop	'c_nop'
✓	✓	c.addi	'c_addi'
	✓	c.addiw	'c_addiw'
✓	✓	c.lui	'c_lui'
✓	✓	c.srli	'c_srli'
✓	✓	c.srai	'c_srai'
✓	✓	c.andi	'c_andi'
✓	✓	c.sub	'c_sub'
✓	✓	c.xor	'c_xor'
✓	✓	c.or	'c_or'
✓	✓	c.and	'c_and'

RV32	RV64	Mnemonic	Instruction
	□	c.subw	'c_subw'
	□	c.addw	'c_addw'
□	□	c.slli	'c_slli'
□	□	c.mv	'c_mv'
□	□	c.add	'c_add'

RVK (Scalar Cryptography)

All K-specific instructions are included. Additionally, `seed` CSR latency should be independent of `ES16` state output `entropy` bits, as that is a sensitive security parameter. See <<crypto_scalar_appx_es_access'.

RV32	RV64	Mnemonic	Instruction
□		aes32dsi	'aes32dsi'
□		aes32dsmi	'aes32dsmi'
□		aes32esi	'aes32esi'
□		aes32esmi	'aes32esmi'
	□	aes64ds	'aes64ds'
	□	aes64dsm	'aes64dsm'
	□	aes64es	'aes64es'
	□	aes64esm	'aes64esm'
	□	aes64im	'aes64im'
	□	aes64ks1i	'aes64ks1i'
	□	aes64ks2	'aes64ks2'
□	□	sha256sig0	'sha256sig0'
□	□	sha256sig1	'sha256sig1'
□	□	sha256sum0	'sha256sum0'
□	□	sha256sum1	'sha256sum1'
□		sha512sig0h	'sha512sig0h'
□		sha512sig0l	'sha512sig0l'
□		sha512sig1h	'sha512sig1h'
□		sha512sig1l	'sha512sig1l'
□		sha512sum0r	'sha512sum0r'
□		sha512sum1r	'sha512sum1r'
	□	sha512sig0	'sha512sig0'
	□	sha512sig1	'sha512sig1'

RV32	RV64	Mnemonic	Instruction
	✓	sha512sum0	'sha512sum0'
	✓	sha512sum1	'sha512sum1'
✓	✓	sm3p0	'sm3p0'
✓	✓	sm3p1	'sm3p1'
✓	✓	sm4ed	'sm4ed'
✓	✓	sm4ks	'sm4ks'

RVB (Bitmanip)

The `Zbkb`, `Zbkc' and `Zbkx' extensions are included in their entirety.

Note to implementers

Recall that `rev`, zip and unzip are pseudoinstructions representing specific instances of grevi, shfli and unshfli respectively.

RV32	RV64	Mnemonic	Instruction
✓	✓	clmul	'clmul-sc'
✓	✓	clmulh	'clmulh-sc'
✓	✓	xperm4	'xperm4-sc'
✓	✓	xperm8	'xperm8-sc'
✓	✓	ror	'ror-sc'
✓	✓	rol	'rol-sc'
✓	✓	rori	'rori-sc'
	✓	rorw	'rorw-sc'
	✓	rolw	'rolw-sc'
	✓	roriw	'roriw-sc'
✓	✓	andn	'andn-sc'
✓	✓	orn	'orn-sc'
✓	✓	xnor	'xnor-sc'
✓	✓	pack	'pack-sc'
✓	✓	packh	'packh-sc'
	✓	packw	'packw-sc'
✓	✓	brev8	'brev8-sc'
✓	✓	rev8	'rev8-sc'
✓		zip	'zip-sc'
✓		unzip	'unzip-sc'

Appendix B: Instruction Details

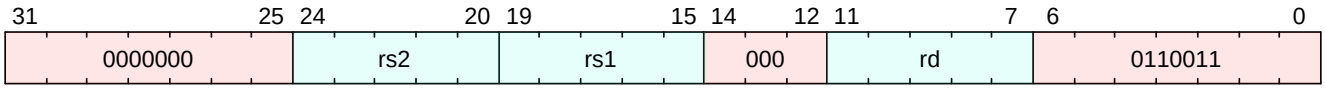
B.1. add

Integer add

This instruction is defined by:

- I, version >= 0

B.1.1. Encoding



B.1.2. Synopsis

Add the value in rs1 to rs2, and store the result in rd. Any overflow is thrown away.

B.1.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.1.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.1.5. Execution

```
X[rd] = X[rs1] + X[rs2];
```

B.1.6. Exceptions

This instruction does not generate synchronous exceptions.

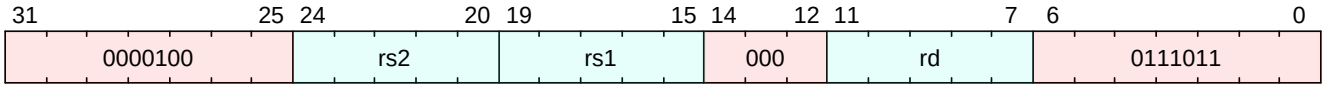
B.2. add.uw

Add unsigned word

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zba, version >= 0

B.2.1. Encoding



B.2.2. Synopsis

This instruction performs an XLEN-wide addition between rs2 and the zero-extended least-significant word of rs1.

B.2.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.2.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.2.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = X[rs2] + X[rs1][31:0];
```

B.2.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

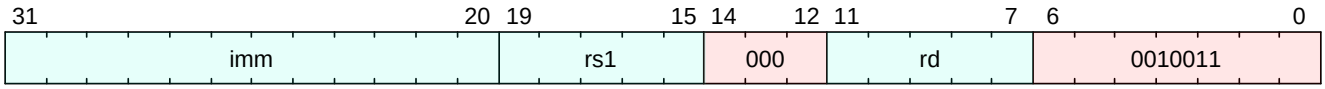
B.3. addi

Add immediate

This instruction is defined by:

- I, version >= 0

B.3.1. Encoding



B.3.2. Synopsis

Add an immediate to the value in rs1, and store the result in rd

B.3.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.3.4. Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.3.5. Execution

```
X[rd] = X[rs1] + imm;
```

B.3.6. Exceptions

This instruction does not generate synchronous exceptions.

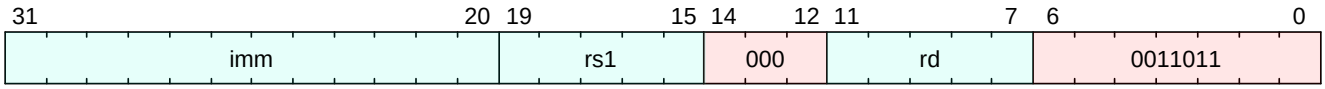
B.4. addiw

Add immediate word

This instruction is defined by:

- I, version >= 0

B.4.1. Encoding



B.4.2. Synopsis

Add an immediate to the 32-bit value in rs1, and store the sign extended result in rd

B.4.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.4.4. Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.4.5. Execution

```
XReg operand = sext(X[rs1], 31);
X[rd] = sext(operand + imm, 31);
```

B.4.6. Exceptions

This instruction does not generate synchronous exceptions.

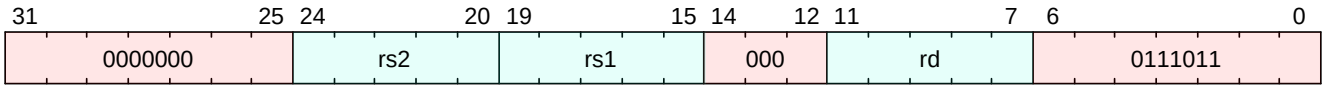
B.5. addw

Add word

This instruction is defined by:

- I, version >= 0

B.5.1. Encoding



B.5.2. Synopsis

Add the 32-bit values in rs1 to rs2, and store the sign-extended result in rd. Any overflow is thrown away.

B.5.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.5.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.5.5. Execution

```
XReg operand1 = sext(X[rs1], 31);
XReg operand2 = sext(X[rs2], 31);
X[rd] = sext(operand1 + operand2, 31);
```

B.5.6. Exceptions

This instruction does not generate synchronous exceptions.

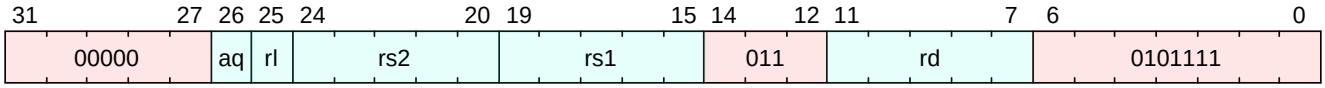
B.6. amoadd.d

Atomic fetch-and-add doubleword

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zaamo, version >= 0

B.6.1. Encoding



B.6.2. Synopsis

Atomically:

- Load the doubleword at address *rs1*
- Write the loaded value into *rd*
- Add the value of register *rs2* to the loaded value
- Write the sum to the address in *rs1*

B.6.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.6.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.6.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
X[rd] = amo<64>(virtual_address, X[rs2], AmoOperation::Add, aq, rl, $encoding);
```

B.6.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

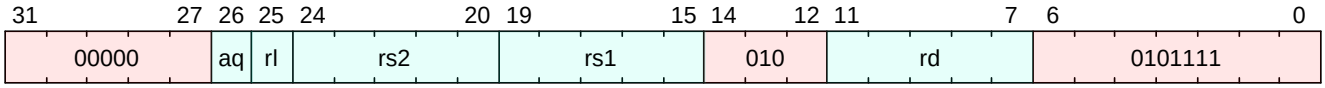
B.7. amoadd.w

Atomic fetch-and-add word

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zaamo, version >= 0

B.7.1. Encoding



B.7.2. Synopsis

Atomically:

- Load the word at address *rs1*
- Write the sign-extended value into *rd*
- Add the least-significant word of register *rs2* to the loaded value
- Write the sum to the address in *rs1*

B.7.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.7.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.7.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
X[rd] = amo<32>(virtual_address, X[rs2][31:0], AmoOperation::Add, aq, rl, $encoding);
```

B.7.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

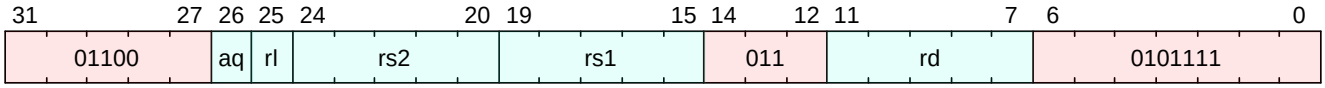
B.8. amoand.d

Atomic fetch-and-and doubleword

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zaamo, version >= 0

B.8.1. Encoding



B.8.2. Synopsis

Atomically:

- Load the doubleword at address *rs1*
- Write the loaded value into *rd*
- AND the value of register *rs2* to the loaded value
- Write the result to the address in *rs1*

B.8.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.8.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.8.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
X[rd] = amo<64>(virtual_address, X[rs2], AmoOperation::And, aq, rl, $encoding);
```

B.8.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

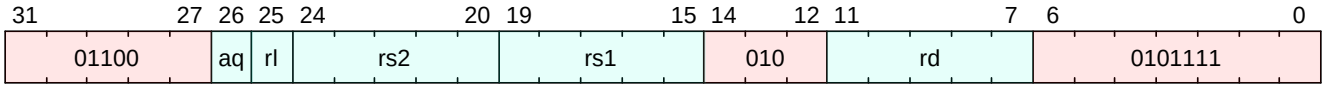
B.9. amoand.w

Atomic fetch-and-and word

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zaamo, version >= 0

B.9.1. Encoding



B.9.2. Synopsis

Atomically:

- Load the word at address *rs1*
- Write the sign-extended value into *rd*
- AND the least-significant word of register *rs2* to the loaded value
- Write the result to the address in *rs1*

B.9.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.9.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.9.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
X[rd] = amo<32>(virtual_address, X[rs2][31:0], AmoOperation::And, aq, rl, $encoding);
```

B.9.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

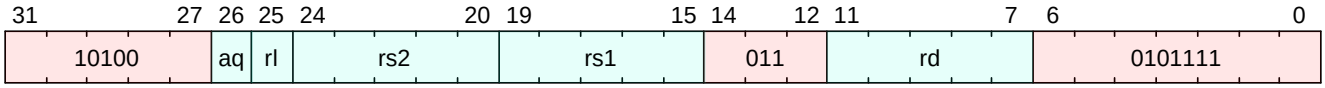
B.10. amomax.d

Atomic MAX doubleword

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zaamo, version >= 0

B.10.1. Encoding



B.10.2. Synopsis

Atomically:

- Load the doubleword at address *rs1*
- Write the loaded value into *rd*
- Signed compare the value of register *rs2* to the loaded value, and select the maximum value
- Write the maximum to the address in *rs1*

B.10.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.10.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.10.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
X[rd] = amo<64>(virtual_address, X[rs2], AmoOperation::Max, aq, rl, $encoding);
```

B.10.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

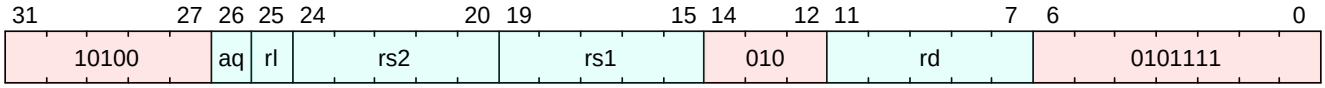
B.11. amomax.w

Atomic MAX word

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zaamo, version >= 0

B.11.1. Encoding



B.11.2. Synopsis

Atomically:

- Load the word at address *rs1*
- Write the sign-extended value into *rd*
- Signed compare the least-significant word of register *rs2* to the loaded value, and select the maximum value
- Write the maximum to the address in *rs1*

B.11.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.11.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.11.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
```

```
X[rd] = amo<32>(virtual_address, X[rs2][31:0], AmoOperation::Max, aq, rl, $encoding);
```

B.11.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

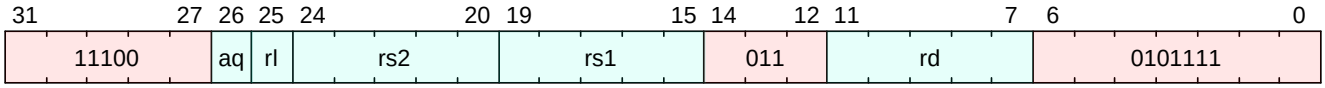
B.12. amomaxu.d

Atomic MAX unsigned doubleword

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zaamo, version >= 0

B.12.1. Encoding



B.12.2. Synopsis

Atomically:

- Load the doubleword at address *rs1*
- Write the loaded value into *rd*
- Unsigned compare the value of register *rs2* to the loaded value, and select the maximum value
- Write the maximum to the address in *rs1*

B.12.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.12.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.12.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
X[rd] = amo<64>(virtual_address, X[rs2], AmoOperation::Maxu, aq, rl, $encoding);
```

B.12.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

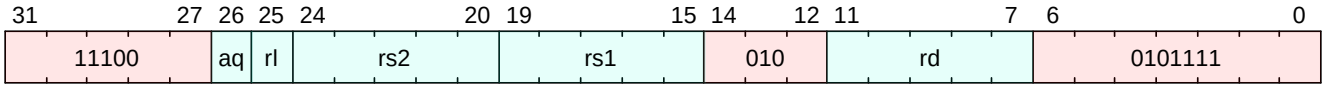
B.13. amomaxu.w

Atomic MAX unsigned word

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zaamo, version >= 0

B.13.1. Encoding



B.13.2. Synopsis

Atomically:

- Load the word at address *rs1*
- Write the sign-extended value into *rd*
- Unsigned compare the least-significant word of register *rs2* to the loaded value, and select the maximum value
- Write the maximum to the address in *rs1*

B.13.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.13.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.13.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
```

```
X[rd] = amo<32>(virtual_address, X[rs2][31:0], AmoOperation::Maxu, aq, rl, $encoding);
```

B.13.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

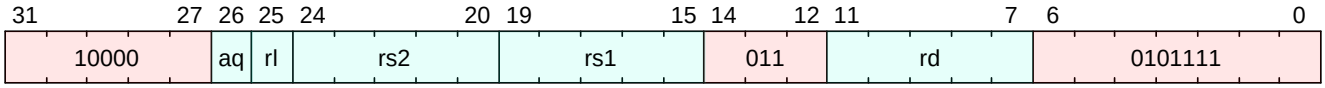
B.14. amomin.d

Atomic MIN doubleword

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zaamo, version >= 0

B.14.1. Encoding



B.14.2. Synopsis

Atomically:

- Load the doubleword at address *rs1*
- Write the loaded value into *rd*
- Signed compare the value of register *rs2* to the loaded value, and select the mimimum value
- Write the minimum to the address in *rs1*

B.14.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.14.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.14.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
X[rd] = amo<64>(virtual_address, X[rs2], AmoOperation::Min, aq, rl, $encoding);
```

B.14.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

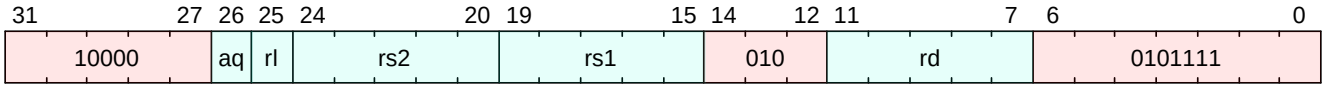
B.15. amomin.w

Atomic MIN word

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zaamo, version >= 0

B.15.1. Encoding



B.15.2. Synopsis

Atomically:

- Load the word at address *rs1*
- Write the sign-extended value into *rd*
- Signed compare the least-significant word of register *rs2* to the loaded value, and select the mimimum value
- Write the result to the address in *rs1*

B.15.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.15.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.15.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
```

```
X[rd] = amo<32>(virtual_address, X[rs2][31:0], AmoOperation::Min, aq, rl, $encoding);
```

B.15.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

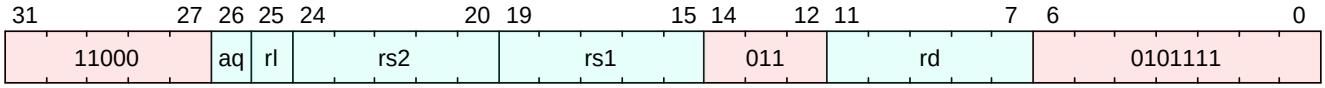
B.16. amominu.d

Atomic MIN unsigned doubleword

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zaamo, version >= 0

B.16.1. Encoding



B.16.2. Synopsis

Atomically:

- Load the doubleword at address *rs1*
- Write the loaded value into *rd*
- Unsigned compare the value of register *rs2* to the loaded value, and select the mimimum value
- Write the minimum to the address in *rs1*

B.16.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.16.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.16.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
X[rd] = amo<64>(virtual_address, X[rs2], AmoOperation::Minu, aq, rl, $encoding);
```

B.16.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

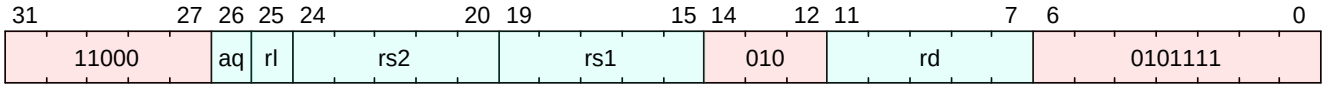
B.17. amominu.w

Atomic MIN unsigned word

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zaamo, version >= 0

B.17.1. Encoding



B.17.2. Synopsis

Atomically:

- Load the word at address *rs1*
- Write the sign-extended value into *rd*
- Unsigned compare the least-significant word of register *rs2* to the loaded word, and select the mimimum value
- Write the result to the address in *rs1*

B.17.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.17.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.17.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
```

```
X[rd] = amo<32>(virtual_address, X[rs2][31:0], AmoOperation::Minu, aq, rl, $encoding);
```

B.17.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

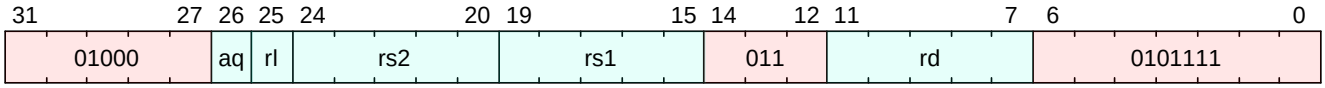
B.18. amoor.d

Atomic fetch-and-or doubleword

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zaamo, version >= 0

B.18.1. Encoding



B.18.2. Synopsis

Atomically:

- Load the doubleword at address *rs1*
- Write the loaded value into *rd*
- OR the value of register *rs2* to the loaded value
- Write the result to the address in *rs1*

B.18.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.18.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.18.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
X[rd] = amo<64>(virtual_address, X[rs2], AmoOperation::Or, aq, rl, $encoding);
```

B.18.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

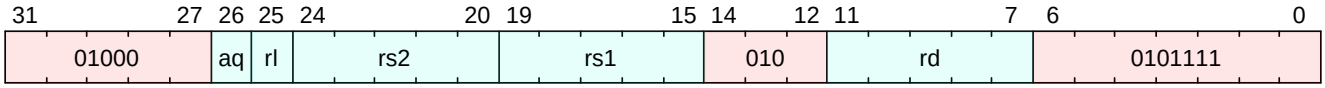
B.19. amoor.w

Atomic fetch-and-or word

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zaamo, version >= 0

B.19.1. Encoding



B.19.2. Synopsis

Atomically:

- Load the word at address *rs1*
- Write the sign-extended value into *rd*
- OR the least-significant word of register *rs2* to the loaded value
- Write the result to the address in *rs1*

B.19.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.19.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.19.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
X[rd] = amo<32>(virtual_address, X[rs2][31:0], AmoOperation::Or, aq, rl, $encoding);
```

B.19.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

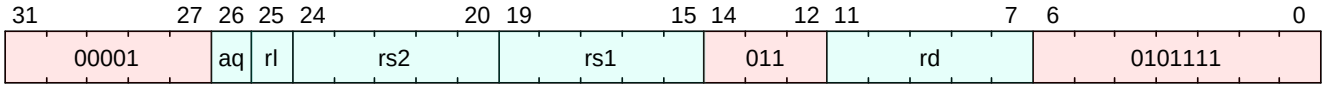
B.20. amoswap.d

Atomic SWAP doubleword

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zaamo, version >= 0

B.20.1. Encoding



B.20.2. Synopsis

Atomically:

- Load the doubleword at address *rs1*
- Write the value into *rd*
- Store the value of register *rs2* to the address in *rs1*

B.20.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.20.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.20.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
X[rd] = amo<64>(virtual_address, X[rs2], AmoOperation::Swap, aq, rl, $encoding);
```

B.20.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

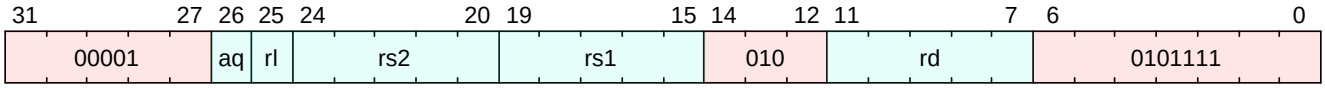
B.21. amoswap.w

Atomic SWAP word

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zaamo, version >= 0

B.21.1. Encoding



B.21.2. Synopsis

Atomically:

- Load the word at address *rs1*
- Write the sign-extended value into *rd*
- Store the least-significant word of register *rs2* to the address in *rs1*

B.21.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.21.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.21.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
X[rd] = amo<32>(virtual_address, X[rs2][31:0], AmoOperation::Swap, aq, rl, $encoding);
```

B.21.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

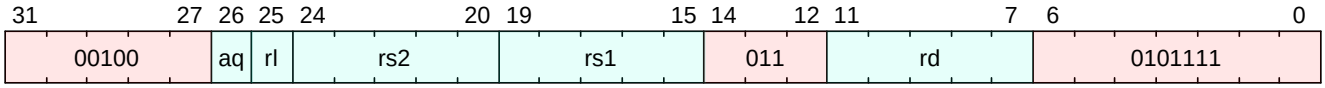
B.22. amoxor.d

Atomic fetch-and-xor doubleword

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zaamo, version >= 0

B.22.1. Encoding



B.22.2. Synopsis

Atomically:

- Load the doubleword at address *rs1*
- Write the loaded value into *rd*
- XOR the value of register *rs2* to the loaded value
- Write the result to the address in *rs1*

B.22.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.22.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.22.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
X[rd] = amo<64>(virtual_address, X[rs2], AmoOperation::Xor, aq, rl, $encoding);
```

B.22.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

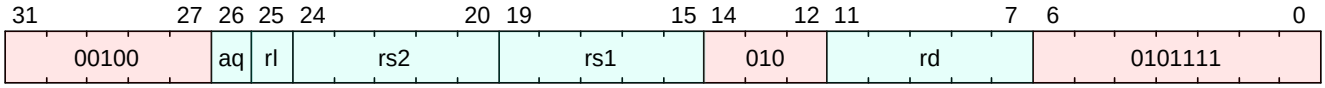
B.23. amoxor.w

Atomic fetch-and-xor word

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zaamo, version >= 0

B.23.1. Encoding



B.23.2. Synopsis

Atomically:

- Load the word at address *rs1*
- Write the sign-extended value into *rd*
- XOR the least-significant word of register *rs2* to the loaded value
- Write the result to the address in *rs1*

B.23.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.23.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.23.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
X[rd] = amo<32>(virtual_address, X[rs2][31:0], AmoOperation::Xor, aq, rl, $encoding);
```

B.23.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

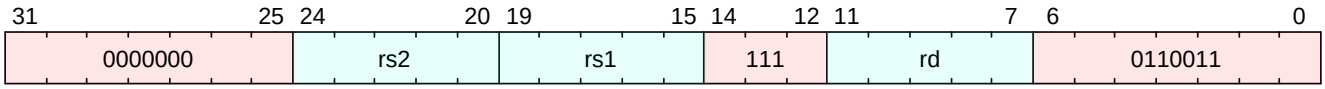
B.24. and

And

This instruction is defined by:

- I, version >= 0

B.24.1. Encoding



B.24.2. Synopsis

And rs1 with rs2, and store the result in rd

B.24.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.24.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.24.5. Execution

```
X[rd] = X[rs1] & X[rs2];
```

B.24.6. Exceptions

This instruction does not generate synchronous exceptions.

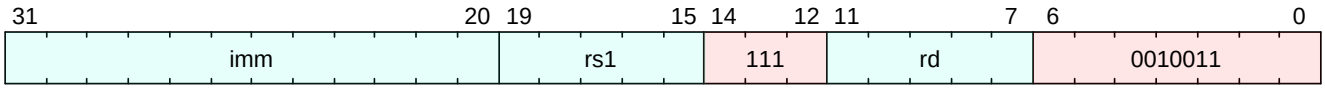
B.25. andi

And immediate

This instruction is defined by:

- I, version >= 0

B.25.1. Encoding



B.25.2. Synopsis

And an immediate to the value in rs1, and store the result in rd

B.25.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.25.4. Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.25.5. Execution

```
X[rd] = X[rs1] & imm;
```

B.25.6. Exceptions

This instruction does not generate synchronous exceptions.

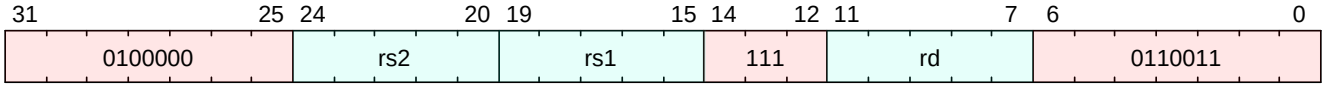
B.26. andn

AND with inverted operand

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0
  - Zbkb, version >= 0
  - Zk, version >= 0
  - Zkn, version >= 0
  - Zks, version >= 0

B.26.1. Encoding



B.26.2. Synopsis

This instruction performs the bitwise logical AND operation between `rs1` and the bitwise inversion of `rs2`.

B.26.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.26.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.26.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = X[rs2] & ~X[rs1];
```

B.26.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

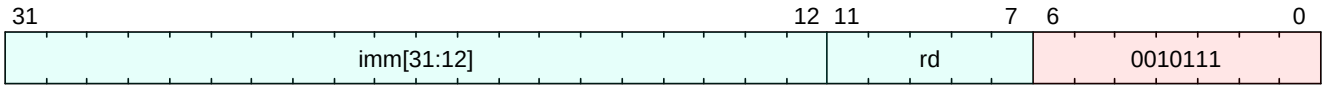
B.27. auipc

Add upper immediate to pc

This instruction is defined by:

- I, version >= 0

B.27.1. Encoding



B.27.2. Synopsis

Add an immediate to the current PC.

B.27.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.27.4. Decode Variables

```
Bits<32> imm = {$encoding[31:12], 12'd0};
Bits<5> rd = $encoding[11:7];
```

B.27.5. Execution

```
X[rd] = $pc + imm;
```

B.27.6. Exceptions

This instruction does not generate synchronous exceptions.

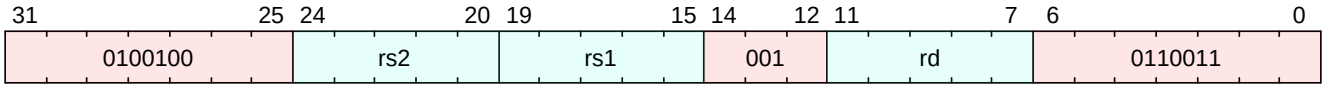
B.28. bclr

Single-Bit clear (Register)

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbs, version >= 0

B.28.1. Encoding



B.28.2. Synopsis

This instruction returns rs1 with a single bit cleared at the index specified in rs2. The index is read from the lower log2(XLEN) bits of rs2.

B.28.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.28.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.28.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg index = X[rs2] & (xlen() - 1);
X[rd] = X[rs1] & ~(1 << index);
```

B.28.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

B.29. bclri

Single-Bit clear (Immediate)

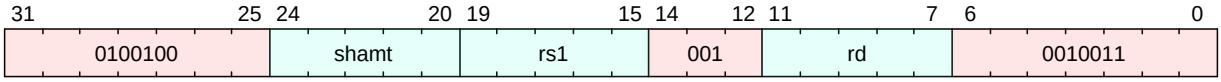
This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbs, version >= 0

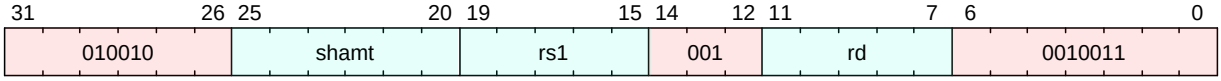
B.29.1. Encoding

This instruction has different encodings in RV32 and RV64.

RV32



RV64



B.29.2. Synopsis

This instruction returns rs1 with a single bit cleared at the index specified in shamt. The index is read from the lower log2(XLEN) bits of shamt. For RV32, the encodings corresponding to shamt[5]=1 are reserved.

B.29.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.29.4. Decode Variables

RV32

```
Bits<5> shamt = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

RV64

```
Bits<6> shamt = $encoding[25:20];
```

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.29.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg index = shamt & (xlen() - 1);
X[rd] = X[rs1] & ~(1 << index);
```

B.29.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

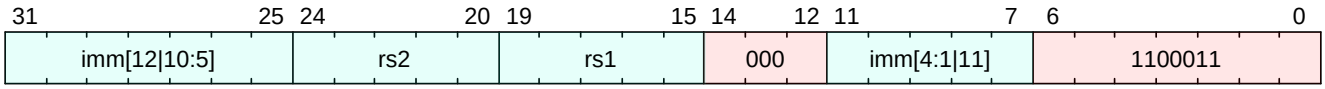
B.30. beq

Branch if equal

This instruction is defined by:

- I, version >= 0

B.30.1. Encoding



B.30.2. Synopsis

Branch to PC + imm if the value in register rs1 is equal to the value in register rs2.

Raise a `MisalignedAddress` exception if PC + imm is misaligned.

B.30.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.30.4. Decode Variables

```
Bits<13> imm = {$encoding[31], $encoding[7], $encoding[30:25], $encoding[11:8], 1'd0};
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

B.30.5. Execution

```
XReg lhs = X[rs1];
XReg rhs = X[rs2];
if (lhs == rhs) {
  jump_halfword($pc + imm);
}
```

B.30.6. Exceptions

This instruction may result in the following synchronous exceptions:

- InstructionAddressMisaligned

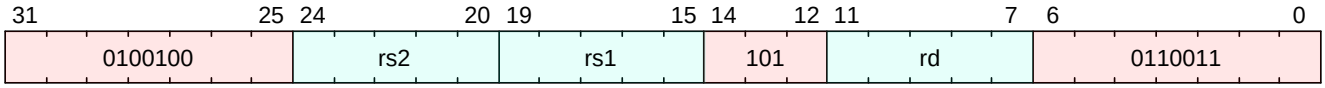
B.31. bext

Single-Bit extract (Register)

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbs, version >= 0

B.31.1. Encoding



B.31.2. Synopsis

This instruction returns a single bit extracted from rs1 at the index specified in rs2. The index is read from the lower log2(XLEN) bits of rs2.

B.31.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.31.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.31.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg index = X[rs2] & (xlen() - 1);
X[rd] = (X[rs1] >> index) & 1;
```

B.31.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

B.32. bexti

Single-Bit extract (Immediate)

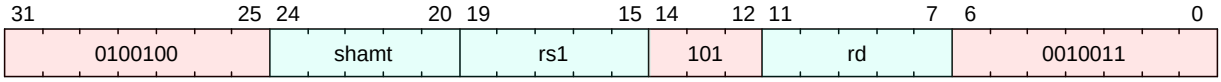
This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbs, version >= 0

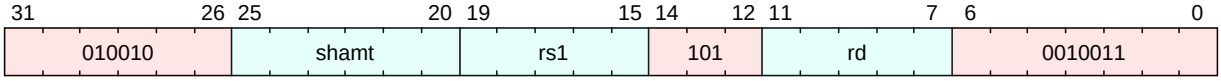
B.32.1. Encoding

This instruction has different encodings in RV32 and RV64.

RV32



RV64



B.32.2. Synopsis

This instruction returns a single bit extracted from rs1 at the index specified in rs2. The index is read from the lower log2(XLEN) bits of shamt. For RV32, the encodings corresponding to shamt[5]=1 are reserved.

B.32.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.32.4. Decode Variables

RV32

```
Bits<5> shamt = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

RV64

```
Bits<6> shamt = $encoding[25:20];
```

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.32.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg index = shamt & (xlen() - 1);
X[rd] = (X[rs1] >> index) & 1;
```

B.32.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

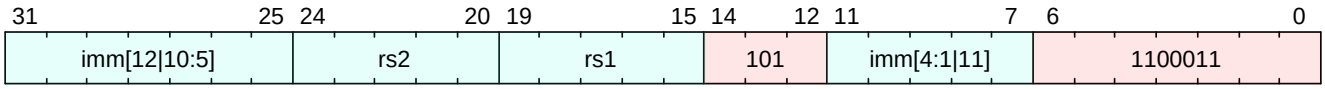
B.33. bge

Branch if greater than or equal

This instruction is defined by:

- I, version >= 0

B.33.1. Encoding



B.33.2. Synopsis

Branch to PC + imm if the signed value in register rs1 is greater than or equal to the signed value in register rs2.

Raise a `MisalignedAddress` exception if PC + imm is misaligned.

B.33.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.33.4. Decode Variables

```
Bits<13> imm = {$encoding[31], $encoding[7], $encoding[30:25], $encoding[11:8], 1'd0};
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

B.33.5. Execution

```
XReg lhs = X[rs1];
XReg rhs = X[rs2];
if ($signed(lhs) >= $signed(rhs)) {
  jump_halfword($pc + imm);
}
```

B.33.6. Exceptions

This instruction may result in the following synchronous exceptions:

- InstructionAddressMisaligned

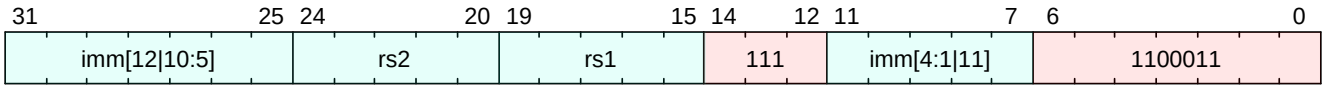
B.34. bgeu

Branch if greater than or equal unsigned

This instruction is defined by:

- I, version >= 0

B.34.1. Encoding



B.34.2. Synopsis

Branch to PC + imm if the unsigned value in register rs1 is greater than or equal to the unsigned value in register rs2.

Raise a `MisalignedAddress` exception if PC + imm is misaligned.

B.34.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.34.4. Decode Variables

```
Bits<13> imm = {$encoding[31], $encoding[7], $encoding[30:25], $encoding[11:8], 1'd0};
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

B.34.5. Execution

```
XReg lhs = X[rs1];
XReg rhs = X[rs2];
if (lhs >= rhs) {
  jump_halfword($pc + imm);
}
```

B.34.6. Exceptions

This instruction may result in the following synchronous exceptions:

- InstructionAddressMisaligned

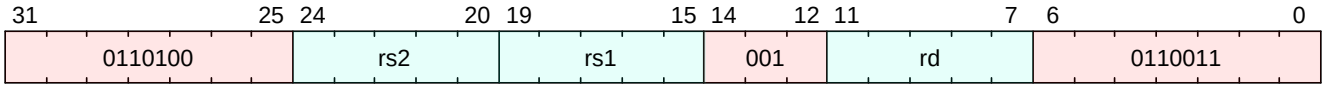
B.35. binv

Single-Bit invert (Register)

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbs, version >= 0

B.35.1. Encoding



B.35.2. Synopsis

This instruction returns rs1 with a single bit inverted at the index specified in rs2. The index is read from the lower log2(XLEN) bits of rs2.

B.35.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.35.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.35.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg index = X[rs2] & (xlen() - 1);
X[rd] = X[rs1] ^ (1 << index);
```

B.35.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

B.36. binvi

Single-Bit invert (Immediate)

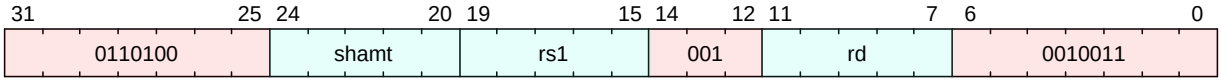
This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbs, version >= 0

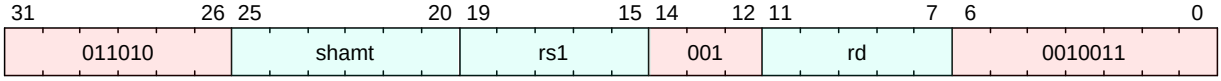
B.36.1. Encoding

This instruction has different encodings in RV32 and RV64.

RV32



RV64



B.36.2. Synopsis

This instruction returns rs1 with a single bit inverted at the index specified in shamt. The index is read from the lower log2(XLEN) bits of shamt. For RV32, the encodings corresponding to shamt[5]=1 are reserved.

B.36.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.36.4. Decode Variables

RV32

```
Bits<5> shamt = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

RV64

```
Bits<6> shamt = $encoding[25:20];
```

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.36.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg index = shamt & (xlen() - 1);
X[rd] = X[rs1] ^ (1 << index);
```

B.36.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

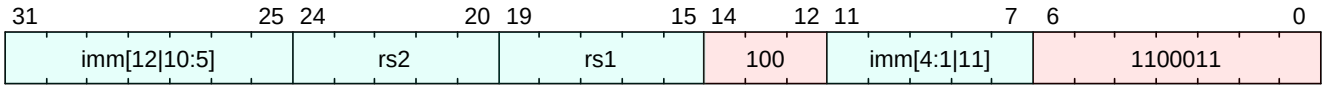
B.37. blt

Branch if less than

This instruction is defined by:

- I, version >= 0

B.37.1. Encoding



B.37.2. Synopsis

Branch to PC + imm if the signed value in register rs1 is less than the signed value in register rs2.

Raise a `MisalignedAddress` exception if PC + imm is misaligned.

B.37.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.37.4. Decode Variables

```
Bits<13> imm = {$encoding[31], $encoding[7], $encoding[30:25], $encoding[11:8], 1'd0};
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

B.37.5. Execution

```
XReg lhs = X[rs1];
XReg rhs = X[rs2];
if ($signed(lhs) < $signed(rhs)) {
  jump_halfword($pc + imm);
}
```

B.37.6. Exceptions

This instruction may result in the following synchronous exceptions:

- InstructionAddressMisaligned

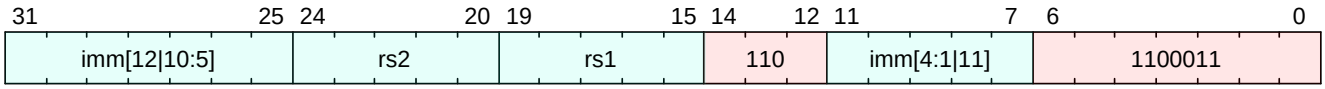
B.38. bltu

Branch if less than unsigned

This instruction is defined by:

- I, version >= 0

B.38.1. Encoding



B.38.2. Synopsis

Branch to PC + imm if the unsigned value in register rs1 is less than the unsigned value in register rs2.

Raise a `MisalignedAddress` exception if PC + imm is misaligned.

B.38.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.38.4. Decode Variables

```
Bits<13> imm = {$encoding[31], $encoding[7], $encoding[30:25], $encoding[11:8], 1'd0};
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

B.38.5. Execution

```
XReg lhs = X[rs1];
XReg rhs = X[rs2];
if (lhs < rhs) {
  jump_halfword($pc + imm);
}
```

B.38.6. Exceptions

This instruction may result in the following synchronous exceptions:

- InstructionAddressMisaligned

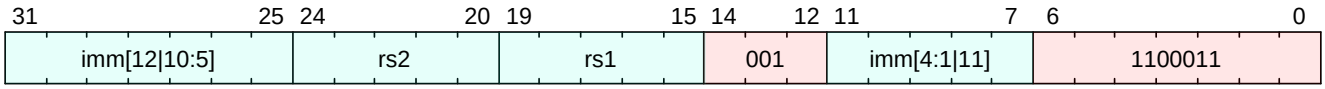
B.39. bne

Branch if not equal

This instruction is defined by:

- I, version >= 0

B.39.1. Encoding



B.39.2. Synopsis

Branch to PC + imm if the value in register rs1 is not equal to the value in register rs2.

Raise a `MisalignedAddress` exception if PC + imm is misaligned.

B.39.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.39.4. Decode Variables

```
Bits<13> imm = {$encoding[31], $encoding[7], $encoding[30:25], $encoding[11:8], 1'd0};
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

B.39.5. Execution

```
XReg lhs = X[rs1];
XReg rhs = X[rs2];
if (lhs != rhs) {
  jump_halfword($pc + imm);
}
```

B.39.6. Exceptions

This instruction may result in the following synchronous exceptions:

- InstructionAddressMisaligned

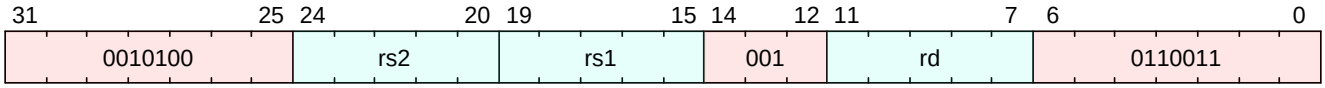
B.40. bset

Single-Bit set (Register)

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbs, version >= 0

B.40.1. Encoding



B.40.2. Synopsis

This instruction returns rs1 with a single bit set at the index specified in rs2. The index is read from the lower log2(XLEN) bits of rs2.

B.40.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.40.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.40.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg index = X[rs2] & (xlen() - 1);
X[rd] = X[rs1] | (1 << index);
```

B.40.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

B.41. bseti

Single-Bit set (Immediate)

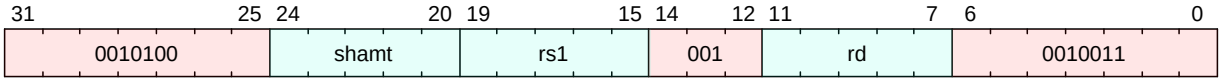
This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbs, version >= 0

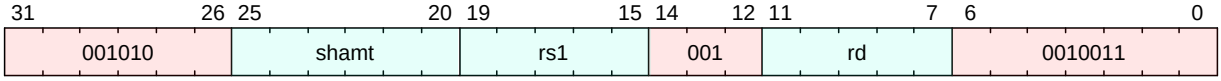
B.41.1. Encoding

This instruction has different encodings in RV32 and RV64.

RV32



RV64



B.41.2. Synopsis

This instruction returns rs1 with a single bit set at the index specified in shamt. The index is read from the lower log2(XLEN) bits of shamt. For RV32, the encodings corresponding to shamt[5]=1 are reserved.

B.41.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.41.4. Decode Variables

RV32

```
Bits<5> shamt = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

RV64

```
Bits<6> shamt = $encoding[25:20];
```

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.41.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg index = shamt & (xlen() - 1);
X[rd] = X[rs1] | (1 << index);
```

B.41.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

B.42. c.add

Add

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.42.1. Encoding

B.42.2. Synopsis

Add the value in rs2 to rd, and store the result in rd. C.ADD expands into `add rd, rd, rs2`.

B.42.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.42.4. Decode Variables

```
Bits<5> rs2 = $encoding[6:2];
Bits<5> rd = $encoding[11:7];
```

B.42.5. Execution

```
XReg t0 = X[rd];
XReg t1 = X[rs2];
X[rd] = t0 + t1;
```

B.42.6. Exceptions

This instruction does not generate synchronous exceptions.

B.43. c.addi

Add a sign-extended non-zero immediate

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.43.1. Encoding

B.43.2. Synopsis

C.ADDI adds the non-zero sign-extended 6-bit immediate to the value in register rd then writes the result to rd. C.ADDI expands into <code>addi rd, rd, imm</code>. C.ADDI is only valid when rd ≠ x0 and imm ≠ 0. The code points with rd=x0 encode the C.NOP instruction; the remaining code points with imm=0 encode HINTs.

B.43.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.43.4. Decode Variables

```
Bits<6> imm = {$encoding[12], $encoding[6:2]};
Bits<5> rd = $encoding[11:7];
```

B.43.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = X[rd] + imm;
```

B.43.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

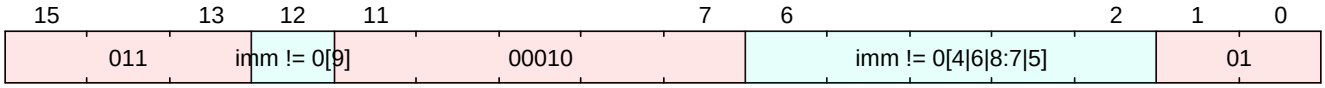
B.44. c.addi16sp

Add a sign-extended non-zero immediate

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.44.1. Encoding



B.44.2. Synopsis

C.ADDI16SP adds the non-zero sign-extended 6-bit immediate to the value in the stack pointer (sp=x2), where the immediate is scaled to represent multiples of 16 in the range (-512,496). C.ADDI16SP is used to adjust the stack pointer in procedure prologues and epilogues. It expands into <code>addi x2, x2, nzimm[9:4]</code>. C.ADDI16SP is only valid when nzimm ≠ 0; the code point with nzimm=0 is reserved.

B.44.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.44.4. Decode Variables

```
Bits<10> imm = {$encoding[12], $encoding[4:3], $encoding[5], $encoding[2], $encoding[
6], 4'd0};
```

B.44.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[2] = X[2] + imm;
```

B.44.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

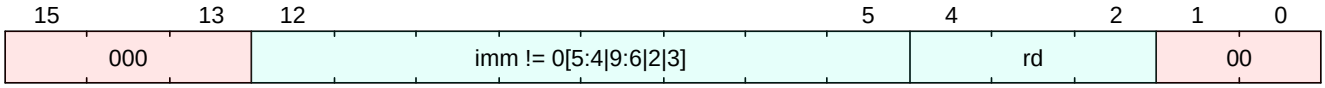
B.45. c.addi4spn

Add a zero-extended non-zero immediate, scaled by 4, to the stack pointer

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.45.1. Encoding



B.45.2. Synopsis

Adds a zero-extended non-zero immediate, scaled by 4, to the stack pointer, x2, and writes the result to rd'. This instruction is used to generate pointers to stack-allocated variables. It expands to <code>addi rd', x2, nzuimm[9:2]</code>. C.ADDI4SPN is only valid when nzuimm ≠ 0; the code points with nzuimm=0 are reserved.

B.45.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.45.4. Decode Variables

```
Bits<10> imm = {$encoding[10:7], $encoding[12:11], $encoding[5], $encoding[6], 2'd0};
Bits<3> rd = $encoding[4:2];
```

B.45.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd + 8] = X[2] + imm;
```

B.45.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

B.46. c.addiw

Add a sign-extended non-zero immediate

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.46.1. Encoding

B.46.2. Synopsis

C.ADDIW is an RV64C/RV128C-only instruction that performs the same computation as C.ADDI but produces a 32-bit result, then sign-extends result to 64 bits. C.ADDIW expands into <code>addiw rd, rd, imm</code>. The immediate can be zero for C.ADDIW, where this corresponds to <code>sext.w rd</code>. C.ADDIW is only valid when rd ≠ x0; the code points with rd=x0 are reserved.

B.46.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.46.4. Decode Variables

```
Bits<6> imm = {$encoding[12], $encoding[6:2]};
Bits<5> rd = $encoding[11:7];
```

B.46.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = sext((X[rd] + imm), 32);
```

B.46.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

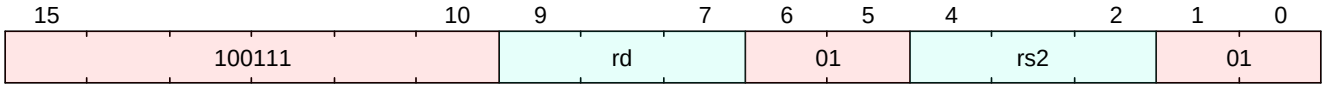
B.47. c.addw

Add word

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.47.1. Encoding



B.47.2. Synopsis

Add the 32-bit values in rs2 from rd, and store the result in rd. The rd and rs2 register indexes should be used as rd+8 and rs2+8 (registers x8-x15). C.ADDW expands into `addw rd, rd, rs2`.

B.47.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.47.4. Decode Variables

```
Bits<3> rs2 = $encoding[4:2];
Bits<3> rd = $encoding[9:7];
```

B.47.5. Execution

```
Bits<32> t0 = X[rd + 8][31:0];
Bits<32> t1 = X[rs2 + 8][31:0];
X[rd + 8] = sext(t0 + t1, 31);
```

B.47.6. Exceptions

This instruction does not generate synchronous exceptions.

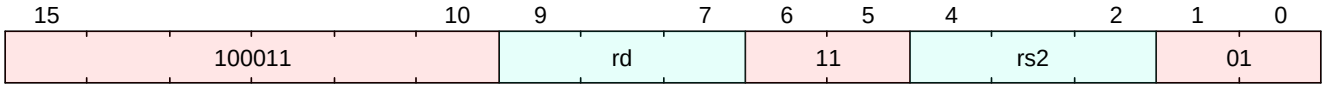
B.48. c.and

And

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.48.1. Encoding



B.48.2. Synopsis

And rd with rs2, and store the result in rd The rd and rs2 register indexes should be used as rd+8 and rs2+8 (registers x8-x15). C.AND expands into `and rd, rd, rs2`.

B.48.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.48.4. Decode Variables

```
Bits<3> rs2 = $encoding[4:2];
Bits<3> rd = $encoding[9:7];
```

B.48.5. Execution

```
XReg t0 = X[rd + 8];
XReg t1 = X[rs2 + 8];
X[rd + 8] = t0 & t1;
```

B.48.6. Exceptions

This instruction does not generate synchronous exceptions.

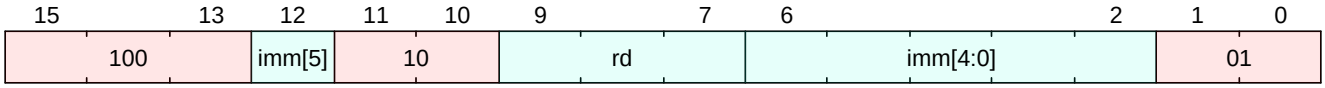
B.49. c.andi

And immediate

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.49.1. Encoding



B.49.2. Synopsis

And an immediate to the value in rd, and store the result in rd. The rd register index should be used as rd+8 (registers x8-x15). C.ANDI expands into `andi rd, rd, imm`.

B.49.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.49.4. Decode Variables

```
Bits<6> imm = {$encoding[12], $encoding[6:2]};
Bits<3> rd = $encoding[9:7];
```

B.49.5. Execution

```
X[rd + 8] = X[rd + 8] & imm;
```

B.49.6. Exceptions

This instruction does not generate synchronous exceptions.

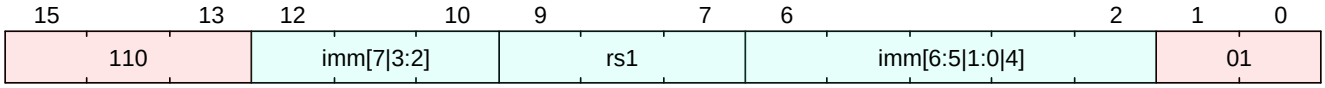
B.50. c.beqz

Branch if Equal Zero

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.50.1. Encoding



B.50.2. Synopsis

C.BEQZ performs conditional control transfers. The offset is sign-extended and added to the pc to form the branch target address. It can therefore target a ±256 B range. C.BEQZ takes the branch if the value in register rs1' is zero. It expands to beq <code>rs1, x0, offset</code>.

B.50.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.50.4. Decode Variables

```
Bits<8> imm = {$encoding[12], $encoding[6:5], $encoding[2], $encoding[11:10],
$encoding[4:3]};
Bits<3> rs1 = $encoding[9:7];
```

B.50.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
if (X[rs1] != 0) {
  jump($pc + imm);
}
```

B.50.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- InstructionAddressMisaligned

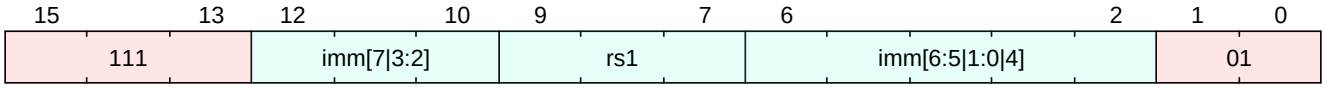
B.51. c.bnez

Branch if NOT Equal Zero

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.51.1. Encoding



B.51.2. Synopsis

C.BEQZ performs conditional control transfers. The offset is sign-extended and added to the pc to form the branch target address. It can therefore target a ±256 B range. C.BEQZ takes the branch if the value in register rs1' is NOT zero. It expands to beq <code>rs1, x0, offset</code>.

B.51.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.51.4. Decode Variables

```
Bits<8> imm = {$encoding[12], $encoding[6:5], $encoding[2], $encoding[11:10],
$encoding[4:3]};
Bits<3> rs1 = $encoding[9:7];
```

B.51.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
if (X[rs1] != 0) {
  jump($pc + imm);
}
```

B.51.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- InstructionAddressMisaligned

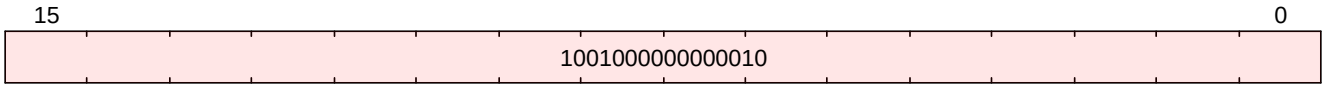
B.52. c.ebreak

Breakpoint exception.

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.52.1. Encoding



B.52.2. Synopsis

The C.EBREAK instruction is used by debuggers to cause control to be transferred back to a debugging environment. Unless overridden by an external debug environment, C.EBREAK raises a breakpoint exception and performs no other operation.

As described in the C Standard Extension for Compressed Instructions, the c.ebreak instruction performs the same operation as the EBREAK instruction.

EBREAK causes the receiving privilege mode's epc register to be set to the address of the EBREAK instruction itself, not the address of the following instruction. As EBREAK causes a synchronous exception, it is not considered to retire, and should not increment the minstret CSR.

B.52.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.52.4. Decode Variables

```
```

B.52.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
if (TRAP_ON_EBREAK) {
  raise_precise(ExceptionCode::Breakpoint, mode(), $pc);
} else {
  eei_ebreak();
```

```
}
```

B.52.6. Exceptions

This instruction may result in the following synchronous exceptions:

- Breakpoint
- IllegalInstruction

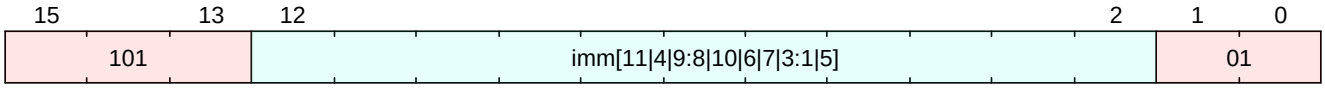
B.53. c.j

Jump

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.53.1. Encoding



B.53.2. Synopsis

C.J performs an unconditional control transfer. The offset is sign-extended and added to the pc to form the jump target address. C.J can therefore target a ±2 KiB range. It expands to jal <code>x0, offset</code>.

B.53.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.53.4. Decode Variables

```
signed Bits<12> imm = sext({$encoding[12], $encoding[8], $encoding[10:9], $encoding[
6], $encoding[7], $encoding[2], $encoding[11], $encoding[5:3], 1'd0});
```

B.53.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
jump($pc + imm);
```

B.53.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- InstructionAddressMisaligned

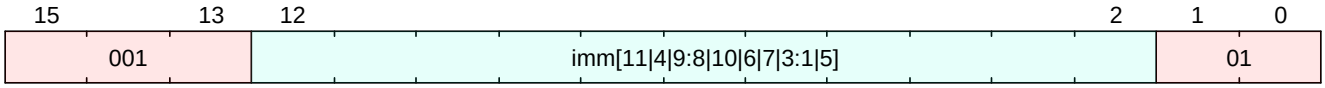
B.54. c.jal

Jump and Link

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.54.1. Encoding



B.54.2. Synopsis

C.JAL is an RV32C-only instruction that performs the same operation as C.J, but additionally writes the address of the instruction following the jump (pc+2) to the link register, x1. It expands to `jal x1, offset`.

B.54.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.54.4. Decode Variables

```
signed Bits<12> imm = sext({$encoding[12], $encoding[8], $encoding[10:9], $encoding[
6], $encoding[7], $encoding[2], $encoding[11], $encoding[5:3], 1'd0});
```

B.54.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg retrun_addr = $pc + 2;
jump_halfword($pc + imm);
X[1] = retrun_addr;
```

B.54.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

- InstructionAddressMisaligned

B.55. c.jalr

Jump and Link Register.

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.55.1. Encoding

B.55.2. Synopsis

C.JALR (jump and link register) performs the same operation as C.JR, but additionally writes the address of the instruction following the jump (pc+2) to the link register, x1. C.JALR expands to jalr x1, 0(rs1).

B.55.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.55.4. Decode Variables

```
Bits<5> rs1 = $encoding[11:7];
```

B.55.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg returnaddr;
returnaddr = $pc + 2;
jump(X[rs1]);
X[1] = returnaddr;
```

B.55.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

- InstructionAddressMisaligned

B.56. c.jr

Jump Register

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.56.1. Encoding

B.56.2. Synopsis

C.JR (jump register) performs an unconditional control transfer to the address in register rs1. C.JR expands to jalr x0, 0(rs1).

B.56.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.56.4. Decode Variables

```
Bits<5> rs1 = $encoding[11:7];
```

B.56.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
jump(X[rs1]);
```

B.56.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- InstructionAddressMisaligned

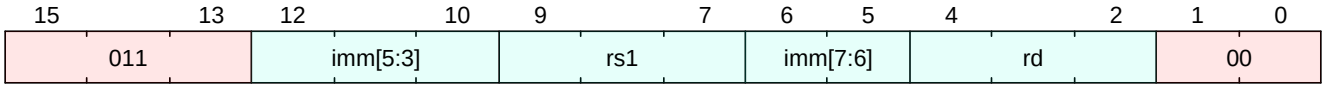
B.57. c.ld

Load double

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.57.1. Encoding



B.57.2. Synopsis

Loads a 64-bit value from memory into register rd. It computes an effective address by adding the zero-extended offset, scaled by 8, to the base address in register rs1. It expands to `ld rd, offset(rs1)`.

B.57.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.57.4. Decode Variables

```
Bits<8> imm = {$encoding[6:5], $encoding[12:10], 3'd0};
Bits<3> rd = $encoding[4:2];
Bits<3> rs1 = $encoding[9:7];
```

B.57.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1] + imm;
X[rd] = sext(read_memory<64>(virtual_address, $encoding), 64);
```

B.57.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

- LoadAccessFault
- LoadAddressMisaligned
- LoadPageFault

B.58. c.ldsp

Load doubleword from stack pointer

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.58.1. Encoding

B.58.2. Synopsis

C.LDSP is an RV64C/RV128C-only instruction that loads a 64-bit value from memory into register rd. It computes its effective address by adding the zero-extended offset, scaled by 8, to the stack pointer, x2. It expands to ld <code>rd, offset(x2)</code>. C.LDSP is only valid when rd ≠ x0 the code points with rd=x0 are reserved.

B.58.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.58.4. Decode Variables

```
Bits<9> imm = {$encoding[4:2], $encoding[12], $encoding[6:5], 3'd0};
Bits<5> rd = $encoding[11:7];
```

B.58.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[2] + imm;
X[rd] = read_memory<64>(virtual_address, $encoding);
```

B.58.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

- LoadAccessFault
- LoadAddressMisaligned
- LoadPageFault

B.59. c.li

Load the sign-extended 6-bit immediate

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.59.1. Encoding

B.59.2. Synopsis

C.LI loads the sign-extended 6-bit immediate, imm, into register rd. C.LI expands into <code>addi rd, x0, imm</code>. C.LI is only valid when rd ≠ x0; the code points with rd=x0 encode HINTs.

B.59.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.59.4. Decode Variables

```
Bits<6> imm = {$encoding[12], $encoding[6:2]};
Bits<5> rd = $encoding[11:7];
```

B.59.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = imm;
```

B.59.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

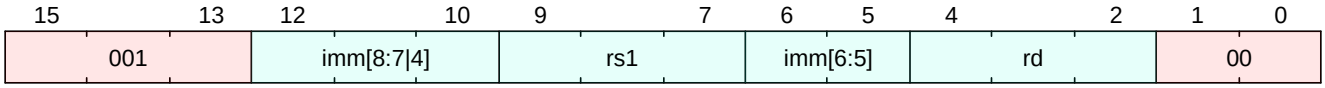
B.60. c.lq

Load quadruple

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.60.1. Encoding



B.60.2. Synopsis

Loads a 128-bit value from memory into register rd. It computes an effective address by adding the zero-extended offset, scaled by 16, to the base address in register rs1. It expands to `lq rd, offset(rs1)`.

B.60.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.60.4. Decode Variables

```
Bits<9> imm = {$encoding[12:11], $encoding[6:5], $encoding[10], 4'd0};
Bits<3> rd = $encoding[4:2];
Bits<3> rs1 = $encoding[9:7];
```

B.60.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1] + imm;
X[rd] = sext(read_memory<128>(virtual_address, $encoding), 128);
```

B.60.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

- LoadAccessFault
- LoadAddressMisaligned
- LoadPageFault

B.61. c.lqsp

Load quadruple word from stack pointer

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.61.1. Encoding

B.61.2. Synopsis

C.LQSP is an RV128C-only instruction that loads a 128-bit value from memory into register rd. It computes its effective address by adding the zero-extended offset, scaled by 16, to the stack pointer, x2. It expands to <code>lq</code> <code>rd, offset(x2)</code>. C.LQSP is only valid when rd ≠ x0 the code points with rd=x0 are reserved.

B.61.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.61.4. Decode Variables

```
Bits<10> imm = {$encoding[5:2], $encoding[12], $encoding[6], 4'd0};
Bits<5> rd = $encoding[11:7];
```

B.61.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[2] + imm;
X[rd] = read_memory<128>(virtual_address, $encoding);
```

B.61.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

- LoadAccessFault
- LoadAddressMisaligned
- LoadPageFault

B.62. c.lui

Load the non-zero 6-bit immediate field into bits 17-12 of the destination register

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.62.1. Encoding

B.62.2. Synopsis

C.LUI loads the non-zero 6-bit immediate field into bits 17-12 of the destination register, clears the bottom 12 bits, and sign-extends bit 17 into all higher bits of the destination. C.LUI expands into <code>lui rd, imm</code>. C.LUI is only valid when rd≠x0 and rd≠x2, and when the immediate is not equal to zero. The code points with imm=0 are reserved; the remaining code points with rd=x0 are HINTs; and the remaining code points with rd=x2 correspond to the C.ADDI16SP instruction

B.62.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.62.4. Decode Variables

```
Bits<18> imm = {$encoding[12], $encoding[6:2], 12'd0};
Bits<5> rd = $encoding[11:7];
```

B.62.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = imm;
```

B.62.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

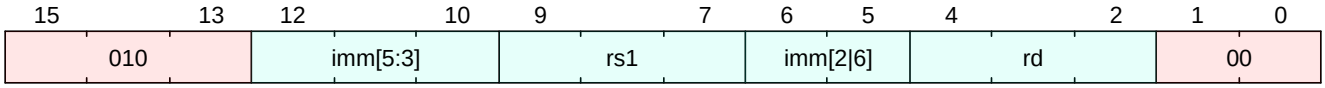
B.63. c.lw

Load word

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.63.1. Encoding



B.63.2. Synopsis

Loads a 32-bit value from memory into register rd. It computes an effective address by adding the zero-extended offset, scaled by 4, to the base address in register rs1. It expands to lw `rd, offset(rs1)`.

B.63.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.63.4. Decode Variables

```
Bits<7> imm = {$encoding[5], $encoding[12:10], $encoding[6], 2'd0};
Bits<3> rd = $encoding[4:2];
Bits<3> rs1 = $encoding[9:7];
```

B.63.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1] + imm;
X[rd] = sext(read_memory<32>(virtual_address, $encoding), 32);
```

B.63.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

- LoadAccessFault
- LoadAddressMisaligned
- LoadPageFault

B.64. c.lwsp

Load word from stack pointer

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.64.1. Encoding

B.64.2. Synopsis

Loads a 32-bit value from memory into register rd. It computes an effective address by adding the zero-extended offset, scaled by 4, to the stack pointer, x2. It expands to lw <code>rd, offset(x2)</code>. C.LWSP is only valid when rd ≠ x0. The code points with rd=x0 are reserved.

B.64.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.64.4. Decode Variables

```
Bits<8> imm = {$encoding[3:2], $encoding[12], $encoding[6:4], 2'd0};
Bits<5> rd = $encoding[11:7];
```

B.64.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[2] + imm;
X[rd] = sext(read_memory<32>(virtual_address, $encoding), 32);
```

B.64.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

- LoadAccessFault
- LoadAddressMisaligned
- LoadPageFault

B.65. c.mv

Move Register

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.65.1. Encoding

B.65.2. Synopsis

C.MV (move register) performs copy of the data in register rs2 to register rd C.MV expands to addi rd, x0, rs2.

B.65.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.65.4. Decode Variables

```
Bits<5> rd = $encoding[11:7];
Bits<5> rs2 = $encoding[6:2];
```

B.65.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = X[rs2];
```

B.65.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

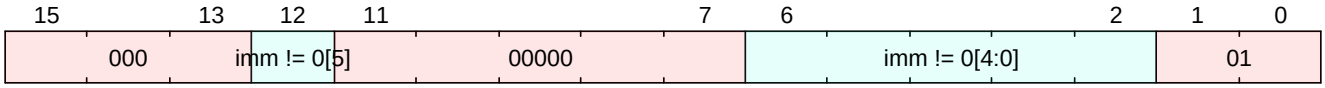
B.66. c.nop

Non-operation

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.66.1. Encoding



B.66.2. Synopsis

C.NOP expands into `addi x0, x0, imm`.

B.66.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.66.4. Decode Variables

```
Bits<6> imm = {$encoding[12], $encoding[6:2]};
```

B.66.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
```

B.66.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

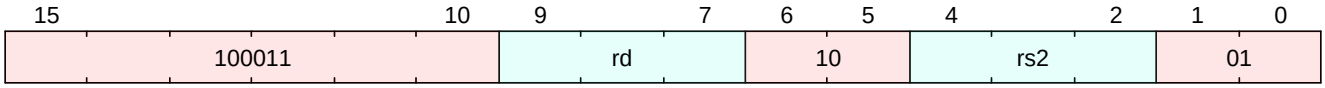
B.67. c.or

Or

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.67.1. Encoding



B.67.2. Synopsis

Or rd with rs2, and store the result in rd The rd and rs2 register indexes should be used as rd+8 and rs2+8 (registers x8-x15). C.OR expands into `or rd, rd, rs2`.

B.67.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.67.4. Decode Variables

```
Bits<3> rs2 = $encoding[4:2];
Bits<3> rd = $encoding[9:7];
```

B.67.5. Execution

```
XReg t0 = X[rd + 8];
XReg t1 = X[rs2 + 8];
X[rd + 8] = t0 | t1;
```

B.67.6. Exceptions

This instruction does not generate synchronous exceptions.

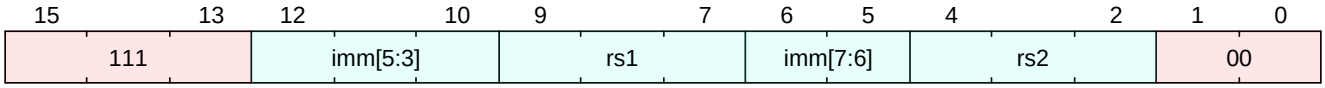
B.68. c.sd

Store double

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.68.1. Encoding



B.68.2. Synopsis

Stores a 64-bit value in register rs2 to memory. It computes an effective address by adding the zero-extended offset, scaled by 8, to the base address in register rs1. It expands to sd `rs2, offset(rs1)`.

B.68.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.68.4. Decode Variables

```
Bits<8> imm = {$encoding[6:5], $encoding[12:10], 3'd0};
Bits<3> rs2 = $encoding[4:2];
Bits<3> rs1 = $encoding[9:7];
```

B.68.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1] + imm;
write_memory<64>(virtual_address, X[rs2], $encoding);
```

B.68.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

B.69. c.sdsp

Store doubleword to stack

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.69.1. Encoding

B.69.2. Synopsis

Stores a 64-bit value in register rs2 to memory. It computes an effective address by adding the zero-extended offset, scaled by 8, to the stack pointer, x2. It expands to `sd rs2, offset(x2)`.

B.69.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.69.4. Decode Variables

```
Bits<9> imm = {$encoding[9:7], $encoding[12:10], 3'd0};
Bits<5> rs2 = $encoding[6:2];
```

B.69.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[2] + imm;
write_memory<64>(virtual_address, X[rs2], $encoding);
```

B.69.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault

- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

B.70. c.slli

Shift left logical immediate

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.70.1. Encoding

B.70.2. Synopsis

Shift the value in rd left by shamt, and store the result back in rd. C.SLLI expands into `slli rd, rd, shamt`.

B.70.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.70.4. Decode Variables

```
Bits<6> shamt = {$encoding[12], $encoding[6:2]};
Bits<5> rd = $encoding[11:7];
```

B.70.5. Execution

```
X[rd] = X[rd] << shamt;
```

B.70.6. Exceptions

This instruction does not generate synchronous exceptions.

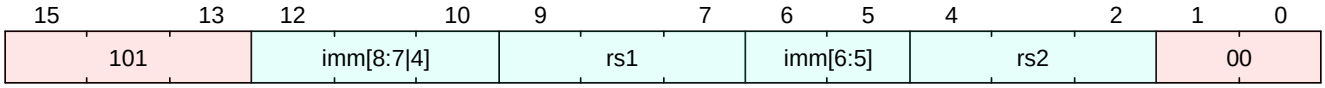
B.71. c.sq

Store quadruple

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.71.1. Encoding



B.71.2. Synopsis

Stores a 128-bit value in register rs2 to memory. It computes an effective address by adding the zero-extended offset, scaled by 16, to the base address in register rs1. It expands to `sq rs2, offset(rs1)`.

B.71.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.71.4. Decode Variables

```
Bits<9> imm = {$encoding[12:11], $encoding[6:5], $encoding[10], 4'd0};
Bits<3> rs2 = $encoding[4:2];
Bits<3> rs1 = $encoding[9:7];
```

B.71.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1] + imm;
write_memory<128>(virtual_address, X[rs2], $encoding);
```

B.71.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

B.72. c.sqsp

Store quadruple word to stack

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.72.1. Encoding

B.72.2. Synopsis

Stores a 128-bit value in register rs2 to memory. It computes an effective address by adding the zero-extended offset, scaled by 16, to the stack pointer, x2. It expands to `sq rs2, offset(x2)`.

B.72.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.72.4. Decode Variables

```
Bits<10> imm = {$encoding[10:7], $encoding[12:11], 4'd0};
Bits<5> rs2 = $encoding[6:2];
```

B.72.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[2] + imm;
write_memory<128>(virtual_address, X[rs2], $encoding);
```

B.72.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault

- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

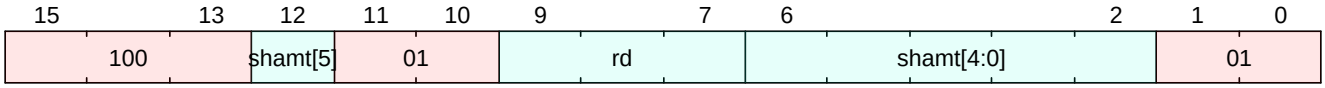
B.73. c.srai

Shift right arithmetical immediate

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.73.1. Encoding



B.73.2. Synopsis

Arithmetic shift (the original sign bit is copied into the vacated upper bits) the value in rd right by shamt, and store the result in rd. The rd register index should be used as rd+8 (registers x8-x15). C.SRAI expands into `srai rd, rd, shamt`.

B.73.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.73.4. Decode Variables

```
Bits<6> shamt = {$encoding[12], $encoding[6:2]};
Bits<3> rd = $encoding[9:7];
```

B.73.5. Execution

```
X[rd + 8] = X[rd + 8] >>> shamt;
```

B.73.6. Exceptions

This instruction does not generate synchronous exceptions.

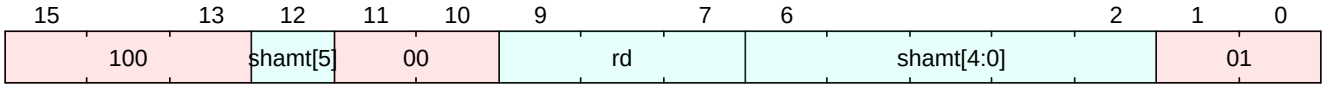
B.74. c.srli

Shift right logical immediate

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.74.1. Encoding



B.74.2. Synopsis

Shift the value in rd right by shamt, and store the result back in rd. The rd register index should be used as rd+8 (registers x8-x15). C.SRLI expands into `srli rd, rd, shamt`.

B.74.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.74.4. Decode Variables

```
Bits<6> shamt = {$encoding[12], $encoding[6:2]};
Bits<3> rd = $encoding[9:7];
```

B.74.5. Execution

```
X[rd + 8] = X[rd + 8] >> shamt;
```

B.74.6. Exceptions

This instruction does not generate synchronous exceptions.

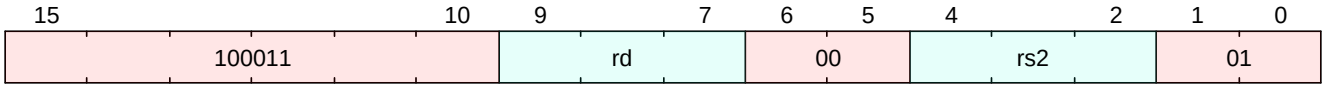
B.75. c.sub

Subtract

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.75.1. Encoding



B.75.2. Synopsis

Subtract the value in rs2 from rd, and store the result in rd. The rd and rs2 register indexes should be used as rd+8 and rs2+8 (registers x8-x15). C.SUB expands into `sub rd, rd, rs2`.

B.75.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.75.4. Decode Variables

```
Bits<3> rs2 = $encoding[4:2];
Bits<3> rd = $encoding[9:7];
```

B.75.5. Execution

```
XReg t0 = X[rd + 8];
XReg t1 = X[rs2 + 8];
X[rd + 8] = t0 - t1;
```

B.75.6. Exceptions

This instruction does not generate synchronous exceptions.

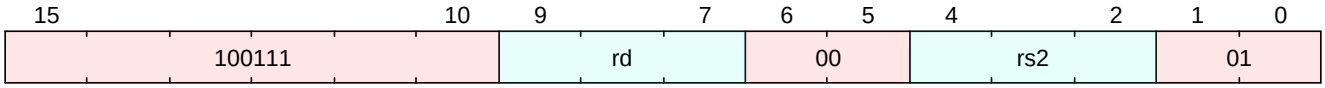
B.76. c.subw

Subtract word

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.76.1. Encoding



B.76.2. Synopsis

Subtract the 32-bit values in rs2 from rd, and store the result in rd. The rd and rs2 register indexes should be used as rd+8 and rs2+8 (registers x8-x15). C.SUBW expands into `subw rd, rd, rs2`.

B.76.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.76.4. Decode Variables

```
Bits<3> rs2 = $encoding[4:2];
Bits<3> rd = $encoding[9:7];
```

B.76.5. Execution

```
Bits<32> t0 = X[rd + 8][31:0];
Bits<32> t1 = X[rs2 + 8][31:0];
X[rd + 8] = sext(t0 - t1, 31);
```

B.76.6. Exceptions

This instruction does not generate synchronous exceptions.

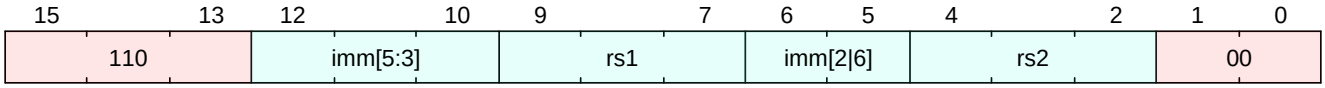
B.77. c.sw

Store word

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.77.1. Encoding



B.77.2. Synopsis

Stores a 32-bit value in register rs2 to memory. It computes an effective address by adding the zero-extended offset, scaled by 4, to the base address in register rs1. It expands to sw `rs2, offset(rs1)`.

B.77.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.77.4. Decode Variables

```
Bits<7> imm = {$encoding[5], $encoding[12:10], $encoding[6], 2'd0};
Bits<3> rs2 = $encoding[4:2];
Bits<3> rs1 = $encoding[9:7];
```

B.77.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1] + imm;
write_memory<32>(virtual_address, X[rs2][31:0], $encoding);
```

B.77.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

B.78. c.swsp

Store word to stack

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.78.1. Encoding

B.78.2. Synopsis

Stores a 32-bit value in register rs2 to memory. It computes an effective address by adding the zero-extended offset, scaled by 4, to the stack pointer, x2. It expands to sw `rs2, offset(x2)`.

B.78.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.78.4. Decode Variables

```
Bits<8> imm = {$encoding[8:7], $encoding[12:9], 2'd0};
Bits<5> rs2 = $encoding[6:2];
```

B.78.5. Execution

```
if (implemented?(ExtensionName::C) && (CSR[misa].C == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[2] + imm;
write_memory<32>(virtual_address, X[rs2][31:0], $encoding);
```

B.78.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault

- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

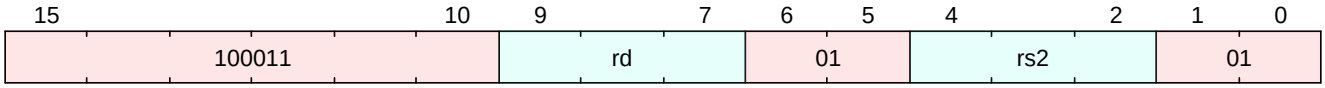
B.79. c.xor

Exclusive Or

This instruction is defined by:

- anyOf:
  - C, version >= 0
  - Zca, version >= 0

B.79.1. Encoding



B.79.2. Synopsis

Exclusive or rd with rs2, and store the result in rd The rd and rs2 register indexes should be used as rd+8 and rs2+8 (registers x8-x15). C.XOR expands into `xor rd, rd, rs2`.

B.79.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.79.4. Decode Variables

```
Bits<3> rs2 = $encoding[4:2];
Bits<3> rd = $encoding[9:7];
```

B.79.5. Execution

```
XReg t0 = X[rd + 8];
XReg t1 = X[rs2 + 8];
X[rd + 8] = t0 ^ t1;
```

B.79.6. Exceptions

This instruction does not generate synchronous exceptions.

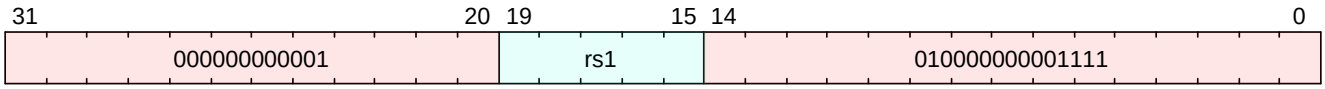
B.80. cbo.clean

Cache Block Clean

This instruction is defined by:

- Zicbom, version >= 0

B.80.1. Encoding



B.80.2. Synopsis

Cleans an entire cache block globally throughout the system.

Exactly what happens is coherence protocol-dependent, but in general it is expected that after this operation():

- The cache block will be in the clean (not dirty) state in any coherent cache holding a valid copy of the line.
- The data will be cleaned to a point such that an incoherent load can observe the cleaed data.

cbo.clean is ordered by `FENCE` instructions but not `FENCE.I` or `SFENCE.VMA`.

<%- if CACHE_BLOCK_SIZE.bit_length > [PMP_GRANULARITY, PMA_GRANULARITY].min -%> Both PMP and PMA access control must be the same for all bytes in the block; otherwise, cbo.clean has UNSPECIFIED behavior. <%- end -%>

Clean operations are treated as stores for page and access permissions. If permission checks fail, one of the following exceptions will occur:

```
<%- if ext?(:H) -%>
* `Store/AMO Guest-Page Fault` if virtual memory translation fails during G-stage
translation.
<%- end -%>
* `Store/AMO Page Fault` if virtual memory translation fails <% if ext?(:H) %>when V=0
or during VS-stage translation<% end %>
* `Store/AMO Access Fault` if a PMP or PMA access check fails
```

<%- if CACHE_BLOCK_SIZE.bit_length ⇐ [PMP_GRANULARITY, PMA_GRANULARITY].min -%> Because cache blocks are naturally aligned and always fit in a single PMP or PMA regions, the PMP and PMA access checks only need to check a single address in the line. <%- end -%>

CBO operations never raise a misaligned address fault.

B.80.3. Access

M	HS	U	VS	VU
Always	Sometimes	Sometimes	Sometimes	Sometimes

Access is controled through menvcfg.CBZE, senvcfg.CBZE, and henvcfg.CBZE. When access is denied, the instruction either raises an `Illegal Instruction` or `Virtual Instruction` exception according to the table below.

menvcfg.CBCFE	**senvcfg.CBCFE**	**henvcfg.CBCFE**	**cbo.clean Instruction Behavior**			
			S-mode	**U-mode**	**VS-mode**	**VU-mode**
0	-	-	`Illegal Instruction`	`Illegal Instruction`	`Virtual Instruction`	`Virtual Instruction`
1	0	0	executes	`Illegal Instruction`	`Virtual Instruction`	`Virtual Instruction`
1	1	0	executes	executes	`Virtual Instruction`	`Virtual Instruction`
1	0	1	executes	`Illegal Instruction`	executes	`Virtual Instruction`
1	1	1	executes	executes	executes	executes

B.80.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
```

B.80.5. Execution

B.80.6. Exceptions

This instruction does not generate synchronous exceptions.

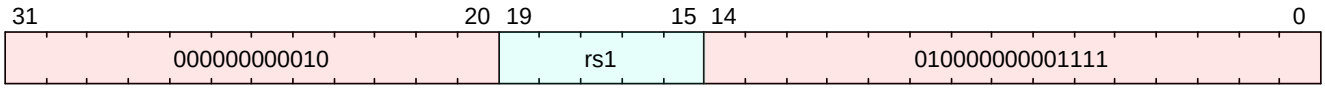
B.81. cbo.flush

Cache Block Flush

This instruction is defined by:

- Zicbom, version >= 0

B.81.1. Encoding



B.81.2. Synopsis

Flushes an entire cache block by cleaning it and then invalidating it in all caches.

cbo.flush is ordered by `FENCE` instructions but not `FENCE.I` or `SFENCE.VMA`.

<%- if CACHE_BLOCK_SIZE.bit_length > [PMP_GRANULARITY, PMA_GRANULARITY].min -%> Both PMP and PMA access control must be the same for all bytes in the block; otherwise, cbo.flush has UNSPECIFIED behavior. <%- end -%>

Flush operations are treated as stores for page and access permissions. If permission checks fail, one of the following exceptions will occur:

```
<%- if ext?(:H) -%>
* `Store/AMO Guest-Page Fault` if virtual memory translation fails during G-stage
translation.
<%- end -%>
* `Store/AMO Page Fault` if virtual memory translation fails <% if ext?(:H) %>when V=0
or during VS-stage translation<% end %>
* `Store/AMO Access Fault` if a PMP or PMA access check fails.
```

<%- if CACHE_BLOCK_SIZE.bit_length ⇐ [PMP_GRANULARITY, PMA_GRANULARITY].min -%> Because cache blocks are naturally aligned and always fit in a single PMP or PMA regions, the PMP and PMA access checks only need to check a single address in the line. <%- end -%>

CBO operations never raise a misaligned address fault.

B.81.3. Access

M	HS	U	VS	VU
Always	Sometimes	Sometimes	Sometimes	Sometimes

Access is controled through menvcfg.CBZE, senvcfg.CBZE, and henvcfg.CBZE. When access is denied, the instruction either raises an `Illegal Instruction` or `Virtual Instruction` exception according to the table below.

menvcfg.CBCFE	senvcfg.CBCFE	henvcfg.CBCFE	cbo.flush Instruction Behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	Illegal Instruction	Illegal Instruction	Virtual Instruction	Virtual Instruction
1	0	0	executes	Illegal Instruction	Virtual Instruction	Virtual Instruction
1	1	0	executes	executes	Virtual Instruction	Virtual Instruction
1	0	1	executes	Illegal Instruction	executes	Virtual Instruction
1	1	1	executes	executes	executes	executes

B.81.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
```

B.81.5. Execution

B.81.6. Exceptions

This instruction does not generate synchronous exceptions.

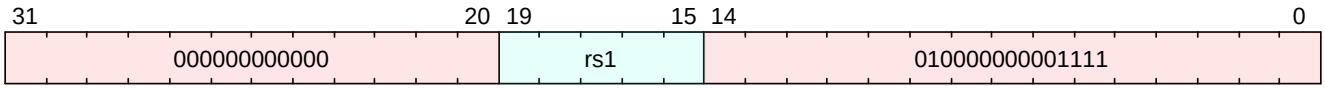
B.82. cbo.inval

Cache Block Invalidate

This instruction is defined by:

- Zicbom, version >= 0

B.82.1. Encoding



B.82.2. Synopsis

Either invalidates or flushes (clean + invalidate) a cache block, depending on the current mode and value of menvcfg.CBIE, senvcfg.CBIE, and/or henvcfg.CBIE.

The instruction is an invalidate (without a clean) when:

- In M-mode
- In (H)S-mode and menvcfg.CBIE == 11
- In U-mode and menvcfg.CBIE == 11 and senvcfg.CBIE == 11
- In VS-mode and menvcfg.CBIE == 11 and henvcfg.CBIE == 11
- In VU-mode and menvcfg.CBIE == 11 and henvcfg.CBIE == 11 and senvcfg.CBIE == 11

Otherwise, if the instruction does not trap (see Access section), the operation is a flush. The table below summarizes the options.

menvcfg.CBIE	**senvcfg.CBIE**	**henvcfg.CBIE**	`cbe.inval` **Operation**				
			M-mode	**S-mode**	**U-mode**	**VS-mode**	**VU-mode**
00	-	-	Invalidate	`Illegal Instruction`	`Illegal Instruction`	`Virtual Instruction`	`Virtual Instruction`
01	00	00	Invalidate	Flush	`Illegal Instruction`	`Virtual Instruction`	`Virtual Instruction`
01	00	01	Invalidate	Flush	`Illegal Instruction`	Flush	`Virtual Instruction`
01	00	11	Invalidate	Flush	`Illegal Instruction`	Flush	`Virtual Instruction`
01	01	00	Invalidate	Flush	Flush	`Virtual Instruction`	`Virtual Instruction`
01	01	01	Invalidate	Flush	Flush	Flush	Flush

01	01	11	Invalidate	Flush	Flush	Flush	Flush
01	11	00	Invalidate	Flush	Flush	Virtual Instruction	Virtual Instruction
01	11	01	Invalidate	Flush	Flush	Flush	Flush
01	11	11	Invalidate	Flush	Flush	Flush	Flush
11	00	00	Invalidate	Invalidate	Illegal Instruction	Virtual Instruction	Virtual Instruction
11	00	01	Invalidate	Invalidate	Illegal Instruction	Flush	Virtual Instruction
11	00	11	Invalidate	Invalidate	Illegal Instruction	Invalidate	Virtual Instruction
11	01	00	Invalidate	Invalidate	Flush	Virtual Instruction	Virtual Instruction
11	01	01	Invalidate	Invalidate	Flush	Flush	Flush
11	01	11	Invalidate	Invalidate	Flush	Invalidate	Flush
11	11	00	Invalidate	Invalidate	Invalidate	Virtual Instruction	Virtual Instruction
11	11	01	Invalidate	Invalidate	Invalidate	Flush	Flush
11	11	11	Invalidate	Invalidate	Invalidate	Invalidate	Invalidate

cbo.inval is ordered by FENCE instructions but not FENCE.I or SFENCE.VMA.

<%- if CACHE_BLOCK_SIZE.bit_length > [PMP_GRANULARITY, PMA_GRANULARITY].min -%> Both PMP and PMA access control must be the same for all bytes in the block; otherwise, cbo.zero has UNSPECIFIED behavior. <%- end -%>

Invalidate operations are treated as stores for page and access permissions. If permission checks fail, one of the following exceptions will occur:

```
<%- if ext?(:H) -%>
* `Store/AMO Guest-Page Fault` if virtual memory translation fails during G-stage
translation.
<%- end -%>
* `Store/AMO Page Fault` if virtual memory translation fails <% if ext?(:H) %>when V=0
or during VS-stage translation<% end %>
* `Store/AMO Access Fault` if a PMP or PMA access check fails.
```

<%- if CACHE_BLOCK_SIZE.bit_length ⇐ [PMP_GRANULARITY, PMA_GRANULARITY].min -%> Because cache blocks are naturally aligned and always fit in a single PMP or PMA regions, the PMP and PMA access checks only need to check a single address in the line. <%- end -%>

CBO operations never raise a misaligned address fault.

B.82.3. Access

M	HS	U	VS	VU
Always	Sometimes	Sometimes	Sometimes	Sometimes

Access is controled through menvcfg.CBIE, senvcfg.CBIE, and henvcfg.CBIE. When access is denied, the instruction either raises an `Illegal Instruction` or `Virtual Instruction` exception according to the table below.

`xenvcfg.CBIE` == 10 is reserved, and cannot be written by software. As such, that pattern is excluded from the table below.

menvcfg.CBIE	**senvcfg.CBIE**	**henvcfg.CBIE**	**cbo.inval Instruction Behavior**			
			S-mode	**U-mode**	**VS-mode**	**VU-mode**
00	-	-	`Illegal Instruction`	`Illegal Instruction`	`Virtual Instruction`	`Virtual Instruction`
01/11	00	00	executes	`Illegal Instruction`	`Virtual Instruction`	`Virtual Instruction`
01/11	01/11	00	executes	executes	`Virtual Instruction`	`Virtual Instruction`
01/11	00	01/11	executes	`Illegal Instruction`	executes	`Virtual Instruction`
01/11	01/11	01/11	executes	executes	executes	executes

B.82.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
```

B.82.5. Execution

B.82.6. Exceptions

This instruction does not generate synchronous exceptions.

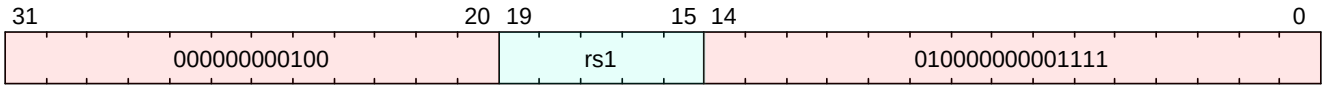
B.83. cbo.zero

Cache Block Zero

This instruction is defined by:

- Zicboz, version >= 0

B.83.1. Encoding



B.83.2. Synopsis

Zeros an entire cache block

The block zeroing does not need to be atomic.

cbo.zero is ordered by `FENCE` instructions but not `FENCE.I` or `SFENCE.VMA`.

<%- if CACHE_BLOCK_SIZE.bit_length > [PMP_GRANULARITY, PMA_GRANULARITY].min -%> Both PMP and PMA access control must be the same for all bytes in the block; otherwise, cbo.zero has UNSPECIFIED behavior. <%- end -%>

Clean operations are treated as stores for page and access permissions. If permission checks fail, one of the following exceptions will occur:

```
<%- if ext?(:H) -%>
* `Store/AMO Guest-Page Fault` if virtual memory translation fails during G-stage
translation.
<%- end -%>
* `Store/AMO Page Fault` if virtual memory translation fails <% if ext?(:H) %>when V=0
or during VS-stage translation<% end %>
* `Store/AMO Access Fault` if a PMP or PMA access check fails.
```

<%- if CACHE_BLOCK_SIZE.bit_length ⇐ [PMP_GRANULARITY, PMA_GRANULARITY].min -%> Because cache blocks are naturally aligned and always fit in a single PMP or PMA regions, the PMP and PMA access checks only need to check a single address in the line. <%- end -%>

CBO operations never raise a misaligned address fault.

B.83.3. Access

M	HS	U	VS	VU
Always	Sometimes	Sometimes	Sometimes	Sometimes

Access is controled through menvcfg.CBZE, senvcfg.CBZE, and henvcfg.CBZE. When access is denied,

the instruction either raises an Illegal Instruction or Virtual Instruction exception according to the table below.

menvcfg.CBZE	senvcfg.CBZE	henvcfg.CBZE	cbo.zero Instruction Behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	Illegal Instruction	Illegal Instruction	Virtual Instruction	Virtual Instruction
1	0	0	executes	Illegal Instruction	Virtual Instruction	Virtual Instruction
1	1	0	executes	executes	Virtual Instruction	Virtual Instruction
1	0	1	executes	Illegal Instruction	executes	Virtual Instruction
1	1	1	executes	executes	executes	executes

B.83.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
```

B.83.5. Execution

```
if ((mode() == PrivilegeMode::M && CSR[menvcfg].CBZE == 0) || (mode() ==
PrivilegeMode::U && CSR[senvcfg].CBZE == 0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
} else if ((mode() == PrivilegeMode::VS && CSR[henvcfg].CBZE == 0) || (mode() ==
PrivilegeMode::VU && (CSR[henvcfg].CBZE | CSR[senvcfg].CBZE) == 0)) {
  raise(ExceptionCode::VirtualInstruction, mode(), $encoding);
} else {
  XReg mask = CACHE_BLOCK_SIZE - 1;
  XReg cache_block_vaddr = X[rs1] & ~mask;
  TranslationResult result;
  result = translate(cache_block_vaddr, MemoryOperation::Write, effective_ldst_mode(),
$encoding);
  access_check(result.paddr, CACHE_BLOCK_SIZE * 8, cache_block_vaddr,
MemoryOperation::Write, ExceptionCode::StoreAmoAccessFault, effective_ldst_mode());
  cache_block_zero(result.paddr);
}
```

B.83.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault

- StoreAmoPageFault
- VirtualInstruction

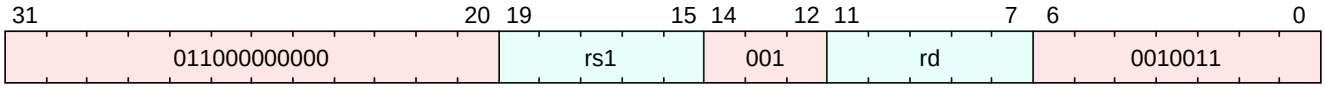
B.84. clz

Count leading zero bits

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0

B.84.1. Encoding



B.84.2. Synopsis

This instruction counts the number of 0's before the first 1, starting at the most-significant bit (i.e., XLEN-1) and progressing to bit 0. Accordingly, if the input is 0, the output is XLEN, and if the most-significant bit of the input is a 1, the output is 0.

B.84.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.84.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.84.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = (xlen() - 1) - $signed(highest_set_bit(X[rs1]));
```

B.84.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

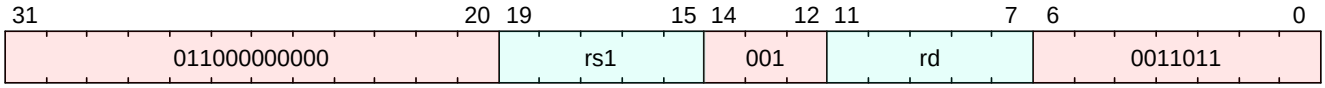
B.85. clzw

Count leading zero bits in word

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0

B.85.1. Encoding



B.85.2. Synopsis

This instruction counts the number of 0's before the first 1 starting at bit 31 and progressing to bit 0. Accordingly, if the least-significant word is 0, the output is 32, and if the most-significant bit of the word (*i.e.*, bit 31) is a 1, the output is 0.

B.85.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.85.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.85.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = 31 - $signed(highest_set_bit(X[rs1][31:0]));
```

B.85.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

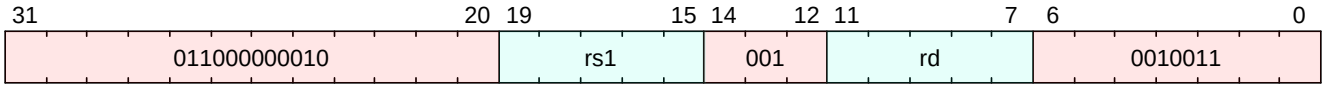
B.86. cpop

Count set bits

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0

B.86.1. Encoding



B.86.2. Synopsis

This instructions counts the number of 1's (i.e., set bits) in the source register.

Listing 1. Software Hint

```
This operations is known as population count, popcount, sideways sum,
bit summation, or Hamming weight.

The GCC builtin function `__builtin_popcount (unsigned int x)` is
implemented by cpop on RV32 and by cpopw on RV64. The GCC builtin
function `__builtin_popcountl (unsigned long x)` for LP64 is
implemented by cpop on RV64.
```

B.86.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.86.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.86.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg bitcount = 0;
```

```
XReg rs1_val = X[rs1];
for (U32 i = 0; i < xlen(); i++) {
  if (rs1_val[i] == 1'b1) {
    bitcount = bitcount + 1;
  }
}
X[rd] = bitcount;
```

B.86.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

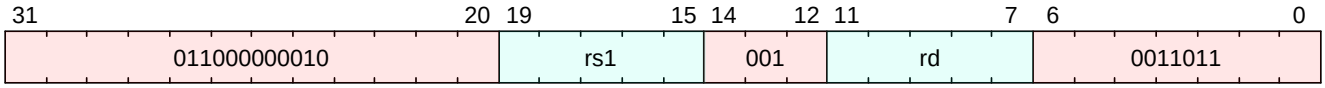
B.87. cpopw

Count set bits in word

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0

B.87.1. Encoding



B.87.2. Synopsis

This instructions counts the number of 1's (i.e., set bits) in the least-significant word of the source egister.

Listing 2. Software Hint

```
This operations is known as population count, popcount, sideways sum,
bit summation, or Hamming weight.

The GCC builtin function `__builtin_popcount (unsigned int x)` is
implemented by cpop on RV32 and by cpopw on RV64. The GCC builtin
function `__builtin_popcountl (unsigned long x)` for LP64 is
implemented by cpop on RV64.
```

B.87.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.87.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.87.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
```

```
XReg bitcount = 0;
XReg rs1_val = X[rs1];
for (U32 i = 0; i < 32; i++) {
  if (rs1_val[i] == 1'b1) {
    bitcount = bitcount + 1;
  }
}
X[rd] = bitcount;
```

B.87.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

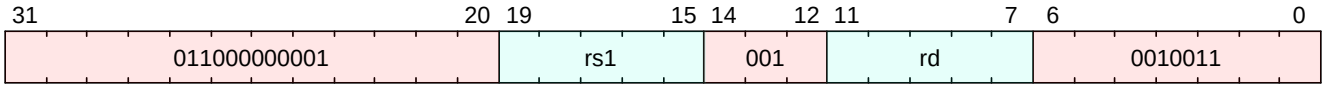
B.88. ctz

Count trailing zero bits

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0

B.88.1. Encoding



B.88.2. Synopsis

This instruction counts the number of 0's before the first 1, starting at the least-significant bit (i.e., 0) and progressing to the most-significant bit (i.e., XLEN-1). Accordingly, if the input is 0, the output is XLEN, and if the least-significant bit of the input is a 1, the output is 0.

B.88.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.88.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.88.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = (xlen() - 1) - $signed(lowest_set_bit(X[rs1]));
```

B.88.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

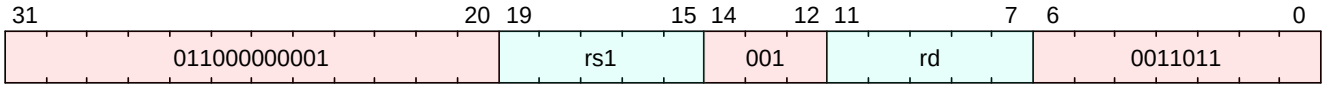
B.89. ctzw

Count trailing zero bits in word

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0

B.89.1. Encoding



B.89.2. Synopsis

This instruction counts the number of 0's before the first 1, starting at the least-significant bit (i.e., 0) and progressing to the most-significant bit of the least-significant word (i.e., 31). Accordingly, if the least-significant word is 0, the output is 32, and if the least-significant bit of the input is a 1, the output is 0.

B.89.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.89.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.89.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = (xlen() - 1) - $signed(lowest_set_bit(X[rs1][31:0]));
```

B.89.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

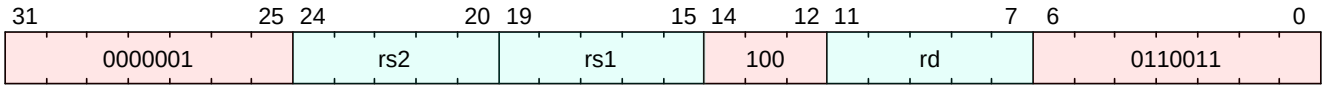
B.90. div

Signed division

This instruction is defined by:

- M, version >= 0

B.90.1. Encoding



B.90.2. Synopsis

Divide rs1 by rs2, and store the result in rd. The remainder is discarded.

Division by zero will put -1 into rd.

Division resulting in signed overflow (when most negative number is divided by -1) will put the most negative number into rd;

B.90.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.90.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.90.5. Execution

```
if (implemented?(ExtensionName::M) && (CSR[misa].M == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg src1 = X[rs1];
XReg src2 = X[rs2];
if (src2 == 0) {
  X[rd] = {XLEN{1'b1}};
} else if ((src1 == {1'b1, {XLEN - 1{1'b0}}}) && (src2 == {XLEN{1'b1}})) {
  X[rd] = {1'b1, {XLEN - 1{1'b0}}};
} else {
  X[rd] = $signed(src1) / $signed(src2);
```

```
}
```

B.90.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

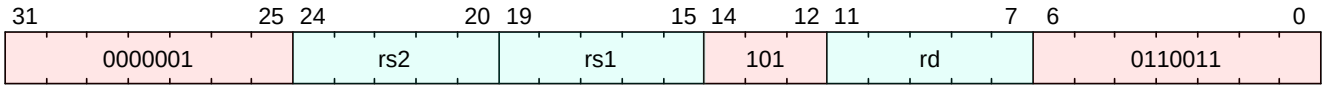
B.91. divu

Unsigned division

This instruction is defined by:

- M, version >= 0

B.91.1. Encoding



B.91.2. Synopsis

Divide unsigned values in rs1 by rs2, and store the result in rd.

The remainder is discarded.

If the value in rs2 is zero, rd gets the largest unsigned value.

B.91.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.91.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.91.5. Execution

```
if (implemented?(ExtensionName::M) && (CSR[misa].M == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg src1 = X[rs1];
XReg src2 = X[rs2];
if (src2 == 0) {
  X[rd] = {XLEN{1'b1}};
} else {
  X[rd] = src1 / src2;
}
```

B.91.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

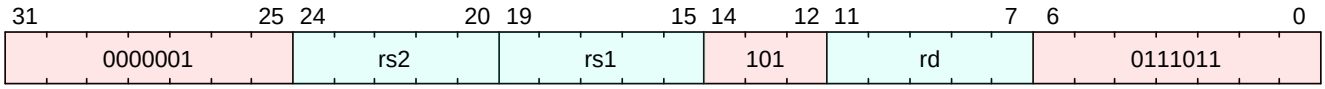
B.92. divuw

Unsigned 32-bit division

This instruction is defined by:

- M, version >= 0

B.92.1. Encoding



B.92.2. Synopsis

Divide the unsigned 32-bit values in rs1 and rs2, and store the sign-extended result in rd.

The remainder is discarded.

If the value in rs2 is zero, rd is written with all 1s.

B.92.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.92.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.92.5. Execution

```
if (implemented?(ExtensionName::M) && (CSR[misa].M == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
Bits<32> src1 = X[rs1][31:0];
Bits<32> src2 = X[rs2][31:0];
if (src2 == 0) {
  X[rd] = {64{1'b1}};
} else {
  Bits<32> result = src1 / src2;
  Bits<1> sign_bit = result[31];
  X[rd] = {{32{sign_bit}}, result};
}
```

B.92.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

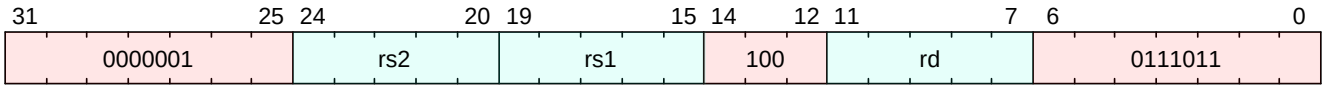
B.93. divw

Signed 32-bit division

This instruction is defined by:

- M, version >= 0

B.93.1. Encoding



B.93.2. Synopsis

Divide the lower 32-bits of register rs1 by the lower 32-bits of register rs2, and store the sign-extended result in rd.

The remainder is discarded.

Division by zero will put -1 into rd.

Division resulting in signed overflow (when most negative number is divided by -1) will put the most negative number into rd;

B.93.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.93.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.93.5. Execution

```
if (implemented?(ExtensionName::M) && (CSR[misa].M == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
Bits<32> src1 = X[rs1][31:0];
Bits<32> src2 = X[rs2][31:0];
if (src2 == 0) {
  X[rd] = {XLEN{1'b1}};
} else if ((src1 == {33'b1, 31'b0}) && (src2 == 32'b1)) {
  X[rd] = {33'b1, 31'b0};
```

```
} else {
  Bits<32> result = $signed(src1) / $signed(src2);
  Bits<1> sign_bit = result[31];
  X[rd] = {{32{sign_bit}}, result};
}
```

B.93.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

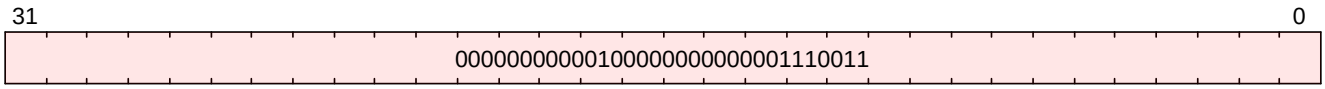
B.94. ebreak

Breakpoint exception

This instruction is defined by:

- I, version >= 0

B.94.1. Encoding



B.94.2. Synopsis

The EBREAK instruction is used by debuggers to cause control to be transferred back to a debugging environment. Unless overridden by an external debug environment, EBREAK raises a breakpoint exception and performs no other operation.

As described in the C Standard Extension for Compressed Instructions, the c.ebreak instruction performs the same operation as the EBREAK instruction.

EBREAK causes the receiving privilege mode's epc register to be set to the address of the EBREAK instruction itself, not the address of the following instruction. As EBREAK causes a synchronous exception, it is not considered to retire, and should not increment the minstret CSR.

B.94.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.94.4. Decode Variables

```
```

B.94.5. Execution

```
if (TRAP_ON_EBREAK) {
  raise_precise(ExceptionCode::Breakpoint, mode(), $pc);
} else {
  eei_ebreak();
}
```

B.94.6. Exceptions

This instruction may result in the following synchronous exceptions:

- Breakpoint

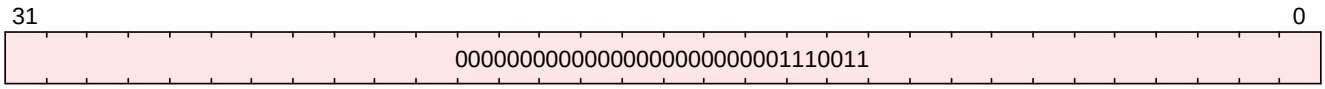
B.95. ecall

Environment call

This instruction is defined by:

- I, version >= 0

B.95.1. Encoding



B.95.2. Synopsis

The ECALL instruction is used to make a request to the supporting execution environment. When executed in U-mode, S-mode, or M-mode, it generates an environment-call-from-U-mode exception, environment-call-from-S-mode exception, or environment-call-from-M-mode exception, respectively, and performs no other operation.

ECALL generates a different exception for each originating privilege mode so that environment call exceptions can be selectively delegated. A typical use case for Unix-like operating systems is to delegate to S-mode the environment-call-from-U-mode exception but not the others.

ECALL causes the receiving privilege mode's epc register to be set to the address of the ECALL instruction itself, not the address of the following instruction. As ECALL causes a synchronous exception, it is not considered to retire, and should not increment the minstret CSR.

B.95.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.95.4. Decode Variables

```
```

B.95.5. Execution

```
if (mode() == PrivilegeMode::M) {
  if (TRAP_ON_ECALL_FROM_M) {
    raise_precise(ExceptionCode::Mcall, PrivilegeMode::M, 0);
  } else {
    eei_ecall_from_m();
  }
} else if (mode() == PrivilegeMode::S) {
```

```
  if (TRAP_ON_ECALL_FROM_S) {
    raise_precise(ExceptionCode::Scall, PrivilegeMode::S, 0);
  } else {
    eei_ecall_from_s();
  }
} else if (mode() == PrivilegeMode::U || mode() == PrivilegeMode::VU) {
  if (TRAP_ON_ECALL_FROM_U) {
    raise_precise(ExceptionCode::Ucall, mode(), 0);
  } else {
    eei_ecall_from_u();
  }
} else if (mode() == PrivilegeMode::VS) {
  if (TRAP_ON_ECALL_FROM_VS) {
    raise_precise(ExceptionCode::VScall, PrivilegeMode::VS, 0);
  } else {
    eei_ecall_from_vs();
  }
}
```

B.95.6. Exceptions

This instruction may result in the following synchronous exceptions:

- Mcall
- Scall
- Ucall
- VScall

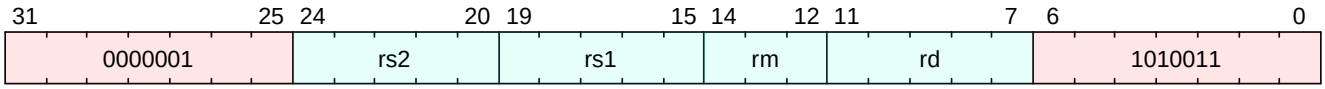
B.96. fadd.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.96.1. Encoding



B.96.2. Synopsis

No description available.

B.96.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.96.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.96.5. Execution

```

```

B.96.6. Exceptions

This instruction does not generate synchronous exceptions.

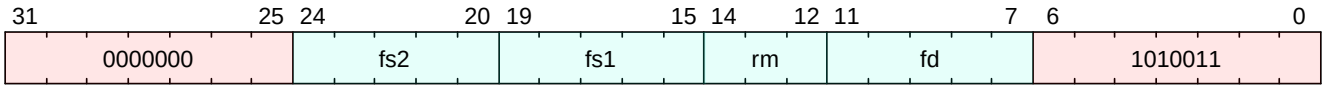
B.97. fadd.s

No synopsis available.

This instruction is defined by:

- F, version >= 0

B.97.1. Encoding



B.97.2. Synopsis

No description available.

B.97.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.97.4. Decode Variables

```
Bits<5> fs2 = $encoding[24:20];
Bits<5> fs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> fd = $encoding[11:7];
```

B.97.5. Execution

```

```

B.97.6. Exceptions

This instruction does not generate synchronous exceptions.

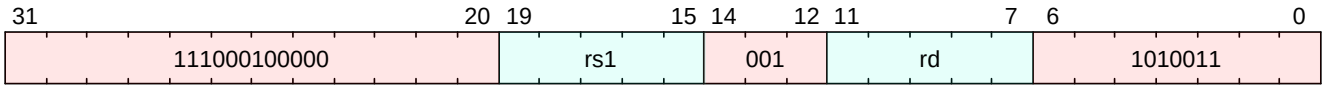
B.98. fclass.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.98.1. Encoding



B.98.2. Synopsis

No description available.

B.98.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.98.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.98.5. Execution

```

```

B.98.6. Exceptions

This instruction does not generate synchronous exceptions.

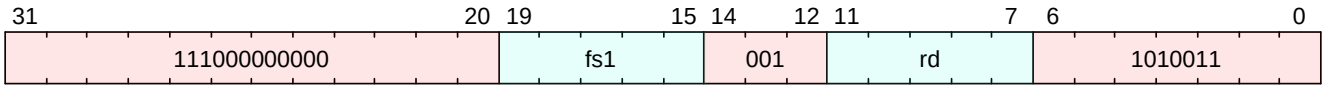
B.99. fclass.s

Single-precision floating-point classify.

This instruction is defined by:

- F, version >= 0

B.99.1. Encoding



B.99.2. Synopsis

The fclass.s instruction examines the value in floating-point register *fs1* and writes to integer register *rd* a 10-bit mask that indicates the class of the floating-point number. The format of the mask is described in the table below. The corresponding bit in *rd* will be set if the property is true and clear otherwise. All other bits in *rd* are cleared. Note that exactly one bit in rd will be set. fclass.s does not set the floating-point exception flags.

Table 51. Format of result of `fclass` instruction.

rd bit	Meaning
0	*rs1* is `-\infty`.
1	*rs1* is a negative normal number.
2	*rs1* is a negative subnormal number.
3	*rs1* is `-0`.
4	*rs1* is `+0`.
5	*rs1* is a positive subnormal number.
6	*rs1* is a positive normal number.
7	*rs1* is `+\infty`.
8	*rs1* is a signaling NaN.
9	*rs1* is a quiet NaN.

B.99.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.99.4. Decode Variables

```
Bits<5> fs1 = $encoding[19:15];
```

```
Bits<5> rd = $encoding[11:7];
```

B.99.5. Execution

```
check_f_ok($encoding);
Bits<32> sp_value = f[fs1][31:0];
if (is_sp_neg_inf?(sp_value)) {
  X[rd] = 1 << 0;
} else if (is_sp_neg_norm?(sp_value)) {
  X[rd] = 1 << 1;
} else if (is_sp_neg_subnorm?(sp_value)) {
  X[rd] = 1 << 2;
} else if (is_sp_neg_zero?(sp_value)) {
  X[rd] = 1 << 3;
} else if (is_sp_pos_zero?(sp_value)) {
  X[rd] = 1 << 4;
} else if (is_sp_pos_subnorm?(sp_value)) {
  X[rd] = 1 << 5;
} else if (is_sp_pos_norm?(sp_value)) {
  X[rd] = 1 << 6;
} else if (is_sp_pos_inf?(sp_value)) {
  X[rd] = 1 << 7;
} else if (is_sp_signaling_nan?(sp_value)) {
  X[rd] = 1 << 8;
} else {
  assert(is_sp_quiet_nan?(sp_value), "Unexpected SP value");
  X[rd] = 1 << 9;
}
```

B.99.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

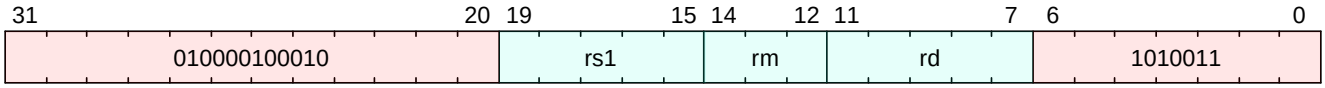
B.100. fcvt.d.h

No synopsis available.

This instruction is defined by:

- anyOf:
  - D, version >= 0
  - Zfh, version >= 0

B.100.1. Encoding



B.100.2. Synopsis

No description available.

B.100.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.100.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.100.5. Execution

```

```

B.100.6. Exceptions

This instruction does not generate synchronous exceptions.

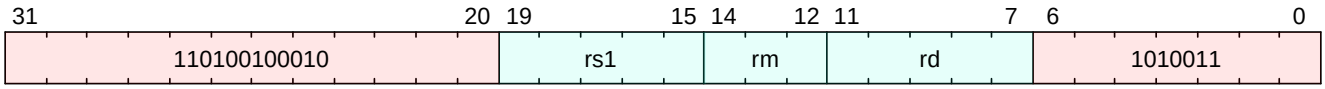
B.101. fcvt.d.l

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.101.1. Encoding



B.101.2. Synopsis

No description available.

B.101.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.101.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.101.5. Execution

```

```

B.101.6. Exceptions

This instruction does not generate synchronous exceptions.

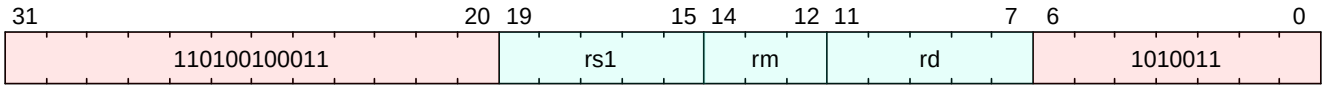
B.102. fcvt.d.lu

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.102.1. Encoding



B.102.2. Synopsis

No description available.

B.102.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.102.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.102.5. Execution

```

```

B.102.6. Exceptions

This instruction does not generate synchronous exceptions.

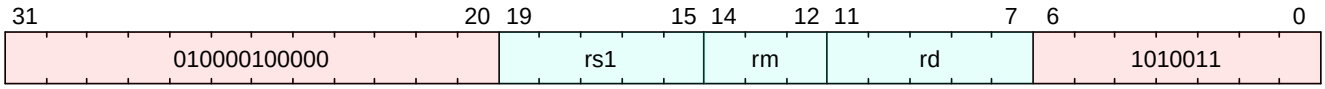
B.103. fcvt.d.s

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.103.1. Encoding



B.103.2. Synopsis

No description available.

B.103.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.103.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.103.5. Execution

```

```

B.103.6. Exceptions

This instruction does not generate synchronous exceptions.

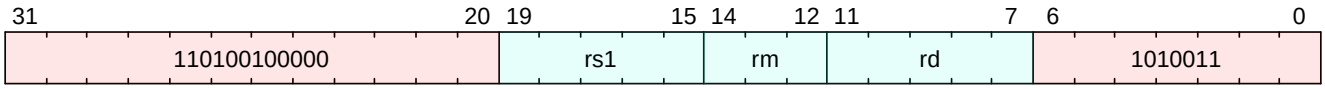
B.104. fcvt.d.w

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.104.1. Encoding



B.104.2. Synopsis

No description available.

B.104.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.104.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.104.5. Execution

```

```

B.104.6. Exceptions

This instruction does not generate synchronous exceptions.

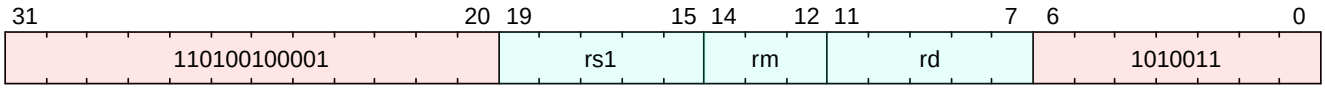
B.105. fcvt.d.wu

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.105.1. Encoding



B.105.2. Synopsis

No description available.

B.105.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.105.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.105.5. Execution

```

```

B.105.6. Exceptions

This instruction does not generate synchronous exceptions.

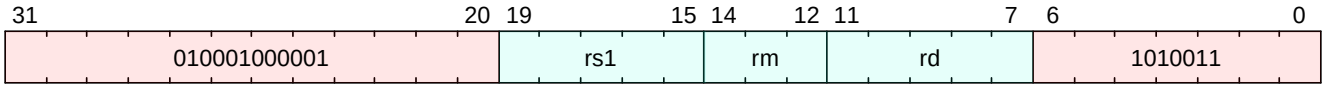
B.106. fcvt.h.d

No synopsis available.

This instruction is defined by:

- anyOf:
  - D, version >= 0
  - Zfh, version >= 0

B.106.1. Encoding



B.106.2. Synopsis

No description available.

B.106.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.106.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.106.5. Execution

```

```

B.106.6. Exceptions

This instruction does not generate synchronous exceptions.

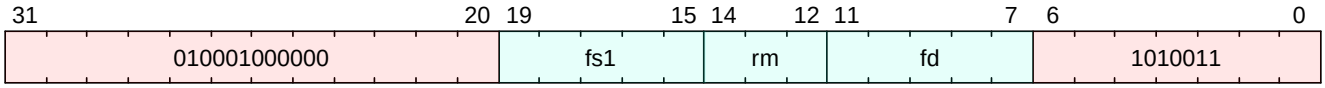
B.107. fcvt.h.s

Convert half-precision float to a single-precision float

This instruction is defined by:

- anyOf:
  - Zfh, version >= 0
  - Zfhmin, version >= 0

B.107.1. Encoding



B.107.2. Synopsis

Converts a half-precision number in floating-point register *fs1* into a single-precision floating-point number in floating-point register *fd*.

fcvt.h.s rounds according to the *rm* field.

All floating-point conversion instructions set the Inexact exception flag if the rounded result differs from the operand value and the Invalid exception flag is not set.

B.107.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.107.4. Decode Variables

```
Bits<5> fs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> fd = $encoding[11:7];
```

B.107.5. Execution

```
check_f_ok($encoding);
Bits<16> hp_value = f[fs1][15:0];
Bits<1> sign = hp_value[15];
Bits<5> exp = hp_value[14:10];
Bits<10> frac = hp_value[9:0];
if (exp == 0x1F) {
  if (frac != 0) {
    if ((hp_value & 0x0200) != 0) {
```

```
      set_fp_flag(FpFlag::NV);
    }
    f[fd] = HP_CANONICAL_NAN;
  } else {
    f[fd] = packToF32UI(sign, 0xFF, 0);
  }
} else {
  if (exp != 0) {
    if (frac != 0) {
      f[fd] = packToF32UI(sign, 0, 0);
    } else {
      Bits<6> norm_exp;
      (norm_exp, frac = softfloat_normSubnormalF16Sig(frac));
      exp = norm_exp - 1;
      f[fd] = packToF32UI(sign, exp + 0x70, frac << 13);
    }
  } else {
    f[fd] = packToF32UI(sign, exp + 0x70, frac << 13);
  }
}
mark_f_state_dirty();
```

B.107.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

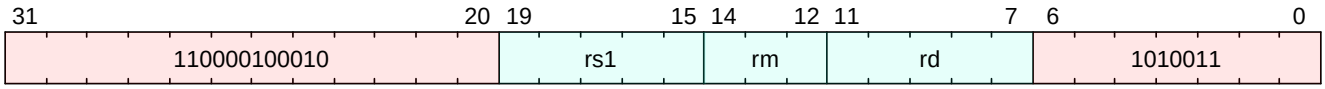
B.108. fcvt.l.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.108.1. Encoding



B.108.2. Synopsis

No description available.

B.108.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.108.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.108.5. Execution

```

```

B.108.6. Exceptions

This instruction does not generate synchronous exceptions.

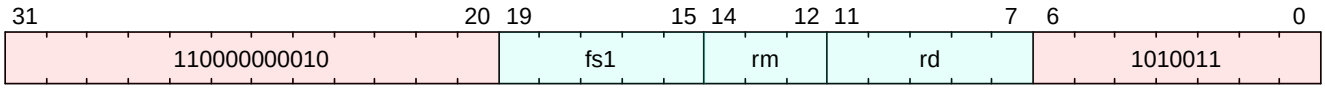
B.109. fcvt.l.s

No synopsis available.

This instruction is defined by:

- F, version >= 0

B.109.1. Encoding



B.109.2. Synopsis

No description available.

B.109.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.109.4. Decode Variables

```
Bits<5> fs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.109.5. Execution

```

```

B.109.6. Exceptions

This instruction does not generate synchronous exceptions.

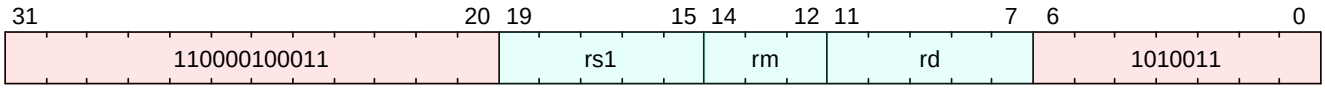
B.110. fcvt.lu.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.110.1. Encoding



B.110.2. Synopsis

No description available.

B.110.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.110.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.110.5. Execution

```

```

B.110.6. Exceptions

This instruction does not generate synchronous exceptions.

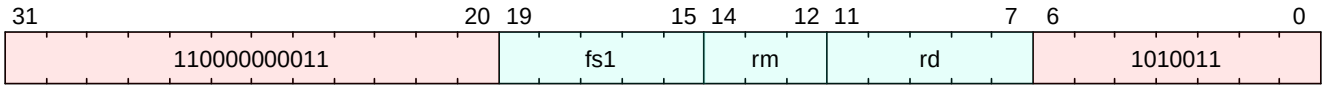
B.111. fcvt.lu.s

No synopsis available.

This instruction is defined by:

- F, version >= 0

B.111.1. Encoding



B.111.2. Synopsis

No description available.

B.111.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.111.4. Decode Variables

```
Bits<5> fs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.111.5. Execution

```

```

B.111.6. Exceptions

This instruction does not generate synchronous exceptions.

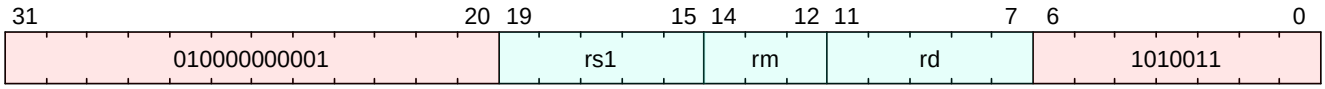
B.112. fcvt.s.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.112.1. Encoding



B.112.2. Synopsis

No description available.

B.112.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.112.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.112.5. Execution

```

```

B.112.6. Exceptions

This instruction does not generate synchronous exceptions.

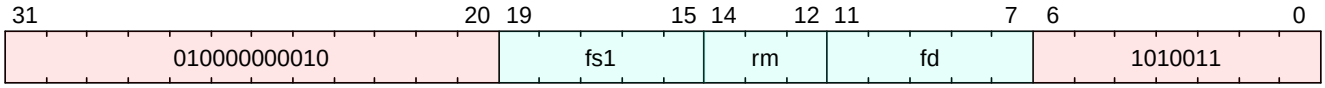
B.113. fcvt.s.h

Convert single-precision float to a half-precision float

This instruction is defined by:

- anyOf:
  - Zfh, version >= 0
  - Zfhmin, version >= 0

B.113.1. Encoding



B.113.2. Synopsis

Converts a single-precision number in floating-point register *fs1* into a half-precision floating-point number in floating-point register *fd*.

fcvt.s.h will never round, and so the 'rm' field is effectively ignored.

B.113.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.113.4. Decode Variables

```
Bits<5> fs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> fd = $encoding[11:7];
```

B.113.5. Execution

```
check_f_ok($encoding);
Bits<32> sp_value = f[fs1][31:0];
Bits<1> sign = sp_value[31];
Bits<8> exp = sp_value[30:23];
Bits<23> frac = sp_value[22:0];
if (exp == 0xFF) {
  if (frac != 0) {
    if ((sp_value & 0x00400000) != 0) {
      set_fp_flag(FpFlag::NV);
    }
    f[fd] = nan_box<16, FLEN>(HP_CANONICAL_NAN);
```

```
  } else {
    f[fd] = nan_box<16, FLEN>(packToF16UI(sign, 0x1F, 0));
  }
} else {
  Bits<16> frac16 = (frac >> 9) | frac & 0x1ff) != 0 ? 1 : 0);   if ((exp | frac16) ==
0) {     f[fd] = nan_box<16, FLEN>(packToF16UI(sign, 0, 0;
  } else {
    assert(false, "TODO: implement roundPackToF16");
  }
}
mark_f_state_dirty();
```

B.113.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

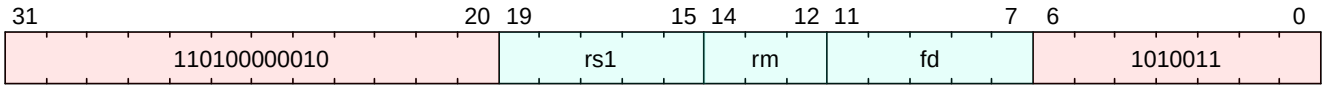
B.114. fcvt.s.l

No synopsis available.

This instruction is defined by:

- F, version >= 0

B.114.1. Encoding



B.114.2. Synopsis

No description available.

B.114.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.114.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> fd = $encoding[11:7];
```

B.114.5. Execution

```

```

B.114.6. Exceptions

This instruction does not generate synchronous exceptions.

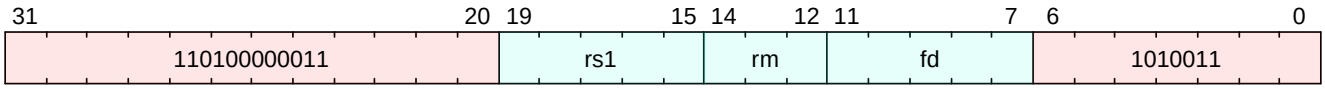
B.115. fcvt.s.lu

No synopsis available.

This instruction is defined by:

- F, version >= 0

B.115.1. Encoding



B.115.2. Synopsis

No description available.

B.115.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.115.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> fd = $encoding[11:7];
```

B.115.5. Execution

```

```

B.115.6. Exceptions

This instruction does not generate synchronous exceptions.

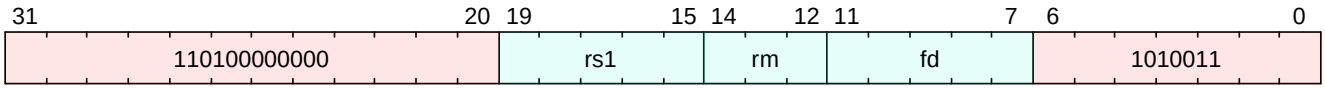
B.116. fcvt.s.w

Convert signed 32-bit integer to single-precision float

This instruction is defined by:

- F, version >= 0

B.116.1. Encoding



B.116.2. Synopsis

Converts a 32-bit signed integer in integer register *rs1* into a floating-point number in floating-point register *fd*.

All floating-point to integer and integer to floating-point conversion instructions round according to the *rm* field. A floating-point register can be initialized to floating-point positive zero using `fcvt.s.w rd, x0`, which will never set any exception flags.

All floating-point conversion instructions set the Inexact exception flag if the rounded result differs from the operand value and the Invalid exception flag is not set.

B.116.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.116.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> fd = $encoding[11:7];
```

B.116.5. Execution

```
check_f_ok($encoding);
Bits<32> int_value = X[rs1];
Bits<1> sign = int_value[31];
RoundingMode rounding_mode = rm_to_mode(rm, $encoding);
if ((int_value & 32'h7fff_ffff) == 0) {
  X[fd] = (sign == 1) ? packToF32UI(1, 0x9E, 0) : 0;
} else {
  Bits<32> absA = (sign == 1) ? -int_value : int_value;
  X[fd] = softfloat_normRoundPackToF32(sign, 0x9C, absA, rounding_mode);
```

```
}
mark_f_state_dirty();
```

B.116.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

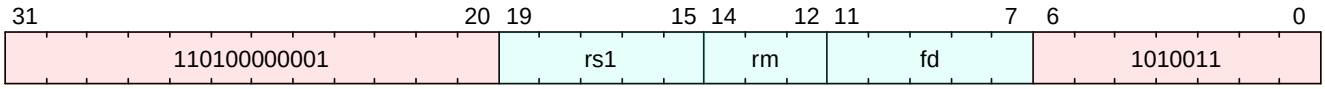
B.117. fcvt.s.wu

No synopsis available.

This instruction is defined by:

- F, version >= 0

B.117.1. Encoding



B.117.2. Synopsis

No description available.

B.117.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.117.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> fd = $encoding[11:7];
```

B.117.5. Execution

```

```

B.117.6. Exceptions

This instruction does not generate synchronous exceptions.

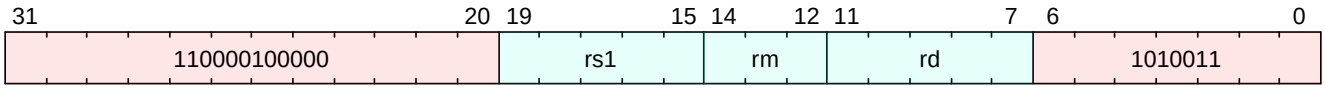
B.118. fcvt.w.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.118.1. Encoding



B.118.2. Synopsis

No description available.

B.118.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.118.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.118.5. Execution

```

```

B.118.6. Exceptions

This instruction does not generate synchronous exceptions.

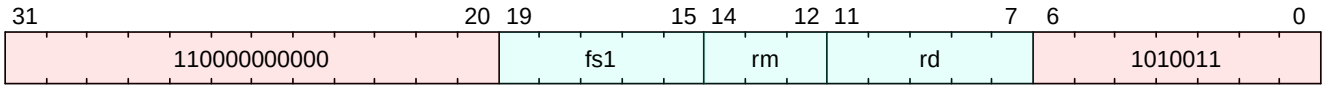
B.119. fcvt.w.s

Convert single-precision float to integer word to signed 32-bit integer.

This instruction is defined by:

- F, version >= 0

B.119.1. Encoding



B.119.2. Synopsis

Converts a floating-point number in floating-point register *fs1* to a signed 32-bit integer indicates integer register *rd*.

For XLEN >32, fcvt.w.s sign-extends the 32-bit result to the destination register width.

If the rounded result is not representable as a 32-bit signed integer, it is clipped to the nearest value and the invalid flag is set.

The range of valid inputs and behavior for invalid inputs are:

	Value
Minimum valid input (after rounding)	`-2^31`
Maximum valid input (after rounding)	`2^31 - 1`
Output for out-of-range negative input	`-2^31`
Output for <code>-∞</code>	`-2^31`
Output for out-of-range positive input	`2^31 - 1`
Output for <code>+∞</code> for <code>NaN</code>	`2^31 - 1`

All floating-point to integer and integer to floating-point conversion instructions round according to the *rm* field. A floating-point register can be initialized to floating-point positive zero using `fcvt.s.w rd, x0`, which will never set any exception flags.

All floating-point conversion instructions set the Inexact exception flag if the rounded result differs from the operand value and the Invalid exception flag is not set.

B.119.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.119.4. Decode Variables

```
Bits<5> fs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.119.5. Execution

```
check_f_ok($encoding);
Bits<32> sp_value = f[fs1][31:0];
Bits<1> sign = sp_value[31];
Bits<8> exp = sp_value[30:23];
Bits<23> sig = sp_value[22:0];
RoundingMode rounding_mode = rm_to_mode(rm, $encoding);
if ((exp == 0xff) && (sig != 0)) {
  sign = 0;
  set_fp_flag(FpFlag::NV);
  X[rd] = SP_CANONICAL_NAN;
} else {
  if (exp != 0) {
    sig = sig | 0x00800000;
  }
  Bits<64> sig64 = sig << 32;
  Bits<16> shift_dist = 0xAA - exp;
  if (0 < shift_dist) {
    sig64 = softfloat_shiftRightJam64(sig64, shift_dist);
  }
  X[rd] = softfloat_roundToI32(sign, sig64, rounding_mode);
}
```

B.119.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

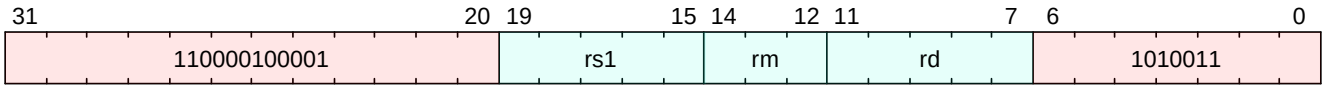
B.120. fcvt.wu.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.120.1. Encoding



B.120.2. Synopsis

No description available.

B.120.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.120.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.120.5. Execution

```

```

B.120.6. Exceptions

This instruction does not generate synchronous exceptions.

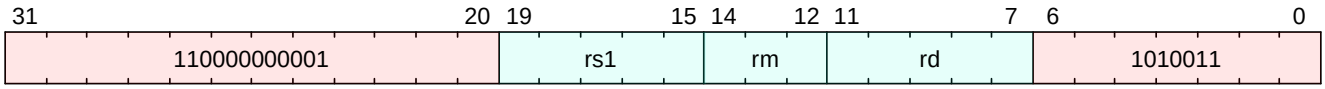
B.121. fcvt.wu.s

No synopsis available.

This instruction is defined by:

- F, version >= 0

B.121.1. Encoding



B.121.2. Synopsis

No description available.

B.121.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.121.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.121.5. Execution

```

```

B.121.6. Exceptions

This instruction does not generate synchronous exceptions.

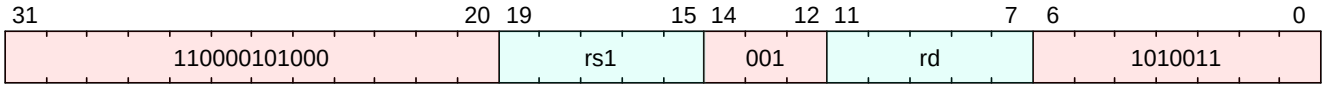
B.122. fcvtmod.w.d

No synopsis available.

This instruction is defined by:

- anyOf:
  - D, version >= 0
  - Zfa, version >= 0

B.122.1. Encoding



B.122.2. Synopsis

No description available.

B.122.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.122.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.122.5. Execution

```

```

B.122.6. Exceptions

This instruction does not generate synchronous exceptions.

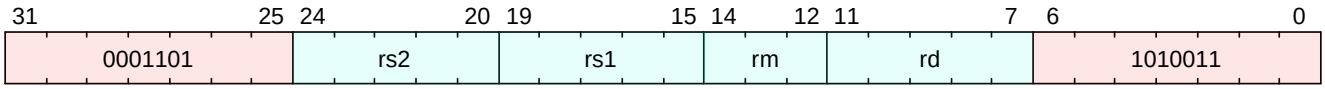
B.123. fdiv.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.123.1. Encoding



B.123.2. Synopsis

No description available.

B.123.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.123.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.123.5. Execution

```

```

B.123.6. Exceptions

This instruction does not generate synchronous exceptions.

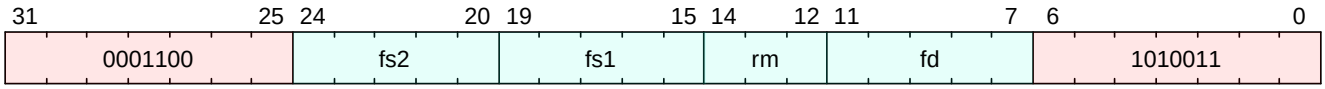
B.124. fdiv.s

No synopsis available.

This instruction is defined by:

- F, version >= 0

B.124.1. Encoding



B.124.2. Synopsis

No description available.

B.124.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.124.4. Decode Variables

```
Bits<5> fs2 = $encoding[24:20];
Bits<5> fs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> fd = $encoding[11:7];
```

B.124.5. Execution

```

```

B.124.6. Exceptions

This instruction does not generate synchronous exceptions.

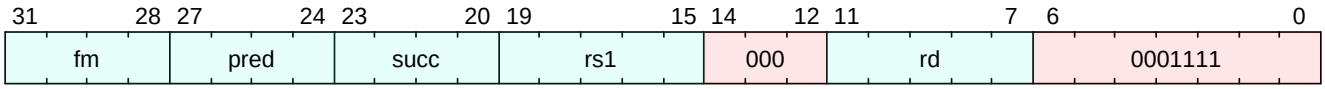
B.125. fence

Memory ordering fence

This instruction is defined by:

- I, version >= 0

B.125.1. Encoding



B.125.2. Synopsis

Orders memory operations.

The fence instruction is used to order device I/O and memory accesses as viewed by other RISC-V harts and external devices or coprocessors. Any combination of device input (I), device output (O), memory reads (R), and memory writes (W) may be ordered with respect to any combination of the same. Informally, no other RISC-V hart or external device can observe any operation in the *successor* set following a fence before any operation in the *predecessor* set preceding the fence.

The predecessor and successor fields have the same format to specify operation types:

`pred`				`succ`			
27	26	25	24	23	22	21	20
PI	PO	PR	PW	SI	SO	SR	SW

Table 52. Fence mode encoding

fm field	Mnemonic	Meaning
0000	*none*	Normal Fence
1000	TSO	With `FENCE RW,RW`: exclude write-to-read ordering; otherwise: *Reserved for future use.*
other		*Reserved for future use.*

When the mode field *fm* is `0001` and both the predecessor and successor sets are 'RW', then the instruction acts as a special-case `fence.tso`. `fence.tso` orders all load operations in its predecessor set before all memory operations in its successor set, and all store operations in its predecessor set before all store operations in its successor set. This leaves non-AMO store operations in the 'fence.tso's predecessor set unordered with non-AMO loads in its successor set.

When mode field *fm* is not `0001`, or when mode field *fm* is `0001` but the *pred* and *succ* fields are not both 'RW' (0x3), then the fence acts as a baseline fence (*e.g.*, *fm* is effectively `0000`). This is unaffected by the FIOM bits, described below (implicit promotion does not change how `fence.tso` is decoded).

The rs1 and rd fields are unused and ignored.

In modes other than M-mode, fence is further affected by menvcfg.FIOM, senvcfg.FIOM<% if ext?(:H) %>, and/or henvcfg.FIOM<% end %> as follows:

Table 53. Effective PR/PW/SR/SW in (H)S-mode

menvcfg.FIOM	pred.PI → **effective** PR pred.PO → **effective** PW succ.SI → **effective** SR succ.SO → **effective** SW		
0	-		from encoding
1	0		from encoding
1	1		1

Table 54. Effective PR/PW/SR/SW in U-mode

menvcfg.FIOM	**senvcfg.FIOM**	pred.PI → **effective** PR pred.PO → **effective** PW succ.SI → **effective** SR succ.SO → **effective** SW		
0	0	-		from encoding
0	1	0		from encoding
0	1	1		1
1	-	0		from encoding
1	-	1		1

<%- if ext?(:H) -%> .Effective PR/PW/SR/SW in VS-mode and VU-mode

menvcfg.FIOM	**henvcfg.FIOM**	pred.PI → **effective** PR pred.PO → **effective** PW succ.SI → **effective** SR succ.SO → **effective** SW		
0	0	-		from encoding
0	1	0		from encoding
0	1	1		1
1	-	0		from encoding
1	-	1		1

<%- end -%>

B.125.3. Access

M	HS	U	VS	VU

Always	Always	Always	Always	Always

B.125.4. Decode Variables

```
Bits<4> fm = $encoding[31:28];
Bits<4> pred = $encoding[27:24];
Bits<4> succ = $encoding[23:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.125.5. Execution

```
Boolean is_fence_tso;
Boolean is_pause;
if (fm == 1) {
  if (pred == 0x3 && succ == 0x3) {
    is_fence_tso = true;
  }
}
if (implemented?(ExtensionName::Zihintpause)) {
  if ((pred == 1) && (succ == 0) && (fm == 0) && (rd == 0) && (rs1 == 0)) {
    is_pause = true;
  }
}
Boolean pred_i = pred[3] == 1;
Boolean pred_o = pred[2] == 1;
Boolean pred_r = pred[1] == 1;
Boolean pred_w = pred[0] == 1;
Boolean succ_i = succ[3] == 1;
Boolean succ_o = succ[2] == 1;
Boolean succ_r = succ[1] == 1;
Boolean succ_w = succ[0] == 1;
if (is_fence_tso) {
  fence_tso();
} else if (is_pause) {
  pause();
} else {
  if (mode() == PrivilegeMode::S) {
    if (CSR[menvcfg].FIOM == 1) {
      if (pred_i) {
        pred_r = true;
      }
      if (pred_o) {
        pred_w = true;
      }
      if (succ_i) {
        succ_r = true;
      }
```

```
      if (succ_o) {
        succ_w = true;
      }
    }
  } else if (mode() == PrivilegeMode::U) {
    if ((CSR[menvcfg].FIOM | CSR[senvcfg].FIOM) == 1) {
      if (pred_i) {
        pred_r = true;
      }
      if (pred_o) {
        pred_w = true;
      }
      if (succ_i) {
        succ_r = true;
      }
      if (succ_o) {
        succ_w = true;
      }
    }
  } else if (mode() == PrivilegeMode::VS || mode() == PrivilegeMode::VU) {
    if ((CSR[menvcfg].FIOM | CSR[henvcfg].FIOM) == 1) {
      if (pred_i) {
        pred_r = true;
      }
      if (pred_o) {
        pred_w = true;
      }
      if (succ_i) {
        succ_r = true;
      }
      if (succ_o) {
        succ_w = true;
      }
    }
  }
  fence(pred_i, pred_o, pred_r, pred_w, succ_i, succ_o, succ_r, succ_w);
}
```

B.125.6. Exceptions

This instruction does not generate synchronous exceptions.

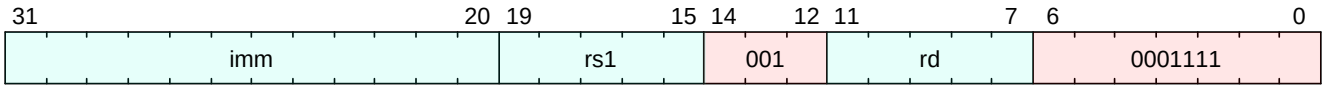
B.126. fence.i

Instruction fence

This instruction is defined by:

- Zifencei, version >= 0

B.126.1. Encoding



B.126.2. Synopsis

The FENCE.I instruction is used to synchronize the instruction and data streams. RISC-V does not guarantee that stores to instruction memory will be made visible to instruction fetches on a RISC-V hart until that hart executes a FENCE.I instruction. A FENCE.I instruction ensures that a subsequent instruction fetch on a RISC-V hart will see any previous data stores already visible to the same RISC-V hart. FENCE.I does *not* ensure that other RISC-V harts' instruction fetches will observe the local hart's stores in a multiprocessor system. To make a store to instruction memory visible to all RISC-V harts, the writing hart also has to execute a data FENCE before requesting that all remote RISC-V harts execute a FENCE.I.

The unused fields in the FENCE.I instruction, *imm[11:0]*, *rs1*, and *rd*, are reserved for finer-grain fences in future extensions. For forward compatibility, base implementations shall ignore these fields, and standard software shall zero these fields.

Because FENCE.I only orders stores with a hart's own instruction fetches, application code should only rely upon FENCE.I if the application thread will not be migrated to a different hart. The EEI can provide mechanisms for efficient multiprocessor instruction-stream synchronization.

B.126.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.126.4. Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.126.5. Execution

```
ifence();
```

B.126.6. Exceptions

This instruction does not generate synchronous exceptions.

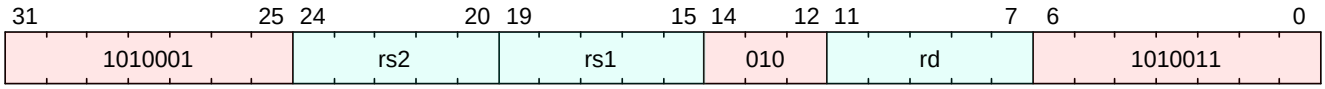
B.127. feq.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.127.1. Encoding



B.127.2. Synopsis

No description available.

B.127.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.127.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.127.5. Execution

```

```

B.127.6. Exceptions

This instruction does not generate synchronous exceptions.

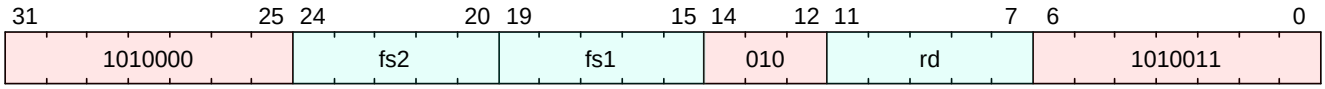
B.128. feq.s

Single-precision floating-point equal

This instruction is defined by:

- F, version >= 0

B.128.1. Encoding



B.128.2. Synopsis

Writes 1 to *rd* if *fs1* and *fs2* are equal, and 0 otherwise.

If either operand is NaN, the result is 0 (not equal). If either operand is a signaling NaN, the invalid flag is set.

Positive zero is considered equal to negative zero.

B.128.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.128.4. Decode Variables

```
Bits<5> fs2 = $encoding[24:20];
Bits<5> fs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.128.5. Execution

```
check_f_ok($encoding);
Bits<32> sp_value_a = f[fs1][31:0];
Bits<32> sp_value_b = f[fs1][31:0];
if (is_sp_nan?(sp_value_a) || is_sp_nan?(sp_value_b)) {
  if (is_sp_signaling_nan?(sp_value_a) || is_sp_signaling_nan?(sp_value_b)) {
    set_fp_flag(FpFlag::NV);
  }
  X[rd] = 0;
} else {
  X[rd] = sp_value_a == sp_value_b) || ((sp_value_a | sp_value_b)[30:0] == 0 ? 1 : 0;
}
```

B.128.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

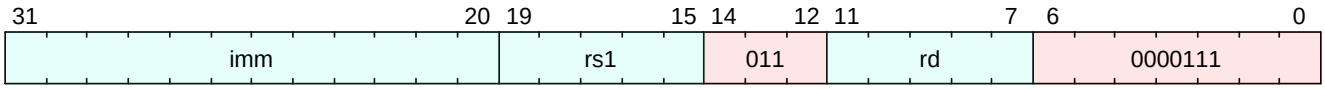
B.129. fld

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.129.1. Encoding



B.129.2. Synopsis

No description available.

B.129.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.129.4. Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.129.5. Execution

```

```

B.129.6. Exceptions

This instruction does not generate synchronous exceptions.

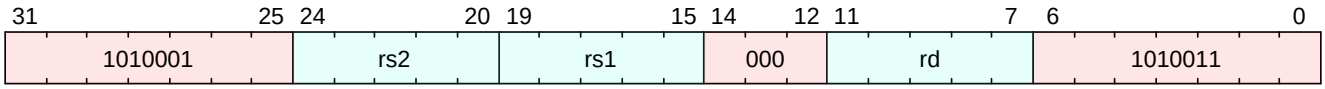
B.130. fle.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.130.1. Encoding



B.130.2. Synopsis

No description available.

B.130.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.130.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.130.5. Execution

```

```

B.130.6. Exceptions

This instruction does not generate synchronous exceptions.

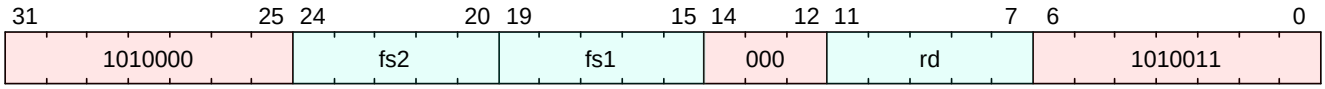
B.131. fle.s

Single-precision floating-point less than or equal

This instruction is defined by:

- F, version >= 0

B.131.1. Encoding



B.131.2. Synopsis

Writes 1 to *rd* if *fs1* is less than or equal to *fs2*, and 0 otherwise.

If either operand is NaN, the result is 0 (not equal). If either operand is a NaN (signaling or quiet), the invalid flag is set.

Positive zero and negative zero are considered equal.

B.131.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.131.4. Decode Variables

```
Bits<5> fs2 = $encoding[24:20];
Bits<5> fs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.131.5. Execution

```
check_f_ok($encoding);
Bits<32> sp_value_a = f[fs1][31:0];
Bits<32> sp_value_b = f[fs2][31:0];
if (is_sp_nan?(sp_value_a) || is_sp_nan?(sp_value_b)) {
  if (is_sp_signaling_nan?(sp_value_a) || is_sp_signaling_nan?(sp_value_b)) {
    set_fp_flag(FpFlag::NV);
  }
  X[rd] = 0;
} else {
  X[rd] = sp_value_a == sp_value_b) || ((sp_value_a | sp_value_b)[30:0] == 0 ? 1 : 0;
}
```

B.131.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

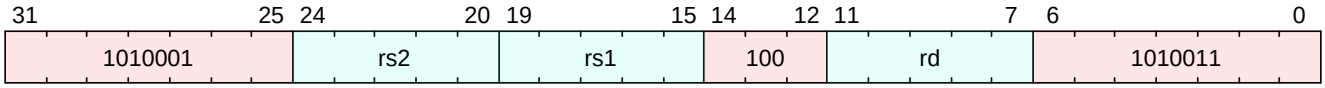
B.132. fleq.d

No synopsis available.

This instruction is defined by:

- anyOf:
  - D, version >= 0
  - Zfa, version >= 0

B.132.1. Encoding



B.132.2. Synopsis

No description available.

B.132.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.132.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.132.5. Execution

```

```

B.132.6. Exceptions

This instruction does not generate synchronous exceptions.

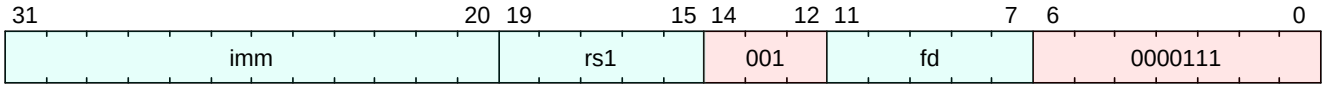
B.133. flh

Half-precision floating-point load

This instruction is defined by:

- anyOf:
  - Zfh, version >= 0
  - Zfhmin, version >= 0

B.133.1. Encoding



B.133.2. Synopsis

The flh instruction loads a single-precision floating-point value from memory at address *rs1* + *imm* into floating-point register *rd*.

flh does not modify the bits being transferred; in particular, the payloads of non-canonical NaNs are preserved.

flh is only guaranteed to execute atomically if the effective address is naturally aligned.

B.133.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.133.4. Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> fd = $encoding[11:7];
```

B.133.5. Execution

```
check_f_ok($encoding);
XReg virtual_address = X[rs1] + $signed(imm);
Bits<16> hp_value = read_memory<16>(virtual_address, $encoding);
f[fd] = nan_box<16, FLEN>(hp_value);
mark_f_state_dirty();
```

B.133.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- LoadAddressMisaligned
- LoadPageFault

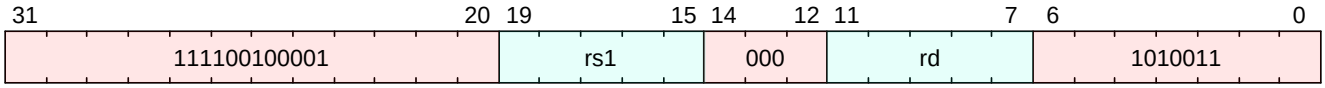
B.134. fli.d

No synopsis available.

This instruction is defined by:

- anyOf:
  - D, version >= 0
  - Zfa, version >= 0

B.134.1. Encoding



B.134.2. Synopsis

No description available.

B.134.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.134.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.134.5. Execution

```

```

B.134.6. Exceptions

This instruction does not generate synchronous exceptions.

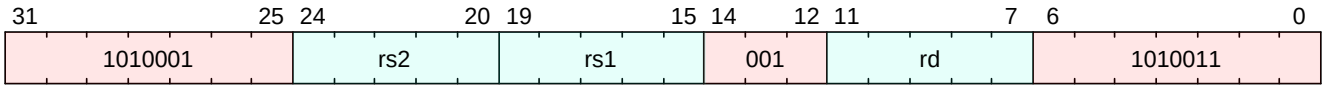
B.135. flt.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.135.1. Encoding



B.135.2. Synopsis

No description available.

B.135.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.135.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.135.5. Execution

```

```

B.135.6. Exceptions

This instruction does not generate synchronous exceptions.

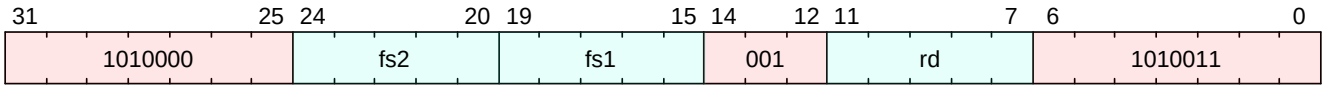
B.136. flt.s

Single-precision floating-point less than

This instruction is defined by:

- F, version >= 0

B.136.1. Encoding



B.136.2. Synopsis

Writes 1 to *rd* if *fs1* is less than *fs2*, and 0 otherwise.

If either operand is NaN, the result is 0 (not equal). If either operand is a NaN (signaling or quiet), the invalid flag is set.

B.136.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.136.4. Decode Variables

```
Bits<5> fs2 = $encoding[24:20];
Bits<5> fs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.136.5. Execution

```
check_f_ok($encoding);
Bits<32> sp_value_a = f[fs1][31:0];
Bits<32> sp_value_b = f[fs2][31:0];
if (is_sp_nan?(sp_value_a) || is_sp_nan?(sp_value_b)) {
  set_fp_flag(FpFlag::NV);
  X[rd] = 0;
} else {
  Boolean sign_a = sp_value_a[31] == 1;
  Boolean sign_b = sp_value_b[31] == 1;
  Boolean a_lt_b = (sign_a != sign_b) ? (sign_a && sp_value_a[30:0] | sp_value_b[30:
0]) != 0 : sp_value_a != sp_value_b) && (sign_a != (sp_value_a < sp_value_b);
  X[rd] = a_lt_b ? 1 : 0;
}
```

B.136.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

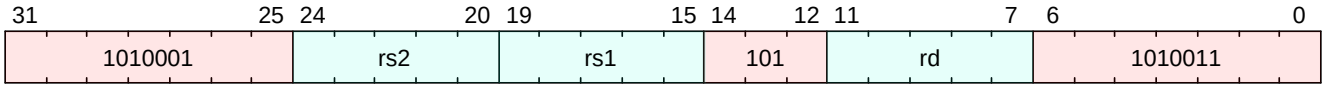
B.137. fltq.d

No synopsis available.

This instruction is defined by:

- anyOf:
  - D, version >= 0
  - Zfa, version >= 0

B.137.1. Encoding



B.137.2. Synopsis

No description available.

B.137.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.137.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.137.5. Execution

```

```

B.137.6. Exceptions

This instruction does not generate synchronous exceptions.

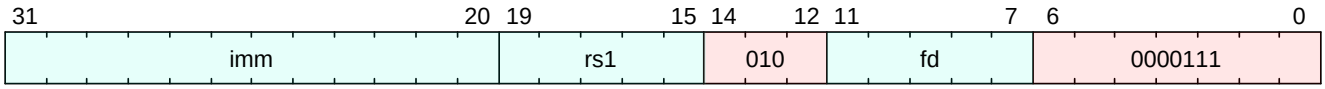
B.138. flw

Single-precision floating-point load

This instruction is defined by:

- F, version >= 0

B.138.1. Encoding



B.138.2. Synopsis

The flw instruction loads a single-precision floating-point value from memory at address *rs1* + *imm* into floating-point register *fd*.

flw does not modify the bits being transferred; in particular, the payloads of non-canonical NaNs are preserved.

B.138.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.138.4. Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> fd = $encoding[11:7];
```

B.138.5. Execution

```
check_f_ok($encoding);
XReg virtual_address = X[rs1] + $signed(imm);
Bits<32> sp_value = read_memory<32>(virtual_address, $encoding);
if (implemented?(ExtensionName::D)) {
  f[fd] = nan_box<32, 64>(sp_value);
} else {
  f[fd] = sp_value;
}
mark_f_state_dirty();
```

B.138.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- LoadAddressMisaligned
- LoadPageFault

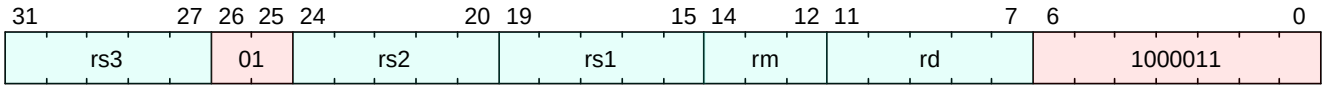
B.139. fmadd.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.139.1. Encoding



B.139.2. Synopsis

No description available.

B.139.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.139.4. Decode Variables

```
Bits<5> rs3 = $encoding[31:27];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.139.5. Execution

```

```

B.139.6. Exceptions

This instruction does not generate synchronous exceptions.

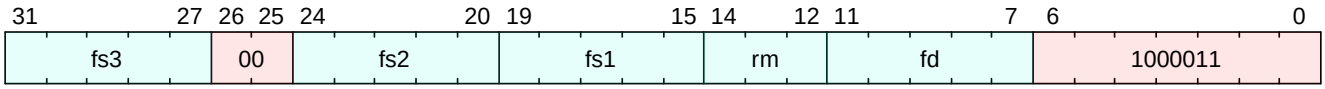
B.140. fmadd.s

No synopsis available.

This instruction is defined by:

- F, version >= 0

B.140.1. Encoding



B.140.2. Synopsis

No description available.

B.140.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.140.4. Decode Variables

```
Bits<5> fs3 = $encoding[31:27];
Bits<5> fs2 = $encoding[24:20];
Bits<5> fs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> fd = $encoding[11:7];
```

B.140.5. Execution

```

```

B.140.6. Exceptions

This instruction does not generate synchronous exceptions.

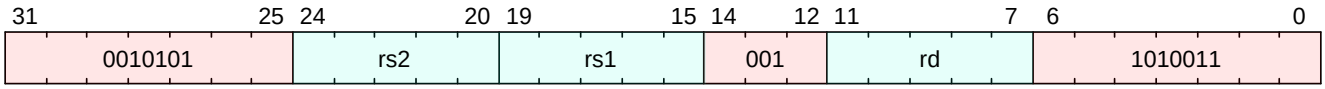
B.141. fmax.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.141.1. Encoding



B.141.2. Synopsis

No description available.

B.141.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.141.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.141.5. Execution

```

```

B.141.6. Exceptions

This instruction does not generate synchronous exceptions.

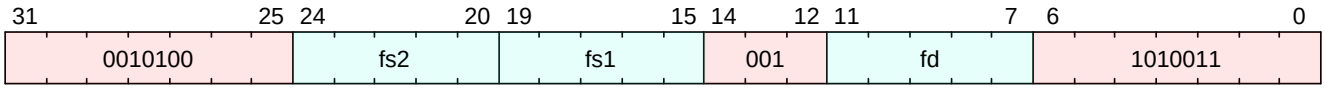
B.142. fmax.s

No synopsis available.

This instruction is defined by:

- F, version >= 0

B.142.1. Encoding



B.142.2. Synopsis

No description available.

B.142.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.142.4. Decode Variables

```
Bits<5> fs2 = $encoding[24:20];
Bits<5> fs1 = $encoding[19:15];
Bits<5> fd = $encoding[11:7];
```

B.142.5. Execution

```

```

B.142.6. Exceptions

This instruction does not generate synchronous exceptions.

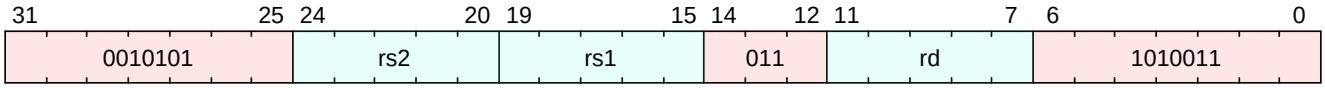
B.143. fmaxm.d

No synopsis available.

This instruction is defined by:

- anyOf:
  - D, version >= 0
  - Zfa, version >= 0

B.143.1. Encoding



B.143.2. Synopsis

No description available.

B.143.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.143.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.143.5. Execution

```

```

B.143.6. Exceptions

This instruction does not generate synchronous exceptions.

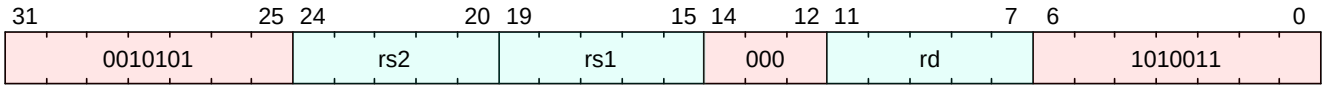
B.144. fmin.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.144.1. Encoding



B.144.2. Synopsis

No description available.

B.144.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.144.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.144.5. Execution

```

```

B.144.6. Exceptions

This instruction does not generate synchronous exceptions.

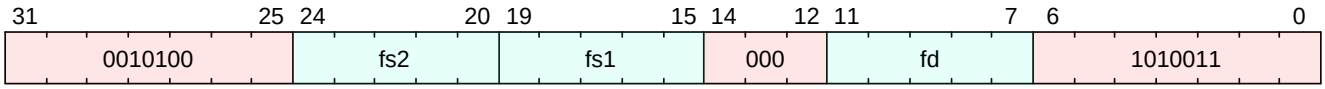
B.145. fmin.s

No synopsis available.

This instruction is defined by:

- F, version >= 0

B.145.1. Encoding



B.145.2. Synopsis

No description available.

B.145.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.145.4. Decode Variables

```
Bits<5> fs2 = $encoding[24:20];
Bits<5> fs1 = $encoding[19:15];
Bits<5> fd = $encoding[11:7];
```

B.145.5. Execution

```

```

B.145.6. Exceptions

This instruction does not generate synchronous exceptions.

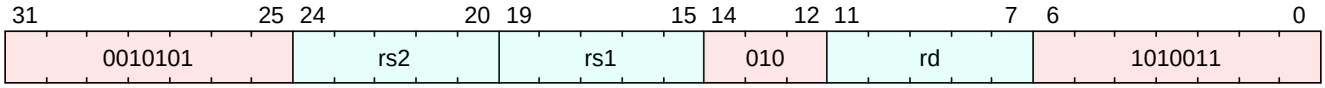
B.146. fminm.d

No synopsis available.

This instruction is defined by:

- anyOf:
  - D, version >= 0
  - Zfa, version >= 0

B.146.1. Encoding



B.146.2. Synopsis

No description available.

B.146.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.146.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.146.5. Execution

```

```

B.146.6. Exceptions

This instruction does not generate synchronous exceptions.

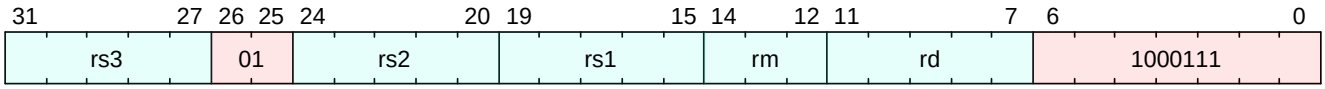
B.147. fmsub.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.147.1. Encoding



B.147.2. Synopsis

No description available.

B.147.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.147.4. Decode Variables

```
Bits<5> rs3 = $encoding[31:27];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.147.5. Execution

```

```

B.147.6. Exceptions

This instruction does not generate synchronous exceptions.

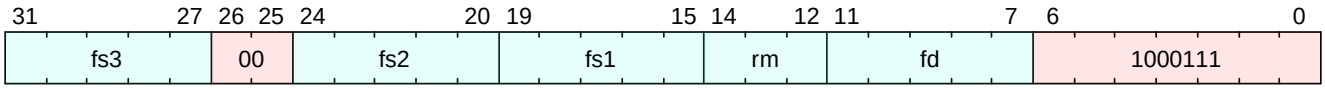
B.148. fmsub.s

No synopsis available.

This instruction is defined by:

- F, version >= 0

B.148.1. Encoding



B.148.2. Synopsis

No description available.

B.148.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.148.4. Decode Variables

```
Bits<5> fs3 = $encoding[31:27];
Bits<5> fs2 = $encoding[24:20];
Bits<5> fs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> fd = $encoding[11:7];
```

B.148.5. Execution

```

```

B.148.6. Exceptions

This instruction does not generate synchronous exceptions.

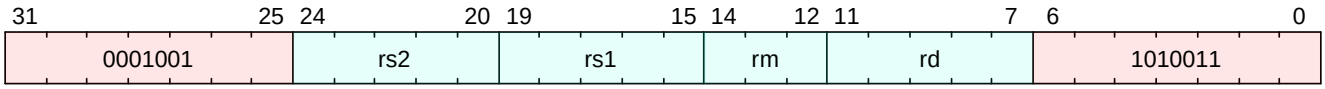
B.149. fmul.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.149.1. Encoding



B.149.2. Synopsis

No description available.

B.149.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.149.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.149.5. Execution

```

```

B.149.6. Exceptions

This instruction does not generate synchronous exceptions.

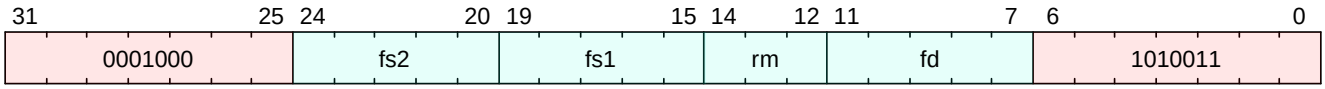
B.150. fmul.s

No synopsis available.

This instruction is defined by:

- F, version >= 0

B.150.1. Encoding



B.150.2. Synopsis

No description available.

B.150.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.150.4. Decode Variables

```
Bits<5> fs2 = $encoding[24:20];
Bits<5> fs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> fd = $encoding[11:7];
```

B.150.5. Execution

```

```

B.150.6. Exceptions

This instruction does not generate synchronous exceptions.

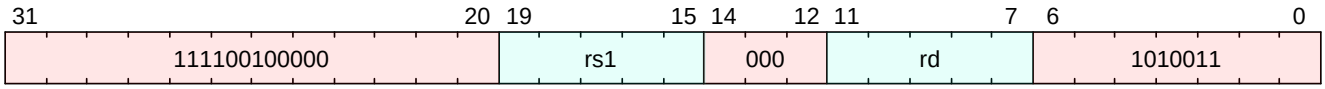
B.151. fmv.d.x

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.151.1. Encoding



B.151.2. Synopsis

No description available.

B.151.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.151.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.151.5. Execution

```

```

B.151.6. Exceptions

This instruction does not generate synchronous exceptions.

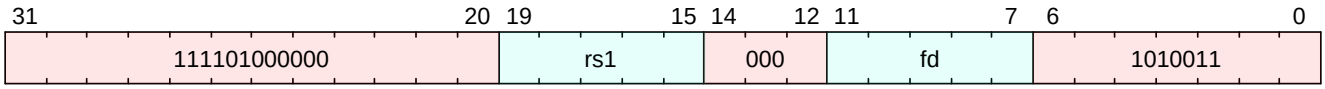
B.152. fmv.h.x

Half-precision floating-point move from integer

This instruction is defined by:

- F, version >= 0

B.152.1. Encoding



B.152.2. Synopsis

Moves the half-precision value encoded in IEEE 754-2008 standard encoding from the lower 16 bits of integer register `rs1` to the floating-point register `fd`. The bits are not modified in the transfer, and in particular, the payloads of non-canonical NaNs are preserved.

B.152.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.152.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> fd = $encoding[11:7];
```

B.152.5. Execution

```
check_f_ok($encoding);
Bits<16> hp_value = X[rs1][15:0];
f[fd] = nan_box<16, FLEN>(hp_value);
mark_f_state_dirty();
```

B.152.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

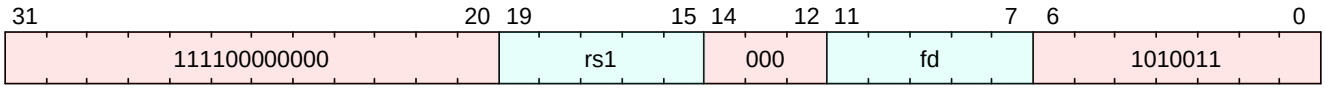
B.153. fmv.w.x

Single-precision floating-point move from integer

This instruction is defined by:

- F, version >= 0

B.153.1. Encoding



B.153.2. Synopsis

Moves the single-precision value encoded in IEEE 754-2008 standard encoding from the lower 32 bits of integer register `rs1` to the floating-point register `fd`. The bits are not modified in the transfer, and in particular, the payloads of non-canonical NaNs are preserved.

B.153.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.153.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> fd = $encoding[11:7];
```

B.153.5. Execution

```
check_f_ok($encoding);
Bits<32> sp_value = X[rs1][31:0];
if (implemented?(ExtensionName::D)) {
  f[fd] = nan_box<32, 64>(sp_value);
} else {
  f[fd] = sp_value;
}
mark_f_state_dirty();
```

B.153.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

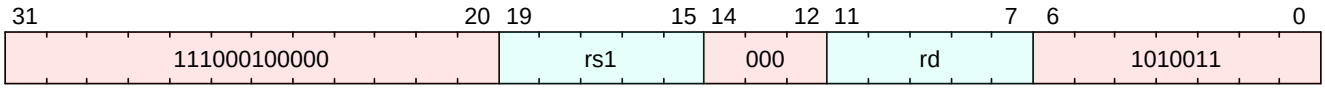
B.154. fmv.x.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.154.1. Encoding



B.154.2. Synopsis

No description available.

B.154.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.154.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.154.5. Execution

```

```

B.154.6. Exceptions

This instruction does not generate synchronous exceptions.

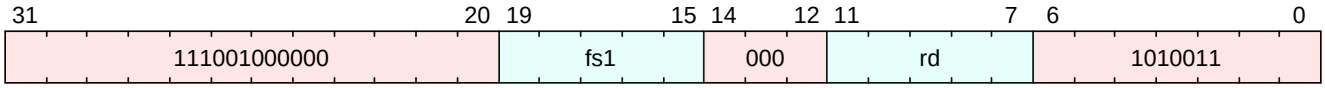
B.155. fmv.x.h

Move half-precision value from floating-point to integer register

This instruction is defined by:

- anyOf:
  - Zfh, version >= 0
  - Zfhmin, version >= 0

B.155.1. Encoding



B.155.2. Synopsis

Moves the half-precision value in floating-point register rs1 represented in IEEE 754-2008 encoding to the lower 16 bits of integer register rd.

The bits are not modified in the transfer, and in particular, the payloads of non-canonical NaNs are preserved.

The highest XLEN-16 bits of the destination register are filled with copies of the floating-point number's sign bit.

B.155.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.155.4. Decode Variables

```
Bits<5> fs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.155.5. Execution

```
check_f_ok($encoding);
X[rd] = sext(f[fs1][15:0], 16);
```

B.155.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

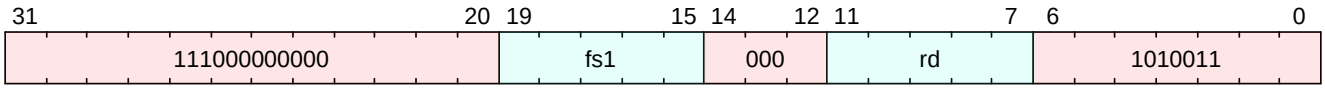
B.156. fmv.x.w

Move single-precision value from floating-point to integer register

This instruction is defined by:

- F, version >= 0

B.156.1. Encoding



B.156.2. Synopsis

Moves the single-precision value in floating-point register rs1 represented in IEEE 754-2008 encoding to the lower 32 bits of integer register rd. The bits are not modified in the transfer, and in particular, the payloads of non-canonical NaNs are preserved. For RV64, the higher 32 bits of the destination register are filled with copies of the floating-point number's sign bit.

B.156.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.156.4. Decode Variables

```
Bits<5> fs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.156.5. Execution

```
check_f_ok($encoding);
X[rd] = sext(f[fs1][31:0], 32);
```

B.156.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

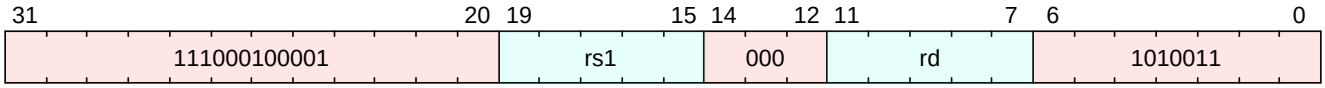
B.157. fmvh.x.d

No synopsis available.

This instruction is defined by:

- anyOf:
  - D, version >= 0
  - Zfa, version >= 0

B.157.1. Encoding



B.157.2. Synopsis

No description available.

B.157.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.157.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.157.5. Execution

```

```

B.157.6. Exceptions

This instruction does not generate synchronous exceptions.

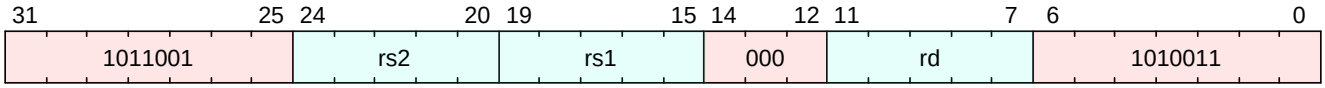
B.158. fmvp.d.x

No synopsis available.

This instruction is defined by:

- anyOf:
  - D, version >= 0
  - Zfa, version >= 0

B.158.1. Encoding



B.158.2. Synopsis

No description available.

B.158.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.158.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.158.5. Execution

```

```

B.158.6. Exceptions

This instruction does not generate synchronous exceptions.

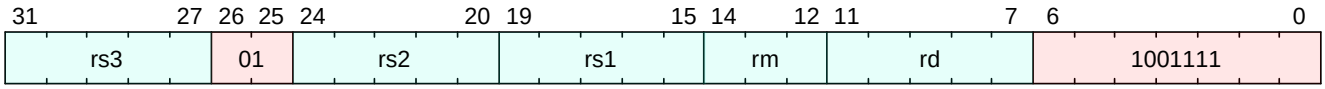
B.159. fnmadd.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.159.1. Encoding



B.159.2. Synopsis

No description available.

B.159.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.159.4. Decode Variables

```
Bits<5> rs3 = $encoding[31:27];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.159.5. Execution

```

```

B.159.6. Exceptions

This instruction does not generate synchronous exceptions.

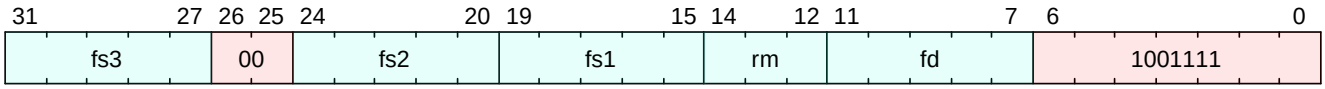
B.160. fnmadd.s

No synopsis available.

This instruction is defined by:

- F, version >= 0

B.160.1. Encoding



B.160.2. Synopsis

No description available.

B.160.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.160.4. Decode Variables

```
Bits<5> fs3 = $encoding[31:27];
Bits<5> fs2 = $encoding[24:20];
Bits<5> fs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> fd = $encoding[11:7];
```

B.160.5. Execution

```

```

B.160.6. Exceptions

This instruction does not generate synchronous exceptions.

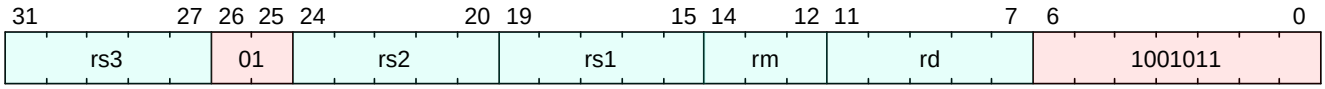
B.161. fnmsub.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.161.1. Encoding



B.161.2. Synopsis

No description available.

B.161.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.161.4. Decode Variables

```
Bits<5> rs3 = $encoding[31:27];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.161.5. Execution

```

```

B.161.6. Exceptions

This instruction does not generate synchronous exceptions.

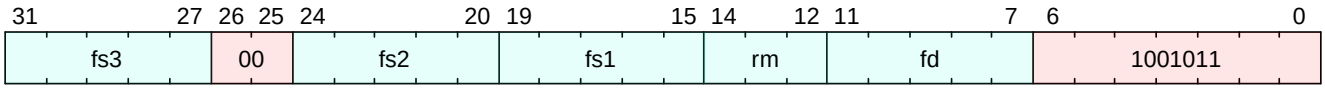
B.162. fnmsub.s

No synopsis available.

This instruction is defined by:

- F, version >= 0

B.162.1. Encoding



B.162.2. Synopsis

No description available.

B.162.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.162.4. Decode Variables

```
Bits<5> fs3 = $encoding[31:27];
Bits<5> fs2 = $encoding[24:20];
Bits<5> fs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> fd = $encoding[11:7];
```

B.162.5. Execution

```

```

B.162.6. Exceptions

This instruction does not generate synchronous exceptions.

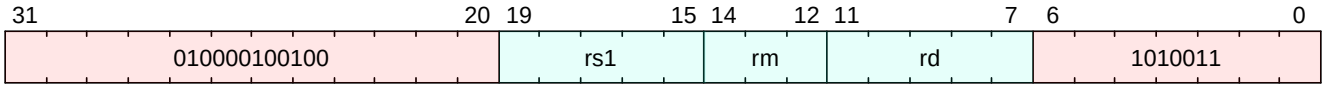
B.163. fround.d

No synopsis available.

This instruction is defined by:

- anyOf:
  - D, version >= 0
  - Zfa, version >= 0

B.163.1. Encoding



B.163.2. Synopsis

No description available.

B.163.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.163.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.163.5. Execution

```

```

B.163.6. Exceptions

This instruction does not generate synchronous exceptions.

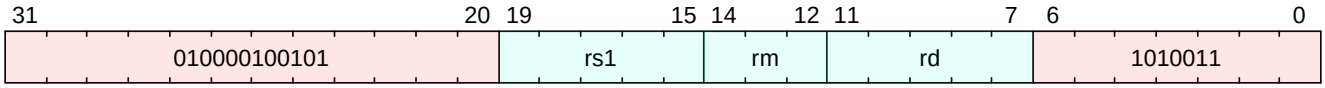
B.164. froundnx.d

No synopsis available.

This instruction is defined by:

- anyOf:
  - D, version >= 0
  - Zfa, version >= 0

B.164.1. Encoding



B.164.2. Synopsis

No description available.

B.164.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.164.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.164.5. Execution

```

```

B.164.6. Exceptions

This instruction does not generate synchronous exceptions.

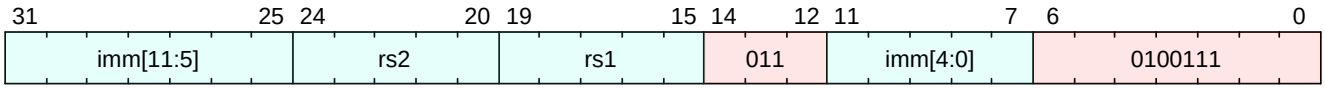
B.165. fsd

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.165.1. Encoding



B.165.2. Synopsis

No description available.

B.165.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.165.4. Decode Variables

```
Bits<12> imm = {$encoding[31:25], $encoding[11:7]};
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

B.165.5. Execution

```

```

B.165.6. Exceptions

This instruction does not generate synchronous exceptions.

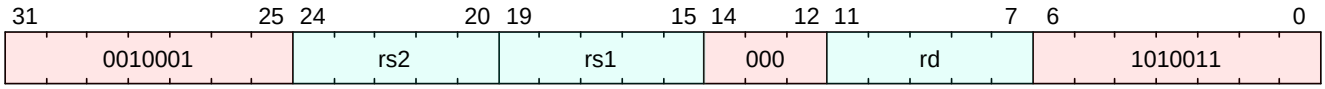
B.166. fsgnj.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.166.1. Encoding



B.166.2. Synopsis

No description available.

B.166.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.166.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.166.5. Execution

```

```

B.166.6. Exceptions

This instruction does not generate synchronous exceptions.

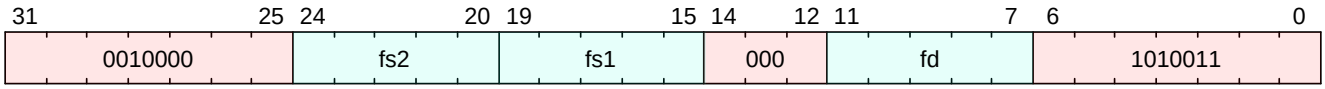
B.167. fsgnj.s

Single-precision sign inject

This instruction is defined by:

- F, version >= 0

B.167.1. Encoding



B.167.2. Synopsis

Writes *fd* with sign bit of *fs2* and the exponent and mantissa of *fs1*.

Sign-injection instructions do not set floating-point exception flags, nor do they canonicalize NaNs.

B.167.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.167.4. Decode Variables

```
Bits<5> fs2 = $encoding[24:20];
Bits<5> fs1 = $encoding[19:15];
Bits<5> fd = $encoding[11:7];
```

B.167.5. Execution

```
check_f_ok($encoding);
Bits<32> sp_value = {f[fs2][31], f[fs1][30:0]};
if (implemented?(ExtensionName::D)) {
  f[fd] = nan_box<32, 64>(sp_value);
} else {
  f[fd] = sp_value;
}
mark_f_state_dirty();
```

B.167.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

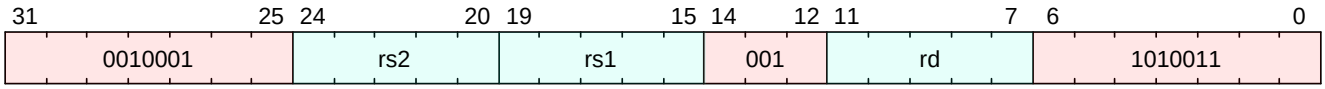
B.168. fsgnjn.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.168.1. Encoding



B.168.2. Synopsis

No description available.

B.168.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.168.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.168.5. Execution

```

```

B.168.6. Exceptions

This instruction does not generate synchronous exceptions.

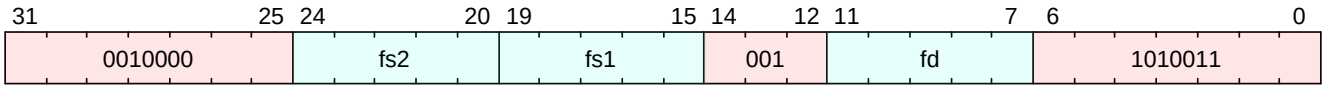
B.169. fsgnjn.s

Single-precision sign inject negate

This instruction is defined by:

- F, version >= 0

B.169.1. Encoding



B.169.2. Synopsis

Writes *fd* with the opposite of the sign bit of *fs2* and the exponent and mantissa of *fs1*.

Sign-injection instructions do not set floating-point exception flags, nor do they canonicalize NaNs.

B.169.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.169.4. Decode Variables

```
Bits<5> fs2 = $encoding[24:20];
Bits<5> fs1 = $encoding[19:15];
Bits<5> fd = $encoding[11:7];
```

B.169.5. Execution

```
check_f_ok($encoding);
Bits<32> sp_value = {~f[fs2][31], f[fs1][30:0]};
if (implemented?(ExtensionName::D)) {
  f[fd] = nan_box<32, 64>(sp_value);
} else {
  f[fd] = sp_value;
}
mark_f_state_dirty();
```

B.169.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

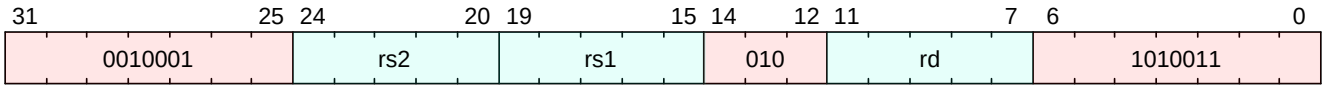
B.170. fsgnjx.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.170.1. Encoding



B.170.2. Synopsis

No description available.

B.170.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.170.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.170.5. Execution

```

```

B.170.6. Exceptions

This instruction does not generate synchronous exceptions.

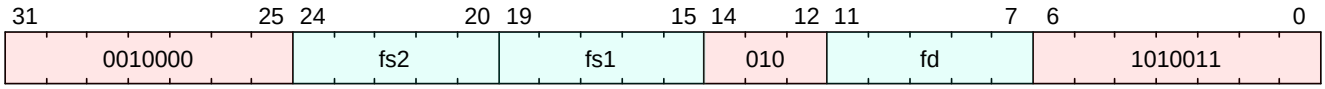
B.171. fsgnjx.s

Single-precision sign inject exclusive or

This instruction is defined by:

- F, version >= 0

B.171.1. Encoding



B.171.2. Synopsis

Writes *fd* with the xor of the sign bits of *fs2* and *fs1* and the exponent and mantissa of *fs1*.

Sign-injection instructions do not set floating-point exception flags, nor do they canonicalize NaNs.

B.171.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.171.4. Decode Variables

```
Bits<5> fs2 = $encoding[24:20];
Bits<5> fs1 = $encoding[19:15];
Bits<5> fd = $encoding[11:7];
```

B.171.5. Execution

```
check_f_ok($encoding);
Bits<32> sp_value = {f[fs1][31] ^ f[fs2][31], f[fs1][30:0]};
if (implemented?(ExtensionName::D)) {
  f[fd] = nan_box<32, 64>(sp_value);
} else {
  f[fd] = sp_value;
}
mark_f_state_dirty();
```

B.171.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

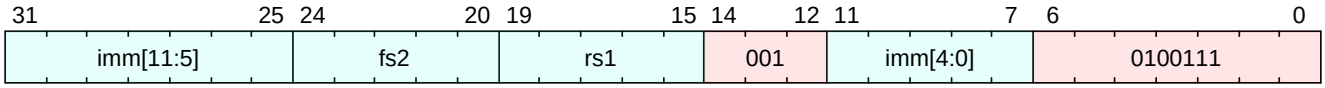
B.172. fsh

Half-precision floating-point store

This instruction is defined by:

- anyOf:
  - Zfh, version >= 0
  - Zfhmin, version >= 0

B.172.1. Encoding



B.172.2. Synopsis

The fsh instruction stores a half-precision floating-point value from register *rd* to memory at address *rs1* + *imm*.

fsh does not modify the bits being transferred; in particular, the payloads of non-canonical NaNs are preserved.

fsh ignores all but the lower 16 bits in *rs2*.

fsh is only guaranteed to execute atomically if the effective address is naturally aligned.

B.172.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.172.4. Decode Variables

```
Bits<12> imm = {$encoding[31:25], $encoding[11:7]};
Bits<5> rs1 = $encoding[19:15];
Bits<5> fs2 = $encoding[24:20];
```

B.172.5. Execution

```
check_f_ok($encoding);
XReg virtual_address = X[rs1] + $signed(imm);
Bits<16> hp_value = f[fs2][15:0];
write_memory<16>(virtual_address, hp_value, $encoding);
```

B.172.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

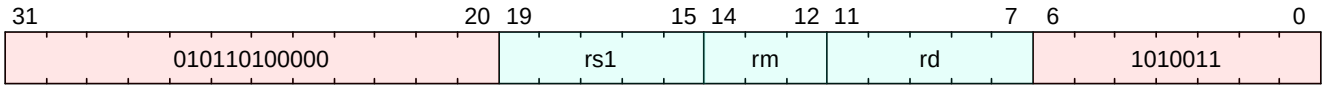
B.173. fsqrt.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.173.1. Encoding



B.173.2. Synopsis

No description available.

B.173.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.173.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.173.5. Execution

```

```

B.173.6. Exceptions

This instruction does not generate synchronous exceptions.

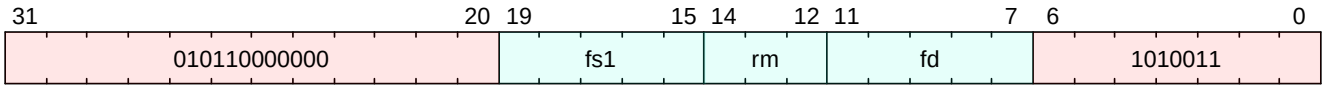
B.174. fsqrt.s

No synopsis available.

This instruction is defined by:

- F, version >= 0

B.174.1. Encoding



B.174.2. Synopsis

No description available.

B.174.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.174.4. Decode Variables

```
Bits<5> fs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> fd = $encoding[11:7];
```

B.174.5. Execution

```

```

B.174.6. Exceptions

This instruction does not generate synchronous exceptions.

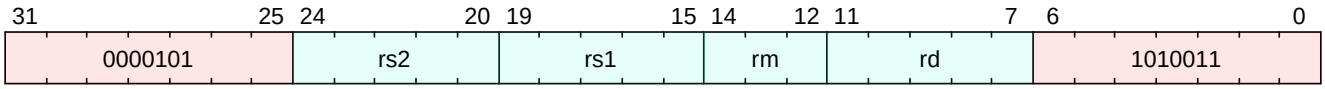
B.175. fsub.d

No synopsis available.

This instruction is defined by:

- D, version >= 0

B.175.1. Encoding



B.175.2. Synopsis

No description available.

B.175.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.175.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> rd = $encoding[11:7];
```

B.175.5. Execution

```

```

B.175.6. Exceptions

This instruction does not generate synchronous exceptions.

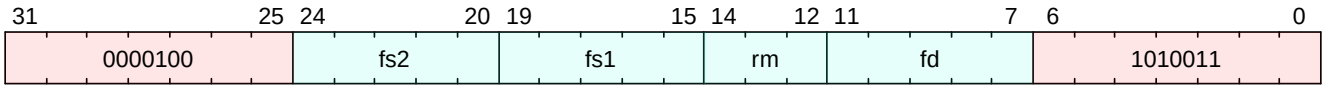
B.176. fsub.s

No synopsis available.

This instruction is defined by:

- F, version >= 0

B.176.1. Encoding



B.176.2. Synopsis

No description available.

B.176.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.176.4. Decode Variables

```
Bits<5> fs2 = $encoding[24:20];
Bits<5> fs1 = $encoding[19:15];
Bits<3> rm = $encoding[14:12];
Bits<5> fd = $encoding[11:7];
```

B.176.5. Execution

```

```

B.176.6. Exceptions

This instruction does not generate synchronous exceptions.

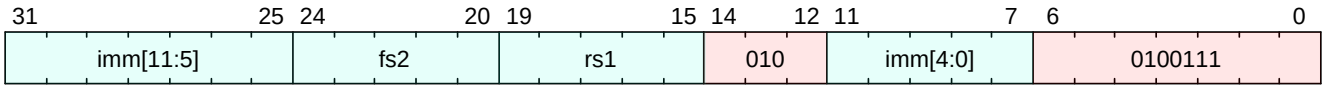
B.177. fsw

Single-precision floating-point store

This instruction is defined by:

- F, version >= 0

B.177.1. Encoding



B.177.2. Synopsis

The fsw instruction stores a single-precision floating-point value in *fs2* to memory at address *rs1* + *imm*.

fsw does not modify the bits being transferred; in particular, the payloads of non-canonical NaNs are preserved.

B.177.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.177.4. Decode Variables

```
Bits<12> imm = {$encoding[31:25], $encoding[11:7]};
Bits<5> fs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

B.177.5. Execution

```
check_f_ok($encoding);
XReg virtual_address = X[rs1] + $signed(imm);
write_memory<32>(virtual_address, f[fs2][31:0], $encoding);
```

B.177.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- StoreAmoAccessFault

- StoreAmoAddressMisaligned
- StoreAmoPageFault

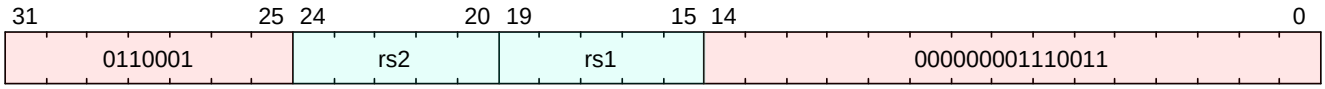
B.178. hfence.gvma

No synopsis available.

This instruction is defined by:

- H, version >= 0

B.178.1. Encoding



B.178.2. Synopsis

No description available.

B.178.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.178.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

B.178.5. Execution

```

```

B.178.6. Exceptions

This instruction does not generate synchronous exceptions.

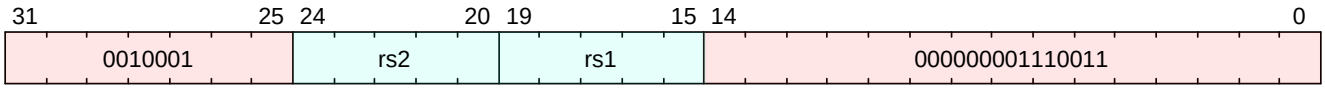
B.179. hfence.vvma

No synopsis available.

This instruction is defined by:

- H, version >= 0

B.179.1. Encoding



B.179.2. Synopsis

No description available.

B.179.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.179.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

B.179.5. Execution

```

```

B.179.6. Exceptions

This instruction does not generate synchronous exceptions.

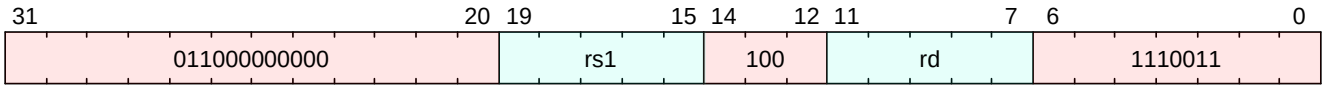
B.180. hlv.b

No synopsis available.

This instruction is defined by:

- H, version >= 0

B.180.1. Encoding



B.180.2. Synopsis

No description available.

B.180.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.180.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.180.5. Execution

```

```

B.180.6. Exceptions

This instruction does not generate synchronous exceptions.

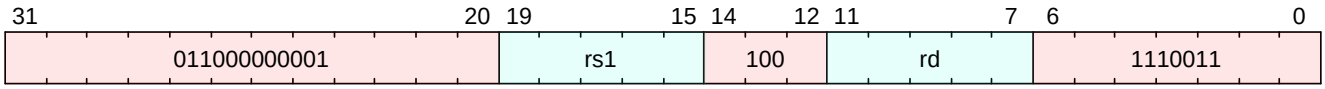
B.181. hlv.bu

No synopsis available.

This instruction is defined by:

- H, version >= 0

B.181.1. Encoding



B.181.2. Synopsis

No description available.

B.181.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.181.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.181.5. Execution

```

```

B.181.6. Exceptions

This instruction does not generate synchronous exceptions.

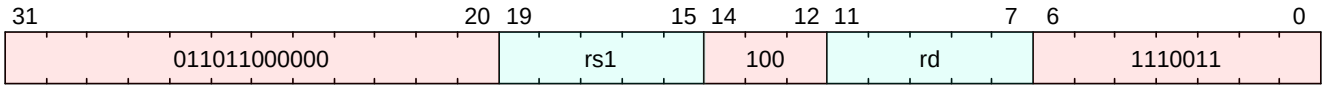
B.182. hlv.d

No synopsis available.

This instruction is defined by:

- H, version >= 0

B.182.1. Encoding



B.182.2. Synopsis

No description available.

B.182.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.182.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.182.5. Execution

```

```

B.182.6. Exceptions

This instruction does not generate synchronous exceptions.

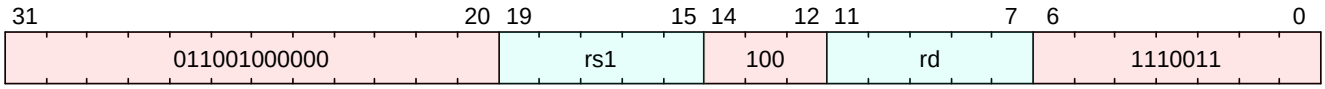
B.183. hlv.h

No synopsis available.

This instruction is defined by:

- H, version >= 0

B.183.1. Encoding



B.183.2. Synopsis

No description available.

B.183.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.183.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.183.5. Execution

```

```

B.183.6. Exceptions

This instruction does not generate synchronous exceptions.

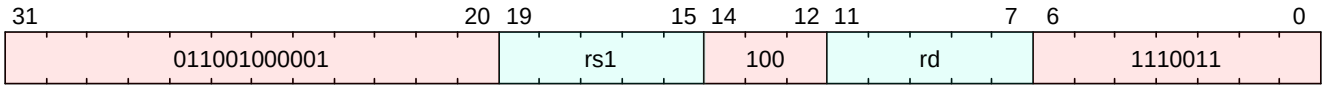
B.184. hlv.hu

No synopsis available.

This instruction is defined by:

- H, version >= 0

B.184.1. Encoding



B.184.2. Synopsis

No description available.

B.184.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.184.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.184.5. Execution

```

```

B.184.6. Exceptions

This instruction does not generate synchronous exceptions.

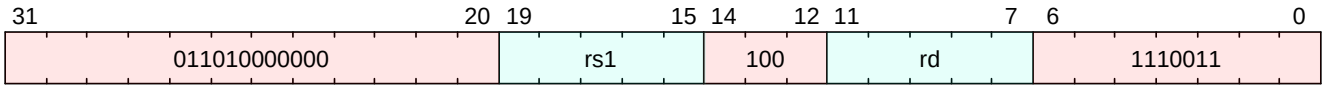
B.185. hlv.w

No synopsis available.

This instruction is defined by:

- H, version >= 0

B.185.1. Encoding



B.185.2. Synopsis

No description available.

B.185.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.185.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.185.5. Execution

```

```

B.185.6. Exceptions

This instruction does not generate synchronous exceptions.

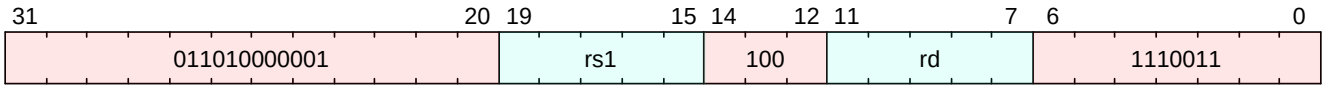
B.186. hlv.wu

No synopsis available.

This instruction is defined by:

- H, version >= 0

B.186.1. Encoding



B.186.2. Synopsis

No description available.

B.186.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.186.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.186.5. Execution

```

```

B.186.6. Exceptions

This instruction does not generate synchronous exceptions.

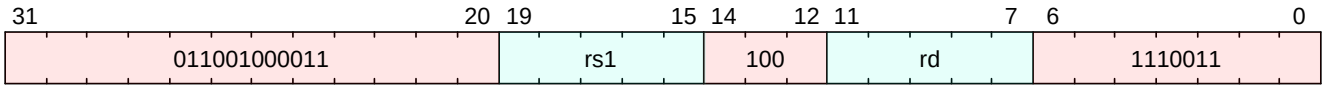
B.187. hlvx.hu

No synopsis available.

This instruction is defined by:

- H, version >= 0

B.187.1. Encoding



B.187.2. Synopsis

No description available.

B.187.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.187.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.187.5. Execution

```

```

B.187.6. Exceptions

This instruction does not generate synchronous exceptions.

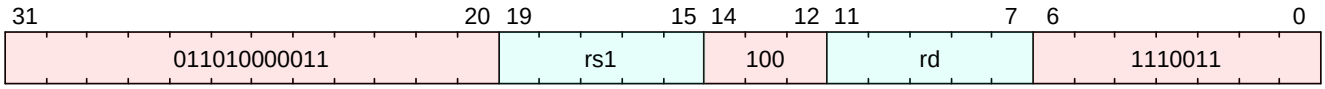
B.188. hlvx.wu

No synopsis available.

This instruction is defined by:

- H, version >= 0

B.188.1. Encoding



B.188.2. Synopsis

No description available.

B.188.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.188.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.188.5. Execution

```

```

B.188.6. Exceptions

This instruction does not generate synchronous exceptions.

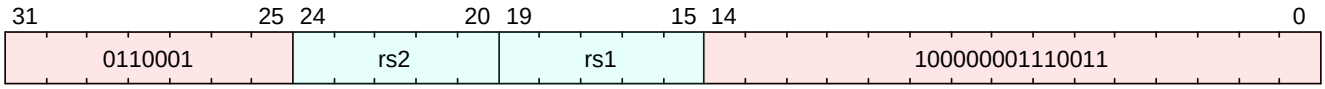
B.189. hsv.b

No synopsis available.

This instruction is defined by:

- H, version >= 0

B.189.1. Encoding



B.189.2. Synopsis

No description available.

B.189.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.189.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

B.189.5. Execution

```

```

B.189.6. Exceptions

This instruction does not generate synchronous exceptions.

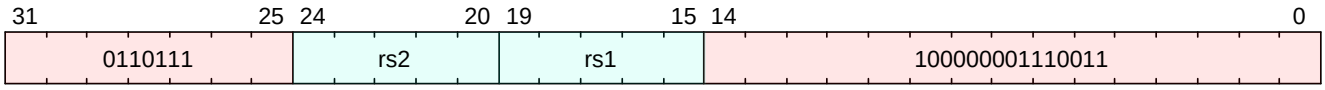
B.190. hsv.d

No synopsis available.

This instruction is defined by:

- H, version >= 0

B.190.1. Encoding



B.190.2. Synopsis

No description available.

B.190.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.190.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

B.190.5. Execution

```

```

B.190.6. Exceptions

This instruction does not generate synchronous exceptions.

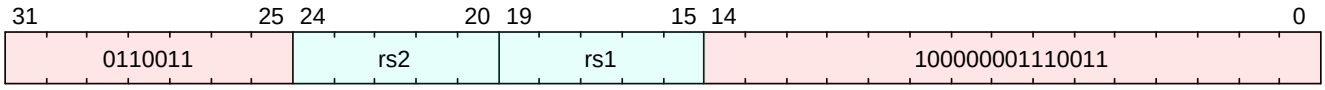
B.191. hsv.h

No synopsis available.

This instruction is defined by:

- H, version >= 0

B.191.1. Encoding



B.191.2. Synopsis

No description available.

B.191.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.191.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

B.191.5. Execution

```

```

B.191.6. Exceptions

This instruction does not generate synchronous exceptions.

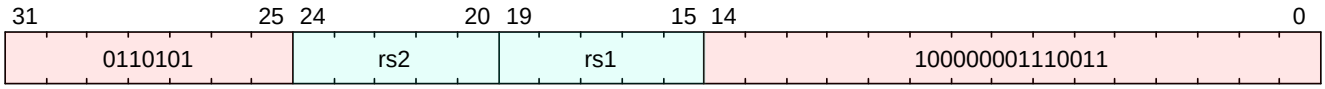
B.192. hsv.w

No synopsis available.

This instruction is defined by:

- H, version >= 0

B.192.1. Encoding



B.192.2. Synopsis

No description available.

B.192.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.192.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

B.192.5. Execution

```

```

B.192.6. Exceptions

This instruction does not generate synchronous exceptions.

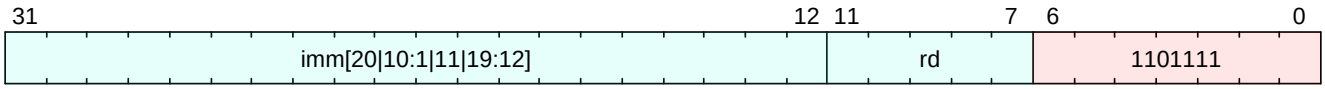
B.193. jal

Jump and link

This instruction is defined by:

- I, version >= 0

B.193.1. Encoding



B.193.2. Synopsis

Jump to a PC-relative offset and store the return address in rd.

B.193.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.193.4. Decode Variables

```
signed Bits<21> imm = sext({$encoding[31], $encoding[19:12], $encoding[20], $encoding
[30:21], 1'd0});
Bits<5> rd = $encoding[11:7];
```

B.193.5. Execution

```
XReg retrun_addr = $pc + 4;
jump_halfword($pc + imm);
X[rd] = retrun_addr;
```

B.193.6. Exceptions

This instruction may result in the following synchronous exceptions:

- InstructionAddressMisaligned

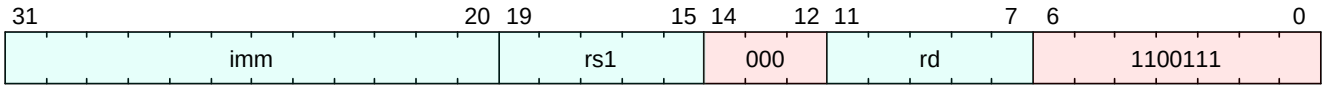
B.194. jalr

Jump and link register

This instruction is defined by:

- I, version >= 0

B.194.1. Encoding



B.194.2. Synopsis

Jump to an address formed by adding rs1 to a signed offset, and store the return address in rd.

B.194.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.194.4. Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.194.5. Execution

```
XReg returnaddr;
returnaddr = $pc + 4;
jump(X[rs1] + imm);
X[rd] = returnaddr;
```

B.194.6. Exceptions

This instruction may result in the following synchronous exceptions:

- InstructionAddressMisaligned

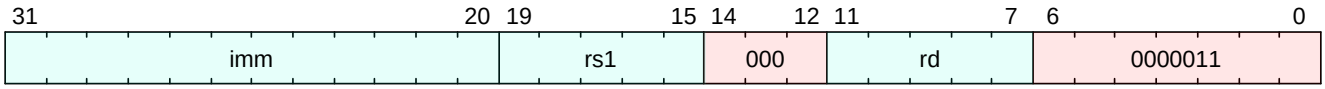
B.195. lb

Load byte

This instruction is defined by:

- I, version >= 0

B.195.1. Encoding



B.195.2. Synopsis

Load 8 bits of data into register `rd` from an address formed by adding `rs1` to a signed offset. Sign extend the result.

B.195.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.195.4. Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.195.5. Execution

```
XReg virtual_address = X[rs1] + imm;
X[rd] = sext(read_memory<8>(virtual_address, $encoding), 8);
```

B.195.6. Exceptions

This instruction may result in the following synchronous exceptions:

- LoadAccessFault
- LoadAddressMisaligned
- LoadPageFault

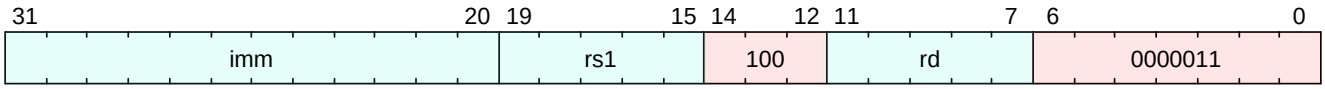
B.196. lbu

Load byte unsigned

This instruction is defined by:

- I, version >= 0

B.196.1. Encoding



B.196.2. Synopsis

Load 8 bits of data into register `rd` from an address formed by adding `rs1` to a signed offset. Zero extend the result.

B.196.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.196.4. Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.196.5. Execution

```
XReg virtual_address = X[rs1] + imm;
X[rd] = read_memory<8>(virtual_address, $encoding);
```

B.196.6. Exceptions

This instruction may result in the following synchronous exceptions:

- LoadAccessFault
- LoadAddressMisaligned
- LoadPageFault

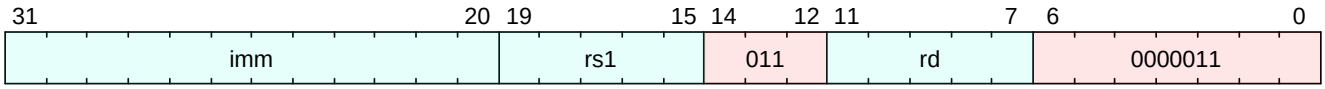
B.197. ld

Load doubleword

This instruction is defined by:

- I, version >= 0

B.197.1. Encoding



B.197.2. Synopsis

Load 64 bits of data into register `rd` from an address formed by adding `rs1` to a signed offset.

B.197.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.197.4. Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.197.5. Execution

```
XReg virtual_address = X[rs1] + imm;
X[rd] = read_memory<64>(virtual_address, $encoding);
```

B.197.6. Exceptions

This instruction may result in the following synchronous exceptions:

- LoadAccessFault
- LoadAddressMisaligned
- LoadPageFault

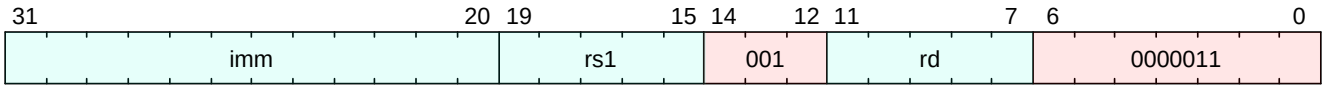
B.198. lh

Load halfword

This instruction is defined by:

- I, version >= 0

B.198.1. Encoding



B.198.2. Synopsis

Load 16 bits of data into register `rd` from an address formed by adding `rs1` to a signed offset. Sign extend the result.

B.198.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.198.4. Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.198.5. Execution

```
XReg virtual_address = X[rs1] + imm;
X[rd] = sext(read_memory<16>(virtual_address, $encoding), 16);
```

B.198.6. Exceptions

This instruction may result in the following synchronous exceptions:

- LoadAccessFault
- LoadAddressMisaligned
- LoadPageFault

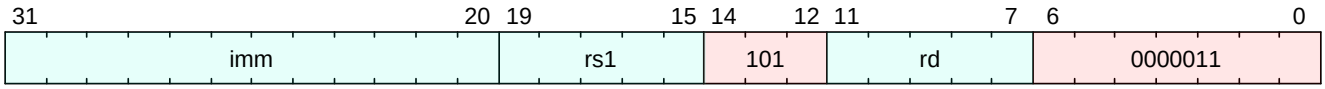
B.199. lhu

Load halfword unsigned

This instruction is defined by:

- I, version >= 0

B.199.1. Encoding



B.199.2. Synopsis

Load 16 bits of data into register `rd` from an address formed by adding `rs1` to a signed offset. Zero extend the result.

B.199.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.199.4. Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.199.5. Execution

```
XReg virtual_address = X[rs1] + imm;
X[rd] = read_memory<16>(virtual_address, $encoding);
```

B.199.6. Exceptions

This instruction may result in the following synchronous exceptions:

- LoadAccessFault
- LoadAddressMisaligned
- LoadPageFault

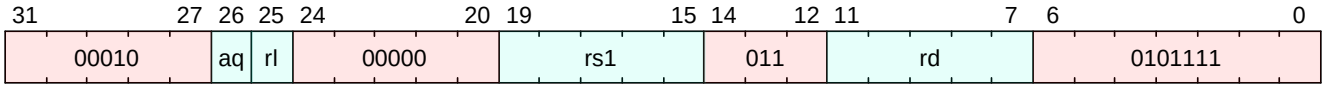
B.200. lr.d

Load reserved doubleword

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zalrsc, version >= 0

B.200.1. Encoding



B.200.2. Synopsis

Loads a word from the address in rs1, places the value in rd, and registers a *reservation set* — a set of bytes that subsumes the bytes in the addressed word.

The address in rs1 must be 8-byte aligned.

If the address is not naturally aligned, a `LoadAddressMisaligned` exception or an `LoadAccessFault` exception will be generated. The access-fault exception can be generated for a memory access that would otherwise be able to complete except for the misalignment, if the misaligned access should not be emulated.

An implementation can register an arbitrarily large reservation set on each LR, provided the reservation set includes all bytes of the addressed data word or doubleword. An SC can only pair with the most recent LR in program order. An SC may succeed only if no store from another hart to the reservation set can be observed to have occurred between the LR and the SC, and if there is no other SC between the LR and itself in program order. An SC may succeed only if no write from a device other than a hart to the bytes accessed by the LR instruction can be observed to have occurred between the LR and SC. Note this LR might have had a different effective address and data size, but reserved the SC's address as part of the reservation set.

```
Following this model, in systems with memory translation, an SC is allowed to succeed
if the
earlier LR reserved the same location using an alias with a different virtual address,
but is
also allowed to fail if the virtual address is different.

To accommodate legacy devices and buses, writes from devices other than RISC-V harts
are only
required to invalidate reservations when they overlap the bytes accessed by the LR.
These writes are not required to invalidate the reservation when they access other
bytes in
```

```
the reservation set.
```

Software should not set the *rl* bit on an LR instruction unless the *aq* bit is also set. LR.rl and SC.aq instructions are not guaranteed to provide any stronger ordering than those with both bits clear, but may result in lower performance.

B.200.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.200.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.200.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
if (!is_naturally_aligned<64>(virtual_address)) {
  if (LRSC_MISALIGNED_BEHAVIOR == "always raise misaligned exception") {
    raise(ExceptionCode::LoadAddressMisaligned, effective_ldst_mode(),
virtual_address);
  } else if (LRSC_MISALIGNED_BEHAVIOR == "always raise access fault") {
    raise(ExceptionCode::LoadAccessFault, effective_ldst_mode(), virtual_address);
  } else {
    unpredictable("Implementations may raise either a LoadAddressMisaligned or a
LoadAccessFault when an LR/SC address is misaligned");
  }
}
X[rd] = load_reserved<32>(virtual_address, aq, rl, $encoding);
```

B.200.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- LoadAddressMisaligned
- LoadPageFault

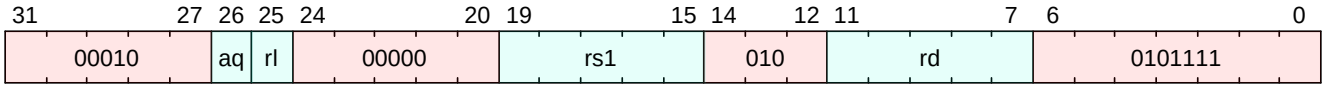
B.201. lr.w

Load reserved word

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zalrsc, version >= 0

B.201.1. Encoding



B.201.2. Synopsis

Loads a word from the address in rs1, places the sign-extended value in rd, and registers a *reservation set* — a set of bytes that subsumes the bytes in the addressed word.

<%- if XLEN == 64 -%> The 32-bit load result is sign-extended to 64-bits. <%- end -%>

The address in rs1 must be naturally aligned to the size of the operand (*i.e.*, eight-byte aligned for doublewords and four-byte aligned for words).

If the address is not naturally aligned, a `LoadAddressMisaligned` exception or an `LoadAccessFault` exception will be generated. The access-fault exception can be generated for a memory access that would otherwise be able to complete except for the misalignment, if the misaligned access should not be emulated.

An implementation can register an arbitrarily large reservation set on each LR, provided the reservation set includes all bytes of the addressed data word or doubleword. An SC can only pair with the most recent LR in program order. An SC may succeed only if no store from another hart to the reservation set can be observed to have occurred between the LR and the SC, and if there is no other SC between the LR and itself in program order. An SC may succeed only if no write from a device other than a hart to the bytes accessed by the LR instruction can be observed to have occurred between the LR and SC. Note this LR might have had a different effective address and data size, but reserved the SC's address as part of the reservation set.

```
Following this model, in systems with memory translation, an SC is allowed to succeed
if the
earlier LR reserved the same location using an alias with a different virtual address,
but is
also allowed to fail if the virtual address is different.

To accommodate legacy devices and buses, writes from devices other than RISC-V harts
are only
required to invalidate reservations when they overlap the bytes accessed by the LR.
```

```
These writes are not required to invalidate the reservation when they access other
bytes in
the reservation set.
```

Software should not set the *rl* bit on an LR instruction unless the *aq* bit is also set. LR.rl and SC.aq instructions are not guaranteed to provide any stronger ordering than those with both bits clear, but may result in lower performance.

B.201.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.201.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.201.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
if (!is_naturally_aligned<32>(virtual_address)) {
  if (LRSC_MISALIGNED_BEHAVIOR == "always raise misaligned exception") {
    raise(ExceptionCode::LoadAddressMisaligned, effective_ldst_mode(),
virtual_address);
  } else if (LRSC_MISALIGNED_BEHAVIOR == "always raise access fault") {
    raise(ExceptionCode::LoadAccessFault, effective_ldst_mode(), virtual_address);
  } else {
    unpredictable("Implementations may raise either a LoadAddressMisaligned or a
LoadAccessFault when an LR/SC address is misaligned");
  }
}
XReg load_value = load_reserved<32>(virtual_address, aq, rl, $encoding);
if (xlen() == 64) {
  X[rd] = load_value;
} else {
  X[rd] = sext(load_value[31:0], 32);
}
```

B.201.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- LoadAddressMisaligned
- LoadPageFault

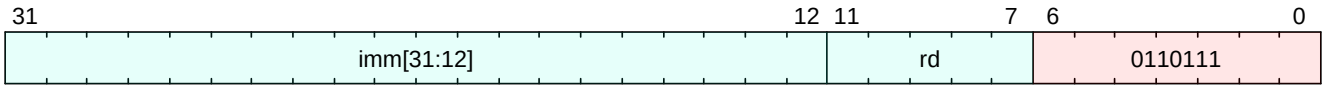
B.202. lui

Load upper immediate

This instruction is defined by:

- I, version >= 0

B.202.1. Encoding



B.202.2. Synopsis

Load the zero-extended imm into rd.

B.202.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.202.4. Decode Variables

```
Bits<32> imm = {$encoding[31:12], 12'd0};
Bits<5> rd = $encoding[11:7];
```

B.202.5. Execution

```
X[rd] = imm;
```

B.202.6. Exceptions

This instruction does not generate synchronous exceptions.

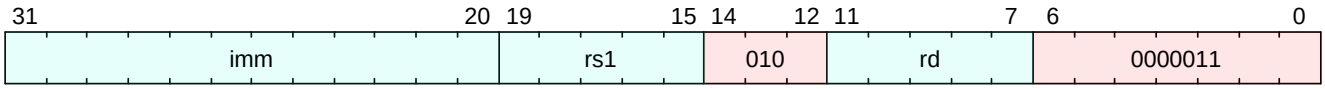
B.203. lw

Load word

This instruction is defined by:

- I, version >= 0

B.203.1. Encoding



B.203.2. Synopsis

Load 32 bits of data into register `rd` from an address formed by adding `rs1` to a signed offset. Sign extend the result.

B.203.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.203.4. Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.203.5. Execution

```
XReg virtual_address = X[rs1] + imm;
X[rd] = read_memory<32>(virtual_address, $encoding);
```

B.203.6. Exceptions

This instruction may result in the following synchronous exceptions:

- LoadAccessFault
- LoadAddressMisaligned
- LoadPageFault

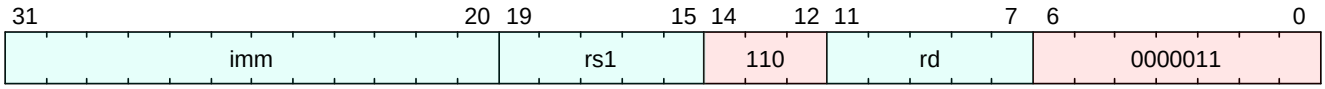
B.204. lwu

Load word unsigned

This instruction is defined by:

- I, version >= 0

B.204.1. Encoding



B.204.2. Synopsis

Load 64 bits of data into register `rd` from an address formed by adding `rs1` to a signed offset. Zero extend the result.

B.204.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.204.4. Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.204.5. Execution

```
XReg virtual_address = X[rs1] + imm;
X[rd] = read_memory<32>(virtual_address, $encoding);
```

B.204.6. Exceptions

This instruction may result in the following synchronous exceptions:

- LoadAccessFault
- LoadAddressMisaligned
- LoadPageFault

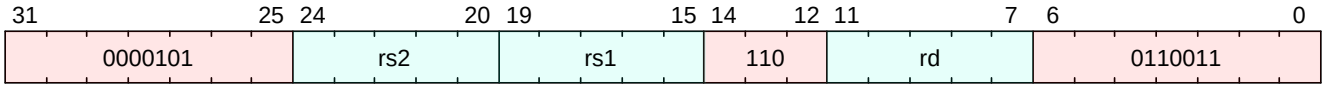
B.205. max

Maximum

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0

B.205.1. Encoding



B.205.2. Synopsis

This instruction returns the larger of two signed integers.

Software Hint

Calculating the absolute value of a signed integer can be performed using the following sequence: `neg rD,rS` followed by `max rD,rS,rD`. When using this common sequence, it is suggested that they are scheduled with no intervening instructions so that implementations that are so optimized can fuse them together.

B.205.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.205.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.205.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = ($signed(X[rs1]) > $signed(X[rs2])) ? X[rs1] : X[rs2];
```

B.205.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

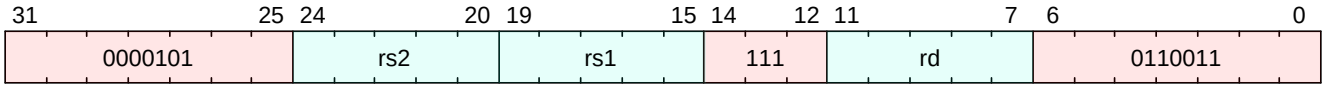
B.206. maxu

Unsigned maximum

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0

B.206.1. Encoding



B.206.2. Synopsis

This instruction returns the larger of two unsigned integers.

B.206.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.206.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.206.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = (X[rs1] > X[rs2]) ? X[rs1] : X[rs2];
```

B.206.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

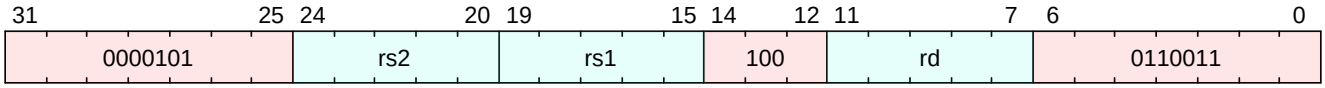
B.207. min

Minimum

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0

B.207.1. Encoding



B.207.2. Synopsis

This instruction returns the smaller of two signed integers.

B.207.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.207.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.207.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = ($signed(X[rs1]) < $signed(X[rs2])) ? X[rs1] : X[rs2];
```

B.207.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

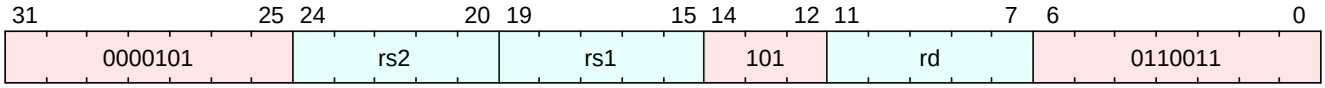
B.208. minu

Unsigned minumum

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0

B.208.1. Encoding



B.208.2. Synopsis

This instruction returns the smaller of two unsigned integers.

B.208.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.208.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.208.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = (X[rs1] < X[rs2]) ? X[rs1] : X[rs2];
```

B.208.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

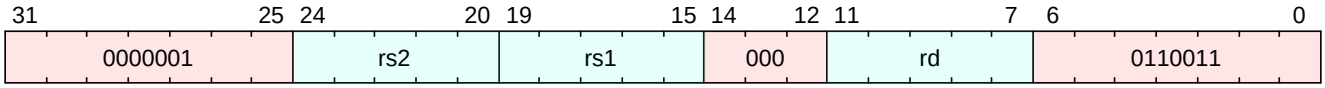
B.209. mul

Signed multiply

This instruction is defined by:

- anyOf:
  - M, version >= 0
  - Zmmul, version >= 0

B.209.1. Encoding



B.209.2. Synopsis

MUL performs an XLEN-bitxXLEN-bit multiplication of `rs1` by `rs2` and places the lower XLEN bits in the destination register. Any overflow is thrown away.

If both the high and low bits of the same product are required, then the recommended code sequence is: MULH[[S]U] rdh, rs1, rs2; MUL rdl, rs1, rs2 (source register specifiers must be in same order and rdh cannot be the same as rs1 or rs2). Microarchitectures can then fuse these into a single multiply operation instead of performing two separate multiplies.

B.209.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.209.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.209.5. Execution

```
if (implemented?(ExtensionName::M) && (CSR[misa].M == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg src1 = X[rs1];
XReg src2 = X[rs2];
```

```
X[rd] = (src1 * src2)[XLEN - 1:0];
```

B.209.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

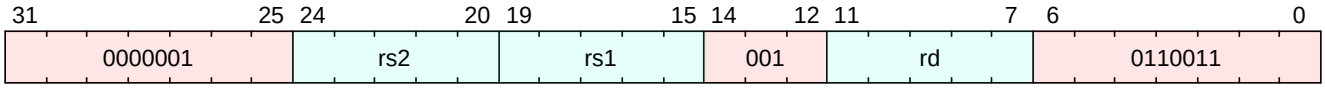
B.210. mulh

Signed multiply high

This instruction is defined by:

- anyOf:
  - M, version >= 0
  - Zmmul, version >= 0

B.210.1. Encoding



B.210.2. Synopsis

Multiply the signed values in rs1 to rs2, and store the upper half of the result in rd. The lower half is thrown away.

If both the upper and lower halves are needed, it suggested to use the sequence:

```
  mulh rdh, rs1, rs2
  mul  rdl, rs1, rs2
---
```

Microarchitectures may look for that sequence and fuse the operations.

B.210.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.210.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.210.5. Execution

```
if (implemented?(ExtensionName::M) && (CSR[misa].M == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
```

```
}
Bits<1> rs1_sign_bit = X[rs1][xlen() - 1];
Bits<XLEN * 2> src1 = {{xlen(){rs1_sign_bit}}, X[rs1]};
Bits<1> rs2_sign_bit = X[rs2][xlen() - 1];
Bits<XLEN * 2> src2 = {{xlen(){rs2_sign_bit}}, X[rs2]};
X[rd] = (src1 * src2)[(xlen() * 8'd2) - 1:xlen()];
```

B.210.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

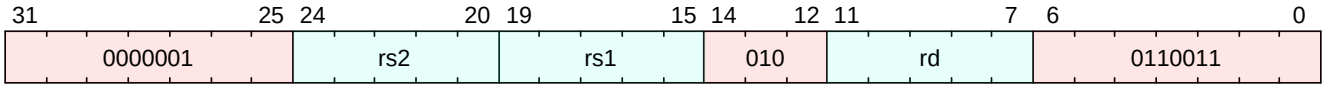
B.211. mulhsu

Signed/unsigned multiply high

This instruction is defined by:

- anyOf:
  - M, version >= 0
  - Zmmul, version >= 0

B.211.1. Encoding



B.211.2. Synopsis

Multiply the signed value in rs1 by the unsigned value in rs2, and store the upper half of the result in rd. The lower half is thrown away.

If both the upper and lower halves are needed, it suggested to use the sequence:

```
  mulhsu rdh, rs1, rs2
  mul    rdl, rs1, rs2
---
```

Microarchitectures may look for that sequence and fuse the operations.

B.211.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.211.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.211.5. Execution

```
if (implemented?(ExtensionName::M) && (CSR[misa].M == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
```

```
}
Bits<1> rs1_sign_bit = X[rs1][XLEN - 1];
Bits<XLEN * 8'd2> src1 = {{XLEN{rs1_sign_bit}}, X[rs1]};
Bits<XLEN * 8'd2> src2 = {{XLEN{1'b0}}, X[rs2]};
X[rd] = (src1 * src2)[(XLEN * 8'd2) - 1:XLEN];
```

B.211.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

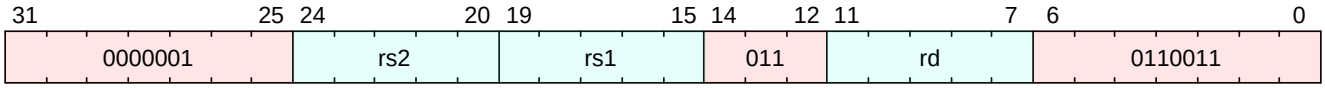
B.212. mulhu

Unsigned multiply high

This instruction is defined by:

- anyOf:
  - M, version >= 0
  - Zmmul, version >= 0

B.212.1. Encoding



B.212.2. Synopsis

Multiply the unsigned values in rs1 to rs2, and store the upper half of the result in rd. The lower half is thrown away.

If both the upper and lower halves are needed, it suggested to use the sequence:

```
  mulhu rdh, rs1, rs2
  mul   rdl, rs1, rs2
---
```

Microarchitectures may look for that sequence and fuse the operations.

B.212.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.212.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.212.5. Execution

```
if (implemented?(ExtensionName::M) && (CSR[misa].M == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
```

```
}
Bits<XLEN * 8'd2> src1 = {{XLEN{1'b0}}, X[rs1]};
Bits<XLEN * 8'd2> src2 = {{XLEN{1'b0}}, X[rs2]};
X[rd] = (src1 * src2)[(XLEN * 8'd2) - 1:XLEN];
```

B.212.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

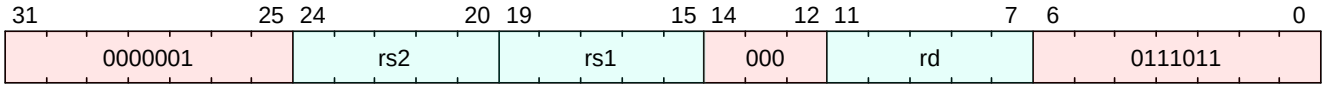
B.213. mulw

Signed 32-bit multiply

This instruction is defined by:

- anyOf:
  - M, version >= 0
  - Zmmul, version >= 0

B.213.1. Encoding



B.213.2. Synopsis

Multiplies the lower 32 bits of the source registers, placing the sign-extension of the lower 32 bits of the result into the destination register.

Any overflow is thrown away.

In RV64, MUL can be used to obtain the upper 32 bits of the 64-bit product, but signed arguments must be proper 32-bit signed values, whereas unsigned arguments must have their upper 32 bits clear. If the arguments are not known to be sign- or zero-extended, an alternative is to shift both arguments left by 32 bits, then use MULH[[S]U].

B.213.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.213.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.213.5. Execution

```
if (implemented?(ExtensionName::M) && (CSR[misa].M == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
Bits<32> src1 = X[rs1][31:0];
```

```
Bits<32> src2 = X[rs2][31:0];
Bits<32> result = src1 * src2;
Bits<1> sign_bit = result[31];
X[rd] = {{32{sign_bit}}, result};
```

B.213.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

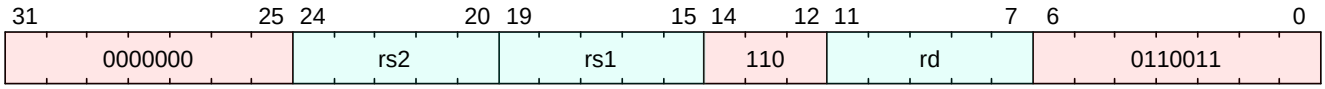
B.214. or

Or

This instruction is defined by:

- I, version >= 0

B.214.1. Encoding



B.214.2. Synopsis

Or rs1 with rs2, and store the result in rd

B.214.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.214.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.214.5. Execution

```
X[rd] = X[rs1] | X[rs2];
```

B.214.6. Exceptions

This instruction does not generate synchronous exceptions.

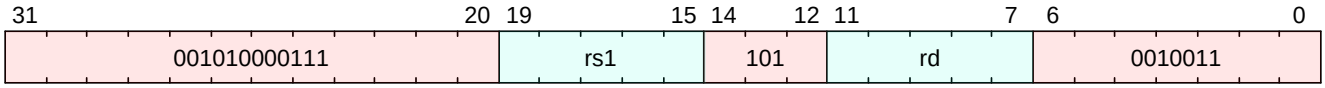
B.215. orc.b

Bitware OR-combine, byte granule

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0

B.215.1. Encoding



B.215.2. Synopsis

Combines the bits within each byte using bitwise logical OR. This sets the bits of each byte in the result rd to all zeros if no bit within the respective byte of rs is set, or to all ones if any bit within the respective byte of rs is set.

B.215.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.215.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.215.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg input = X[rs1];
XReg output = 0;
for (U32 i = 0; i < (xlen() - 8); i = i + 8) {
  output[(i + 7):i] = (input[(i + 7):i] == 0) ? 8'd0 : ~8'd0;
}
X[rd] = output;
```

B.215.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

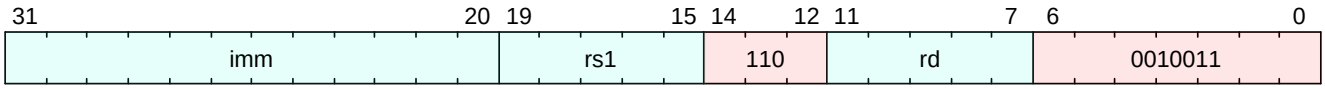
B.216. ori

Or immediate

This instruction is defined by:

- I, version >= 0

B.216.1. Encoding



B.216.2. Synopsis

Or an immediate to the value in rs1, and store the result in rd

B.216.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.216.4. Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.216.5. Execution

```
if (implemented?(ExtensionName::Zicbop)) {
  if (rd == 0) {
    if (imm[4:0] == 0) {
      Bits<12> offset = {imm[11:5], rd};
      prefetch_instruction(offset);
    } else if (imm[4:0] == 1) {
      Bits<12> offset = {imm[11:5], rd};
      prefetch_read(offset);
    } else if (imm[4:0] == 3) {
      Bits<12> offset = {imm[11:5], rd};
      prefetch_write(offset);
    }
  }
}
X[rd] = X[rs1] | imm;
```

B.216.6. Exceptions

This instruction does not generate synchronous exceptions.

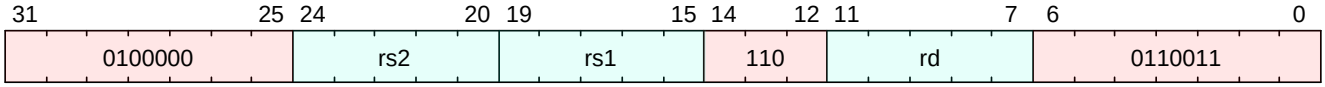
B.217. orn

OR with inverted operand

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0
  - Zbkb, version >= 0
  - Zk, version >= 0
  - Zkn, version >= 0
  - Zks, version >= 0

B.217.1. Encoding



B.217.2. Synopsis

This instruction performs the bitwise logical OR operation between rs1 and the bitwise inversion of rs2.

B.217.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.217.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.217.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = X[rs1] | ~X[rs2];
```

B.217.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

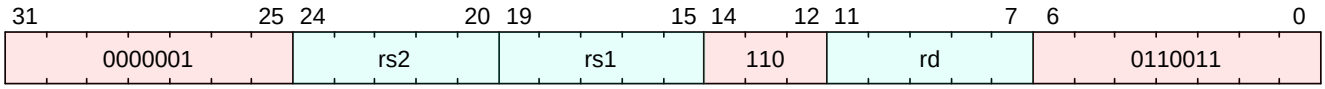
B.218. rem

Signed remainder

This instruction is defined by:

- M, version >= 0

B.218.1. Encoding



B.218.2. Synopsis

Calculate the remainder of signed division of rs1 by rs2, and store the result in rd.

If the value in register rs2 is zero, write the value in rs1 into rd;

If the result of the division overflows, write zero into rd;

B.218.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.218.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.218.5. Execution

```
if (implemented?(ExtensionName::M) && (CSR[misa].M == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg src1 = X[rs1];
XReg src2 = X[rs2];
if (src2 == 0) {
  X[rd] = src1;
} else if ((src1 == {1'b1, {XLEN - 1{1'b0}}}) && (src2 == {XLEN{1'b1}})) {
  X[rd] = 0;
} else {
  X[rd] = $signed(src1) % $signed(src2);
}
```

B.218.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

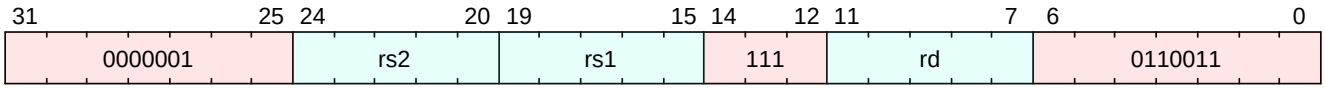
B.219. remu

Unsigned remainder

This instruction is defined by:

- M, version >= 0

B.219.1. Encoding



B.219.2. Synopsis

Calculate the remainder of unsigned division of rs1 by rs2, and store the result in rd.

B.219.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.219.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.219.5. Execution

```
if (implemented?(ExtensionName::M) && (CSR[misa].M == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg src1 = X[rs1];
XReg src2 = X[rs2];
if (src2 == 0) {
  X[rd] = src1;
} else {
  X[rd] = src1 % src2;
}
```

B.219.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

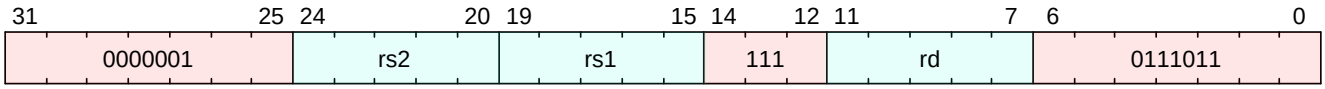
B.220. remuw

Unsigned 32-bit remainder

This instruction is defined by:

- M, version >= 0

B.220.1. Encoding



B.220.2. Synopsis

Calculate the remainder of unsigned division of the 32-bit values in rs1 by rs2, and store the sign-extended result in rd.

If the value in rs2 is zero, rd gets the sign-extended value in rs1.

B.220.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.220.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.220.5. Execution

```
if (implemented?(ExtensionName::M) && (CSR[misa].M == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
Bits<32> src1 = X[rs1][31:0];
Bits<32> src2 = X[rs2][31:0];
if (src2 == 0) {
  Bits<1> sign_bit = src1[31];
  X[rd] = {{32{sign_bit}}, src1};
} else {
  Bits<32> result = src1 % src2;
  Bits<1> sign_bit = result[31];
  X[rd] = {{32{sign_bit}}, result};
}
```

B.220.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

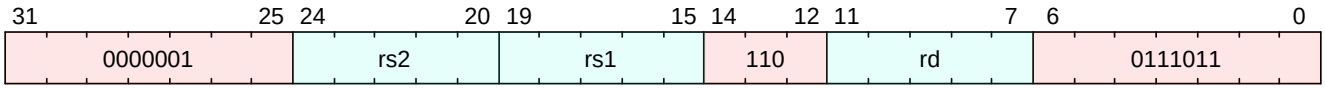
B.221. remw

Signed 32-bit remainder

This instruction is defined by:

- M, version >= 0

B.221.1. Encoding



B.221.2. Synopsis

Calculate the remainder of signed division of the 32-bit values rs1 by rs2, and store the sign-extended result in rd.

If the value in register rs2 is zero, write the sign-extended 32-bit value in rs1 into rd;

If the result of the division overflows, write zero into rd;

B.221.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.221.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.221.5. Execution

```
if (implemented?(ExtensionName::M) && (CSR[misa].M == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
Bits<32> src1 = X[rs1][31:0];
Bits<32> src2 = X[rs2][31:0];
if (src2 == 0) {
  Bits<1> sign_bit = src1[31];
  X[rd] = {{32{sign_bit}}, src1};
} else if ((src1 == {33'b1, 31'b0}) && (src2 == 32'b1)) {
  X[rd] = 0;
} else {
  Bits<32> result = $signed(src1) % $signed(src2);
```

```
  Bits<1> sign_bit = result[31];
  X[rd] = {{32{sign_bit}}, result};
}
```

B.221.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

B.222. rev8

Byte-reverse register (RV64 encoding)

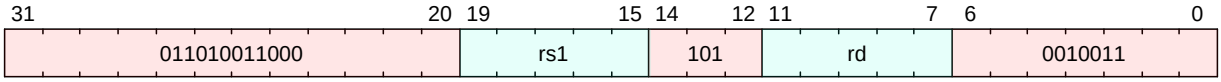
This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0
  - Zbkb, version >= 0

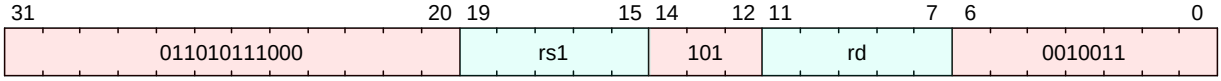
B.222.1. Encoding

This instruction has different encodings in RV32 and RV64.

RV32



RV64



B.222.2. Synopsis

This instruction reverses the order of the bytes in rs1.

The rev8 mnemonic corresponds to different instruction encodings in RV32 and RV64.

The byte-reverse operation is only available for the full register width. To emulate word-sized and halfword-sized byte-reversal, perform a `rev8 rd,rs` followed by a `srai rd,rd,K`, where K is XLEN-32 and XLEN-16, respectively.

B.222.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.222.4. Decode Variables

RV32

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

RV64

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.222.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg input = X[rs1];
XReg output = 0;
XReg j = xlen() - 1;
for (U32 i = 0; i < (xlen() - 8); i = i + 8) {
  output[(i + 7):i] = input[j:(j - 7)];
  j = j - 8;
}
X[rd] = output;
```

B.222.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

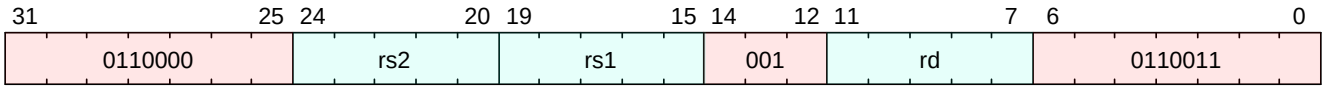
B.223. rol

Rotate left (Register)

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0
  - Zbkb, version >= 0
  - Zk, version >= 0
  - Zkn, version >= 0
  - Zks, version >= 0

B.223.1. Encoding



B.223.2. Synopsis

This instruction performs a rotate left of rs1 by the amount in least-significant `log2(XLEN)` bits of rs2.

B.223.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.223.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.223.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg shamt = (xlen() == 32) ? X[rs2][4:0] : X[rs2][5:0];
X[rd] = (X[rs1] << shamt) | (X[rs1] >> (xlen() - shamt));
```

B.223.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

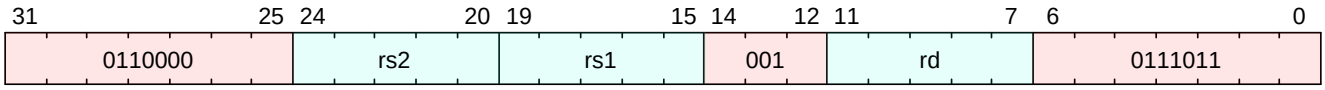
B.224. rolw

Rotate left word (Register)

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0
  - Zbkb, version >= 0
  - Zk, version >= 0
  - Zkn, version >= 0
  - Zks, version >= 0

B.224.1. Encoding



B.224.2. Synopsis

This instruction performs a rotate left of the least-significant word of rs1 by the amount in least-significant 5 bits of rs2. The resulting word value is sign-extended by copying bit 31 to all of the more-significant bits.

B.224.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.224.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.224.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg rs1_word = X[rs1][31:0];
XReg shamt = X[rs2][4:0];
XReg unextended_result = (rs1_word << shamt) | (rs1_word >> (32 - shamt));
```

```
X[rd] = {{32{unextended_result[31]}}, unextended_result};
```

B.224.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

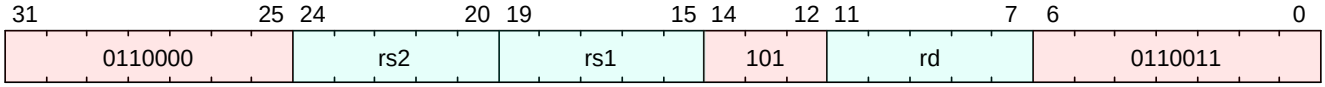
B.225. ror

Rotate right (Register)

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0
  - Zbkb, version >= 0
  - Zk, version >= 0
  - Zkn, version >= 0
  - Zks, version >= 0

B.225.1. Encoding



B.225.2. Synopsis

This instruction performs a rotate right of rs1 by the amount in least-significant `log2(XLEN)` bits of rs2.

B.225.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.225.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.225.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg shamt = (xlen() == 32) ? X[rs2][4:0] : X[rs2][5:0];
X[rd] = (X[rs1] >> shamt) | (X[rs1] << (xlen() - shamt));
```

B.225.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

B.226. rori

Rotate right (Immediate)

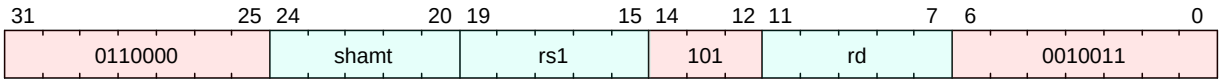
This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0
  - Zbkb, version >= 0
  - Zk, version >= 0
  - Zkn, version >= 0
  - Zks, version >= 0

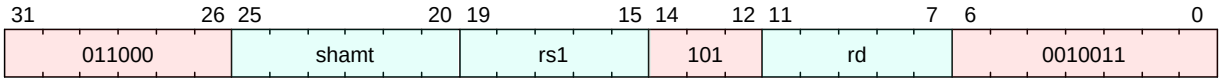
B.226.1. Encoding

This instruction has different encodings in RV32 and RV64.

RV32



RV64



B.226.2. Synopsis

This instruction performs a rotate right of rs1 by the amount in the least-significant log2(XLEN) bits of shamt. For RV32, the encodings corresponding to shamt[5]=1 are reserved.

B.226.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.226.4. Decode Variables

RV32

```
Bits<5> shamt = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

```
Bits<5> rd = $encoding[11:7];
```

RV64

```
Bits<6> shamt = $encoding[25:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.226.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg shamt = (xlen() == 32) ? shamt[4:0] : shamt[5:0];
X[rd] = (X[rs1] >> shamt) | (X[rs1] << (xlen() - shamt));
```

B.226.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

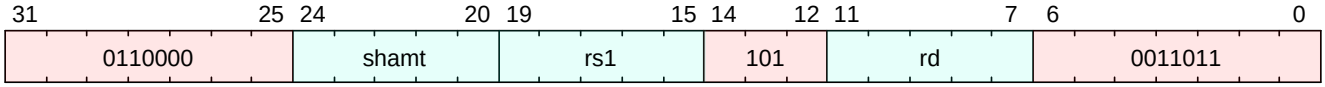
B.227. roriw

Rotate right word (Immediate)

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0
  - Zbkb, version >= 0
  - Zk, version >= 0
  - Zkn, version >= 0
  - Zks, version >= 0

B.227.1. Encoding



B.227.2. Synopsis

This instruction performs a rotate right on the least-significant word of rs1 by the amount in the least-significant log2(XLEN) bits of shamt. The resulting word value is sign-extended by copying bit 31 to all of the more-significant bits.

B.227.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.227.4. Decode Variables

```
Bits<5> shamt = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.227.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg rs1_word = X[rs1][31:0];
XReg unextended_result = (X[rs1] >> shamt) | (X[rs1] << (32 - shamt));
```

```
X[rd] = {{32{unextended_result[31]}}, unextended_result};
```

B.227.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

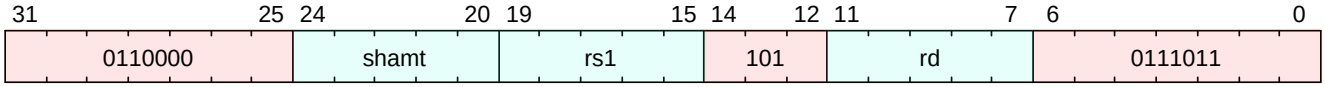
B.228. rorw

Rotate right word (Register)

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0
  - Zbkb, version >= 0
  - Zk, version >= 0
  - Zkn, version >= 0
  - Zks, version >= 0

B.228.1. Encoding



B.228.2. Synopsis

This instruction performs a rotate right on the least-significant word of rs1 by the amount in least-significant 5 bits of rs2. The resultant word is sign-extended by copying bit 31 to all of the more-significant bits.

B.228.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.228.4. Decode Variables

```
Bits<5> shamt = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.228.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg rs1_word = X[rs1][31:0];
XReg shamt = X[rs1][4:0];
XReg unextended_result = (X[rs1] >> shamt) | (X[rs1] << (32 - shamt));
```

```
X[rd] = {{32{unextended_result[31]}}, unextended_result};
```

B.228.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

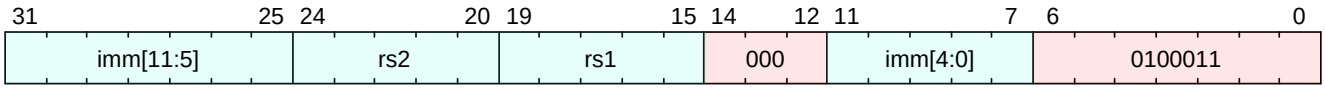
B.229. sb

Store byte

This instruction is defined by:

- I, version >= 0

B.229.1. Encoding



B.229.2. Synopsis

Store 8 bits of data from register `rs2` to an address formed by adding `rs1` to a signed offset.

B.229.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.229.4. Decode Variables

```
Bits<12> imm = {$encoding[31:25], $encoding[11:7]};
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

B.229.5. Execution

```
XReg virtual_address = X[rs1] + imm;
write_memory<8>(virtual_address, X[rs2][7:0], $encoding);
```

B.229.6. Exceptions

This instruction may result in the following synchronous exceptions:

- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

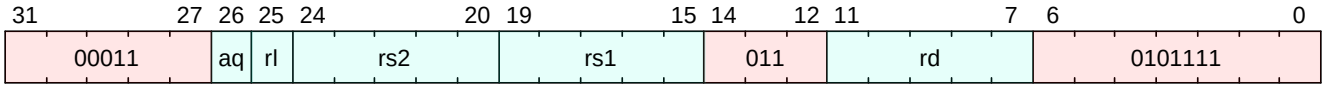
B.230. sc.d

Store conditional doubleword

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zalrsc, version >= 0

B.230.1. Encoding



B.230.2. Synopsis

sc.d conditionally writes a doubleword in *rs2* to the address in *rs1*: the sc.d succeeds only if the reservation is still valid and the reservation set contains the bytes being written. If the sc.d succeeds, the instruction writes the doubleword in *rs2* to memory, and it writes zero to *rd*. If the sc.d fails, the instruction does not write to memory, and it writes a nonzero value to *rd*. For the purposes of memory protection, a failed sc.d may be treated like a store. Regardless of success or failure, executing an sc.d instruction invalidates any reservation held by this hart.

The failure code with value 1 encodes an unspecified failure. Other failure codes are reserved at this time. Portable software should only assume the failure code will be non-zero.

The address held in *rs1* must be naturally aligned to the size of the operand (*i.e.*, eight-byte aligned). If the address is not naturally aligned, an address-misaligned exception or an access-fault exception will be generated. The access-fault exception can be generated for a memory access that would otherwise be able to complete except for the misalignment, if the misaligned access should not be emulated.

Emulating misaligned LR/SC sequences is impractical in most systems.

Misaligned LR/SC sequences also raise the possibility of accessing multiple reservation sets at once, which present definitions do not provide for.

An implementation can register an arbitrarily large reservation set on each LR, provided the reservation set includes all bytes of the addressed data word or doubleword. An SC can only pair with the most recent LR in program order. An SC may succeed only if no store from another hart to the reservation set can be observed to have occurred between the LR and the SC, and if there is no other SC between the LR and itself in program order. An SC may succeed only if no write from a device other than a hart to the bytes accessed by the LR instruction can be observed to have occurred between the LR and SC. Note this LR might have had a different effective address and data size, but reserved the SC's address as part of the reservation set.

```
Following this model, in systems with memory translation, an SC is allowed to succeed
if the
earlier LR reserved the same location using an alias with a different virtual address,
but is
also allowed to fail if the virtual address is different.

To accommodate legacy devices and buses, writes from devices other than RISC-V harts
are only
required to invalidate reservations when they overlap the bytes accessed by the LR.
These writes are not required to invalidate the reservation when they access other
bytes in
the reservation set.
```

The SC must fail if the address is not within the reservation set of the most recent LR in program order. The SC must fail if a store to the reservation set from another hart can be observed to occur between the LR and SC. The SC must fail if a write from some other device to the bytes accessed by the LR can be observed to occur between the LR and SC. (If such a device writes the reservation set but does not write the bytes accessed by the LR, the SC may or may not fail.) An SC must fail if there is another SC (to any address) between the LR and the SC in program order. The precise statement of the atomicity requirements for successful LR/SC sequences is defined by the Atomicity Axiom of the memory model.

The platform should provide a means to determine the size and shape of the reservation set.

A platform specification may constrain the size and shape of the reservation set.

A store-conditional instruction to a scratch word of memory should be used to forcibly invalidate any existing load reservation:

- during a preemptive context switch, and
- if necessary when changing virtual to physical address mappings, such as when migrating pages that might contain an active reservation.

The invalidation of a hart's reservation when it executes an LR or SC imply that a hart can only hold one reservation at a time, and that an SC can only pair with the most recent LR, and LR with the next following SC, in program order. This is a restriction to the Atomicity Axiom in Section 18.1 that ensures software runs correctly on expected common implementations that operate in this manner.

An SC instruction can never be observed by another RISC-V hart before the LR instruction that established the reservation.

The LR/SC sequence can be given acquire semantics by setting the aq bit on the LR instruction. The LR/SC sequence can be given release semantics by by setting the rl bit on the SC instruction. Assuming suitable mappings for other atomic operations, setting the aq bit on the LR instruction, and setting the rl bit on the SC instruction makes the LR/SC sequence sequentially consistent in the C memory_order_seq_cst

> sense. Such a sequence does not act as a fence for ordering ordinary load and store instructions before and after the sequence. Specific instruction mappings for other C atomic operations, or stronger notions of "sequential consistency", may require both bits to be set on either or both of the LR or SC instruction.
>
> If neither bit is set on either LR or SC, the LR/SC sequence can be observed to occur before or after surrounding memory operations from the same RISC-V hart. This can be appropriate when the LR/SC sequence is used to implement a parallel reduction operation.

Software should not set the *rl* bit on an LR instruction unless the *aq* bit is also set. LR.rl and SC.aq instructions are not guaranteed to provide any stronger ordering than those with both bits clear, but may result in lower performance.

B.230.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.230.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.230.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
XReg value = X[rs2];
if (!is_naturally_aligned<64>(virtual_address)) {
  if (LRSC_MISALIGNED_BEHAVIOR == "always raise misaligned exception") {
    raise(ExceptionCode::LoadAddressMisaligned, effective_ldst_mode(),
virtual_address);
  } else if (LRSC_MISALIGNED_BEHAVIOR == "always raise access fault") {
    raise(ExceptionCode::LoadAccessFault, effective_ldst_mode(), virtual_address);
  } else {
    unpredictable("Implementations may raise either a LoadAddressMisaligned or a
LoadAccessFault when an LR/SC address is misaligned");
  }
}
Boolean success = store_conditional<64>(virtual_address, value, aq, rl, $encoding);
```

```
X[rd] = success ? 0 : 1;
```

B.230.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- LoadAddressMisaligned
- StoreAmoAccessFault
- StoreAmoPageFault

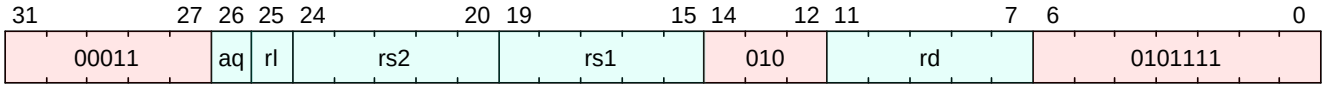
B.231. sc.w

Store conditional word

This instruction is defined by:

- anyOf:
  - A, version >= 0
  - Zalrsc, version >= 0

B.231.1. Encoding



B.231.2. Synopsis

sc.w conditionally writes a word in *rs2* to the address in *rs1*: the sc.w succeeds only if the reservation is still valid and the reservation set contains the bytes being written. If the sc.w succeeds, the instruction writes the word in *rs2* to memory, and it writes zero to *rd*. If the sc.w fails, the instruction does not write to memory, and it writes a nonzero value to *rd*. For the purposes of memory protection, a failed sc.w may be treated like a store. Regardless of success or failure, executing an sc.w instruction invalidates any reservation held by this hart.

<%- if XLEN == 64 -%>

If a value other than 0 or 1 is defined as a result for sc.w, the value will before sign-extended into *rd*. <%- end -%>

The failure code with value 1 encodes an unspecified failure. Other failure codes are reserved at this time. Portable software should only assume the failure code will be non-zero.

The address held in *rs1* must be naturally aligned to the size of the operand (*i.e.*, eight-byte aligned for doublewords and four-byte aligned for words). If the address is not naturally aligned, an address-misaligned exception or an access-fault exception will be generated. The access-fault exception can be generated for a memory access that would otherwise be able to complete except for the misalignment, if the misaligned access should not be emulated.

Emulating misaligned LR/SC sequences is impractical in most systems.

Misaligned LR/SC sequences also raise the possibility of accessing multiple reservation sets at once, which present definitions do not provide for.

An implementation can register an arbitrarily large reservation set on each LR, provided the reservation set includes all bytes of the addressed data word or doubleword. An SC can only pair with the most recent LR in program order. An SC may succeed only if no store from another hart to the reservation set can be observed to have occurred between the LR and the SC, and if there is no other SC between the LR and itself in program order. An SC may succeed only if no write from a

device other than a hart to the bytes accessed by the LR instruction can be observed to have occurred between the LR and SC. Note this LR might have had a different effective address and data size, but reserved the SC's address as part of the reservation set.

```
Following this model, in systems with memory translation, an SC is allowed to succeed
if the
earlier LR reserved the same location using an alias with a different virtual address,
but is
also allowed to fail if the virtual address is different.

To accommodate legacy devices and buses, writes from devices other than RISC-V harts
are only
required to invalidate reservations when they overlap the bytes accessed by the LR.
These writes are not required to invalidate the reservation when they access other
bytes in
the reservation set.
```

The SC must fail if the address is not within the reservation set of the most recent LR in program order. The SC must fail if a store to the reservation set from another hart can be observed to occur between the LR and SC. The SC must fail if a write from some other device to the bytes accessed by the LR can be observed to occur between the LR and SC. (If such a device writes the reservation set but does not write the bytes accessed by the LR, the SC may or may not fail.) An SC must fail if there is another SC (to any address) between the LR and the SC in program order. The precise statement of the atomicity requirements for successful LR/SC sequences is defined by the Atomicity Axiom of the memory model.

The platform should provide a means to determine the size and shape of the reservation set.

A platform specification may constrain the size and shape of the reservation set.

A store-conditional instruction to a scratch word of memory should be used to forcibly invalidate any existing load reservation:

- during a preemptive context switch, and
- if necessary when changing virtual to physical address mappings, such as when migrating pages that might contain an active reservation.

The invalidation of a hart's reservation when it executes an LR or SC imply that a hart can only hold one reservation at a time, and that an SC can only pair with the most recent LR, and LR with the next following SC, in program order. This is a restriction to the Atomicity Axiom in Section 18.1 that ensures software runs correctly on expected common implementations that operate in this manner.

An SC instruction can never be observed by another RISC-V hart before the LR instruction that established the reservation.

The LR/SC sequence can be given acquire semantics by setting the aq bit on the LR

> instruction. The LR/SC sequence can be given release semantics by by setting the rl bit on the SC instruction. Assuming suitable mappings for other atomic operations, setting the aq bit on the LR instruction, and setting the rl bit on the SC instruction makes the LR/SC sequence sequentially consistent in the C memory_order_seq_cst sense. Such a sequence does not act as a fence for ordering ordinary load and store instructions before and after the sequence. Specific instruction mappings for other C atomic operations, or stronger notions of "sequential consistency", may require both bits to be set on either or both of the LR or SC instruction.
>
> If neither bit is set on either LR or SC, the LR/SC sequence can be observed to occur before or after surrounding memory operations from the same RISC-V hart. This can be appropriate when the LR/SC sequence is used to implement a parallel reduction operation.

Software should not set the *rl* bit on an LR instruction unless the *aq* bit is also set. LR.rl and SC.aq instructions are not guaranteed to provide any stronger ordering than those with both bits clear, but may result in lower performance.

B.231.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.231.4. Decode Variables

```
Bits<1> aq = $encoding[26];
Bits<1> rl = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.231.5. Execution

```
if (implemented?(ExtensionName::A) && (CSR[misa].A == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
XReg virtual_address = X[rs1];
XReg value = X[rs2];
if (!is_naturally_aligned<32>(virtual_address)) {
  if (LRSC_MISALIGNED_BEHAVIOR == "always raise misaligned exception") {
    raise(ExceptionCode::LoadAddressMisaligned, effective_ldst_mode(),
virtual_address);
  } else if (LRSC_MISALIGNED_BEHAVIOR == "always raise access fault") {
    raise(ExceptionCode::LoadAccessFault, effective_ldst_mode(), virtual_address);
  } else {
    unpredictable("Implementations may raise either a LoadAddressMisaligned or a
LoadAccessFault when an LR/SC address is misaligned");
```

```
  }
}
Boolean success = store_conditional<32>(virtual_address, value, aq, rl, $encoding);
X[rd] = success ? 0 : 1;
```

B.231.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- LoadAccessFault
- LoadAddressMisaligned
- StoreAmoAccessFault
- StoreAmoPageFault

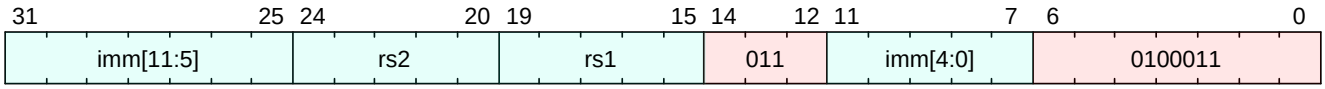
B.232. sd

Store doubleword

This instruction is defined by:

- I, version >= 0

B.232.1. Encoding



B.232.2. Synopsis

Store 64 bits of data from register `rs2` to an address formed by adding `rs1` to a signed offset.

B.232.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.232.4. Decode Variables

```
signed Bits<12> imm = sext({$encoding[31:25], $encoding[11:7]});
Bits<5> rs1 = $encoding[19:15];
Bits<5> rs2 = $encoding[24:20];
```

B.232.5. Execution

```
XReg virtual_address = X[rs1] + imm;
write_memory<64>(virtual_address, X[rs2], $encoding);
```

B.232.6. Exceptions

This instruction may result in the following synchronous exceptions:

- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

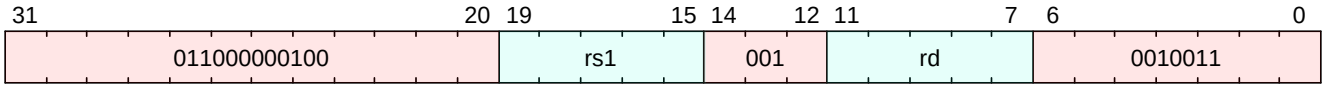
B.233. sext.b

Sign-extend byte

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0

B.233.1. Encoding



B.233.2. Synopsis

This instruction sign-extends the least-significant byte in the source to XLEN by copying the most-significant bit in the byte (i.e., bit 7) to all of the more-significant bits.

B.233.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.233.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.233.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
if (xlen() == 32) {
  X[rd] = {{24{X[rs1][7]}}, X[rs1][7:0]};
} else if (xlen() == 64) {
  X[rd] = {{56{X[rs1][7]}}, X[rs1][7:0]};
}
```

B.233.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

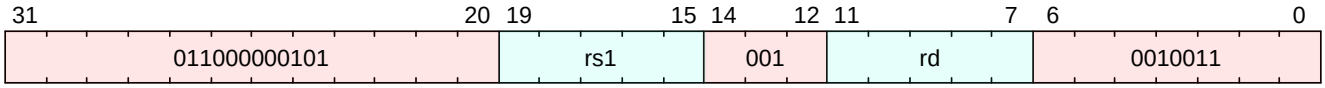
B.234. sext.h

Sign-extend halfword

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0

B.234.1. Encoding



B.234.2. Synopsis

This instruction sign-extends the least-significant halfword in the source to XLEN by copying the most-significant bit in the halfword (i.e., bit 15) to all of the more-significant bits.

B.234.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.234.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.234.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
if (xlen() == 32) {
  X[rd] = {{16{X[rs1][15]}}, X[rs1][15:0]};
} else if (xlen() == 64) {
  X[rd] = {{48{X[rs1][15]}}, X[rs1][15:0]};
}
```

B.234.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

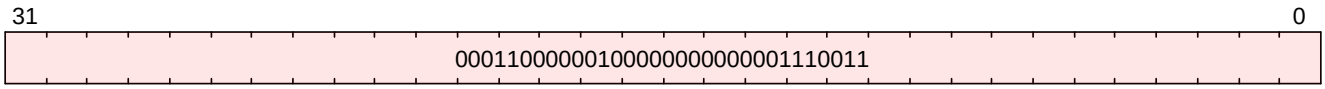
B.235. sfence.inval.ir

Order implicit page table reads after invalidation

This instruction is defined by:

- Svinval, version >= 0

B.235.1. Encoding



B.235.2. Synopsis

The sfence.inval.ir instruction guarantees that any previous sinval.vma instructions executed by the current hart are ordered before subsequent implicit references by that hart to the memory-management data structures.

B.235.3. Access

M	HS	U	VS	VU
Always	Sometimes	Never	Sometimes	Never

B.235.4. Decode Variables

```
```

B.235.5. Execution

```
if (mode() == PrivilegeMode::U) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
if (CSR[misa].H == 1 && mode() == PrivilegeMode::VU) {
  raise(ExceptionCode::VirtualInstruction, mode(), $encoding);
}
VmaOrderType vma_type;
vma_type.global = true;
vma_type.smode = true;
if (CSR[misa].H == 1) {
  vma_type.vsmode = true;
  vma_type.gstage = true;
}
order_pgtbl_reads_after_vmafence(vma_type);
```

B.235.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- VirtualInstruction

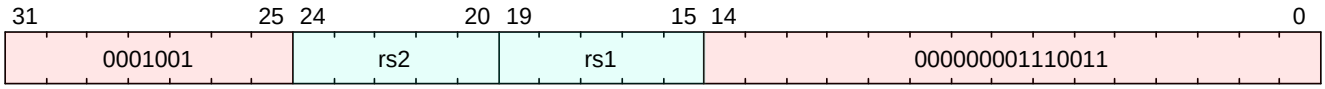
B.236. sfence.vma

Supervisor memory-management fence

This instruction is defined by:

- S, version >= 0

B.236.1. Encoding



B.236.2. Synopsis

The supervisor memory-management fence instruction `SFENCE.VMA` is used to synchronize updates to in-memory memory-management data structures with current execution. Instruction execution causes implicit reads and writes to these data structures; however, these implicit references are ordinarily not ordered with respect to explicit loads and stores. Executing an SFENCE.VMA instruction guarantees that any previous stores already visible to the current RISC-V hart are ordered before certain implicit references by subsequent instructions in that hart to the memory-management data structures. The specific set of operations ordered by SFENCE.VMA is determined by *rs1* and *rs2*, as described below. SFENCE.VMA is also used to invalidate entries in the address-translation cache associated with a hart (see [sv32algorithm]). Further details on the behavior of this instruction are described in [virt-control] and [pmp-vmem].

The SFENCE.VMA is used to flush any local hardware caches related to address translation. It is specified as a fence rather than a TLB flush to provide cleaner semantics with respect to which instructions are affected by the flush operation and to support a wider variety of dynamic caching structures and memory-management schemes. SFENCE.VMA is also used by higher privilege levels to synchronize page table writes and the address translation hardware.

SFENCE.VMA orders only the local hart's implicit references to the memory-management data structures.

Consequently, other harts must be notified separately when the memory-management data structures have been modified. One approach is to use 1) a local data fence to ensure local writes are visible globally, then 2) an interprocessor interrupt to the other thread, then 3) a local SFENCE.VMA in the interrupt handler of the remote thread, and finally 4) signal back to originating thread that operation is complete. This is, of course, the RISC-V analog to a TLB shootdown.

For the common case that the translation data structures have only been modified for a single address mapping (i.e., one page or superpage), *rs1* can specify a virtual address within that mapping to effect a translation fence for that mapping only. Furthermore, for the common case that the translation data structures have only been modified for a single address-space identifier, *rs2* can specify the address space. The behavior of SFENCE.VMA depends on *rs1* and *rs2* as follows:

- If *rs1*=`x0` and *rs2*=`x0`, the fence orders all reads and writes made to any level of the page tables, for all address spaces. The fence also invalidates all address-translation cache entries, for all address spaces.
- If *rs1*=`x0` and *rs2*≠`x0`, the fence orders all reads and writes made to any level of the page tables, but only for the address space identified by integer register *rs2*. Accesses to *global* mappings (see [translation]) are not ordered. The fence also invalidates all address-translation cache entries matching the address space identified by integer register *rs2*, except for entries containing global mappings.
- If *rs1*≠`x0` and *rs2*=`x0`, the fence orders only reads and writes made to leaf page table entries corresponding to the virtual address in *rs1*, for all address spaces. The fence also invalidates all address-translation cache entries that contain leaf page table entries corresponding to the virtual address in *rs1*, for all address spaces.
- If *rs1*≠`x0` and *rs2*≠`x0`, the fence orders only reads and writes made to leaf page table entries corresponding to the virtual address in *rs1*, for the address space identified by integer register *rs2*. Accesses to global mappings are not ordered. The fence also invalidates all address-translation cache entries that contain leaf page table entries corresponding to the virtual address in *rs1* and that match the address space identified by integer register *rs2*, except for entries containing global mappings.

If the value held in *rs1* is not a valid virtual address, then the SFENCE.VMA instruction has no effect. No exception is raised in this case.

When *rs2*≠`x0`, bits SXLEN-1:ASIDMAX of the value held in *rs2* are reserved for future standard use. Until their use is defined by a standard extension, they should be zeroed by software and ignored by current implementations. Furthermore, if ASIDLEN<ASIDMAX, the implementation shall ignore bits ASIDMAX-1:ASIDLEN of the value held in *rs2*.

It is always legal to over-fence, e.g., by fencing only based on a subset of the bits in *rs1* and/or *rs2*, and/or by simply treating all SFENCE.VMA instructions as having *rs1*=`x0` and/or *rs2*=`x0`. For example, simpler implementations can ignore the virtual address in *rs1* and the ASID value in *rs2* and always perform a global fence. The choice not to raise an exception when an invalid virtual address is held in *rs1* facilitates this type of simplification.

An implicit read of the memory-management data structures may return any translation for an address that was valid at any time since the most recent SFENCE.VMA that subsumes that address. The ordering implied by SFENCE.VMA does not place implicit reads and writes to the memory-management data structures into the global memory order in a way that interacts cleanly with the standard RVWMO ordering rules. In particular, even though an SFENCE.VMA orders prior explicit accesses before subsequent implicit accesses, and those implicit accesses are ordered before their associated explicit accesses, SFENCE.VMA does not necessarily place prior explicit accesses before subsequent explicit accesses in the global memory order. These implicit loads also need not otherwise obey normal program order semantics with respect to prior loads or stores to the same address.

A consequence of this specification is that an implementation may use any translation for an address that was valid at any time since the most recent

SFENCE.VMA that subsumes that address. In particular, if a leaf PTE is modified but a subsuming SFENCE.VMA is not executed, either the old translation or the new translation will be used, but the choice is unpredictable. The behavior is otherwise well-defined.

In a conventional TLB design, it is possible for multiple entries to match a single address if, for example, a page is upgraded to a superpage without first clearing the original non-leaf PTE's valid bit and executing an SFENCE.VMA with *rs1*=`x0`. In this case, a similar remark applies: it is unpredictable whether the old non-leaf PTE or the new leaf PTE is used, but the behavior is otherwise well defined.

Another consequence of this specification is that it is generally unsafe to update a PTE using a set of stores of a width less than the width of the PTE, as it is legal for the implementation to read the PTE at any time, including when only some of the partial stores have taken effect.

This specification permits the caching of PTEs whose V (Valid) bit is clear. Operating systems must be written to cope with this possibility, but implementers are reminded that eagerly caching invalid PTEs will reduce performance by causing additional page faults.

Implementations must only perform implicit reads of the translation data structures pointed to by the current contents of the satp register or a subsequent valid (V=1) translation data structure entry, and must only raise exceptions for implicit accesses that are generated as a result of instruction execution, not those that are performed speculatively.

Changes to the sstatus fields SUM and MXR take effect immediately, without the need to execute an SFENCE.VMA instruction. Changing satp.MODE from Bare to other modes and vice versa also takes effect immediately, without the need to execute an SFENCE.VMA instruction. Likewise, changes to satp.ASID take effect immediately.

The following common situations typically require executing an SFENCE.VMA instruction:

- When software recycles an ASID (i.e., reassociates it with a different page table), it should *first* change satp to point to the new page table using the recycled ASID, *then* execute SFENCE.VMA with *rs1*=`x0` and *rs2* set to the recycled ASID. Alternatively, software can execute the same SFENCE.VMA instruction while a different ASID is loaded into satp, provided the next time satp is loaded with the recycled ASID, it is simultaneously loaded with the new page table.
- If the implementation does not provide ASIDs, or software chooses to always use ASID 0, then after every satp write, software should execute SFENCE.VMA with *rs1*=`x0`. In the common case that no global translations have been modified, *rs2* should be set to a register other than `x0` but which contains the value zero, so that global translations are not flushed.

- If software modifies a non-leaf PTE, it should execute SFENCE.VMA with *rs1*=`x0`. If any PTE along the traversal path had its G bit set, *rs2* must be `x0`; otherwise, *rs2* should be set to the ASID for which the translation is being modified.
- If software modifies a leaf PTE, it should execute SFENCE.VMA with *rs1* set to a virtual address within the page. If any PTE along the traversal path had its G bit set, *rs2* must be `x0`; otherwise, *rs2* should be set to the ASID for which the translation is being modified.
- For the special cases of increasing the permissions on a leaf PTE and changing an invalid PTE to a valid leaf, software may choose to execute the SFENCE.VMA lazily. After modifying the PTE but before executing SFENCE.VMA, either the new or old permissions will be used. In the latter case, a page-fault exception might occur, at which point software should execute SFENCE.VMA in accordance with the previous bullet point.

If a hart employs an address-translation cache, that cache must appear to be private to that hart. In particular, the meaning of an ASID is local to a hart; software may choose to use the same ASID to refer to different address spaces on different harts.

A future extension could redefine ASIDs to be global across the SEE, enabling such options as shared translation caches and hardware support for broadcast TLB shootdown. However, as OSes have evolved to significantly reduce the scope of TLB shootdowns using novel ASID-management techniques, we expect the local-ASID scheme to remain attractive for its simplicity and possibly better scalability.

For implementations that make satp.MODE read-only zero (always Bare), attempts to execute an SFENCE.VMA instruction might raise an illegal-instruction exception.

B.236.3. Access

M	HS	U	VS	VU
Always	Always	Never	Always	Never

B.236.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

B.236.5. Execution

```
XReg vaddr = X[rs1];
Bits<16> asid = X[rs2][ASID_WIDTH - 1:0];
if (mode() == PrivilegeMode::U) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
```

```
if (CSR[misa].H == 1 && mode() == PrivilegeMode::VU) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
if (CSR[mstatus].TVM == 1 && mode() == PrivilegeMode::S) || (mode() == PrivilegeMode
::VS) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
if (CSR[misa].H == 1 && CSR[hstatus].VTVM == 1 && mode() == PrivilegeMode::VS) {
  raise(ExceptionCode::VirtualInstruction, mode(), $encoding);
}
if (!implemented?(ExtensionName::Sv32) && !implemented?(ExtensionName::Sv39) &&
!implemented?(ExtensionName::Sv48) && !implemented?(ExtensionName::Sv57)) {
  if (TRAP_ON_SFENCE_VMA_WHEN_SATP_MODE_IS_READ_ONLY) {
    raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
  }
}
VmaOrderType vma_type;
if (CSR[misa].H == 1 && mode() == PrivilegeMode::VS) {
  vma_type.vsmode = true;
  vma_type.single_vmid = true;
  vma_type.vmid = CSR[hgatp].VMID;
} else {
  vma_type.smode = true;
}
if ((rs1 == 0) && (rs2 == 0)) {
  vma_type.global = true;
  order_pgtbl_writes_before_vmafence(vma_type);
  invalidate_translations(vma_type);
  order_pgtbl_reads_after_vmafence(vma_type);
} else if ((rs1 == 0) && (rs2 != 0)) {
  vma_type.single_asid = true;
  vma_type.asid = asid;
  order_pgtbl_writes_before_vmafence(vma_type);
  invalidate_translations(vma_type);
  order_pgtbl_reads_after_vmafence(vma_type);
} else if ((rs1 != 0) && (rs2 == 0)) {
  if (canonical_vaddr?(vaddr)) {
    vma_type.single_vaddr = true;
    vma_type.vaddr = vaddr;
    order_pgtbl_writes_before_vmafence(vma_type);
    invalidate_translations(vma_type);
    order_pgtbl_reads_after_vmafence(vma_type);
  }
} else {
  if (canonical_vaddr?(vaddr)) {
    vma_type.single_asid = true;
    vma_type.asid = asid;
    vma_type.single_vaddr = true;
    vma_type.vaddr = vaddr;
    order_pgtbl_writes_before_vmafence(vma_type);
    invalidate_translations(vma_type);
```

```
    order_pgtbl_reads_after_vmafence(vma_type);
  }
}
```

B.236.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- VirtualInstruction

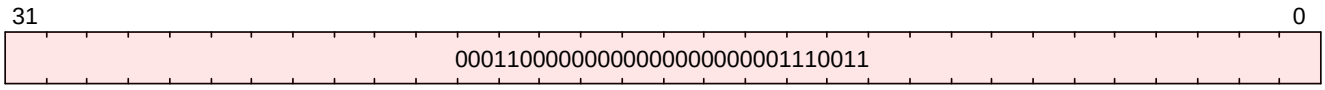
B.237. sfence.w.inval

Order writes before sfence

This instruction is defined by:

- Svinval, version >= 0

B.237.1. Encoding



B.237.2. Synopsis

The sfence.w.inval instruction guarantees that any previous stores already visible to the current RISC-V hart are ordered before subsequent sinval.vma instructions executed by the same hart.

B.237.3. Access

M	HS	U	VS	VU
Always	Sometimes	Never	Sometimes	Never

B.237.4. Decode Variables

```
```

B.237.5. Execution

```
if (mode() == PrivilegeMode::U) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
if (CSR[misa].H == 1 && mode() == PrivilegeMode::VU) {
  raise(ExceptionCode::VirtualInstruction, mode(), $encoding);
}
VmaOrderType vma_type;
vma_type.global = true;
vma_type.smode = true;
if (CSR[misa].H == 1) {
  vma_type.vsmode = true;
  vma_type.gstage = true;
}
order_pgtbl_writes_before_vmafence(vma_type);
```

B.237.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- VirtualInstruction

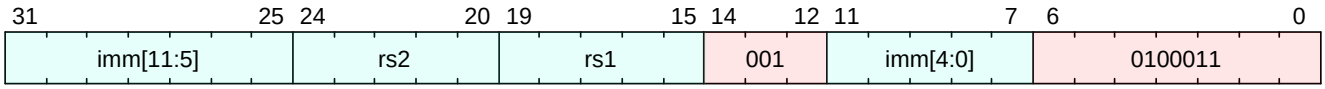
B.238. sh

Store halfword

This instruction is defined by:

- I, version >= 0

B.238.1. Encoding



B.238.2. Synopsis

Store 16 bits of data from register `rs2` to an address formed by adding `rs1` to a signed offset.

B.238.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.238.4. Decode Variables

```
Bits<12> imm = {$encoding[31:25], $encoding[11:7]};
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

B.238.5. Execution

```
XReg virtual_address = X[rs1] + imm;
write_memory<16>(virtual_address, X[rs2][15:0], $encoding);
```

B.238.6. Exceptions

This instruction may result in the following synchronous exceptions:

- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

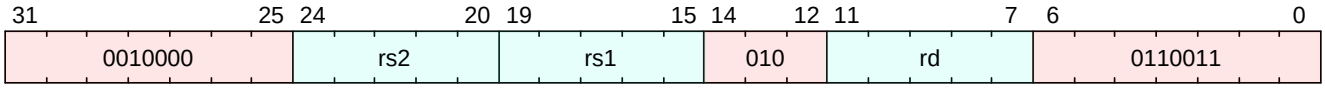
B.239. sh1add

Shift left by 1 and add

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zba, version >= 0

B.239.1. Encoding



B.239.2. Synopsis

This instruction shifts `rs1` to the left by 1 bit and adds it to `rs2`.

B.239.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.239.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.239.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = X[rs2] + (X[rs1] << 1);
```

B.239.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

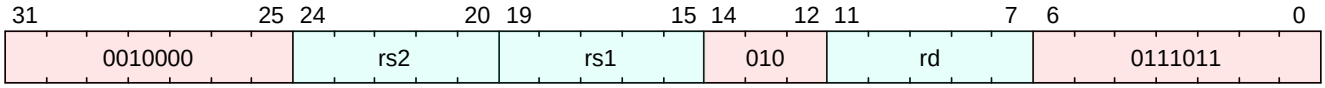
B.240. sh1add.uw

Shift unsigend word left by 1 and add

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zba, version >= 0

B.240.1. Encoding



B.240.2. Synopsis

This instruction performs an XLEN-wide addition of two addends. The first addend is rs2. The second addend is the unsigned value formed by extracting the least-significant word of rs1 and shifting it left by 1 place.

B.240.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.240.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.240.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = X[rs2] + (X[rs1][31:0] << 1);
```

B.240.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

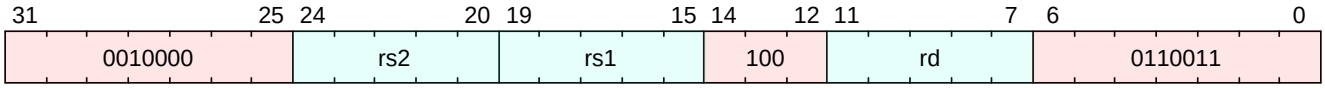
B.241. sh2add

Shift left by 2 and add

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zba, version >= 0

B.241.1. Encoding



B.241.2. Synopsis

This instruction shifts `rs1` to the left by 2 places and adds it to `rs2`.

B.241.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.241.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.241.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = X[rs2] + (X[rs1] << 2);
```

B.241.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

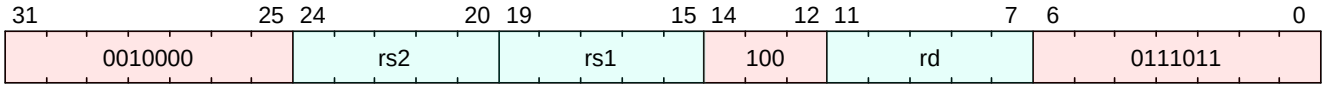
B.242. sh2add.uw

Shift unsigend word left by 2 and add

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zba, version >= 0

B.242.1. Encoding



B.242.2. Synopsis

This instruction performs an XLEN-wide addition of two addends. The first addend is rs2. The second addend is the unsigned value formed by extracting the least-significant word of rs1 and shifting it left by 2 places.

B.242.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.242.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.242.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = X[rs2] + (X[rs1][31:0] << 2);
```

B.242.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

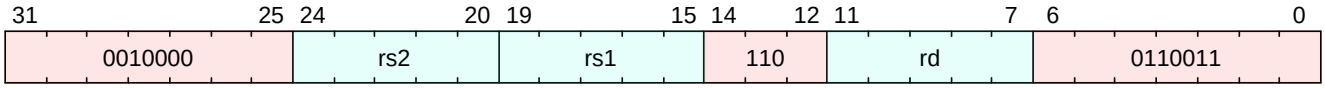
B.243. sh3add

Shift left by 3 and add

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zba, version >= 0

B.243.1. Encoding



B.243.2. Synopsis

This instruction shifts `rs1` to the left by 3 places and adds it to `rs2`.

B.243.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.243.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.243.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = X[rs2] + (X[rs1] << 3);
```

B.243.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

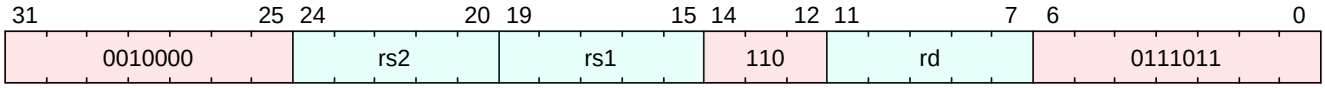
B.244. sh3add.uw

Shift unsigend word left by 3 and add

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zba, version >= 0

B.244.1. Encoding



B.244.2. Synopsis

This instruction performs an XLEN-wide addition of two addends. The first addend is rs2. The second addend is the unsigned value formed by extracting the least-significant word of rs1 and shifting it left by 3 places.

B.244.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.244.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.244.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = X[rs2] + (X[rs1][31:0] << 3);
```

B.244.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

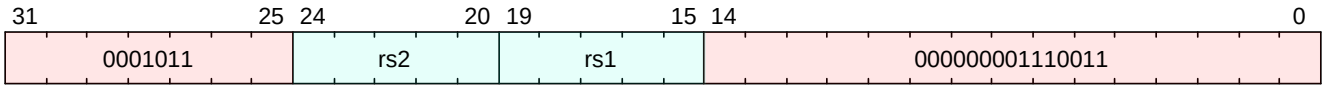
B.245. sinval.vma

Invalidate cached address translations

This instruction is defined by:

- Svinval, version >= 0

B.245.1. Encoding



B.245.2. Synopsis

The sinval.vma instruction invalidates any address-translation cache entries that an sfence.vma instruction with the same values of rs1 and rs2 would invalidate. However, unlike sfence.vma, sinval.vma instructions are only ordered with respect to sfence.vma, sfence.w.inval, and sfence.inval.ir instructions as defined below.

B.245.3. Access

M	HS	U	VS	VU
Always	Sometimes	Never	Sometimes	Never

B.245.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

B.245.5. Execution

```
XReg vaddr = X[rs1];
Bits<ASID_WIDTH> asid = X[rs2][ASID_WIDTH - 1:0];
if (CSR[mstatus].TVM == 1 && mode() == PrivilegeMode::S) || (mode() == PrivilegeMode
::VS) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
if (CSR[misa].H == 1 && CSR[hstatus].VTVM == 1 && mode() == PrivilegeMode::VS) {
  raise(ExceptionCode::VirtualInstruction, mode(), $encoding);
}
if (mode() == PrivilegeMode::U) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
if (CSR[misa].H == 1 && mode() == PrivilegeMode::VU) {
  raise(ExceptionCode::VirtualInstruction, mode(), $encoding);
}
```

```
VmaOrderType vma_type;
if (CSR[misa].H == 1 && mode() == PrivilegeMode::VS) {
  vma_type.vsmode = true;
  vma_type.single_vmid = true;
  vma_type.vmid = CSR[hgatp].VMID;
} else {
  vma_type.smode = true;
}
if ((rs1 == 0) && (rs2 == 0)) {
  vma_type.global = true;
  invalidate_translations(vma_type);
} else if ((rs1 == 0) && (rs2 != 0)) {
  vma_type.single_asid = true;
  vma_type.asid = asid;
  invalidate_translations(vma_type);
} else if ((rs1 != 0) && (rs2 == 0)) {
  if (canonical_vaddr?(vaddr)) {
    vma_type.single_vaddr = true;
    vma_type.vaddr = vaddr;
    invalidate_translations(vma_type);
  }
} else {
  if (canonical_vaddr?(vaddr)) {
    vma_type.single_asid = true;
    vma_type.asid = asid;
    vma_type.single_vaddr = true;
    vma_type.vaddr = vaddr;
    invalidate_translations(vma_type);
  }
}
```

B.245.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- VirtualInstruction

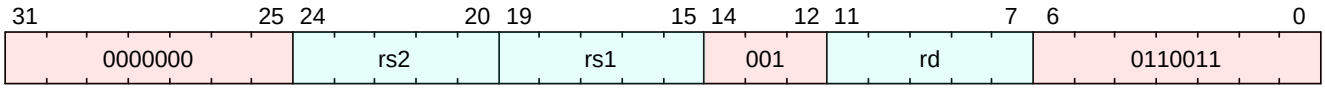
B.246. sll

Shift left logical

This instruction is defined by:

- I, version >= 0

B.246.1. Encoding



B.246.2. Synopsis

Shift the value in `rs1` left by the value in the lower 6 bits of `rs2`, and store the result in `rd`.

B.246.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.246.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.246.5. Execution

```
if (xlen() == 64) {
  X[rd] = X[rs1] << X[rs2][5:0];
} else {
  X[rd] = X[rs1] << X[rs2][4:0];
}
```

B.246.6. Exceptions

This instruction does not generate synchronous exceptions.

B.247. slli

Shift left logical immediate

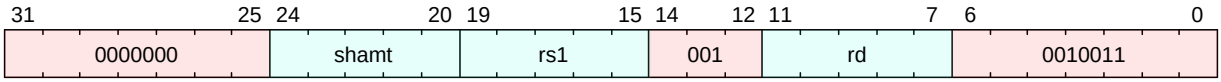
This instruction is defined by:

- I, version >= 0

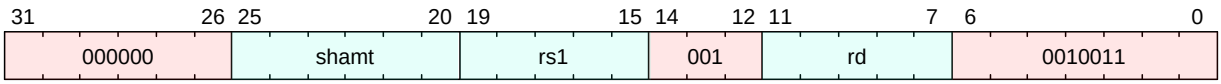
B.247.1. Encoding

This instruction has different encodings in RV32 and RV64.

RV32



RV64



B.247.2. Synopsis

Shift the value in rs1 left by shamt, and store the result in rd

B.247.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.247.4. Decode Variables

RV32

```
Bits<5> shamt = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

RV64

```
Bits<6> shamt = $encoding[25:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.247.5. Execution

```
X[rd] = X[rs1] << shamt;
```

B.247.6. Exceptions

This instruction does not generate synchronous exceptions.

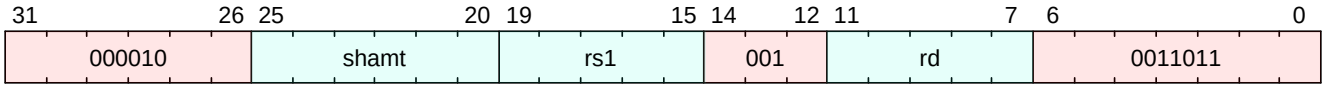
B.248. slli.uw

Shift left unsigned word (Immediate)

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zba, version >= 0

B.248.1. Encoding



B.248.2. Synopsis

This instruction takes the least-significant word of rs1, zero-extends it, and shifts it left by the immediate.

This instruction is the same as slli with `zext.w` performed on rs1 before shifting.

B.248.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.248.4. Decode Variables

```
Bits<6> shamt = $encoding[25:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.248.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = X[rs1][31:0] << shamt;
```

B.248.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

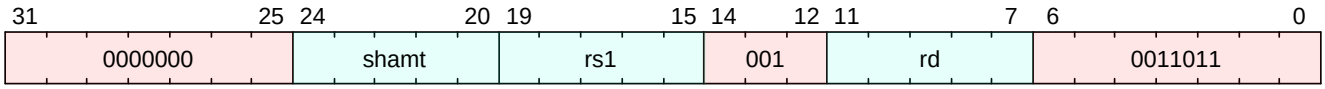
B.249. slliw

Shift left logical immediate word

This instruction is defined by:

- I, version >= 0

B.249.1. Encoding



B.249.2. Synopsis

Shift the 32-bit value in rs1 left by shamt, and store the sign-extended result in rd

B.249.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.249.4. Decode Variables

```
Bits<5> shamt = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.249.5. Execution

```
X[rd] = sext(X[rs1] << shamt, 31);
```

B.249.6. Exceptions

This instruction does not generate synchronous exceptions.

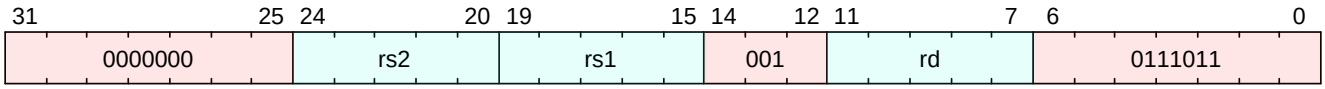
B.250. sllw

Shift left logical word

This instruction is defined by:

- I, version >= 0

B.250.1. Encoding



B.250.2. Synopsis

Shift the 32-bit value in `rs1` left by the value in the lower 5 bits of `rs2`, and store the sign-extended result in `rd`.

B.250.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.250.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.250.5. Execution

```
X[rd] = sext(X[rs1] << X[rs2][4:0], 31);
```

B.250.6. Exceptions

This instruction does not generate synchronous exceptions.

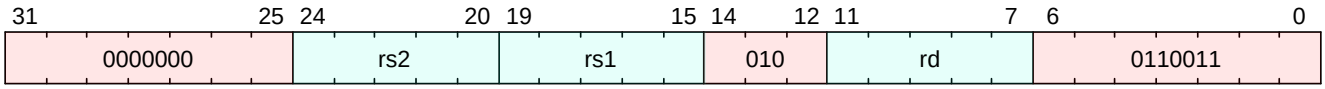
B.251. slt

Set on less than

This instruction is defined by:

- I, version >= 0

B.251.1. Encoding



B.251.2. Synopsis

Places the value 1 in register `rd` if register `rs1` is less than the value in register `rs2`, where both sources are treated as signed numbers, else 0 is written to `rd`.

B.251.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.251.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.251.5. Execution

```
XReg src1 = X[rs1];
XReg src2 = X[rs2];
X[rd] = ($signed(src1) < $signed(src2)) ? '1 : '0;
```

B.251.6. Exceptions

This instruction does not generate synchronous exceptions.

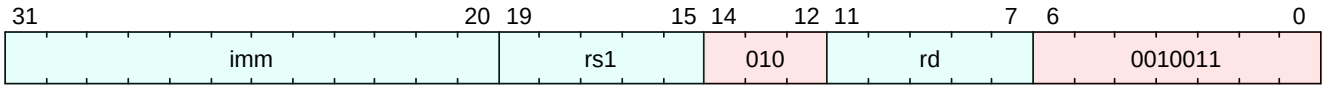
B.252. slti

Set on less than immediate

This instruction is defined by:

- I, version >= 0

B.252.1. Encoding



B.252.2. Synopsis

Places the value 1 in register `rd` if register `rs1` is less than the sign-extended immediate when both are treated as signed numbers, else 0 is written to `rd`.

B.252.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.252.4. Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.252.5. Execution

```
X[rd] = ($signed(X[rs1]) < $signed(imm)) ? '1 : '0;
```

B.252.6. Exceptions

This instruction does not generate synchronous exceptions.

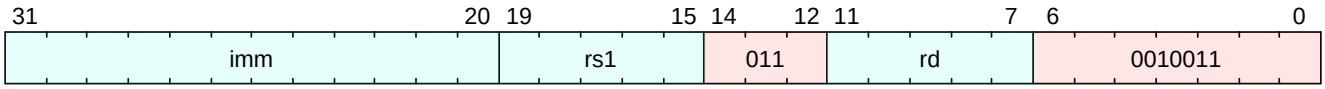
B.253. sltiu

Set on less than immediate unsigned

This instruction is defined by:

- I, version >= 0

B.253.1. Encoding



B.253.2. Synopsis

Places the value 1 in register `rd` if register `rs1` is less than the sign-extended immediate when both are treated as unsigned numbers (*i.e.*, the immediate is first sign-extended to XLEN bits then treated as an unsigned number), else 0 is written to `rd`.

`sltiu rd, rs1, 1` sets `rd` to 1 if `rs1` equals zero, otherwise sets `rd` to 0 (assembler pseudoinstruction `SEQZ rd, rs`).

B.253.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.253.4. Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.253.5. Execution

```
X[rd] = (X[rs1] < imm) ? 1 : 0;
```

B.253.6. Exceptions

This instruction does not generate synchronous exceptions.

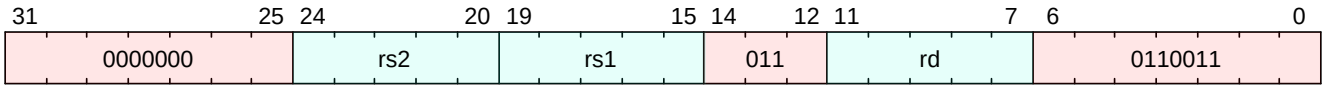
B.254. sltu

Set on less than unsigned

This instruction is defined by:

- I, version >= 0

B.254.1. Encoding



B.254.2. Synopsis

Places the value 1 in register `rd` if register `rs1` is less than the value in register `rs2`, where both sources are treated as unsigned numbers, else 0 is written to `rd`.

B.254.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.254.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.254.5. Execution

```
X[rd] = (X[rs1] < X[rs2]) ? 1 : 0;
```

B.254.6. Exceptions

This instruction does not generate synchronous exceptions.

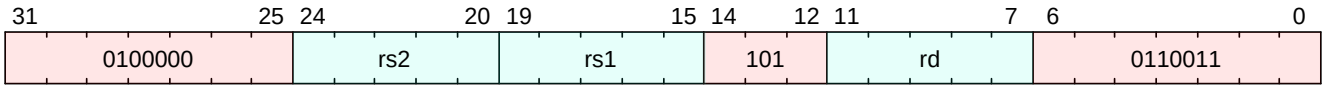
B.255. sra

Shift right arithmetic

This instruction is defined by:

- I, version >= 0

B.255.1. Encoding



B.255.2. Synopsis

Arithmetic shift the value in `rs1` right by the value in the lower 5 bits of `rs2`, and store the result in `rd`.

B.255.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.255.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.255.5. Execution

```
if (xlen() == 64) {
  X[rd] = X[rs1] >>> X[rs2][5:0];
} else {
  X[rd] = X[rs1] >>> X[rs2][4:0];
}
```

B.255.6. Exceptions

This instruction does not generate synchronous exceptions.

B.256. srai

Shift right arithmetic immediate

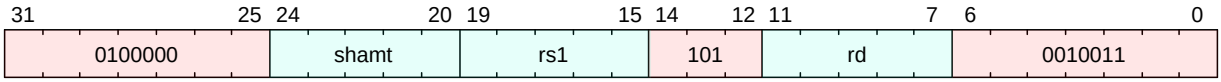
This instruction is defined by:

- I, version >= 0

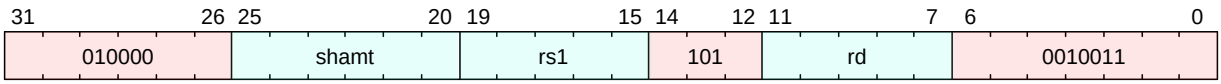
B.256.1. Encoding

This instruction has different encodings in RV32 and RV64.

RV32



RV64



B.256.2. Synopsis

Arithmetic shift (the original sign bit is copied into the vacated upper bits) the value in rs1 right by shamt, and store the result in rd.

B.256.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.256.4. Decode Variables

RV32

```
Bits<5> shamt = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

RV64

```
Bits<6> shamt = $encoding[25:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.256.5. Execution

```
X[rd] = X[rs1] >>> shamt;
```

B.256.6. Exceptions

This instruction does not generate synchronous exceptions.

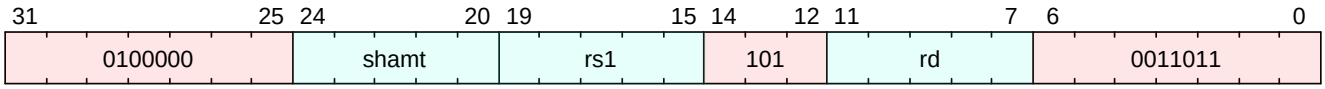
B.257. sraiw

Shift right arithmetic immediate word

This instruction is defined by:

- I, version >= 0

B.257.1. Encoding



B.257.2. Synopsis

Arithmetic shift (the original sign bit is copied into the vacated upper bits) the 32-bit value in rs1 right by shamt, and store the sign-extended result in rd.

B.257.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.257.4. Decode Variables

```
Bits<5> shamt = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.257.5. Execution

```
XReg operand = sext(X[rs1], 31);
X[rd] = sext(operand >>> shamt, 31);
```

B.257.6. Exceptions

This instruction does not generate synchronous exceptions.

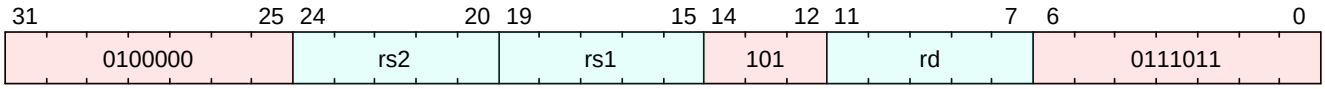
B.258. sraw

Shift right arithmetic word

This instruction is defined by:

- I, version >= 0

B.258.1. Encoding



B.258.2. Synopsis

Arithmetic shift the 32-bit value in `rs1` right by the value in the lower 5 bits of `rs2`, and store the sign-extended result in `rd`.

B.258.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.258.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.258.5. Execution

```
XReg operand1 = sext(X[rs1], 31);
X[rd] = sext(operand1 >>> X[rs2][4:0], 31);
```

B.258.6. Exceptions

This instruction does not generate synchronous exceptions.

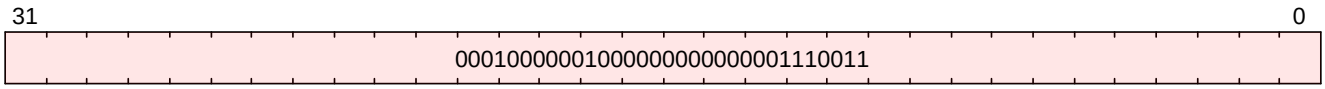
B.259. sret

Supervisor Exception Return

This instruction is defined by:

- S, version >= 0

B.259.1. Encoding



B.259.2. Synopsis

Returns from an exception.

When sret is allowed to execute, its behavior depends on whether or not the current privilege mode is virtualized.

When the current privlege mode is (H)S-mode or M-mode

sret sets hstatus = 0, mstatus.SPP = 0, mstatus.SIE = mstatus.SPIE, and mstatus.SPIE = 1, changes the privlege mode according to the table below, and then jumps to the address in sepc.

Table 55. Next privilege mode following an sret in (H)S-mode or M-mode

mstatus.SPP	hstatus.SPV	Mode after sret
0	0	U-mode
0	1	VU-mode
1	0	(H)S-mode
1	1	VS-mode

When the current privlege mode is VS-mode

sret sets vsstatus.SPP = 0, vsstatus.SIE = `vstatus.SPIE`, and vsstatus.SPIE = 1, changes the privlege mode according to the table below, and then jumps to the address in vsepc.

Table 56. Next privilege mode following an sret in (H)S-mode or M-mode

vsstatus.SPP	Mode after sret
0	VU-mode
1	VS-mode

B.259.3. Access

M	HS	U	VS	VU
Always	Sometimes	Never	Sometimes	Never

Access is determined as follows:

mstatus.TSR	hstatus.VTSR	Behavior when executed from:				
		M-mode	**U-mode**	**(H)S-mode**	**VU-mode**	**VS-mode**
0	0	executes	Illegal Instruction	executes	Virtual Instruction	executes
0	1	executes	Illegal Instruction	executes	Virtual Instruction	Virtual Instruction
1	0	executes	Illegal Instruction	Illegal Instruction	Virtual Instruction	executes
1	1	executes	Illegal Instruction	Illegal Instruction	Virtual Instruction	Virtual Instruction

B.259.4. Decode Variables

```
```

B.259.5. Execution

```
if (implemented?(ExtensionName::H)) {
  if (CSR[mstatus].TSR == 1'b0 && CSR[hstatus].VTSR == 1'b0) {
    if (mode() == PrivilegeMode::U) {
      raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
    } else if (mode() == PrivilegeMode::VU) {
      raise(ExceptionCode::VirtualInstruction, mode(), $encoding);
    }
  } else if (CSR[mstatus].TSR == 1'b0 && CSR[hstatus].VTSR == 1'b1) {
    if (mode() == PrivilegeMode::U) {
      raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
    } else if (mode() == PrivilegeMode::VU || mode() == PrivilegeMode::VS) {
      raise(ExceptionCode::VirtualInstruction, mode(), $encoding);
    }
  } else if (CSR[mstatus].TSR == 1'b1 && CSR[hstatus].VTSR == 1'b0) {
    if (mode() == PrivilegeMode::U || mode() == PrivilegeMode::S) {
      raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
    } else if (mode() == PrivilegeMode::VU) {
      raise(ExceptionCode::VirtualInstruction, mode(), $encoding);
    }
  } else if (CSR[mstatus].TSR == 1'b1 && CSR[hstatus].VTSR == 1'b1) {
    if (mode() == PrivilegeMode::U || mode() == PrivilegeMode::S) {
```

```
      raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
    } else if (mode() == PrivilegeMode::VU || mode() == PrivilegeMode::VS) {
      raise(ExceptionCode::VirtualInstruction, mode(), $encoding);
    }
  }
} else {
  if (mode() != PrivilegeMode::U) {
    raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
  }
}
if (!virtual_mode?()) {
  if (implemented?(ExtensionName::H)) {
    if (CSR[hstatus].SPV == 1'b1) {
      if (CSR[mstatus].SPP == 2'b01) {
        set_mode(PrivilegeMode::VS);
      } else if (CSR[mstatus].SPP == 2'b00) {
        set_mode(PrivilegeMode::VU);
      }
    }
    CSR[hstatus].SPV = 0;
  }
  CSR[mstatus].SIE = CSR[mstatus].SPIE;
  CSR[mstatus].SPIE = 1;
  CSR[mstatus].SPP = 2'b00;
  $pc = $bits(CSR[sepc]);
} else {
  if (CSR[mstatus].TSR == 1'b1) {
    raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
  }
  CSR[vsstatus].SPP = 0;
  CSR[vsstatus].SIE = CSR[vsstatus].SPIE;
  CSR[vsstatus].SPIE = 1;
  $pc = $bits(CSR[vsepc]);
}
```

B.259.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction
- VirtualInstruction

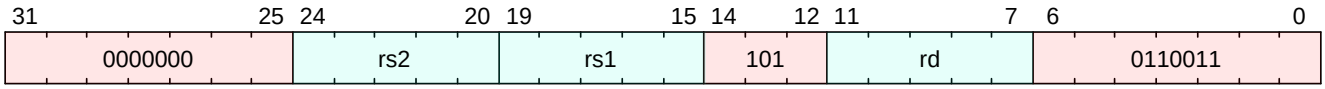
B.260. srl

Shift right logical

This instruction is defined by:

- I, version >= 0

B.260.1. Encoding



B.260.2. Synopsis

Logical shift the value in `rs1` right by the value in the lower bits of `rs2`, and store the result in `rd`.

B.260.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.260.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.260.5. Execution

```
if (xlen() == 64) {
  X[rd] = X[rs1] >> X[rs2][5:0];
} else {
  X[rd] = X[rs1] >> X[rs2][4:0];
}
```

B.260.6. Exceptions

This instruction does not generate synchronous exceptions.

B.261. srli

Shift right logical immediate

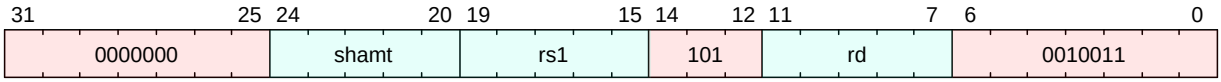
This instruction is defined by:

- I, version >= 0

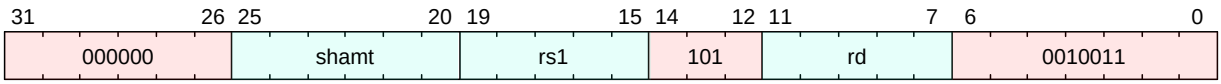
B.261.1. Encoding

This instruction has different encodings in RV32 and RV64.

RV32



RV64



B.261.2. Synopsis

Shift the value in rs1 right by shamt, and store the result in rd

B.261.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.261.4. Decode Variables

RV32

```
Bits<5> shamt = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

RV64

```
Bits<6> shamt = $encoding[25:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.261.5. Execution

```
X[rd] = X[rs1] >> shamt;
```

B.261.6. Exceptions

This instruction does not generate synchronous exceptions.

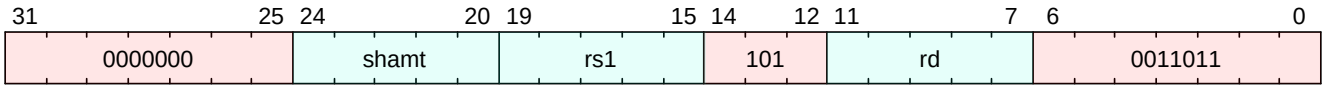
B.262. srliw

Shift right logical immediate word

This instruction is defined by:

- I, version >= 0

B.262.1. Encoding



B.262.2. Synopsis

Shift the 32-bit value in rs1 right by shamt, and store the sign-extended result in rd

B.262.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.262.4. Decode Variables

```
Bits<5> shamt = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.262.5. Execution

```
XReg operand = X[rs1][31:0];
X[rd] = sext(operand >> shamt, 31);
```

B.262.6. Exceptions

This instruction does not generate synchronous exceptions.

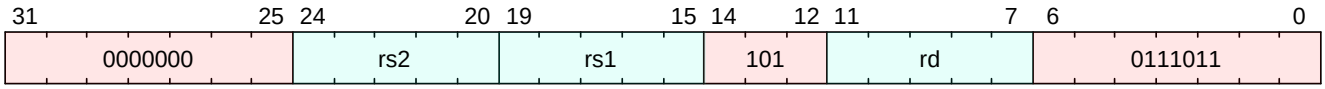
B.263. srlw

Shift right logical word

This instruction is defined by:

- I, version >= 0

B.263.1. Encoding



B.263.2. Synopsis

Logical shift the 32-bit value in `rs1` right by the value in the lower 5 bits of `rs2`, and store the sign-extended result in `rd`.

B.263.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.263.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.263.5. Execution

```
X[rd] = sext(X[rs1][31:0] >> X[rs2][4:0], 31);
```

B.263.6. Exceptions

This instruction does not generate synchronous exceptions.

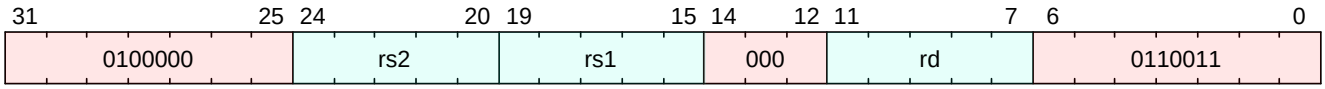
B.264. sub

Subtract

This instruction is defined by:

- I, version >= 0

B.264.1. Encoding



B.264.2. Synopsis

Subtract the value in rs2 from rs1, and store the result in rd

B.264.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.264.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.264.5. Execution

```
XReg t0 = X[rs1];
XReg t1 = X[rs2];
X[rd] = t0 - t1;
```

B.264.6. Exceptions

This instruction does not generate synchronous exceptions.

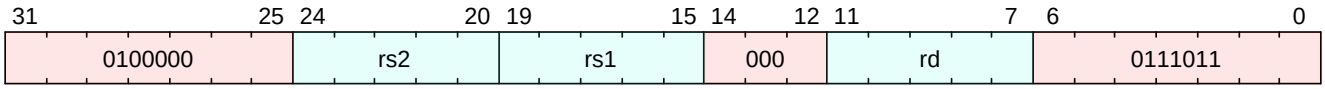
B.265. subw

Subtract word

This instruction is defined by:

- I, version >= 0

B.265.1. Encoding



B.265.2. Synopsis

Subtract the 32-bit values in rs2 from rs1, and store the sign-extended result in rd

B.265.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.265.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.265.5. Execution

```
Bits<32> t0 = X[rs1][31:0];
Bits<32> t1 = X[rs2][31:0];
X[rd] = sext(t0 - t1, 31);
```

B.265.6. Exceptions

This instruction does not generate synchronous exceptions.

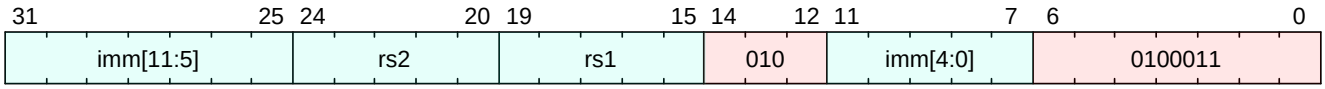
B.266. sw

Store word

This instruction is defined by:

- I, version >= 0

B.266.1. Encoding



B.266.2. Synopsis

Store 32 bits of data from register `rs2` to an address formed by adding `rs1` to a signed offset.

B.266.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.266.4. Decode Variables

```
Bits<12> imm = {$encoding[31:25], $encoding[11:7]};
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
```

B.266.5. Execution

```
XReg virtual_address = X[rs1] + imm;
write_memory<32>(virtual_address, X[rs2][31:0], $encoding);
```

B.266.6. Exceptions

This instruction may result in the following synchronous exceptions:

- LoadAccessFault
- StoreAmoAccessFault
- StoreAmoAddressMisaligned
- StoreAmoPageFault

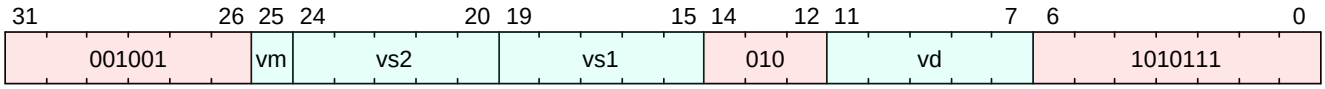
B.267. vaadd.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.267.1. Encoding



B.267.2. Synopsis

No description available.

B.267.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.267.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.267.5. Execution

```

```

B.267.6. Exceptions

This instruction does not generate synchronous exceptions.

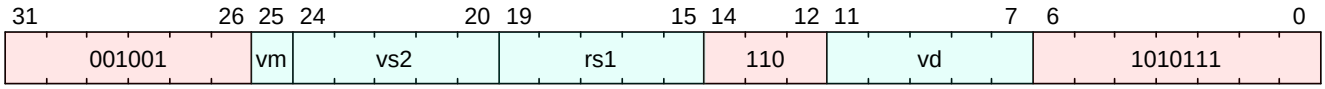
B.268. vaadd.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.268.1. Encoding



B.268.2. Synopsis

No description available.

B.268.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.268.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.268.5. Execution

```

```

B.268.6. Exceptions

This instruction does not generate synchronous exceptions.

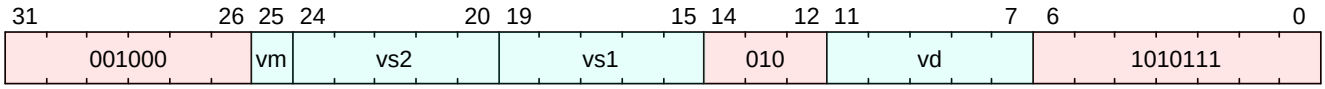
B.269. vaaddu.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.269.1. Encoding



B.269.2. Synopsis

No description available.

B.269.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.269.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.269.5. Execution

```

```

B.269.6. Exceptions

This instruction does not generate synchronous exceptions.

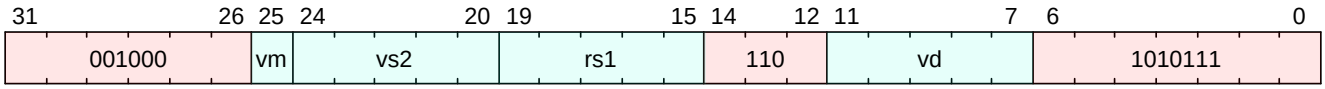
B.270. vaaddu.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.270.1. Encoding



B.270.2. Synopsis

No description available.

B.270.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.270.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.270.5. Execution

```

```

B.270.6. Exceptions

This instruction does not generate synchronous exceptions.

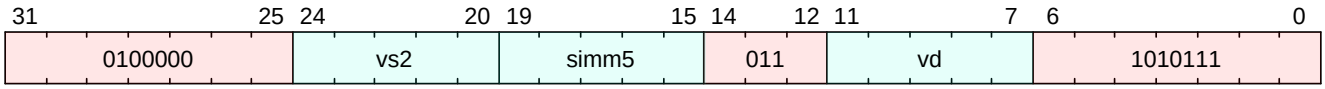
B.271. vadc.vim

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.271.1. Encoding



B.271.2. Synopsis

No description available.

B.271.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.271.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.271.5. Execution

```

```

B.271.6. Exceptions

This instruction does not generate synchronous exceptions.

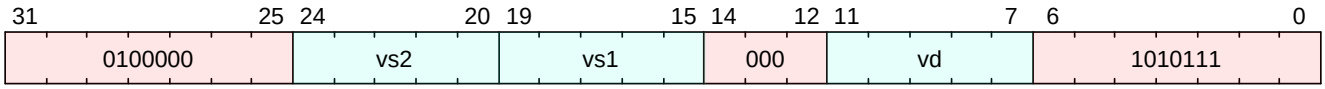
B.272. vadc.vvm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.272.1. Encoding



B.272.2. Synopsis

No description available.

B.272.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.272.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.272.5. Execution

```

```

B.272.6. Exceptions

This instruction does not generate synchronous exceptions.

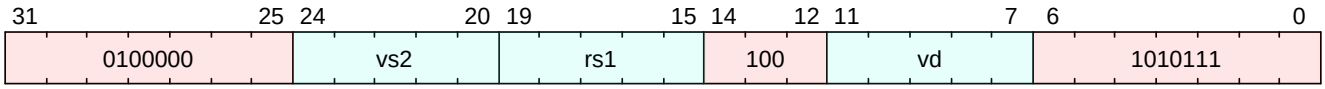
B.273. vadc.vxm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.273.1. Encoding



B.273.2. Synopsis

No description available.

B.273.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.273.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.273.5. Execution

```

```

B.273.6. Exceptions

This instruction does not generate synchronous exceptions.

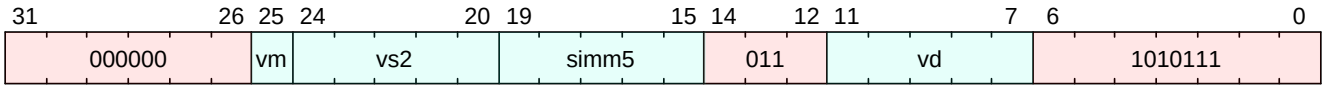
B.274. vadd.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.274.1. Encoding



B.274.2. Synopsis

No description available.

B.274.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.274.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.274.5. Execution

```

```

B.274.6. Exceptions

This instruction does not generate synchronous exceptions.

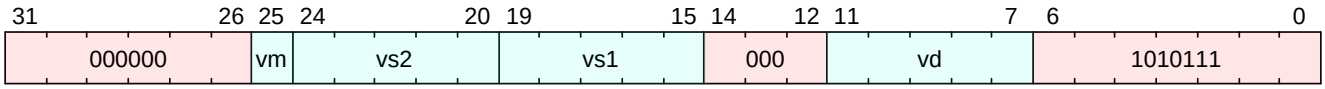
B.275. vadd.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.275.1. Encoding



B.275.2. Synopsis

No description available.

B.275.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.275.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.275.5. Execution

```

```

B.275.6. Exceptions

This instruction does not generate synchronous exceptions.

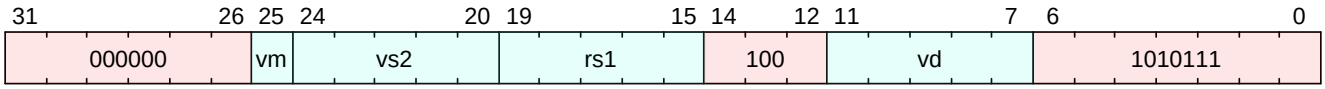
B.276. vadd.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.276.1. Encoding



B.276.2. Synopsis

No description available.

B.276.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.276.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.276.5. Execution

```

```

B.276.6. Exceptions

This instruction does not generate synchronous exceptions.

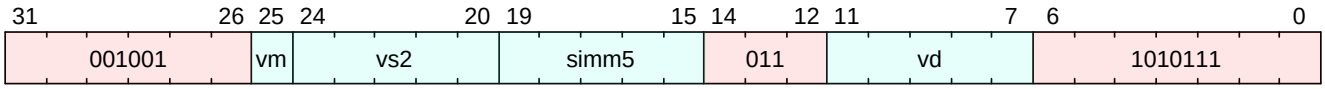
B.277. vand.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.277.1. Encoding



B.277.2. Synopsis

No description available.

B.277.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.277.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.277.5. Execution

```

```

B.277.6. Exceptions

This instruction does not generate synchronous exceptions.

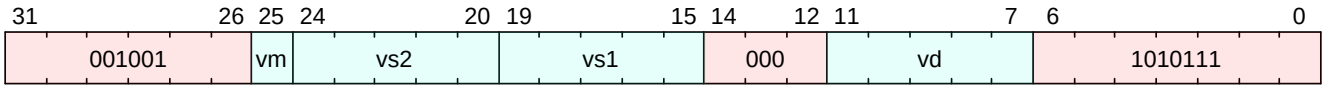
B.278. vand.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.278.1. Encoding



B.278.2. Synopsis

No description available.

B.278.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.278.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.278.5. Execution

```

```

B.278.6. Exceptions

This instruction does not generate synchronous exceptions.

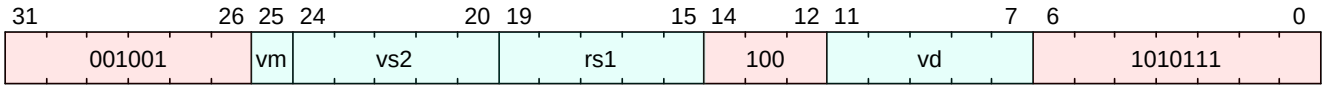
B.279. vand.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.279.1. Encoding



B.279.2. Synopsis

No description available.

B.279.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.279.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.279.5. Execution

```

```

B.279.6. Exceptions

This instruction does not generate synchronous exceptions.

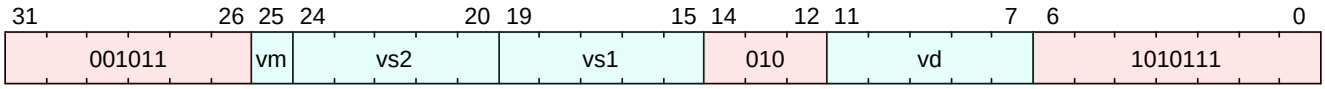
B.280. vasub.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.280.1. Encoding



B.280.2. Synopsis

No description available.

B.280.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.280.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.280.5. Execution

```

```

B.280.6. Exceptions

This instruction does not generate synchronous exceptions.

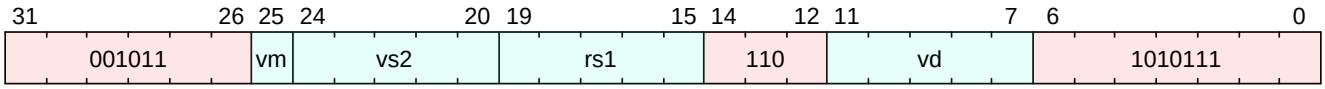
B.281. vasub.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.281.1. Encoding



B.281.2. Synopsis

No description available.

B.281.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.281.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.281.5. Execution

```

```

B.281.6. Exceptions

This instruction does not generate synchronous exceptions.

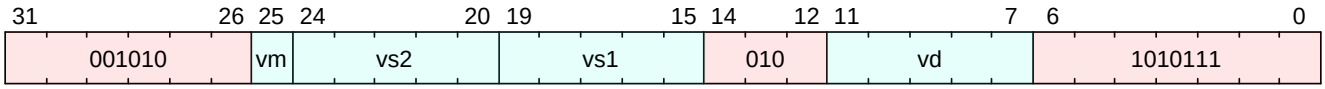
B.282. vasubu.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.282.1. Encoding



B.282.2. Synopsis

No description available.

B.282.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.282.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.282.5. Execution

```

```

B.282.6. Exceptions

This instruction does not generate synchronous exceptions.

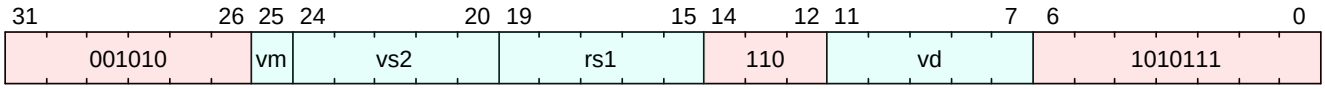
B.283. vasubu.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.283.1. Encoding



B.283.2. Synopsis

No description available.

B.283.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.283.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.283.5. Execution

```

```

B.283.6. Exceptions

This instruction does not generate synchronous exceptions.

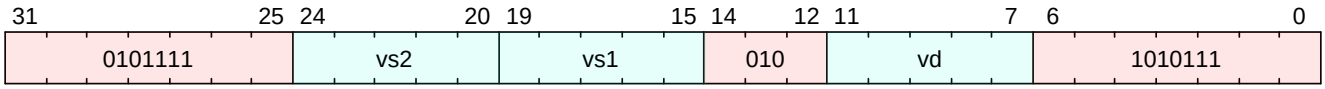
B.284. vcompress.vm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.284.1. Encoding



B.284.2. Synopsis

No description available.

B.284.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.284.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.284.5. Execution

```

```

B.284.6. Exceptions

This instruction does not generate synchronous exceptions.

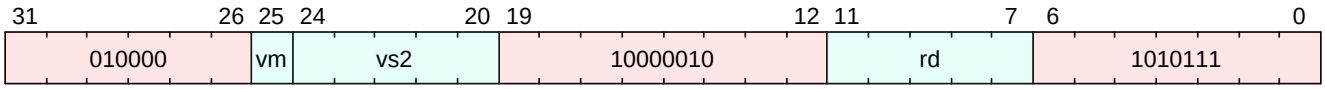
B.285. vcpop.m

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.285.1. Encoding



B.285.2. Synopsis

No description available.

B.285.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.285.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rd = $encoding[11:7];
```

B.285.5. Execution

```

```

B.285.6. Exceptions

This instruction does not generate synchronous exceptions.

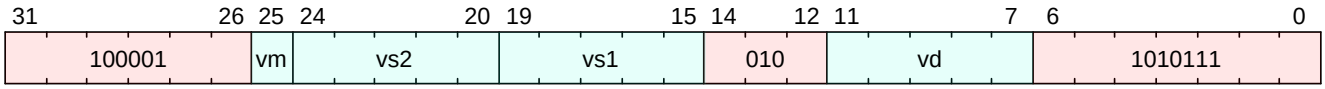
B.286. vdiv.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.286.1. Encoding



B.286.2. Synopsis

No description available.

B.286.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.286.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.286.5. Execution

```

```

B.286.6. Exceptions

This instruction does not generate synchronous exceptions.

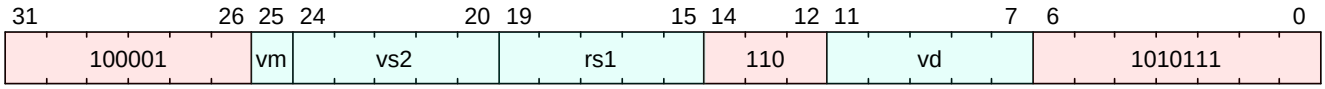
B.287. vdiv.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.287.1. Encoding



B.287.2. Synopsis

No description available.

B.287.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.287.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.287.5. Execution

```

```

B.287.6. Exceptions

This instruction does not generate synchronous exceptions.

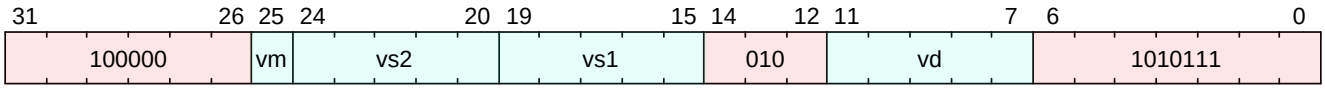
B.288. vdivu.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.288.1. Encoding



B.288.2. Synopsis

No description available.

B.288.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.288.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.288.5. Execution

```

```

B.288.6. Exceptions

This instruction does not generate synchronous exceptions.

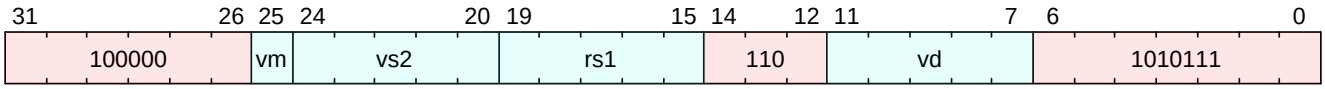
B.289. vdivu.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.289.1. Encoding



B.289.2. Synopsis

No description available.

B.289.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.289.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.289.5. Execution

```

```

B.289.6. Exceptions

This instruction does not generate synchronous exceptions.

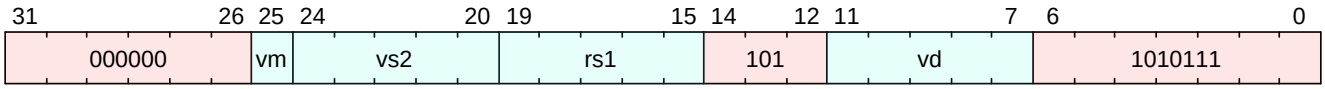
B.290. vfadd.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.290.1. Encoding



B.290.2. Synopsis

No description available.

B.290.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.290.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.290.5. Execution

```

```

B.290.6. Exceptions

This instruction does not generate synchronous exceptions.

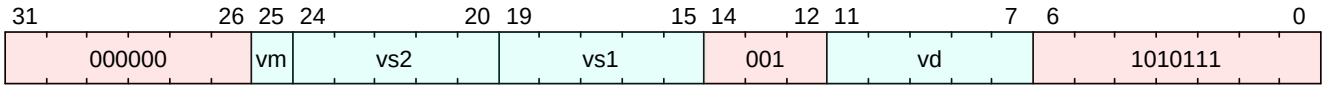
B.291. vfadd.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.291.1. Encoding



B.291.2. Synopsis

No description available.

B.291.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.291.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.291.5. Execution

```

```

B.291.6. Exceptions

This instruction does not generate synchronous exceptions.

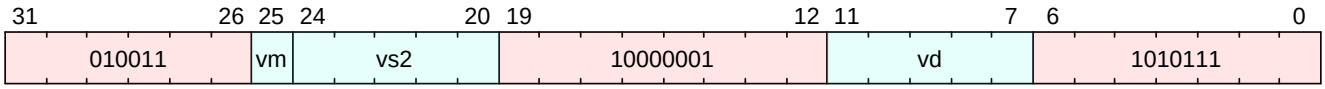
B.292. vfclass.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.292.1. Encoding



B.292.2. Synopsis

No description available.

B.292.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.292.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.292.5. Execution

```

```

B.292.6. Exceptions

This instruction does not generate synchronous exceptions.

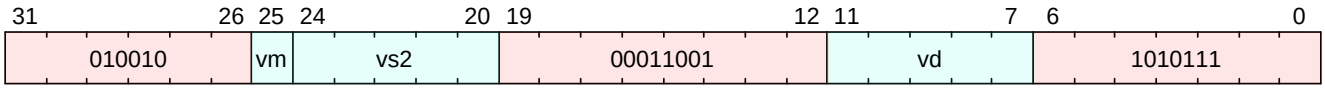
B.293. vfcvt.f.x.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.293.1. Encoding



B.293.2. Synopsis

No description available.

B.293.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.293.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.293.5. Execution

```

```

B.293.6. Exceptions

This instruction does not generate synchronous exceptions.

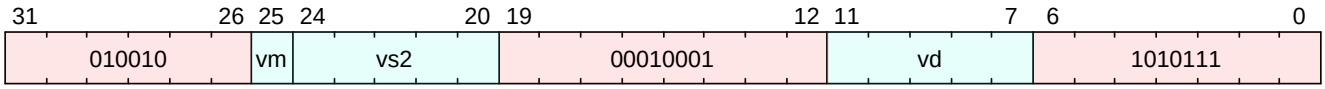
B.294. vfcvt.f.xu.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.294.1. Encoding



B.294.2. Synopsis

No description available.

B.294.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.294.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.294.5. Execution

```

```

B.294.6. Exceptions

This instruction does not generate synchronous exceptions.

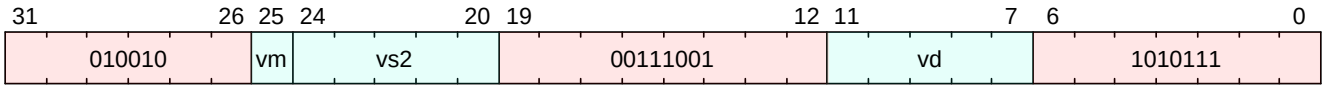
B.295. vfcvt.rtz.x.f.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.295.1. Encoding



B.295.2. Synopsis

No description available.

B.295.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.295.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.295.5. Execution

```

```

B.295.6. Exceptions

This instruction does not generate synchronous exceptions.

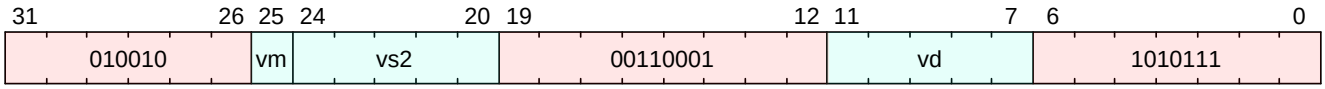
B.296. vfcvt.rtz.xu.f.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.296.1. Encoding



B.296.2. Synopsis

No description available.

B.296.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.296.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.296.5. Execution

```

```

B.296.6. Exceptions

This instruction does not generate synchronous exceptions.

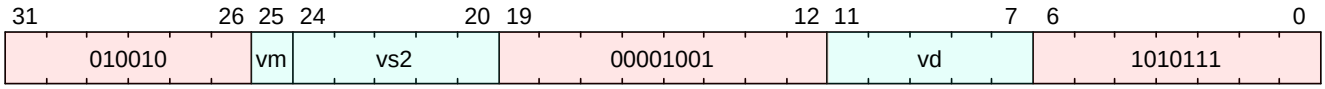
B.297. vfcvt.x.f.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.297.1. Encoding



B.297.2. Synopsis

No description available.

B.297.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.297.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.297.5. Execution

```

```

B.297.6. Exceptions

This instruction does not generate synchronous exceptions.

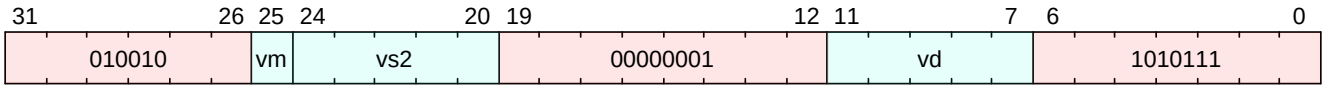
B.298. vfcvt.xu.f.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.298.1. Encoding



B.298.2. Synopsis

No description available.

B.298.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.298.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.298.5. Execution

```

```

B.298.6. Exceptions

This instruction does not generate synchronous exceptions.

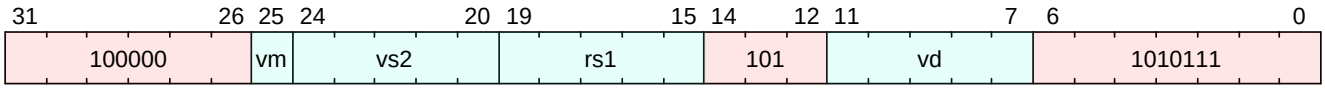
B.299. vfdiv.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.299.1. Encoding



B.299.2. Synopsis

No description available.

B.299.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.299.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.299.5. Execution

```

```

B.299.6. Exceptions

This instruction does not generate synchronous exceptions.

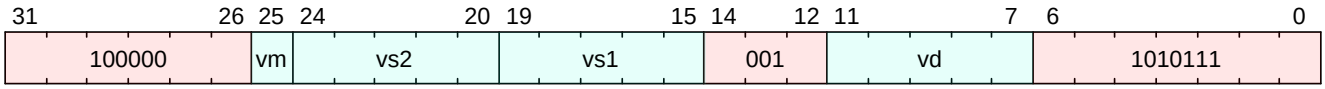
B.300. vfdiv.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.300.1. Encoding



B.300.2. Synopsis

No description available.

B.300.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.300.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.300.5. Execution

```

```

B.300.6. Exceptions

This instruction does not generate synchronous exceptions.

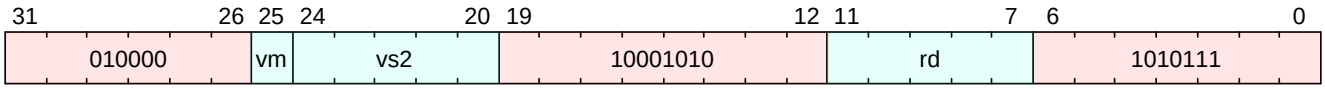
B.301. vfirst.m

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.301.1. Encoding



B.301.2. Synopsis

No description available.

B.301.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.301.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rd = $encoding[11:7];
```

B.301.5. Execution

```

```

B.301.6. Exceptions

This instruction does not generate synchronous exceptions.

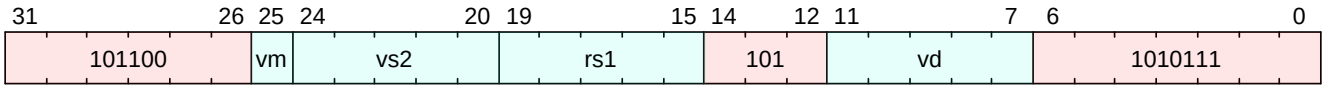
B.302. vfmacc.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.302.1. Encoding



B.302.2. Synopsis

No description available.

B.302.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.302.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.302.5. Execution

```

```

B.302.6. Exceptions

This instruction does not generate synchronous exceptions.

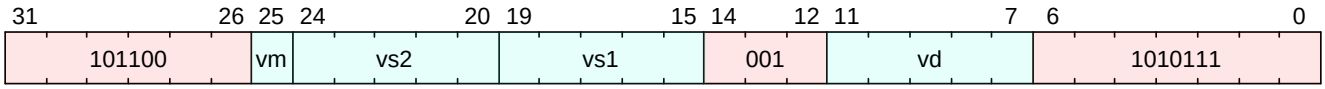
B.303. vfmacc.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.303.1. Encoding



B.303.2. Synopsis

No description available.

B.303.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.303.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.303.5. Execution

```

```

B.303.6. Exceptions

This instruction does not generate synchronous exceptions.

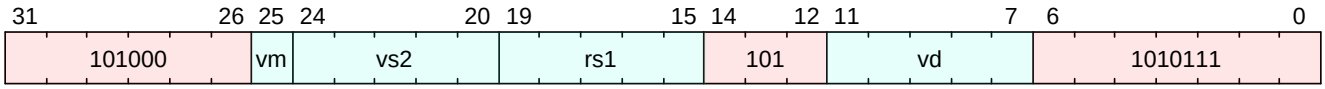
B.304. vfmadd.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.304.1. Encoding



B.304.2. Synopsis

No description available.

B.304.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.304.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.304.5. Execution

```

```

B.304.6. Exceptions

This instruction does not generate synchronous exceptions.

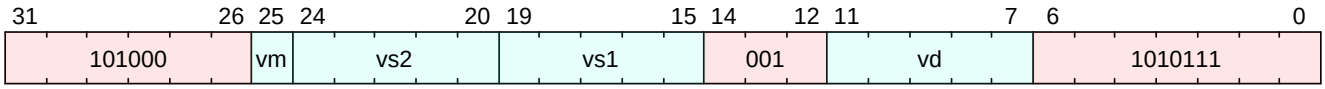
B.305. vfmadd.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.305.1. Encoding



B.305.2. Synopsis

No description available.

B.305.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.305.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.305.5. Execution

```

```

B.305.6. Exceptions

This instruction does not generate synchronous exceptions.

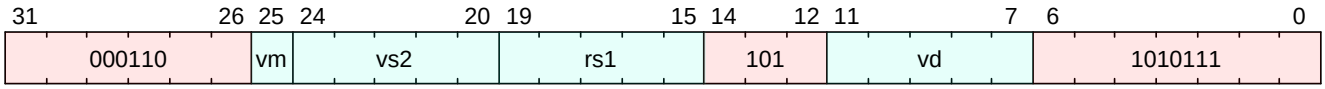
B.306. vfmax.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.306.1. Encoding



B.306.2. Synopsis

No description available.

B.306.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.306.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.306.5. Execution

```

```

B.306.6. Exceptions

This instruction does not generate synchronous exceptions.

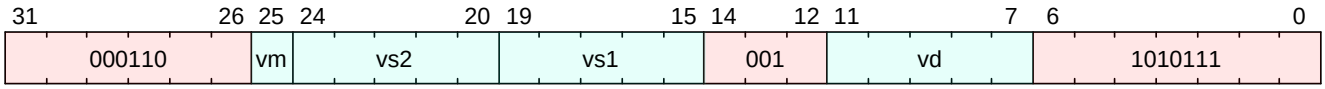
B.307. vfmax.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.307.1. Encoding



B.307.2. Synopsis

No description available.

B.307.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.307.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.307.5. Execution

```

```

B.307.6. Exceptions

This instruction does not generate synchronous exceptions.

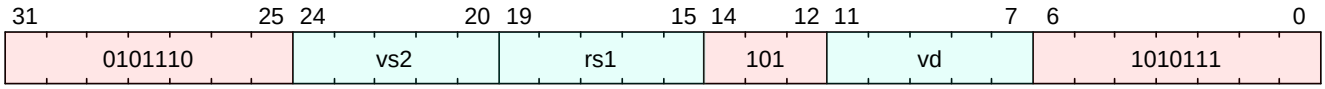
B.308. vfmerge.vfm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.308.1. Encoding



B.308.2. Synopsis

No description available.

B.308.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.308.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.308.5. Execution

```

```

B.308.6. Exceptions

This instruction does not generate synchronous exceptions.

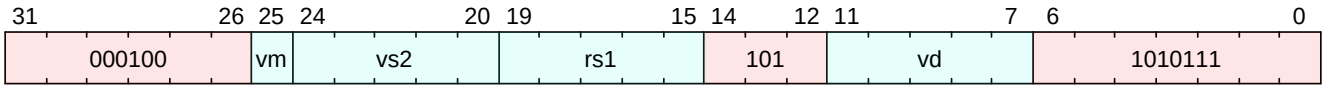
B.309. vfmin.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.309.1. Encoding



B.309.2. Synopsis

No description available.

B.309.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.309.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.309.5. Execution

```

```

B.309.6. Exceptions

This instruction does not generate synchronous exceptions.

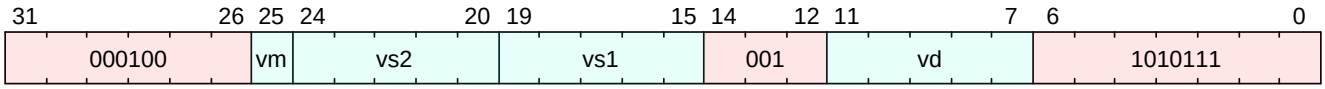
B.310. vfmin.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.310.1. Encoding



B.310.2. Synopsis

No description available.

B.310.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.310.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.310.5. Execution

```

```

B.310.6. Exceptions

This instruction does not generate synchronous exceptions.

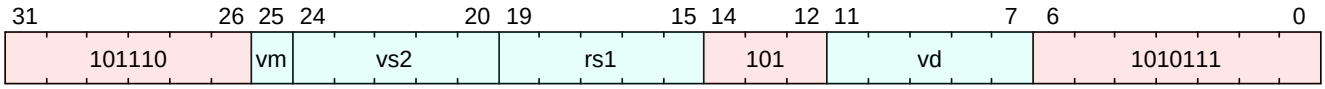
B.311. vfmsac.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.311.1. Encoding



B.311.2. Synopsis

No description available.

B.311.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.311.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.311.5. Execution

```

```

B.311.6. Exceptions

This instruction does not generate synchronous exceptions.

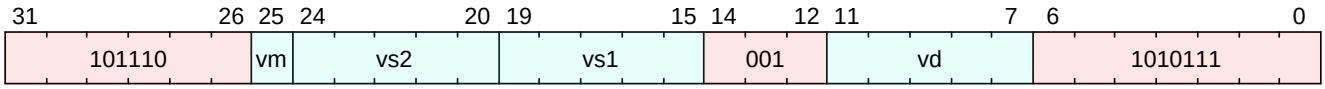
B.312. vfmsac.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.312.1. Encoding



B.312.2. Synopsis

No description available.

B.312.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.312.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.312.5. Execution

```

```

B.312.6. Exceptions

This instruction does not generate synchronous exceptions.

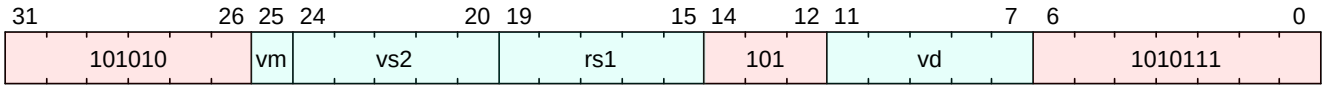
B.313. vfmsub.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.313.1. Encoding



B.313.2. Synopsis

No description available.

B.313.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.313.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.313.5. Execution

```

```

B.313.6. Exceptions

This instruction does not generate synchronous exceptions.

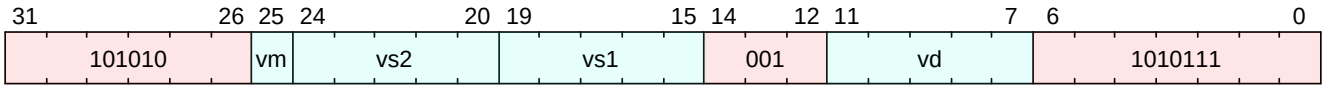
B.314. vfmsub.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.314.1. Encoding



B.314.2. Synopsis

No description available.

B.314.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.314.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.314.5. Execution

```

```

B.314.6. Exceptions

This instruction does not generate synchronous exceptions.

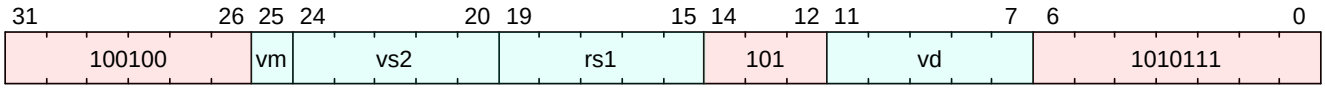
B.315. vfmul.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.315.1. Encoding



B.315.2. Synopsis

No description available.

B.315.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.315.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.315.5. Execution

```

```

B.315.6. Exceptions

This instruction does not generate synchronous exceptions.

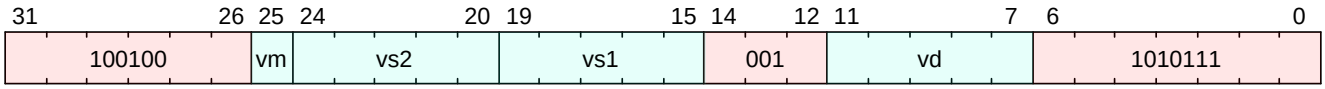
B.316. vfmul.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.316.1. Encoding



B.316.2. Synopsis

No description available.

B.316.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.316.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.316.5. Execution

```

```

B.316.6. Exceptions

This instruction does not generate synchronous exceptions.

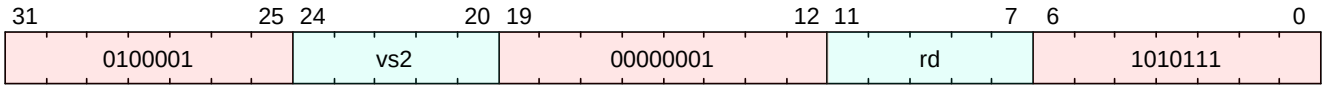
B.317. vfmv.f.s

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.317.1. Encoding



B.317.2. Synopsis

No description available.

B.317.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.317.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> rd = $encoding[11:7];
```

B.317.5. Execution

```

```

B.317.6. Exceptions

This instruction does not generate synchronous exceptions.

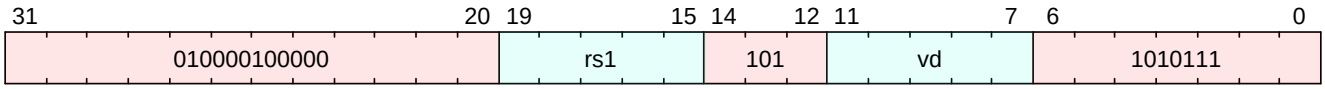
B.318. vfmv.s.f

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.318.1. Encoding



B.318.2. Synopsis

No description available.

B.318.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.318.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.318.5. Execution

```

```

B.318.6. Exceptions

This instruction does not generate synchronous exceptions.

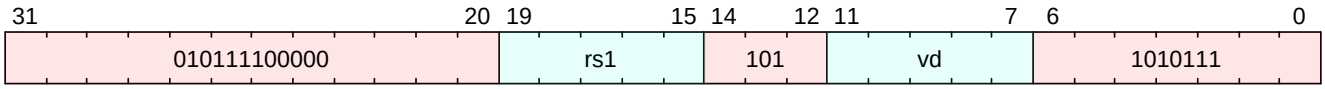
B.319. vfmv.v.f

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.319.1. Encoding



B.319.2. Synopsis

No description available.

B.319.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.319.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.319.5. Execution

```

```

B.319.6. Exceptions

This instruction does not generate synchronous exceptions.

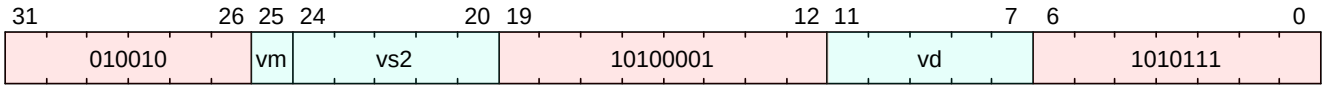
B.320. vfncvt.f.f.w

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.320.1. Encoding



B.320.2. Synopsis

No description available.

B.320.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.320.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.320.5. Execution

```

```

B.320.6. Exceptions

This instruction does not generate synchronous exceptions.

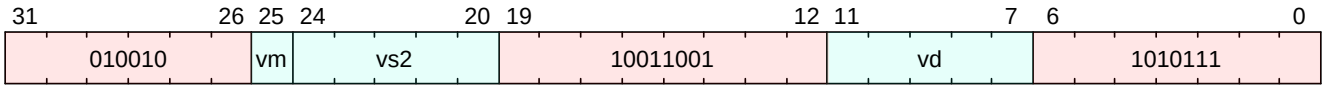
B.321. vfncvt.f.x.w

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.321.1. Encoding



B.321.2. Synopsis

No description available.

B.321.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.321.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.321.5. Execution

```

```

B.321.6. Exceptions

This instruction does not generate synchronous exceptions.

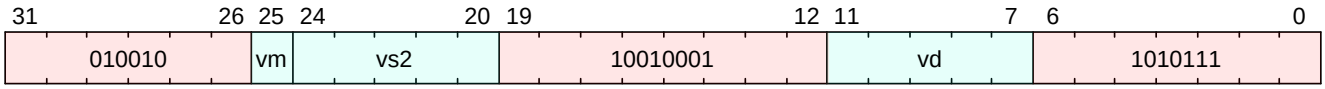
B.322. vfncvt.f.xu.w

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.322.1. Encoding



B.322.2. Synopsis

No description available.

B.322.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.322.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.322.5. Execution

```

```

B.322.6. Exceptions

This instruction does not generate synchronous exceptions.

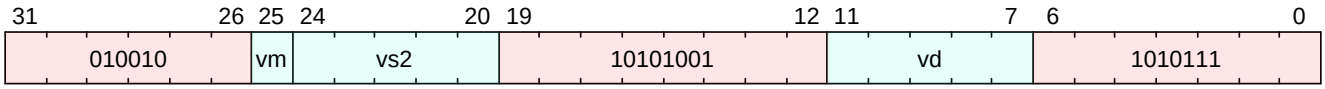
B.323. vfncvt.rod.f.f.w

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.323.1. Encoding



B.323.2. Synopsis

No description available.

B.323.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.323.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.323.5. Execution

```

```

B.323.6. Exceptions

This instruction does not generate synchronous exceptions.

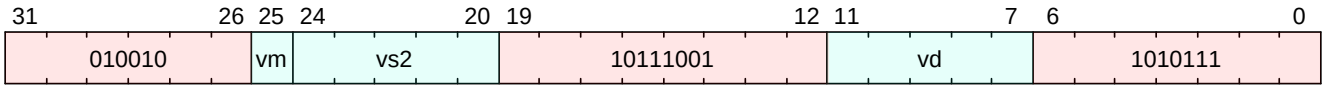
B.324. vfncvt.rtz.x.f.w

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.324.1. Encoding



B.324.2. Synopsis

No description available.

B.324.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.324.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.324.5. Execution

```

```

B.324.6. Exceptions

This instruction does not generate synchronous exceptions.

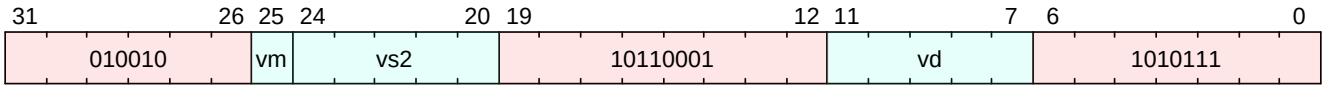
B.325. vfncvt.rtz.xu.f.w

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.325.1. Encoding



B.325.2. Synopsis

No description available.

B.325.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.325.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.325.5. Execution

```

```

B.325.6. Exceptions

This instruction does not generate synchronous exceptions.

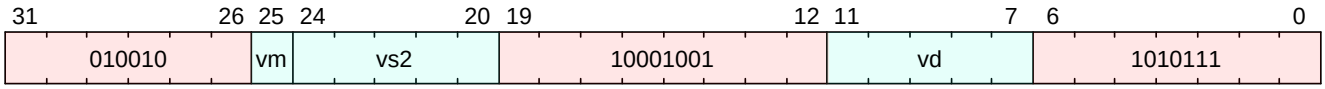
B.326. vfncvt.x.f.w

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.326.1. Encoding



B.326.2. Synopsis

No description available.

B.326.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.326.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.326.5. Execution

```

```

B.326.6. Exceptions

This instruction does not generate synchronous exceptions.

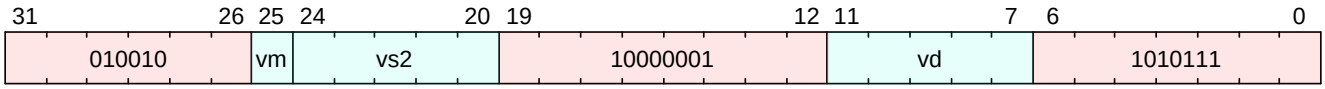
B.327. vfncvt.xu.f.w

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.327.1. Encoding



B.327.2. Synopsis

No description available.

B.327.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.327.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.327.5. Execution

```

```

B.327.6. Exceptions

This instruction does not generate synchronous exceptions.

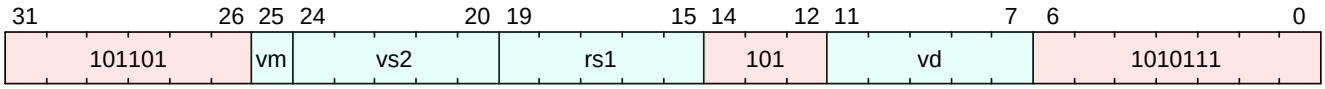
B.328. vfnmacc.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.328.1. Encoding



B.328.2. Synopsis

No description available.

B.328.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.328.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.328.5. Execution

```

```

B.328.6. Exceptions

This instruction does not generate synchronous exceptions.

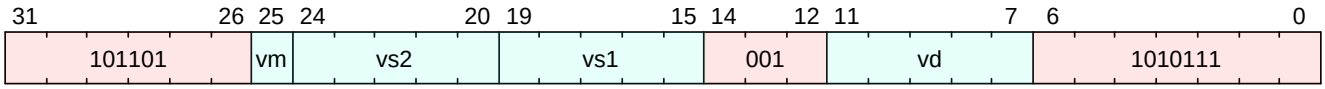
B.329. vfnmacc.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.329.1. Encoding



B.329.2. Synopsis

No description available.

B.329.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.329.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.329.5. Execution

```

```

B.329.6. Exceptions

This instruction does not generate synchronous exceptions.

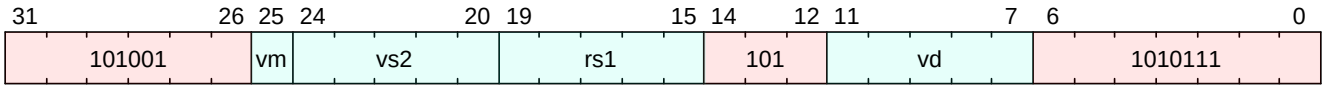
B.330. vfnmadd.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.330.1. Encoding



B.330.2. Synopsis

No description available.

B.330.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.330.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.330.5. Execution

```

```

B.330.6. Exceptions

This instruction does not generate synchronous exceptions.

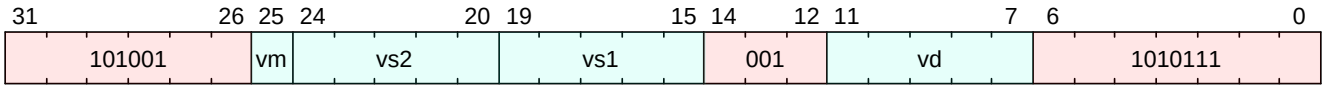
B.331. vfnmadd.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.331.1. Encoding



B.331.2. Synopsis

No description available.

B.331.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.331.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.331.5. Execution

```

```

B.331.6. Exceptions

This instruction does not generate synchronous exceptions.

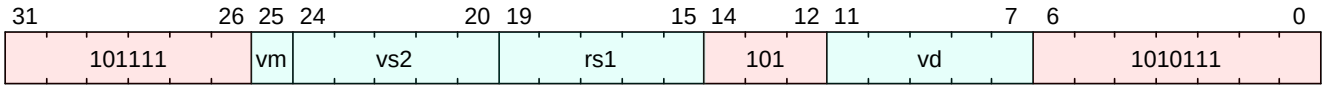
B.332. vfnmsac.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.332.1. Encoding



B.332.2. Synopsis

No description available.

B.332.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.332.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.332.5. Execution

```

```

B.332.6. Exceptions

This instruction does not generate synchronous exceptions.

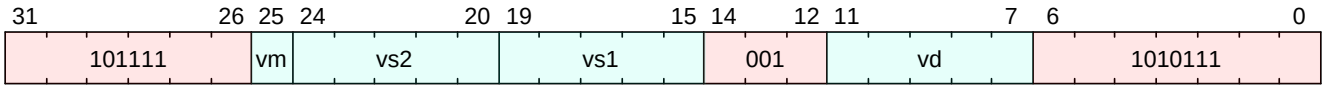
B.333. vfnmsac.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.333.1. Encoding



B.333.2. Synopsis

No description available.

B.333.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.333.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.333.5. Execution

```

```

B.333.6. Exceptions

This instruction does not generate synchronous exceptions.

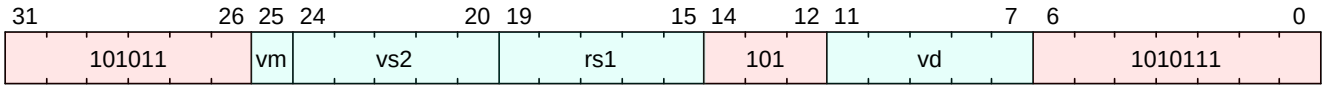
B.334. vfnmsub.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.334.1. Encoding



B.334.2. Synopsis

No description available.

B.334.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.334.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.334.5. Execution

```

```

B.334.6. Exceptions

This instruction does not generate synchronous exceptions.

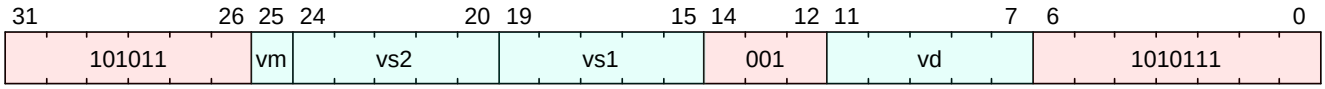
B.335. vfnmsub.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.335.1. Encoding



B.335.2. Synopsis

No description available.

B.335.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.335.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.335.5. Execution

```

```

B.335.6. Exceptions

This instruction does not generate synchronous exceptions.

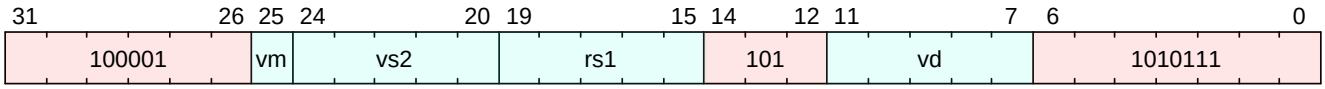
B.336. vfrdiv.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.336.1. Encoding



B.336.2. Synopsis

No description available.

B.336.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.336.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.336.5. Execution

```

```

B.336.6. Exceptions

This instruction does not generate synchronous exceptions.

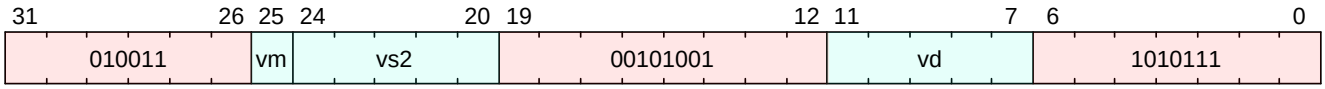
B.337. vfrec7.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.337.1. Encoding



B.337.2. Synopsis

No description available.

B.337.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.337.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.337.5. Execution

```

```

B.337.6. Exceptions

This instruction does not generate synchronous exceptions.

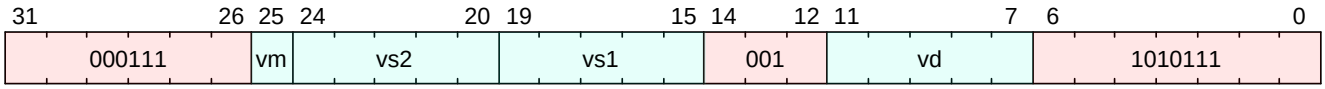
B.338. vfredmax.vs

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.338.1. Encoding



B.338.2. Synopsis

No description available.

B.338.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.338.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.338.5. Execution

```

```

B.338.6. Exceptions

This instruction does not generate synchronous exceptions.

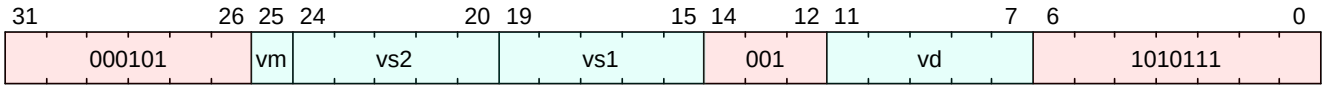
B.339. vfredmin.vs

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.339.1. Encoding



B.339.2. Synopsis

No description available.

B.339.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.339.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.339.5. Execution

```

```

B.339.6. Exceptions

This instruction does not generate synchronous exceptions.

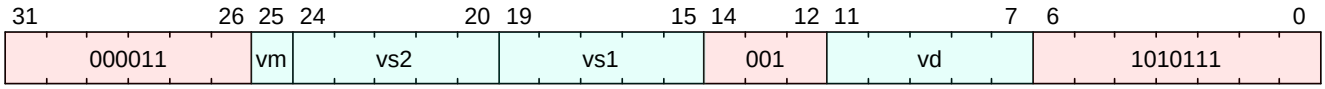
B.340. vfredosum.vs

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.340.1. Encoding



B.340.2. Synopsis

No description available.

B.340.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.340.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.340.5. Execution

```

```

B.340.6. Exceptions

This instruction does not generate synchronous exceptions.

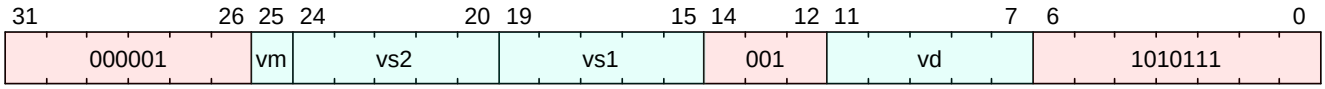
B.341. vfredusum.vs

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.341.1. Encoding



B.341.2. Synopsis

No description available.

B.341.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.341.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.341.5. Execution

```

```

B.341.6. Exceptions

This instruction does not generate synchronous exceptions.

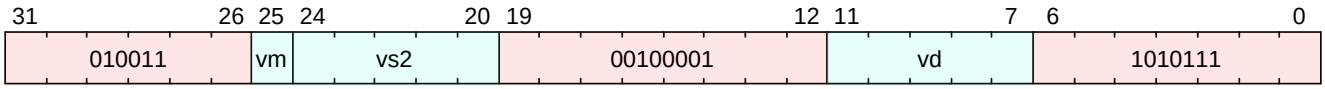
B.342. vfrsqrt7.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.342.1. Encoding



B.342.2. Synopsis

No description available.

B.342.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.342.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.342.5. Execution

```

```

B.342.6. Exceptions

This instruction does not generate synchronous exceptions.

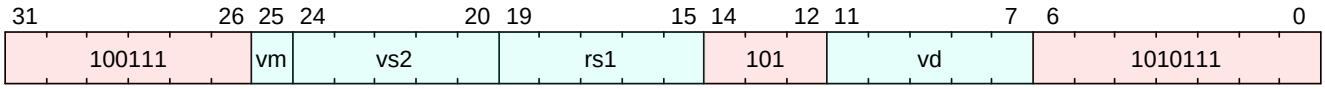
B.343. vfrsub.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.343.1. Encoding



B.343.2. Synopsis

No description available.

B.343.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.343.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.343.5. Execution

```

```

B.343.6. Exceptions

This instruction does not generate synchronous exceptions.

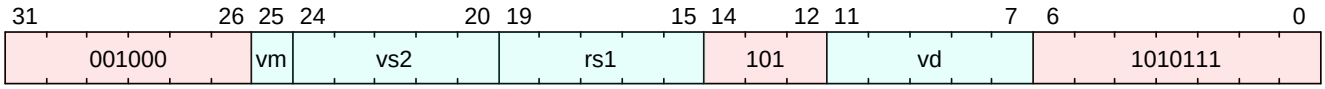
B.344. vfsgnj.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.344.1. Encoding



B.344.2. Synopsis

No description available.

B.344.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.344.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.344.5. Execution

```

```

B.344.6. Exceptions

This instruction does not generate synchronous exceptions.

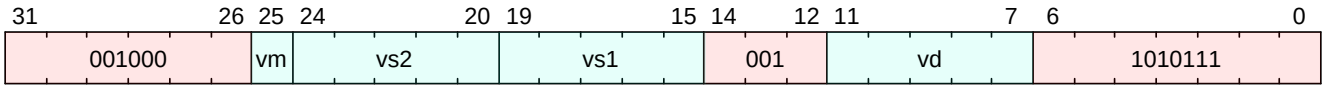
B.345. vfsgnj.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.345.1. Encoding



B.345.2. Synopsis

No description available.

B.345.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.345.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.345.5. Execution

```

```

B.345.6. Exceptions

This instruction does not generate synchronous exceptions.

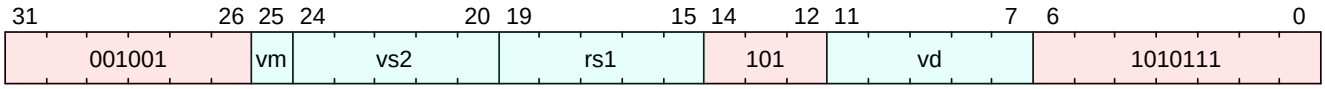
B.346. vfsgnjn.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.346.1. Encoding



B.346.2. Synopsis

No description available.

B.346.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.346.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.346.5. Execution

```

```

B.346.6. Exceptions

This instruction does not generate synchronous exceptions.

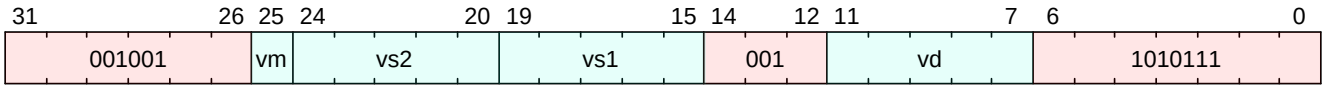
B.347. vfsgnjn.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.347.1. Encoding



B.347.2. Synopsis

No description available.

B.347.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.347.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.347.5. Execution

```

```

B.347.6. Exceptions

This instruction does not generate synchronous exceptions.

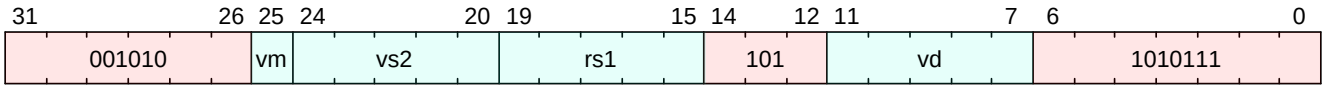
B.348. vfsgnjx.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.348.1. Encoding



B.348.2. Synopsis

No description available.

B.348.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.348.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.348.5. Execution

```

```

B.348.6. Exceptions

This instruction does not generate synchronous exceptions.

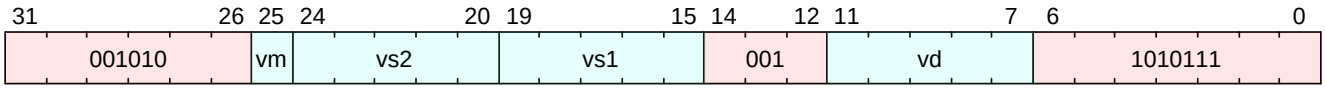
B.349. vfsgnjx.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.349.1. Encoding



B.349.2. Synopsis

No description available.

B.349.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.349.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.349.5. Execution

```

```

B.349.6. Exceptions

This instruction does not generate synchronous exceptions.

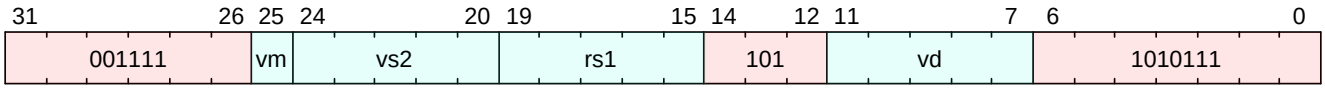
B.350. vfslide1down.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.350.1. Encoding



B.350.2. Synopsis

No description available.

B.350.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.350.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.350.5. Execution

```

```

B.350.6. Exceptions

This instruction does not generate synchronous exceptions.

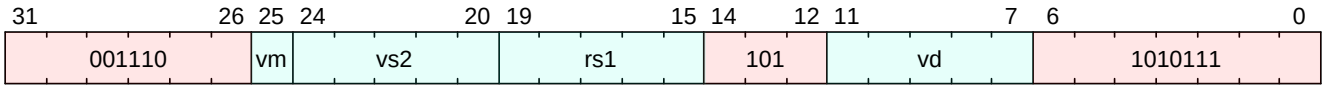
B.351. vfslide1up.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.351.1. Encoding



B.351.2. Synopsis

No description available.

B.351.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.351.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.351.5. Execution

```

```

B.351.6. Exceptions

This instruction does not generate synchronous exceptions.

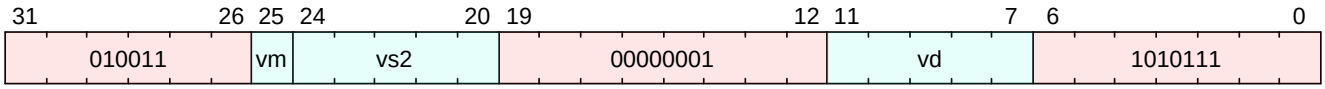
B.352. vfsqrt.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.352.1. Encoding



B.352.2. Synopsis

No description available.

B.352.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.352.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.352.5. Execution

```

```

B.352.6. Exceptions

This instruction does not generate synchronous exceptions.

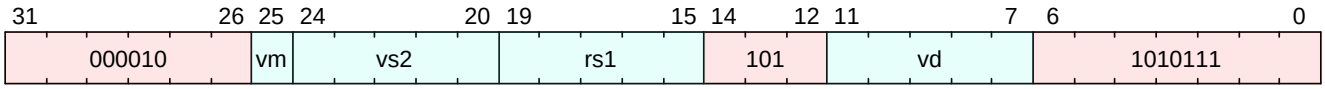
B.353. vfsub.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.353.1. Encoding



B.353.2. Synopsis

No description available.

B.353.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.353.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.353.5. Execution

```

```

B.353.6. Exceptions

This instruction does not generate synchronous exceptions.

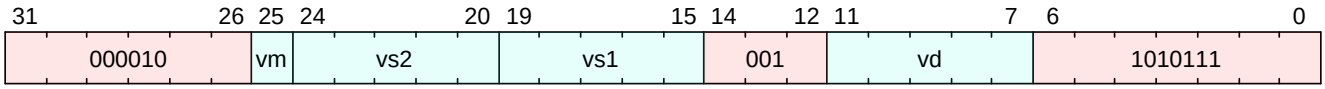
B.354. vfsub.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.354.1. Encoding



B.354.2. Synopsis

No description available.

B.354.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.354.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.354.5. Execution

```

```

B.354.6. Exceptions

This instruction does not generate synchronous exceptions.

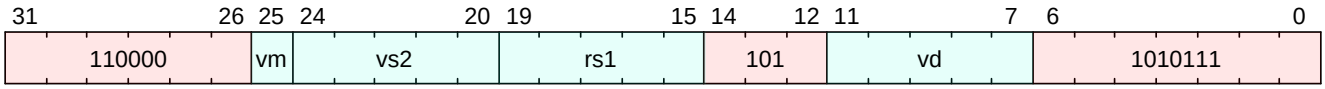
B.355. vfwadd.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.355.1. Encoding



B.355.2. Synopsis

No description available.

B.355.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.355.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.355.5. Execution

```

```

B.355.6. Exceptions

This instruction does not generate synchronous exceptions.

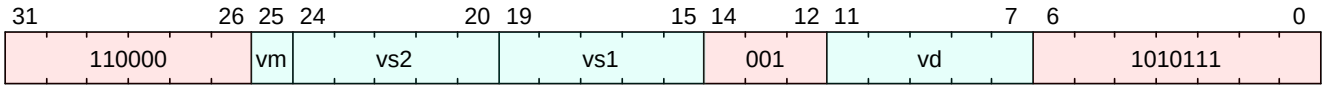
B.356. vfwadd.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.356.1. Encoding



B.356.2. Synopsis

No description available.

B.356.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.356.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.356.5. Execution

```

```

B.356.6. Exceptions

This instruction does not generate synchronous exceptions.

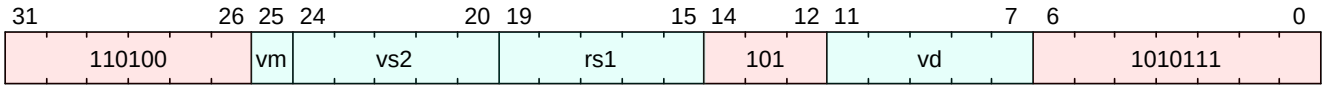
B.357. vfwadd.wf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.357.1. Encoding



B.357.2. Synopsis

No description available.

B.357.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.357.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.357.5. Execution

```

```

B.357.6. Exceptions

This instruction does not generate synchronous exceptions.

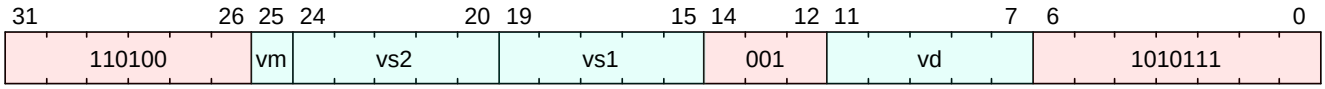
B.358. vfwadd.wv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.358.1. Encoding



B.358.2. Synopsis

No description available.

B.358.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.358.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.358.5. Execution

```

```

B.358.6. Exceptions

This instruction does not generate synchronous exceptions.

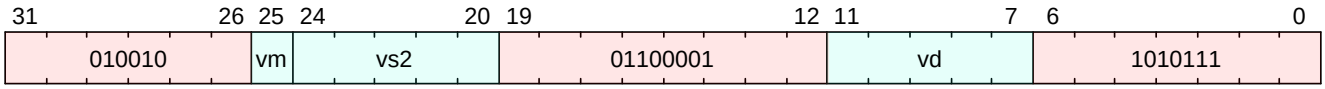
B.359. vfwcvt.f.f.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.359.1. Encoding



B.359.2. Synopsis

No description available.

B.359.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.359.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.359.5. Execution

```

```

B.359.6. Exceptions

This instruction does not generate synchronous exceptions.

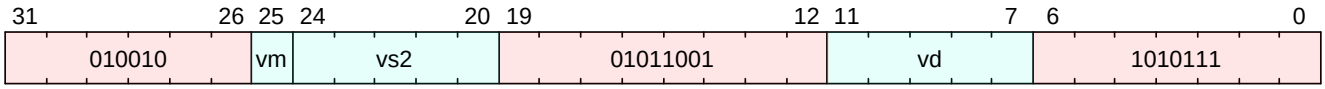
B.360. vfwcvt.f.x.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.360.1. Encoding



B.360.2. Synopsis

No description available.

B.360.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.360.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.360.5. Execution

```

```

B.360.6. Exceptions

This instruction does not generate synchronous exceptions.

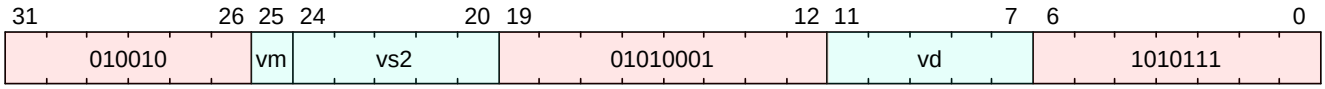
B.361. vfwcvt.f.xu.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.361.1. Encoding



B.361.2. Synopsis

No description available.

B.361.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.361.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.361.5. Execution

```

```

B.361.6. Exceptions

This instruction does not generate synchronous exceptions.

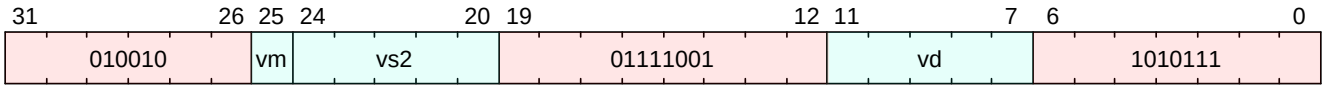
B.362. vfwcvt.rtz.x.f.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.362.1. Encoding



B.362.2. Synopsis

No description available.

B.362.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.362.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.362.5. Execution

```

```

B.362.6. Exceptions

This instruction does not generate synchronous exceptions.

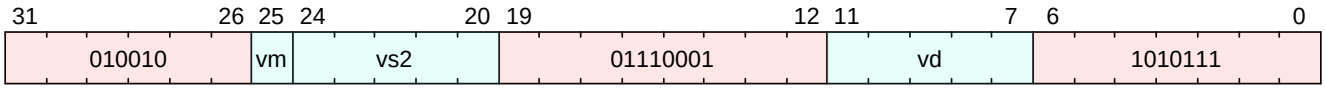
B.363. vfwcvt.rtz.xu.f.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.363.1. Encoding



B.363.2. Synopsis

No description available.

B.363.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.363.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.363.5. Execution

```

```

B.363.6. Exceptions

This instruction does not generate synchronous exceptions.

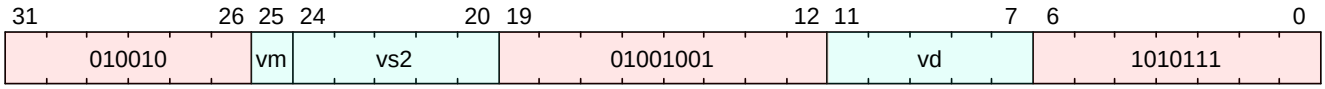
B.364. vfwcvt.x.f.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.364.1. Encoding



B.364.2. Synopsis

No description available.

B.364.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.364.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.364.5. Execution

```

```

B.364.6. Exceptions

This instruction does not generate synchronous exceptions.

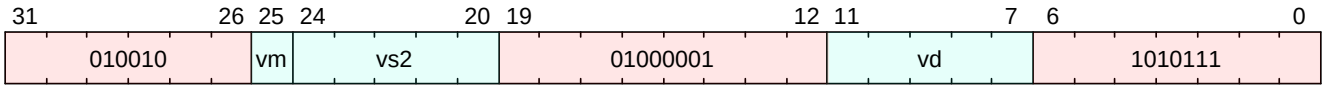
B.365. vfwcvt.xu.f.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.365.1. Encoding



B.365.2. Synopsis

No description available.

B.365.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.365.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.365.5. Execution

```

```

B.365.6. Exceptions

This instruction does not generate synchronous exceptions.

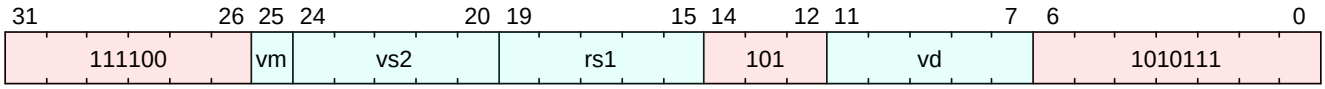
B.366. vfwmacc.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.366.1. Encoding



B.366.2. Synopsis

No description available.

B.366.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.366.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.366.5. Execution

```

```

B.366.6. Exceptions

This instruction does not generate synchronous exceptions.

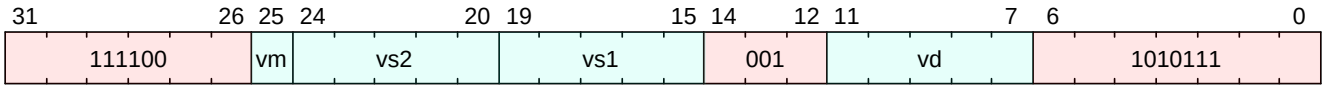
B.367. vfwmacc.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.367.1. Encoding



B.367.2. Synopsis

No description available.

B.367.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.367.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.367.5. Execution

```

```

B.367.6. Exceptions

This instruction does not generate synchronous exceptions.

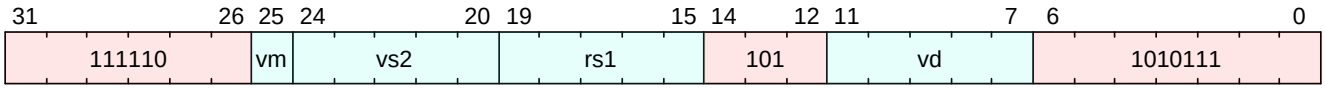
B.368. vfwmsac.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.368.1. Encoding



B.368.2. Synopsis

No description available.

B.368.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.368.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.368.5. Execution

```

```

B.368.6. Exceptions

This instruction does not generate synchronous exceptions.

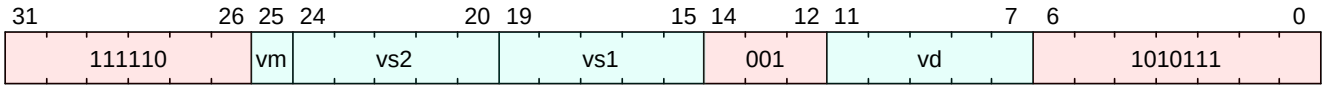
B.369. vfwmsac.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.369.1. Encoding



B.369.2. Synopsis

No description available.

B.369.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.369.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.369.5. Execution

```

```

B.369.6. Exceptions

This instruction does not generate synchronous exceptions.

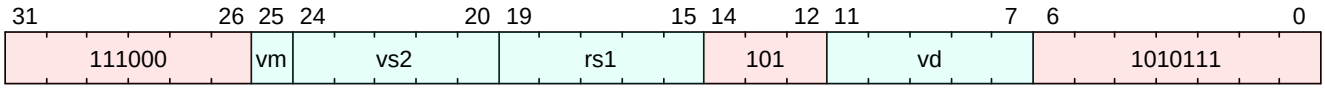
B.370. vfwmul.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.370.1. Encoding



B.370.2. Synopsis

No description available.

B.370.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.370.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.370.5. Execution

```

```

B.370.6. Exceptions

This instruction does not generate synchronous exceptions.

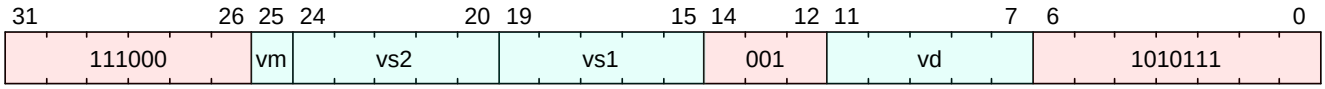
B.371. vfwmul.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.371.1. Encoding



B.371.2. Synopsis

No description available.

B.371.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.371.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.371.5. Execution

```

```

B.371.6. Exceptions

This instruction does not generate synchronous exceptions.

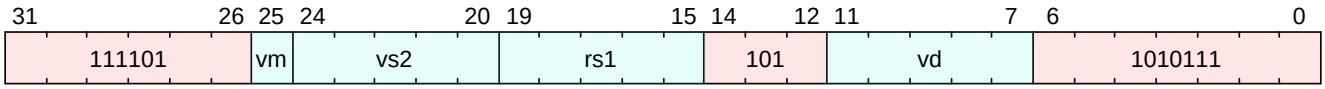
B.372. vfwnmacc.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.372.1. Encoding



B.372.2. Synopsis

No description available.

B.372.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.372.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.372.5. Execution

```

```

B.372.6. Exceptions

This instruction does not generate synchronous exceptions.

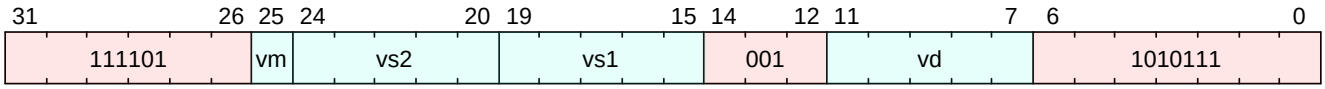
B.373. vfwnmacc.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.373.1. Encoding



B.373.2. Synopsis

No description available.

B.373.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.373.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.373.5. Execution

```

```

B.373.6. Exceptions

This instruction does not generate synchronous exceptions.

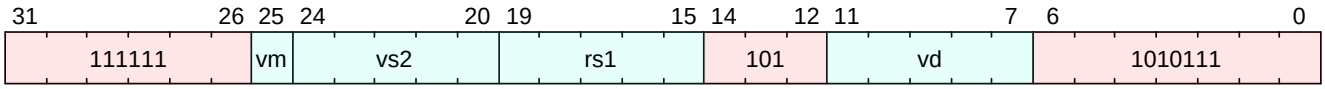
B.374. vfwnmsac.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.374.1. Encoding



B.374.2. Synopsis

No description available.

B.374.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.374.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.374.5. Execution

```

```

B.374.6. Exceptions

This instruction does not generate synchronous exceptions.

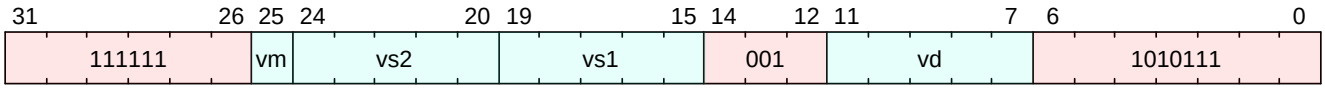
B.375. vfwnmsac.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.375.1. Encoding



B.375.2. Synopsis

No description available.

B.375.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.375.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.375.5. Execution

```

```

B.375.6. Exceptions

This instruction does not generate synchronous exceptions.

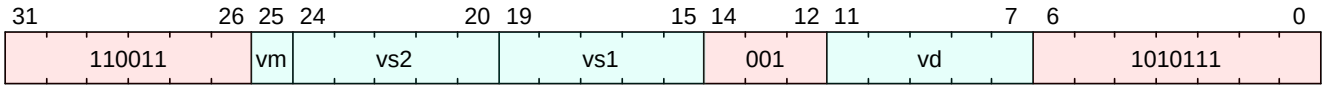
B.376. vfwredosum.vs

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.376.1. Encoding



B.376.2. Synopsis

No description available.

B.376.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.376.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.376.5. Execution

```

```

B.376.6. Exceptions

This instruction does not generate synchronous exceptions.

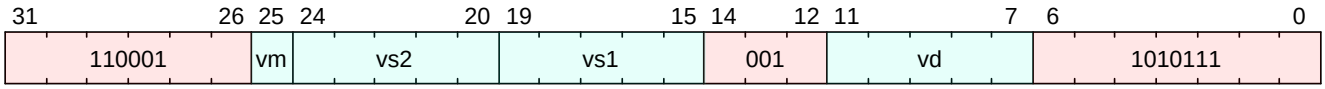
B.377. vfwredusum.vs

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.377.1. Encoding



B.377.2. Synopsis

No description available.

B.377.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.377.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.377.5. Execution

```

```

B.377.6. Exceptions

This instruction does not generate synchronous exceptions.

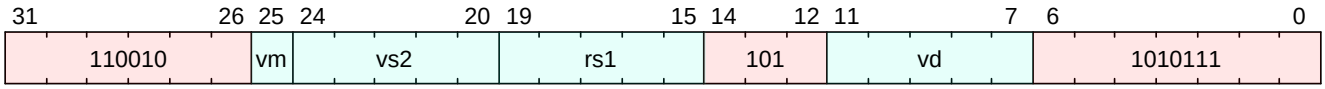
B.378. vfwsub.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.378.1. Encoding



B.378.2. Synopsis

No description available.

B.378.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.378.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.378.5. Execution

```

```

B.378.6. Exceptions

This instruction does not generate synchronous exceptions.

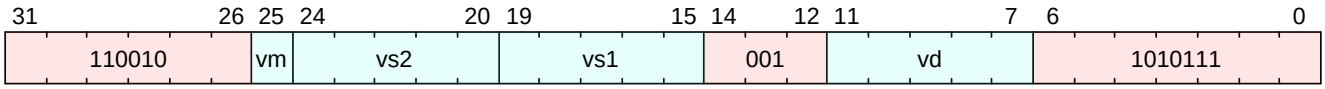
B.379. vfwsub.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.379.1. Encoding



B.379.2. Synopsis

No description available.

B.379.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.379.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.379.5. Execution

```

```

B.379.6. Exceptions

This instruction does not generate synchronous exceptions.

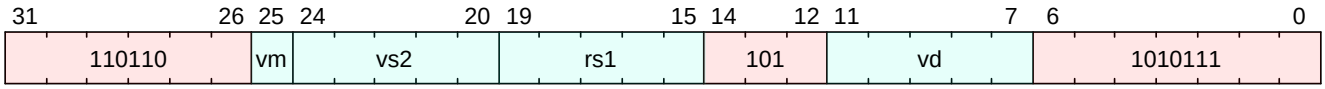
B.380. vfwsub.wf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.380.1. Encoding



B.380.2. Synopsis

No description available.

B.380.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.380.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.380.5. Execution

```

```

B.380.6. Exceptions

This instruction does not generate synchronous exceptions.

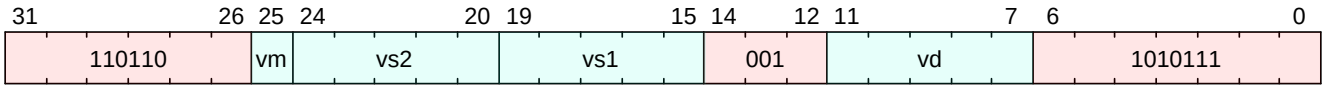
B.381. vfwsub.wv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.381.1. Encoding



B.381.2. Synopsis

No description available.

B.381.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.381.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.381.5. Execution

```

```

B.381.6. Exceptions

This instruction does not generate synchronous exceptions.

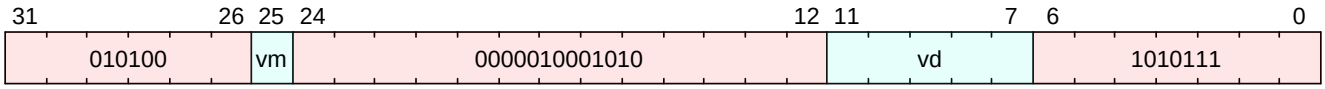
B.382. vid.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.382.1. Encoding



B.382.2. Synopsis

No description available.

B.382.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.382.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vd = $encoding[11:7];
```

B.382.5. Execution

```

```

B.382.6. Exceptions

This instruction does not generate synchronous exceptions.

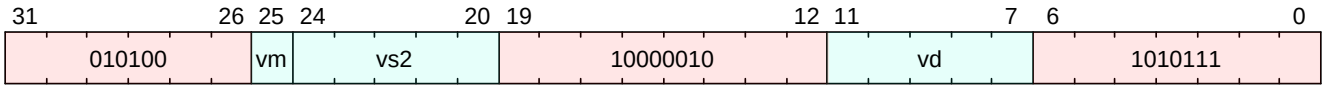
B.383. viota.m

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.383.1. Encoding



B.383.2. Synopsis

No description available.

B.383.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.383.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.383.5. Execution

```

```

B.383.6. Exceptions

This instruction does not generate synchronous exceptions.

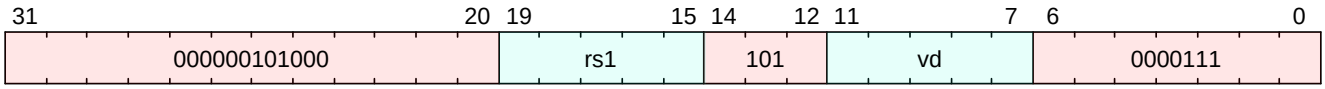
B.384. vl1re16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.384.1. Encoding



B.384.2. Synopsis

No description available.

B.384.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.384.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.384.5. Execution

```

```

B.384.6. Exceptions

This instruction does not generate synchronous exceptions.

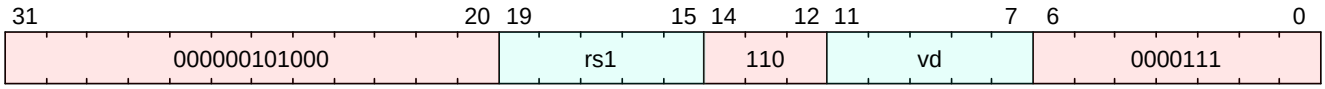
B.385. vl1re32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.385.1. Encoding



B.385.2. Synopsis

No description available.

B.385.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.385.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.385.5. Execution

```

```

B.385.6. Exceptions

This instruction does not generate synchronous exceptions.

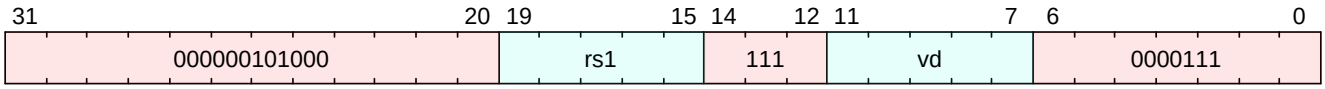
B.386. vl1re64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.386.1. Encoding



B.386.2. Synopsis

No description available.

B.386.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.386.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.386.5. Execution

```

```

B.386.6. Exceptions

This instruction does not generate synchronous exceptions.

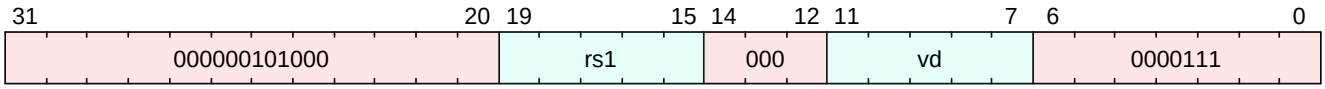
B.387. vl1re8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.387.1. Encoding



B.387.2. Synopsis

No description available.

B.387.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.387.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.387.5. Execution

```

```

B.387.6. Exceptions

This instruction does not generate synchronous exceptions.

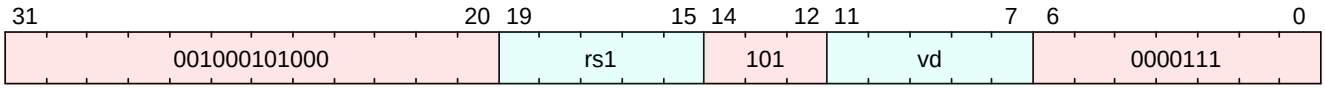
B.388. vl2re16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.388.1. Encoding



B.388.2. Synopsis

No description available.

B.388.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.388.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.388.5. Execution

```

```

B.388.6. Exceptions

This instruction does not generate synchronous exceptions.

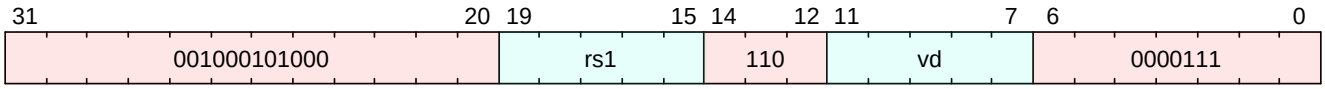
B.389. vl2re32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.389.1. Encoding



B.389.2. Synopsis

No description available.

B.389.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.389.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.389.5. Execution

```

```

B.389.6. Exceptions

This instruction does not generate synchronous exceptions.

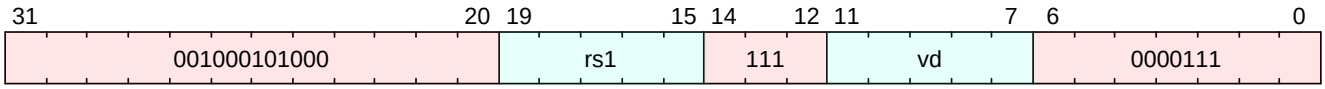
B.390. vl2re64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.390.1. Encoding



B.390.2. Synopsis

No description available.

B.390.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.390.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.390.5. Execution

```

```

B.390.6. Exceptions

This instruction does not generate synchronous exceptions.

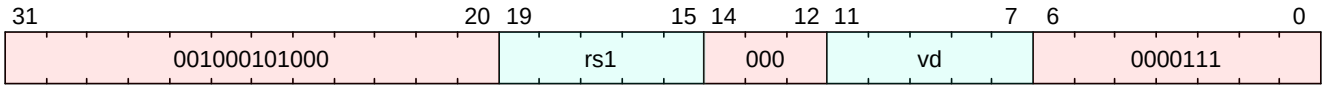
B.391. vl2re8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.391.1. Encoding



B.391.2. Synopsis

No description available.

B.391.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.391.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.391.5. Execution

```

```

B.391.6. Exceptions

This instruction does not generate synchronous exceptions.

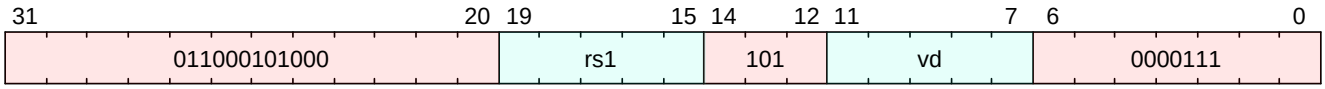
B.392. vl4re16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.392.1. Encoding



B.392.2. Synopsis

No description available.

B.392.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.392.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.392.5. Execution

```

```

B.392.6. Exceptions

This instruction does not generate synchronous exceptions.

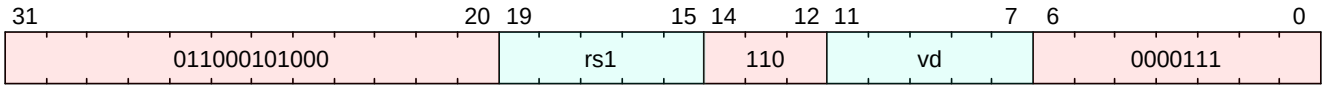
B.393. vl4re32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.393.1. Encoding



B.393.2. Synopsis

No description available.

B.393.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.393.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.393.5. Execution

```

```

B.393.6. Exceptions

This instruction does not generate synchronous exceptions.

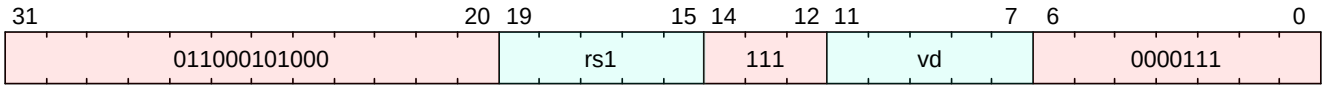
B.394. vl4re64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.394.1. Encoding



B.394.2. Synopsis

No description available.

B.394.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.394.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.394.5. Execution

```

```

B.394.6. Exceptions

This instruction does not generate synchronous exceptions.

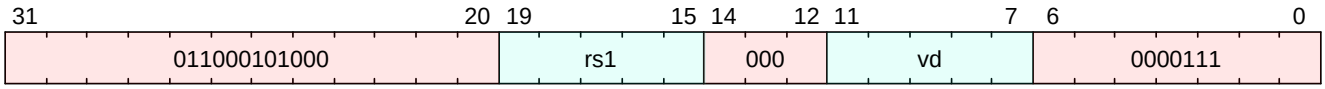
B.395. vl4re8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.395.1. Encoding



B.395.2. Synopsis

No description available.

B.395.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.395.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.395.5. Execution

```

```

B.395.6. Exceptions

This instruction does not generate synchronous exceptions.

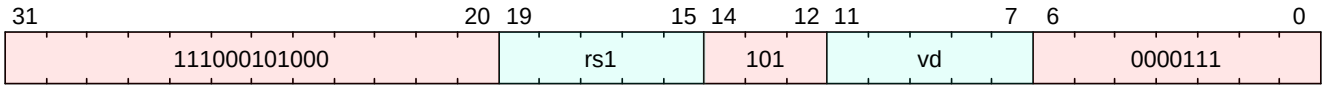
B.396. vl8re16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.396.1. Encoding



B.396.2. Synopsis

No description available.

B.396.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.396.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.396.5. Execution

```

```

B.396.6. Exceptions

This instruction does not generate synchronous exceptions.

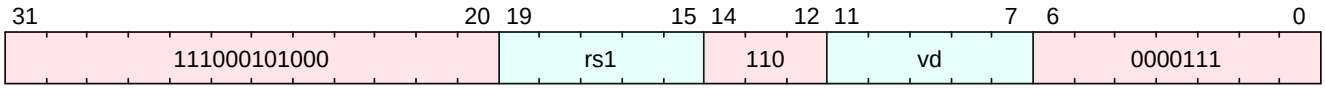
B.397. vl8re32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.397.1. Encoding



B.397.2. Synopsis

No description available.

B.397.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.397.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.397.5. Execution

```

```

B.397.6. Exceptions

This instruction does not generate synchronous exceptions.

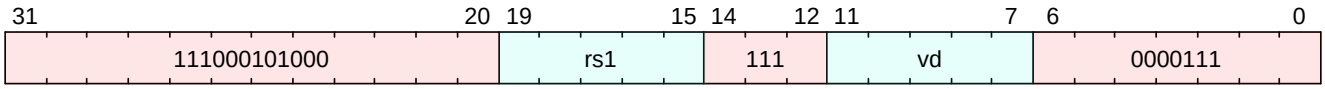
B.398. vl8re64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.398.1. Encoding



B.398.2. Synopsis

No description available.

B.398.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.398.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.398.5. Execution

```

```

B.398.6. Exceptions

This instruction does not generate synchronous exceptions.

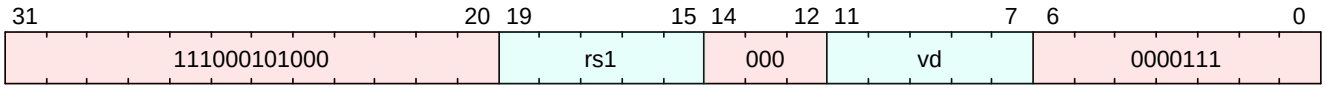
B.399. vl8re8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.399.1. Encoding



B.399.2. Synopsis

No description available.

B.399.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.399.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.399.5. Execution

```

```

B.399.6. Exceptions

This instruction does not generate synchronous exceptions.

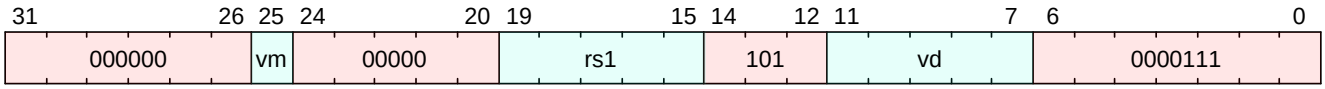
B.400. vle16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.400.1. Encoding



B.400.2. Synopsis

No description available.

B.400.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.400.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.400.5. Execution

```

```

B.400.6. Exceptions

This instruction does not generate synchronous exceptions.

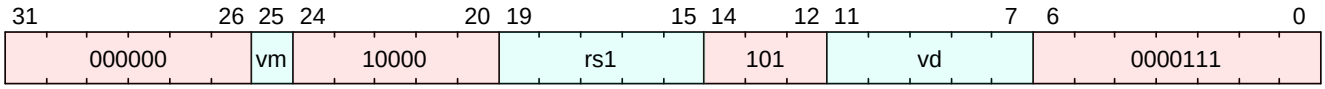
B.401. vle16ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.401.1. Encoding



B.401.2. Synopsis

No description available.

B.401.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.401.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.401.5. Execution

```

```

B.401.6. Exceptions

This instruction does not generate synchronous exceptions.

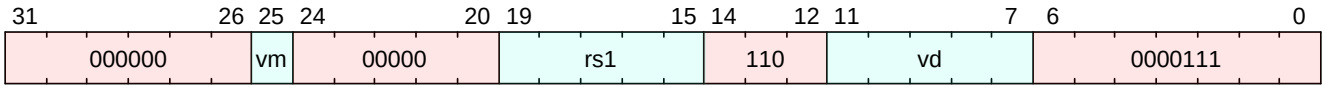
B.402. vle32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.402.1. Encoding



B.402.2. Synopsis

No description available.

B.402.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.402.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.402.5. Execution

```

```

B.402.6. Exceptions

This instruction does not generate synchronous exceptions.

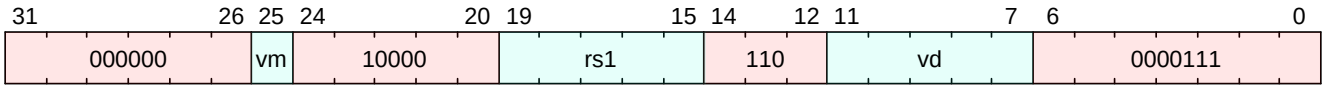
B.403. vle32ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.403.1. Encoding



B.403.2. Synopsis

No description available.

B.403.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.403.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.403.5. Execution

```

```

B.403.6. Exceptions

This instruction does not generate synchronous exceptions.

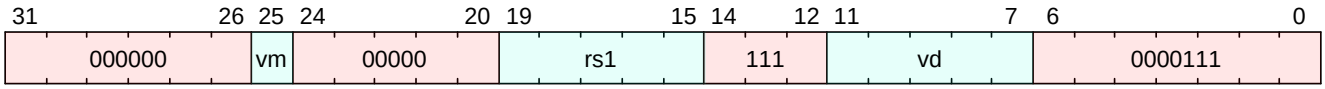
B.404. vle64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.404.1. Encoding



B.404.2. Synopsis

No description available.

B.404.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.404.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.404.5. Execution

```

```

B.404.6. Exceptions

This instruction does not generate synchronous exceptions.

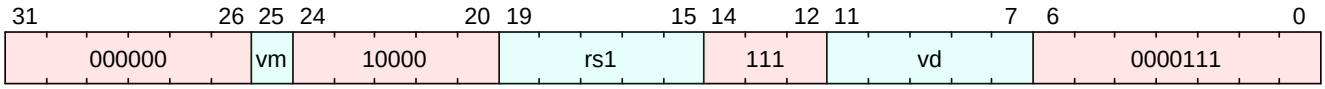
B.405. vle64ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.405.1. Encoding



B.405.2. Synopsis

No description available.

B.405.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.405.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.405.5. Execution

```

```

B.405.6. Exceptions

This instruction does not generate synchronous exceptions.

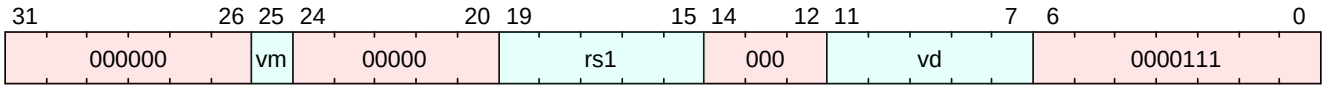
B.406. vle8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.406.1. Encoding



B.406.2. Synopsis

No description available.

B.406.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.406.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.406.5. Execution

```

```

B.406.6. Exceptions

This instruction does not generate synchronous exceptions.

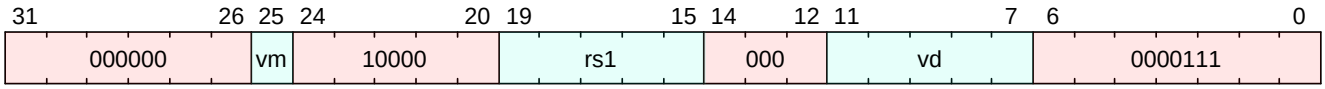
B.407. vle8ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.407.1. Encoding



B.407.2. Synopsis

No description available.

B.407.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.407.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.407.5. Execution

```

```

B.407.6. Exceptions

This instruction does not generate synchronous exceptions.

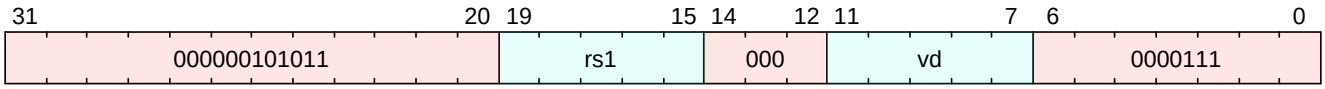
B.408. vlm.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.408.1. Encoding



B.408.2. Synopsis

No description available.

B.408.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.408.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.408.5. Execution

```

```

B.408.6. Exceptions

This instruction does not generate synchronous exceptions.

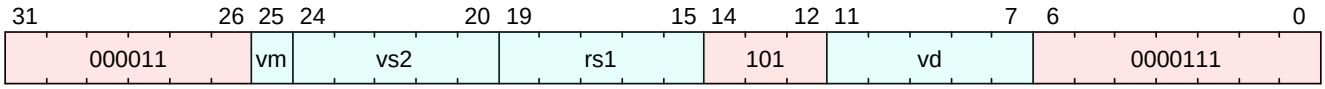
B.409. vloxei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.409.1. Encoding



B.409.2. Synopsis

No description available.

B.409.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.409.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.409.5. Execution

```

```

B.409.6. Exceptions

This instruction does not generate synchronous exceptions.

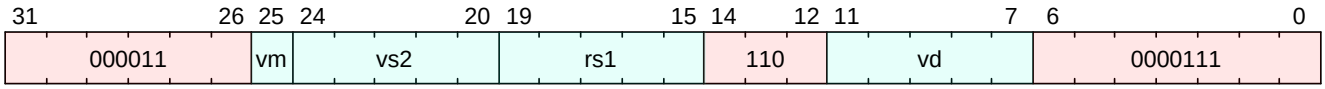
B.410. vloxei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.410.1. Encoding



B.410.2. Synopsis

No description available.

B.410.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.410.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.410.5. Execution

```

```

B.410.6. Exceptions

This instruction does not generate synchronous exceptions.

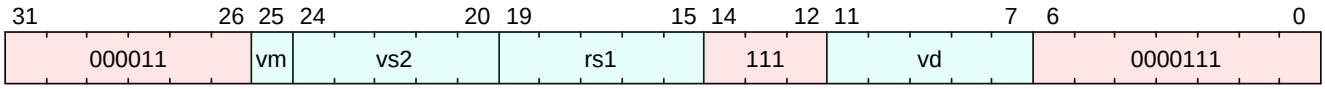
B.411. vloxei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.411.1. Encoding



B.411.2. Synopsis

No description available.

B.411.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.411.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.411.5. Execution

```

```

B.411.6. Exceptions

This instruction does not generate synchronous exceptions.

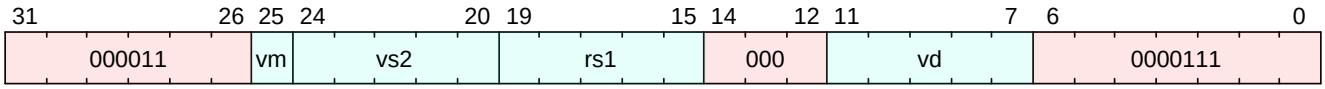
B.412. vloxei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.412.1. Encoding



B.412.2. Synopsis

No description available.

B.412.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.412.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.412.5. Execution

```

```

B.412.6. Exceptions

This instruction does not generate synchronous exceptions.

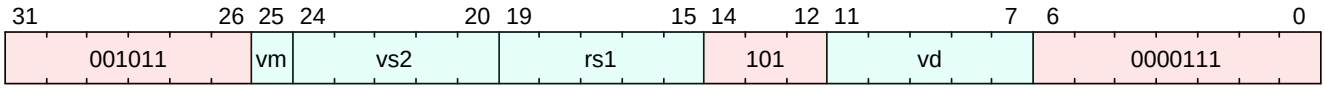
B.413. vloxseg2ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.413.1. Encoding



B.413.2. Synopsis

No description available.

B.413.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.413.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.413.5. Execution

```

```

B.413.6. Exceptions

This instruction does not generate synchronous exceptions.

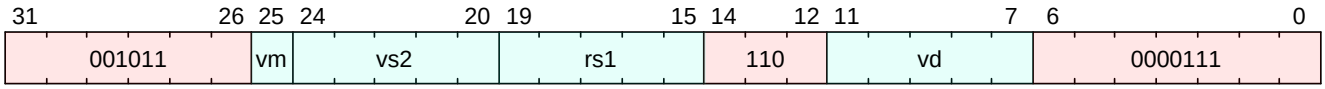
B.414. vloxseg2ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.414.1. Encoding



B.414.2. Synopsis

No description available.

B.414.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.414.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.414.5. Execution

```

```

B.414.6. Exceptions

This instruction does not generate synchronous exceptions.

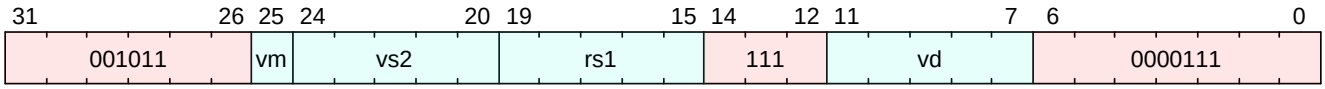
B.415. vloxseg2ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.415.1. Encoding



B.415.2. Synopsis

No description available.

B.415.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.415.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.415.5. Execution

```

```

B.415.6. Exceptions

This instruction does not generate synchronous exceptions.

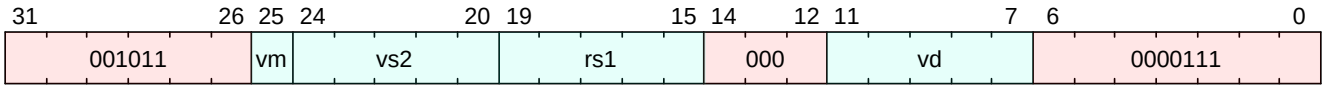
B.416. vloxseg2ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.416.1. Encoding



B.416.2. Synopsis

No description available.

B.416.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.416.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.416.5. Execution

```

```

B.416.6. Exceptions

This instruction does not generate synchronous exceptions.

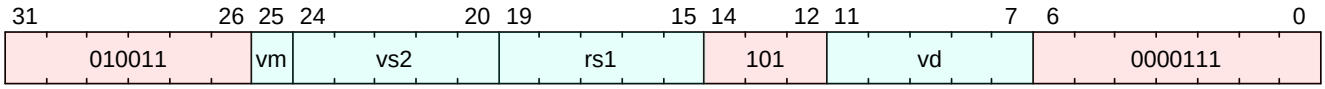
B.417. vloxseg3ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.417.1. Encoding



B.417.2. Synopsis

No description available.

B.417.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.417.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.417.5. Execution

```

```

B.417.6. Exceptions

This instruction does not generate synchronous exceptions.

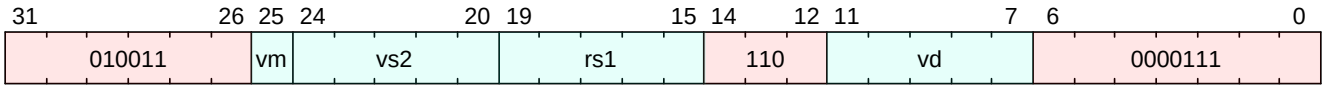
B.418. vloxseg3ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.418.1. Encoding



B.418.2. Synopsis

No description available.

B.418.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.418.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.418.5. Execution

```

```

B.418.6. Exceptions

This instruction does not generate synchronous exceptions.

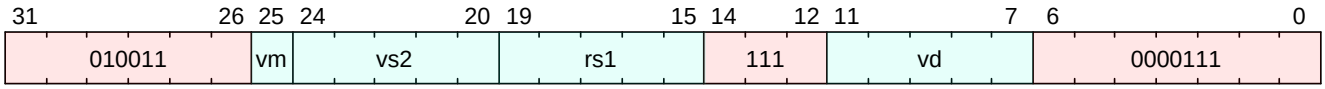
B.419. vloxseg3ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.419.1. Encoding



B.419.2. Synopsis

No description available.

B.419.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.419.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.419.5. Execution

```

```

B.419.6. Exceptions

This instruction does not generate synchronous exceptions.

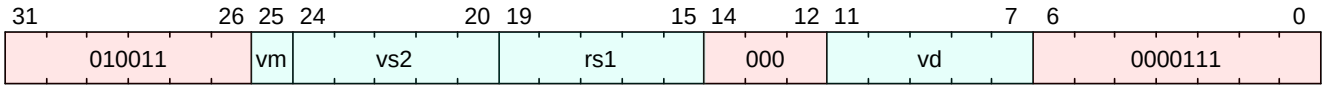
B.420. vloxseg3ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.420.1. Encoding



B.420.2. Synopsis

No description available.

B.420.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.420.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.420.5. Execution

```

```

B.420.6. Exceptions

This instruction does not generate synchronous exceptions.

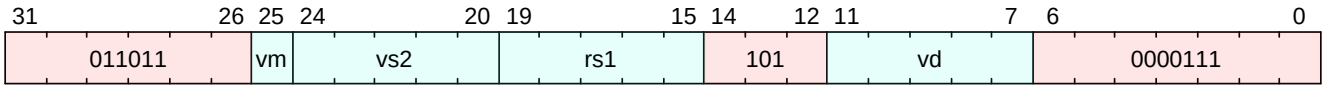
B.421. vloxseg4ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.421.1. Encoding



B.421.2. Synopsis

No description available.

B.421.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.421.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.421.5. Execution

```

```

B.421.6. Exceptions

This instruction does not generate synchronous exceptions.

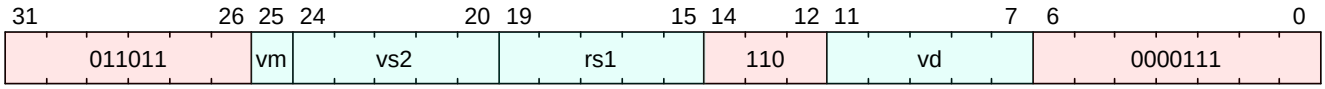
B.422. vloxseg4ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.422.1. Encoding



B.422.2. Synopsis

No description available.

B.422.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.422.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.422.5. Execution

```

```

B.422.6. Exceptions

This instruction does not generate synchronous exceptions.

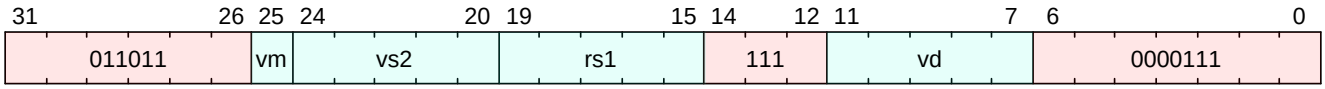
B.423. vloxseg4ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.423.1. Encoding



B.423.2. Synopsis

No description available.

B.423.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.423.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.423.5. Execution

```

```

B.423.6. Exceptions

This instruction does not generate synchronous exceptions.

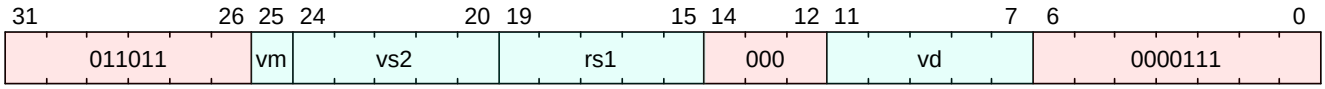
B.424. vloxseg4ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.424.1. Encoding



B.424.2. Synopsis

No description available.

B.424.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.424.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.424.5. Execution

```

```

B.424.6. Exceptions

This instruction does not generate synchronous exceptions.

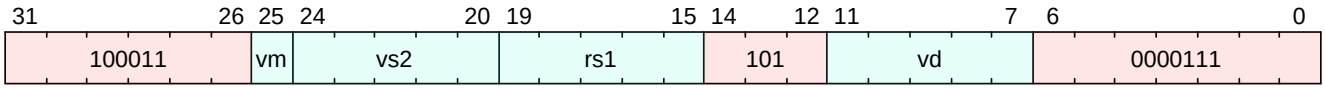
B.425. vloxseg5ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.425.1. Encoding



B.425.2. Synopsis

No description available.

B.425.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.425.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.425.5. Execution

```

```

B.425.6. Exceptions

This instruction does not generate synchronous exceptions.

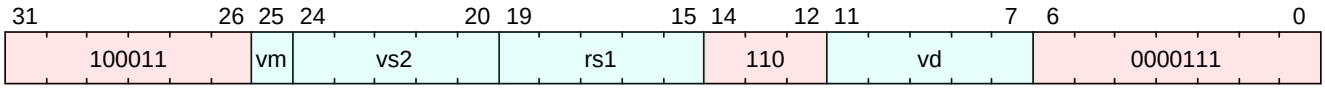
B.426. vloxseg5ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.426.1. Encoding



B.426.2. Synopsis

No description available.

B.426.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.426.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.426.5. Execution

```

```

B.426.6. Exceptions

This instruction does not generate synchronous exceptions.

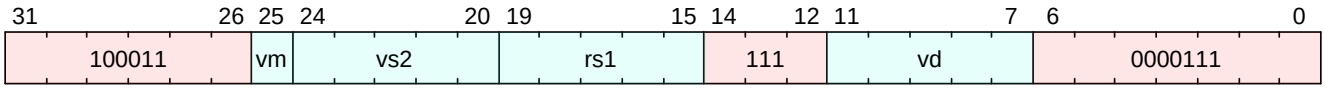
B.427. vloxseg5ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.427.1. Encoding



B.427.2. Synopsis

No description available.

B.427.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.427.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.427.5. Execution

```

```

B.427.6. Exceptions

This instruction does not generate synchronous exceptions.

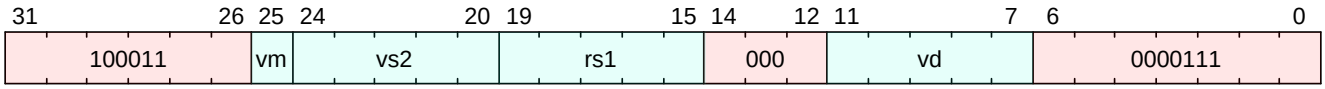
B.428. vloxseg5ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.428.1. Encoding



B.428.2. Synopsis

No description available.

B.428.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.428.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.428.5. Execution

```

```

B.428.6. Exceptions

This instruction does not generate synchronous exceptions.

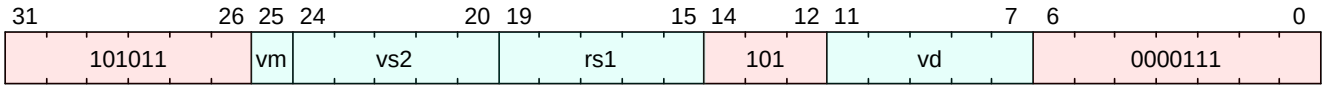
B.429. vloxseg6ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.429.1. Encoding



B.429.2. Synopsis

No description available.

B.429.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.429.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.429.5. Execution

```

```

B.429.6. Exceptions

This instruction does not generate synchronous exceptions.

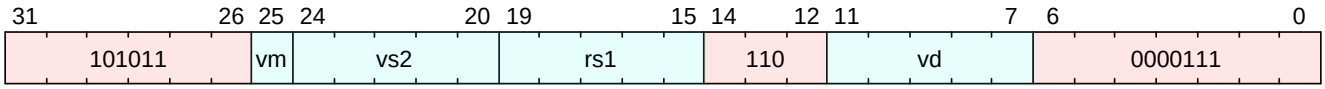
B.430. vloxseg6ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.430.1. Encoding



B.430.2. Synopsis

No description available.

B.430.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.430.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.430.5. Execution

```

```

B.430.6. Exceptions

This instruction does not generate synchronous exceptions.

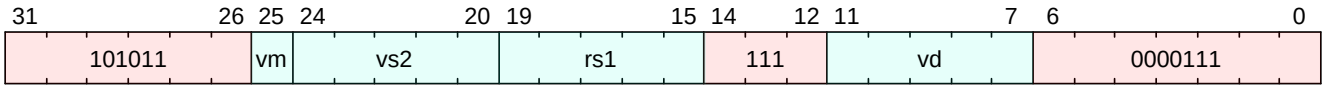
B.431. vloxseg6ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.431.1. Encoding



B.431.2. Synopsis

No description available.

B.431.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.431.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.431.5. Execution

```

```

B.431.6. Exceptions

This instruction does not generate synchronous exceptions.

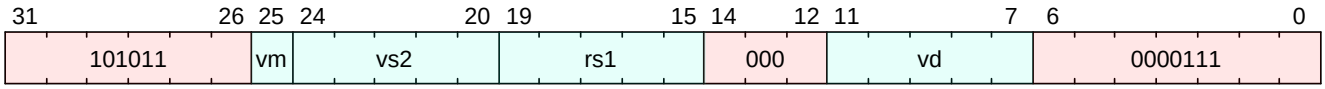
B.432. vloxseg6ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.432.1. Encoding



B.432.2. Synopsis

No description available.

B.432.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.432.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.432.5. Execution

```

```

B.432.6. Exceptions

This instruction does not generate synchronous exceptions.

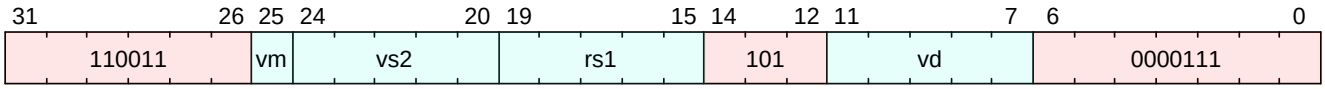
B.433. vloxseg7ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.433.1. Encoding



B.433.2. Synopsis

No description available.

B.433.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.433.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.433.5. Execution

```

```

B.433.6. Exceptions

This instruction does not generate synchronous exceptions.

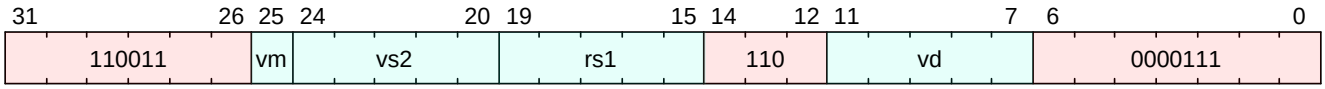
B.434. vloxseg7ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.434.1. Encoding



B.434.2. Synopsis

No description available.

B.434.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.434.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.434.5. Execution

```

```

B.434.6. Exceptions

This instruction does not generate synchronous exceptions.

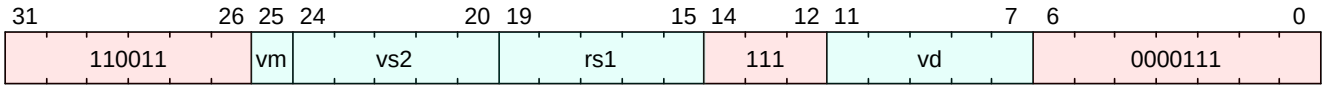
B.435. vloxseg7ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.435.1. Encoding



B.435.2. Synopsis

No description available.

B.435.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.435.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.435.5. Execution

```

```

B.435.6. Exceptions

This instruction does not generate synchronous exceptions.

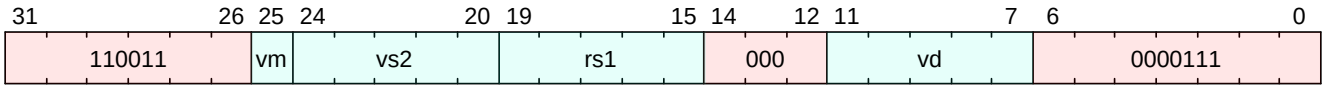
B.436. vloxseg7ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.436.1. Encoding



B.436.2. Synopsis

No description available.

B.436.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.436.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.436.5. Execution

```

```

B.436.6. Exceptions

This instruction does not generate synchronous exceptions.

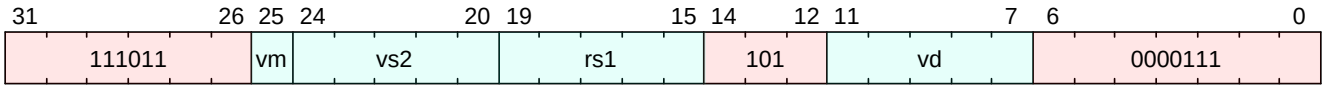
B.437. vloxseg8ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.437.1. Encoding



B.437.2. Synopsis

No description available.

B.437.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.437.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.437.5. Execution

```

```

B.437.6. Exceptions

This instruction does not generate synchronous exceptions.

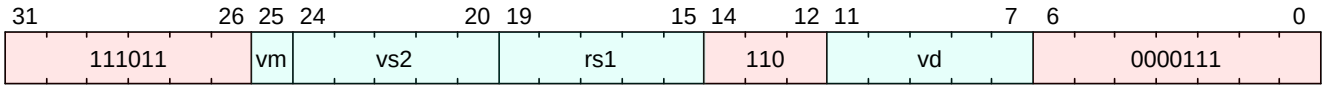
B.438. vloxseg8ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.438.1. Encoding



B.438.2. Synopsis

No description available.

B.438.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.438.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.438.5. Execution

```

```

B.438.6. Exceptions

This instruction does not generate synchronous exceptions.

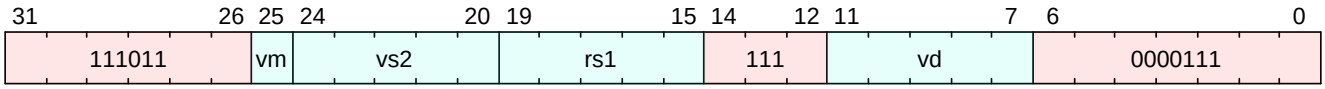
B.439. vloxseg8ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.439.1. Encoding



B.439.2. Synopsis

No description available.

B.439.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.439.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.439.5. Execution

```

```

B.439.6. Exceptions

This instruction does not generate synchronous exceptions.

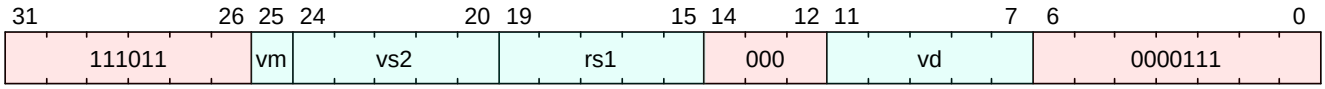
B.440. vloxseg8ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.440.1. Encoding



B.440.2. Synopsis

No description available.

B.440.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.440.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.440.5. Execution

```

```

B.440.6. Exceptions

This instruction does not generate synchronous exceptions.

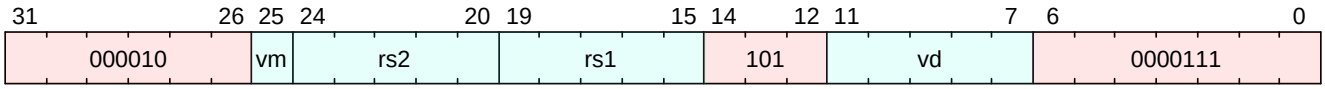
B.441. vlse16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.441.1. Encoding



B.441.2. Synopsis

No description available.

B.441.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.441.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.441.5. Execution

```

```

B.441.6. Exceptions

This instruction does not generate synchronous exceptions.

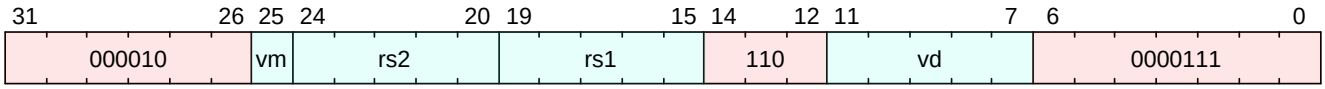
B.442. vlse32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.442.1. Encoding



B.442.2. Synopsis

No description available.

B.442.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.442.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.442.5. Execution

```

```

B.442.6. Exceptions

This instruction does not generate synchronous exceptions.

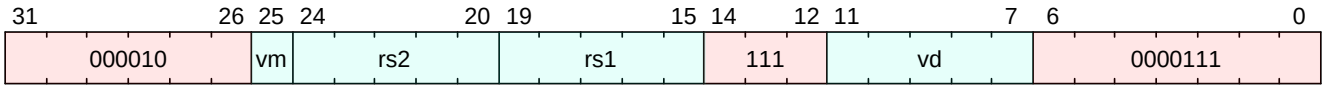
B.443. vlse64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.443.1. Encoding



B.443.2. Synopsis

No description available.

B.443.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.443.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.443.5. Execution

```

```

B.443.6. Exceptions

This instruction does not generate synchronous exceptions.

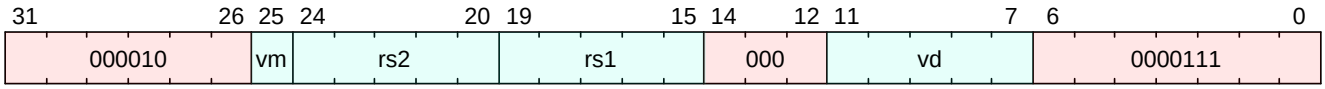
B.444. vlse8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.444.1. Encoding



B.444.2. Synopsis

No description available.

B.444.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.444.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.444.5. Execution

```

```

B.444.6. Exceptions

This instruction does not generate synchronous exceptions.

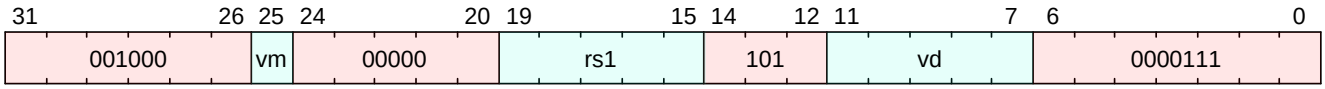
B.445. vlseg2e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.445.1. Encoding



B.445.2. Synopsis

No description available.

B.445.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.445.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.445.5. Execution

```

```

B.445.6. Exceptions

This instruction does not generate synchronous exceptions.

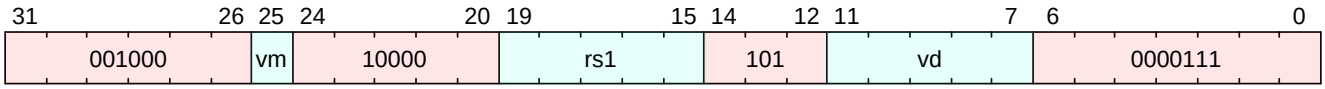
B.446. vlseg2e16ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.446.1. Encoding



B.446.2. Synopsis

No description available.

B.446.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.446.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.446.5. Execution

```

```

B.446.6. Exceptions

This instruction does not generate synchronous exceptions.

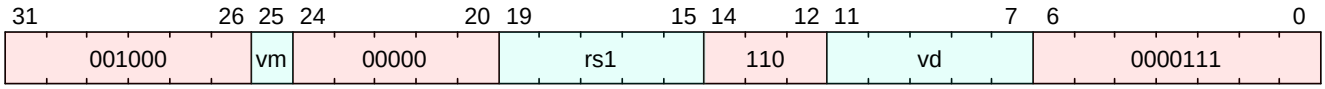
B.447. vlseg2e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.447.1. Encoding



B.447.2. Synopsis

No description available.

B.447.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.447.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.447.5. Execution

```

```

B.447.6. Exceptions

This instruction does not generate synchronous exceptions.

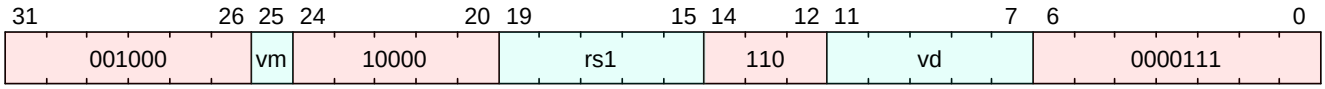
B.448. vlseg2e32ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.448.1. Encoding



B.448.2. Synopsis

No description available.

B.448.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.448.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.448.5. Execution

```

```

B.448.6. Exceptions

This instruction does not generate synchronous exceptions.

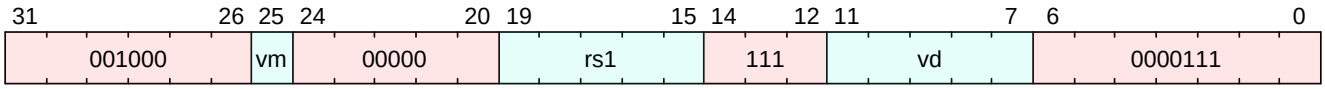
B.449. vlseg2e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.449.1. Encoding



B.449.2. Synopsis

No description available.

B.449.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.449.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.449.5. Execution

```

```

B.449.6. Exceptions

This instruction does not generate synchronous exceptions.

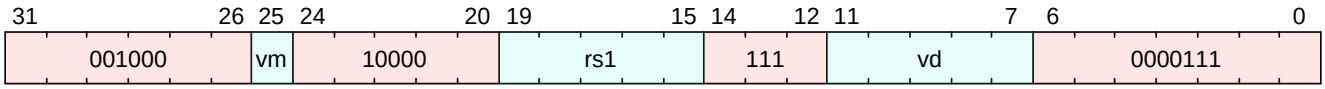
B.450. vlseg2e64ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.450.1. Encoding



B.450.2. Synopsis

No description available.

B.450.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.450.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.450.5. Execution

```

```

B.450.6. Exceptions

This instruction does not generate synchronous exceptions.

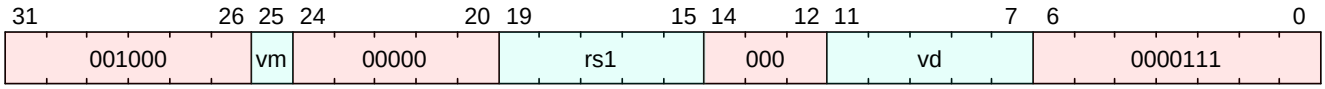
B.451. vlseg2e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.451.1. Encoding



B.451.2. Synopsis

No description available.

B.451.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.451.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.451.5. Execution

```

```

B.451.6. Exceptions

This instruction does not generate synchronous exceptions.

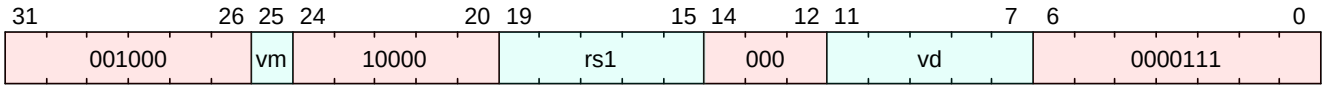
B.452. vlseg2e8ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.452.1. Encoding



B.452.2. Synopsis

No description available.

B.452.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.452.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.452.5. Execution

```

```

B.452.6. Exceptions

This instruction does not generate synchronous exceptions.

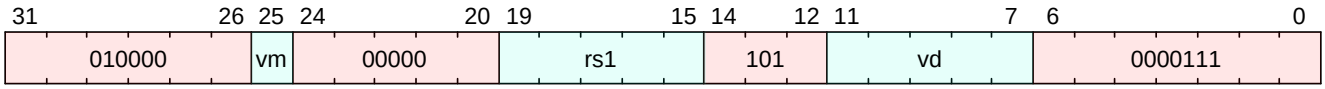
B.453. vlseg3e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.453.1. Encoding



B.453.2. Synopsis

No description available.

B.453.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.453.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.453.5. Execution

```

```

B.453.6. Exceptions

This instruction does not generate synchronous exceptions.

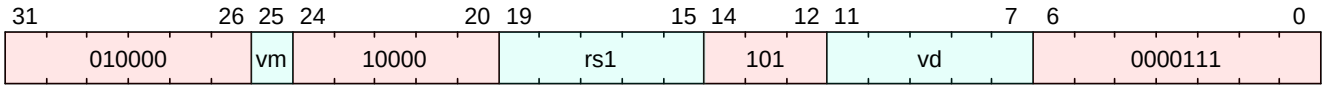
B.454. vlseg3e16ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.454.1. Encoding



B.454.2. Synopsis

No description available.

B.454.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.454.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.454.5. Execution

```

```

B.454.6. Exceptions

This instruction does not generate synchronous exceptions.

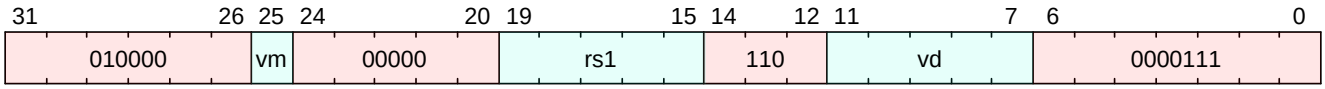
B.455. vlseg3e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.455.1. Encoding



B.455.2. Synopsis

No description available.

B.455.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.455.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.455.5. Execution

```

```

B.455.6. Exceptions

This instruction does not generate synchronous exceptions.

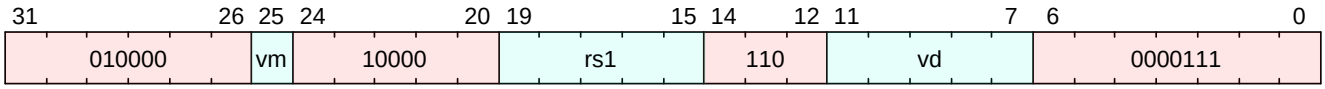
B.456. vlseg3e32ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.456.1. Encoding



B.456.2. Synopsis

No description available.

B.456.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.456.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.456.5. Execution

```

```

B.456.6. Exceptions

This instruction does not generate synchronous exceptions.

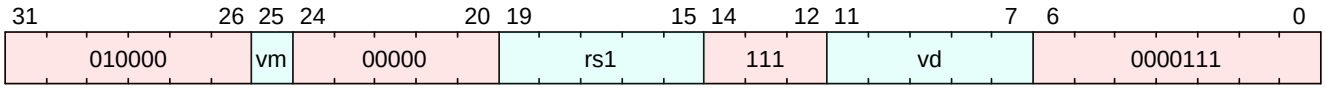
B.457. vlseg3e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.457.1. Encoding



B.457.2. Synopsis

No description available.

B.457.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.457.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.457.5. Execution

```

```

B.457.6. Exceptions

This instruction does not generate synchronous exceptions.

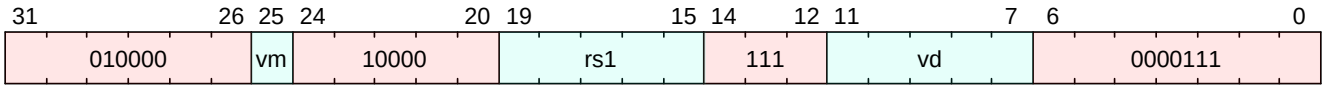
B.458. vlseg3e64ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.458.1. Encoding



B.458.2. Synopsis

No description available.

B.458.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.458.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.458.5. Execution

```

```

B.458.6. Exceptions

This instruction does not generate synchronous exceptions.

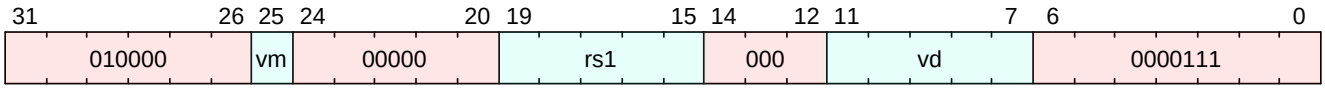
B.459. vlseg3e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.459.1. Encoding



B.459.2. Synopsis

No description available.

B.459.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.459.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.459.5. Execution

```

```

B.459.6. Exceptions

This instruction does not generate synchronous exceptions.

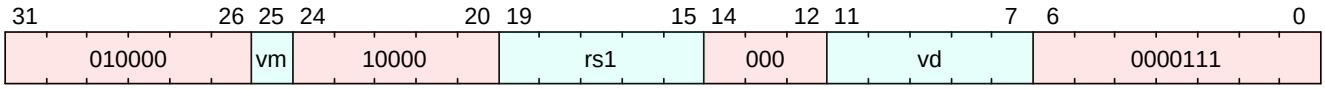
B.460. vlseg3e8ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.460.1. Encoding



B.460.2. Synopsis

No description available.

B.460.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.460.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.460.5. Execution

```

```

B.460.6. Exceptions

This instruction does not generate synchronous exceptions.

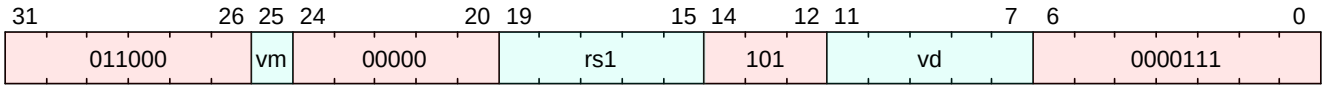
B.461. vlseg4e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.461.1. Encoding



B.461.2. Synopsis

No description available.

B.461.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.461.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.461.5. Execution

```

```

B.461.6. Exceptions

This instruction does not generate synchronous exceptions.

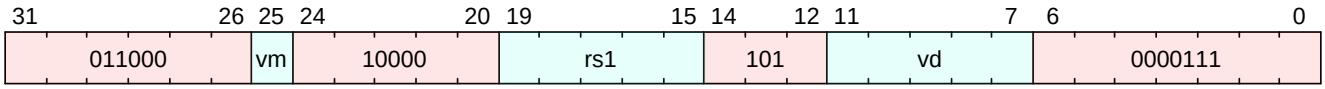
B.462. vlseg4e16ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.462.1. Encoding



B.462.2. Synopsis

No description available.

B.462.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.462.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.462.5. Execution

```

```

B.462.6. Exceptions

This instruction does not generate synchronous exceptions.

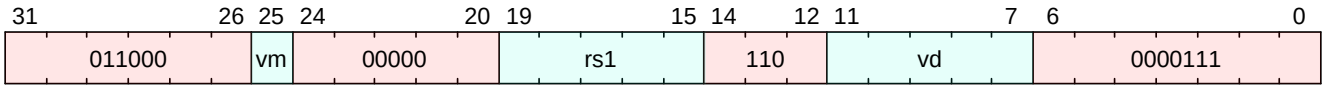
B.463. vlseg4e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.463.1. Encoding



B.463.2. Synopsis

No description available.

B.463.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.463.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.463.5. Execution

```

```

B.463.6. Exceptions

This instruction does not generate synchronous exceptions.

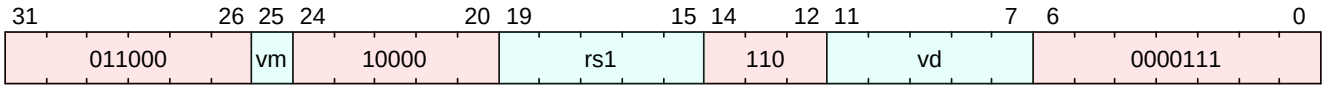
B.464. vlseg4e32ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.464.1. Encoding



B.464.2. Synopsis

No description available.

B.464.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.464.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.464.5. Execution

```

```

B.464.6. Exceptions

This instruction does not generate synchronous exceptions.

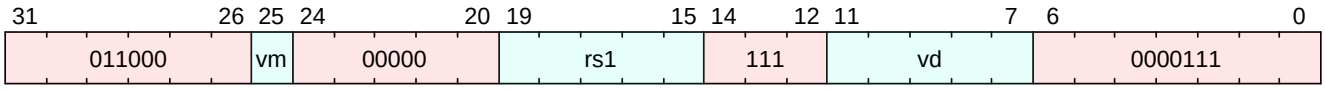
B.465. vlseg4e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.465.1. Encoding



B.465.2. Synopsis

No description available.

B.465.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.465.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.465.5. Execution

```

```

B.465.6. Exceptions

This instruction does not generate synchronous exceptions.

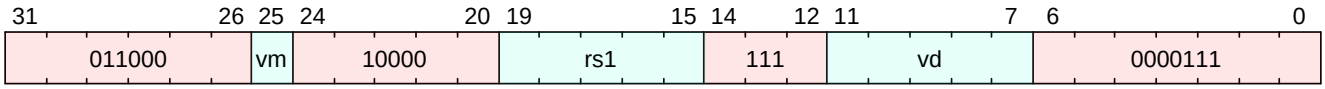
B.466. vlseg4e64ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.466.1. Encoding



B.466.2. Synopsis

No description available.

B.466.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.466.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.466.5. Execution

```

```

B.466.6. Exceptions

This instruction does not generate synchronous exceptions.

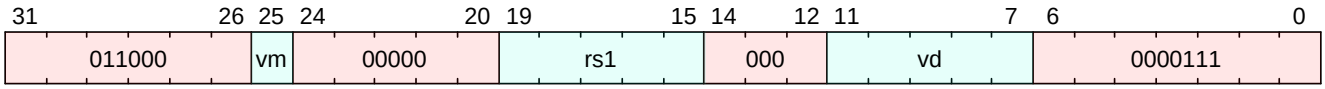
B.467. vlseg4e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.467.1. Encoding



B.467.2. Synopsis

No description available.

B.467.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.467.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.467.5. Execution

```

```

B.467.6. Exceptions

This instruction does not generate synchronous exceptions.

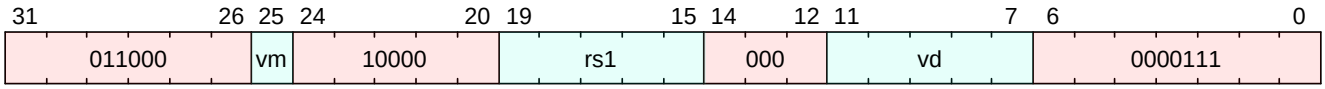
B.468. vlseg4e8ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.468.1. Encoding



B.468.2. Synopsis

No description available.

B.468.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.468.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.468.5. Execution

```

```

B.468.6. Exceptions

This instruction does not generate synchronous exceptions.

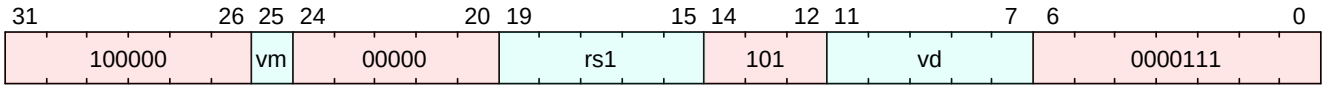
B.469. vlseg5e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.469.1. Encoding



B.469.2. Synopsis

No description available.

B.469.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.469.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.469.5. Execution

```

```

B.469.6. Exceptions

This instruction does not generate synchronous exceptions.

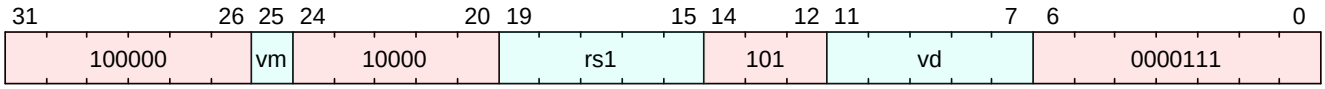
B.470. vlseg5e16ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.470.1. Encoding



B.470.2. Synopsis

No description available.

B.470.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.470.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.470.5. Execution

```

```

B.470.6. Exceptions

This instruction does not generate synchronous exceptions.

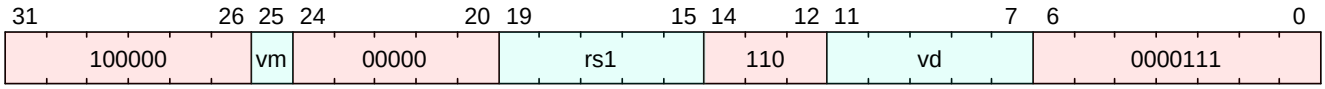
B.471. vlseg5e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.471.1. Encoding



B.471.2. Synopsis

No description available.

B.471.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.471.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.471.5. Execution

```

```

B.471.6. Exceptions

This instruction does not generate synchronous exceptions.

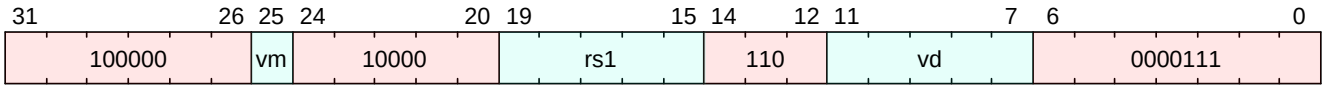
B.472. vlseg5e32ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.472.1. Encoding



B.472.2. Synopsis

No description available.

B.472.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.472.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.472.5. Execution

```

```

B.472.6. Exceptions

This instruction does not generate synchronous exceptions.

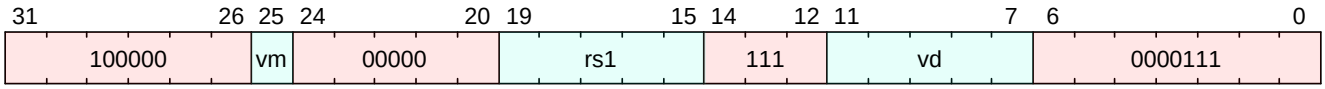
B.473. vlseg5e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.473.1. Encoding



B.473.2. Synopsis

No description available.

B.473.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.473.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.473.5. Execution

```

```

B.473.6. Exceptions

This instruction does not generate synchronous exceptions.

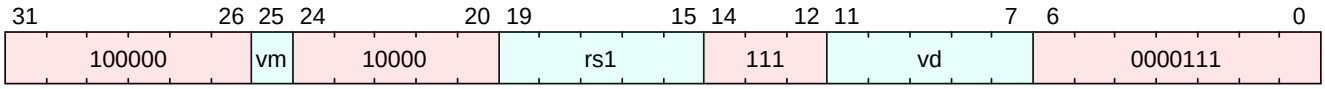
B.474. vlseg5e64ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.474.1. Encoding



B.474.2. Synopsis

No description available.

B.474.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.474.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.474.5. Execution

```

```

B.474.6. Exceptions

This instruction does not generate synchronous exceptions.

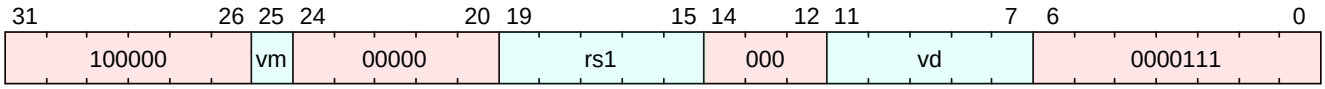
B.475. vlseg5e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.475.1. Encoding



B.475.2. Synopsis

No description available.

B.475.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.475.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.475.5. Execution

```

```

B.475.6. Exceptions

This instruction does not generate synchronous exceptions.

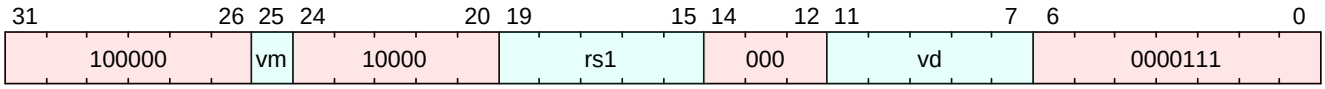
B.476. vlseg5e8ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.476.1. Encoding



B.476.2. Synopsis

No description available.

B.476.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.476.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.476.5. Execution

```

```

B.476.6. Exceptions

This instruction does not generate synchronous exceptions.

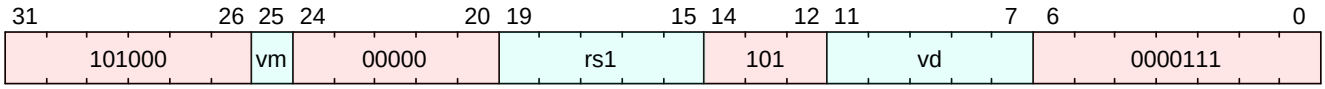
B.477. vlseg6e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.477.1. Encoding



B.477.2. Synopsis

No description available.

B.477.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.477.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.477.5. Execution

```

```

B.477.6. Exceptions

This instruction does not generate synchronous exceptions.

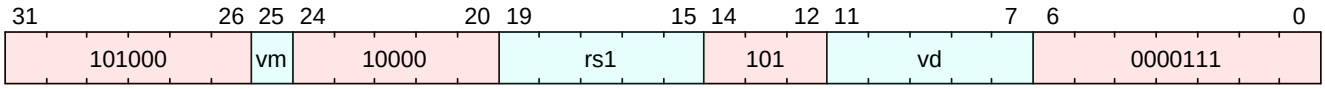
B.478. vlseg6e16ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.478.1. Encoding



B.478.2. Synopsis

No description available.

B.478.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.478.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.478.5. Execution

```

```

B.478.6. Exceptions

This instruction does not generate synchronous exceptions.

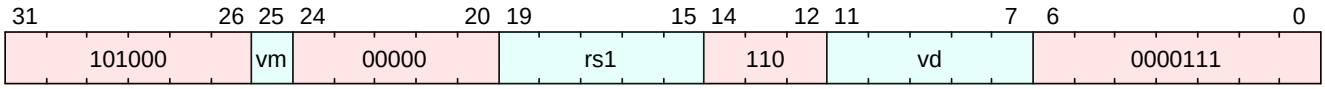
B.479. vlseg6e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.479.1. Encoding



B.479.2. Synopsis

No description available.

B.479.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.479.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.479.5. Execution

```

```

B.479.6. Exceptions

This instruction does not generate synchronous exceptions.

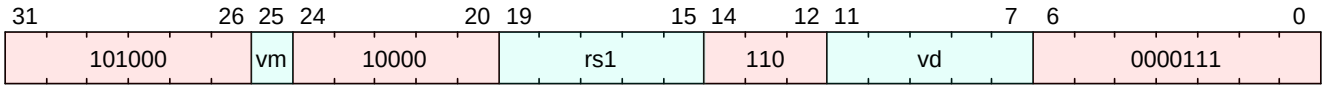
B.480. vlseg6e32ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.480.1. Encoding



B.480.2. Synopsis

No description available.

B.480.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.480.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.480.5. Execution

```

```

B.480.6. Exceptions

This instruction does not generate synchronous exceptions.

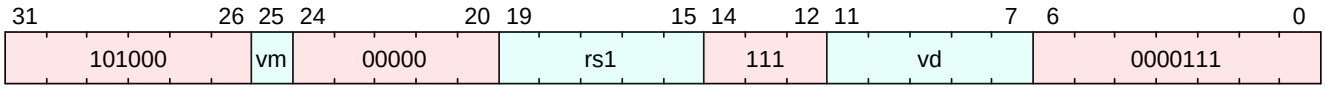
B.481. vlseg6e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.481.1. Encoding



B.481.2. Synopsis

No description available.

B.481.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.481.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.481.5. Execution

```

```

B.481.6. Exceptions

This instruction does not generate synchronous exceptions.

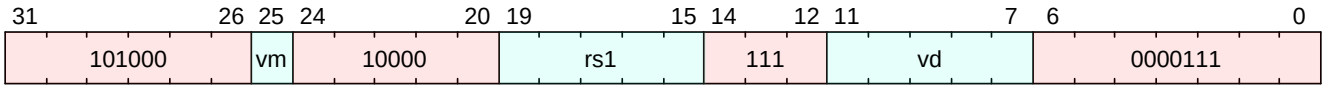
B.482. vlseg6e64ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.482.1. Encoding



B.482.2. Synopsis

No description available.

B.482.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.482.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.482.5. Execution

```

```

B.482.6. Exceptions

This instruction does not generate synchronous exceptions.

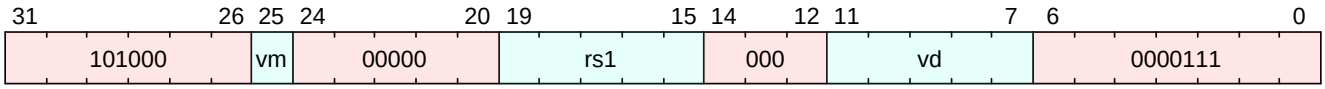
B.483. vlseg6e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.483.1. Encoding



B.483.2. Synopsis

No description available.

B.483.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.483.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.483.5. Execution

```

```

B.483.6. Exceptions

This instruction does not generate synchronous exceptions.

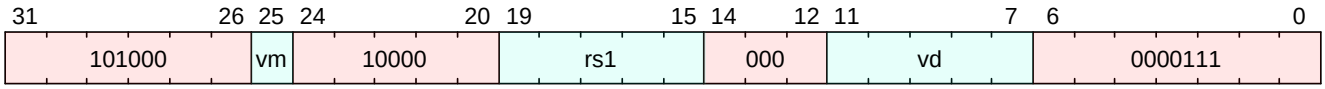
B.484. vlseg6e8ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.484.1. Encoding



B.484.2. Synopsis

No description available.

B.484.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.484.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.484.5. Execution

```

```

B.484.6. Exceptions

This instruction does not generate synchronous exceptions.

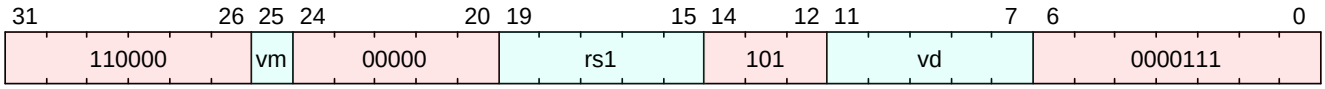
B.485. vlseg7e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.485.1. Encoding



B.485.2. Synopsis

No description available.

B.485.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.485.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.485.5. Execution

```

```

B.485.6. Exceptions

This instruction does not generate synchronous exceptions.

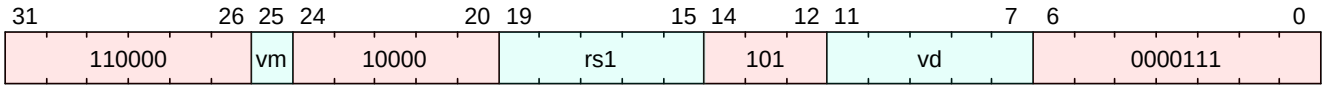
B.486. vlseg7e16ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.486.1. Encoding



B.486.2. Synopsis

No description available.

B.486.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.486.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.486.5. Execution

```

```

B.486.6. Exceptions

This instruction does not generate synchronous exceptions.

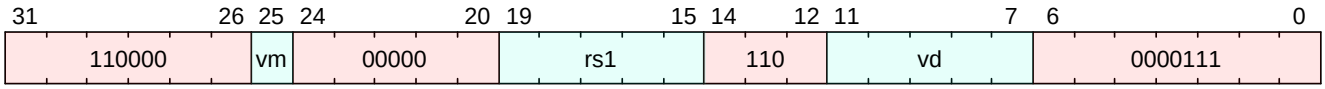
B.487. vlseg7e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.487.1. Encoding



B.487.2. Synopsis

No description available.

B.487.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.487.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.487.5. Execution

```

```

B.487.6. Exceptions

This instruction does not generate synchronous exceptions.

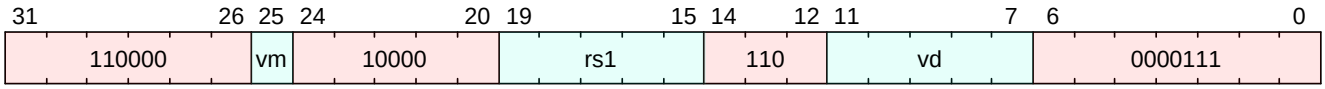
B.488. vlseg7e32ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.488.1. Encoding



B.488.2. Synopsis

No description available.

B.488.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.488.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.488.5. Execution

```

```

B.488.6. Exceptions

This instruction does not generate synchronous exceptions.

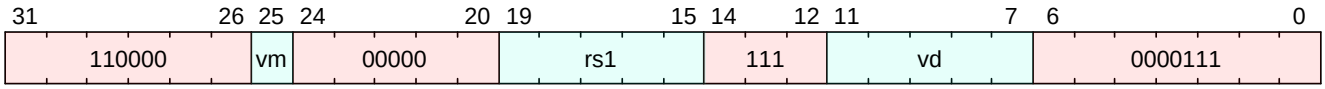
B.489. vlseg7e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.489.1. Encoding



B.489.2. Synopsis

No description available.

B.489.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.489.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.489.5. Execution

```

```

B.489.6. Exceptions

This instruction does not generate synchronous exceptions.

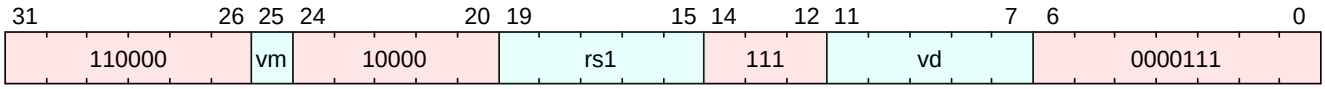
B.490. vlseg7e64ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.490.1. Encoding



B.490.2. Synopsis

No description available.

B.490.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.490.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.490.5. Execution

```

```

B.490.6. Exceptions

This instruction does not generate synchronous exceptions.

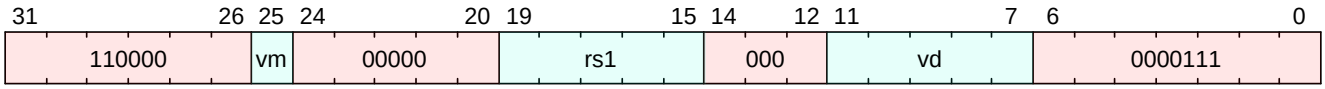
B.491. vlseg7e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.491.1. Encoding



B.491.2. Synopsis

No description available.

B.491.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.491.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.491.5. Execution

```

```

B.491.6. Exceptions

This instruction does not generate synchronous exceptions.

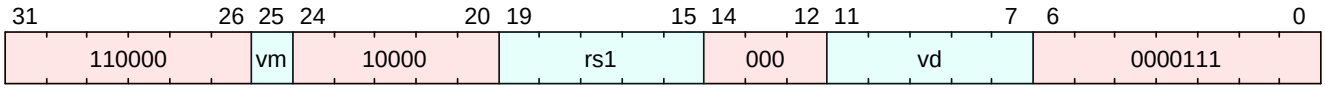
B.492. vlseg7e8ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.492.1. Encoding



B.492.2. Synopsis

No description available.

B.492.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.492.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.492.5. Execution

```

```

B.492.6. Exceptions

This instruction does not generate synchronous exceptions.

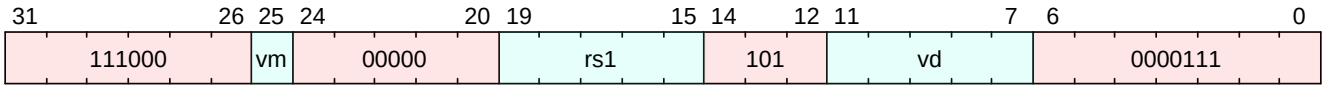
B.493. vlseg8e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.493.1. Encoding



B.493.2. Synopsis

No description available.

B.493.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.493.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.493.5. Execution

```

```

B.493.6. Exceptions

This instruction does not generate synchronous exceptions.

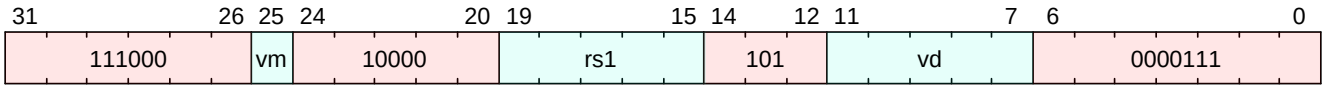
B.494. vlseg8e16ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.494.1. Encoding



B.494.2. Synopsis

No description available.

B.494.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.494.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.494.5. Execution

```

```

B.494.6. Exceptions

This instruction does not generate synchronous exceptions.

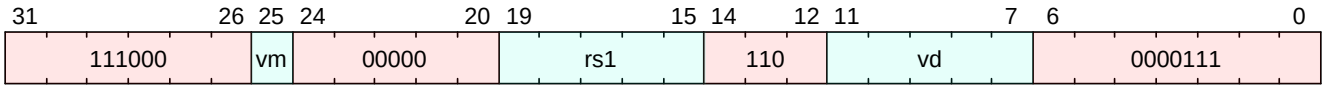
B.495. vlseg8e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.495.1. Encoding



B.495.2. Synopsis

No description available.

B.495.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.495.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.495.5. Execution

```

```

B.495.6. Exceptions

This instruction does not generate synchronous exceptions.

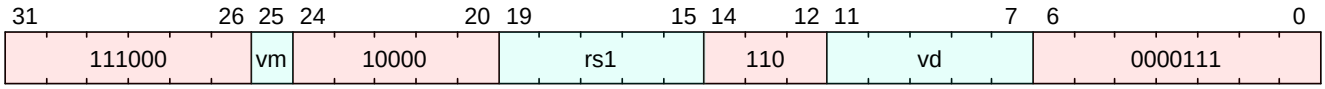
B.496. vlseg8e32ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.496.1. Encoding



B.496.2. Synopsis

No description available.

B.496.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.496.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.496.5. Execution

```

```

B.496.6. Exceptions

This instruction does not generate synchronous exceptions.

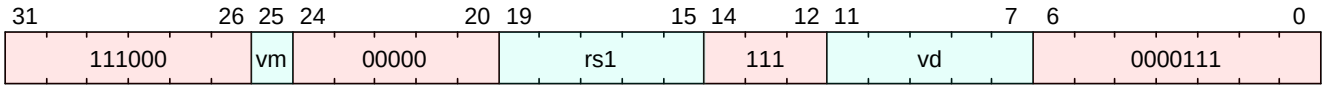
B.497. vlseg8e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.497.1. Encoding



B.497.2. Synopsis

No description available.

B.497.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.497.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.497.5. Execution

```

```

B.497.6. Exceptions

This instruction does not generate synchronous exceptions.

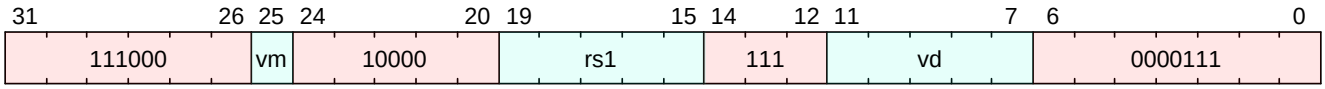
B.498. vlseg8e64ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.498.1. Encoding



B.498.2. Synopsis

No description available.

B.498.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.498.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.498.5. Execution

```

```

B.498.6. Exceptions

This instruction does not generate synchronous exceptions.

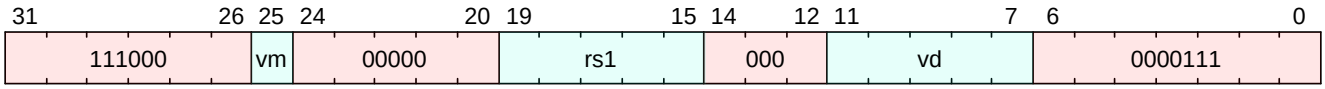
B.499. vlseg8e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.499.1. Encoding



B.499.2. Synopsis

No description available.

B.499.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.499.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.499.5. Execution

```

```

B.499.6. Exceptions

This instruction does not generate synchronous exceptions.

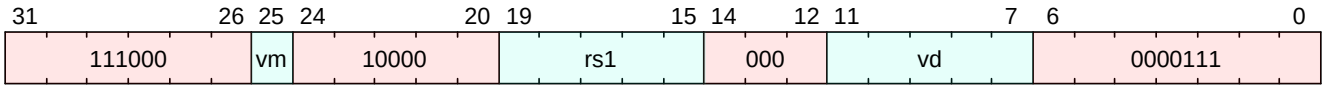
B.500. vlseg8e8ff.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.500.1. Encoding



B.500.2. Synopsis

No description available.

B.500.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.500.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.500.5. Execution

```

```

B.500.6. Exceptions

This instruction does not generate synchronous exceptions.

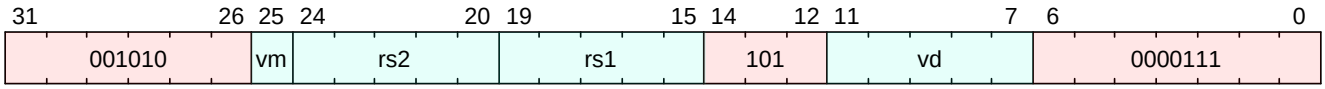
B.501. vlsseg2e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.501.1. Encoding



B.501.2. Synopsis

No description available.

B.501.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.501.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.501.5. Execution

```

```

B.501.6. Exceptions

This instruction does not generate synchronous exceptions.

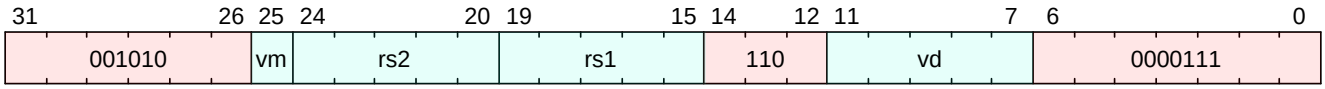
B.502. vlsseg2e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.502.1. Encoding



B.502.2. Synopsis

No description available.

B.502.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.502.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.502.5. Execution

```

```

B.502.6. Exceptions

This instruction does not generate synchronous exceptions.

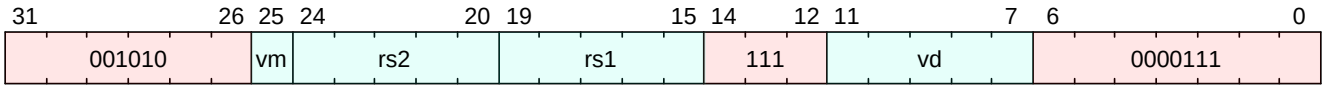
B.503. vlsseg2e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.503.1. Encoding



B.503.2. Synopsis

No description available.

B.503.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.503.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.503.5. Execution

```

```

B.503.6. Exceptions

This instruction does not generate synchronous exceptions.

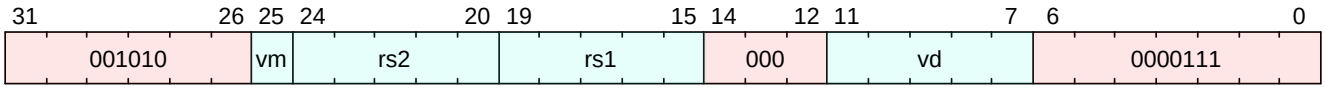
B.504. vlsseg2e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.504.1. Encoding



B.504.2. Synopsis

No description available.

B.504.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.504.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.504.5. Execution

```

```

B.504.6. Exceptions

This instruction does not generate synchronous exceptions.

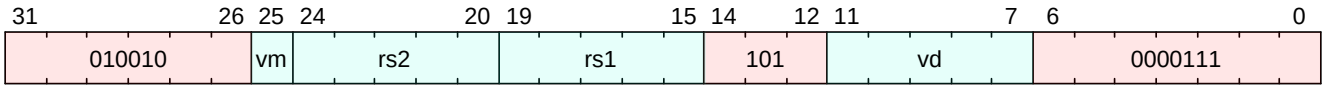
B.505. vlsseg3e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.505.1. Encoding



B.505.2. Synopsis

No description available.

B.505.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.505.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.505.5. Execution

```

```

B.505.6. Exceptions

This instruction does not generate synchronous exceptions.

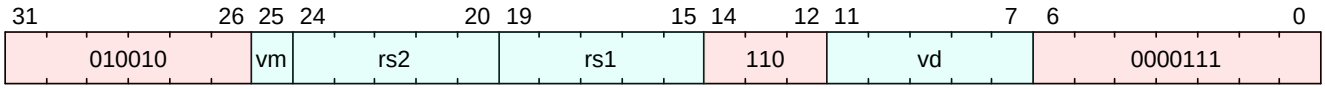
B.506. vlsseg3e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.506.1. Encoding



B.506.2. Synopsis

No description available.

B.506.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.506.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.506.5. Execution

```

```

B.506.6. Exceptions

This instruction does not generate synchronous exceptions.

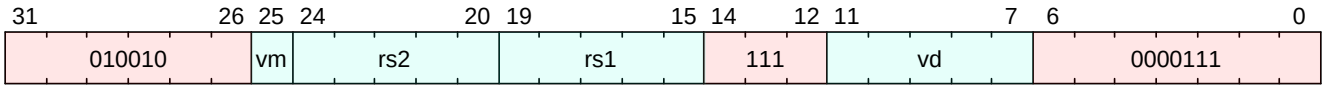
B.507. vlsseg3e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.507.1. Encoding



B.507.2. Synopsis

No description available.

B.507.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.507.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.507.5. Execution

```

```

B.507.6. Exceptions

This instruction does not generate synchronous exceptions.

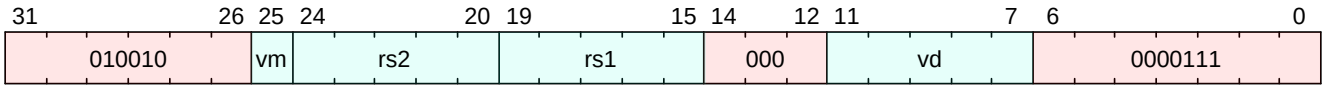
B.508. vlsseg3e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.508.1. Encoding



B.508.2. Synopsis

No description available.

B.508.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.508.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.508.5. Execution

```

```

B.508.6. Exceptions

This instruction does not generate synchronous exceptions.

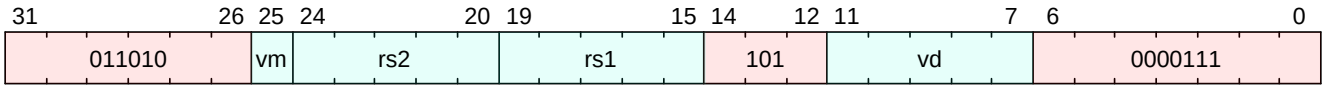
B.509. vlsseg4e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.509.1. Encoding



B.509.2. Synopsis

No description available.

B.509.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.509.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.509.5. Execution

```

```

B.509.6. Exceptions

This instruction does not generate synchronous exceptions.

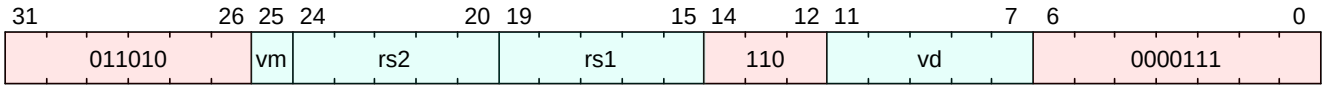
B.510. vlsseg4e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.510.1. Encoding



B.510.2. Synopsis

No description available.

B.510.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.510.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.510.5. Execution

```

```

B.510.6. Exceptions

This instruction does not generate synchronous exceptions.

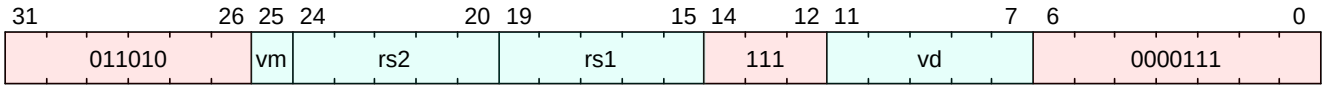
B.511. vlsseg4e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.511.1. Encoding



B.511.2. Synopsis

No description available.

B.511.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.511.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.511.5. Execution

```

```

B.511.6. Exceptions

This instruction does not generate synchronous exceptions.

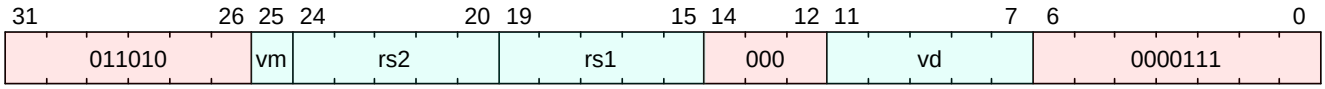
B.512. vlsseg4e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.512.1. Encoding



B.512.2. Synopsis

No description available.

B.512.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.512.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.512.5. Execution

```

```

B.512.6. Exceptions

This instruction does not generate synchronous exceptions.

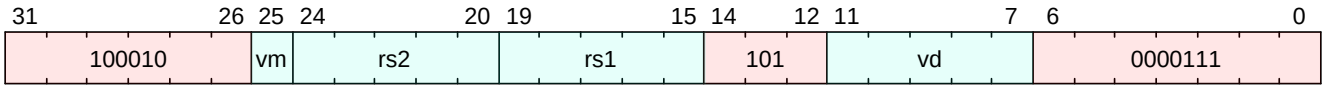
B.513. vlsseg5e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.513.1. Encoding



B.513.2. Synopsis

No description available.

B.513.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.513.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.513.5. Execution

```

```

B.513.6. Exceptions

This instruction does not generate synchronous exceptions.

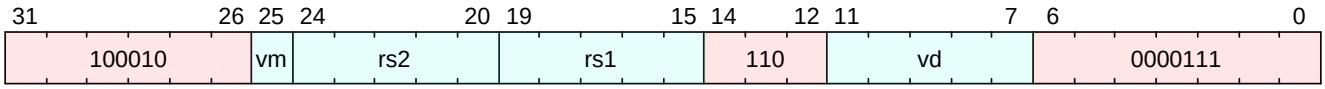
B.514. vlsseg5e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.514.1. Encoding



B.514.2. Synopsis

No description available.

B.514.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.514.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.514.5. Execution

```

```

B.514.6. Exceptions

This instruction does not generate synchronous exceptions.

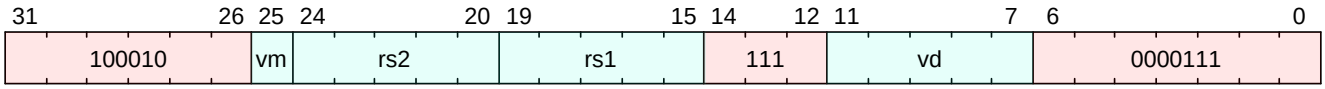
B.515. vlsseg5e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.515.1. Encoding



B.515.2. Synopsis

No description available.

B.515.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.515.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.515.5. Execution

```

```

B.515.6. Exceptions

This instruction does not generate synchronous exceptions.

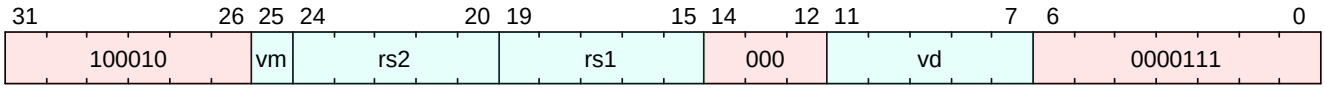
B.516. vlsseg5e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.516.1. Encoding



B.516.2. Synopsis

No description available.

B.516.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.516.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.516.5. Execution

```

```

B.516.6. Exceptions

This instruction does not generate synchronous exceptions.

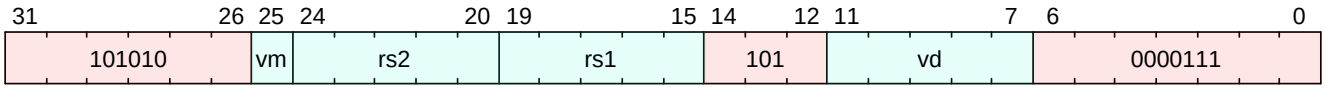
B.517. vlsseg6e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.517.1. Encoding



B.517.2. Synopsis

No description available.

B.517.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.517.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.517.5. Execution

```

```

B.517.6. Exceptions

This instruction does not generate synchronous exceptions.

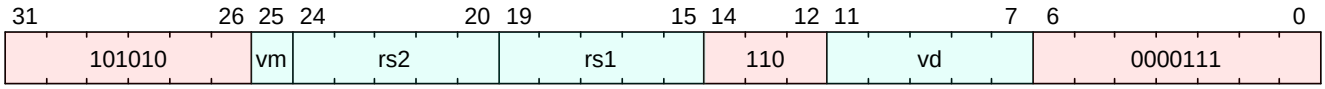
B.518. vlsseg6e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.518.1. Encoding



B.518.2. Synopsis

No description available.

B.518.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.518.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.518.5. Execution

```

```

B.518.6. Exceptions

This instruction does not generate synchronous exceptions.

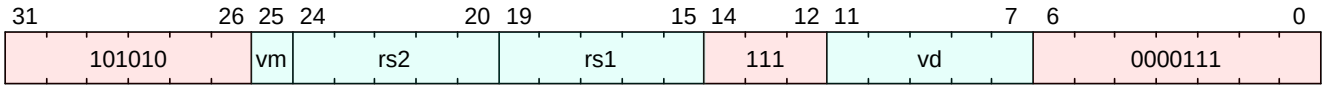
B.519. vlsseg6e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.519.1. Encoding



B.519.2. Synopsis

No description available.

B.519.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.519.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.519.5. Execution

```

```

B.519.6. Exceptions

This instruction does not generate synchronous exceptions.

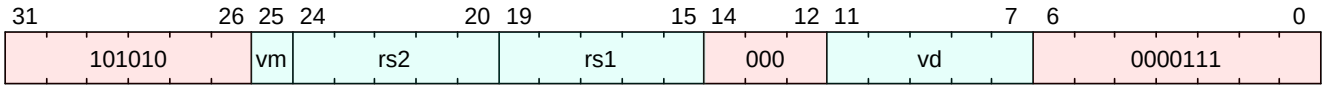
B.520. vlsseg6e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.520.1. Encoding



B.520.2. Synopsis

No description available.

B.520.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.520.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.520.5. Execution

```

```

B.520.6. Exceptions

This instruction does not generate synchronous exceptions.

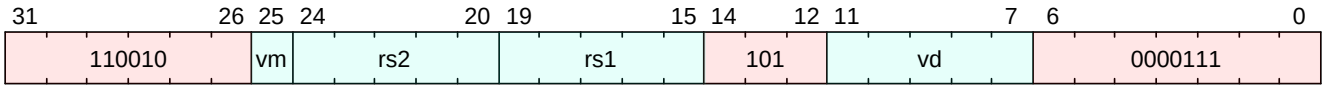
B.521. vlsseg7e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.521.1. Encoding



B.521.2. Synopsis

No description available.

B.521.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.521.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.521.5. Execution

```

```

B.521.6. Exceptions

This instruction does not generate synchronous exceptions.

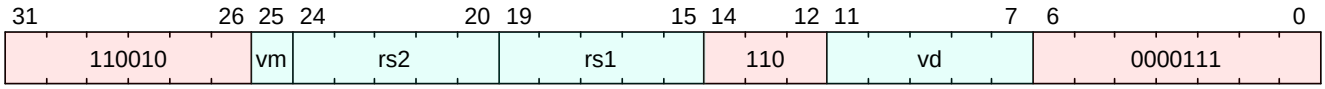
B.522. vlsseg7e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.522.1. Encoding



B.522.2. Synopsis

No description available.

B.522.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.522.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.522.5. Execution

```

```

B.522.6. Exceptions

This instruction does not generate synchronous exceptions.

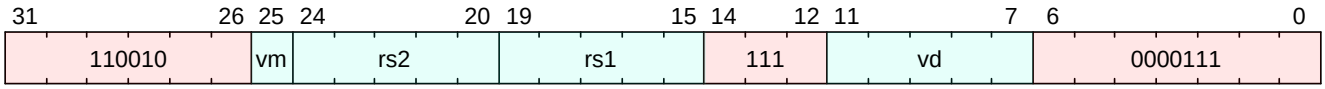
B.523. vlsseg7e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.523.1. Encoding



B.523.2. Synopsis

No description available.

B.523.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.523.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.523.5. Execution

```

```

B.523.6. Exceptions

This instruction does not generate synchronous exceptions.

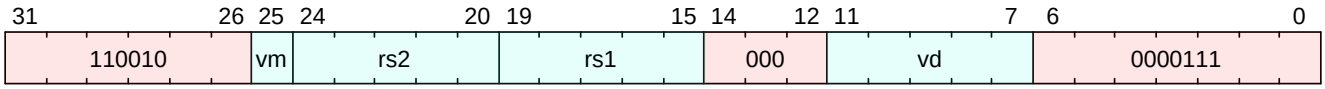
B.524. vlsseg7e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.524.1. Encoding



B.524.2. Synopsis

No description available.

B.524.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.524.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.524.5. Execution

```

```

B.524.6. Exceptions

This instruction does not generate synchronous exceptions.

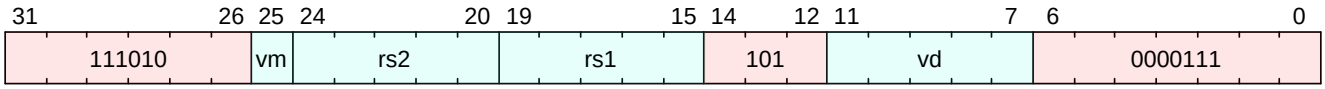
B.525. vlsseg8e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.525.1. Encoding



B.525.2. Synopsis

No description available.

B.525.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.525.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.525.5. Execution

```

```

B.525.6. Exceptions

This instruction does not generate synchronous exceptions.

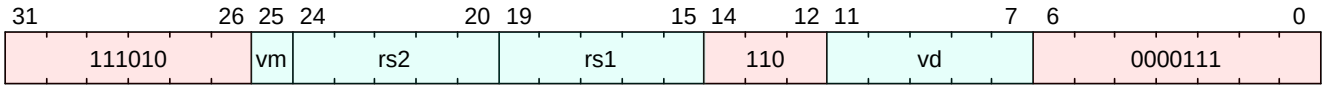
B.526. vlsseg8e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.526.1. Encoding



B.526.2. Synopsis

No description available.

B.526.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.526.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.526.5. Execution

```

```

B.526.6. Exceptions

This instruction does not generate synchronous exceptions.

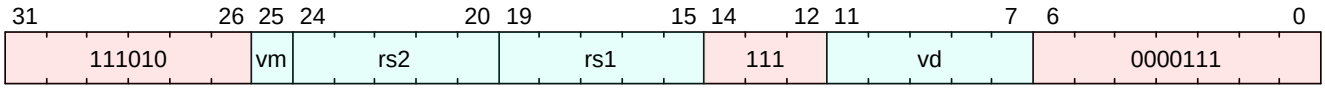
B.527. vlsseg8e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.527.1. Encoding



B.527.2. Synopsis

No description available.

B.527.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.527.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.527.5. Execution

```

```

B.527.6. Exceptions

This instruction does not generate synchronous exceptions.

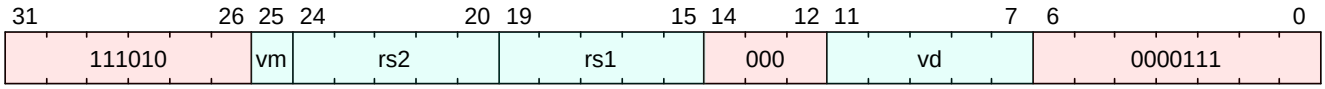
B.528. vlsseg8e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.528.1. Encoding



B.528.2. Synopsis

No description available.

B.528.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.528.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.528.5. Execution

```

```

B.528.6. Exceptions

This instruction does not generate synchronous exceptions.

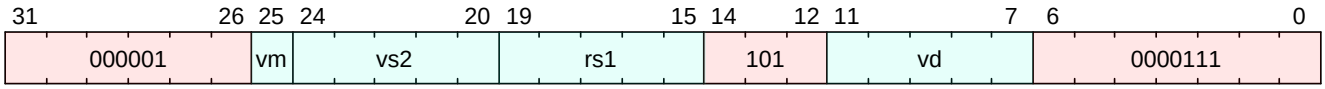
B.529. vluxei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.529.1. Encoding



B.529.2. Synopsis

No description available.

B.529.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.529.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.529.5. Execution

```

```

B.529.6. Exceptions

This instruction does not generate synchronous exceptions.

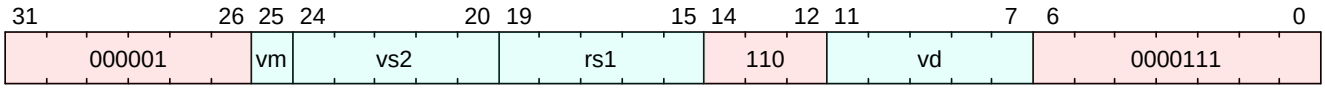
B.530. vluxei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.530.1. Encoding



B.530.2. Synopsis

No description available.

B.530.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.530.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.530.5. Execution

```

```

B.530.6. Exceptions

This instruction does not generate synchronous exceptions.

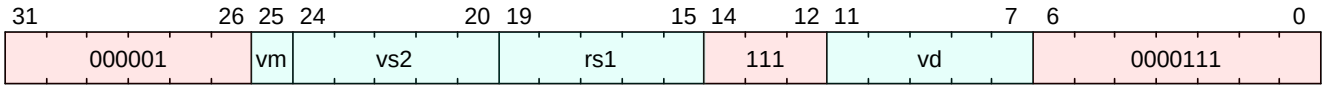
B.531. vluxei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.531.1. Encoding



B.531.2. Synopsis

No description available.

B.531.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.531.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.531.5. Execution

```

```

B.531.6. Exceptions

This instruction does not generate synchronous exceptions.

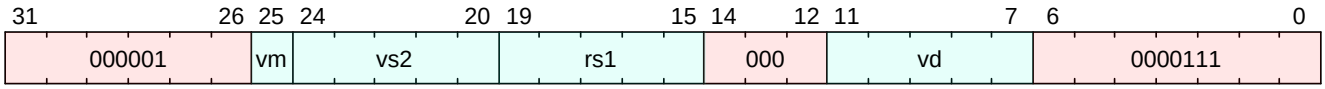
B.532. vluxei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.532.1. Encoding



B.532.2. Synopsis

No description available.

B.532.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.532.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.532.5. Execution

```

```

B.532.6. Exceptions

This instruction does not generate synchronous exceptions.

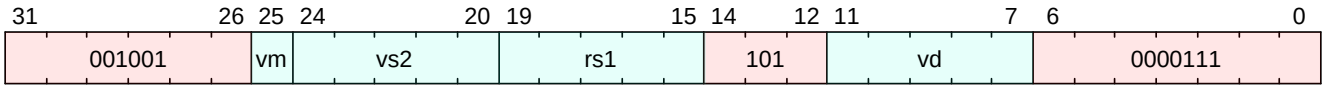
B.533. vluxseg2ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.533.1. Encoding



B.533.2. Synopsis

No description available.

B.533.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.533.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.533.5. Execution

```

```

B.533.6. Exceptions

This instruction does not generate synchronous exceptions.

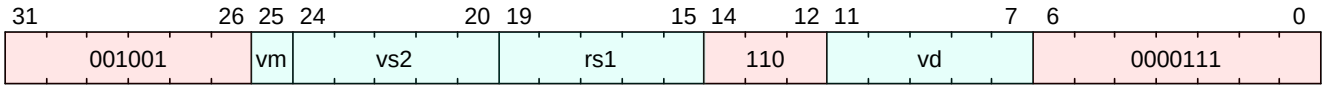
B.534. vluxseg2ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.534.1. Encoding



B.534.2. Synopsis

No description available.

B.534.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.534.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.534.5. Execution

```

```

B.534.6. Exceptions

This instruction does not generate synchronous exceptions.

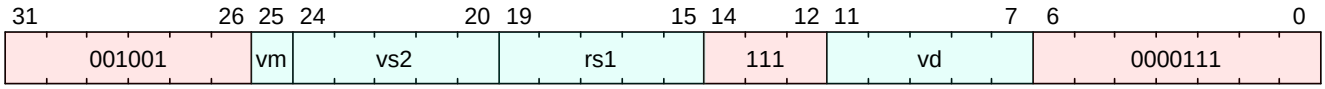
B.535. vluxseg2ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.535.1. Encoding



B.535.2. Synopsis

No description available.

B.535.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.535.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.535.5. Execution

```

```

B.535.6. Exceptions

This instruction does not generate synchronous exceptions.

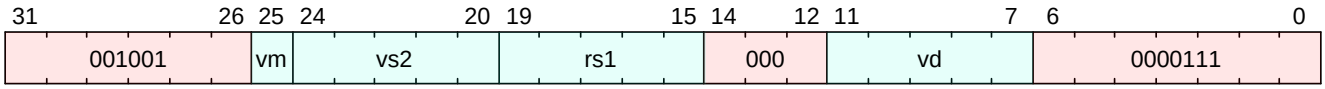
B.536. vluxseg2ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.536.1. Encoding



B.536.2. Synopsis

No description available.

B.536.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.536.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.536.5. Execution

```

```

B.536.6. Exceptions

This instruction does not generate synchronous exceptions.

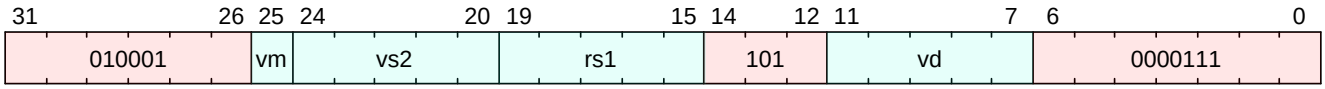
B.537. vluxseg3ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.537.1. Encoding



B.537.2. Synopsis

No description available.

B.537.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.537.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.537.5. Execution

```

```

B.537.6. Exceptions

This instruction does not generate synchronous exceptions.

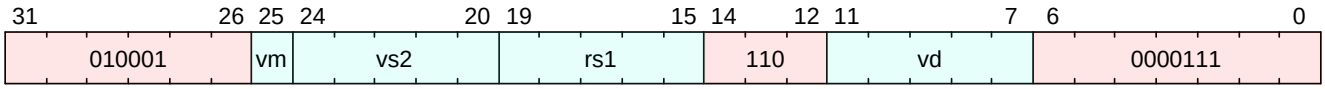
B.538. vluxseg3ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.538.1. Encoding



B.538.2. Synopsis

No description available.

B.538.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.538.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.538.5. Execution

```

```

B.538.6. Exceptions

This instruction does not generate synchronous exceptions.

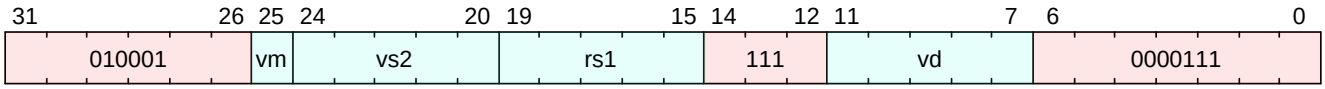
B.539. vluxseg3ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.539.1. Encoding



B.539.2. Synopsis

No description available.

B.539.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.539.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.539.5. Execution

```

```

B.539.6. Exceptions

This instruction does not generate synchronous exceptions.

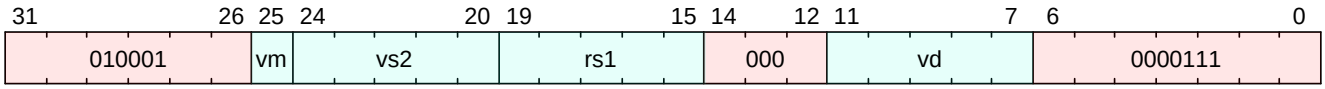
B.540. vluxseg3ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.540.1. Encoding



B.540.2. Synopsis

No description available.

B.540.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.540.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.540.5. Execution

```

```

B.540.6. Exceptions

This instruction does not generate synchronous exceptions.

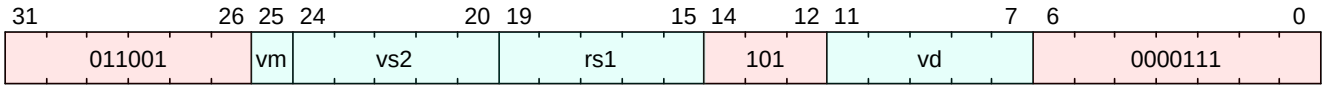
B.541. vluxseg4ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.541.1. Encoding



B.541.2. Synopsis

No description available.

B.541.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.541.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.541.5. Execution

```

```

B.541.6. Exceptions

This instruction does not generate synchronous exceptions.

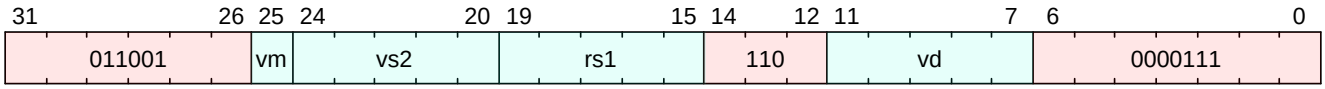
B.542. vluxseg4ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.542.1. Encoding



B.542.2. Synopsis

No description available.

B.542.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.542.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.542.5. Execution

```

```

B.542.6. Exceptions

This instruction does not generate synchronous exceptions.

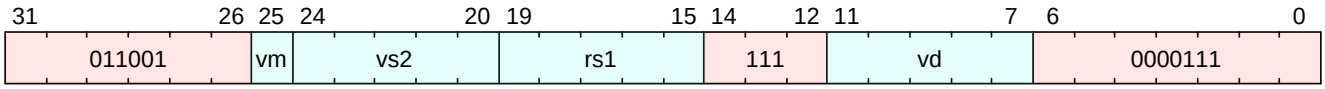
B.543. vluxseg4ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.543.1. Encoding



B.543.2. Synopsis

No description available.

B.543.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.543.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.543.5. Execution

```

```

B.543.6. Exceptions

This instruction does not generate synchronous exceptions.

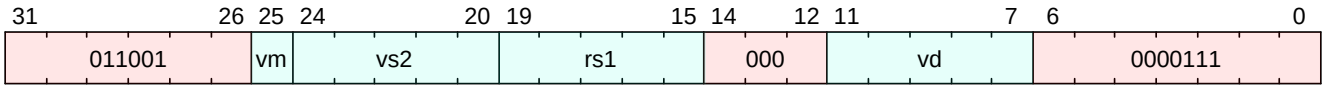
B.544. vluxseg4ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.544.1. Encoding



B.544.2. Synopsis

No description available.

B.544.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.544.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.544.5. Execution

```

```

B.544.6. Exceptions

This instruction does not generate synchronous exceptions.

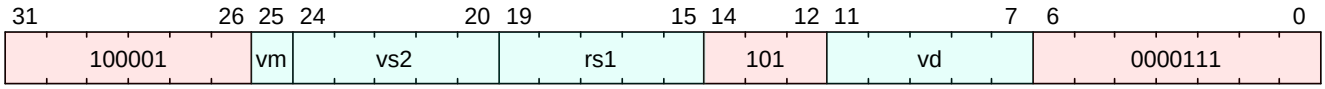
B.545. vluxseg5ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.545.1. Encoding



B.545.2. Synopsis

No description available.

B.545.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.545.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.545.5. Execution

```

```

B.545.6. Exceptions

This instruction does not generate synchronous exceptions.

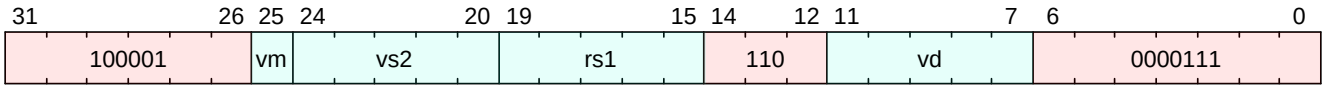
B.546. vluxseg5ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.546.1. Encoding



B.546.2. Synopsis

No description available.

B.546.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.546.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.546.5. Execution

```

```

B.546.6. Exceptions

This instruction does not generate synchronous exceptions.

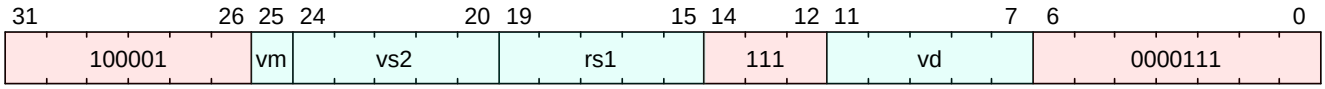
B.547. vluxseg5ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.547.1. Encoding



B.547.2. Synopsis

No description available.

B.547.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.547.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.547.5. Execution

```

```

B.547.6. Exceptions

This instruction does not generate synchronous exceptions.

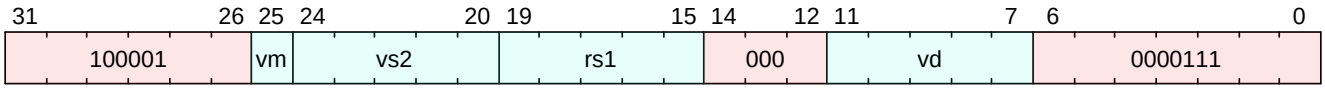
B.548. vluxseg5ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.548.1. Encoding



B.548.2. Synopsis

No description available.

B.548.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.548.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.548.5. Execution

```

```

B.548.6. Exceptions

This instruction does not generate synchronous exceptions.

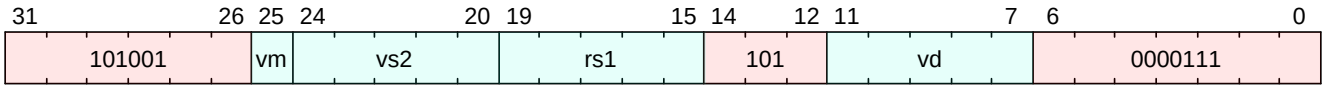
B.549. vluxseg6ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.549.1. Encoding



B.549.2. Synopsis

No description available.

B.549.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.549.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.549.5. Execution

```

```

B.549.6. Exceptions

This instruction does not generate synchronous exceptions.

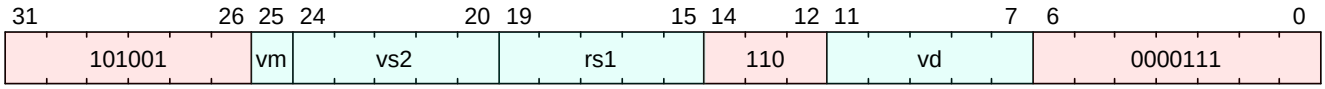
B.550. vluxseg6ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.550.1. Encoding



B.550.2. Synopsis

No description available.

B.550.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.550.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.550.5. Execution

```

```

B.550.6. Exceptions

This instruction does not generate synchronous exceptions.

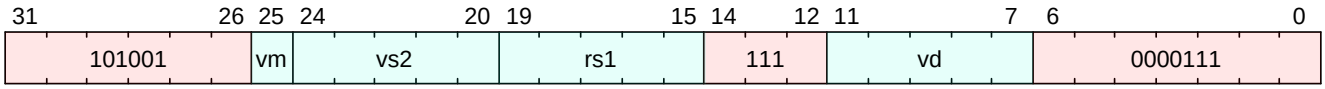
B.551. vluxseg6ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.551.1. Encoding



B.551.2. Synopsis

No description available.

B.551.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.551.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.551.5. Execution

```

```

B.551.6. Exceptions

This instruction does not generate synchronous exceptions.

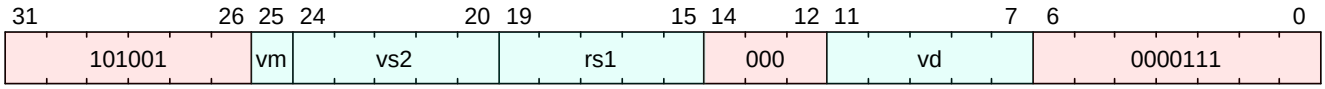
B.552. vluxseg6ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.552.1. Encoding



B.552.2. Synopsis

No description available.

B.552.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.552.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.552.5. Execution

```

```

B.552.6. Exceptions

This instruction does not generate synchronous exceptions.

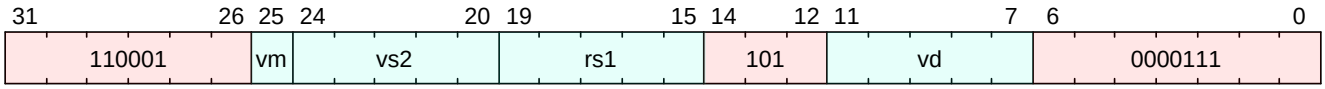
B.553. vluxseg7ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.553.1. Encoding



B.553.2. Synopsis

No description available.

B.553.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.553.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.553.5. Execution

```

```

B.553.6. Exceptions

This instruction does not generate synchronous exceptions.

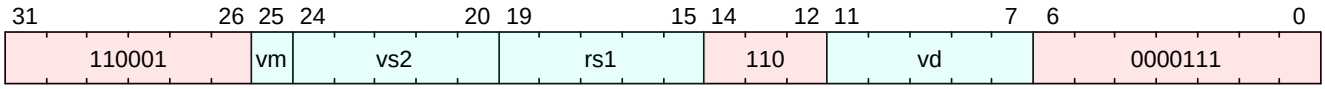
B.554. vluxseg7ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.554.1. Encoding



B.554.2. Synopsis

No description available.

B.554.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.554.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.554.5. Execution

```

```

B.554.6. Exceptions

This instruction does not generate synchronous exceptions.

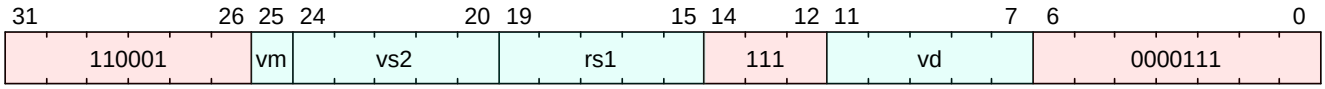
B.555. vluxseg7ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.555.1. Encoding



B.555.2. Synopsis

No description available.

B.555.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.555.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.555.5. Execution

```

```

B.555.6. Exceptions

This instruction does not generate synchronous exceptions.

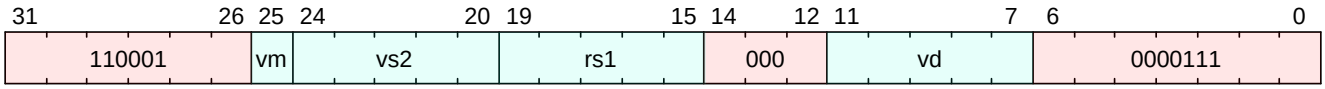
B.556. vluxseg7ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.556.1. Encoding



B.556.2. Synopsis

No description available.

B.556.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.556.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.556.5. Execution

```

```

B.556.6. Exceptions

This instruction does not generate synchronous exceptions.

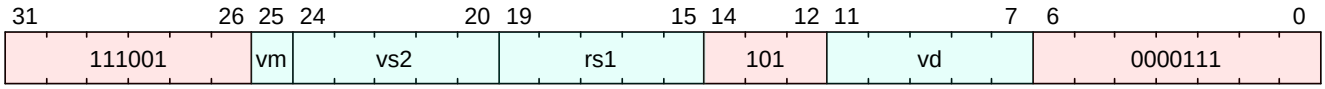
B.557. vluxseg8ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.557.1. Encoding



B.557.2. Synopsis

No description available.

B.557.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.557.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.557.5. Execution

```

```

B.557.6. Exceptions

This instruction does not generate synchronous exceptions.

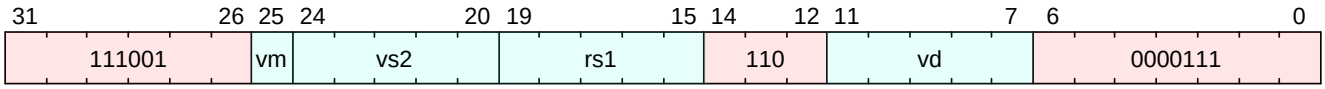
B.558. vluxseg8ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.558.1. Encoding



B.558.2. Synopsis

No description available.

B.558.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.558.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.558.5. Execution

```

```

B.558.6. Exceptions

This instruction does not generate synchronous exceptions.

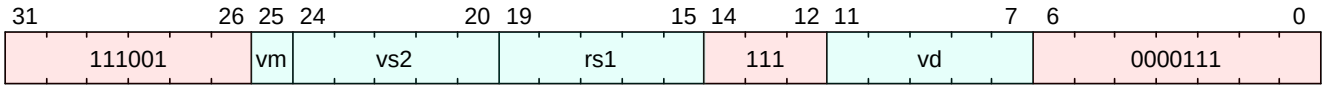
B.559. vluxseg8ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.559.1. Encoding



B.559.2. Synopsis

No description available.

B.559.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.559.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.559.5. Execution

```

```

B.559.6. Exceptions

This instruction does not generate synchronous exceptions.

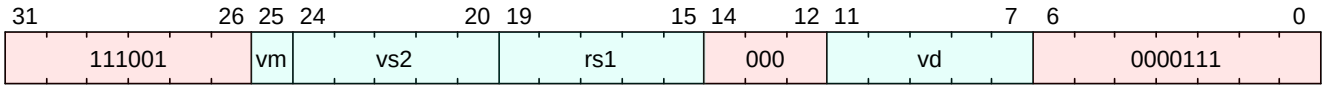
B.560. vluxseg8ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.560.1. Encoding



B.560.2. Synopsis

No description available.

B.560.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.560.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.560.5. Execution

```

```

B.560.6. Exceptions

This instruction does not generate synchronous exceptions.

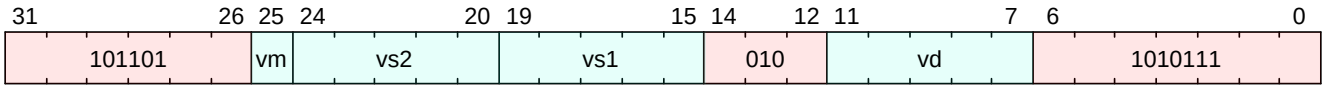
B.561. vmacc.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.561.1. Encoding



B.561.2. Synopsis

No description available.

B.561.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.561.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.561.5. Execution

```

```

B.561.6. Exceptions

This instruction does not generate synchronous exceptions.

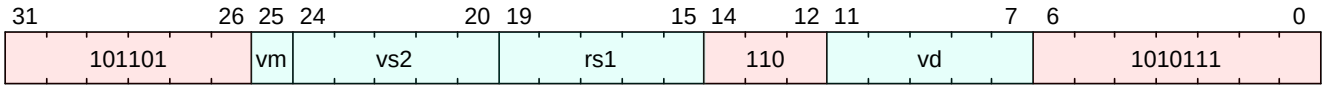
B.562. vmacc.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.562.1. Encoding



B.562.2. Synopsis

No description available.

B.562.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.562.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.562.5. Execution

```

```

B.562.6. Exceptions

This instruction does not generate synchronous exceptions.

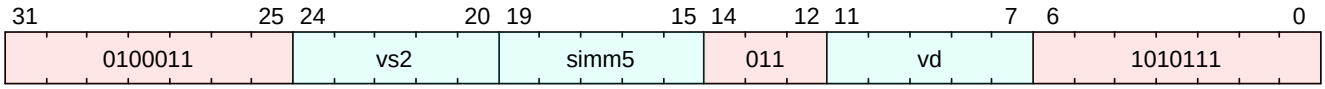
B.563. vmadc.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.563.1. Encoding



B.563.2. Synopsis

No description available.

B.563.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.563.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.563.5. Execution

```

```

B.563.6. Exceptions

This instruction does not generate synchronous exceptions.

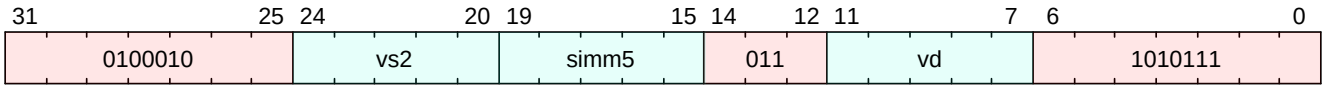
B.564. vmadc.vim

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.564.1. Encoding



B.564.2. Synopsis

No description available.

B.564.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.564.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.564.5. Execution

```

```

B.564.6. Exceptions

This instruction does not generate synchronous exceptions.

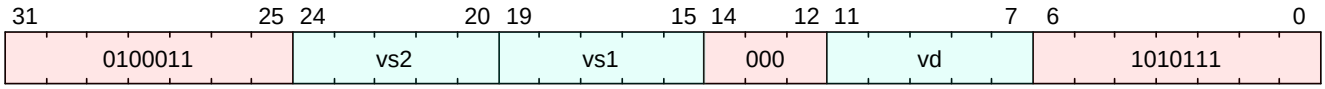
B.565. vmadc.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.565.1. Encoding



B.565.2. Synopsis

No description available.

B.565.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.565.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.565.5. Execution

```

```

B.565.6. Exceptions

This instruction does not generate synchronous exceptions.

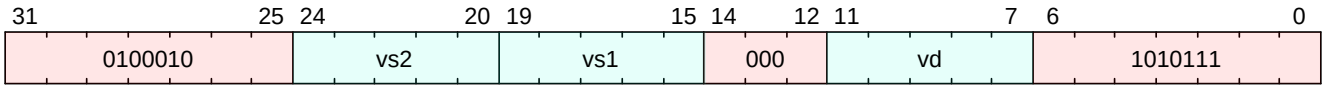
B.566. vmadc.vvm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.566.1. Encoding



B.566.2. Synopsis

No description available.

B.566.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.566.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.566.5. Execution

```

```

B.566.6. Exceptions

This instruction does not generate synchronous exceptions.

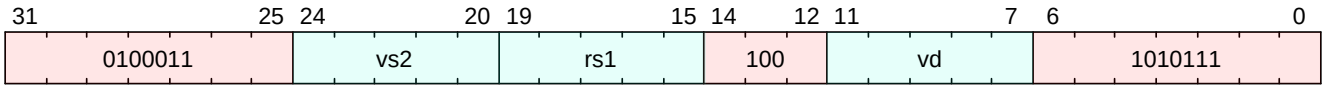
B.567. vmadc.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.567.1. Encoding



B.567.2. Synopsis

No description available.

B.567.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.567.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.567.5. Execution

```

```

B.567.6. Exceptions

This instruction does not generate synchronous exceptions.

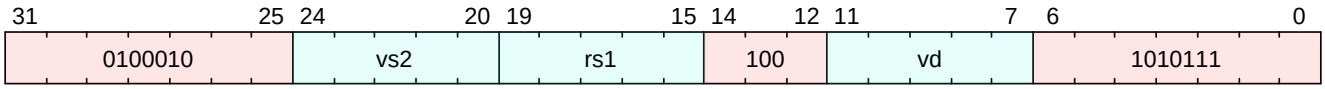
B.568. vmadc.vxm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.568.1. Encoding



B.568.2. Synopsis

No description available.

B.568.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.568.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.568.5. Execution

```

```

B.568.6. Exceptions

This instruction does not generate synchronous exceptions.

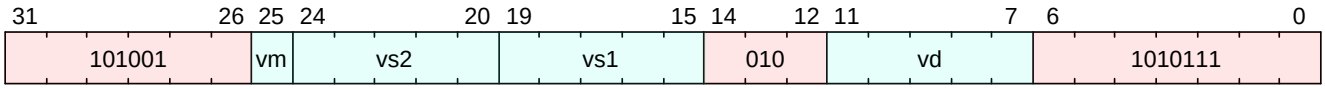
B.569. vmadd.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.569.1. Encoding



B.569.2. Synopsis

No description available.

B.569.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.569.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.569.5. Execution

```

```

B.569.6. Exceptions

This instruction does not generate synchronous exceptions.

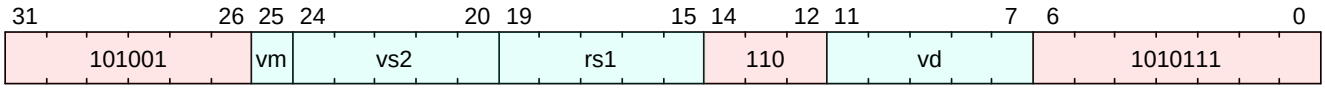
B.570. vmadd.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.570.1. Encoding



B.570.2. Synopsis

No description available.

B.570.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.570.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.570.5. Execution

```

```

B.570.6. Exceptions

This instruction does not generate synchronous exceptions.

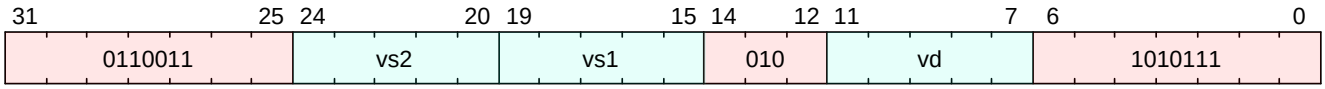
B.571. vmand.mm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.571.1. Encoding



B.571.2. Synopsis

No description available.

B.571.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.571.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.571.5. Execution

```

```

B.571.6. Exceptions

This instruction does not generate synchronous exceptions.

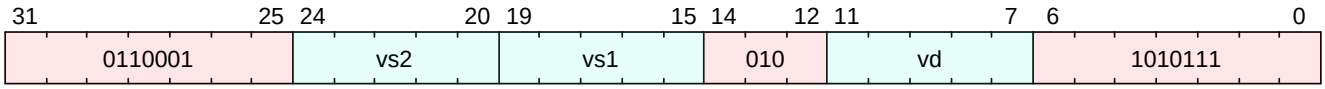
B.572. vmandn.mm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.572.1. Encoding



B.572.2. Synopsis

No description available.

B.572.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.572.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.572.5. Execution

```

```

B.572.6. Exceptions

This instruction does not generate synchronous exceptions.

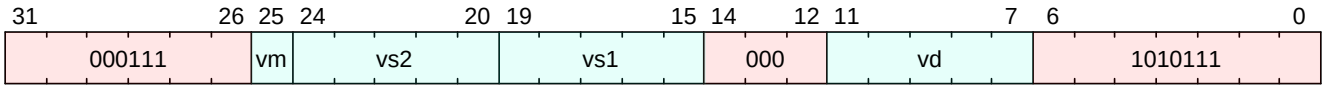
B.573. vmax.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.573.1. Encoding



B.573.2. Synopsis

No description available.

B.573.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.573.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.573.5. Execution

```

```

B.573.6. Exceptions

This instruction does not generate synchronous exceptions.

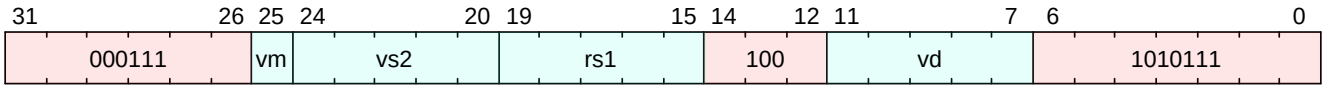
B.574. vmax.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.574.1. Encoding



B.574.2. Synopsis

No description available.

B.574.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.574.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.574.5. Execution

```

```

B.574.6. Exceptions

This instruction does not generate synchronous exceptions.

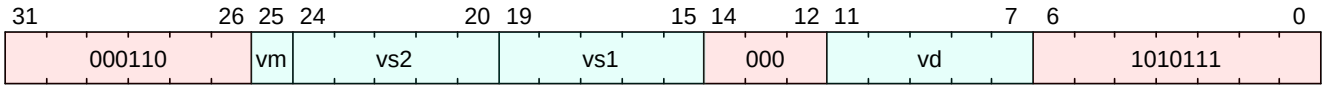
B.575. vmaxu.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.575.1. Encoding



B.575.2. Synopsis

No description available.

B.575.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.575.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.575.5. Execution

```

```

B.575.6. Exceptions

This instruction does not generate synchronous exceptions.

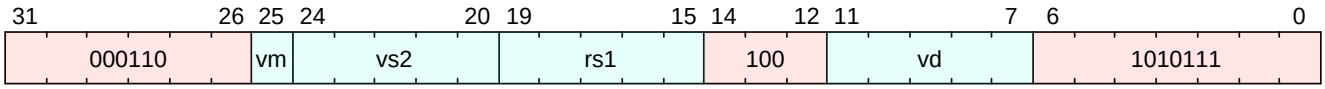
B.576. vmaxu.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.576.1. Encoding



B.576.2. Synopsis

No description available.

B.576.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.576.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.576.5. Execution

```

```

B.576.6. Exceptions

This instruction does not generate synchronous exceptions.

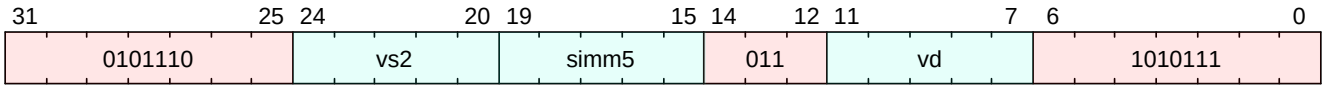
B.577. vmerge.vim

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.577.1. Encoding



B.577.2. Synopsis

No description available.

B.577.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.577.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.577.5. Execution

```

```

B.577.6. Exceptions

This instruction does not generate synchronous exceptions.

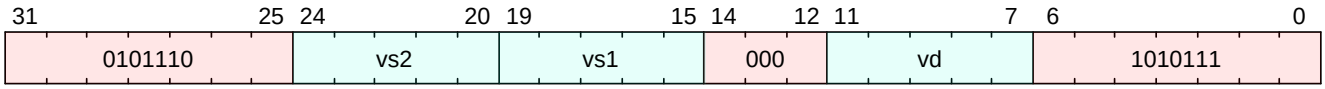
B.578. vmerge.vvm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.578.1. Encoding



B.578.2. Synopsis

No description available.

B.578.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.578.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.578.5. Execution

```

```

B.578.6. Exceptions

This instruction does not generate synchronous exceptions.

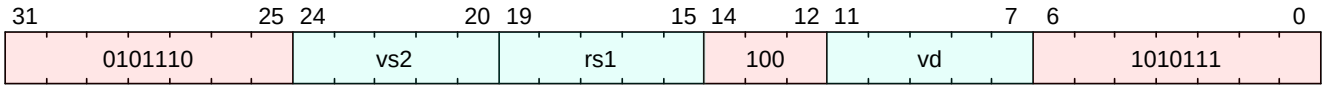
B.579. vmerge.vxm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.579.1. Encoding



B.579.2. Synopsis

No description available.

B.579.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.579.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.579.5. Execution

```

```

B.579.6. Exceptions

This instruction does not generate synchronous exceptions.

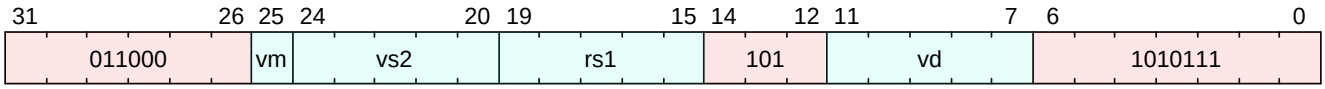
B.580. vmfeq.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.580.1. Encoding



B.580.2. Synopsis

No description available.

B.580.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.580.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.580.5. Execution

```

```

B.580.6. Exceptions

This instruction does not generate synchronous exceptions.

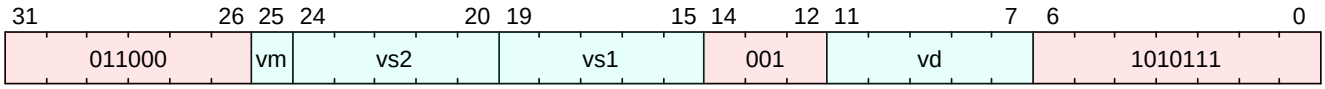
B.581. vmfeq.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.581.1. Encoding



B.581.2. Synopsis

No description available.

B.581.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.581.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.581.5. Execution

```

```

B.581.6. Exceptions

This instruction does not generate synchronous exceptions.

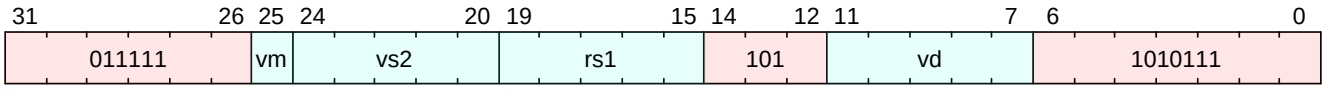
B.582. vmfge.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.582.1. Encoding



B.582.2. Synopsis

No description available.

B.582.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.582.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.582.5. Execution

```

```

B.582.6. Exceptions

This instruction does not generate synchronous exceptions.

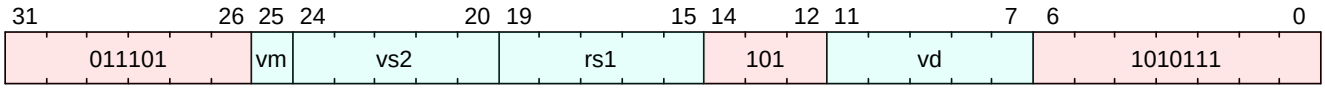
B.583. vmfgt.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.583.1. Encoding



B.583.2. Synopsis

No description available.

B.583.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.583.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.583.5. Execution

```

```

B.583.6. Exceptions

This instruction does not generate synchronous exceptions.

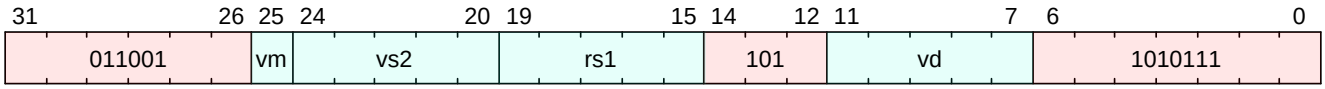
B.584. vmfle.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.584.1. Encoding



B.584.2. Synopsis

No description available.

B.584.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.584.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.584.5. Execution

```

```

B.584.6. Exceptions

This instruction does not generate synchronous exceptions.

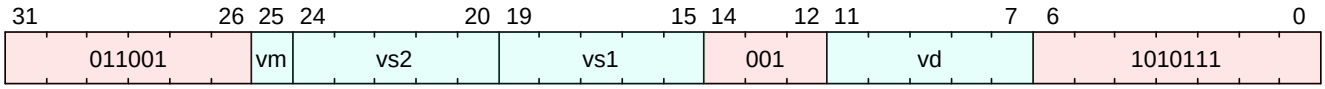
B.585. vmfle.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.585.1. Encoding



B.585.2. Synopsis

No description available.

B.585.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.585.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.585.5. Execution

```

```

B.585.6. Exceptions

This instruction does not generate synchronous exceptions.

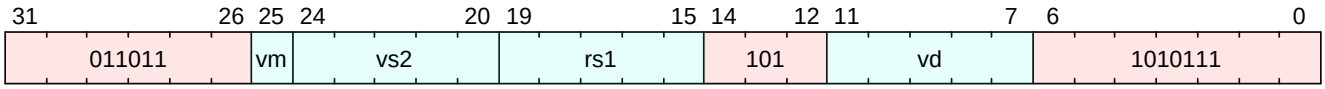
B.586. vmflt.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.586.1. Encoding



B.586.2. Synopsis

No description available.

B.586.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.586.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.586.5. Execution

```

```

B.586.6. Exceptions

This instruction does not generate synchronous exceptions.

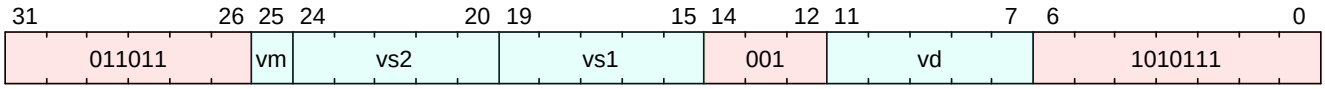
B.587. vmflt.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.587.1. Encoding



B.587.2. Synopsis

No description available.

B.587.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.587.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.587.5. Execution

```

```

B.587.6. Exceptions

This instruction does not generate synchronous exceptions.

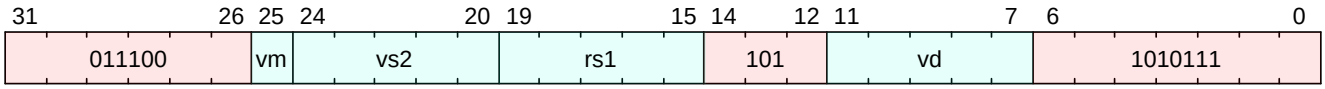
B.588. vmfne.vf

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.588.1. Encoding



B.588.2. Synopsis

No description available.

B.588.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.588.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.588.5. Execution

```

```

B.588.6. Exceptions

This instruction does not generate synchronous exceptions.

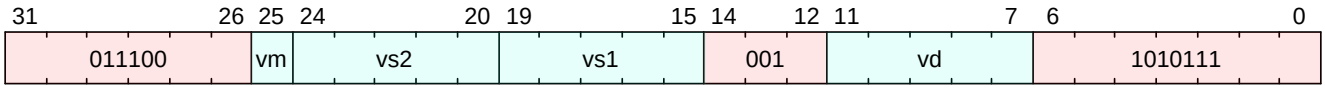
B.589. vmfne.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.589.1. Encoding



B.589.2. Synopsis

No description available.

B.589.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.589.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.589.5. Execution

```

```

B.589.6. Exceptions

This instruction does not generate synchronous exceptions.

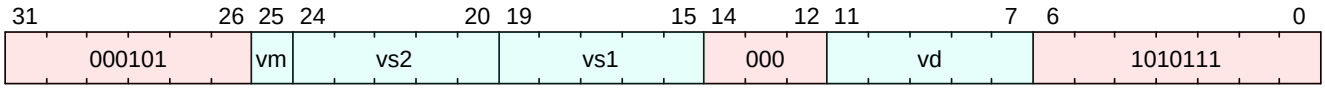
B.590. vmin.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.590.1. Encoding



B.590.2. Synopsis

No description available.

B.590.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.590.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.590.5. Execution

```

```

B.590.6. Exceptions

This instruction does not generate synchronous exceptions.

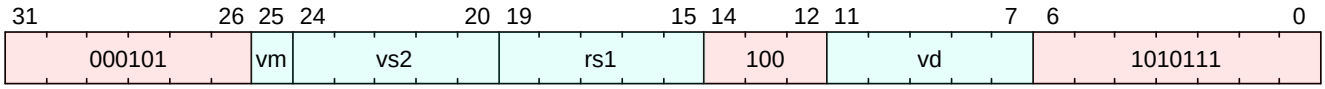
B.591. vmin.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.591.1. Encoding



B.591.2. Synopsis

No description available.

B.591.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.591.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.591.5. Execution

```

```

B.591.6. Exceptions

This instruction does not generate synchronous exceptions.

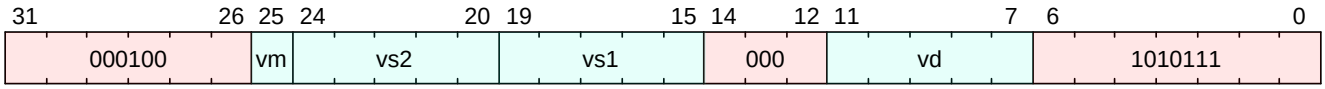
B.592. vminu.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.592.1. Encoding



B.592.2. Synopsis

No description available.

B.592.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.592.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.592.5. Execution

```

```

B.592.6. Exceptions

This instruction does not generate synchronous exceptions.

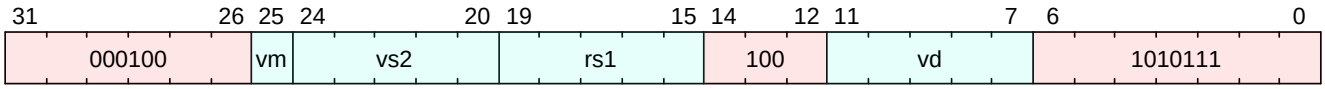
B.593. vminu.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.593.1. Encoding



B.593.2. Synopsis

No description available.

B.593.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.593.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.593.5. Execution

```

```

B.593.6. Exceptions

This instruction does not generate synchronous exceptions.

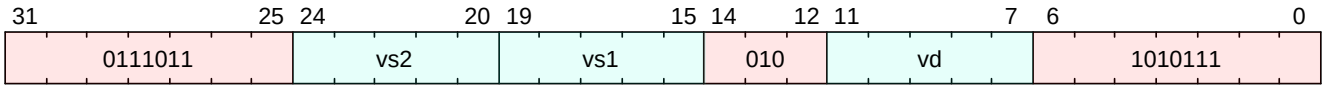
B.594. vmnand.mm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.594.1. Encoding



B.594.2. Synopsis

No description available.

B.594.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.594.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.594.5. Execution

```

```

B.594.6. Exceptions

This instruction does not generate synchronous exceptions.

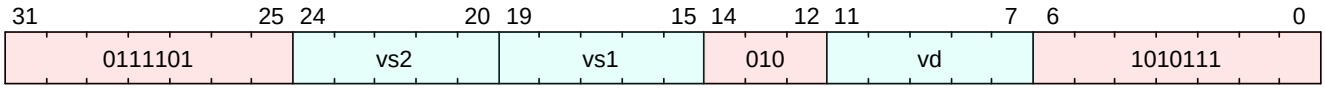
B.595. vmnor.mm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.595.1. Encoding



B.595.2. Synopsis

No description available.

B.595.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.595.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.595.5. Execution

```

```

B.595.6. Exceptions

This instruction does not generate synchronous exceptions.

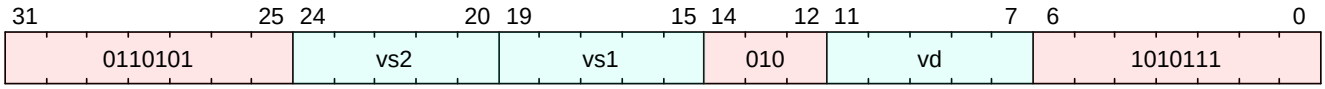
B.596. vmor.mm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.596.1. Encoding



B.596.2. Synopsis

No description available.

B.596.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.596.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.596.5. Execution

```

```

B.596.6. Exceptions

This instruction does not generate synchronous exceptions.

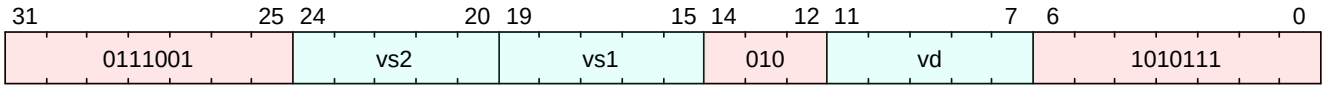
B.597. vmorn.mm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.597.1. Encoding



B.597.2. Synopsis

No description available.

B.597.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.597.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.597.5. Execution

```

```

B.597.6. Exceptions

This instruction does not generate synchronous exceptions.

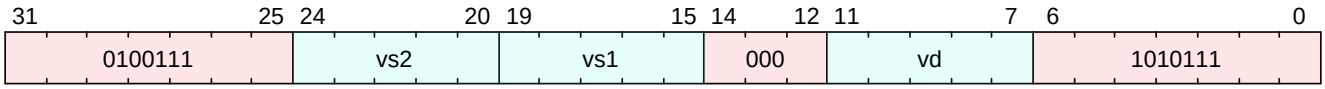
B.598. vmsbc.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.598.1. Encoding



B.598.2. Synopsis

No description available.

B.598.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.598.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.598.5. Execution

```

```

B.598.6. Exceptions

This instruction does not generate synchronous exceptions.

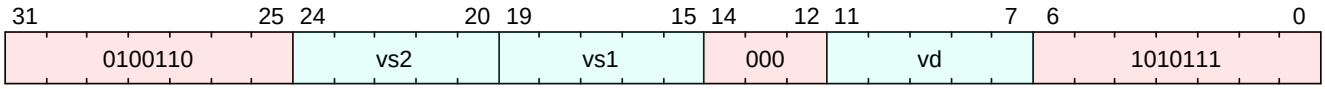
B.599. vmsbc.vvm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.599.1. Encoding



B.599.2. Synopsis

No description available.

B.599.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.599.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.599.5. Execution

```

```

B.599.6. Exceptions

This instruction does not generate synchronous exceptions.

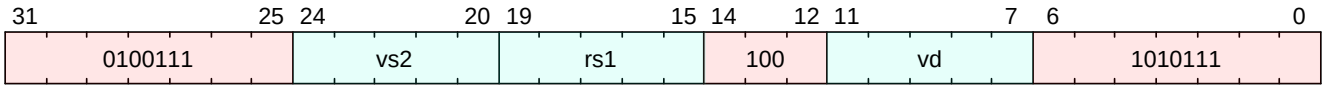
B.600. vmsbc.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.600.1. Encoding



B.600.2. Synopsis

No description available.

B.600.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.600.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.600.5. Execution

```

```

B.600.6. Exceptions

This instruction does not generate synchronous exceptions.

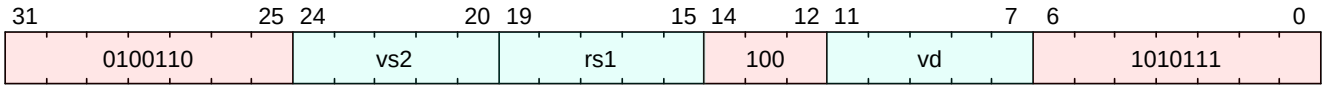
B.601. vmsbc.vxm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.601.1. Encoding



B.601.2. Synopsis

No description available.

B.601.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.601.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.601.5. Execution

```

```

B.601.6. Exceptions

This instruction does not generate synchronous exceptions.

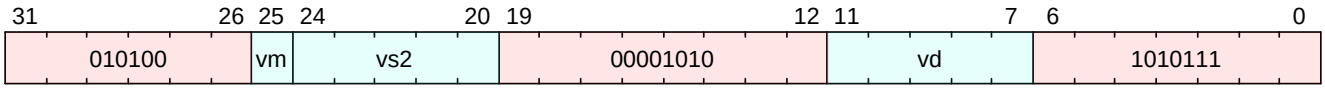
B.602. vmsbf.m

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.602.1. Encoding



B.602.2. Synopsis

No description available.

B.602.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.602.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.602.5. Execution

```

```

B.602.6. Exceptions

This instruction does not generate synchronous exceptions.

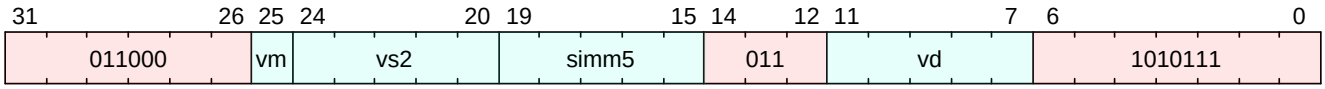
B.603. vmseq.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.603.1. Encoding



B.603.2. Synopsis

No description available.

B.603.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.603.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.603.5. Execution

```

```

B.603.6. Exceptions

This instruction does not generate synchronous exceptions.

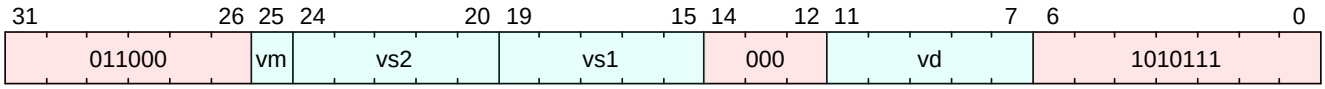
B.604. vmseq.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.604.1. Encoding



B.604.2. Synopsis

No description available.

B.604.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.604.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.604.5. Execution

```

```

B.604.6. Exceptions

This instruction does not generate synchronous exceptions.

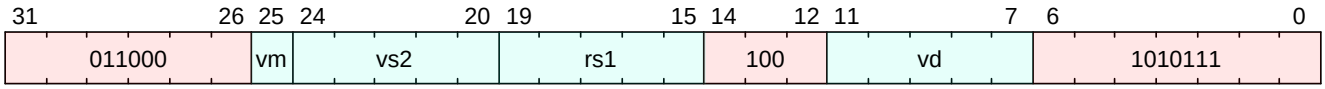
B.605. vmseq.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.605.1. Encoding



B.605.2. Synopsis

No description available.

B.605.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.605.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.605.5. Execution

```

```

B.605.6. Exceptions

This instruction does not generate synchronous exceptions.

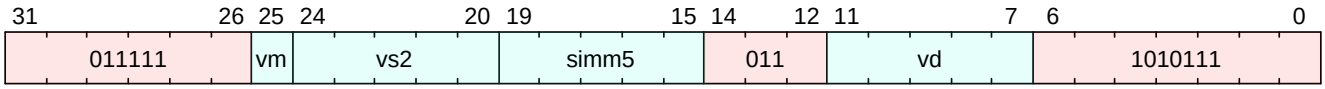
B.606. vmsgt.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.606.1. Encoding



B.606.2. Synopsis

No description available.

B.606.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.606.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.606.5. Execution

```

```

B.606.6. Exceptions

This instruction does not generate synchronous exceptions.

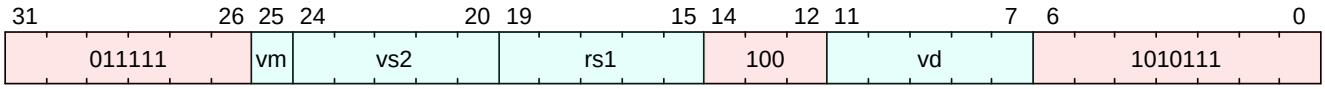
B.607. vmsgt.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.607.1. Encoding



B.607.2. Synopsis

No description available.

B.607.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.607.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.607.5. Execution

```

```

B.607.6. Exceptions

This instruction does not generate synchronous exceptions.

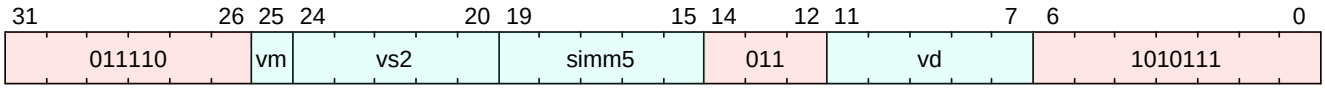
B.608. vmsgtu.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.608.1. Encoding



B.608.2. Synopsis

No description available.

B.608.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.608.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.608.5. Execution

```

```

B.608.6. Exceptions

This instruction does not generate synchronous exceptions.

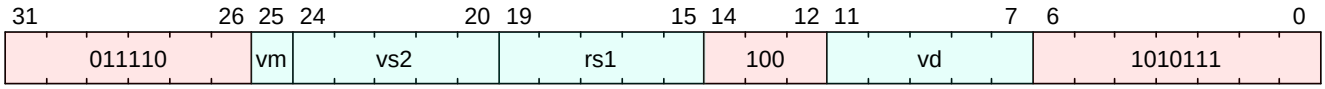
B.609. vmsgtu.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.609.1. Encoding



B.609.2. Synopsis

No description available.

B.609.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.609.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.609.5. Execution

```

```

B.609.6. Exceptions

This instruction does not generate synchronous exceptions.

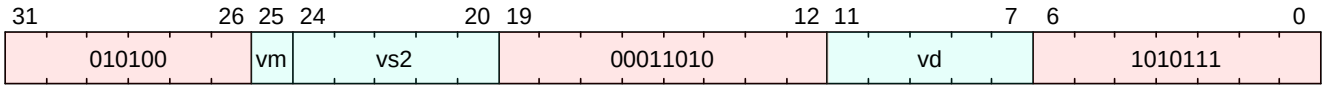
B.610. vmsif.m

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.610.1. Encoding



B.610.2. Synopsis

No description available.

B.610.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.610.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.610.5. Execution

```

```

B.610.6. Exceptions

This instruction does not generate synchronous exceptions.

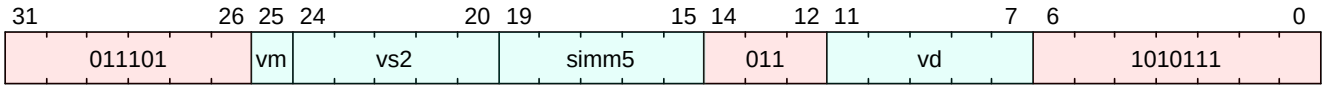
B.611. vmsle.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.611.1. Encoding



B.611.2. Synopsis

No description available.

B.611.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.611.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.611.5. Execution

```

```

B.611.6. Exceptions

This instruction does not generate synchronous exceptions.

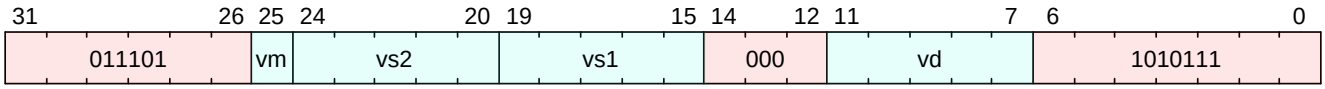
B.612. vmsle.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.612.1. Encoding



B.612.2. Synopsis

No description available.

B.612.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.612.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.612.5. Execution

```

```

B.612.6. Exceptions

This instruction does not generate synchronous exceptions.

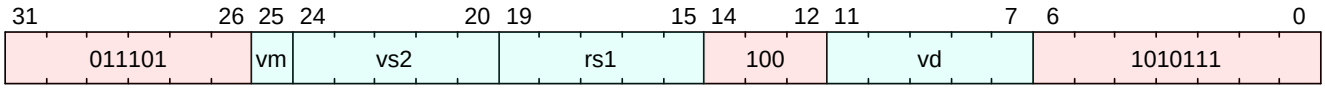
B.613. vmsle.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.613.1. Encoding



B.613.2. Synopsis

No description available.

B.613.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.613.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.613.5. Execution

```

```

B.613.6. Exceptions

This instruction does not generate synchronous exceptions.

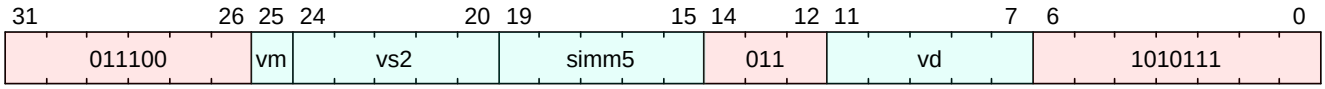
B.614. vmsleu.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.614.1. Encoding



B.614.2. Synopsis

No description available.

B.614.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.614.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.614.5. Execution

```

```

B.614.6. Exceptions

This instruction does not generate synchronous exceptions.

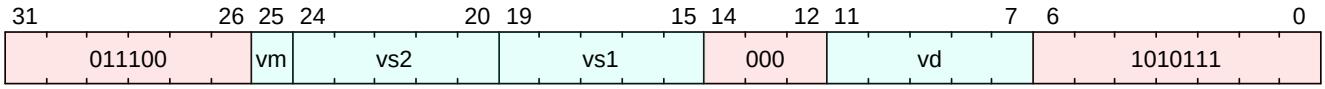
B.615. vmsleu.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.615.1. Encoding



B.615.2. Synopsis

No description available.

B.615.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.615.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.615.5. Execution

```

```

B.615.6. Exceptions

This instruction does not generate synchronous exceptions.

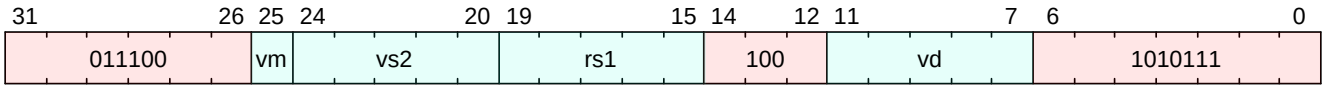
B.616. vmsleu.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.616.1. Encoding



B.616.2. Synopsis

No description available.

B.616.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.616.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.616.5. Execution

```

```

B.616.6. Exceptions

This instruction does not generate synchronous exceptions.

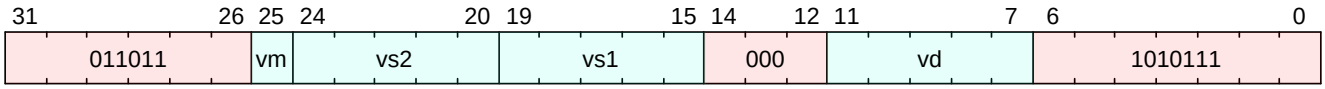
B.617. vmslt.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.617.1. Encoding



B.617.2. Synopsis

No description available.

B.617.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.617.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.617.5. Execution

```

```

B.617.6. Exceptions

This instruction does not generate synchronous exceptions.

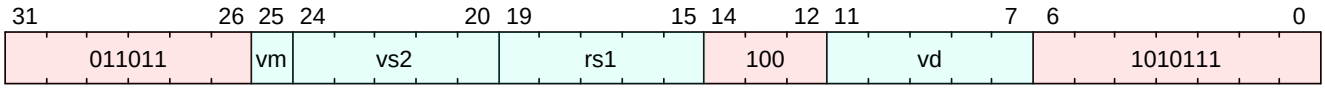
B.618. vmslt.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.618.1. Encoding



B.618.2. Synopsis

No description available.

B.618.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.618.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.618.5. Execution

```

```

B.618.6. Exceptions

This instruction does not generate synchronous exceptions.

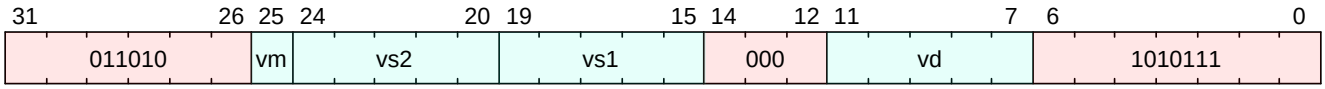
B.619. vmsltu.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.619.1. Encoding



B.619.2. Synopsis

No description available.

B.619.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.619.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.619.5. Execution

```

```

B.619.6. Exceptions

This instruction does not generate synchronous exceptions.

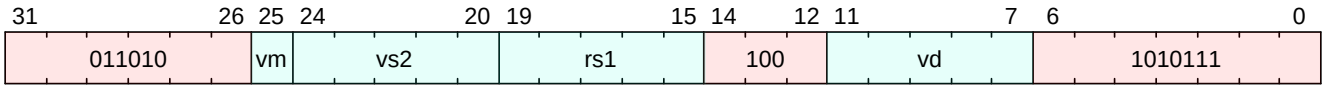
B.620. vmsltu.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.620.1. Encoding



B.620.2. Synopsis

No description available.

B.620.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.620.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.620.5. Execution

```

```

B.620.6. Exceptions

This instruction does not generate synchronous exceptions.

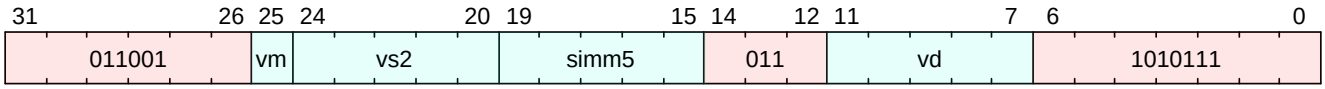
B.621. vmsne.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.621.1. Encoding



B.621.2. Synopsis

No description available.

B.621.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.621.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.621.5. Execution

```

```

B.621.6. Exceptions

This instruction does not generate synchronous exceptions.

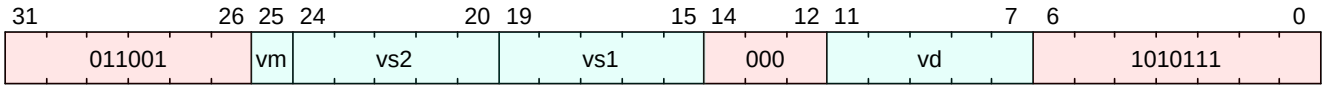
B.622. vmsne.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.622.1. Encoding



B.622.2. Synopsis

No description available.

B.622.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.622.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.622.5. Execution

```

```

B.622.6. Exceptions

This instruction does not generate synchronous exceptions.

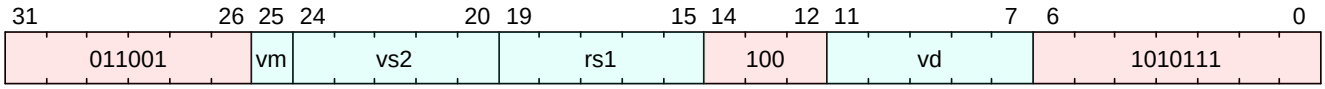
B.623. vmsne.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.623.1. Encoding



B.623.2. Synopsis

No description available.

B.623.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.623.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.623.5. Execution

```

```

B.623.6. Exceptions

This instruction does not generate synchronous exceptions.

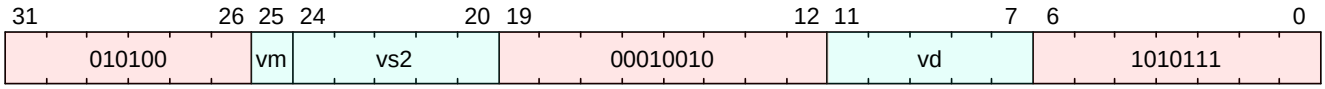
B.624. vmsof.m

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.624.1. Encoding



B.624.2. Synopsis

No description available.

B.624.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.624.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.624.5. Execution

```

```

B.624.6. Exceptions

This instruction does not generate synchronous exceptions.

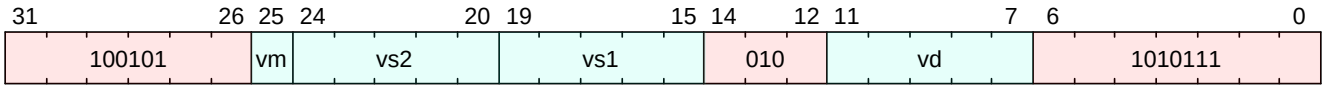
B.625. vmul.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.625.1. Encoding



B.625.2. Synopsis

No description available.

B.625.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.625.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.625.5. Execution

```

```

B.625.6. Exceptions

This instruction does not generate synchronous exceptions.

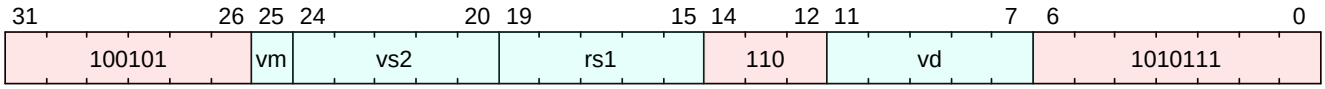
B.626. vmul.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.626.1. Encoding



B.626.2. Synopsis

No description available.

B.626.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.626.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.626.5. Execution

```

```

B.626.6. Exceptions

This instruction does not generate synchronous exceptions.

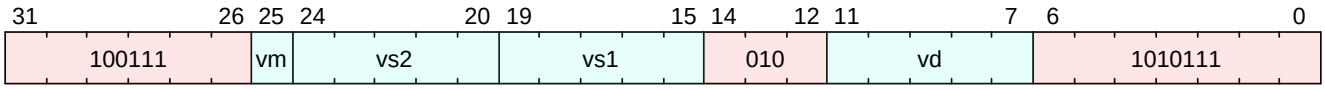
B.627. vmulh.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.627.1. Encoding



B.627.2. Synopsis

No description available.

B.627.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.627.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.627.5. Execution

```

```

B.627.6. Exceptions

This instruction does not generate synchronous exceptions.

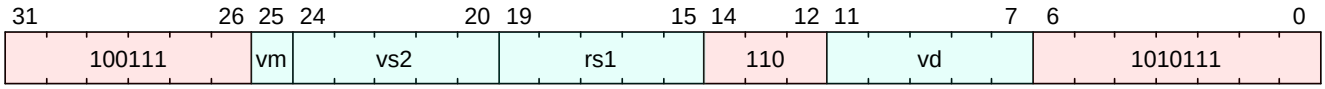
B.628. vmulh.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.628.1. Encoding



B.628.2. Synopsis

No description available.

B.628.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.628.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.628.5. Execution

```

```

B.628.6. Exceptions

This instruction does not generate synchronous exceptions.

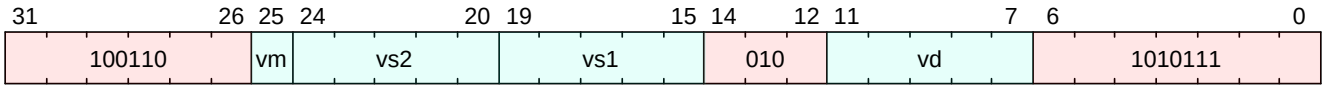
B.629. vmulhsu.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.629.1. Encoding



B.629.2. Synopsis

No description available.

B.629.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.629.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.629.5. Execution

```

```

B.629.6. Exceptions

This instruction does not generate synchronous exceptions.

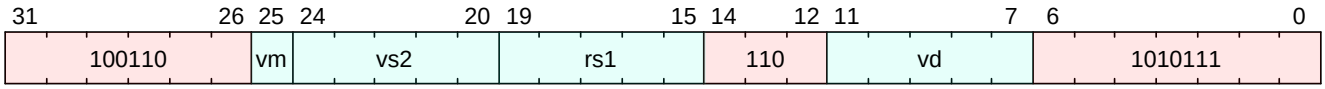
B.630. vmulhsu.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.630.1. Encoding



B.630.2. Synopsis

No description available.

B.630.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.630.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.630.5. Execution

```

```

B.630.6. Exceptions

This instruction does not generate synchronous exceptions.

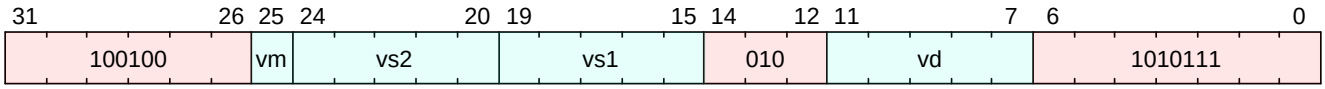
B.631. vmulhu.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.631.1. Encoding



B.631.2. Synopsis

No description available.

B.631.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.631.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.631.5. Execution

```

```

B.631.6. Exceptions

This instruction does not generate synchronous exceptions.

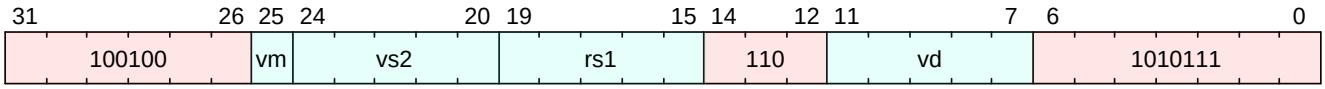
B.632. vmulhu.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.632.1. Encoding



B.632.2. Synopsis

No description available.

B.632.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.632.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.632.5. Execution

```

```

B.632.6. Exceptions

This instruction does not generate synchronous exceptions.

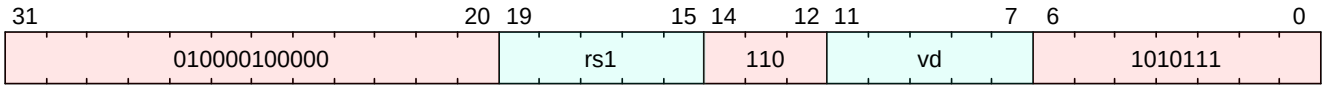
B.633. vmv.s.x

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.633.1. Encoding



B.633.2. Synopsis

No description available.

B.633.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.633.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.633.5. Execution

```

```

B.633.6. Exceptions

This instruction does not generate synchronous exceptions.

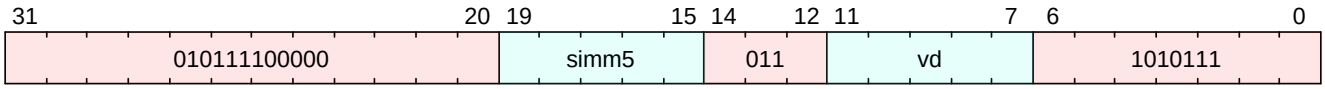
B.634. vmv.v.i

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.634.1. Encoding



B.634.2. Synopsis

No description available.

B.634.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.634.4. Decode Variables

```
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.634.5. Execution

```

```

B.634.6. Exceptions

This instruction does not generate synchronous exceptions.

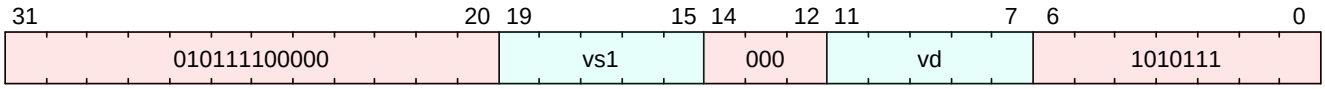
B.635. vmv.v.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.635.1. Encoding



B.635.2. Synopsis

No description available.

B.635.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.635.4. Decode Variables

```
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.635.5. Execution

```

```

B.635.6. Exceptions

This instruction does not generate synchronous exceptions.

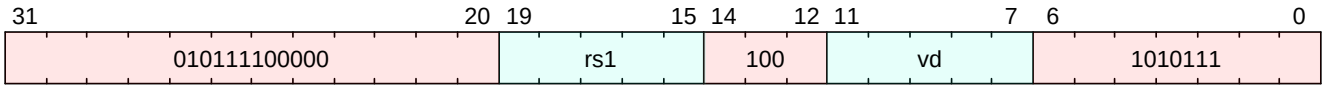
B.636. vmv.v.x

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.636.1. Encoding



B.636.2. Synopsis

No description available.

B.636.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.636.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.636.5. Execution

```

```

B.636.6. Exceptions

This instruction does not generate synchronous exceptions.

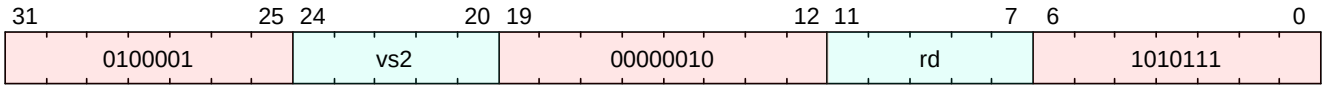
B.637. vmv.x.s

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.637.1. Encoding



B.637.2. Synopsis

No description available.

B.637.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.637.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> rd = $encoding[11:7];
```

B.637.5. Execution

```

```

B.637.6. Exceptions

This instruction does not generate synchronous exceptions.

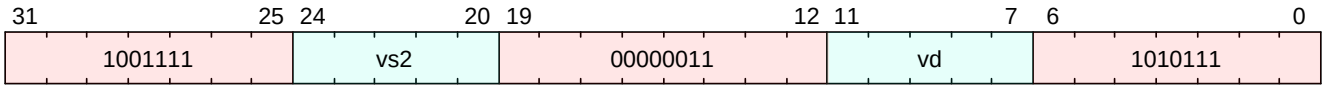
B.638. vmv1r.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.638.1. Encoding



B.638.2. Synopsis

No description available.

B.638.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.638.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.638.5. Execution

```

```

B.638.6. Exceptions

This instruction does not generate synchronous exceptions.

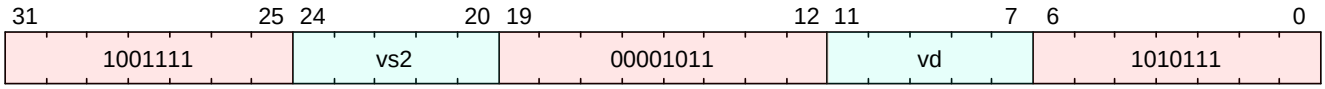
B.639. vmv2r.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.639.1. Encoding



B.639.2. Synopsis

No description available.

B.639.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.639.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.639.5. Execution

```

```

B.639.6. Exceptions

This instruction does not generate synchronous exceptions.

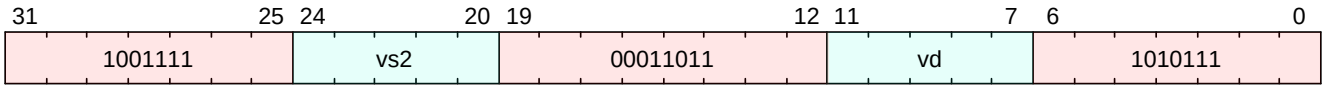
B.640. vmv4r.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.640.1. Encoding



B.640.2. Synopsis

No description available.

B.640.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.640.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.640.5. Execution

```

```

B.640.6. Exceptions

This instruction does not generate synchronous exceptions.

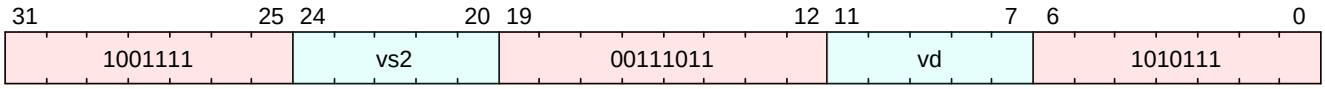
B.641. vmv8r.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.641.1. Encoding



B.641.2. Synopsis

No description available.

B.641.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.641.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.641.5. Execution

```

```

B.641.6. Exceptions

This instruction does not generate synchronous exceptions.

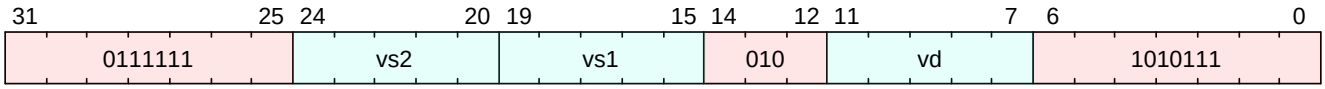
B.642. vmxnor.mm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.642.1. Encoding



B.642.2. Synopsis

No description available.

B.642.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.642.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.642.5. Execution

```

```

B.642.6. Exceptions

This instruction does not generate synchronous exceptions.

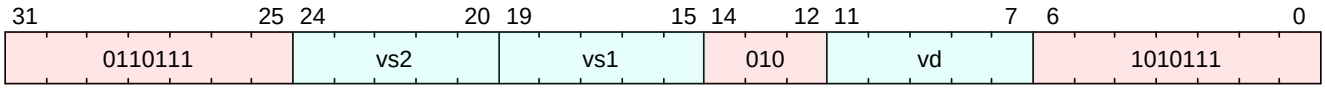
B.643. vmxor.mm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.643.1. Encoding



B.643.2. Synopsis

No description available.

B.643.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.643.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.643.5. Execution

```

```

B.643.6. Exceptions

This instruction does not generate synchronous exceptions.

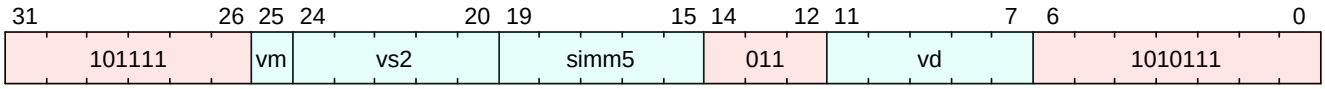
B.644. vnclip.wi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.644.1. Encoding



B.644.2. Synopsis

No description available.

B.644.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.644.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.644.5. Execution

```

```

B.644.6. Exceptions

This instruction does not generate synchronous exceptions.

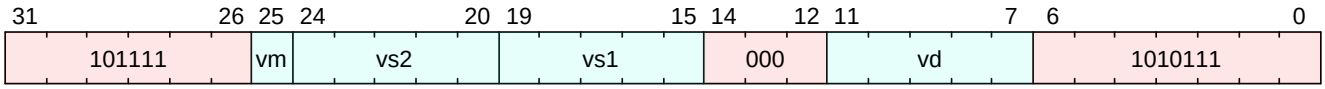
B.645. vnclip.wv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.645.1. Encoding



B.645.2. Synopsis

No description available.

B.645.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.645.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.645.5. Execution

```

```

B.645.6. Exceptions

This instruction does not generate synchronous exceptions.

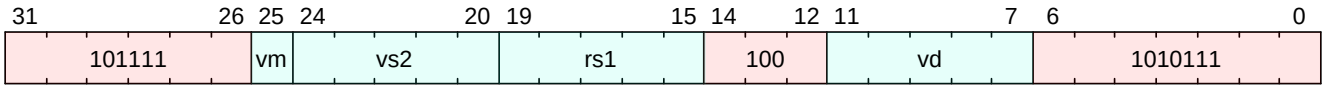
B.646. vnclip.wx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.646.1. Encoding



B.646.2. Synopsis

No description available.

B.646.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.646.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.646.5. Execution

```

```

B.646.6. Exceptions

This instruction does not generate synchronous exceptions.

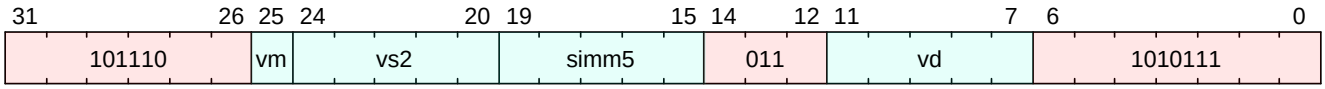
B.647. vnclipu.wi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.647.1. Encoding



B.647.2. Synopsis

No description available.

B.647.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.647.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.647.5. Execution

```

```

B.647.6. Exceptions

This instruction does not generate synchronous exceptions.

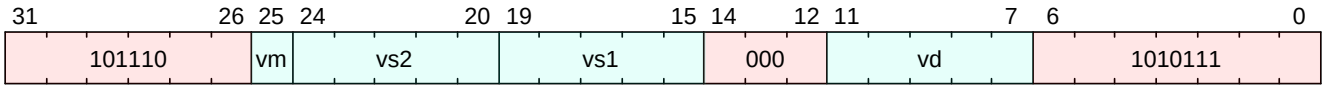
B.648. vnclipu.wv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.648.1. Encoding



B.648.2. Synopsis

No description available.

B.648.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.648.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.648.5. Execution

```

```

B.648.6. Exceptions

This instruction does not generate synchronous exceptions.

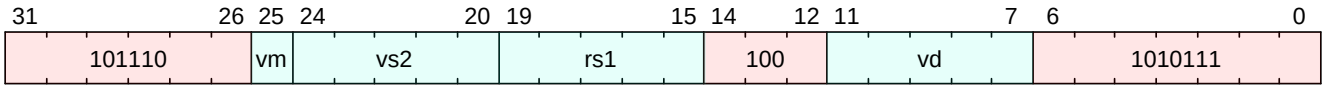
B.649. vnclipu.wx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.649.1. Encoding



B.649.2. Synopsis

No description available.

B.649.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.649.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.649.5. Execution

```

```

B.649.6. Exceptions

This instruction does not generate synchronous exceptions.

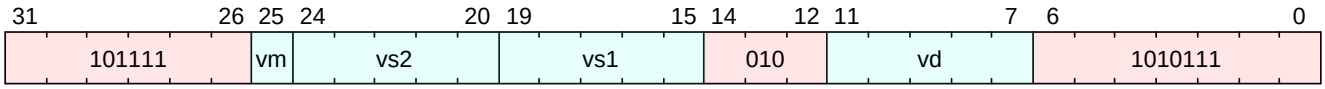
B.650. vnmsac.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.650.1. Encoding



B.650.2. Synopsis

No description available.

B.650.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.650.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.650.5. Execution

```

```

B.650.6. Exceptions

This instruction does not generate synchronous exceptions.

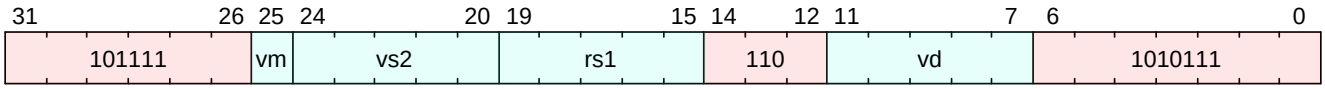
B.651. vnmsac.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.651.1. Encoding



B.651.2. Synopsis

No description available.

B.651.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.651.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.651.5. Execution

```

```

B.651.6. Exceptions

This instruction does not generate synchronous exceptions.

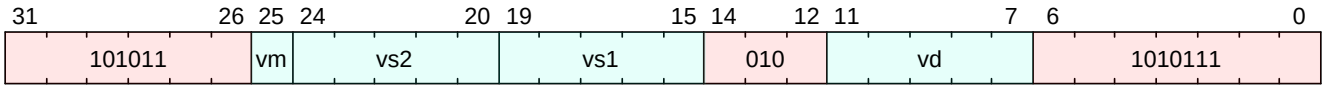
B.652. vnmsub.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.652.1. Encoding



B.652.2. Synopsis

No description available.

B.652.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.652.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.652.5. Execution

```

```

B.652.6. Exceptions

This instruction does not generate synchronous exceptions.

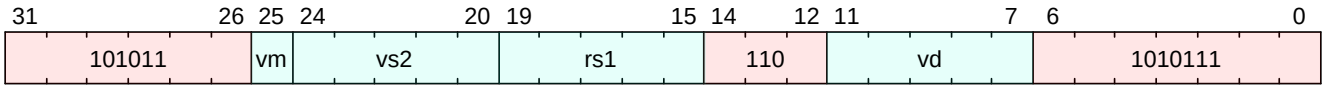
B.653. vnmsub.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.653.1. Encoding



B.653.2. Synopsis

No description available.

B.653.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.653.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.653.5. Execution

```

```

B.653.6. Exceptions

This instruction does not generate synchronous exceptions.

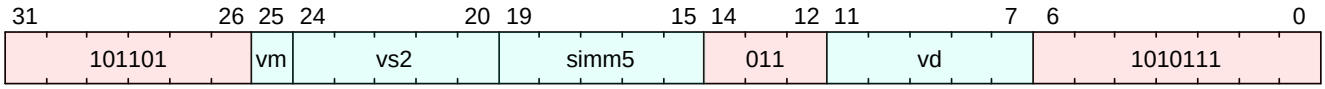
B.654. vnsra.wi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.654.1. Encoding



B.654.2. Synopsis

No description available.

B.654.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.654.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.654.5. Execution

```

```

B.654.6. Exceptions

This instruction does not generate synchronous exceptions.

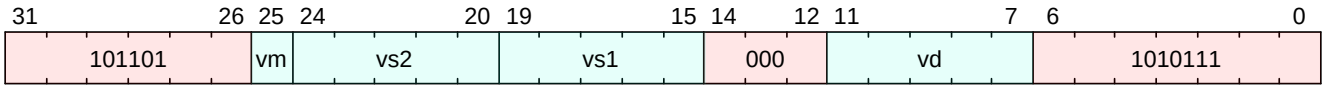
B.655. vnsra.wv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.655.1. Encoding



B.655.2. Synopsis

No description available.

B.655.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.655.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.655.5. Execution

```

```

B.655.6. Exceptions

This instruction does not generate synchronous exceptions.

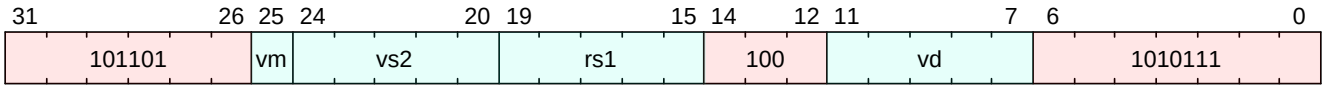
B.656. vnsra.wx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.656.1. Encoding



B.656.2. Synopsis

No description available.

B.656.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.656.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.656.5. Execution

```

```

B.656.6. Exceptions

This instruction does not generate synchronous exceptions.

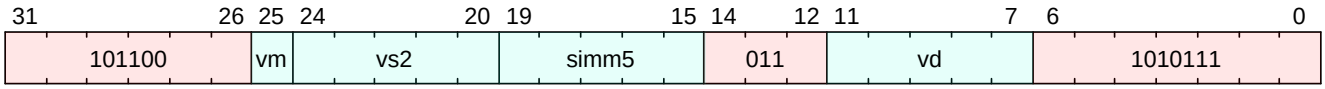
B.657. vnsrl.wi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.657.1. Encoding



B.657.2. Synopsis

No description available.

B.657.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.657.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.657.5. Execution

```

```

B.657.6. Exceptions

This instruction does not generate synchronous exceptions.

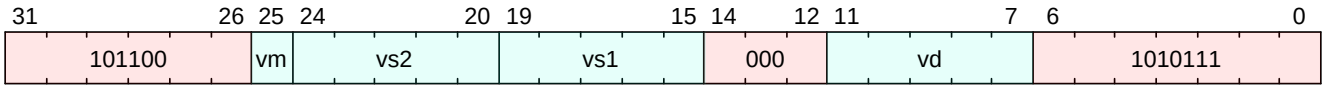
B.658. vnsrl.wv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.658.1. Encoding



B.658.2. Synopsis

No description available.

B.658.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.658.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.658.5. Execution

```

```

B.658.6. Exceptions

This instruction does not generate synchronous exceptions.

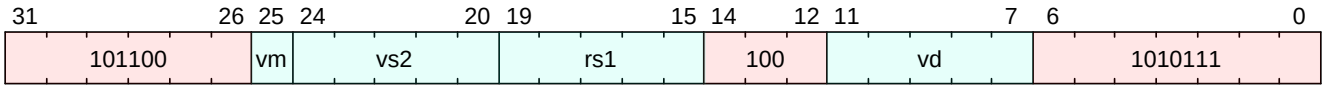
B.659. vnsrl.wx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.659.1. Encoding



B.659.2. Synopsis

No description available.

B.659.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.659.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.659.5. Execution

```

```

B.659.6. Exceptions

This instruction does not generate synchronous exceptions.

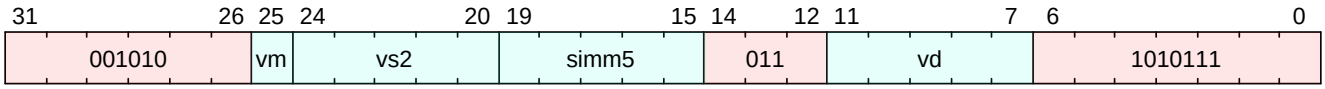
B.660. vor.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.660.1. Encoding



B.660.2. Synopsis

No description available.

B.660.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.660.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.660.5. Execution

```

```

B.660.6. Exceptions

This instruction does not generate synchronous exceptions.

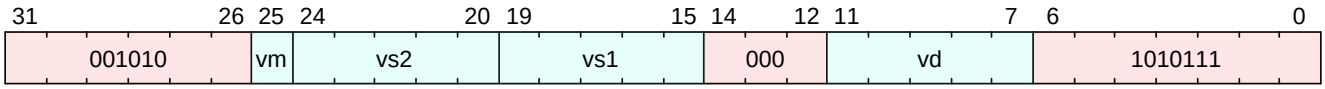
B.661. vor.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.661.1. Encoding



B.661.2. Synopsis

No description available.

B.661.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.661.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.661.5. Execution

```

```

B.661.6. Exceptions

This instruction does not generate synchronous exceptions.

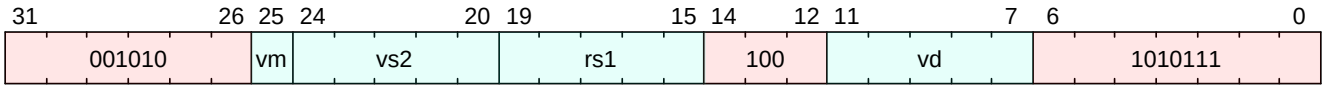
B.662. vor.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.662.1. Encoding



B.662.2. Synopsis

No description available.

B.662.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.662.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.662.5. Execution

```

```

B.662.6. Exceptions

This instruction does not generate synchronous exceptions.

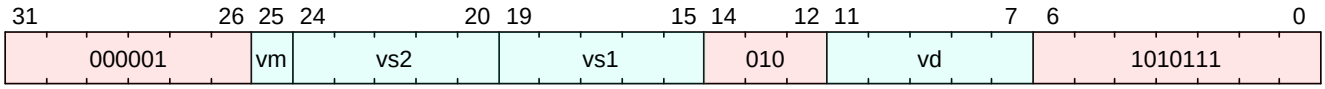
B.663. vredand.vs

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.663.1. Encoding



B.663.2. Synopsis

No description available.

B.663.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.663.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.663.5. Execution

```

```

B.663.6. Exceptions

This instruction does not generate synchronous exceptions.

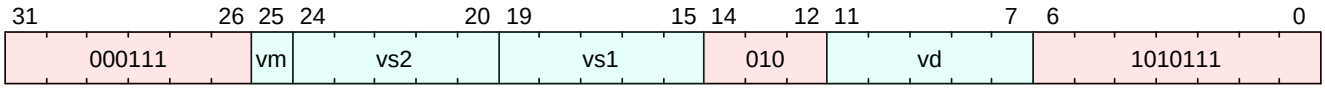
B.664. vredmax.vs

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.664.1. Encoding



B.664.2. Synopsis

No description available.

B.664.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.664.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.664.5. Execution

```

```

B.664.6. Exceptions

This instruction does not generate synchronous exceptions.

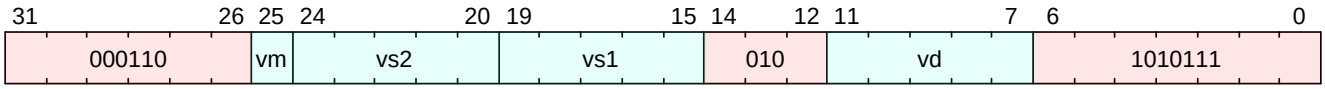
B.665. vredmaxu.vs

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.665.1. Encoding



B.665.2. Synopsis

No description available.

B.665.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.665.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.665.5. Execution

```

```

B.665.6. Exceptions

This instruction does not generate synchronous exceptions.

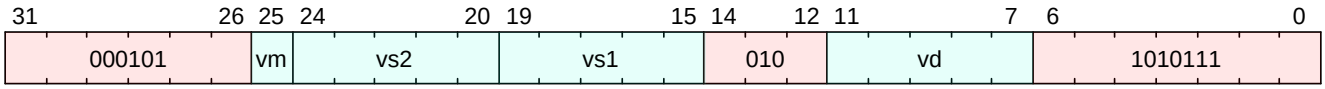
B.666. vredmin.vs

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.666.1. Encoding



B.666.2. Synopsis

No description available.

B.666.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.666.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.666.5. Execution

```

```

B.666.6. Exceptions

This instruction does not generate synchronous exceptions.

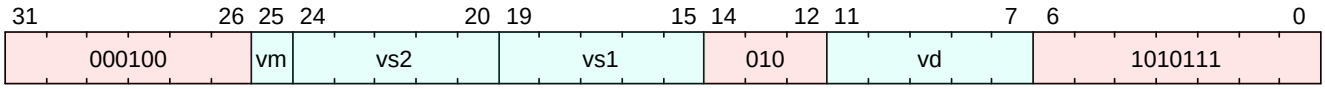
B.667. vredminu.vs

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.667.1. Encoding



B.667.2. Synopsis

No description available.

B.667.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.667.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.667.5. Execution

```

```

B.667.6. Exceptions

This instruction does not generate synchronous exceptions.

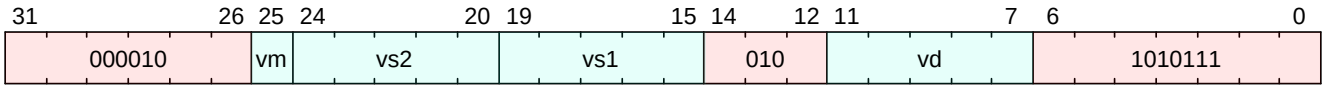
B.668. vredor.vs

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.668.1. Encoding



B.668.2. Synopsis

No description available.

B.668.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.668.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.668.5. Execution

```

```

B.668.6. Exceptions

This instruction does not generate synchronous exceptions.

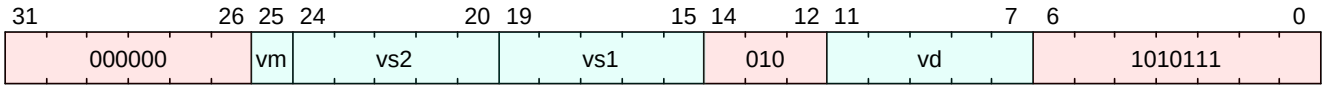
B.669. vredsum.vs

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.669.1. Encoding



B.669.2. Synopsis

No description available.

B.669.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.669.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.669.5. Execution

```

```

B.669.6. Exceptions

This instruction does not generate synchronous exceptions.

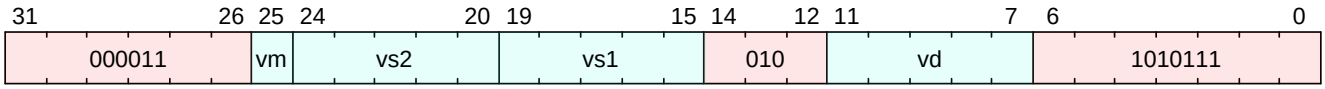
B.670. vredxor.vs

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.670.1. Encoding



B.670.2. Synopsis

No description available.

B.670.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.670.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.670.5. Execution

```

```

B.670.6. Exceptions

This instruction does not generate synchronous exceptions.

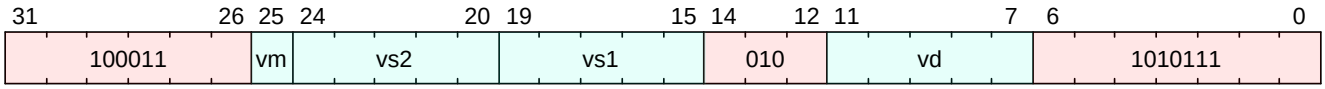
B.671. vrem.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.671.1. Encoding



B.671.2. Synopsis

No description available.

B.671.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.671.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.671.5. Execution

```

```

B.671.6. Exceptions

This instruction does not generate synchronous exceptions.

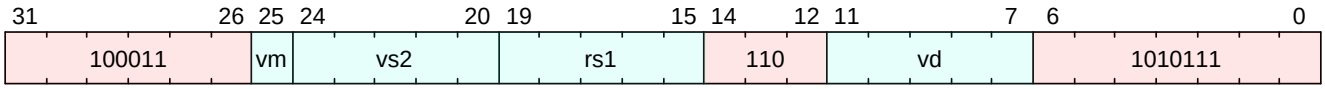
B.672. vrem.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.672.1. Encoding



B.672.2. Synopsis

No description available.

B.672.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.672.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.672.5. Execution

```

```

B.672.6. Exceptions

This instruction does not generate synchronous exceptions.

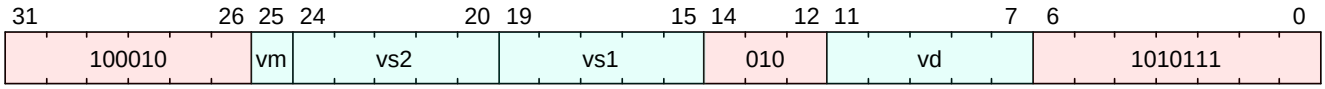
B.673. vremu.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.673.1. Encoding



B.673.2. Synopsis

No description available.

B.673.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.673.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.673.5. Execution

```

```

B.673.6. Exceptions

This instruction does not generate synchronous exceptions.

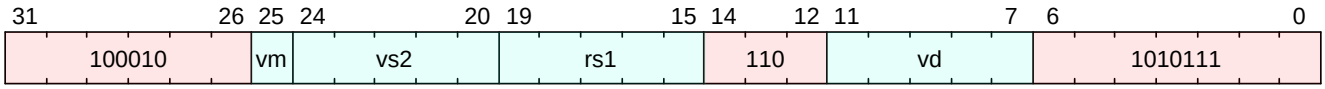
B.674. vremu.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.674.1. Encoding



B.674.2. Synopsis

No description available.

B.674.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.674.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.674.5. Execution

```

```

B.674.6. Exceptions

This instruction does not generate synchronous exceptions.

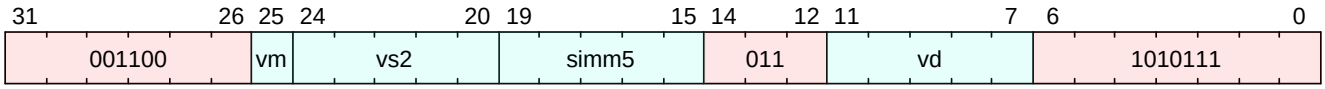
B.675. vrgather.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.675.1. Encoding



B.675.2. Synopsis

No description available.

B.675.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.675.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.675.5. Execution

```

```

B.675.6. Exceptions

This instruction does not generate synchronous exceptions.

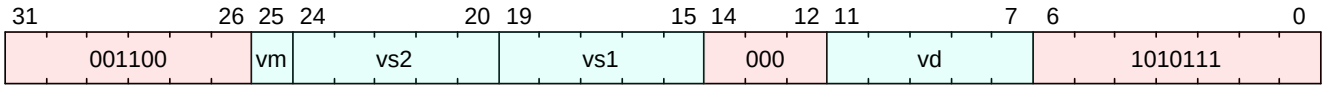
B.676. vrgather.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.676.1. Encoding



B.676.2. Synopsis

No description available.

B.676.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.676.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.676.5. Execution

```

```

B.676.6. Exceptions

This instruction does not generate synchronous exceptions.

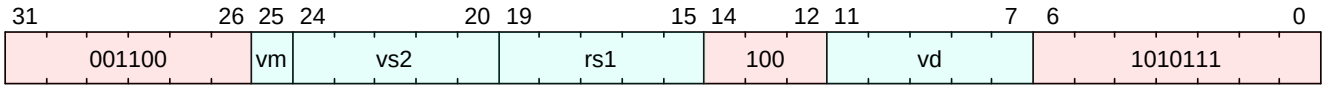
B.677. vrgather.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.677.1. Encoding



B.677.2. Synopsis

No description available.

B.677.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.677.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.677.5. Execution

```

```

B.677.6. Exceptions

This instruction does not generate synchronous exceptions.

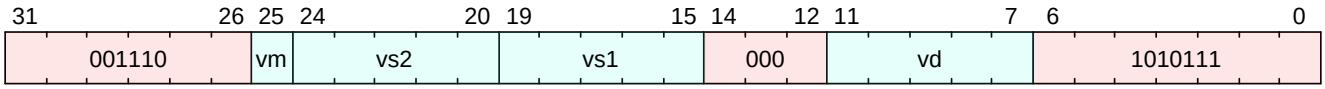
B.678. vrgatherei16.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.678.1. Encoding



B.678.2. Synopsis

No description available.

B.678.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.678.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.678.5. Execution

```

```

B.678.6. Exceptions

This instruction does not generate synchronous exceptions.

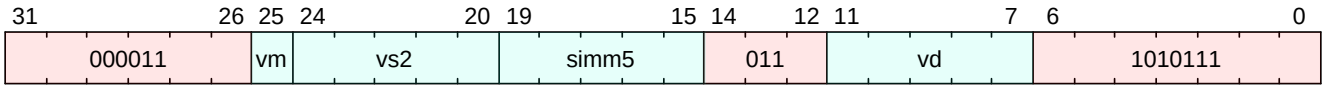
B.679. vrsub.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.679.1. Encoding



B.679.2. Synopsis

No description available.

B.679.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.679.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.679.5. Execution

```

```

B.679.6. Exceptions

This instruction does not generate synchronous exceptions.

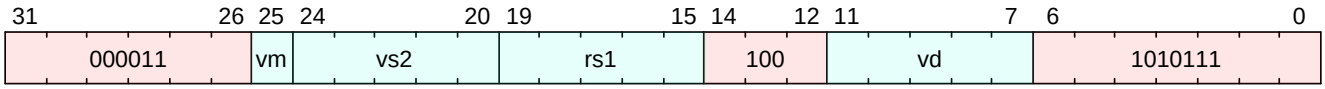
B.680. vrsub.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.680.1. Encoding



B.680.2. Synopsis

No description available.

B.680.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.680.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.680.5. Execution

```

```

B.680.6. Exceptions

This instruction does not generate synchronous exceptions.

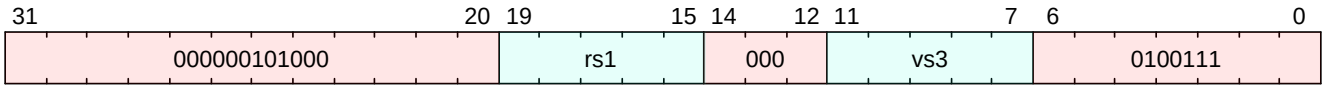
B.681. vs1r.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.681.1. Encoding



B.681.2. Synopsis

No description available.

B.681.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.681.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.681.5. Execution

```

```

B.681.6. Exceptions

This instruction does not generate synchronous exceptions.

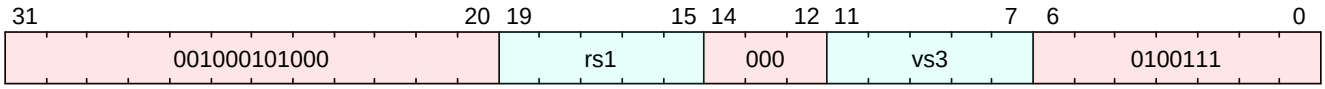
B.682. vs2r.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.682.1. Encoding



B.682.2. Synopsis

No description available.

B.682.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.682.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.682.5. Execution

```

```

B.682.6. Exceptions

This instruction does not generate synchronous exceptions.

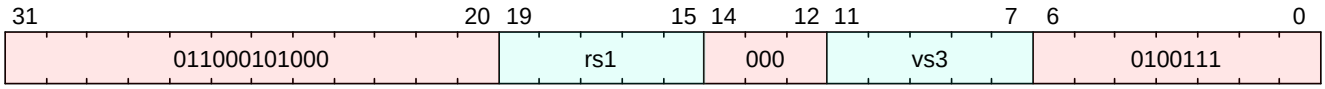
B.683. vs4r.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.683.1. Encoding



B.683.2. Synopsis

No description available.

B.683.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.683.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.683.5. Execution

```

```

B.683.6. Exceptions

This instruction does not generate synchronous exceptions.

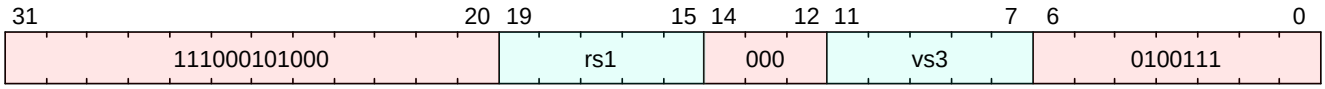
B.684. vs8r.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.684.1. Encoding



B.684.2. Synopsis

No description available.

B.684.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.684.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.684.5. Execution

```

```

B.684.6. Exceptions

This instruction does not generate synchronous exceptions.

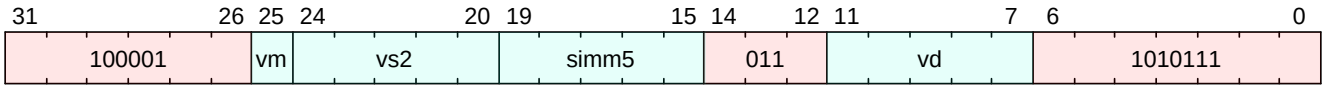
B.685. vsadd.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.685.1. Encoding



B.685.2. Synopsis

No description available.

B.685.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.685.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.685.5. Execution

```

```

B.685.6. Exceptions

This instruction does not generate synchronous exceptions.

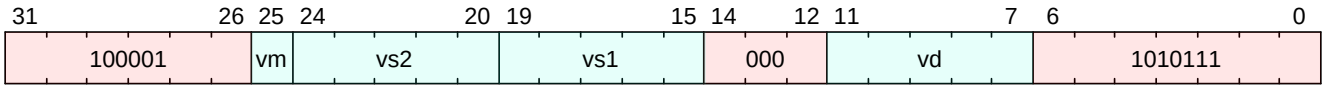
B.686. vsadd.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.686.1. Encoding



B.686.2. Synopsis

No description available.

B.686.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.686.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.686.5. Execution

```

```

B.686.6. Exceptions

This instruction does not generate synchronous exceptions.

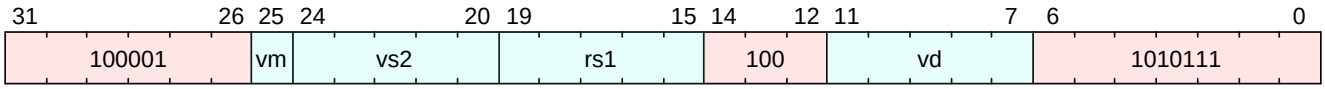
B.687. vsadd.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.687.1. Encoding



B.687.2. Synopsis

No description available.

B.687.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.687.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.687.5. Execution

```

```

B.687.6. Exceptions

This instruction does not generate synchronous exceptions.

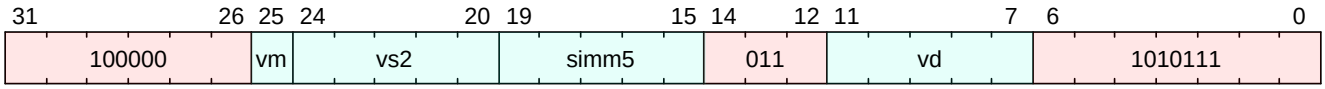
B.688. vsaddu.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.688.1. Encoding



B.688.2. Synopsis

No description available.

B.688.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.688.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.688.5. Execution

```

```

B.688.6. Exceptions

This instruction does not generate synchronous exceptions.

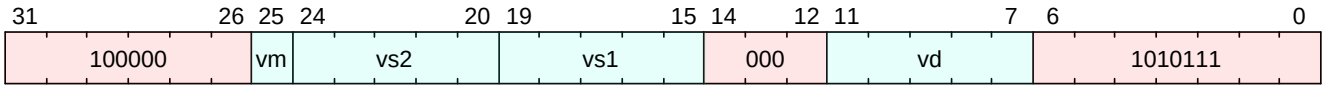
B.689. vsaddu.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.689.1. Encoding



B.689.2. Synopsis

No description available.

B.689.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.689.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.689.5. Execution

```

```

B.689.6. Exceptions

This instruction does not generate synchronous exceptions.

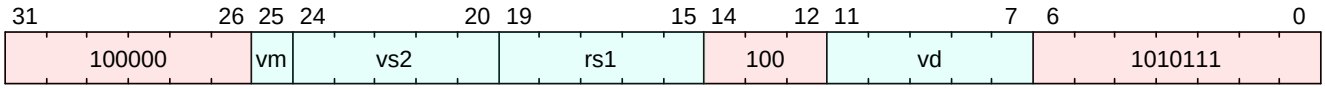
B.690. vsaddu.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.690.1. Encoding



B.690.2. Synopsis

No description available.

B.690.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.690.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.690.5. Execution

```

```

B.690.6. Exceptions

This instruction does not generate synchronous exceptions.

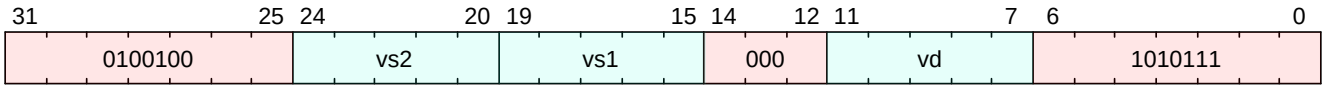
B.691. vsbc.vvm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.691.1. Encoding



B.691.2. Synopsis

No description available.

B.691.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.691.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.691.5. Execution

```

```

B.691.6. Exceptions

This instruction does not generate synchronous exceptions.

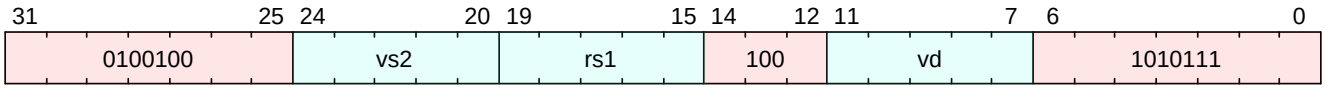
B.692. vsbc.vxm

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.692.1. Encoding



B.692.2. Synopsis

No description available.

B.692.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.692.4. Decode Variables

```
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.692.5. Execution

```

```

B.692.6. Exceptions

This instruction does not generate synchronous exceptions.

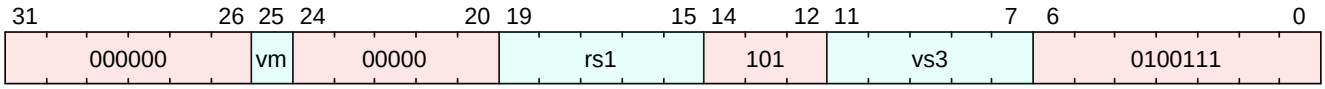
B.693. vse16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.693.1. Encoding



B.693.2. Synopsis

No description available.

B.693.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.693.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.693.5. Execution

```

```

B.693.6. Exceptions

This instruction does not generate synchronous exceptions.

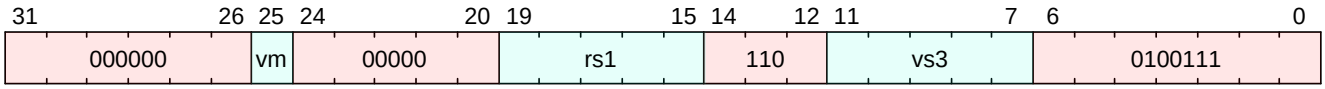
B.694. vse32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.694.1. Encoding



B.694.2. Synopsis

No description available.

B.694.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.694.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.694.5. Execution

```

```

B.694.6. Exceptions

This instruction does not generate synchronous exceptions.

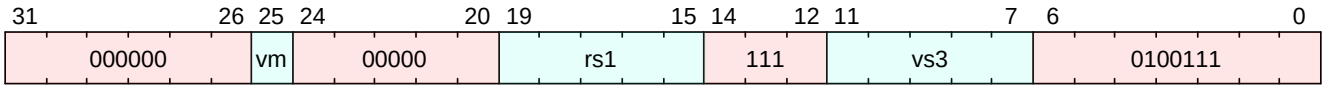
B.695. vse64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.695.1. Encoding



B.695.2. Synopsis

No description available.

B.695.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.695.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.695.5. Execution

```

```

B.695.6. Exceptions

This instruction does not generate synchronous exceptions.

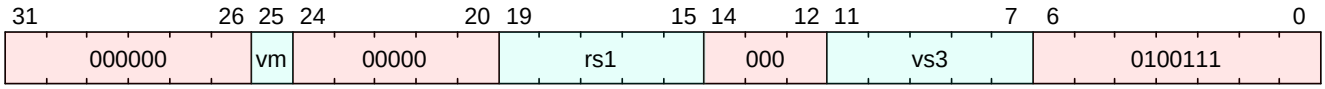
B.696. vse8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.696.1. Encoding



B.696.2. Synopsis

No description available.

B.696.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.696.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.696.5. Execution

```

```

B.696.6. Exceptions

This instruction does not generate synchronous exceptions.

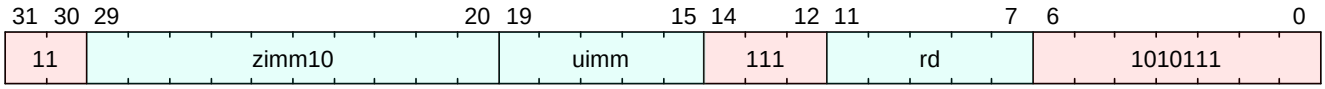
B.697. vsetivli

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.697.1. Encoding



B.697.2. Synopsis

No description available.

B.697.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.697.4. Decode Variables

```
Bits<10> zimm10 = $encoding[29:20];
Bits<5> uimm = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.697.5. Execution

```

```

B.697.6. Exceptions

This instruction does not generate synchronous exceptions.

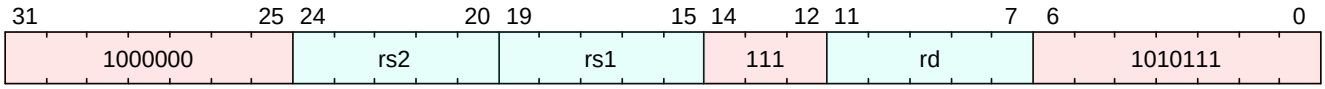
B.698. vsetvl

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.698.1. Encoding



B.698.2. Synopsis

No description available.

B.698.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.698.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.698.5. Execution

```

```

B.698.6. Exceptions

This instruction does not generate synchronous exceptions.

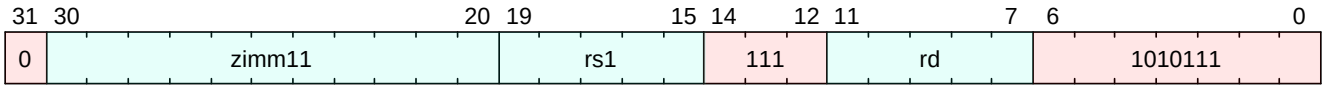
B.699. vsetvli

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.699.1. Encoding



B.699.2. Synopsis

No description available.

B.699.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.699.4. Decode Variables

```
Bits<11> zimm11 = $encoding[30:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.699.5. Execution

```

```

B.699.6. Exceptions

This instruction does not generate synchronous exceptions.

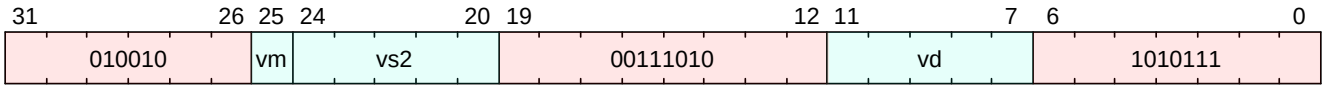
B.700. vsext.vf2

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.700.1. Encoding



B.700.2. Synopsis

No description available.

B.700.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.700.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.700.5. Execution

```

```

B.700.6. Exceptions

This instruction does not generate synchronous exceptions.

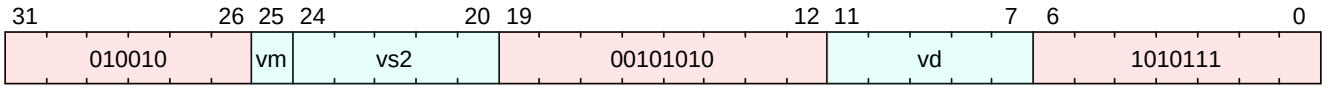
B.701. vsext.vf4

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.701.1. Encoding



B.701.2. Synopsis

No description available.

B.701.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.701.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.701.5. Execution

```

```

B.701.6. Exceptions

This instruction does not generate synchronous exceptions.

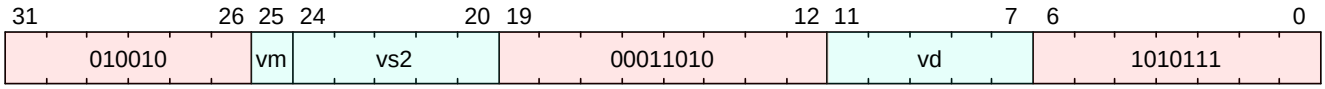
B.702. vsext.vf8

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.702.1. Encoding



B.702.2. Synopsis

No description available.

B.702.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.702.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.702.5. Execution

```

```

B.702.6. Exceptions

This instruction does not generate synchronous exceptions.

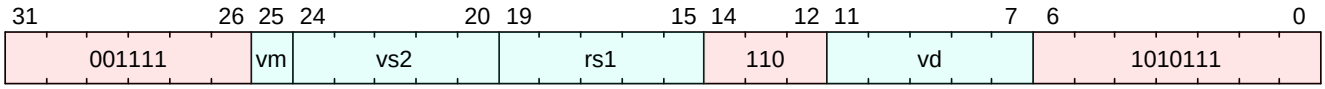
B.703. vslide1down.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.703.1. Encoding



B.703.2. Synopsis

No description available.

B.703.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.703.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.703.5. Execution

```

```

B.703.6. Exceptions

This instruction does not generate synchronous exceptions.

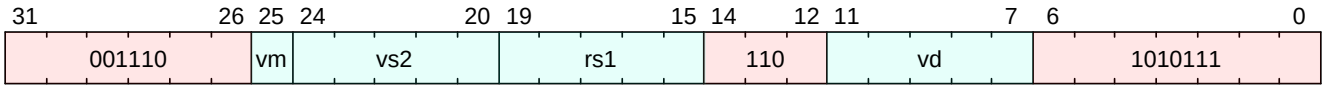
B.704. vslide1up.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.704.1. Encoding



B.704.2. Synopsis

No description available.

B.704.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.704.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.704.5. Execution

```

```

B.704.6. Exceptions

This instruction does not generate synchronous exceptions.

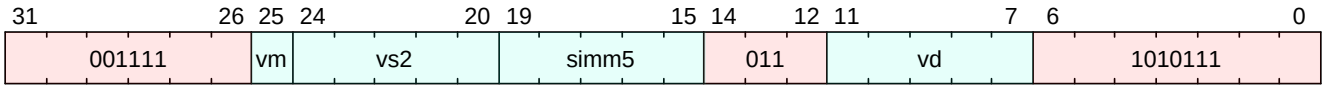
B.705. vslidedown.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.705.1. Encoding



B.705.2. Synopsis

No description available.

B.705.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.705.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.705.5. Execution

```

```

B.705.6. Exceptions

This instruction does not generate synchronous exceptions.

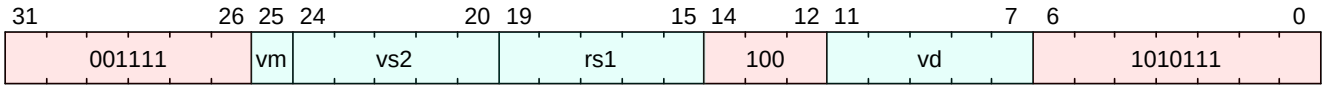
B.706. vslidedown.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.706.1. Encoding



B.706.2. Synopsis

No description available.

B.706.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.706.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.706.5. Execution

```

```

B.706.6. Exceptions

This instruction does not generate synchronous exceptions.

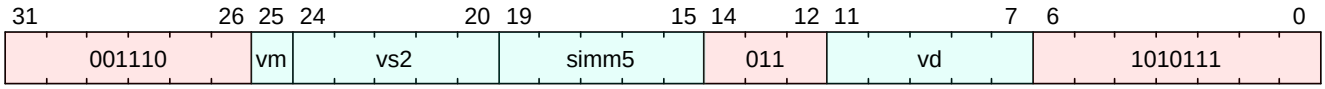
B.707. vslideup.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.707.1. Encoding



B.707.2. Synopsis

No description available.

B.707.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.707.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.707.5. Execution

```

```

B.707.6. Exceptions

This instruction does not generate synchronous exceptions.

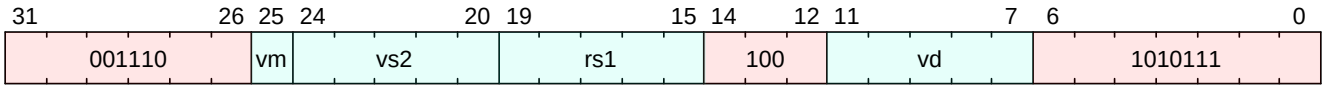
B.708. vslideup.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.708.1. Encoding



B.708.2. Synopsis

No description available.

B.708.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.708.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.708.5. Execution

```

```

B.708.6. Exceptions

This instruction does not generate synchronous exceptions.

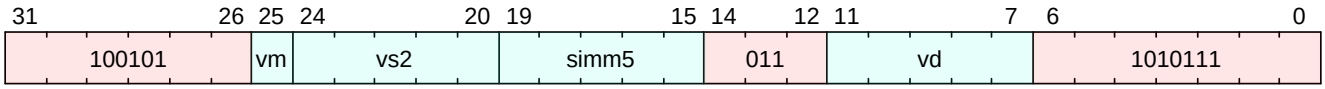
B.709. vsll.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.709.1. Encoding



B.709.2. Synopsis

No description available.

B.709.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.709.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.709.5. Execution

```

```

B.709.6. Exceptions

This instruction does not generate synchronous exceptions.

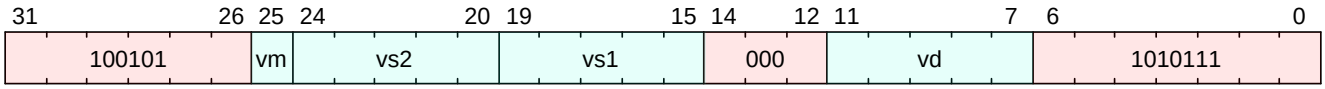
B.710. vsll.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.710.1. Encoding



B.710.2. Synopsis

No description available.

B.710.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.710.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.710.5. Execution

```

```

B.710.6. Exceptions

This instruction does not generate synchronous exceptions.

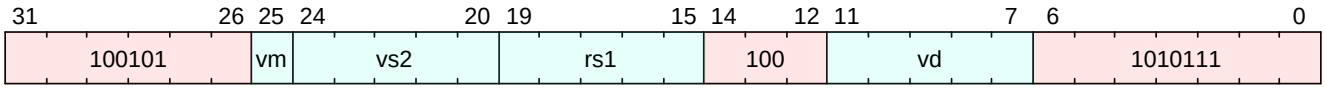
B.711. vsll.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.711.1. Encoding



B.711.2. Synopsis

No description available.

B.711.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.711.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.711.5. Execution

```

```

B.711.6. Exceptions

This instruction does not generate synchronous exceptions.

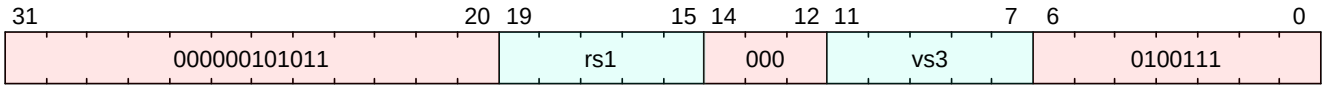
B.712. vsm.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.712.1. Encoding



B.712.2. Synopsis

No description available.

B.712.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.712.4. Decode Variables

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.712.5. Execution

```

```

B.712.6. Exceptions

This instruction does not generate synchronous exceptions.

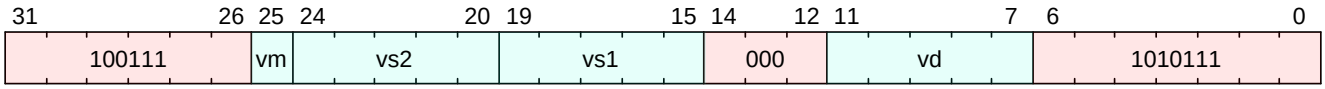
B.713. vsmul.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.713.1. Encoding



B.713.2. Synopsis

No description available.

B.713.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.713.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.713.5. Execution

```

```

B.713.6. Exceptions

This instruction does not generate synchronous exceptions.

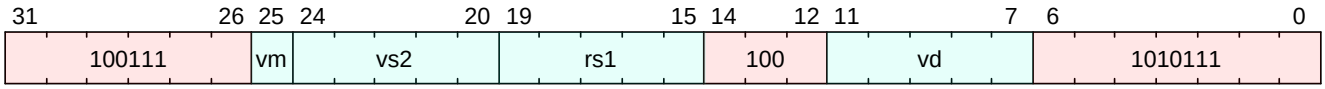
B.714. vsmul.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.714.1. Encoding



B.714.2. Synopsis

No description available.

B.714.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.714.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.714.5. Execution

```

```

B.714.6. Exceptions

This instruction does not generate synchronous exceptions.

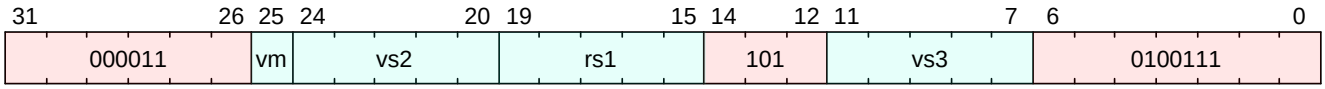
B.715. vsoxei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.715.1. Encoding



B.715.2. Synopsis

No description available.

B.715.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.715.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.715.5. Execution

```

```

B.715.6. Exceptions

This instruction does not generate synchronous exceptions.

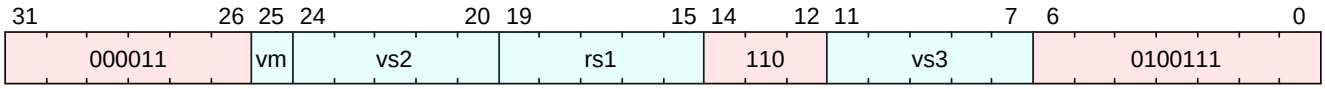
B.716. vsoxei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.716.1. Encoding



B.716.2. Synopsis

No description available.

B.716.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.716.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.716.5. Execution

```

```

B.716.6. Exceptions

This instruction does not generate synchronous exceptions.

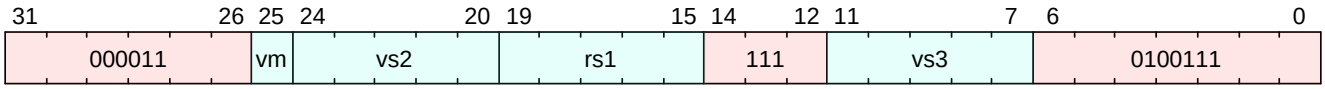
B.717. vsoxei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.717.1. Encoding



B.717.2. Synopsis

No description available.

B.717.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.717.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.717.5. Execution

```

```

B.717.6. Exceptions

This instruction does not generate synchronous exceptions.

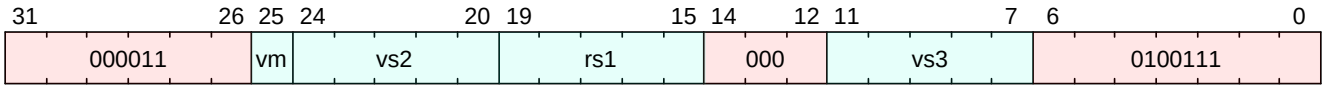
B.718. vsoxei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.718.1. Encoding



B.718.2. Synopsis

No description available.

B.718.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.718.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.718.5. Execution

```

```

B.718.6. Exceptions

This instruction does not generate synchronous exceptions.

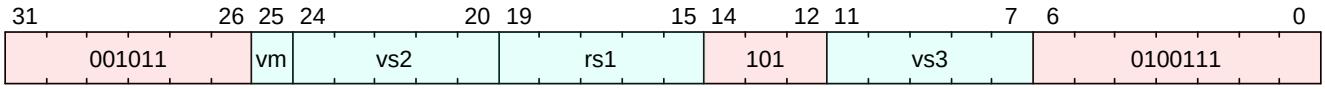
B.719. vsoxseg2ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.719.1. Encoding



B.719.2. Synopsis

No description available.

B.719.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.719.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.719.5. Execution

```

```

B.719.6. Exceptions

This instruction does not generate synchronous exceptions.

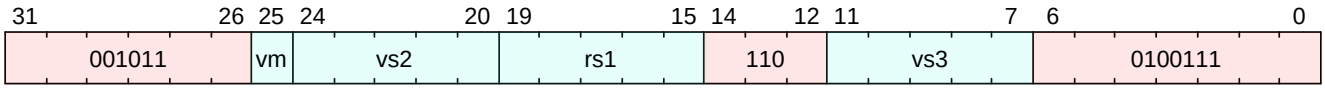
B.720. vsoxseg2ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.720.1. Encoding



B.720.2. Synopsis

No description available.

B.720.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.720.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.720.5. Execution

```

```

B.720.6. Exceptions

This instruction does not generate synchronous exceptions.

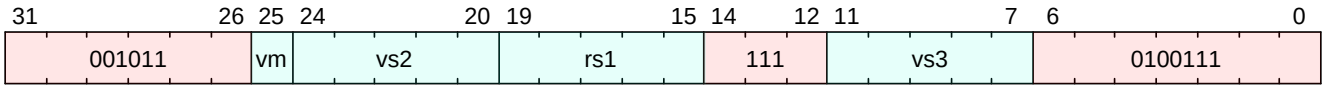
B.721. vsoxseg2ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.721.1. Encoding



B.721.2. Synopsis

No description available.

B.721.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.721.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.721.5. Execution

```

```

B.721.6. Exceptions

This instruction does not generate synchronous exceptions.

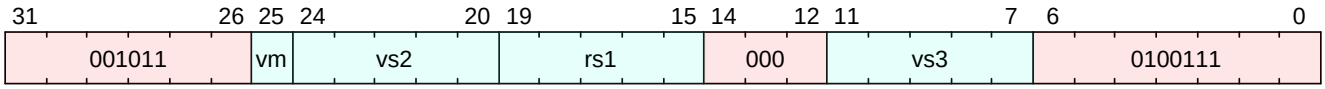
B.722. vsoxseg2ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.722.1. Encoding



B.722.2. Synopsis

No description available.

B.722.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.722.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.722.5. Execution

```

```

B.722.6. Exceptions

This instruction does not generate synchronous exceptions.

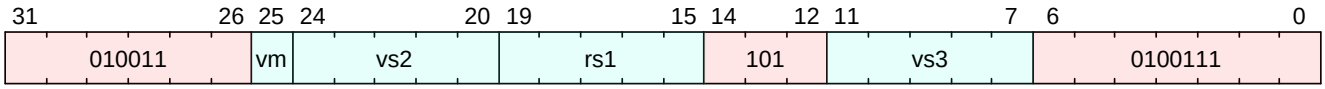
B.723. vsoxseg3ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.723.1. Encoding



B.723.2. Synopsis

No description available.

B.723.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.723.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.723.5. Execution

```

```

B.723.6. Exceptions

This instruction does not generate synchronous exceptions.

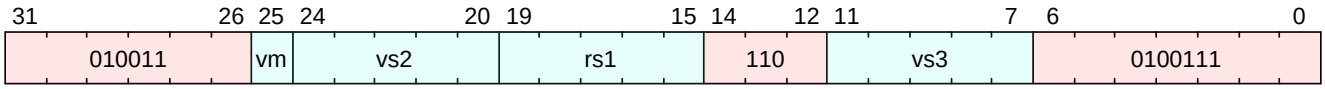
B.724. vsoxseg3ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.724.1. Encoding



B.724.2. Synopsis

No description available.

B.724.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.724.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.724.5. Execution

```

```

B.724.6. Exceptions

This instruction does not generate synchronous exceptions.

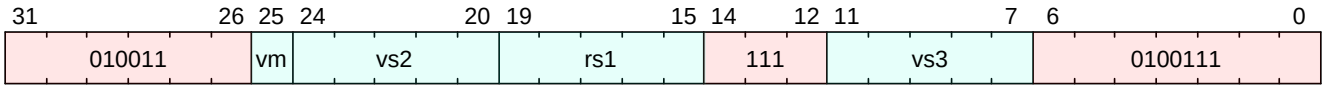
B.725. vsoxseg3ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.725.1. Encoding



B.725.2. Synopsis

No description available.

B.725.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.725.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.725.5. Execution

```

```

B.725.6. Exceptions

This instruction does not generate synchronous exceptions.

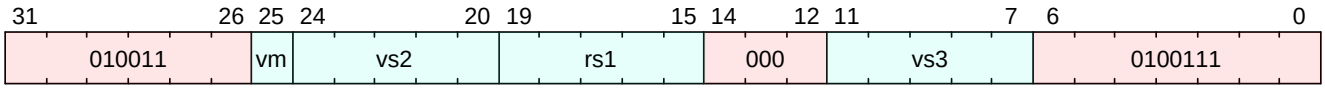
B.726. vsoxseg3ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.726.1. Encoding



B.726.2. Synopsis

No description available.

B.726.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.726.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.726.5. Execution

```

```

B.726.6. Exceptions

This instruction does not generate synchronous exceptions.

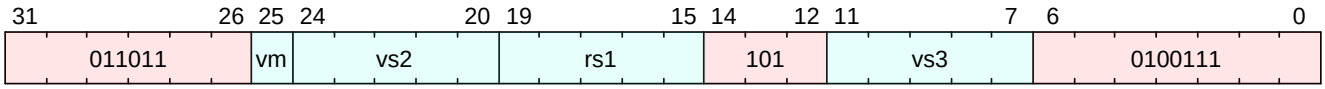
B.727. vsoxseg4ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.727.1. Encoding



B.727.2. Synopsis

No description available.

B.727.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.727.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.727.5. Execution

```

```

B.727.6. Exceptions

This instruction does not generate synchronous exceptions.

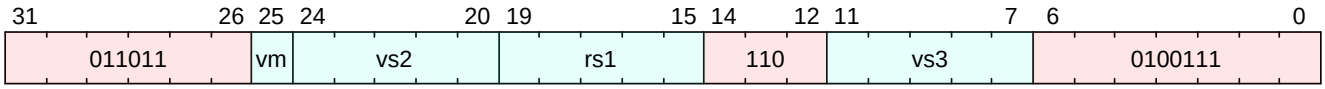
B.728. vsoxseg4ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.728.1. Encoding



B.728.2. Synopsis

No description available.

B.728.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.728.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.728.5. Execution

```

```

B.728.6. Exceptions

This instruction does not generate synchronous exceptions.

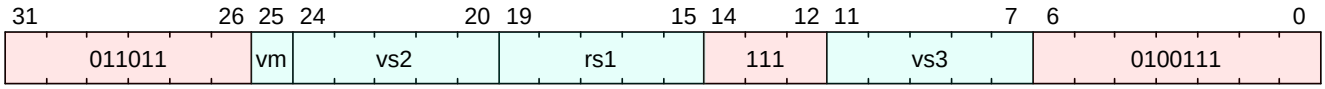
B.729. vsoxseg4ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.729.1. Encoding



B.729.2. Synopsis

No description available.

B.729.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.729.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.729.5. Execution

```

```

B.729.6. Exceptions

This instruction does not generate synchronous exceptions.

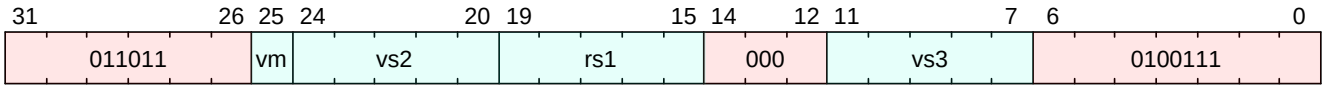
B.730. vsoxseg4ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.730.1. Encoding



B.730.2. Synopsis

No description available.

B.730.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.730.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.730.5. Execution

```

```

B.730.6. Exceptions

This instruction does not generate synchronous exceptions.

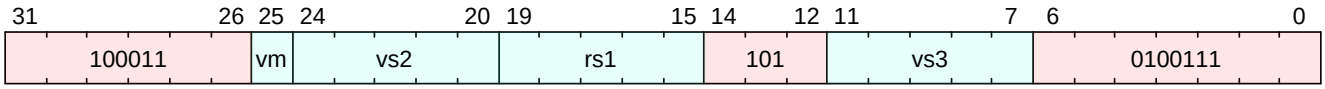
B.731. vsoxseg5ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.731.1. Encoding



B.731.2. Synopsis

No description available.

B.731.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.731.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.731.5. Execution

```

```

B.731.6. Exceptions

This instruction does not generate synchronous exceptions.

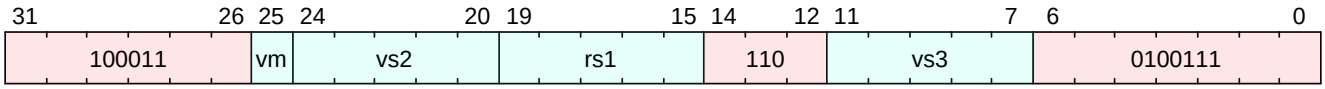
B.732. vsoxseg5ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.732.1. Encoding



B.732.2. Synopsis

No description available.

B.732.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.732.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.732.5. Execution

```

```

B.732.6. Exceptions

This instruction does not generate synchronous exceptions.

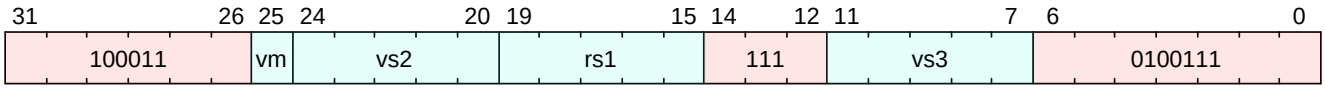
B.733. vsoxseg5ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.733.1. Encoding



B.733.2. Synopsis

No description available.

B.733.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.733.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.733.5. Execution

```

```

B.733.6. Exceptions

This instruction does not generate synchronous exceptions.

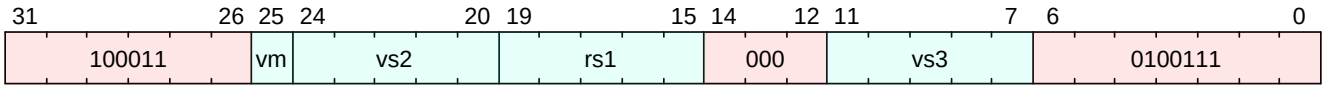
B.734. vsoxseg5ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.734.1. Encoding



B.734.2. Synopsis

No description available.

B.734.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.734.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.734.5. Execution

```

```

B.734.6. Exceptions

This instruction does not generate synchronous exceptions.

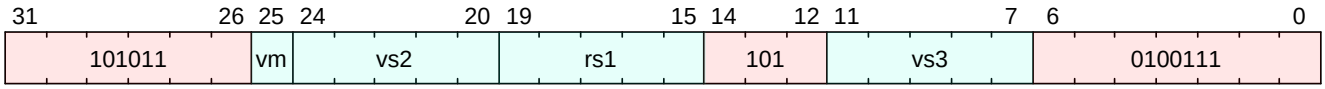
B.735. vsoxseg6ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.735.1. Encoding



B.735.2. Synopsis

No description available.

B.735.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.735.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.735.5. Execution

```

```

B.735.6. Exceptions

This instruction does not generate synchronous exceptions.

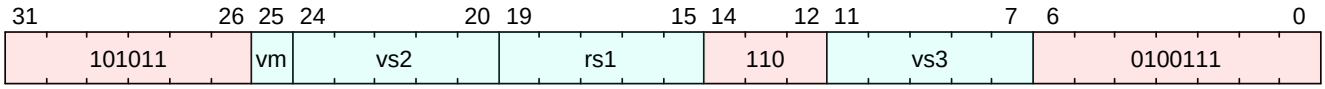
B.736. vsoxseg6ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.736.1. Encoding



B.736.2. Synopsis

No description available.

B.736.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.736.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.736.5. Execution

```

```

B.736.6. Exceptions

This instruction does not generate synchronous exceptions.

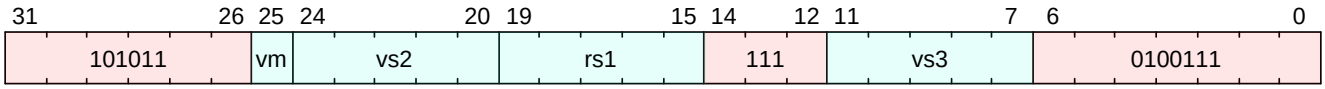
B.737. vsoxseg6ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.737.1. Encoding



B.737.2. Synopsis

No description available.

B.737.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.737.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.737.5. Execution

```

```

B.737.6. Exceptions

This instruction does not generate synchronous exceptions.

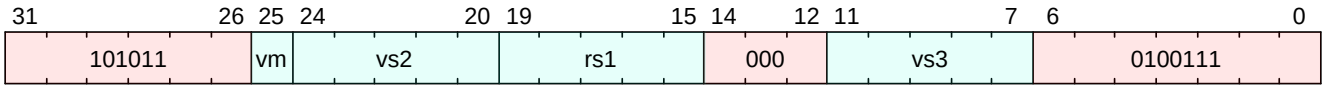
B.738. vsoxseg6ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.738.1. Encoding



B.738.2. Synopsis

No description available.

B.738.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.738.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.738.5. Execution

```

```

B.738.6. Exceptions

This instruction does not generate synchronous exceptions.

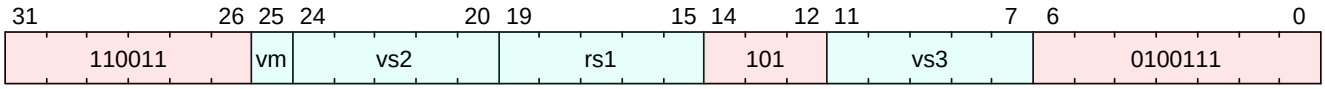
B.739. vsoxseg7ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.739.1. Encoding



B.739.2. Synopsis

No description available.

B.739.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.739.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.739.5. Execution

```

```

B.739.6. Exceptions

This instruction does not generate synchronous exceptions.

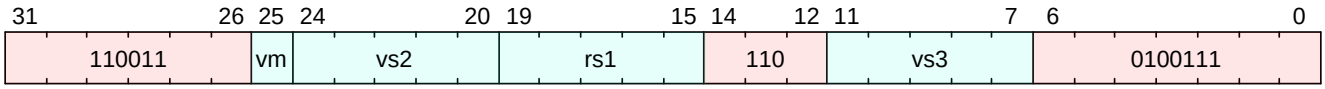
B.740. vsoxseg7ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.740.1. Encoding



B.740.2. Synopsis

No description available.

B.740.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.740.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.740.5. Execution

```

```

B.740.6. Exceptions

This instruction does not generate synchronous exceptions.

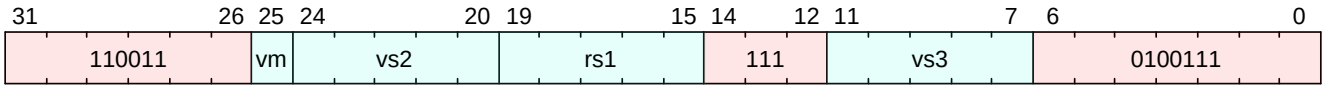
B.741. vsoxseg7ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.741.1. Encoding



B.741.2. Synopsis

No description available.

B.741.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.741.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.741.5. Execution

```

```

B.741.6. Exceptions

This instruction does not generate synchronous exceptions.

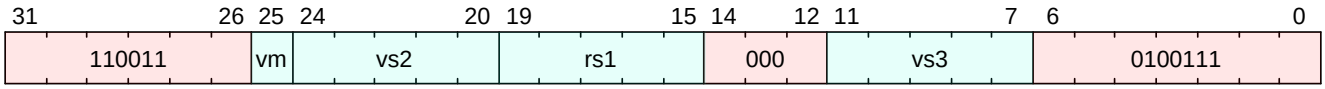
B.742. vsoxseg7ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.742.1. Encoding



B.742.2. Synopsis

No description available.

B.742.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.742.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.742.5. Execution

```

```

B.742.6. Exceptions

This instruction does not generate synchronous exceptions.

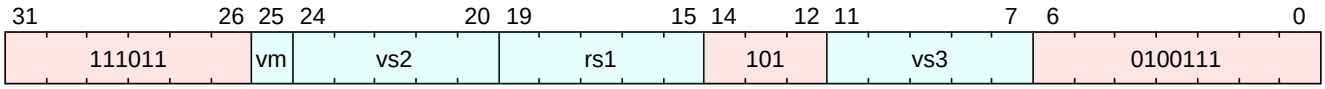
B.743. vsoxseg8ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.743.1. Encoding



B.743.2. Synopsis

No description available.

B.743.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.743.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.743.5. Execution

```

```

B.743.6. Exceptions

This instruction does not generate synchronous exceptions.

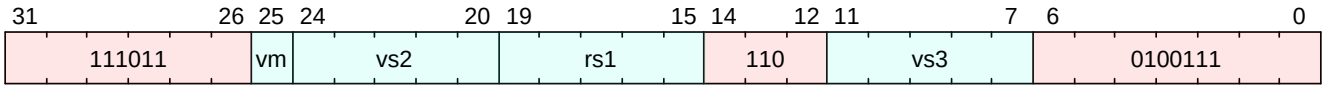
B.744. vsoxseg8ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.744.1. Encoding



B.744.2. Synopsis

No description available.

B.744.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.744.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.744.5. Execution

```

```

B.744.6. Exceptions

This instruction does not generate synchronous exceptions.

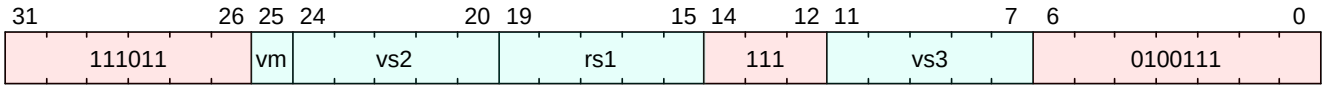
B.745. vsoxseg8ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.745.1. Encoding



B.745.2. Synopsis

No description available.

B.745.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.745.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.745.5. Execution

```

```

B.745.6. Exceptions

This instruction does not generate synchronous exceptions.

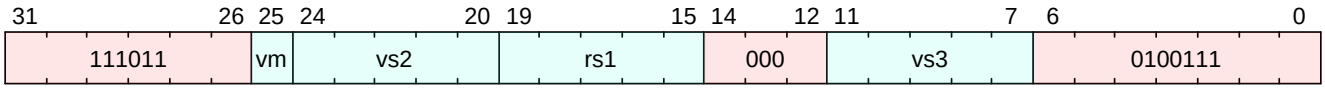
B.746. vsoxseg8ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.746.1. Encoding



B.746.2. Synopsis

No description available.

B.746.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.746.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.746.5. Execution

```

```

B.746.6. Exceptions

This instruction does not generate synchronous exceptions.

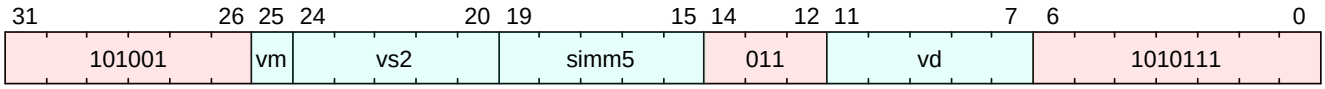
B.747. vsra.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.747.1. Encoding



B.747.2. Synopsis

No description available.

B.747.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.747.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.747.5. Execution

```

```

B.747.6. Exceptions

This instruction does not generate synchronous exceptions.

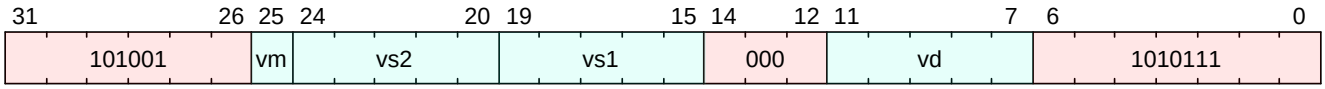
B.748. vsra.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.748.1. Encoding



B.748.2. Synopsis

No description available.

B.748.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.748.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.748.5. Execution

```

```

B.748.6. Exceptions

This instruction does not generate synchronous exceptions.

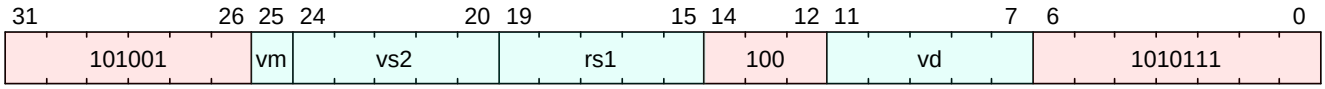
B.749. vsra.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.749.1. Encoding



B.749.2. Synopsis

No description available.

B.749.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.749.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.749.5. Execution

```

```

B.749.6. Exceptions

This instruction does not generate synchronous exceptions.

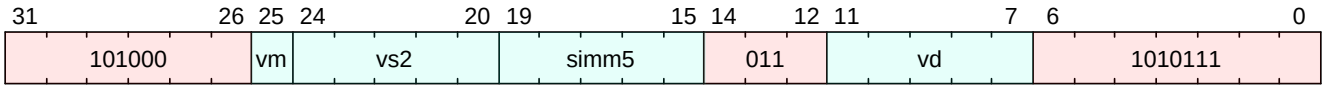
B.750. vsrl.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.750.1. Encoding



B.750.2. Synopsis

No description available.

B.750.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.750.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.750.5. Execution

```

```

B.750.6. Exceptions

This instruction does not generate synchronous exceptions.

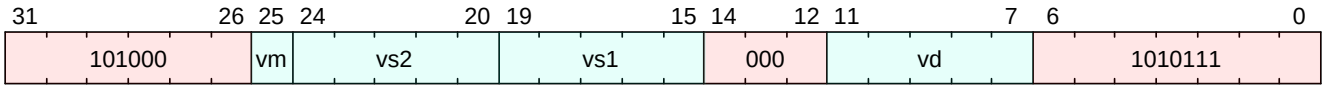
B.751. vsrl.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.751.1. Encoding



B.751.2. Synopsis

No description available.

B.751.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.751.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.751.5. Execution

```

```

B.751.6. Exceptions

This instruction does not generate synchronous exceptions.

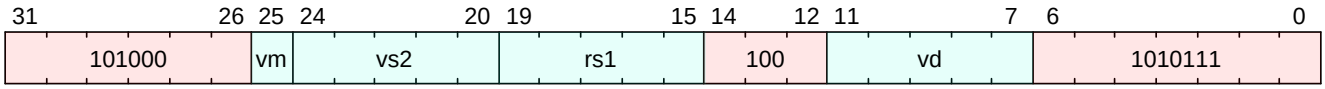
B.752. vsrl.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.752.1. Encoding



B.752.2. Synopsis

No description available.

B.752.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.752.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.752.5. Execution

```

```

B.752.6. Exceptions

This instruction does not generate synchronous exceptions.

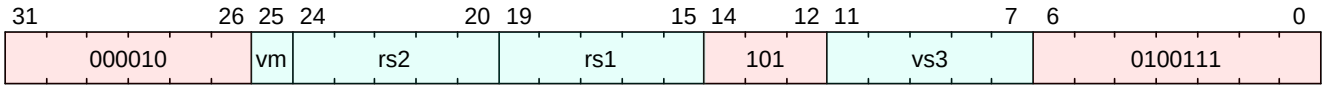
B.753. vsse16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.753.1. Encoding



B.753.2. Synopsis

No description available.

B.753.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.753.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.753.5. Execution

```

```

B.753.6. Exceptions

This instruction does not generate synchronous exceptions.

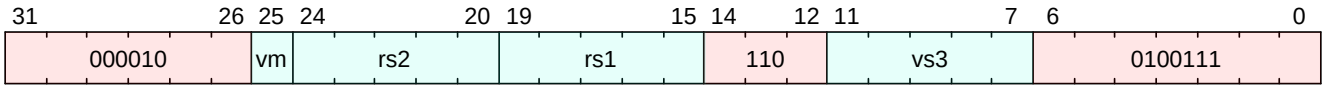
B.754. vsse32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.754.1. Encoding



B.754.2. Synopsis

No description available.

B.754.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.754.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.754.5. Execution

```

```

B.754.6. Exceptions

This instruction does not generate synchronous exceptions.

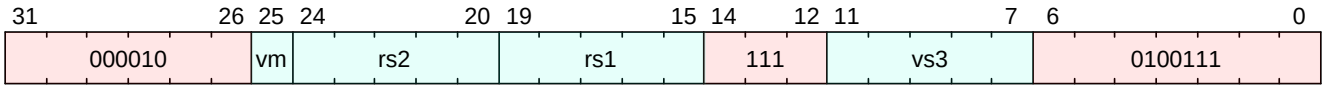
B.755. vsse64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.755.1. Encoding



B.755.2. Synopsis

No description available.

B.755.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.755.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.755.5. Execution

```

```

B.755.6. Exceptions

This instruction does not generate synchronous exceptions.

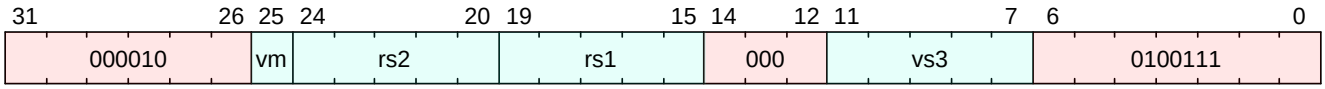
B.756. vsse8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.756.1. Encoding



B.756.2. Synopsis

No description available.

B.756.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.756.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.756.5. Execution

```

```

B.756.6. Exceptions

This instruction does not generate synchronous exceptions.

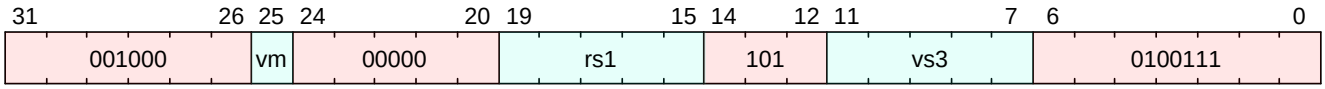
B.757. vsseg2e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.757.1. Encoding



B.757.2. Synopsis

No description available.

B.757.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.757.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.757.5. Execution

```

```

B.757.6. Exceptions

This instruction does not generate synchronous exceptions.

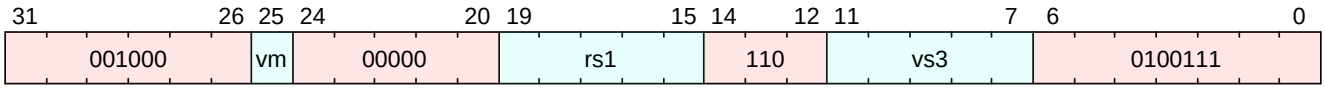
B.758. vsseg2e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.758.1. Encoding



B.758.2. Synopsis

No description available.

B.758.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.758.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.758.5. Execution

```

```

B.758.6. Exceptions

This instruction does not generate synchronous exceptions.

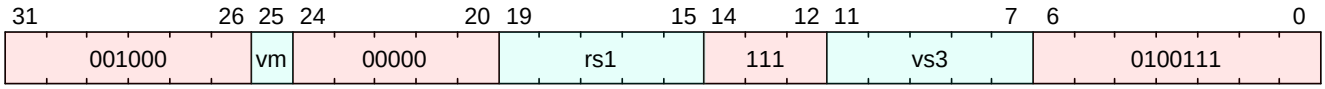
B.759. vsseg2e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.759.1. Encoding



B.759.2. Synopsis

No description available.

B.759.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.759.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.759.5. Execution

```

```

B.759.6. Exceptions

This instruction does not generate synchronous exceptions.

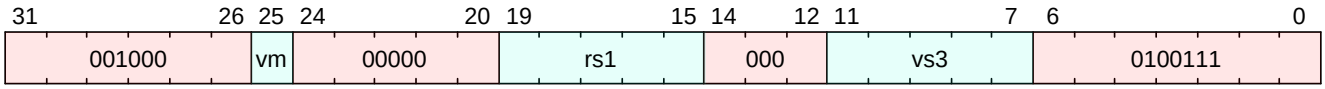
B.760. vsseg2e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.760.1. Encoding



B.760.2. Synopsis

No description available.

B.760.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.760.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.760.5. Execution

```

```

B.760.6. Exceptions

This instruction does not generate synchronous exceptions.

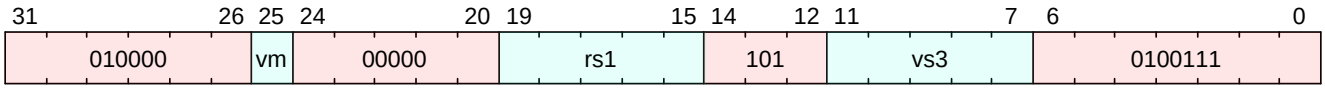
B.761. vsseg3e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.761.1. Encoding



B.761.2. Synopsis

No description available.

B.761.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.761.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.761.5. Execution

```

```

B.761.6. Exceptions

This instruction does not generate synchronous exceptions.

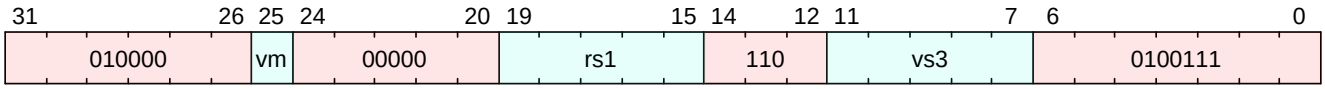
B.762. vsseg3e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.762.1. Encoding



B.762.2. Synopsis

No description available.

B.762.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.762.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.762.5. Execution

```

```

B.762.6. Exceptions

This instruction does not generate synchronous exceptions.

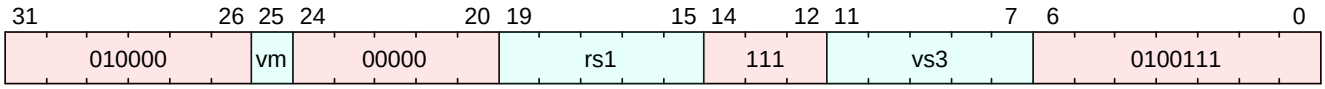
B.763. vsseg3e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.763.1. Encoding



B.763.2. Synopsis

No description available.

B.763.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.763.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.763.5. Execution

```

```

B.763.6. Exceptions

This instruction does not generate synchronous exceptions.

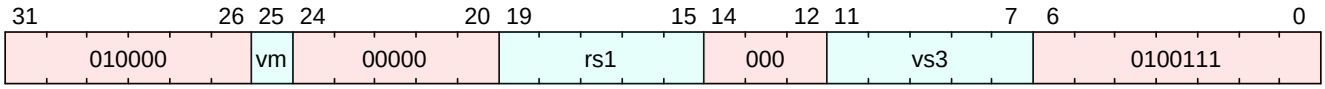
B.764. vsseg3e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.764.1. Encoding



B.764.2. Synopsis

No description available.

B.764.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.764.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.764.5. Execution

```

```

B.764.6. Exceptions

This instruction does not generate synchronous exceptions.

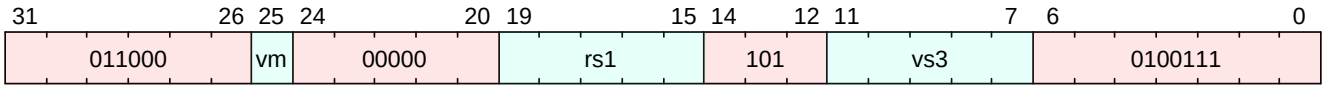
B.765. vsseg4e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.765.1. Encoding



B.765.2. Synopsis

No description available.

B.765.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.765.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.765.5. Execution

```

```

B.765.6. Exceptions

This instruction does not generate synchronous exceptions.

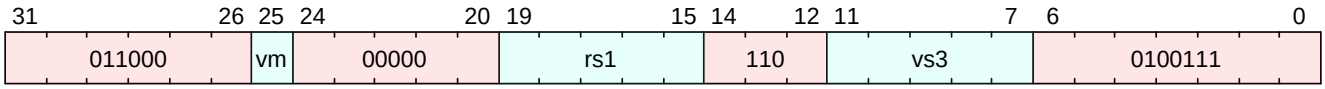
B.766. vsseg4e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.766.1. Encoding



B.766.2. Synopsis

No description available.

B.766.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.766.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.766.5. Execution

```

```

B.766.6. Exceptions

This instruction does not generate synchronous exceptions.

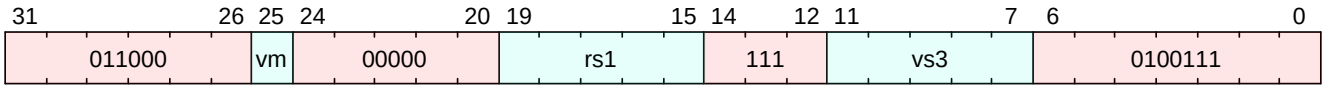
B.767. vsseg4e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.767.1. Encoding



B.767.2. Synopsis

No description available.

B.767.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.767.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.767.5. Execution

```

```

B.767.6. Exceptions

This instruction does not generate synchronous exceptions.

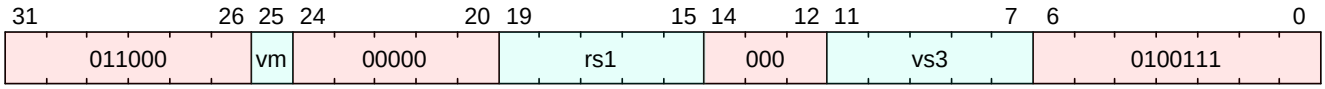
B.768. vsseg4e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.768.1. Encoding



B.768.2. Synopsis

No description available.

B.768.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.768.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.768.5. Execution

```

```

B.768.6. Exceptions

This instruction does not generate synchronous exceptions.

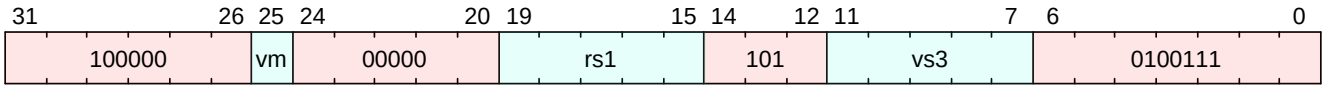
B.769. vsseg5e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.769.1. Encoding



B.769.2. Synopsis

No description available.

B.769.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.769.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.769.5. Execution

```

```

B.769.6. Exceptions

This instruction does not generate synchronous exceptions.

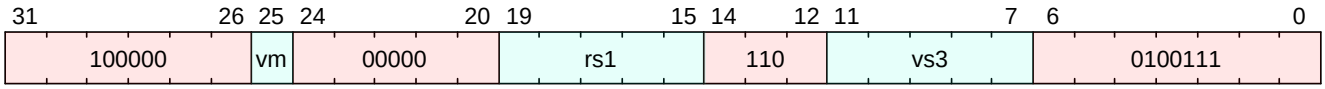
B.770. vsseg5e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.770.1. Encoding



B.770.2. Synopsis

No description available.

B.770.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.770.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.770.5. Execution

```

```

B.770.6. Exceptions

This instruction does not generate synchronous exceptions.

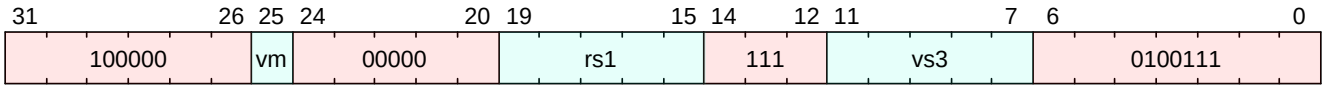
B.771. vsseg5e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.771.1. Encoding



B.771.2. Synopsis

No description available.

B.771.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.771.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.771.5. Execution

```

```

B.771.6. Exceptions

This instruction does not generate synchronous exceptions.

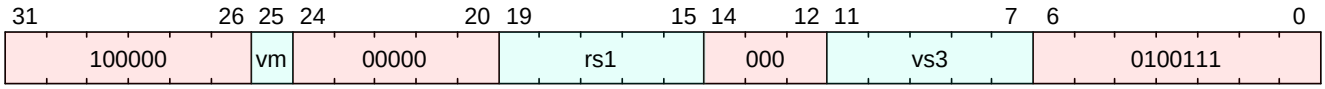
B.772. vsseg5e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.772.1. Encoding



B.772.2. Synopsis

No description available.

B.772.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.772.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.772.5. Execution

```

```

B.772.6. Exceptions

This instruction does not generate synchronous exceptions.

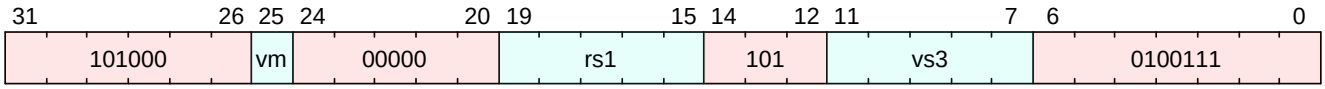
B.773. vsseg6e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.773.1. Encoding



B.773.2. Synopsis

No description available.

B.773.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.773.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.773.5. Execution

```

```

B.773.6. Exceptions

This instruction does not generate synchronous exceptions.

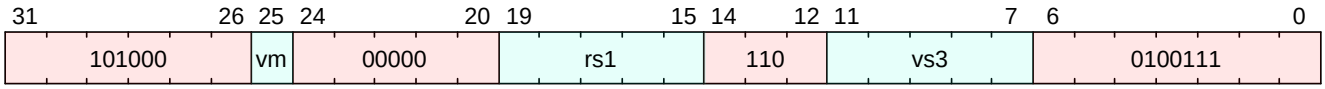
B.774. vsseg6e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.774.1. Encoding



B.774.2. Synopsis

No description available.

B.774.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.774.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.774.5. Execution

```

```

B.774.6. Exceptions

This instruction does not generate synchronous exceptions.

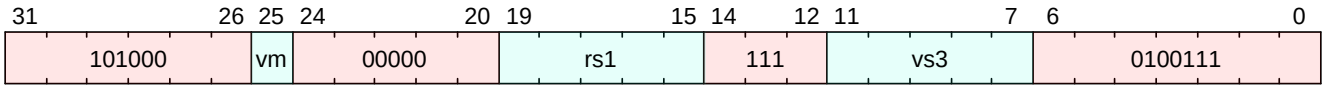
B.775. vsseg6e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.775.1. Encoding



B.775.2. Synopsis

No description available.

B.775.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.775.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.775.5. Execution

```

```

B.775.6. Exceptions

This instruction does not generate synchronous exceptions.

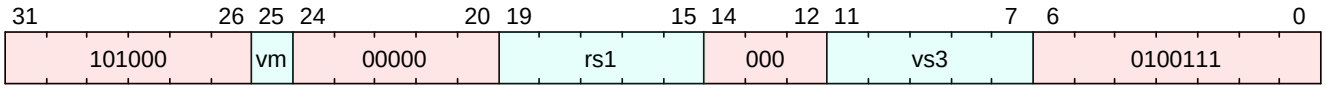
B.776. vsseg6e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.776.1. Encoding



B.776.2. Synopsis

No description available.

B.776.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.776.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.776.5. Execution

```

```

B.776.6. Exceptions

This instruction does not generate synchronous exceptions.

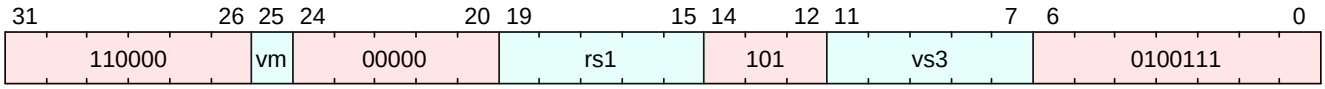
B.777. vsseg7e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.777.1. Encoding



B.777.2. Synopsis

No description available.

B.777.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.777.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.777.5. Execution

```

```

B.777.6. Exceptions

This instruction does not generate synchronous exceptions.

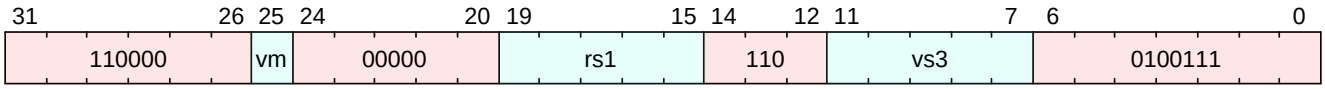
B.778. vsseg7e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.778.1. Encoding



B.778.2. Synopsis

No description available.

B.778.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.778.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.778.5. Execution

```

```

B.778.6. Exceptions

This instruction does not generate synchronous exceptions.

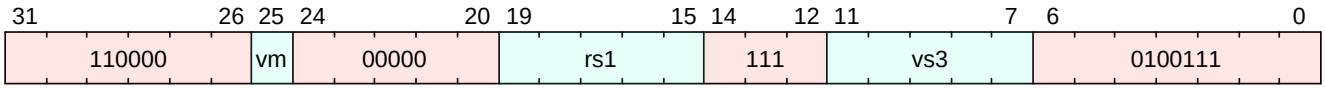
B.779. vsseg7e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.779.1. Encoding



B.779.2. Synopsis

No description available.

B.779.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.779.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.779.5. Execution

```

```

B.779.6. Exceptions

This instruction does not generate synchronous exceptions.

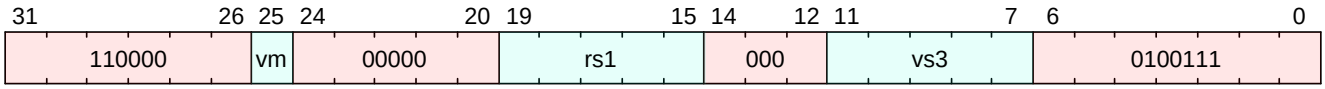
B.780. vsseg7e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.780.1. Encoding



B.780.2. Synopsis

No description available.

B.780.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.780.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.780.5. Execution

```

```

B.780.6. Exceptions

This instruction does not generate synchronous exceptions.

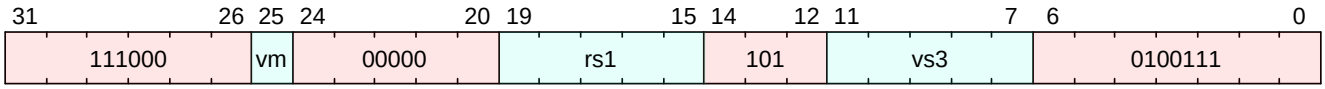
B.781. vsseg8e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.781.1. Encoding



B.781.2. Synopsis

No description available.

B.781.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.781.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.781.5. Execution

```

```

B.781.6. Exceptions

This instruction does not generate synchronous exceptions.

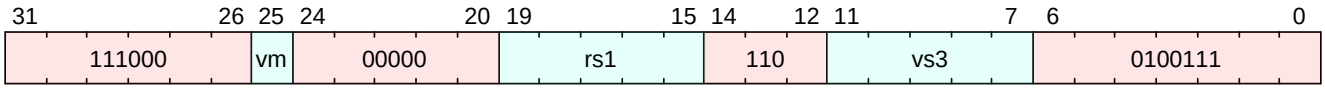
B.782. vsseg8e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.782.1. Encoding



B.782.2. Synopsis

No description available.

B.782.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.782.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.782.5. Execution

```

```

B.782.6. Exceptions

This instruction does not generate synchronous exceptions.

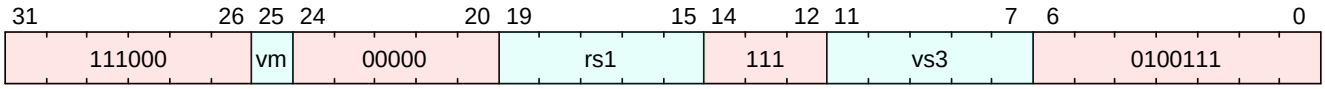
B.783. vsseg8e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.783.1. Encoding



B.783.2. Synopsis

No description available.

B.783.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.783.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.783.5. Execution

```

```

B.783.6. Exceptions

This instruction does not generate synchronous exceptions.

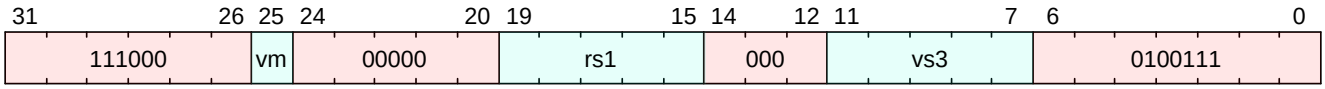
B.784. vsseg8e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.784.1. Encoding



B.784.2. Synopsis

No description available.

B.784.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.784.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.784.5. Execution

```

```

B.784.6. Exceptions

This instruction does not generate synchronous exceptions.

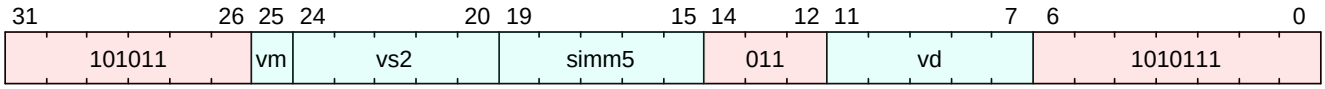
B.785. vssra.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.785.1. Encoding



B.785.2. Synopsis

No description available.

B.785.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.785.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.785.5. Execution

```

```

B.785.6. Exceptions

This instruction does not generate synchronous exceptions.

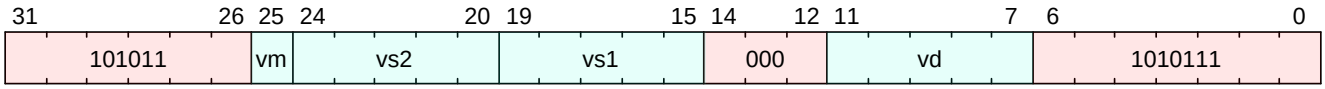
B.786. vssra.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.786.1. Encoding



B.786.2. Synopsis

No description available.

B.786.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.786.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.786.5. Execution

```

```

B.786.6. Exceptions

This instruction does not generate synchronous exceptions.

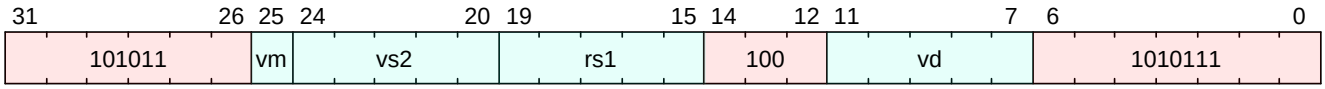
B.787. vssra.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.787.1. Encoding



B.787.2. Synopsis

No description available.

B.787.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.787.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.787.5. Execution

```

```

B.787.6. Exceptions

This instruction does not generate synchronous exceptions.

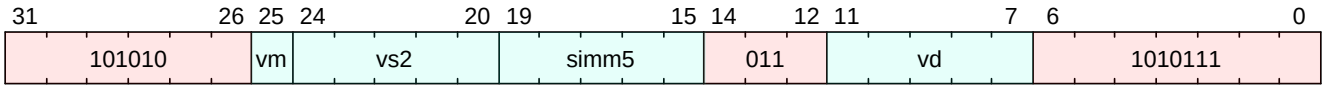
B.788. vssrl.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.788.1. Encoding



B.788.2. Synopsis

No description available.

B.788.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.788.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.788.5. Execution

```

```

B.788.6. Exceptions

This instruction does not generate synchronous exceptions.

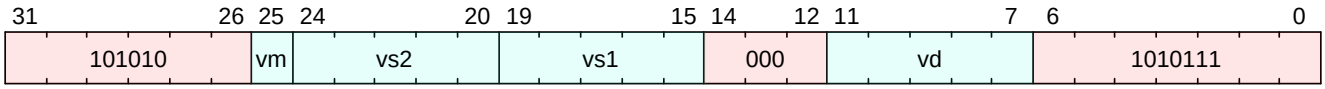
B.789. vssrl.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.789.1. Encoding



B.789.2. Synopsis

No description available.

B.789.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.789.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.789.5. Execution

```

```

B.789.6. Exceptions

This instruction does not generate synchronous exceptions.

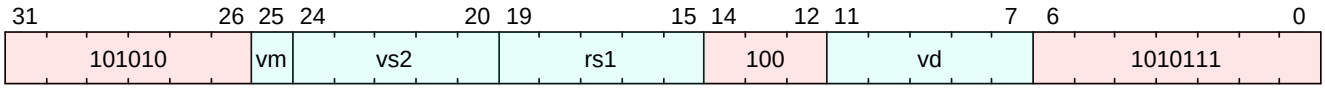
B.790. vssrl.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.790.1. Encoding



B.790.2. Synopsis

No description available.

B.790.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.790.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.790.5. Execution

```

```

B.790.6. Exceptions

This instruction does not generate synchronous exceptions.

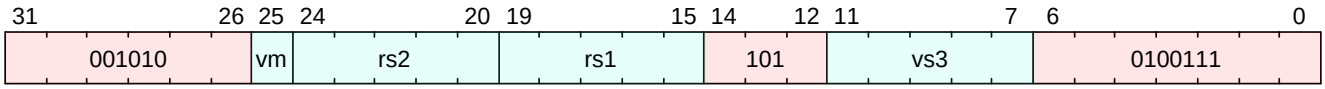
B.791. vssseg2e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.791.1. Encoding



B.791.2. Synopsis

No description available.

B.791.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.791.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.791.5. Execution

```

```

B.791.6. Exceptions

This instruction does not generate synchronous exceptions.

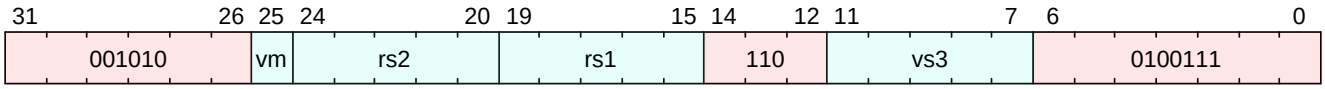
B.792. vssseg2e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.792.1. Encoding



B.792.2. Synopsis

No description available.

B.792.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.792.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.792.5. Execution

```

```

B.792.6. Exceptions

This instruction does not generate synchronous exceptions.

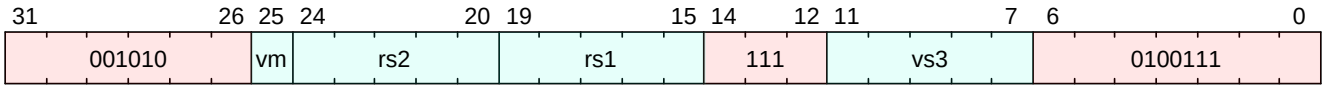
B.793. vssseg2e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.793.1. Encoding



B.793.2. Synopsis

No description available.

B.793.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.793.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.793.5. Execution

```

```

B.793.6. Exceptions

This instruction does not generate synchronous exceptions.

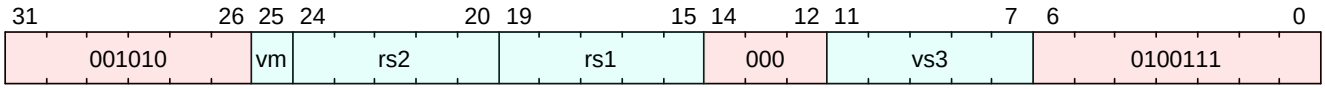
B.794. vssseg2e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.794.1. Encoding



B.794.2. Synopsis

No description available.

B.794.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.794.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.794.5. Execution

```

```

B.794.6. Exceptions

This instruction does not generate synchronous exceptions.

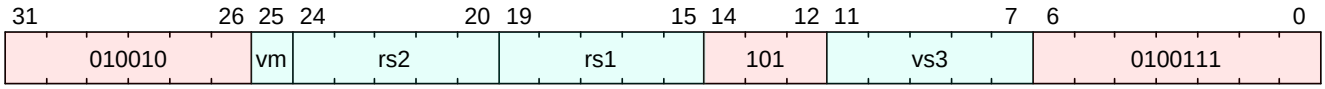
B.795. vssseg3e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.795.1. Encoding



B.795.2. Synopsis

No description available.

B.795.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.795.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.795.5. Execution

```

```

B.795.6. Exceptions

This instruction does not generate synchronous exceptions.

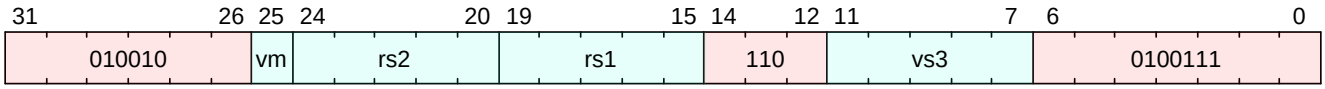
B.796. vssseg3e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.796.1. Encoding



B.796.2. Synopsis

No description available.

B.796.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.796.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.796.5. Execution

```

```

B.796.6. Exceptions

This instruction does not generate synchronous exceptions.

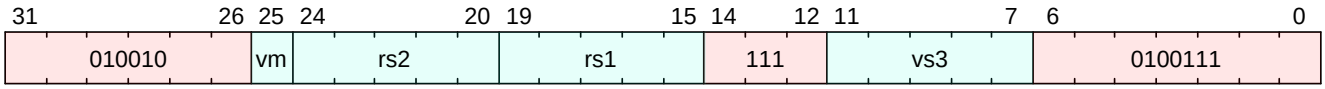
B.797. vssseg3e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.797.1. Encoding



B.797.2. Synopsis

No description available.

B.797.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.797.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.797.5. Execution

```

```

B.797.6. Exceptions

This instruction does not generate synchronous exceptions.

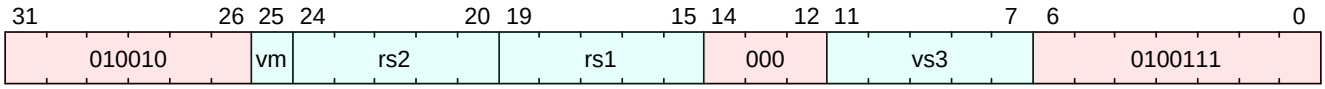
B.798. vssseg3e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.798.1. Encoding



B.798.2. Synopsis

No description available.

B.798.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.798.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.798.5. Execution

```

```

B.798.6. Exceptions

This instruction does not generate synchronous exceptions.

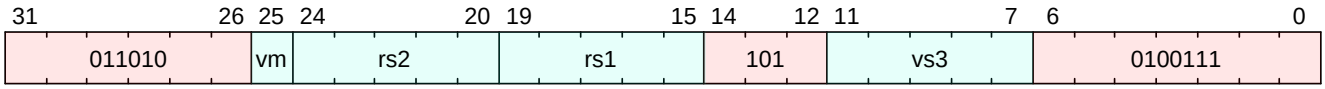
B.799. vssseg4e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.799.1. Encoding



B.799.2. Synopsis

No description available.

B.799.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.799.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.799.5. Execution

```

```

B.799.6. Exceptions

This instruction does not generate synchronous exceptions.

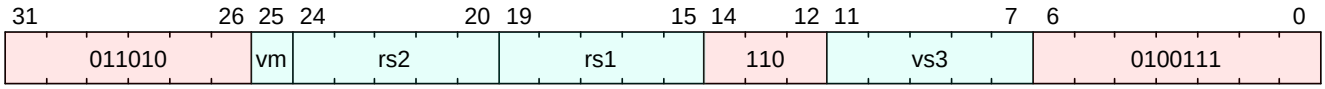
B.800. vssseg4e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.800.1. Encoding



B.800.2. Synopsis

No description available.

B.800.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.800.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.800.5. Execution

```

```

B.800.6. Exceptions

This instruction does not generate synchronous exceptions.

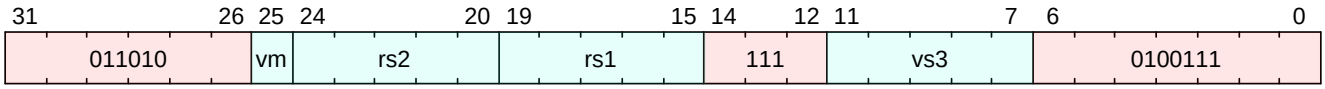
B.801. vssseg4e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.801.1. Encoding



B.801.2. Synopsis

No description available.

B.801.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.801.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.801.5. Execution

```

```

B.801.6. Exceptions

This instruction does not generate synchronous exceptions.

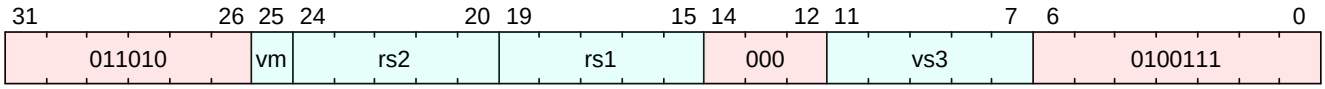
B.802. vssseg4e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.802.1. Encoding



B.802.2. Synopsis

No description available.

B.802.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.802.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.802.5. Execution

```

```

B.802.6. Exceptions

This instruction does not generate synchronous exceptions.

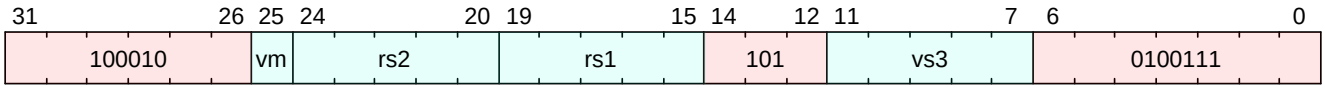
B.803. vssseg5e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.803.1. Encoding



B.803.2. Synopsis

No description available.

B.803.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.803.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.803.5. Execution

```

```

B.803.6. Exceptions

This instruction does not generate synchronous exceptions.

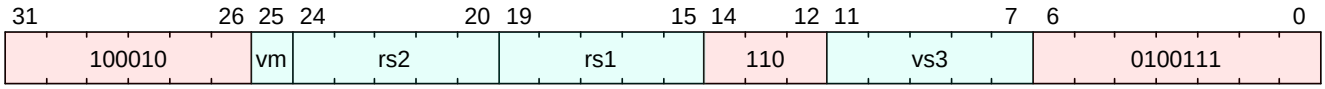
B.804. vssseg5e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.804.1. Encoding



B.804.2. Synopsis

No description available.

B.804.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.804.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.804.5. Execution

```

```

B.804.6. Exceptions

This instruction does not generate synchronous exceptions.

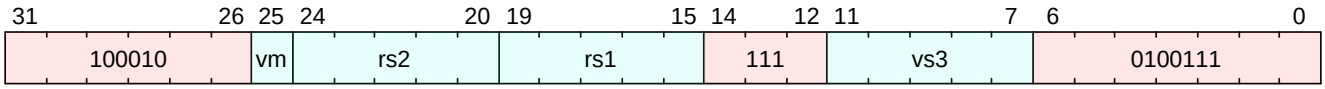
B.805. vssseg5e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.805.1. Encoding



B.805.2. Synopsis

No description available.

B.805.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.805.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.805.5. Execution

```

```

B.805.6. Exceptions

This instruction does not generate synchronous exceptions.

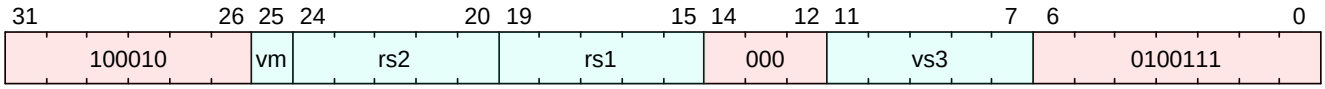
B.806. vssseg5e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.806.1. Encoding



B.806.2. Synopsis

No description available.

B.806.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.806.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.806.5. Execution

```

```

B.806.6. Exceptions

This instruction does not generate synchronous exceptions.

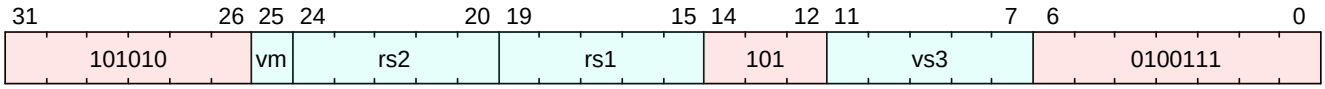
B.807. vssseg6e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.807.1. Encoding



B.807.2. Synopsis

No description available.

B.807.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.807.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.807.5. Execution

```

```

B.807.6. Exceptions

This instruction does not generate synchronous exceptions.

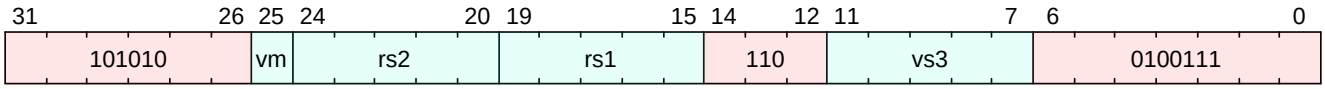
B.808. vssseg6e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.808.1. Encoding



B.808.2. Synopsis

No description available.

B.808.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.808.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.808.5. Execution

```

```

B.808.6. Exceptions

This instruction does not generate synchronous exceptions.

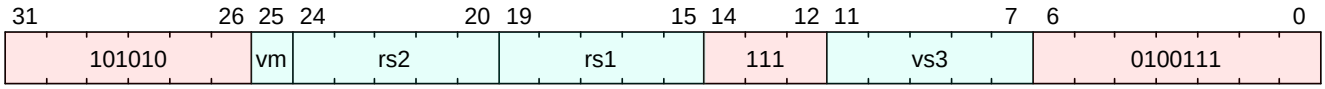
B.809. vssseg6e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.809.1. Encoding



B.809.2. Synopsis

No description available.

B.809.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.809.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.809.5. Execution

```

```

B.809.6. Exceptions

This instruction does not generate synchronous exceptions.

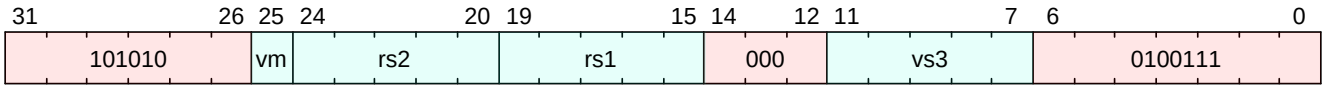
B.810. vssseg6e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.810.1. Encoding



B.810.2. Synopsis

No description available.

B.810.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.810.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.810.5. Execution

```

```

B.810.6. Exceptions

This instruction does not generate synchronous exceptions.

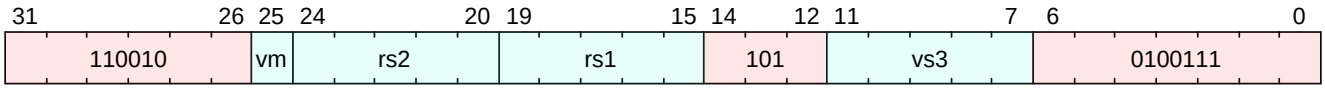
B.811. vssseg7e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.811.1. Encoding



B.811.2. Synopsis

No description available.

B.811.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.811.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.811.5. Execution

```

```

B.811.6. Exceptions

This instruction does not generate synchronous exceptions.

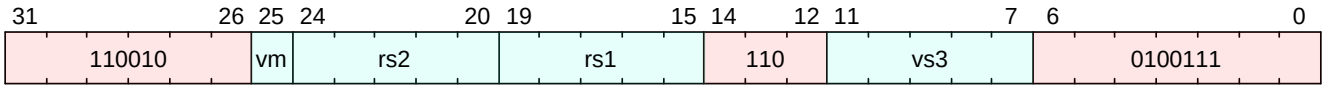
B.812. vssseg7e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.812.1. Encoding



B.812.2. Synopsis

No description available.

B.812.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.812.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.812.5. Execution

```

```

B.812.6. Exceptions

This instruction does not generate synchronous exceptions.

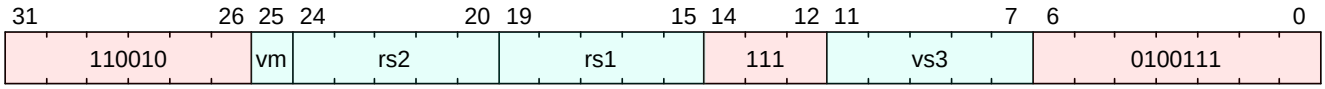
B.813. vssseg7e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.813.1. Encoding



B.813.2. Synopsis

No description available.

B.813.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.813.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.813.5. Execution

```

```

B.813.6. Exceptions

This instruction does not generate synchronous exceptions.

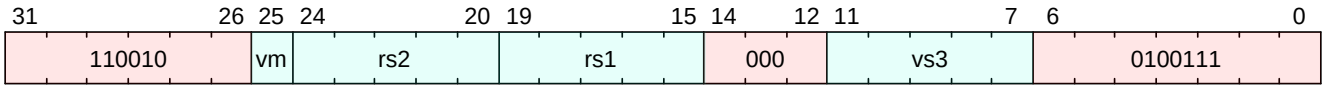
B.814. vssseg7e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.814.1. Encoding



B.814.2. Synopsis

No description available.

B.814.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.814.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.814.5. Execution

```

```

B.814.6. Exceptions

This instruction does not generate synchronous exceptions.

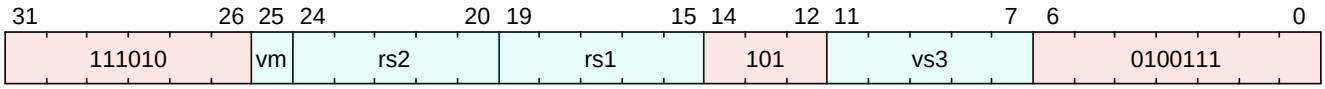
B.815. vssseg8e16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.815.1. Encoding



B.815.2. Synopsis

No description available.

B.815.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.815.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.815.5. Execution

```

```

B.815.6. Exceptions

This instruction does not generate synchronous exceptions.

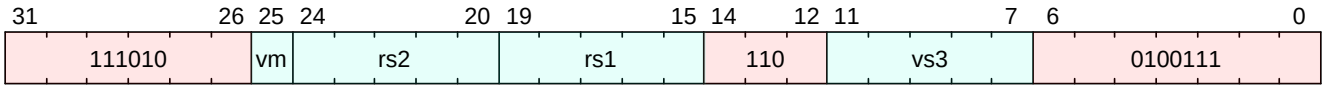
B.816. vssseg8e32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.816.1. Encoding



B.816.2. Synopsis

No description available.

B.816.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.816.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.816.5. Execution

```

```

B.816.6. Exceptions

This instruction does not generate synchronous exceptions.

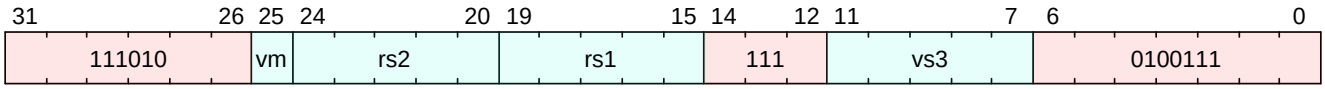
B.817. vssseg8e64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.817.1. Encoding



B.817.2. Synopsis

No description available.

B.817.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.817.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.817.5. Execution

```

```

B.817.6. Exceptions

This instruction does not generate synchronous exceptions.

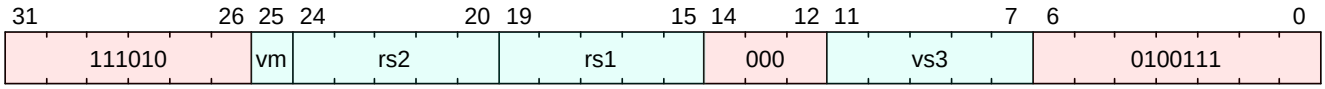
B.818. vssseg8e8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.818.1. Encoding



B.818.2. Synopsis

No description available.

B.818.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.818.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.818.5. Execution

```

```

B.818.6. Exceptions

This instruction does not generate synchronous exceptions.

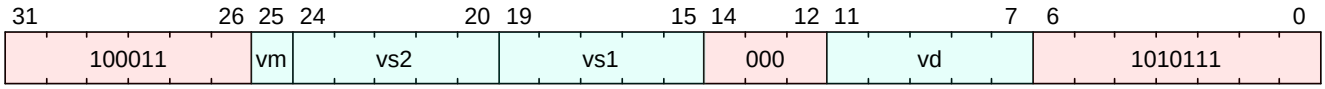
B.819. vssub.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.819.1. Encoding



B.819.2. Synopsis

No description available.

B.819.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.819.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.819.5. Execution

```

```

B.819.6. Exceptions

This instruction does not generate synchronous exceptions.

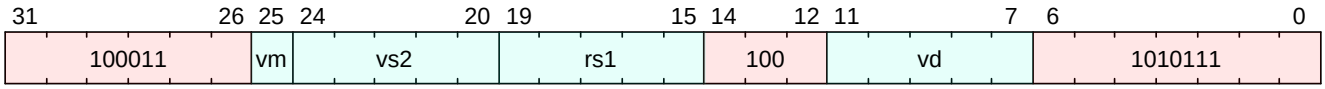
B.820. vssub.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.820.1. Encoding



B.820.2. Synopsis

No description available.

B.820.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.820.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.820.5. Execution

```

```

B.820.6. Exceptions

This instruction does not generate synchronous exceptions.

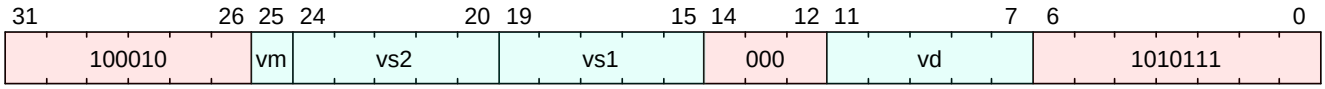
B.821. vssubu.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.821.1. Encoding



B.821.2. Synopsis

No description available.

B.821.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.821.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.821.5. Execution

```

```

B.821.6. Exceptions

This instruction does not generate synchronous exceptions.

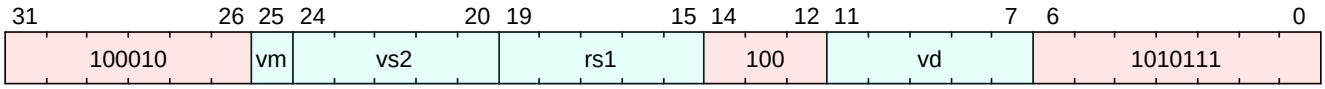
B.822. vssubu.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.822.1. Encoding



B.822.2. Synopsis

No description available.

B.822.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.822.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.822.5. Execution

```

```

B.822.6. Exceptions

This instruction does not generate synchronous exceptions.

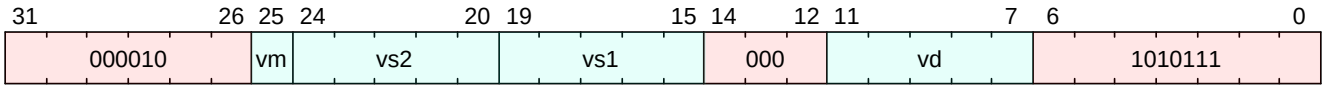
B.823. vsub.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.823.1. Encoding



B.823.2. Synopsis

No description available.

B.823.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.823.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.823.5. Execution

```

```

B.823.6. Exceptions

This instruction does not generate synchronous exceptions.

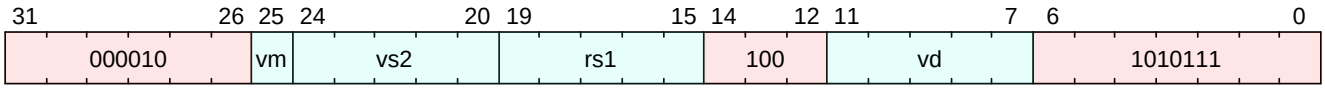
B.824. vsub.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.824.1. Encoding



B.824.2. Synopsis

No description available.

B.824.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.824.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.824.5. Execution

```

```

B.824.6. Exceptions

This instruction does not generate synchronous exceptions.

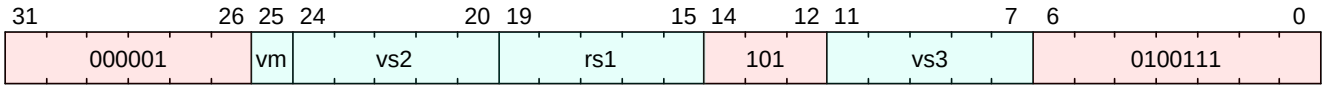
B.825. vsuxei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.825.1. Encoding



B.825.2. Synopsis

No description available.

B.825.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.825.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.825.5. Execution

```

```

B.825.6. Exceptions

This instruction does not generate synchronous exceptions.

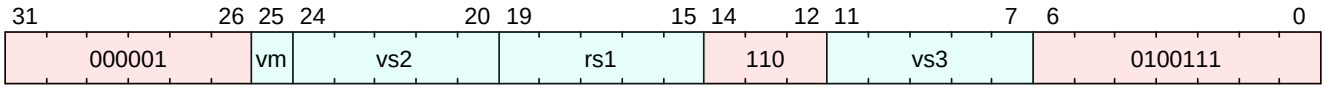
B.826. vsuxei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.826.1. Encoding



B.826.2. Synopsis

No description available.

B.826.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.826.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.826.5. Execution

```

```

B.826.6. Exceptions

This instruction does not generate synchronous exceptions.

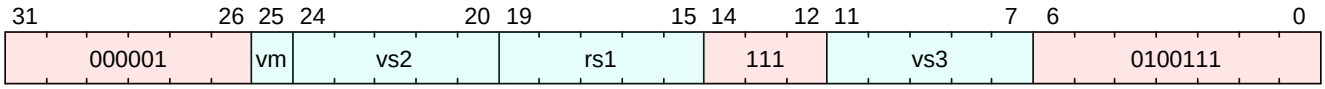
B.827. vsuxei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.827.1. Encoding



B.827.2. Synopsis

No description available.

B.827.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.827.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.827.5. Execution

```

```

B.827.6. Exceptions

This instruction does not generate synchronous exceptions.

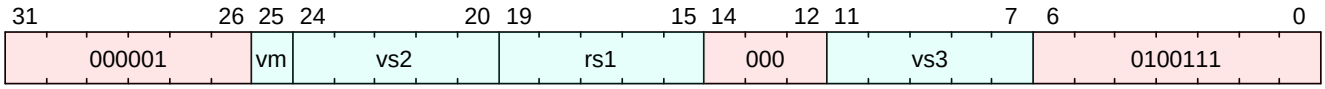
B.828. vsuxei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.828.1. Encoding



B.828.2. Synopsis

No description available.

B.828.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.828.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.828.5. Execution

```

```

B.828.6. Exceptions

This instruction does not generate synchronous exceptions.

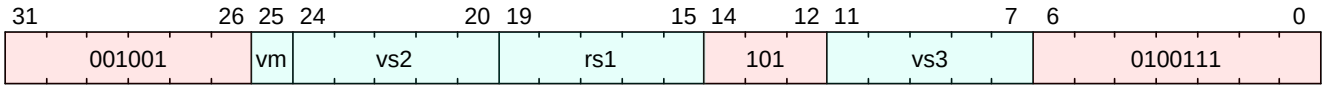
B.829. vsuxseg2ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.829.1. Encoding



B.829.2. Synopsis

No description available.

B.829.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.829.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.829.5. Execution

```

```

B.829.6. Exceptions

This instruction does not generate synchronous exceptions.

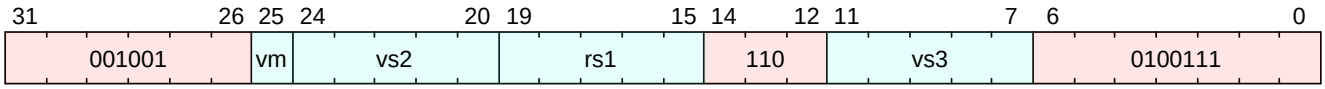
B.830. vsuxseg2ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.830.1. Encoding



B.830.2. Synopsis

No description available.

B.830.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.830.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.830.5. Execution

```

```

B.830.6. Exceptions

This instruction does not generate synchronous exceptions.

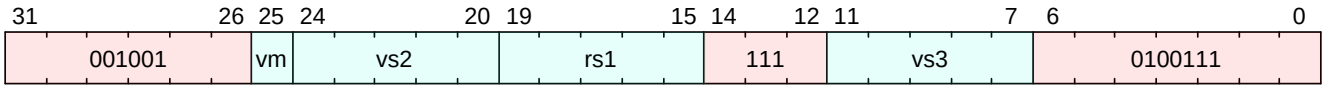
B.831. vsuxseg2ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.831.1. Encoding



B.831.2. Synopsis

No description available.

B.831.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.831.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.831.5. Execution

```

```

B.831.6. Exceptions

This instruction does not generate synchronous exceptions.

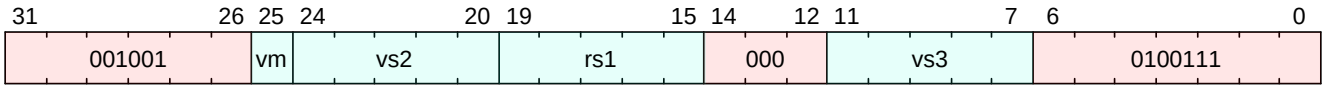
B.832. vsuxseg2ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.832.1. Encoding



B.832.2. Synopsis

No description available.

B.832.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.832.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.832.5. Execution

```

```

B.832.6. Exceptions

This instruction does not generate synchronous exceptions.

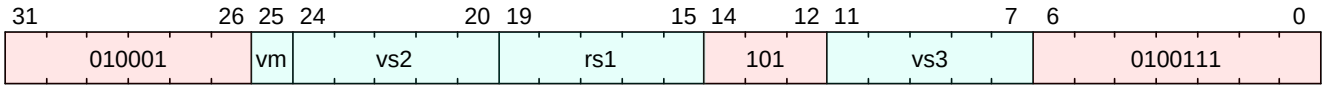
B.833. vsuxseg3ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.833.1. Encoding



B.833.2. Synopsis

No description available.

B.833.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.833.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.833.5. Execution

```

```

B.833.6. Exceptions

This instruction does not generate synchronous exceptions.

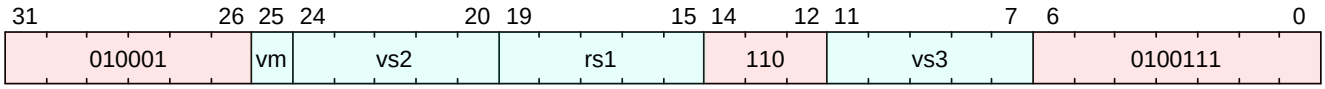
B.834. vsuxseg3ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.834.1. Encoding



B.834.2. Synopsis

No description available.

B.834.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.834.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.834.5. Execution

```

```

B.834.6. Exceptions

This instruction does not generate synchronous exceptions.

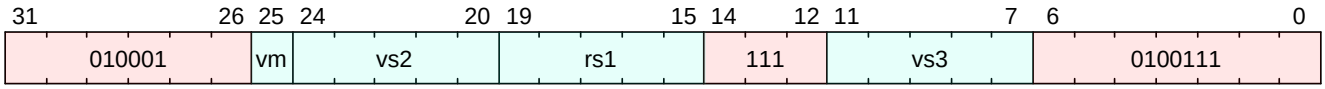
B.835. vsuxseg3ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.835.1. Encoding



B.835.2. Synopsis

No description available.

B.835.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.835.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.835.5. Execution

```

```

B.835.6. Exceptions

This instruction does not generate synchronous exceptions.

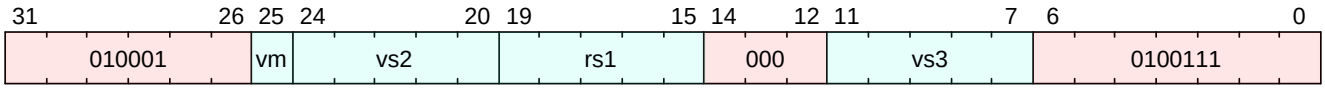
B.836. vsuxseg3ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.836.1. Encoding



B.836.2. Synopsis

No description available.

B.836.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.836.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.836.5. Execution

```

```

B.836.6. Exceptions

This instruction does not generate synchronous exceptions.

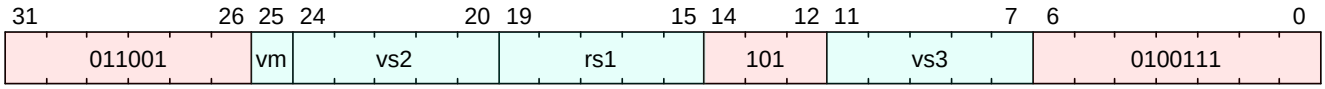
B.837. vsuxseg4ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.837.1. Encoding



B.837.2. Synopsis

No description available.

B.837.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.837.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.837.5. Execution

```

```

B.837.6. Exceptions

This instruction does not generate synchronous exceptions.

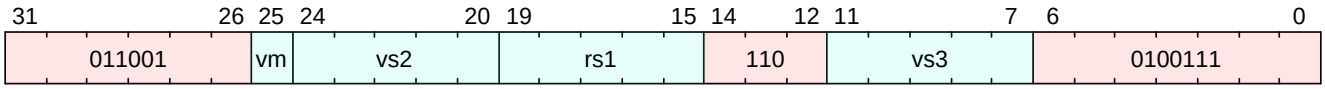
B.838. vsuxseg4ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.838.1. Encoding



B.838.2. Synopsis

No description available.

B.838.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.838.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.838.5. Execution

```

```

B.838.6. Exceptions

This instruction does not generate synchronous exceptions.

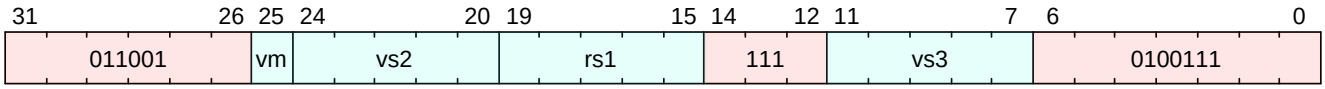
B.839. vsuxseg4ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.839.1. Encoding



B.839.2. Synopsis

No description available.

B.839.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.839.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.839.5. Execution

```

```

B.839.6. Exceptions

This instruction does not generate synchronous exceptions.

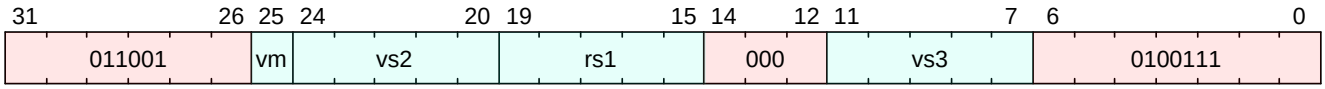
B.840. vsuxseg4ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.840.1. Encoding



B.840.2. Synopsis

No description available.

B.840.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.840.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.840.5. Execution

```

```

B.840.6. Exceptions

This instruction does not generate synchronous exceptions.

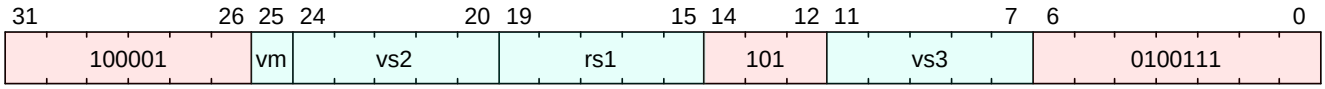
B.841. vsuxseg5ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.841.1. Encoding



B.841.2. Synopsis

No description available.

B.841.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.841.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.841.5. Execution

```

```

B.841.6. Exceptions

This instruction does not generate synchronous exceptions.

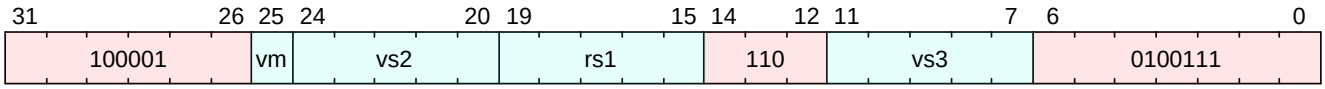
B.842. vsuxseg5ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.842.1. Encoding



B.842.2. Synopsis

No description available.

B.842.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.842.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.842.5. Execution

```

```

B.842.6. Exceptions

This instruction does not generate synchronous exceptions.

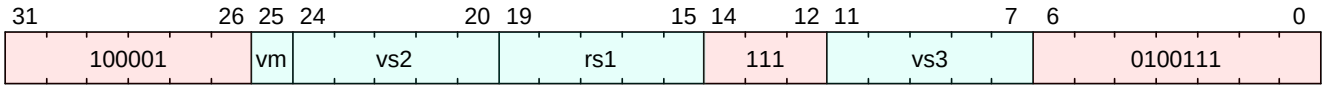
B.843. vsuxseg5ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.843.1. Encoding



B.843.2. Synopsis

No description available.

B.843.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.843.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.843.5. Execution

```

```

B.843.6. Exceptions

This instruction does not generate synchronous exceptions.

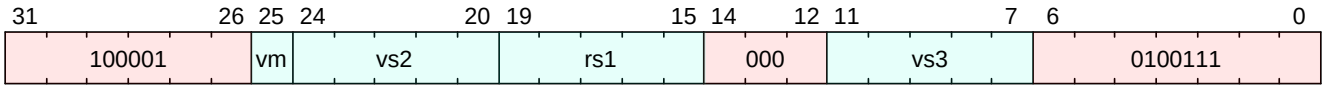
B.844. vsuxseg5ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.844.1. Encoding



B.844.2. Synopsis

No description available.

B.844.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.844.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.844.5. Execution

```

```

B.844.6. Exceptions

This instruction does not generate synchronous exceptions.

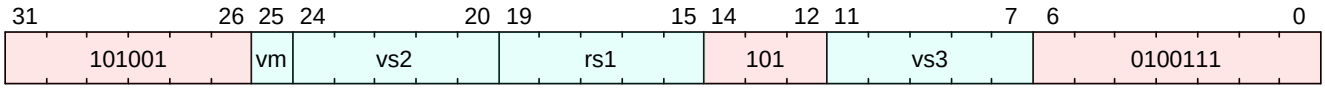
B.845. vsuxseg6ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.845.1. Encoding



B.845.2. Synopsis

No description available.

B.845.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.845.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.845.5. Execution

```

```

B.845.6. Exceptions

This instruction does not generate synchronous exceptions.

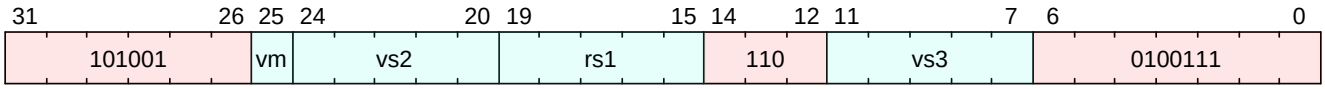
B.846. vsuxseg6ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.846.1. Encoding



B.846.2. Synopsis

No description available.

B.846.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.846.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.846.5. Execution

```

```

B.846.6. Exceptions

This instruction does not generate synchronous exceptions.

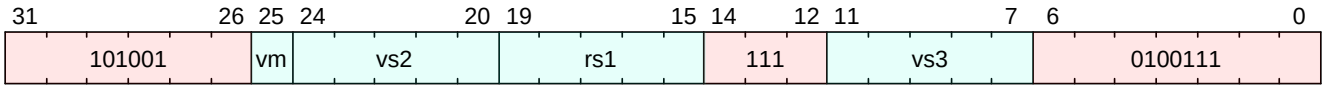
B.847. vsuxseg6ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.847.1. Encoding



B.847.2. Synopsis

No description available.

B.847.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.847.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.847.5. Execution

```

```

B.847.6. Exceptions

This instruction does not generate synchronous exceptions.

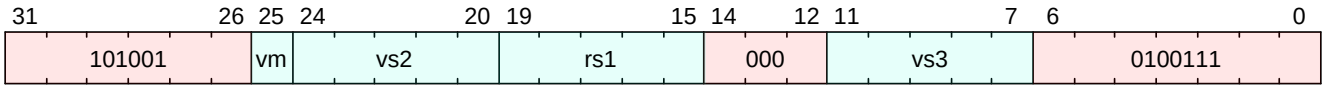
B.848. vsuxseg6ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.848.1. Encoding



B.848.2. Synopsis

No description available.

B.848.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.848.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.848.5. Execution

```

```

B.848.6. Exceptions

This instruction does not generate synchronous exceptions.

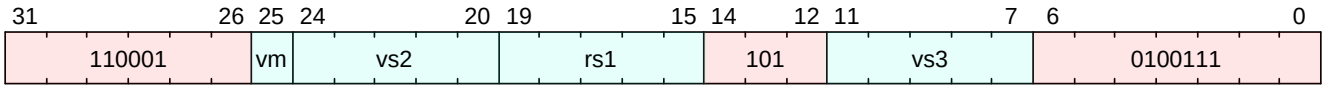
B.849. vsuxseg7ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.849.1. Encoding



B.849.2. Synopsis

No description available.

B.849.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.849.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.849.5. Execution

```

```

B.849.6. Exceptions

This instruction does not generate synchronous exceptions.

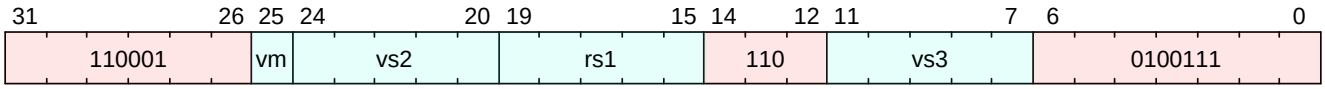
B.850. vsuxseg7ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.850.1. Encoding



B.850.2. Synopsis

No description available.

B.850.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.850.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.850.5. Execution

```

```

B.850.6. Exceptions

This instruction does not generate synchronous exceptions.

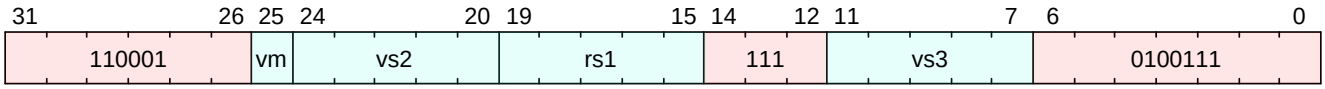
B.851. vsuxseg7ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.851.1. Encoding



B.851.2. Synopsis

No description available.

B.851.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.851.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.851.5. Execution

```

```

B.851.6. Exceptions

This instruction does not generate synchronous exceptions.

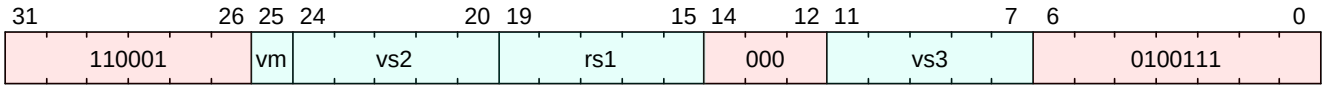
B.852. vsuxseg7ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.852.1. Encoding



B.852.2. Synopsis

No description available.

B.852.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.852.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.852.5. Execution

```

```

B.852.6. Exceptions

This instruction does not generate synchronous exceptions.

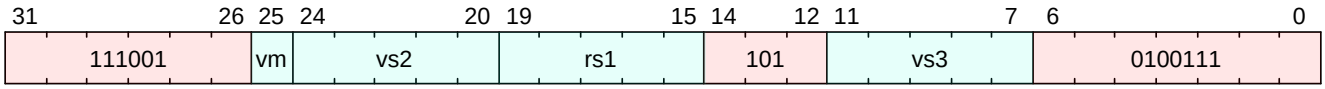
B.853. vsuxseg8ei16.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.853.1. Encoding



B.853.2. Synopsis

No description available.

B.853.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.853.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.853.5. Execution

```

```

B.853.6. Exceptions

This instruction does not generate synchronous exceptions.

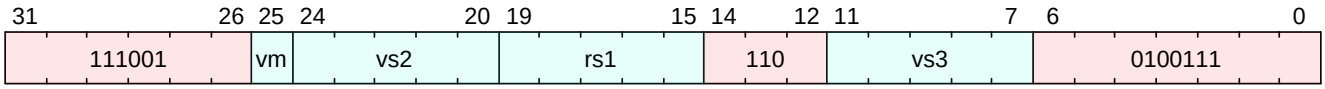
B.854. vsuxseg8ei32.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.854.1. Encoding



B.854.2. Synopsis

No description available.

B.854.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.854.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.854.5. Execution

```

```

B.854.6. Exceptions

This instruction does not generate synchronous exceptions.

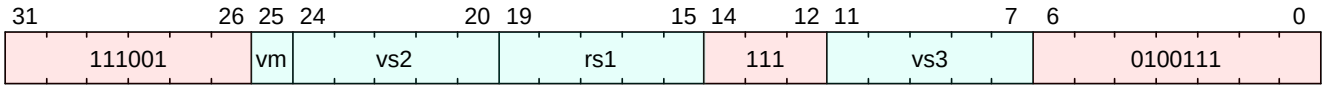
B.855. vsuxseg8ei64.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.855.1. Encoding



B.855.2. Synopsis

No description available.

B.855.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.855.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.855.5. Execution

```

```

B.855.6. Exceptions

This instruction does not generate synchronous exceptions.

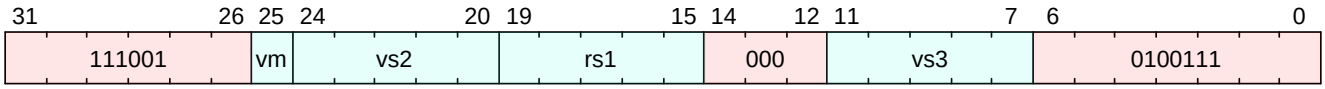
B.856. vsuxseg8ei8.v

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.856.1. Encoding



B.856.2. Synopsis

No description available.

B.856.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.856.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vs3 = $encoding[11:7];
```

B.856.5. Execution

```

```

B.856.6. Exceptions

This instruction does not generate synchronous exceptions.

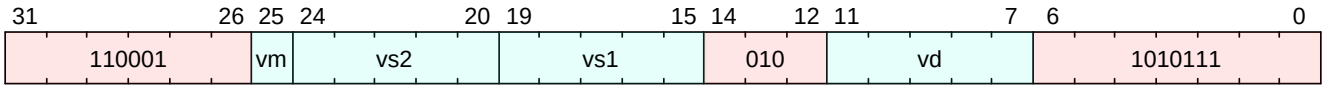
B.857. vwadd.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.857.1. Encoding



B.857.2. Synopsis

No description available.

B.857.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.857.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.857.5. Execution

```

```

B.857.6. Exceptions

This instruction does not generate synchronous exceptions.

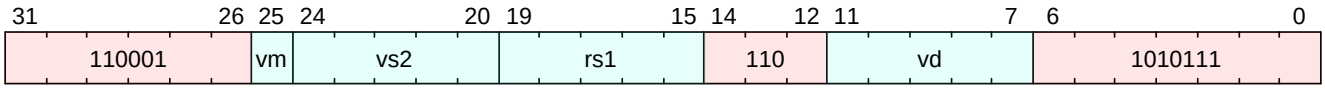
B.858. vwadd.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.858.1. Encoding



B.858.2. Synopsis

No description available.

B.858.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.858.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.858.5. Execution

```

```

B.858.6. Exceptions

This instruction does not generate synchronous exceptions.

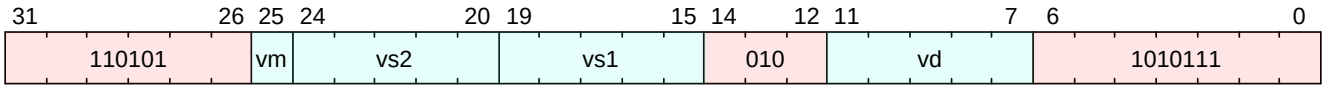
B.859. vwadd.wv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.859.1. Encoding



B.859.2. Synopsis

No description available.

B.859.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.859.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.859.5. Execution

```

```

B.859.6. Exceptions

This instruction does not generate synchronous exceptions.

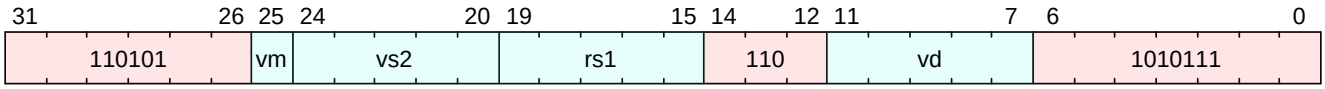
B.860. vwadd.wx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.860.1. Encoding



B.860.2. Synopsis

No description available.

B.860.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.860.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.860.5. Execution

```

```

B.860.6. Exceptions

This instruction does not generate synchronous exceptions.

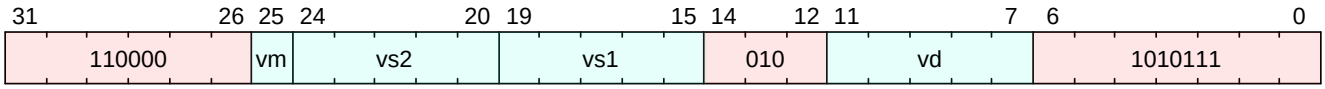
B.861. vwaddu.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.861.1. Encoding



B.861.2. Synopsis

No description available.

B.861.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.861.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.861.5. Execution

```

```

B.861.6. Exceptions

This instruction does not generate synchronous exceptions.

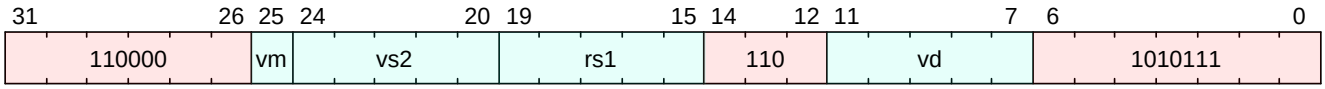
B.862. vwaddu.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.862.1. Encoding



B.862.2. Synopsis

No description available.

B.862.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.862.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.862.5. Execution

```

```

B.862.6. Exceptions

This instruction does not generate synchronous exceptions.

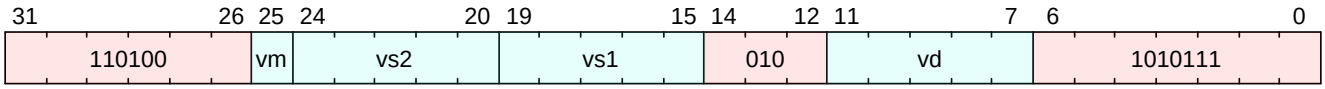
B.863. vwaddu.wv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.863.1. Encoding



B.863.2. Synopsis

No description available.

B.863.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.863.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.863.5. Execution

```

```

B.863.6. Exceptions

This instruction does not generate synchronous exceptions.

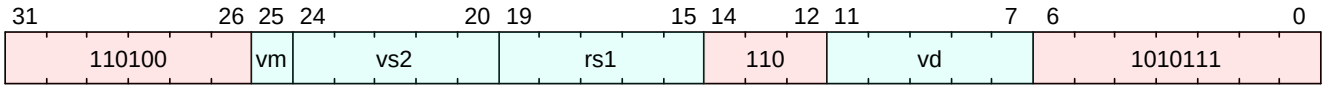
B.864. vwaddu.wx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.864.1. Encoding



B.864.2. Synopsis

No description available.

B.864.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.864.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.864.5. Execution

```

```

B.864.6. Exceptions

This instruction does not generate synchronous exceptions.

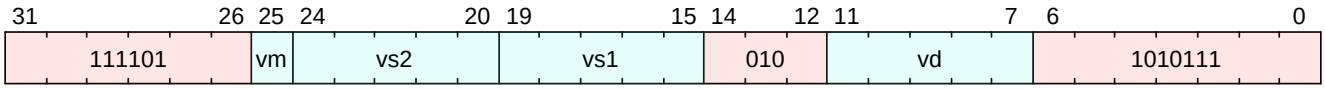
B.865. vwmacc.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.865.1. Encoding



B.865.2. Synopsis

No description available.

B.865.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.865.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.865.5. Execution

```

```

B.865.6. Exceptions

This instruction does not generate synchronous exceptions.

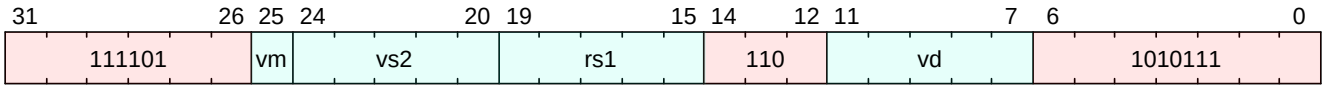
B.866. vwmacc.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.866.1. Encoding



B.866.2. Synopsis

No description available.

B.866.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.866.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.866.5. Execution

```

```

B.866.6. Exceptions

This instruction does not generate synchronous exceptions.

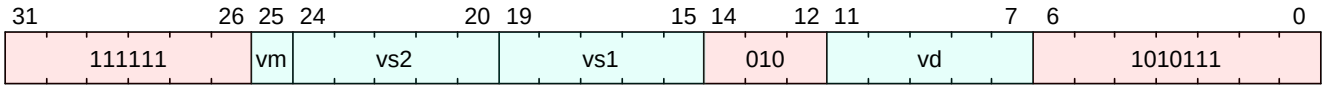
B.867. vwmaccsu.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.867.1. Encoding



B.867.2. Synopsis

No description available.

B.867.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.867.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.867.5. Execution

```

```

B.867.6. Exceptions

This instruction does not generate synchronous exceptions.

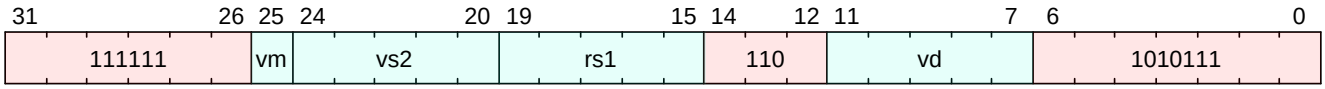
B.868. vwmaccsu.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.868.1. Encoding



B.868.2. Synopsis

No description available.

B.868.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.868.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.868.5. Execution

```

```

B.868.6. Exceptions

This instruction does not generate synchronous exceptions.

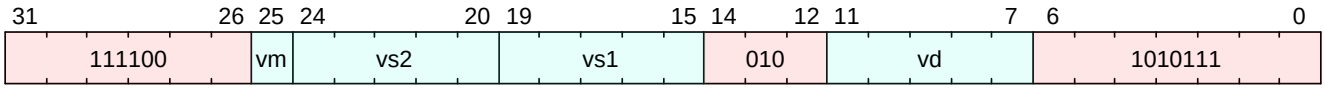
B.869. vwmaccu.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.869.1. Encoding



B.869.2. Synopsis

No description available.

B.869.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.869.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.869.5. Execution

```

```

B.869.6. Exceptions

This instruction does not generate synchronous exceptions.

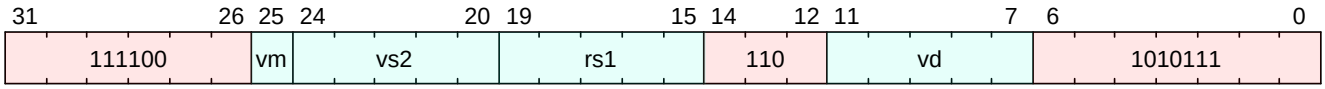
B.870. vwmaccu.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.870.1. Encoding



B.870.2. Synopsis

No description available.

B.870.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.870.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.870.5. Execution

```

```

B.870.6. Exceptions

This instruction does not generate synchronous exceptions.

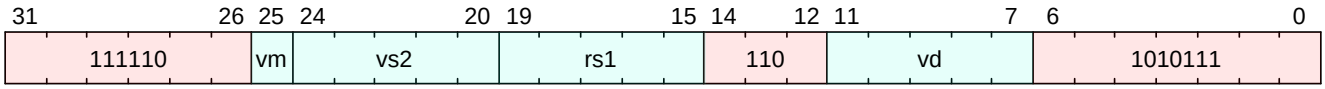
B.871. vwmaccus.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.871.1. Encoding



B.871.2. Synopsis

No description available.

B.871.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.871.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.871.5. Execution

```

```

B.871.6. Exceptions

This instruction does not generate synchronous exceptions.

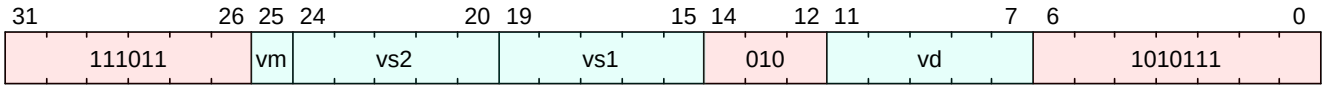
B.872. vwmul.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.872.1. Encoding



B.872.2. Synopsis

No description available.

B.872.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.872.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.872.5. Execution

```

```

B.872.6. Exceptions

This instruction does not generate synchronous exceptions.

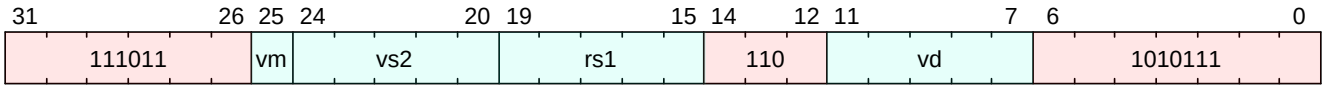
B.873. vwmul.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.873.1. Encoding



B.873.2. Synopsis

No description available.

B.873.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.873.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.873.5. Execution

```

```

B.873.6. Exceptions

This instruction does not generate synchronous exceptions.

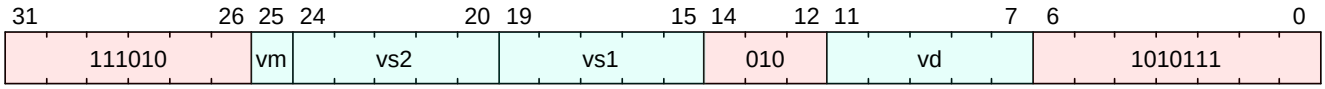
B.874. vwmulsu.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.874.1. Encoding



B.874.2. Synopsis

No description available.

B.874.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.874.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.874.5. Execution

```

```

B.874.6. Exceptions

This instruction does not generate synchronous exceptions.

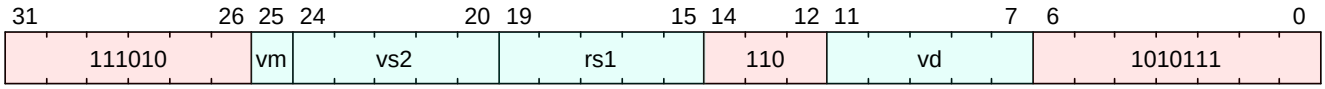
B.875. vwmulsu.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.875.1. Encoding



B.875.2. Synopsis

No description available.

B.875.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.875.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.875.5. Execution

```

```

B.875.6. Exceptions

This instruction does not generate synchronous exceptions.

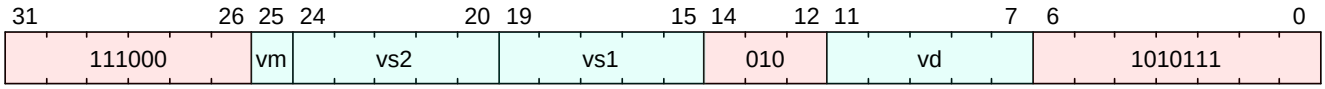
B.876. vwmulu.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.876.1. Encoding



B.876.2. Synopsis

No description available.

B.876.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.876.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.876.5. Execution

```

```

B.876.6. Exceptions

This instruction does not generate synchronous exceptions.

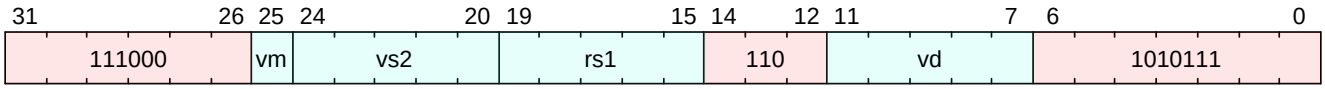
B.877. vwmulu.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.877.1. Encoding



B.877.2. Synopsis

No description available.

B.877.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.877.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.877.5. Execution

```

```

B.877.6. Exceptions

This instruction does not generate synchronous exceptions.

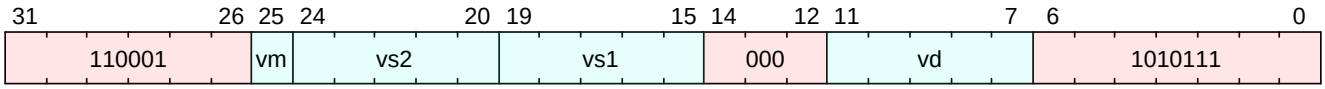
B.878. vwredsum.vs

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.878.1. Encoding



B.878.2. Synopsis

No description available.

B.878.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.878.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.878.5. Execution

```

```

B.878.6. Exceptions

This instruction does not generate synchronous exceptions.

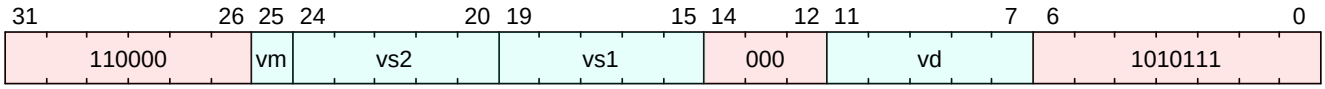
B.879. vwredsumu.vs

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.879.1. Encoding



B.879.2. Synopsis

No description available.

B.879.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.879.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.879.5. Execution

```

```

B.879.6. Exceptions

This instruction does not generate synchronous exceptions.

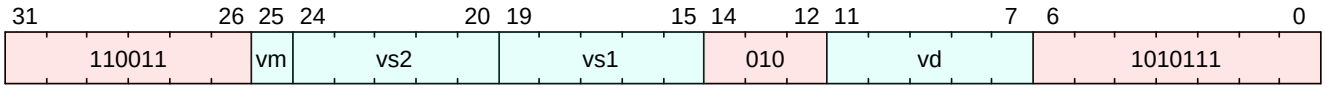
B.880. vwsub.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.880.1. Encoding



B.880.2. Synopsis

No description available.

B.880.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.880.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.880.5. Execution

```

```

B.880.6. Exceptions

This instruction does not generate synchronous exceptions.

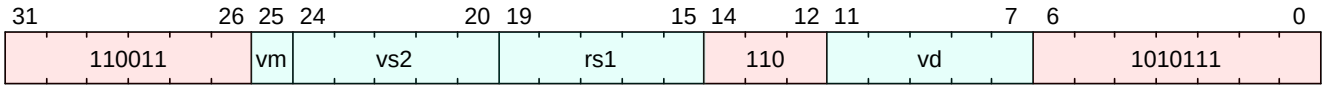
B.881. vwsub.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.881.1. Encoding



B.881.2. Synopsis

No description available.

B.881.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.881.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.881.5. Execution

```

```

B.881.6. Exceptions

This instruction does not generate synchronous exceptions.

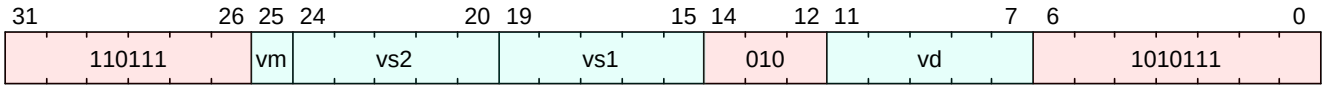
B.882. vwsub.wv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.882.1. Encoding



B.882.2. Synopsis

No description available.

B.882.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.882.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.882.5. Execution

```

```

B.882.6. Exceptions

This instruction does not generate synchronous exceptions.

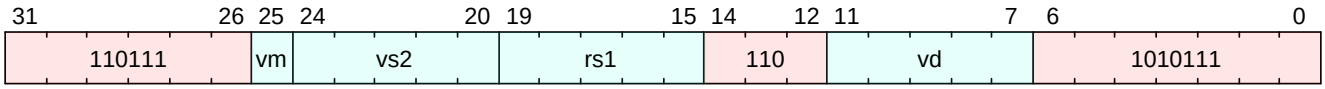
B.883. vwsub.wx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.883.1. Encoding



B.883.2. Synopsis

No description available.

B.883.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.883.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.883.5. Execution

```

```

B.883.6. Exceptions

This instruction does not generate synchronous exceptions.

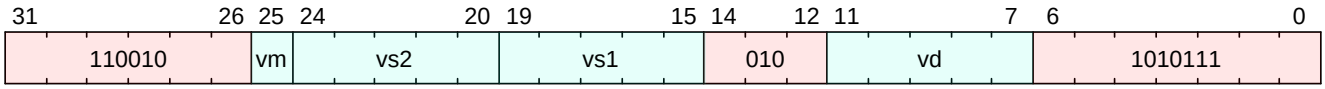
B.884. vwsubu.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.884.1. Encoding



B.884.2. Synopsis

No description available.

B.884.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.884.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.884.5. Execution

```

```

B.884.6. Exceptions

This instruction does not generate synchronous exceptions.

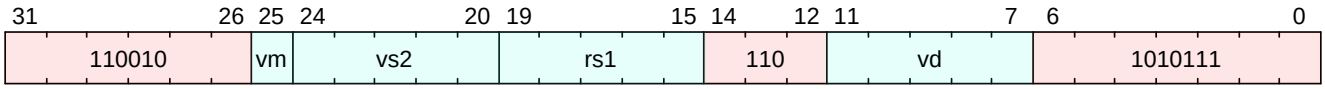
B.885. vwsubu.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.885.1. Encoding



B.885.2. Synopsis

No description available.

B.885.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.885.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.885.5. Execution

```

```

B.885.6. Exceptions

This instruction does not generate synchronous exceptions.

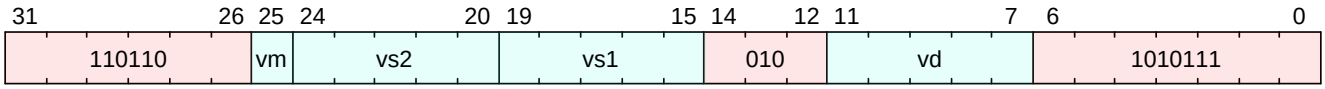
B.886. vwsubu.wv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.886.1. Encoding



B.886.2. Synopsis

No description available.

B.886.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.886.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.886.5. Execution

```

```

B.886.6. Exceptions

This instruction does not generate synchronous exceptions.

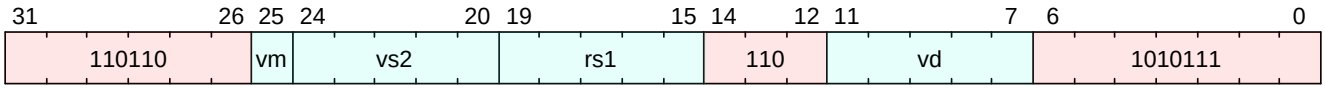
B.887. vwsubu.wx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.887.1. Encoding



B.887.2. Synopsis

No description available.

B.887.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.887.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.887.5. Execution

```

```

B.887.6. Exceptions

This instruction does not generate synchronous exceptions.

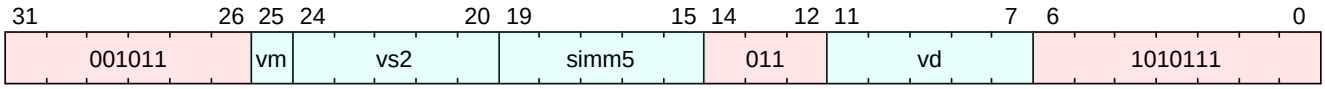
B.888. vxor.vi

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.888.1. Encoding



B.888.2. Synopsis

No description available.

B.888.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.888.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> simm5 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.888.5. Execution

```

```

B.888.6. Exceptions

This instruction does not generate synchronous exceptions.

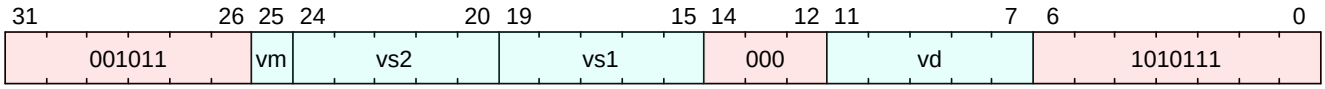
B.889. vxor.vv

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.889.1. Encoding



B.889.2. Synopsis

No description available.

B.889.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.889.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.889.5. Execution

```

```

B.889.6. Exceptions

This instruction does not generate synchronous exceptions.

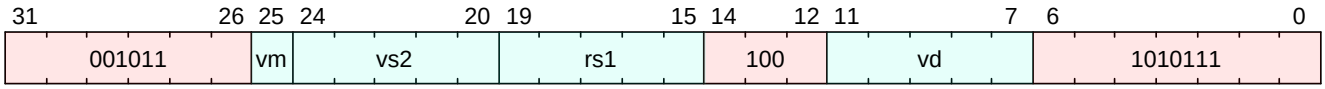
B.890. vxor.vx

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.890.1. Encoding



B.890.2. Synopsis

No description available.

B.890.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.890.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> vd = $encoding[11:7];
```

B.890.5. Execution

```

```

B.890.6. Exceptions

This instruction does not generate synchronous exceptions.

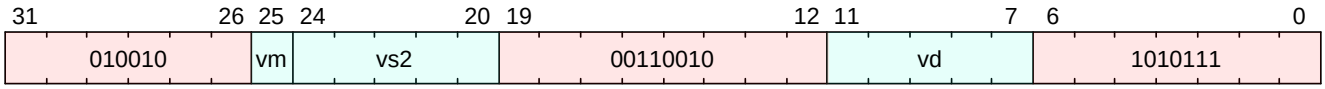
B.891. vzext.vf2

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.891.1. Encoding



B.891.2. Synopsis

No description available.

B.891.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.891.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.891.5. Execution

```

```

B.891.6. Exceptions

This instruction does not generate synchronous exceptions.

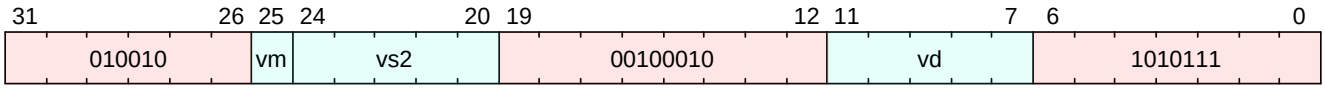
B.892. vzext.vf4

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.892.1. Encoding



B.892.2. Synopsis

No description available.

B.892.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.892.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.892.5. Execution

```

```

B.892.6. Exceptions

This instruction does not generate synchronous exceptions.

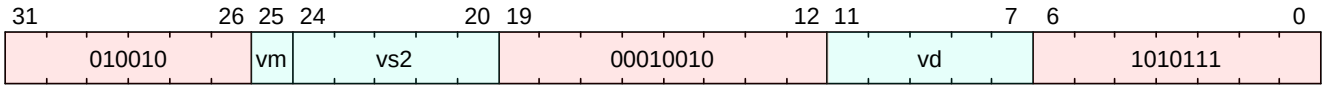
B.893. vzext.vf8

No synopsis available.

This instruction is defined by:

- V, version >= 0

B.893.1. Encoding



B.893.2. Synopsis

No description available.

B.893.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.893.4. Decode Variables

```
Bits<1> vm = $encoding[25];
Bits<5> vs2 = $encoding[24:20];
Bits<5> vd = $encoding[11:7];
```

B.893.5. Execution

```

```

B.893.6. Exceptions

This instruction does not generate synchronous exceptions.

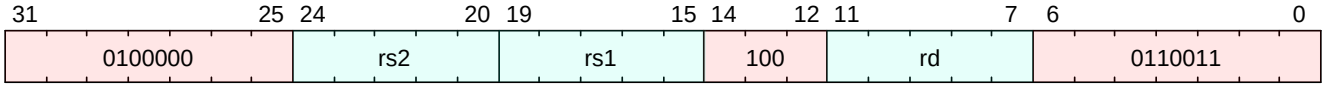
B.894. xnor

Exclusive NOR

This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0
  - Zbkb, version >= 0
  - Zk, version >= 0
  - Zkn, version >= 0
  - Zks, version >= 0

B.894.1. Encoding



B.894.2. Synopsis

This instruction performs the bit-wise exclusive-NOR operation on rs1 and rs2.

B.894.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.894.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.894.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = ~(X[rs1] ^ X[rs2]);
```

B.894.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

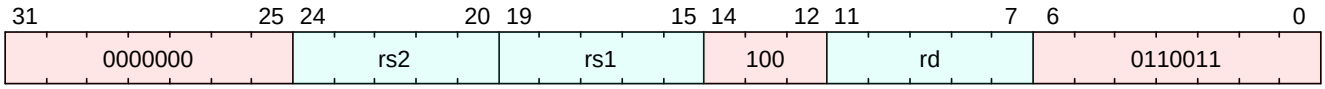
B.895. xor

Exclusive Or

This instruction is defined by:

- I, version >= 0

B.895.1. Encoding



B.895.2. Synopsis

Exclusive or rs1 with rs2, and store the result in rd

B.895.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.895.4. Decode Variables

```
Bits<5> rs2 = $encoding[24:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.895.5. Execution

```
X[rd] = X[rs1] ^ X[rs2];
```

B.895.6. Exceptions

This instruction does not generate synchronous exceptions.

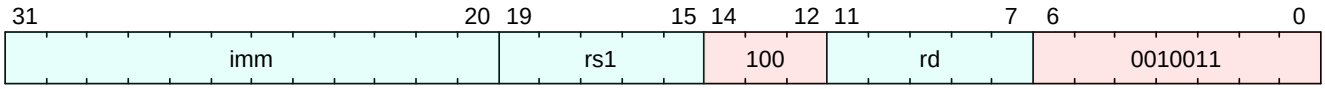
B.896. xori

Exclusive Or immediate

This instruction is defined by:

- I, version >= 0

B.896.1. Encoding



B.896.2. Synopsis

Exclusive or an immediate to the value in rs1, and store the result in rd

B.896.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.896.4. Decode Variables

```
Bits<12> imm = $encoding[31:20];
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.896.5. Execution

```
X[rd] = X[rs1] ^ imm;
```

B.896.6. Exceptions

This instruction does not generate synchronous exceptions.

B.897. zext.h

Zero-extend halfword

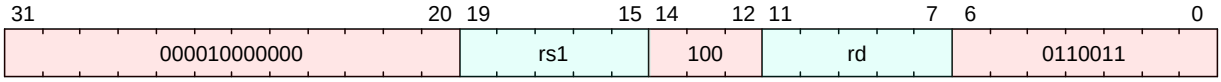
This instruction is defined by:

- anyOf:
  - B, version >= 0
  - Zbb, version >= 0

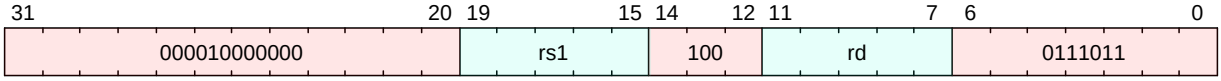
B.897.1. Encoding

This instruction has different encodings in RV32 and RV64.

RV32



RV64



B.897.2. Synopsis

This instruction zero-extends the least-significant halfword of the source to XLEN by inserting 0's into all of the bits more significant than 15.

The **zext.h** instruction is a pseudo-op for pack when Zbkb is implemented and XLEN == 32.

The **zext.h** instruction is a pseudo-op for packw when Zbkb is implemented and XLEN == 64.

B.897.3. Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

B.897.4. Decode Variables

RV32

```
Bits<5> rs1 = $encoding[19:15];
```

```
Bits<5> rd = $encoding[11:7];
```

RV64

```
Bits<5> rs1 = $encoding[19:15];
Bits<5> rd = $encoding[11:7];
```

B.897.5. Execution

```
if (implemented?(ExtensionName::B) && (CSR[misa].B == 1'b0)) {
  raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
}
X[rd] = X[rs1][15:0];
```

B.897.6. Exceptions

This instruction may result in the following synchronous exceptions:

- IllegalInstruction

Appendix C: CSR Details

C.1. cycle

Cycle counter for RDCYCLE Instruction

Alias for M-mode CSR mcycle.

Privilege mode access is controlled with mcounteren.CY, scounteren.CY, and hcounteren.CY as follows:

mcounteren.CY	scounteren.CY	hcounteren.CY	cycle behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.1.1. Attributes

CSR Address	0xc00
Defining extension	• Zicntr, version >= 0
Length	64-bit
Privilege Mode	U

C.1.2. Format

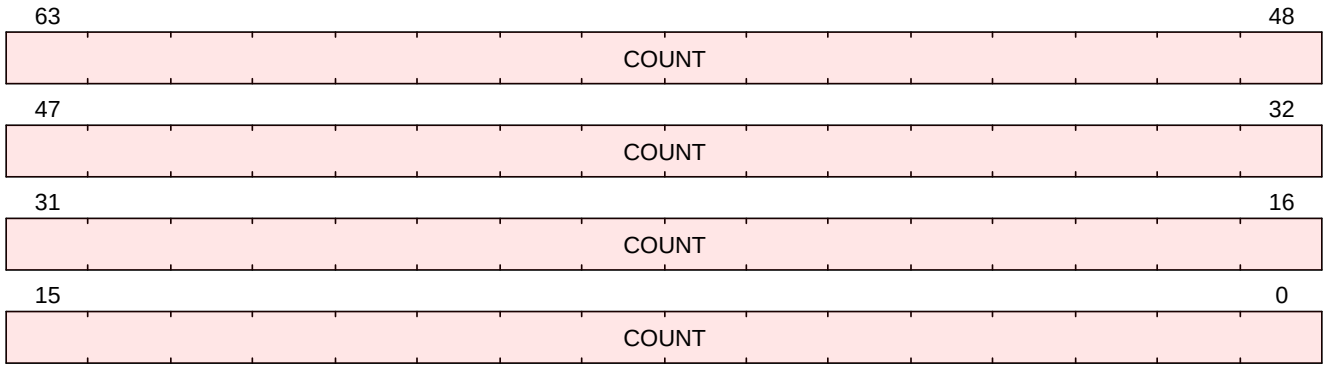


Figure 1. cycle format

C.2. cycleh

High-half cycle counter for RDCYCLE Instruction

cycleh is only defined in RV32.

Alias for M-mode CSR mcycleh.

Privilege mode access is controlled with mcounteren.CY, scounteren.CY, and hcounteren.CY as follows:

mcounteren.CY	scounteren.CY	hcounteren.CY	cycle behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.2.1. Attributes

CSR Address	0xc80
Defining extension	• Zicntr, version >= 0
Length	32-bit
Privilege Mode	U

C.2.2. Format

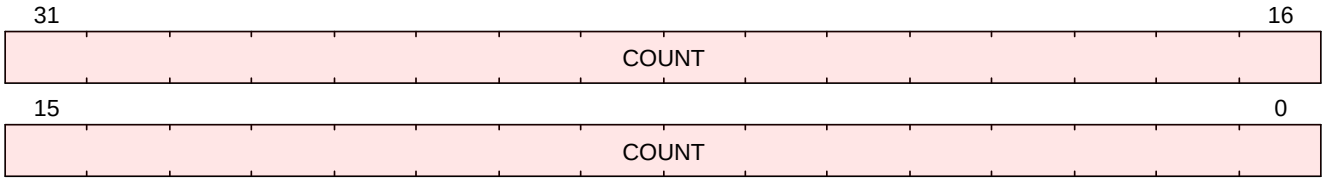


Figure 2. cycleh format

C.3. fcsr

Floating-point control and status register (`frm` + `fflags`)

The floating-point control and status register, fcsr, is a RISC-V control and status register (CSR). It is a 32-bit read/write register that selects the dynamic rounding mode for floating-point arithmetic operations and holds the accrued exception flags, as shown in Floating-Point Control and Status Register.

Floating-point control and status register

Unresolved directive in RVA22.adoc - include::images/wavedrom/float-csr.adoc[]

The fcsr register can be read and written with the FRCSR and FSCSR instructions, which are assembler pseudoinstructions built on the underlying CSR access instructions. FRCSR reads fcsr by copying it into integer register *rd*. FSCSR swaps the value in fcsr by copying the original value into integer register *rd*, and then writing a new value obtained from integer register *rs1* into fcsr.

The fields within the fcsr can also be accessed individually through different CSR addresses, and separate assembler pseudoinstructions are defined for these accesses. The FRRM instruction reads the Rounding Mode field `frm` (fcsr bits 7—5) and copies it into the least-significant three bits of integer register *rd*, with zero in all other bits. FSRM swaps the value in `frm` by copying the original value into integer register *rd*, and then writing a new value obtained from the three least-significant bits of integer register *rs1* into `frm`. FRFLAGS and FSFLAGS are defined analogously for the Accrued Exception Flags field `fflags` (fcsr bits 4—0).

Bits 31—8 of the fcsr are reserved for other standard extensions. If these extensions are not present, implementations shall ignore writes to these bits and supply a zero value when read. Standard software should preserve the contents of these bits.

Floating-point operations use either a static rounding mode encoded in the instruction, or a dynamic rounding mode held in `frm`. Rounding modes are encoded as shown in Table 9. A value of 111 in the instruction's *rm* field selects the dynamic rounding mode held in `frm`. The behavior of floating-point instructions that depend on rounding mode when executed with a reserved rounding mode is *reserved*, including both static reserved rounding modes (101-110) and dynamic reserved rounding modes (101-111). Some instructions, including widening conversions, have the *rm* field but are nevertheless mathematically unaffected by the rounding mode; software should set their *rm* field to RNE (000) but implementations must treat the *rm* field as usual (in particular, with regard to decoding legal vs. reserved encodings).

The C99 language standard effectively mandates the provision of a dynamic rounding mode register. In typical implementations, writes to the dynamic rounding mode CSR state will serialize the pipeline. Static rounding modes are used to implement specialized arithmetic operations that often have to switch frequently between different rounding modes.

The ratified version of the F spec mandated that an illegal-instruction exception was raised when an instruction was executed with a reserved dynamic rounding mode. This has been weakened to reserved, which matches the behavior of static rounding-mode instructions. Raising an illegal-instruction exception is still valid

behavior when encountering a reserved encoding, so implementations compatible with the ratified spec are compatible with the weakened spec.

The accrued exception flags indicate the exception conditions that have arisen on any floating-point arithmetic instruction since the field was last reset by software, as shown in Table 10. The base RISC-V ISA does not support generating a trap on the setting of a floating-point exception flag.

Table 57. Accrued exception flag encoding.

Flag Mnemonic	Flag Meaning
NV	Invalid Operation
DZ	Divide by Zero
OF	Overflow
UF	Underflow
NX	Inexact

As allowed by the standard, we do not support traps on floating-point exceptions in the F extension, but instead require explicit checks of the flags in software. We considered adding branches controlled directly by the contents of the floating-point accrued exception flags, but ultimately chose to omit these instructions to keep the ISA simple.

C.3.1. Attributes

CSR Address	0x3
Defining extension	• F, version >= 0
Length	32-bit
Privilege Mode	U

C.3.2. Format

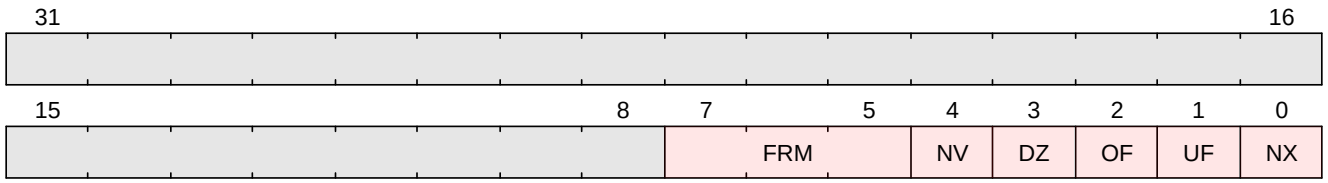


Figure 3. fcsr format

C.4. hcounteren

Hypervisor Counter Enable

Together with scounteren, delegates control of the hardware performance-monitoring counters to VS/VU-mode

See cycle for a table describing how exceptions occur.

C.4.1. Attributes

CSR Address	0x606
Defining extension	• H, version >= 0
Length	32-bit
Privilege Mode	S

C.4.2. Format

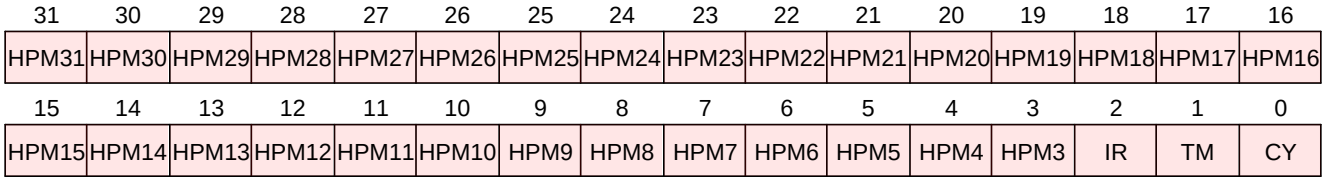


Figure 4. hcounteren format

C.5. hedeleg

Hypervisor Exception Delegation

Controls exception delegation from HS-mode to VS-mode.

By default, all traps at any privilege level are handled in M-mode, though M-mode usually uses the medeleg and mideleg CSRs to delegate some traps to HS-mode. The hedeleg and `hideleg` CSRs allow these traps to be further delegated to a VS-mode guest; their layout is the same as medeleg and mideleg.

A synchronous trap that has been delegated to HS-mode (using medeleg) is further delegated to VS-mode if V=1 before the trap and the corresponding hedeleg bit is set. Each bit of hedeleg shall be either writable or read-only zero. Many bits of hedeleg are required specifically to be writable or zero. Bit 0, corresponding to instruction address misaligned exceptions, must be writable if IALIGN=32.

Requiring that certain bits of hedeleg be writable reduces some of the burden on a hypervisor to handle variations of implementation.

When XLEN=32, hedelegh is a 32-bit read/write register that aliases bits 63:32 of hedeleg. Register hedelegh does not exist when XLEN=64.

C.5.1. Attributes

CSR Address	0x602
Defining extension	• H, version >= 0
Length	64-bit
Privilege Mode	S

C.5.2. Format

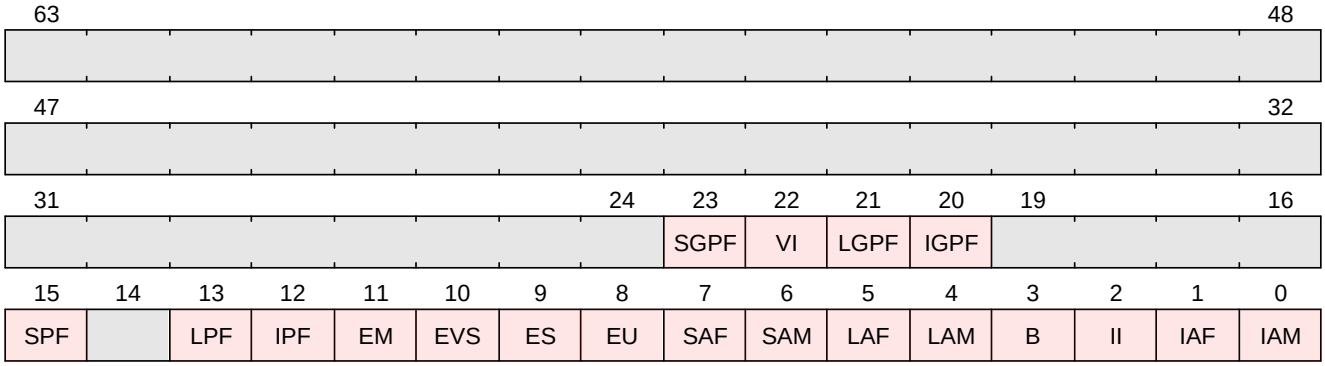


Figure 5. hedeleg format

C.6. hedelegh

Hypervisor Exception Delegation High

hedelegh is only defined in RV32.

Controls exception delegation from HS-mode to VS-mode.

Alias of upper bits of hedeleg[63:32].

C.6.1. Attributes

CSR Address	0x612
Defining extension	• H, version >= 0
Length	32-bit
Privilege Mode	S

C.6.2. Format

Figure 6. hedelegh format

C.7. hgatp

Hypervisor guest address translation and protection

The hgatp register is an HSXLEN-bit read/write register which controls G-stage address translation and protection, the second stage of two-stage translation for guest virtual addresses. Similar to CSR satp, this register holds the physical page number (PPN) of the guest-physical root page table; a virtual machine identifier (VMID), which facilitates address-translation fences on a per-virtual-machine basis; and the MODE field, which selects the address-translation scheme for guest physical addresses. When mstatus.TVM=1, attempts to read or write hgatp while executing in HS-mode will raise an `IllegalInstruction` exception.

Table 58 shows the encodings of the MODE field when HSXLEN=32 and HSXLEN=64. When MODE=Bare, guest physical addresses are equal to supervisor physical addresses, and there is no further memory protection for a guest virtual machine beyond the physical memory protection scheme described in [pmp]. In this case, the remaining fields in hgatp must be set to zeros.

Table 58. Encoding of hgatp MODE field.

HSXLEN=32		
Value	Name	Description
0	Bare	No translation or protection.
1	Sv32x4	Page-based 34-bit virtual addressing (2-bit extension of Sv32).
HSXLEN=64		
Value	Name	Description
0	Bare + — + Sv39x4	No translation or protection.
1-7	Sv48x4	*Reserved*
8	Sv57x4 + —	Page-based 41-bit virtual addressing (2-bit extension of Sv39).
9		Page-based 50-bit virtual addressing (2-bit extension of Sv48).
10		Page-based 59-bit virtual addressing (2-bit extension of Sv57).
11-15		*Reserved*

Implementations are not required to support all defined MODE settings when HSXLEN=64.

A write to hgatp with an unsupported MODE value is not ignored as it is for satp. Instead, the fields of hgatp are **WARL** in the normal way, when so indicated.

As explained in [guest-addr-translation], for the paged virtual-memory schemes (Sv32x4, Sv39x4, Sv48x4, and Sv57x4), the root page table is 16 KiB and must be aligned to a 16-KiB boundary. In these modes, the lowest two bits of the physical page number (PPN) in hgatp always read as zeros. An implementation that supports only the defined paged virtual-memory schemes and/or Bare may make PPN[1:0] read-only zero.

The number of VMID bits is UNSPECIFIED and may be zero. The number of implemented VMID bits, termed *VMIDLEN*, may be determined by writing one to every bit position in the VMID field, then reading back the value in hgatp to see which bit positions in the VMID field hold a one. The least-significant bits of VMID are implemented first: that is, if VMIDLEN > 0, VMID[VMIDLEN-1:0] is writable. The maximal value of VMIDLEN, termed VMIDMAX, is 7 for Sv32x4 or 14 for Sv39x4,

Sv48x4, and Sv57x4.

The hgatp register is considered *active* for the purposes of the address-translation algorithm *unless* the effective privilege mode is U and hstatus.HU=0.

This definition simplifies the implementation of speculative execution of `hlv`, `hlvx`, and 'hsv' instructions.

Note that writing hgatp does not imply any ordering constraints between page-table updates and subsequent G-stage address translations. If the new virtual machine's guest physical page tables have been modified, or if a VMID is reused, it may be necessary to execute an HFENCE.GVMA instruction (see [hfence.vma]) before or after writing hgatp.

C.7.1. Attributes

CSR Address	0x680
Defining extension	• H, version >= 0
Length	32 when CSR[mstatus].SXL == 0 64 when CSR[mstatus].SXL == 1
Privilege Mode	S

C.7.2. Format

This CSR format changes dynamically with XLEN.

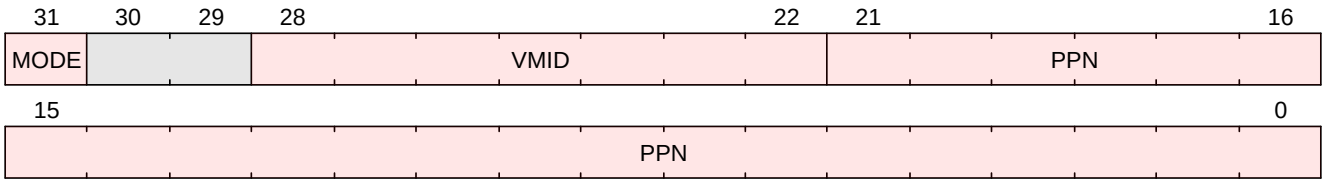


Figure 7. hgatp Format when CSR[mstatus].SXL == 0

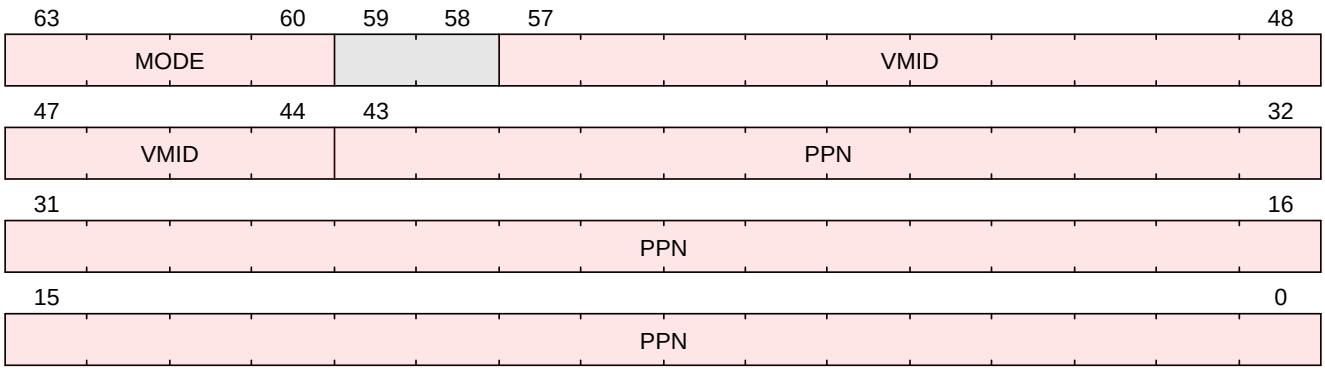


Figure 8. hgatp Format when CSR[mstatus].SXL == 1

C.8. hpmcounter10

User-mode Hardware Performance Counter 7

Alias for M-mode CSR mhpmcounter10.

Privilege mode access is controlled with mcounteren.HPM10 <%- if ext?(:S) -%> , scounteren.HPM10 <%- if ext?(:H) -%> , and hcounteren.HPM10 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM10	scounteren.HPM10	hcounteren.HPM10	hpmcounter10 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM10	scounteren.HPM10	hpmcounter10 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM10	hpmcounter10 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.8.1. Attributes

CSR Address	0xc0a

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.8.2. Format

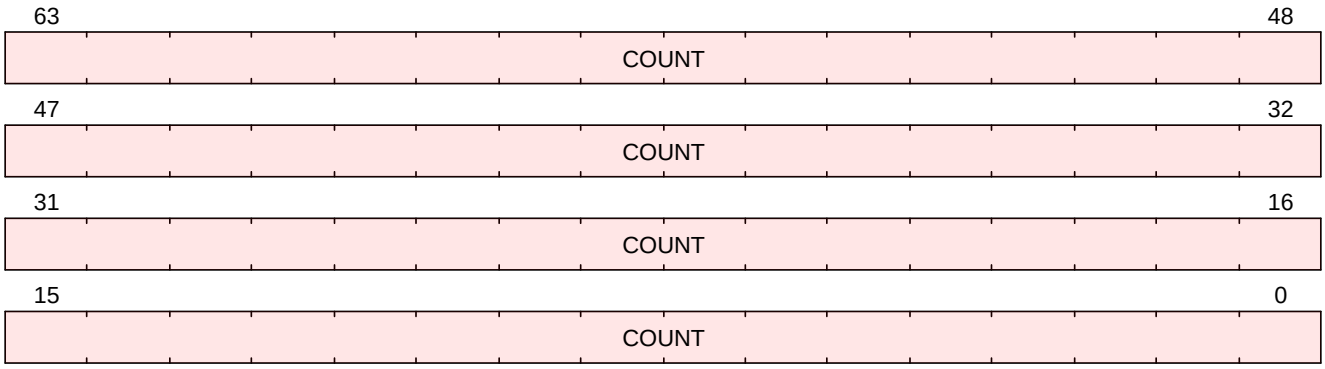


Figure 9. hpmcounter10 format

C.9. hpmcounter10h

User-mode Hardware Performance Counter 7, high half

Alias for M-mode CSR mhpmcounter10h.

Privilege mode access is controlled with mcounteren.HPM10, scounteren.HPM10, and hcounteren.HPM10 as follows:

mcounteren.HPM10	scounteren.HPM10	hcounteren.HPM10	hpmcounter10h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.9.1. Attributes

CSR Address	0xc8a
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.9.2. Format

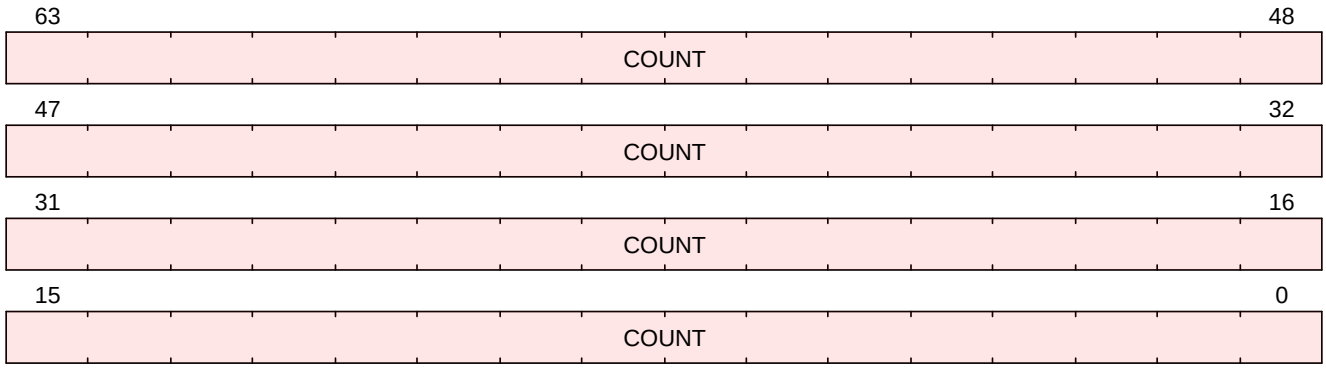


Figure 10. hpmcounter10h format

C.10. hpmcounter11

User-mode Hardware Performance Counter 8

Alias for M-mode CSR mhpmcounter11.

Privilege mode access is controlled with mcounteren.HPM11 <%- if ext?(:S) -%> , scounteren.HPM11 <%- if ext?(:H) -%> , and hcounteren.HPM11 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM11	scounteren.HPM11	hcounteren.HPM11	hpmcounter11 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	`IllegalInstruction`	`IllegalInstruction`	`IllegalInstruction`	`IllegalInstruction`
1	0	0	read-only	`IllegalInstruction`	`VirtualInstruction`	`VirtualInstruction`
1	1	0	read-only	read-only	`VirtualInstruction`	`VirtualInstruction`
1	0	1	read-only	`IllegalInstruction`	read-only	`VirtualInstruction`
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM11	scounteren.HPM11	hpmcounter11 behavior	
		S-mode	U-mode
0	-	`IllegalInstruction`	`IllegalInstruction`
1	0	read-only	`IllegalInstruction`
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM11	hpmcounter11 behavior
	U-mode
0	`IllegalInstruction`
1	read-only

<%- end -%>

C.10.1. Attributes

CSR Address	0xc0b

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.10.2. Format

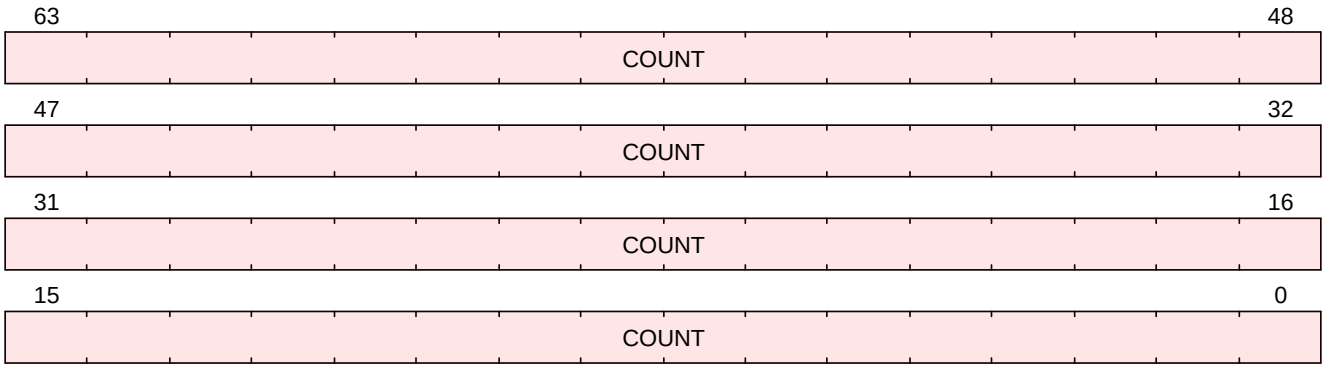


Figure 11. hpmcounter11 format

C.11. hpmcounter11h

User-mode Hardware Performance Counter 8, high half

Alias for M-mode CSR mhpmcounter11h.

Privilege mode access is controlled with mcounteren.HPM11, scounteren.HPM11, and hcounteren.HPM11 as follows:

mcounteren.HPM11	scounteren.HPM11	hcounteren.HPM11	hpmcounter11h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.11.1. Attributes

CSR Address	0xc8b
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.11.2. Format

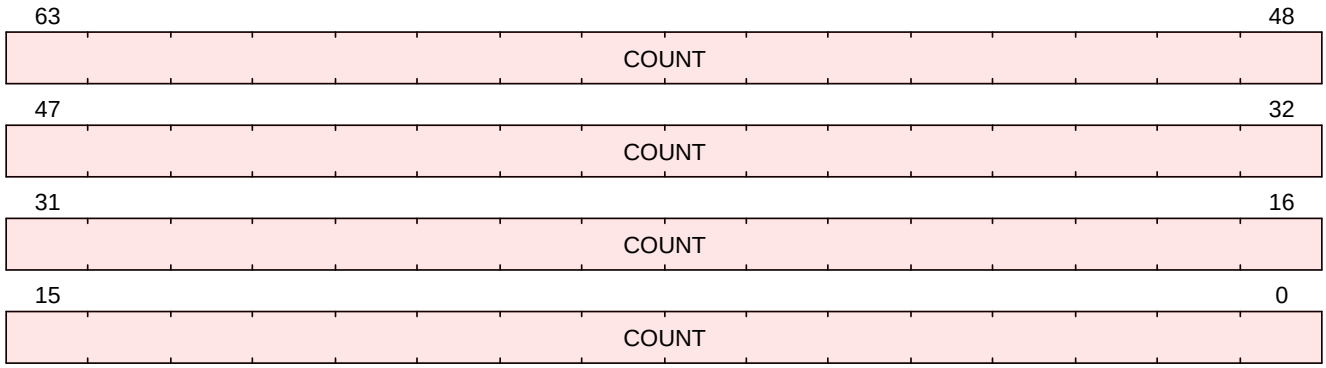


Figure 12. hpmcounter11h format

C.12. hpmcounter12

User-mode Hardware Performance Counter 9

Alias for M-mode CSR mhpmcounter12.

Privilege mode access is controlled with mcounteren.HPM12 <%- if ext?(:S) -%> , scounteren.HPM12 <%- if ext?(:H) -%> , and hcounteren.HPM12 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM12	scounteren.HPM12	hcounteren.HPM12	hpmcounter12 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM12	scounteren.HPM12	hpmcounter12 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM12	hpmcounter12 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.12.1. Attributes

CSR Address	0xc0c

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.12.2. Format

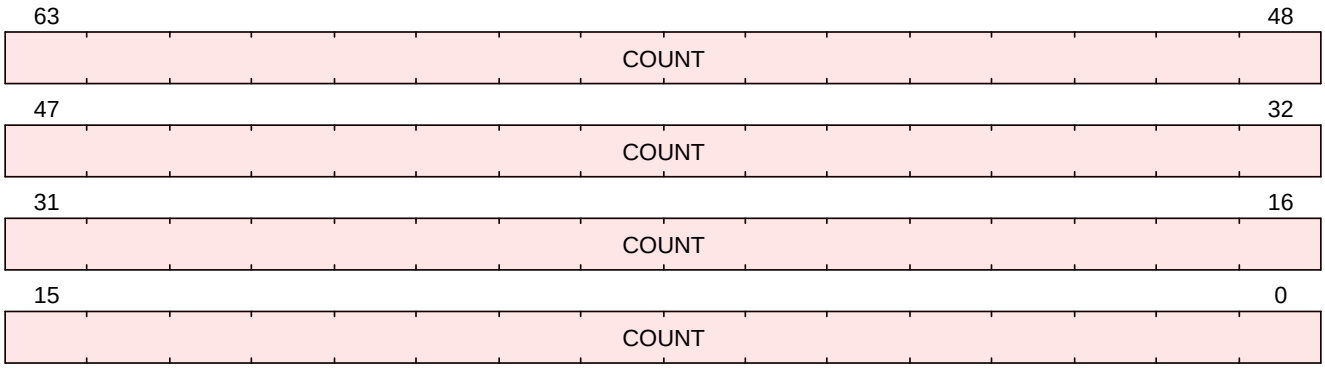


Figure 13. hpmcounter12 format

C.13. hpmcounter12h

User-mode Hardware Performance Counter 9, high half

Alias for M-mode CSR mhpmcounter12h.

Privilege mode access is controlled with mcounteren.HPM12, scounteren.HPM12, and hcounteren.HPM12 as follows:

mcounteren.HPM12	scounteren.HPM12	hcounteren.HPM12	hpmcounter12h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.13.1. Attributes

CSR Address	0xc8c
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.13.2. Format

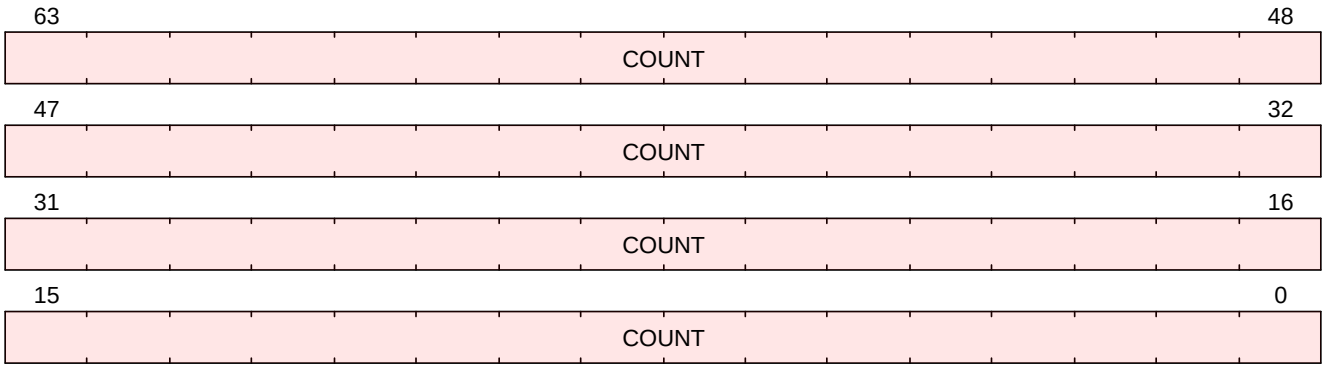


Figure 14. hpmcounter12h format

C.14. hpmcounter13

User-mode Hardware Performance Counter 10

Alias for M-mode CSR mhpmcounter13.

Privilege mode access is controlled with mcounteren.HPM13 <%- if ext?(:S) -%> , scounteren.HPM13 <%- if ext?(:H) -%> , and hcounteren.HPM13 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM13	scounteren.HPM13	hcounteren.HPM13	hpmcounter13 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM13	scounteren.HPM13	hpmcounter13 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM13	hpmcounter13 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.14.1. Attributes

CSR Address	0xc0d

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.14.2. Format

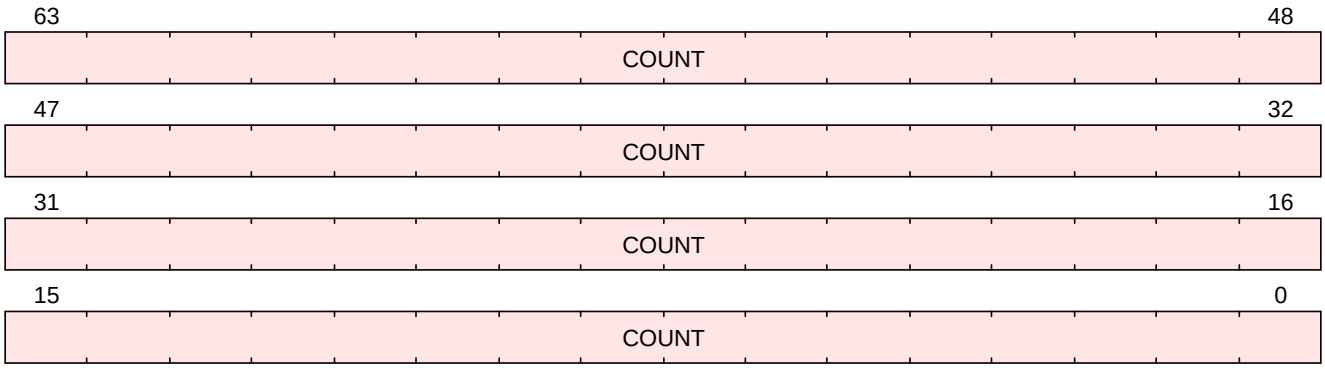


Figure 15. hpmcounter13 format

C.15. hpmcounter13h

User-mode Hardware Performance Counter 10, high half

Alias for M-mode CSR mhpmcounter13h.

Privilege mode access is controlled with mcounteren.HPM13, scounteren.HPM13, and hcounteren.HPM13 as follows:

mcounteren.HPM13	scounteren.HPM13	hcounteren.HPM13	hpmcounter13h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.15.1. Attributes

CSR Address	0xc8d
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.15.2. Format

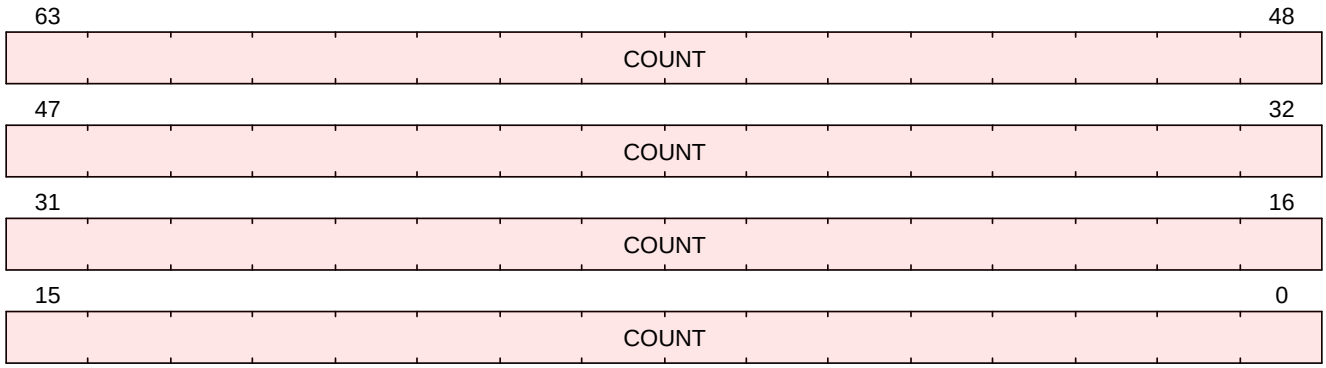


Figure 16. hpmcounter13h format

C.16. hpmcounter14

User-mode Hardware Performance Counter 11

Alias for M-mode CSR mhpmcounter14.

Privilege mode access is controlled with mcounteren.HPM14 <%- if ext?(:S) -%> , scounteren.HPM14 <%- if ext?(:H) -%> , and hcounteren.HPM14 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM14	scounteren.HPM14	hcounteren.HPM14	hpmcounter14 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM14	scounteren.HPM14	hpmcounter14 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM14	hpmcounter14 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.16.1. Attributes

CSR Address	0xc0e

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.16.2. Format

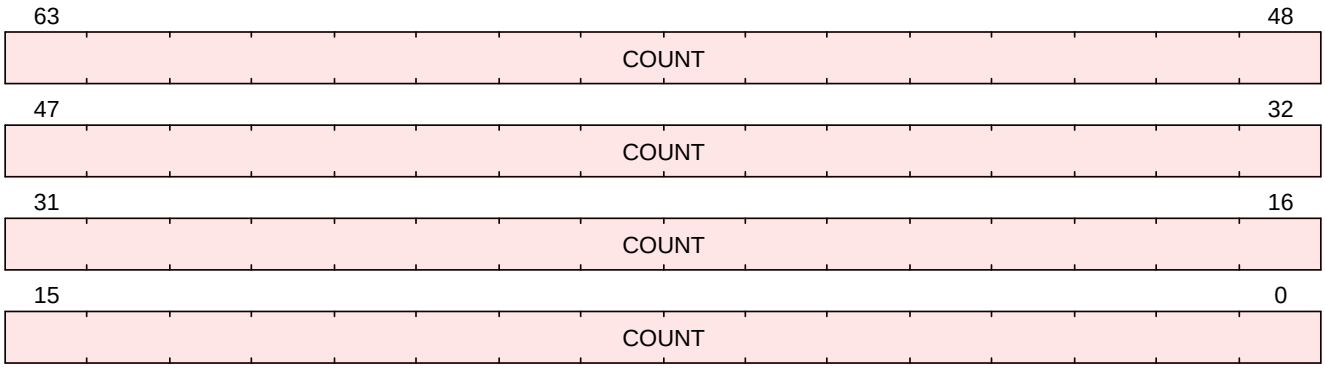


Figure 17. hpmcounter14 format

C.17. hpmcounter14h

User-mode Hardware Performance Counter 11, high half

Alias for M-mode CSR mhpmcounter14h.

Privilege mode access is controlled with mcounteren.HPM14, scounteren.HPM14, and hcounteren.HPM14 as follows:

mcounteren.HPM14	scounteren.HPM14	hcounteren.HPM14	hpmcounter14h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.17.1. Attributes

CSR Address	0xc8e
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.17.2. Format

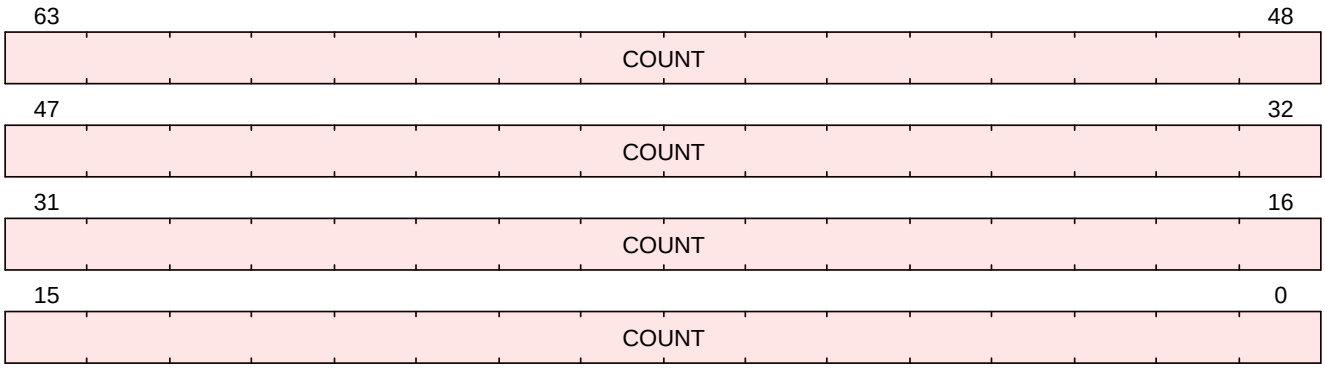


Figure 18. hpmcounter14h format

C.18. hpmcounter15

User-mode Hardware Performance Counter 12

Alias for M-mode CSR mhpmcounter15.

Privilege mode access is controlled with mcounteren.HPM15 <%- if ext?(:S) -%> , scounteren.HPM15 <%- if ext?(:H) -%> , and hcounteren.HPM15 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM15	scounteren.HPM15	hcounteren.HPM15	hpmcounter15 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	`IllegalInstruction`	`IllegalInstruction`	`IllegalInstruction`	`IllegalInstruction`
1	0	0	read-only	`IllegalInstruction`	`VirtualInstruction`	`VirtualInstruction`
1	1	0	read-only	read-only	`VirtualInstruction`	`VirtualInstruction`
1	0	1	read-only	`IllegalInstruction`	read-only	`VirtualInstruction`
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM15	scounteren.HPM15	hpmcounter15 behavior	
		S-mode	U-mode
0	-	`IllegalInstruction`	`IllegalInstruction`
1	0	read-only	`IllegalInstruction`
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM15	hpmcounter15 behavior
	U-mode
0	`IllegalInstruction`
1	read-only

<%- end -%>

C.18.1. Attributes

CSR Address	0xc0f

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.18.2. Format

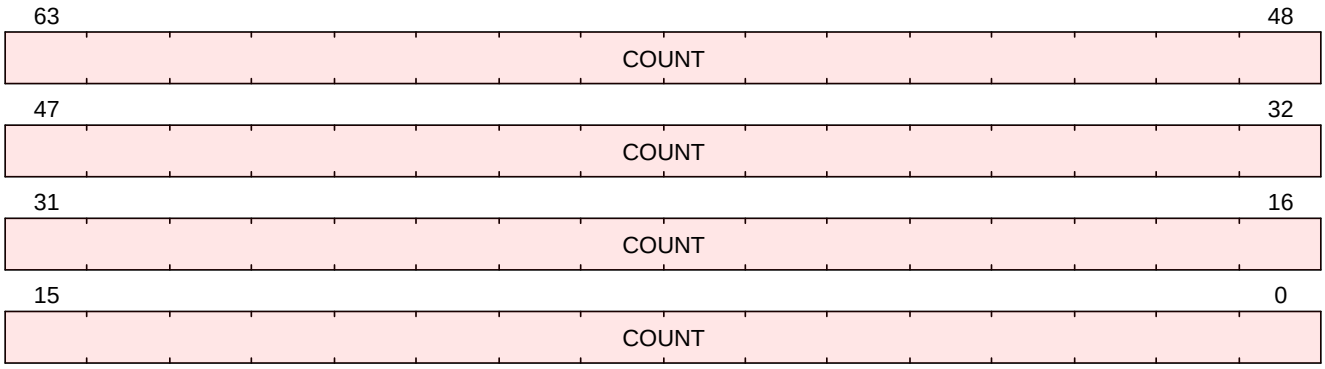


Figure 19. hpmcounter15 format

C.19. hpmcounter15h

User-mode Hardware Performance Counter 12, high half

Alias for M-mode CSR mhpmcounter15h.

Privilege mode access is controlled with mcounteren.HPM15, scounteren.HPM15, and hcounteren.HPM15 as follows:

mcounteren.HPM15	scounteren.HPM15	hcounteren.HPM15	hpmcounter15h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.19.1. Attributes

CSR Address	0xc8f
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.19.2. Format

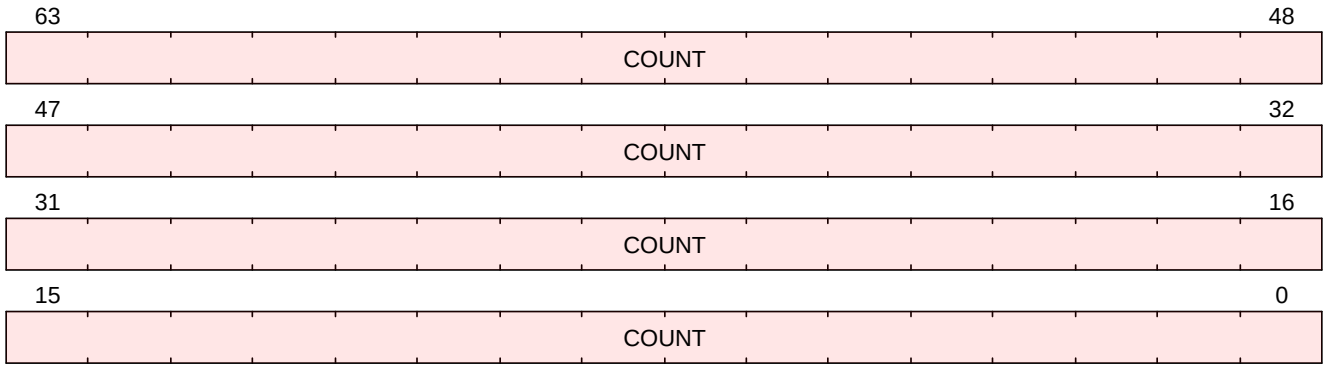


Figure 20. hpmcounter15h format

C.20. hpmcounter16

User-mode Hardware Performance Counter 13

Alias for M-mode CSR mhpmcounter16.

Privilege mode access is controlled with mcounteren.HPM16 <%- if ext?(:S) -%> , scounteren.HPM16 <%- if ext?(:H) -%> , and hcounteren.HPM16 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM16	scounteren.HPM16	hcounteren.HPM16	hpmcounter16 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM16	scounteren.HPM16	hpmcounter16 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM16	hpmcounter16 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.20.1. Attributes

CSR Address	0xc10

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.20.2. Format

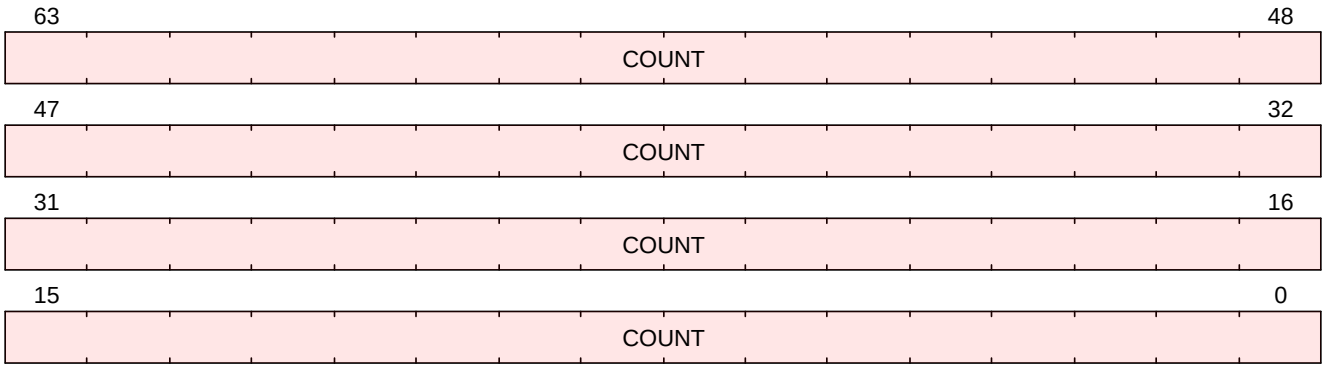


Figure 21. hpmcounter16 format

C.21. hpmcounter16h

User-mode Hardware Performance Counter 13, high half

Alias for M-mode CSR mhpmcounter16h.

Privilege mode access is controlled with mcounteren.HPM16, scounteren.HPM16, and hcounteren.HPM16 as follows:

mcounteren.HPM16	scounteren.HPM16	hcounteren.HPM16	hpmcounter16h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.21.1. Attributes

CSR Address	0xc90
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.21.2. Format

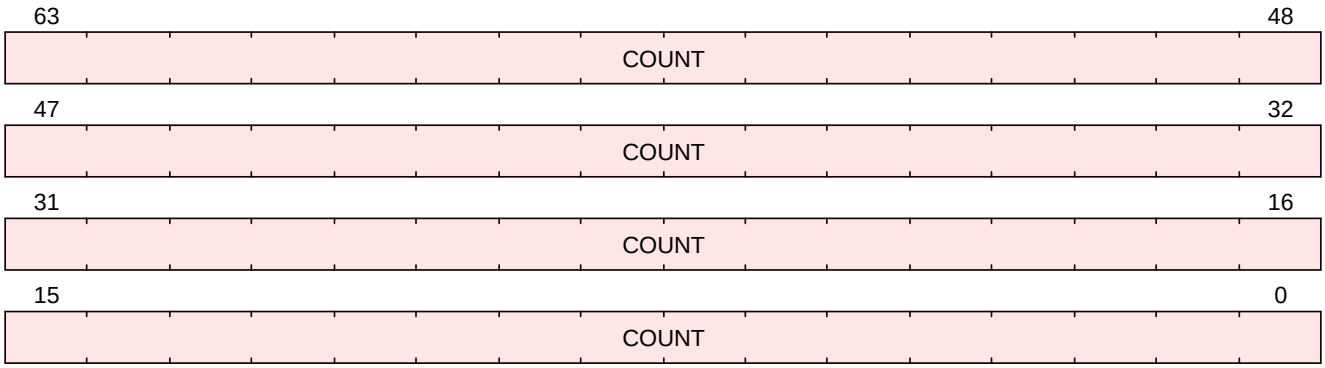


Figure 22. hpmcounter16h format

C.22. hpmcounter17

User-mode Hardware Performance Counter 14

Alias for M-mode CSR mhpmcounter17.

Privilege mode access is controlled with mcounteren.HPM17 <%- if ext?(:S) -%> , scounteren.HPM17 <%- if ext?(:H) -%> , and hcounteren.HPM17 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM17	scounteren.HPM17	hcounteren.HPM17	hpmcounter17 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM17	scounteren.HPM17	hpmcounter17 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM17	hpmcounter17 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.22.1. Attributes

CSR Address	0xc11

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.22.2. Format

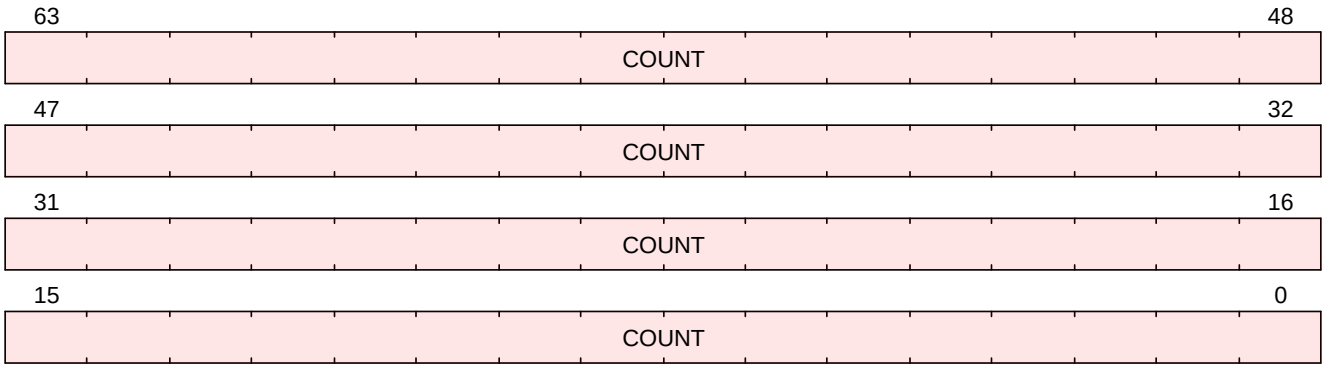


Figure 23. hpmcounter17 format

C.23. hpmcounter17h

User-mode Hardware Performance Counter 14, high half

Alias for M-mode CSR mhpmcounter17h.

Privilege mode access is controlled with mcounteren.HPM17, scounteren.HPM17, and hcounteren.HPM17 as follows:

mcounteren.HPM17	scounteren.HPM17	hcounteren.HPM17	hpmcounter17h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.23.1. Attributes

CSR Address	0xc91
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.23.2. Format

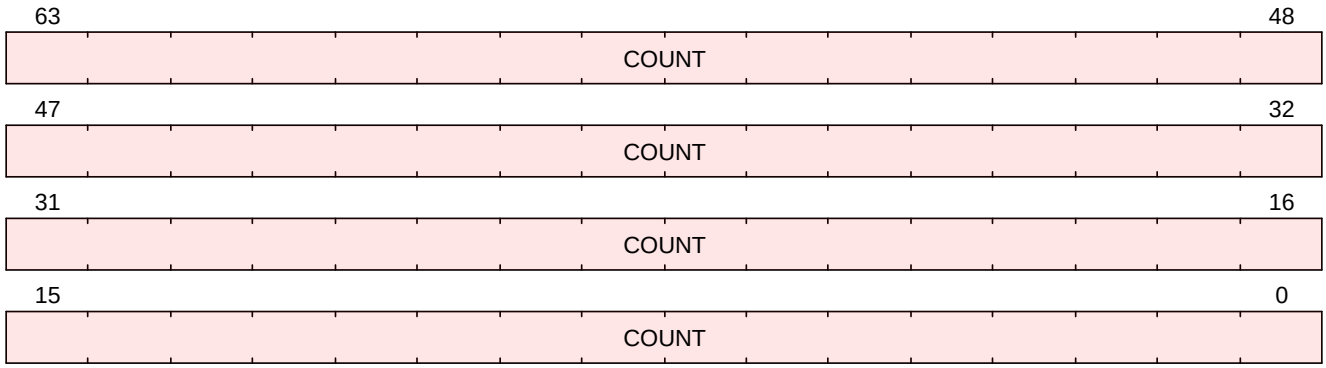


Figure 24. hpmcounter17h format

C.24. hpmcounter18

User-mode Hardware Performance Counter 15

Alias for M-mode CSR mhpmcounter18.

Privilege mode access is controlled with mcounteren.HPM18 <%- if ext?(:S) -%> , scounteren.HPM18 <%- if ext?(:H) -%> , and hcounteren.HPM18 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM18	scounteren.HPM18	hcounteren.HPM18	hpmcounter18 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM18	scounteren.HPM18	hpmcounter18 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM18	hpmcounter18 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.24.1. Attributes

CSR Address	0xc12

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.24.2. Format

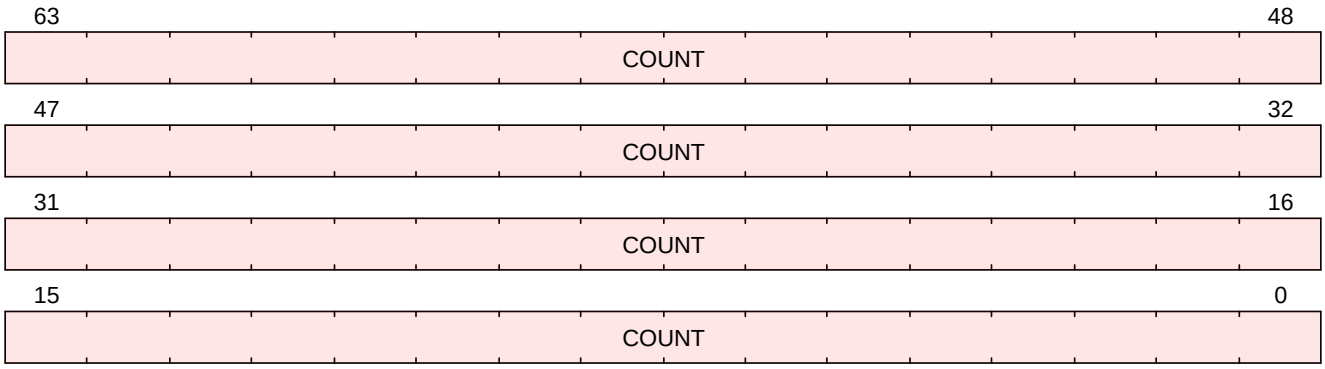


Figure 25. hpmcounter18 format

C.25. hpmcounter18h

User-mode Hardware Performance Counter 15, high half

Alias for M-mode CSR mhpmcounter18h.

Privilege mode access is controlled with mcounteren.HPM18, scounteren.HPM18, and hcounteren.HPM18 as follows:

mcounteren.HPM18	scounteren.HPM18	hcounteren.HPM18	hpmcounter18h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.25.1. Attributes

CSR Address	0xc92
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.25.2. Format

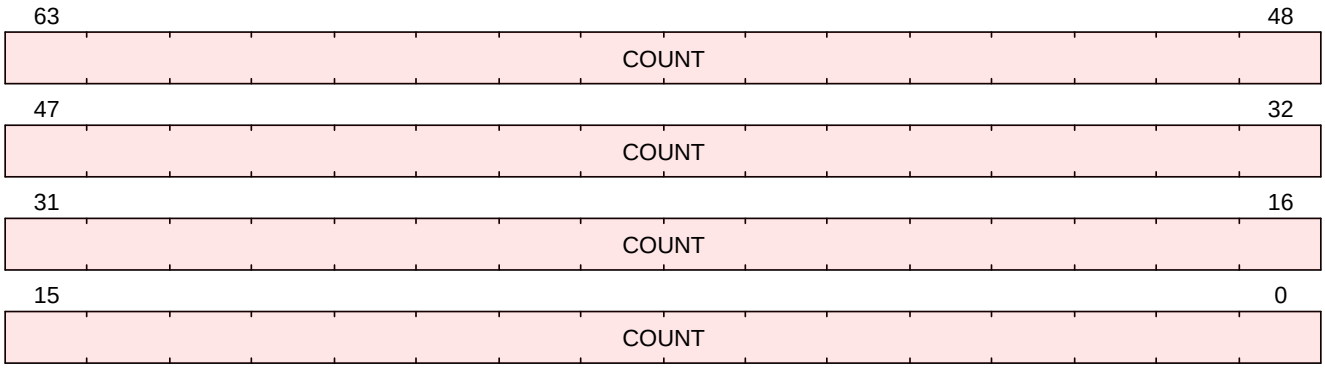


Figure 26. hpmcounter18h format

C.26. hpmcounter19

User-mode Hardware Performance Counter 16

Alias for M-mode CSR mhpmcounter19.

Privilege mode access is controlled with mcounteren.HPM19 <%- if ext?(:S) -%> , scounteren.HPM19 <%- if ext?(:H) -%> , and hcounteren.HPM19 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM19	scounteren.HPM19	hcounteren.HPM19	hpmcounter19 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	`IllegalInstruction`	`IllegalInstruction`	`IllegalInstruction`	`IllegalInstruction`
1	0	0	read-only	`IllegalInstruction`	`VirtualInstruction`	`VirtualInstruction`
1	1	0	read-only	read-only	`VirtualInstruction`	`VirtualInstruction`
1	0	1	read-only	`IllegalInstruction`	read-only	`VirtualInstruction`
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM19	scounteren.HPM19	hpmcounter19 behavior	
		S-mode	U-mode
0	-	`IllegalInstruction`	`IllegalInstruction`
1	0	read-only	`IllegalInstruction`
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM19	hpmcounter19 behavior
	U-mode
0	`IllegalInstruction`
1	read-only

<%- end -%>

C.26.1. Attributes

CSR Address	0xc13

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.26.2. Format

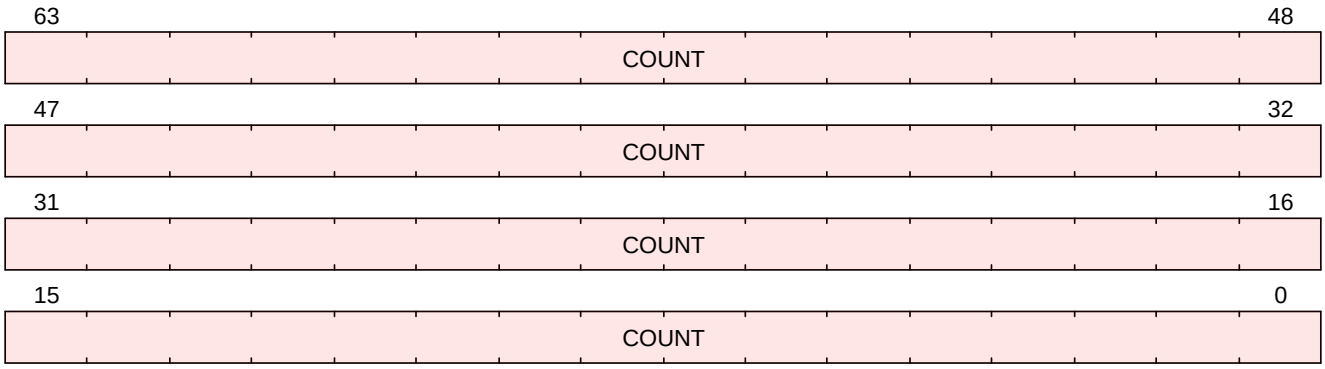


Figure 27. hpmcounter19 format

C.27. hpmcounter19h

User-mode Hardware Performance Counter 16, high half

Alias for M-mode CSR mhpmcounter19h.

Privilege mode access is controlled with mcounteren.HPM19, scounteren.HPM19, and hcounteren.HPM19 as follows:

mcounteren.HPM19	scounteren.HPM19	hcounteren.HPM19	hpmcounter19h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.27.1. Attributes

CSR Address	0xc93
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.27.2. Format

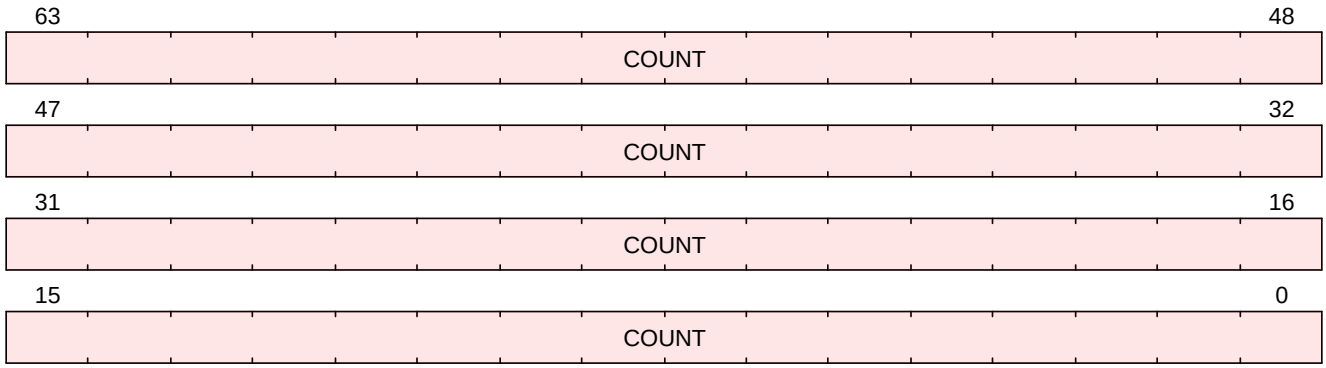


Figure 28. hpmcounter19h format

C.28. hpmcounter20

User-mode Hardware Performance Counter 17

Alias for M-mode CSR mhpmcounter20.

Privilege mode access is controlled with mcounteren.HPM20 <%- if ext?(:S) -%> , scounteren.HPM20 <%- if ext?(:H) -%> , and hcounteren.HPM20 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM20	scounteren.HPM20	hcounteren.HPM20	hpmcounter20 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM20	scounteren.HPM20	hpmcounter20 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM20	hpmcounter20 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.28.1. Attributes

CSR Address	0xc14

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.28.2. Format

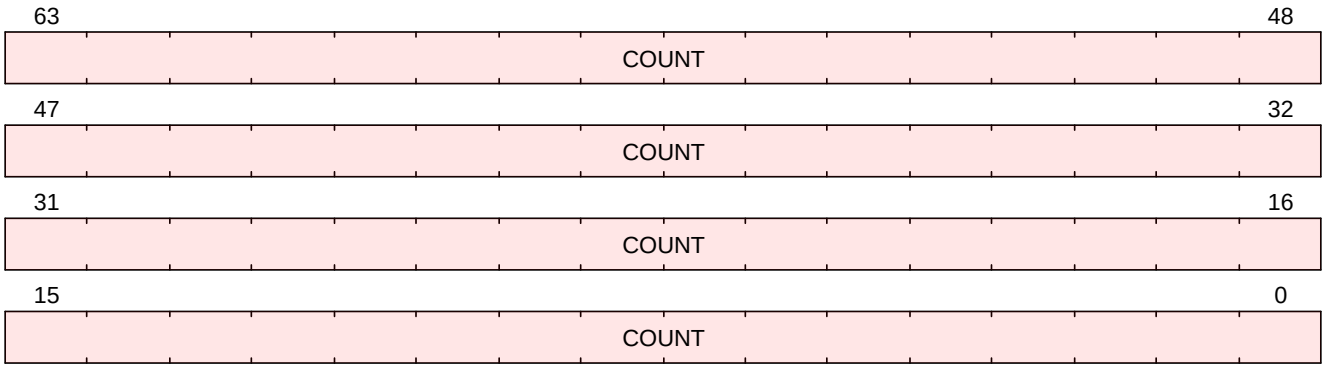


Figure 29. hpmcounter20 format

C.29. hpmcounter20h

User-mode Hardware Performance Counter 17, high half

Alias for M-mode CSR mhpmcounter20h.

Privilege mode access is controlled with mcounteren.HPM20, scounteren.HPM20, and hcounteren.HPM20 as follows:

mcounteren.HPM20	scounteren.HPM20	hcounteren.HPM20	hpmcounter20h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.29.1. Attributes

CSR Address	0xc94
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.29.2. Format

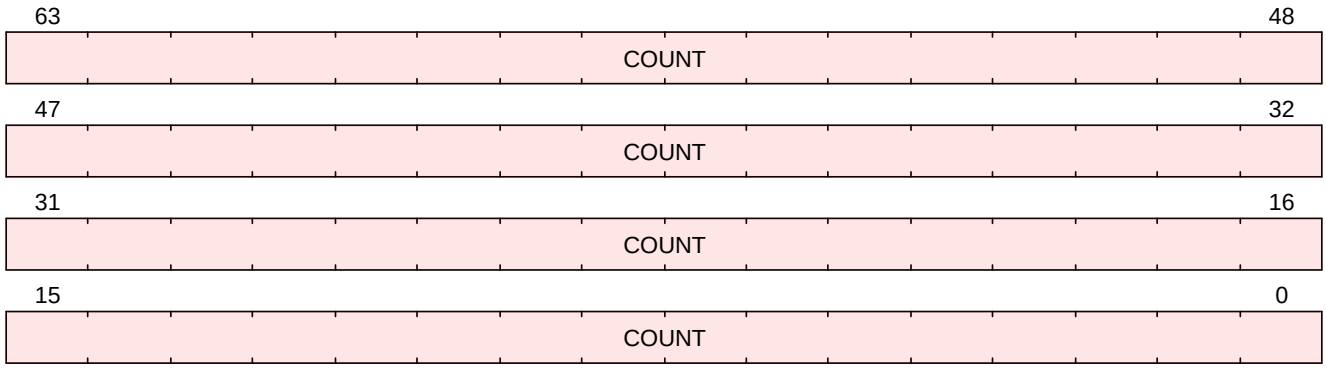


Figure 30. hpmcounter20h format

C.30. hpmcounter21

User-mode Hardware Performance Counter 18

Alias for M-mode CSR mhpmcounter21.

Privilege mode access is controlled with mcounteren.HPM21 <%- if ext?(:S) -%> , scounteren.HPM21 <%- if ext?(:H) -%> , and hcounteren.HPM21 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM21	scounteren.HPM21	hcounteren.HPM21	hpmcounter21 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM21	scounteren.HPM21	hpmcounter21 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM21	hpmcounter21 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.30.1. Attributes

CSR Address	0xc15

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.30.2. Format

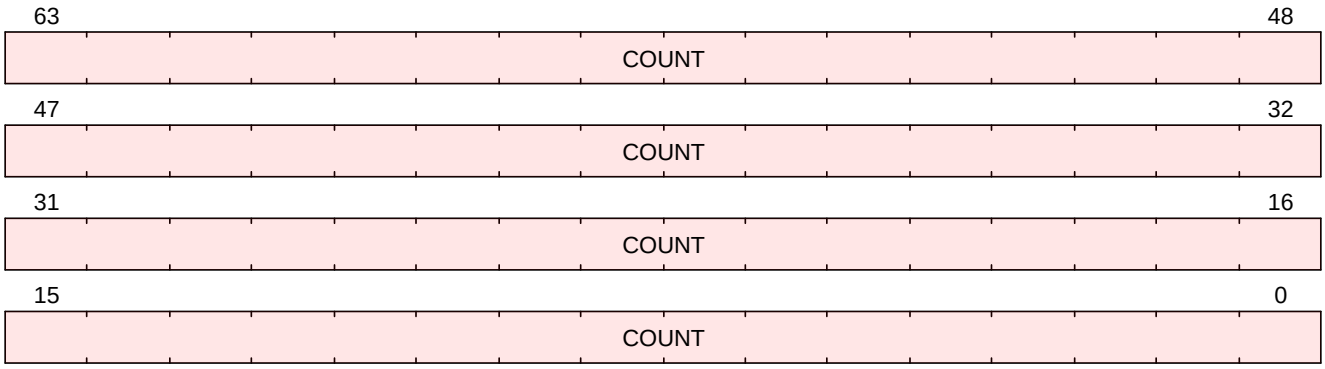


Figure 31. hpmcounter21 format

C.31. hpmcounter21h

User-mode Hardware Performance Counter 18, high half

Alias for M-mode CSR mhpmcounter21h.

Privilege mode access is controlled with mcounteren.HPM21, scounteren.HPM21, and hcounteren.HPM21 as follows:

mcounteren.HPM21	scounteren.HPM21	hcounteren.HPM21	hpmcounter21h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.31.1. Attributes

CSR Address	0xc95
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.31.2. Format

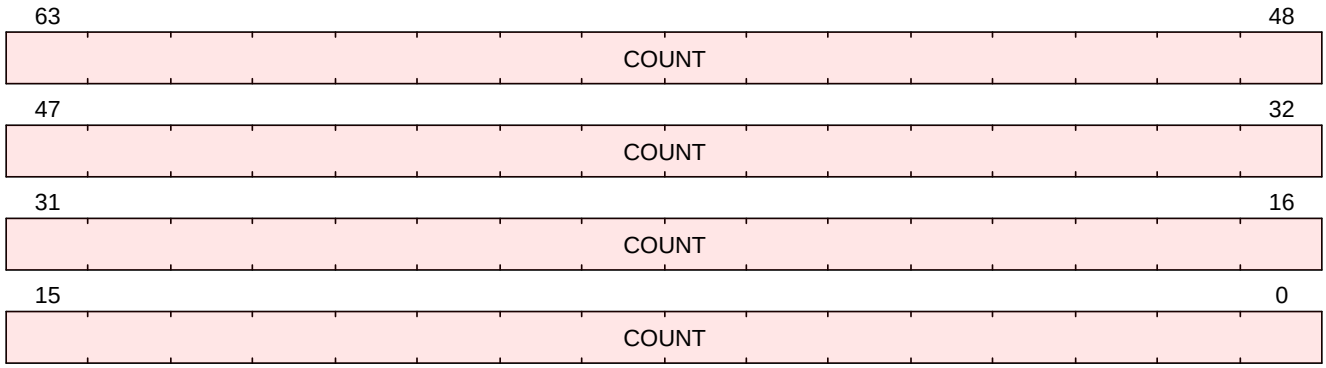


Figure 32. hpmcounter21h format

C.32. hpmcounter22

User-mode Hardware Performance Counter 19

Alias for M-mode CSR mhpmcounter22.

Privilege mode access is controlled with mcounteren.HPM22 <%- if ext?(:S) -%> , scounteren.HPM22 <%- if ext?(:H) -%> , and hcounteren.HPM22 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM22	scounteren.HPM22	hcounteren.HPM22	hpmcounter22 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM22	scounteren.HPM22	hpmcounter22 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM22	hpmcounter22 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.32.1. Attributes

CSR Address	0xc16

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.32.2. Format

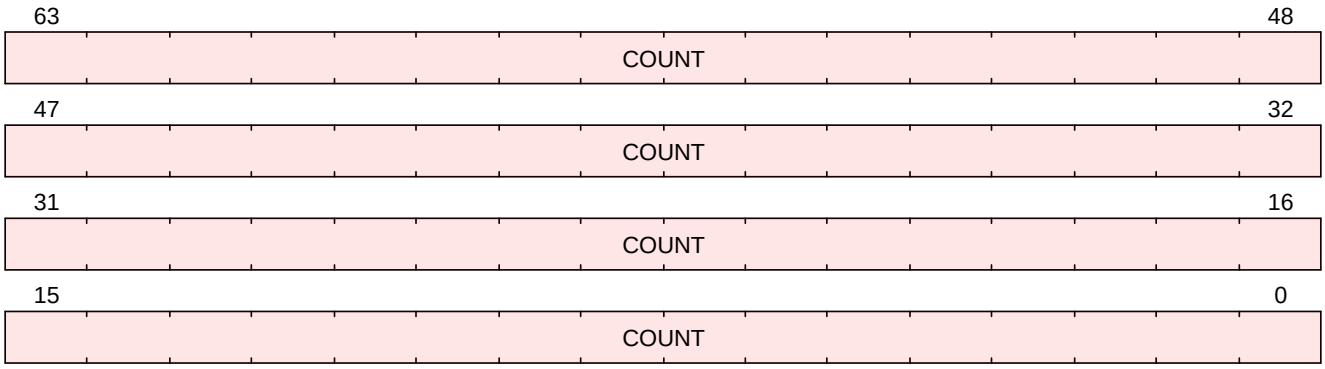


Figure 33. hpmcounter22 format

C.33. hpmcounter22h

User-mode Hardware Performance Counter 19, high half

Alias for M-mode CSR mhpmcounter22h.

Privilege mode access is controlled with mcounteren.HPM22, scounteren.HPM22, and hcounteren.HPM22 as follows:

mcounteren.HPM22	scounteren.HPM22	hcounteren.HPM22	hpmcounter22h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.33.1. Attributes

CSR Address	0xc96
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.33.2. Format

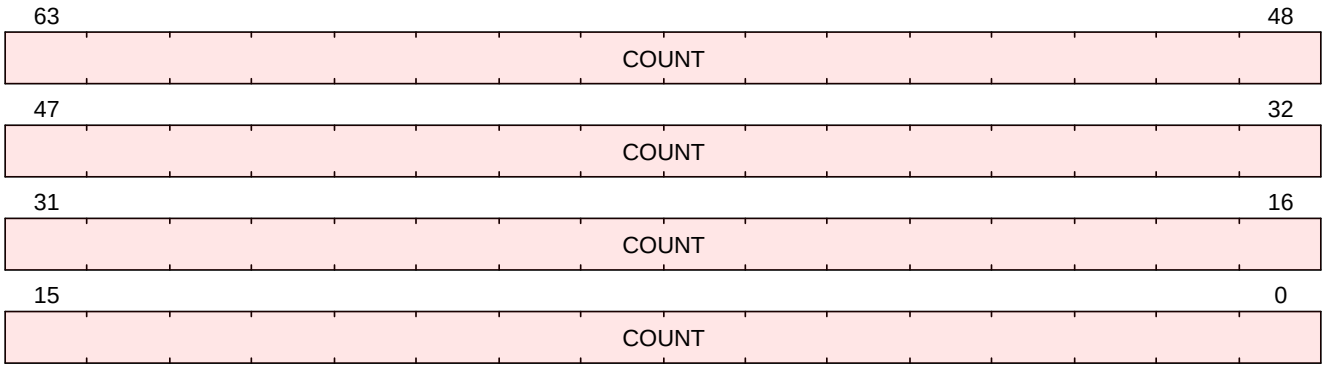


Figure 34. hpmcounter22h format

C.34. hpmcounter23

User-mode Hardware Performance Counter 20

Alias for M-mode CSR mhpmcounter23.

Privilege mode access is controlled with mcounteren.HPM23 <%- if ext?(:S) -%> , scounteren.HPM23 <%- if ext?(:H) -%> , and hcounteren.HPM23 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM23	scounteren.HPM23	hcounteren.HPM23	hpmcounter23 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM23	scounteren.HPM23	hpmcounter23 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM23	hpmcounter23 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.34.1. Attributes

CSR Address	0xc17

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.34.2. Format

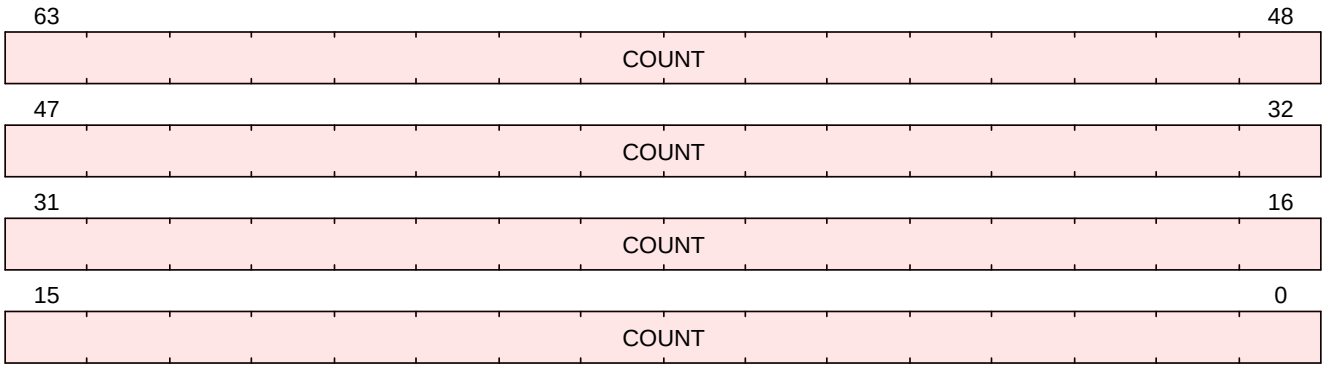


Figure 35. hpmcounter23 format

C.35. hpmcounter23h

User-mode Hardware Performance Counter 20, high half

Alias for M-mode CSR mhpmcounter23h.

Privilege mode access is controlled with mcounteren.HPM23, scounteren.HPM23, and hcounteren.HPM23 as follows:

mcounteren.HPM23	scounteren.HPM23	hcounteren.HPM23	hpmcounter23h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.35.1. Attributes

CSR Address	0xc97
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.35.2. Format

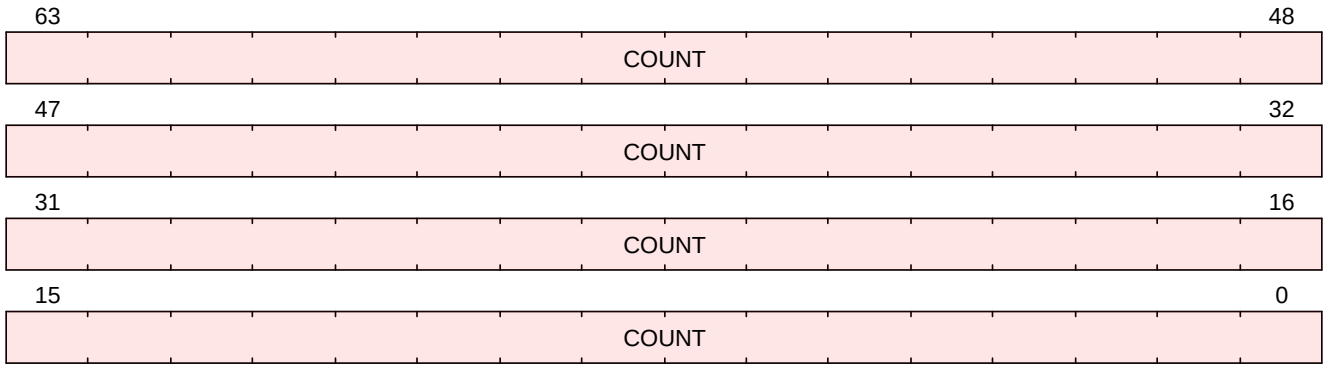


Figure 36. hpmcounter23h format

C.36. hpmcounter24

User-mode Hardware Performance Counter 21

Alias for M-mode CSR mhpmcounter24.

Privilege mode access is controlled with mcounteren.HPM24 <%- if ext?(:S) -%> , scounteren.HPM24 <%- if ext?(:H) -%> , and hcounteren.HPM24 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM24	scounteren.HPM24	hcounteren.HPM24	hpmcounter24 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM24	scounteren.HPM24	hpmcounter24 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM24	hpmcounter24 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.36.1. Attributes

CSR Address	0xc18

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.36.2. Format

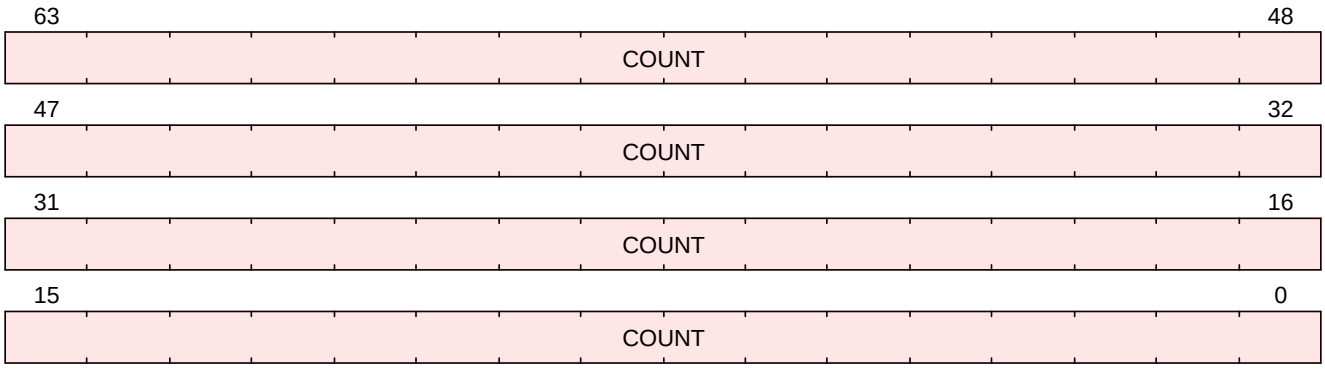


Figure 37. hpmcounter24 format

C.37. hpmcounter24h

User-mode Hardware Performance Counter 21, high half

Alias for M-mode CSR mhpmcounter24h.

Privilege mode access is controlled with mcounteren.HPM24, scounteren.HPM24, and hcounteren.HPM24 as follows:

mcounteren.HPM24	scounteren.HPM24	hcounteren.HPM24	hpmcounter24h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.37.1. Attributes

CSR Address	0xc98
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.37.2. Format

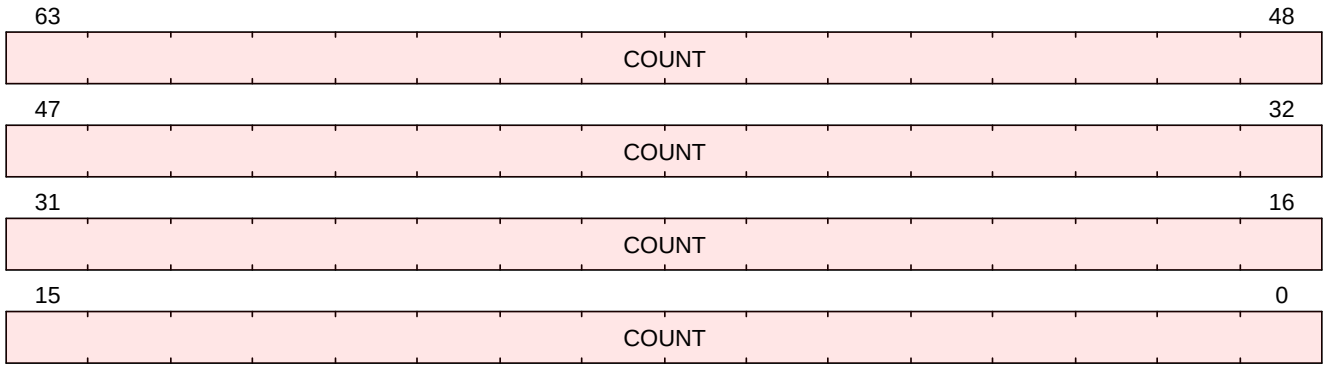


Figure 38. hpmcounter24h format

C.38. hpmcounter25

User-mode Hardware Performance Counter 22

Alias for M-mode CSR mhpmcounter25.

Privilege mode access is controlled with mcounteren.HPM25 <%- if ext?(:S) -%> , scounteren.HPM25 <%- if ext?(:H) -%> , and hcounteren.HPM25 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM25	scounteren.HPM25	hcounteren.HPM25	hpmcounter25 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	`IllegalInstruction`	`IllegalInstruction`	`IllegalInstruction`	`IllegalInstruction`
1	0	0	read-only	`IllegalInstruction`	`VirtualInstruction`	`VirtualInstruction`
1	1	0	read-only	read-only	`VirtualInstruction`	`VirtualInstruction`
1	0	1	read-only	`IllegalInstruction`	read-only	`VirtualInstruction`
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM25	scounteren.HPM25	hpmcounter25 behavior	
		S-mode	U-mode
0	-	`IllegalInstruction`	`IllegalInstruction`
1	0	read-only	`IllegalInstruction`
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM25	hpmcounter25 behavior
	U-mode
0	`IllegalInstruction`
1	read-only

<%- end -%>

C.38.1. Attributes

CSR Address	0xc19

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.38.2. Format

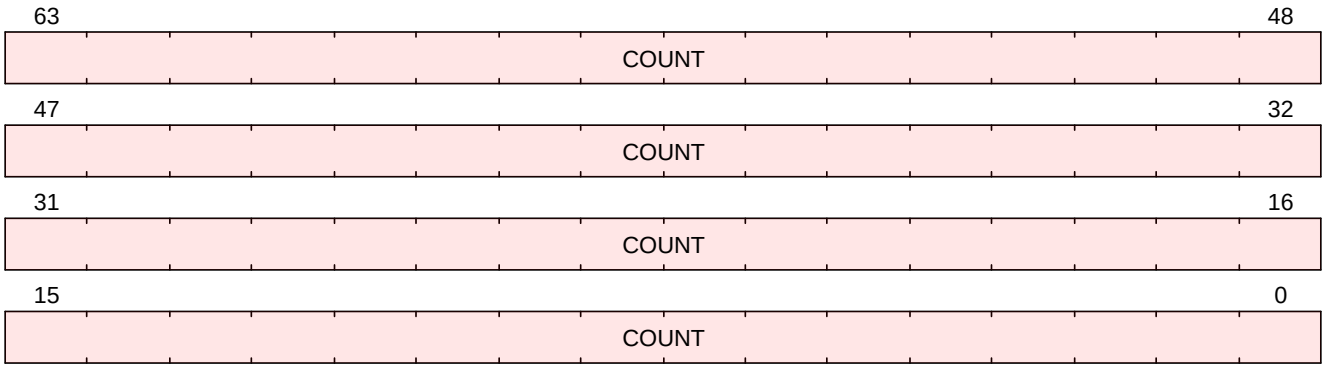


Figure 39. hpmcounter25 format

C.39. hpmcounter25h

User-mode Hardware Performance Counter 22, high half

Alias for M-mode CSR mhpmcounter25h.

Privilege mode access is controlled with mcounteren.HPM25, scounteren.HPM25, and hcounteren.HPM25 as follows:

mcounteren.HPM25	scounteren.HPM25	hcounteren.HPM25	hpmcounter25h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.39.1. Attributes

CSR Address	0xc99
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.39.2. Format

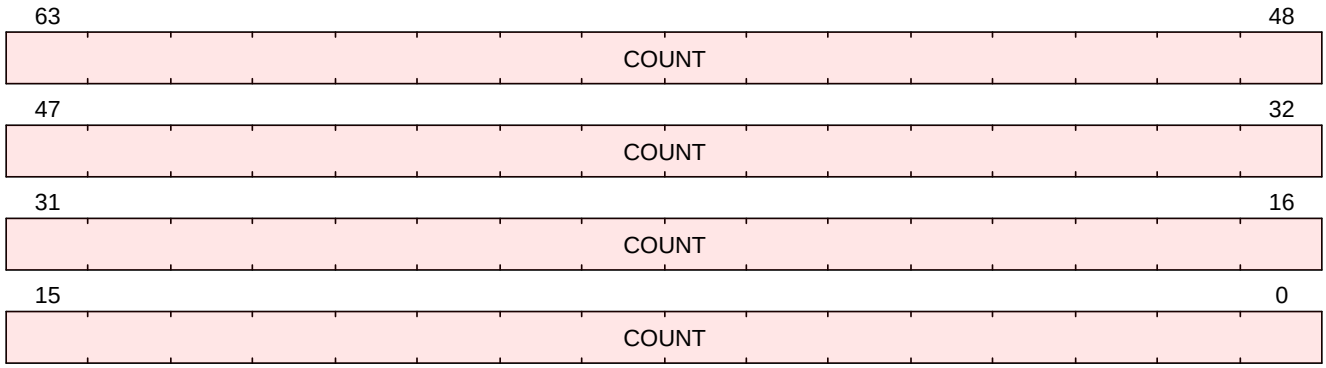


Figure 40. hpmcounter25h format

C.40. hpmcounter26

User-mode Hardware Performance Counter 23

Alias for M-mode CSR mhpmcounter26.

Privilege mode access is controlled with mcounteren.HPM26 <%- if ext?(:S) -%> , scounteren.HPM26 <%- if ext?(:H) -%> , and hcounteren.HPM26 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM26	scounteren.HPM26	hcounteren.HPM26	hpmcounter26 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM26	scounteren.HPM26	hpmcounter26 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM26	hpmcounter26 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.40.1. Attributes

CSR Address	0xc1a

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.40.2. Format

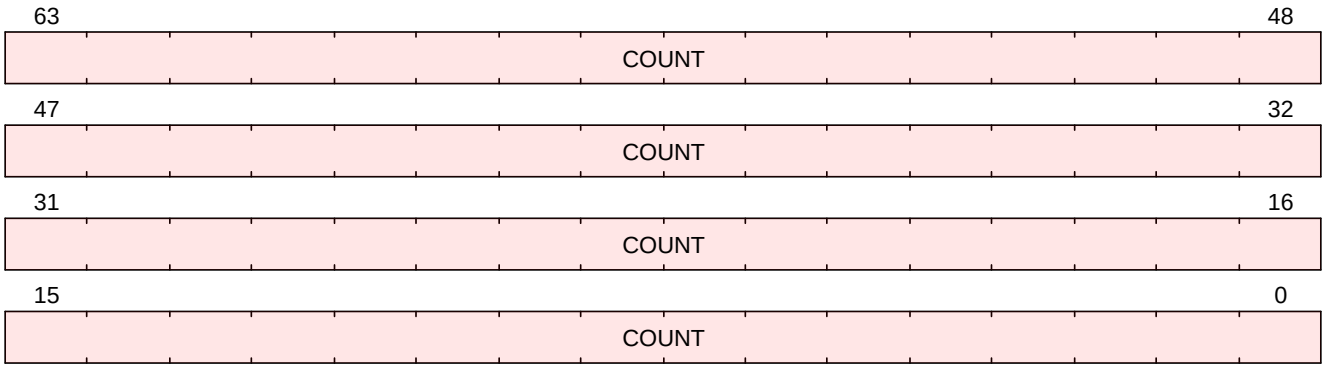


Figure 41. hpmcounter26 format

C.41. hpmcounter26h

User-mode Hardware Performance Counter 23, high half

Alias for M-mode CSR mhpmcounter26h.

Privilege mode access is controlled with mcounteren.HPM26, scounteren.HPM26, and hcounteren.HPM26 as follows:

mcounteren.HPM26	scounteren.HPM26	hcounteren.HPM26	hpmcounter26h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.41.1. Attributes

CSR Address	0xc9a
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.41.2. Format

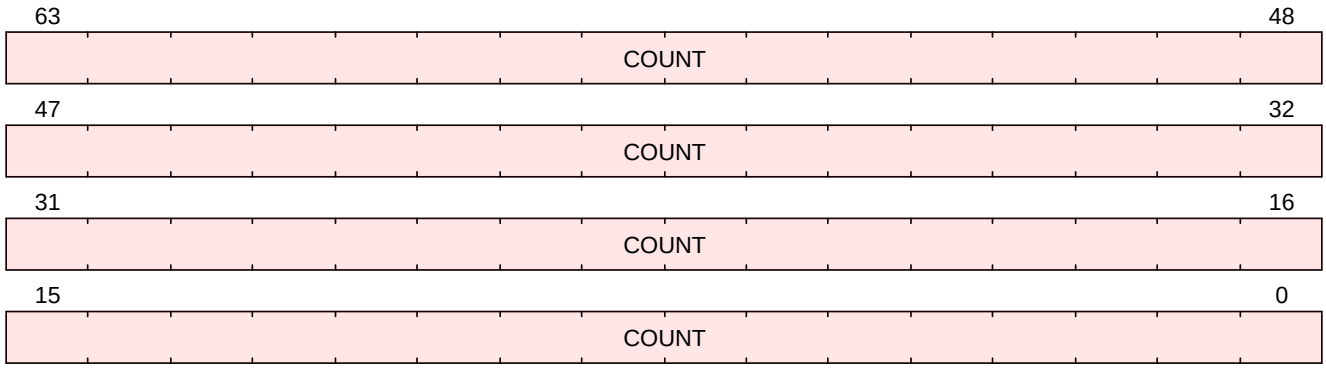


Figure 42. hpmcounter26h format

C.42. hpmcounter27

User-mode Hardware Performance Counter 24

Alias for M-mode CSR mhpmcounter27.

Privilege mode access is controlled with mcounteren.HPM27 <%- if ext?(:S) -%> , scounteren.HPM27 <%- if ext?(:H) -%> , and hcounteren.HPM27 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM27	scounteren.HPM27	hcounteren.HPM27	hpmcounter27 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	`IllegalInstruction`	`IllegalInstruction`	`IllegalInstruction`	`IllegalInstruction`
1	0	0	read-only	`IllegalInstruction`	`VirtualInstruction`	`VirtualInstruction`
1	1	0	read-only	read-only	`VirtualInstruction`	`VirtualInstruction`
1	0	1	read-only	`IllegalInstruction`	read-only	`VirtualInstruction`
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM27	scounteren.HPM27	hpmcounter27 behavior	
		S-mode	U-mode
0	-	`IllegalInstruction`	`IllegalInstruction`
1	0	read-only	`IllegalInstruction`
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM27	hpmcounter27 behavior
	U-mode
0	`IllegalInstruction`
1	read-only

<%- end -%>

C.42.1. Attributes

CSR Address	0xc1b

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.42.2. Format

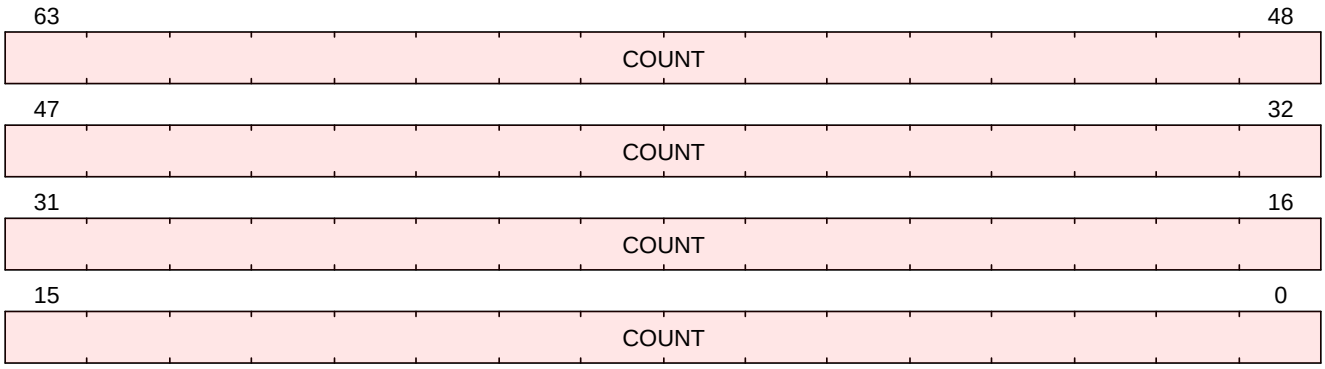


Figure 43. hpmcounter27 format

C.43. hpmcounter27h

User-mode Hardware Performance Counter 24, high half

Alias for M-mode CSR mhpmcounter27h.

Privilege mode access is controlled with mcounteren.HPM27, scounteren.HPM27, and hcounteren.HPM27 as follows:

mcounteren.HPM27	scounteren.HPM27	hcounteren.HPM27	hpmcounter27h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.43.1. Attributes

CSR Address	0xc9b
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.43.2. Format

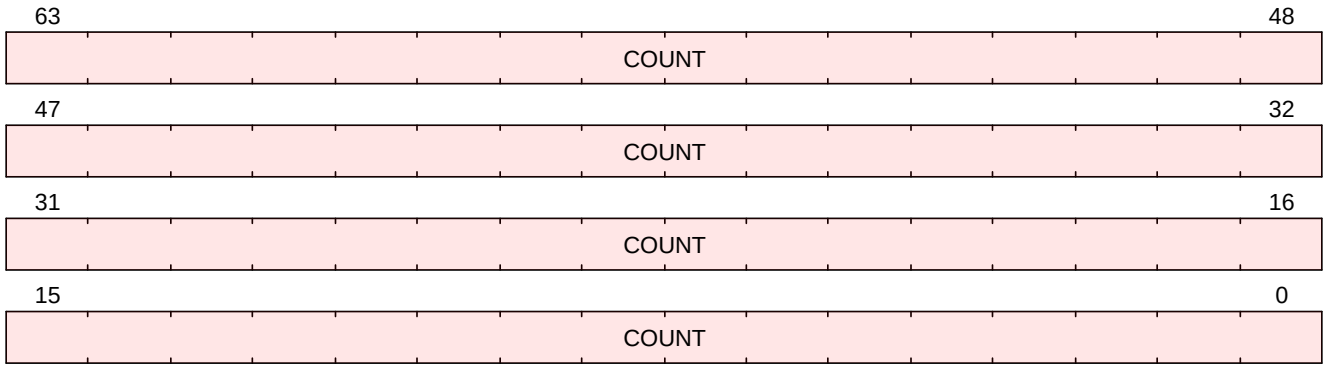


Figure 44. hpmcounter27h format

C.44. hpmcounter28

User-mode Hardware Performance Counter 25

Alias for M-mode CSR mhpmcounter28.

Privilege mode access is controlled with mcounteren.HPM28 <%- if ext?(:S) -%> , scounteren.HPM28 <%- if ext?(:H) -%> , and hcounteren.HPM28 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM28	scounteren.HPM28	hcounteren.HPM28	hpmcounter28 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM28	scounteren.HPM28	hpmcounter28 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM28	hpmcounter28 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.44.1. Attributes

CSR Address	0xc1c

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.44.2. Format

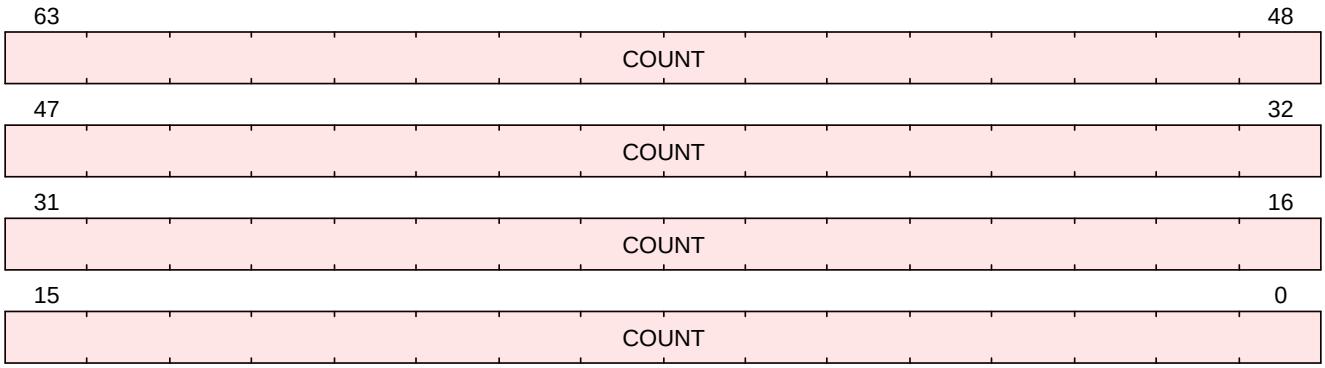


Figure 45. hpmcounter28 format

C.45. hpmcounter28h

User-mode Hardware Performance Counter 25, high half

Alias for M-mode CSR mhpmcounter28h.

Privilege mode access is controlled with mcounteren.HPM28, scounteren.HPM28, and hcounteren.HPM28 as follows:

mcounteren.HPM28	scounteren.HPM28	hcounteren.HPM28	hpmcounter28h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.45.1. Attributes

CSR Address	0xc9c
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.45.2. Format

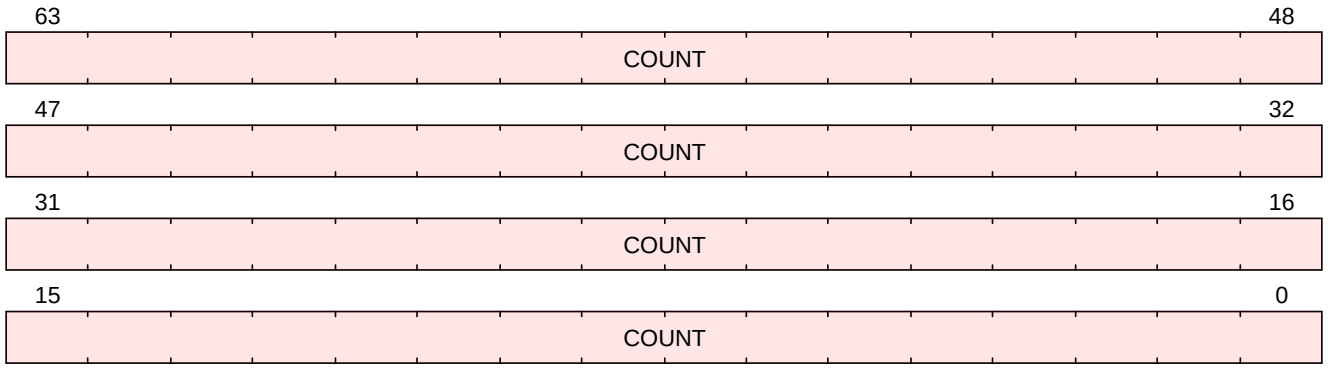


Figure 46. hpmcounter28h format

C.46. hpmcounter29

User-mode Hardware Performance Counter 26

Alias for M-mode CSR mhpmcounter29.

Privilege mode access is controlled with mcounteren.HPM29 <%- if ext?(:S) -%> , scounteren.HPM29 <%- if ext?(:H) -%> , and hcounteren.HPM29 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM29	scounteren.HPM29	hcounteren.HPM29	hpmcounter29 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	`IllegalInstruction`	`IllegalInstruction`	`IllegalInstruction`	`IllegalInstruction`
1	0	0	read-only	`IllegalInstruction`	`VirtualInstruction`	`VirtualInstruction`
1	1	0	read-only	read-only	`VirtualInstruction`	`VirtualInstruction`
1	0	1	read-only	`IllegalInstruction`	read-only	`VirtualInstruction`
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM29	scounteren.HPM29	hpmcounter29 behavior	
		S-mode	U-mode
0	-	`IllegalInstruction`	`IllegalInstruction`
1	0	read-only	`IllegalInstruction`
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM29	hpmcounter29 behavior
	U-mode
0	`IllegalInstruction`
1	read-only

<%- end -%>

C.46.1. Attributes

CSR Address	0xc1d

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.46.2. Format

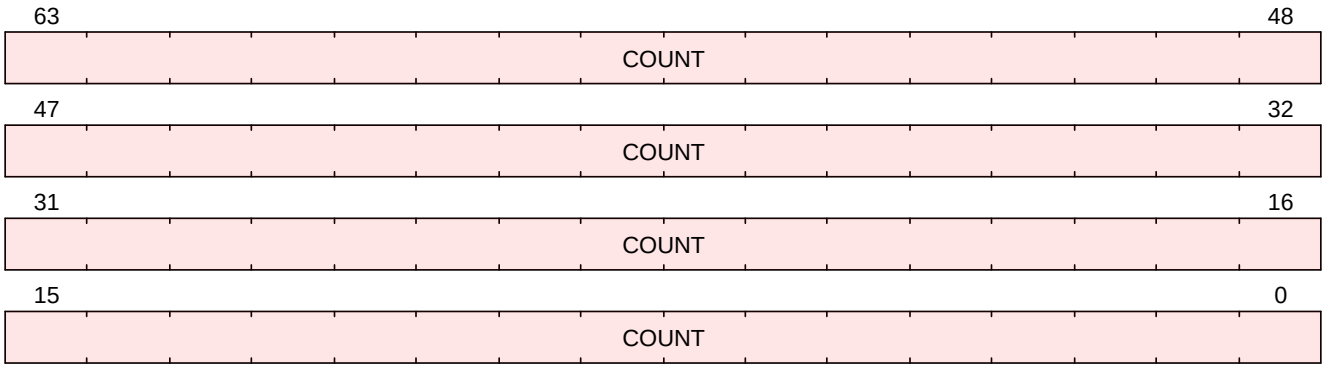


Figure 47. hpmcounter29 format

C.47. hpmcounter29h

User-mode Hardware Performance Counter 26, high half

Alias for M-mode CSR mhpmcounter29h.

Privilege mode access is controlled with mcounteren.HPM29, scounteren.HPM29, and hcounteren.HPM29 as follows:

mcounteren.HPM29	scounteren.HPM29	hcounteren.HPM29	hpmcounter29h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.47.1. Attributes

CSR Address	0xc9d
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.47.2. Format

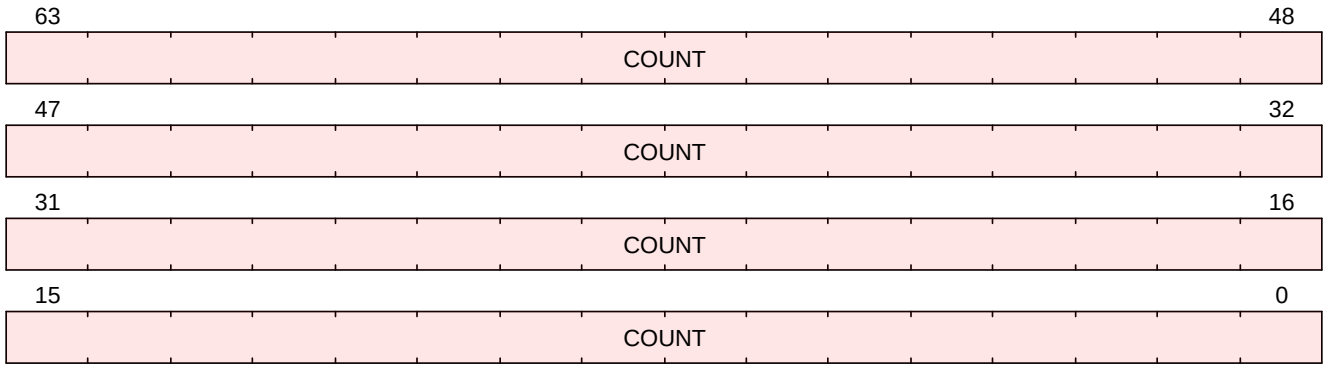


Figure 48. hpmcounter29h format

C.48. hpmcounter3

User-mode Hardware Performance Counter 0

Alias for M-mode CSR mhpmcounter3.

Privilege mode access is controlled with mcounteren.HPM3 <%- if ext?(:S) -%> , scounteren.HPM3 <%- if ext?(:H) -%> , and hcounteren.HPM3 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM3	scounteren.HPM3	hcounteren.HPM3	hpmcounter3 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM3	scounteren.HPM3	hpmcounter3 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM3	hpmcounter3 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.48.1. Attributes

CSR Address	0xc03

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.48.2. Format

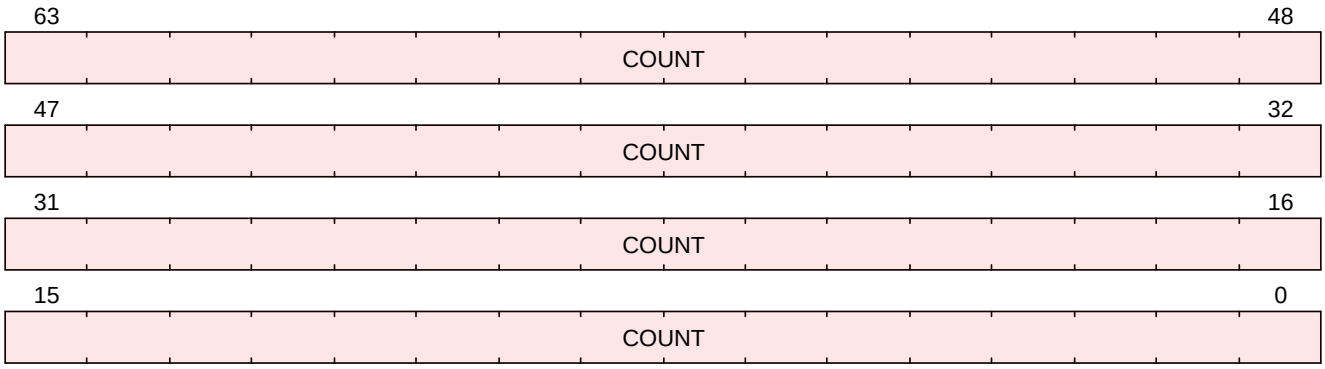


Figure 49. hpmcounter3 format

C.49. hpmcounter30

User-mode Hardware Performance Counter 27

Alias for M-mode CSR mhpmcounter30.

Privilege mode access is controlled with mcounteren.HPM30 <%- if ext?(:S) -%> , scounteren.HPM30 <%- if ext?(:H) -%> , and hcounteren.HPM30 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM30	scounteren.HPM30	hcounteren.HPM30	hpmcounter30 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM30	scounteren.HPM30	hpmcounter30 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM30	hpmcounter30 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.49.1. Attributes

CSR Address	0xc1e

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.49.2. Format

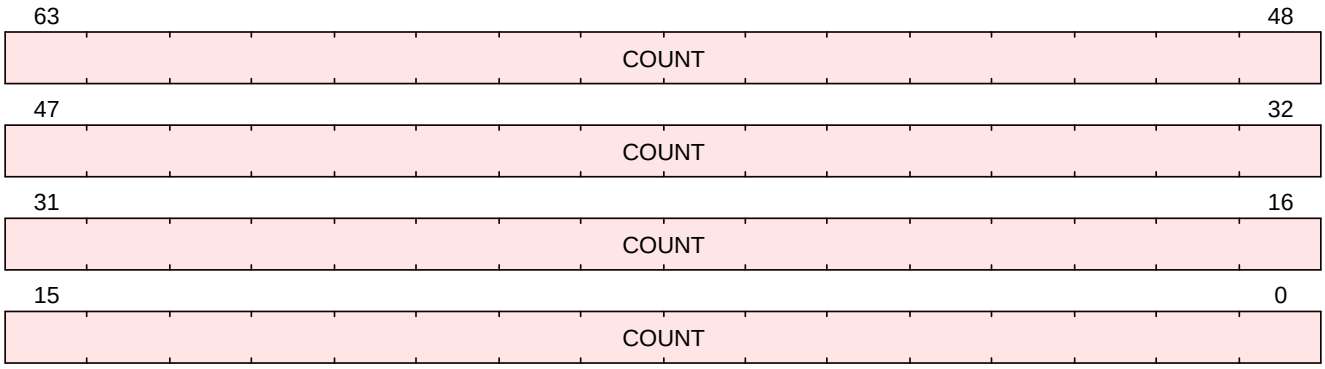


Figure 50. hpmcounter30 format

C.50. hpmcounter30h

User-mode Hardware Performance Counter 27, high half

Alias for M-mode CSR mhpmcounter30h.

Privilege mode access is controlled with mcounteren.HPM30, scounteren.HPM30, and hcounteren.HPM30 as follows:

mcounteren.HPM30	scounteren.HPM30	hcounteren.HPM30	hpmcounter30h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.50.1. Attributes

CSR Address	0xc9e
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.50.2. Format

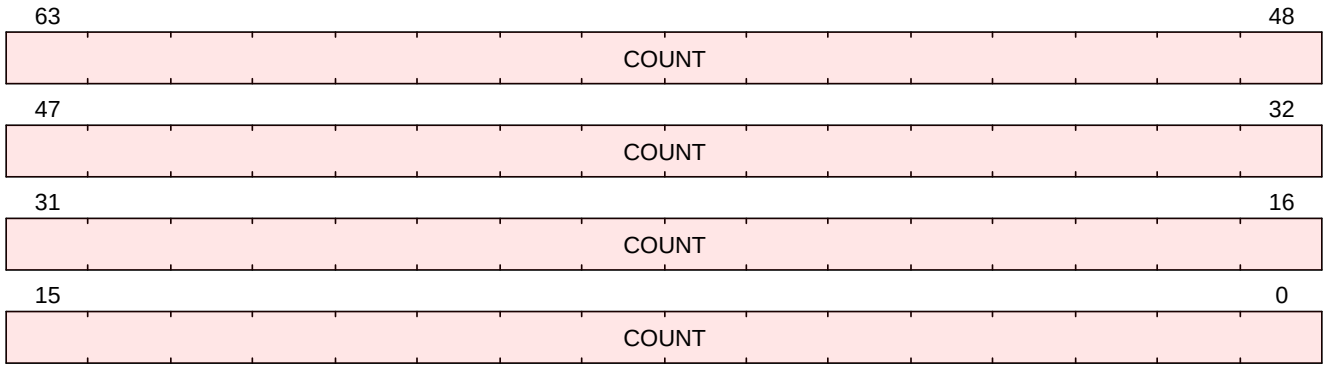


Figure 51. hpmcounter30h format

C.51. hpmcounter31

User-mode Hardware Performance Counter 28

Alias for M-mode CSR mhpmcounter31.

Privilege mode access is controlled with mcounteren.HPM31 <%- if ext?(:S) -%> , scounteren.HPM31 <%- if ext?(:H) -%> , and hcounteren.HPM31 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM31	scounteren.HPM31	hcounteren.HPM31	hpmcounter31 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM31	scounteren.HPM31	hpmcounter31 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM31	hpmcounter31 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.51.1. Attributes

CSR Address	0xc1f

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.51.2. Format

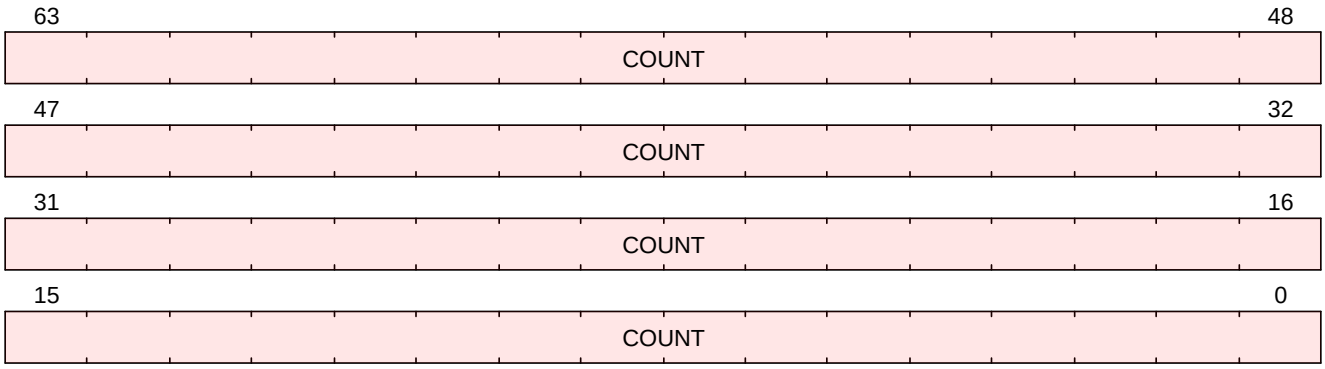


Figure 52. hpmcounter31 format

C.52. hpmcounter31h

User-mode Hardware Performance Counter 28, high half

Alias for M-mode CSR mhpmcounter31h.

Privilege mode access is controlled with mcounteren.HPM31, scounteren.HPM31, and hcounteren.HPM31 as follows:

mcounteren.HPM31	scounteren.HPM31	hcounteren.HPM31	hpmcounter31h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.52.1. Attributes

CSR Address	0xc9f
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.52.2. Format

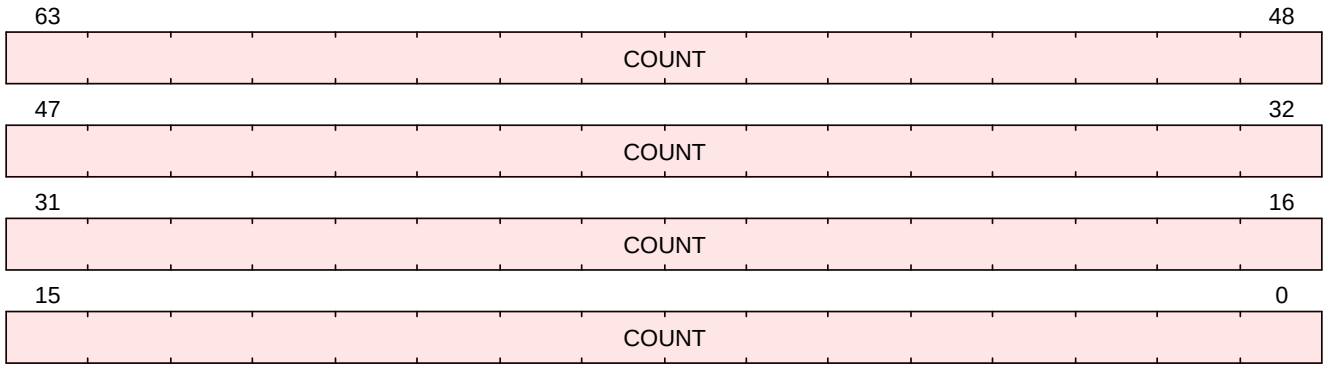


Figure 53. hpmcounter31h format

C.53. hpmcounter3h

User-mode Hardware Performance Counter 0, high half

Alias for M-mode CSR mhpmcounter3h.

Privilege mode access is controlled with mcounteren.HPM3, scounteren.HPM3, and hcounteren.HPM3 as follows:

mcounteren.HPM3	scounteren.HPM3	hcounteren.HPM3	hpmcounter3h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.53.1. Attributes

CSR Address	0xc83
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.53.2. Format

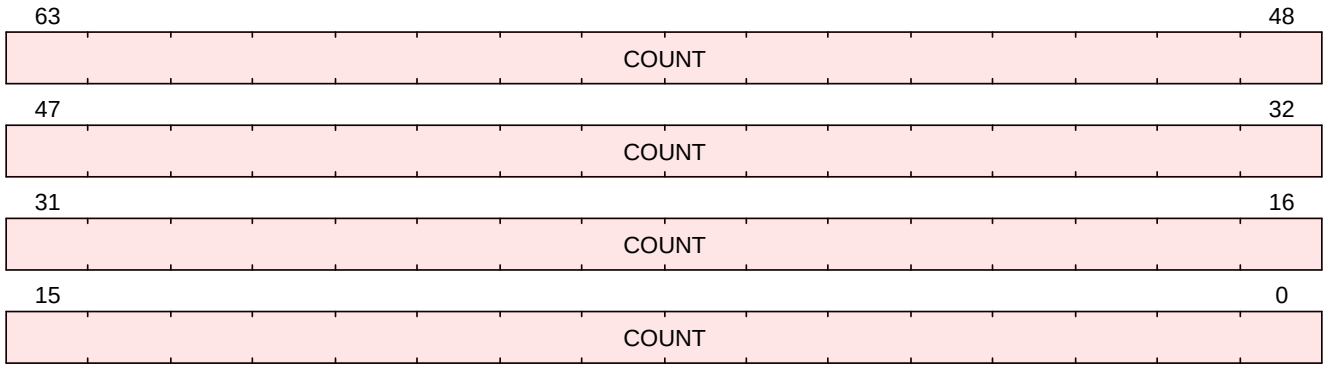


Figure 54. hpmcounter3h format

C.54. hpmcounter4

User-mode Hardware Performance Counter 1

Alias for M-mode CSR mhpmcounter4.

Privilege mode access is controlled with mcounteren.HPM4 <%- if ext?(:S) -%> , scounteren.HPM4 <%- if ext?(:H) -%> , and hcounteren.HPM4 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM4	scounteren.HPM4	hcounteren.HPM4	hpmcounter4 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM4	scounteren.HPM4	hpmcounter4 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM4	hpmcounter4 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.54.1. Attributes

CSR Address	0xc04

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.54.2. Format

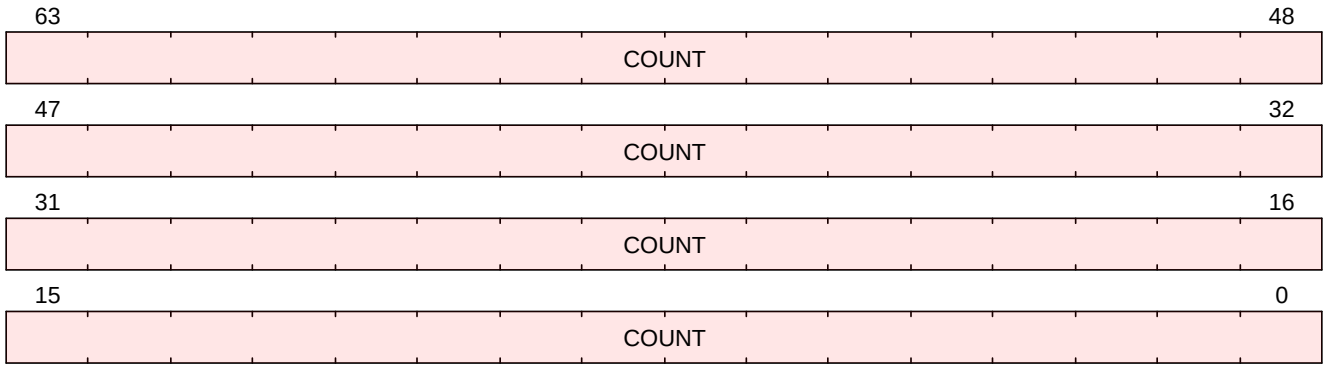


Figure 55. hpmcounter4 format

C.55. hpmcounter4h

User-mode Hardware Performance Counter 1, high half

Alias for M-mode CSR mhpmcounter4h.

Privilege mode access is controlled with mcounteren.HPM4, scounteren.HPM4, and hcounteren.HPM4 as follows:

mcounteren.HPM4	scounteren.HPM4	hcounteren.HPM4	hpmcounter4h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.55.1. Attributes

CSR Address	0xc84
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.55.2. Format

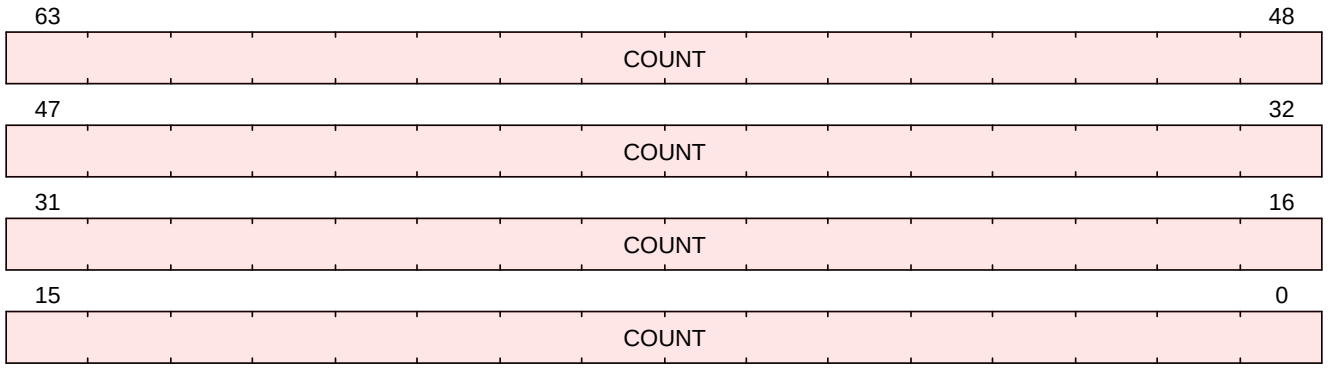


Figure 56. hpmcounter4h format

C.56. hpmcounter5

User-mode Hardware Performance Counter 2

Alias for M-mode CSR mhpmcounter5.

Privilege mode access is controlled with mcounteren.HPM5 <%- if ext?(:S) -%> , scounteren.HPM5 <%- if ext?(:H) -%> , and hcounteren.HPM5 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM5	scounteren.HPM5	hcounteren.HPM5	hpmcounter5 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM5	scounteren.HPM5	hpmcounter5 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM5	hpmcounter5 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.56.1. Attributes

CSR Address	0xc05

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.56.2. Format

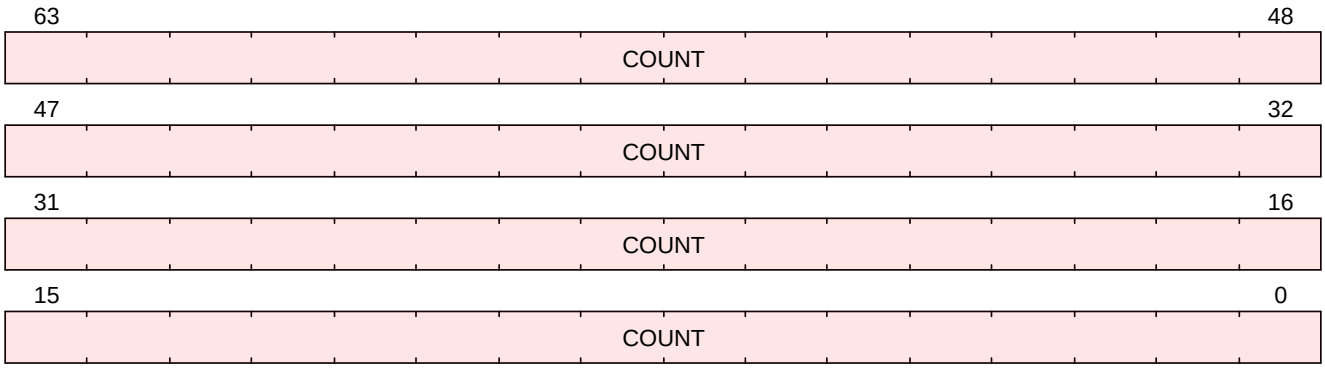


Figure 57. hpmcounter5 format

C.57. hpmcounter5h

User-mode Hardware Performance Counter 2, high half

Alias for M-mode CSR mhpmcounter5h.

Privilege mode access is controlled with mcounteren.HPM5, scounteren.HPM5, and hcounteren.HPM5 as follows:

mcounteren.HPM5	scounteren.HPM5	hcounteren.HPM5	hpmcounter5h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.57.1. Attributes

CSR Address	0xc85
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.57.2. Format

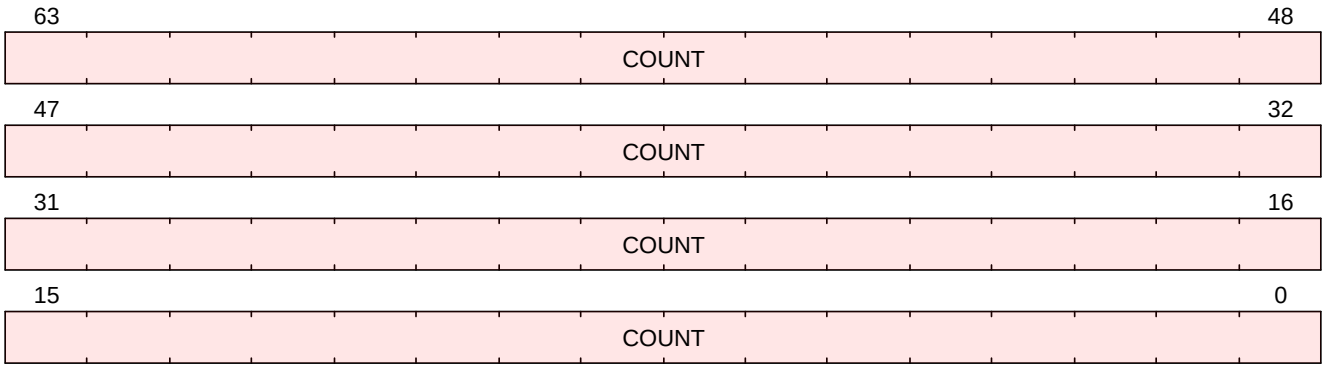


Figure 58. hpmcounter5h format

C.58. hpmcounter6

User-mode Hardware Performance Counter 3

Alias for M-mode CSR mhpmcounter6.

Privilege mode access is controlled with mcounteren.HPM6 <%- if ext?(:S) -%> , scounteren.HPM6 <%- if ext?(:H) -%> , and hcounteren.HPM6 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM6	scounteren.HPM6	hcounteren.HPM6	hpmcounter6 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM6	scounteren.HPM6	hpmcounter6 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM6	hpmcounter6 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.58.1. Attributes

CSR Address	0xc06

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.58.2. Format

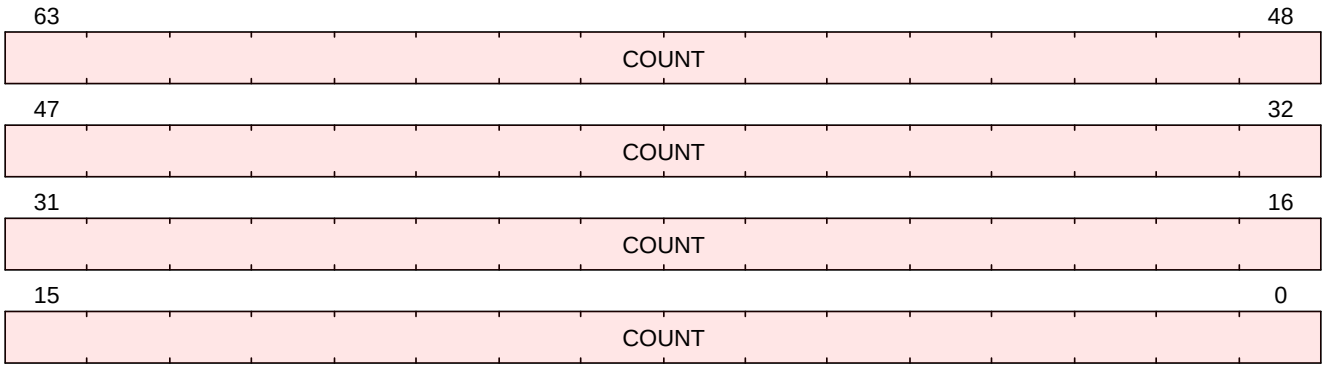


Figure 59. hpmcounter6 format

C.59. hpmcounter6h

User-mode Hardware Performance Counter 3, high half

Alias for M-mode CSR mhpmcounter6h.

Privilege mode access is controlled with mcounteren.HPM6, scounteren.HPM6, and hcounteren.HPM6 as follows:

mcounteren.HPM6	scounteren.HPM6	hcounteren.HPM6	hpmcounter6h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.59.1. Attributes

CSR Address	0xc86
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.59.2. Format

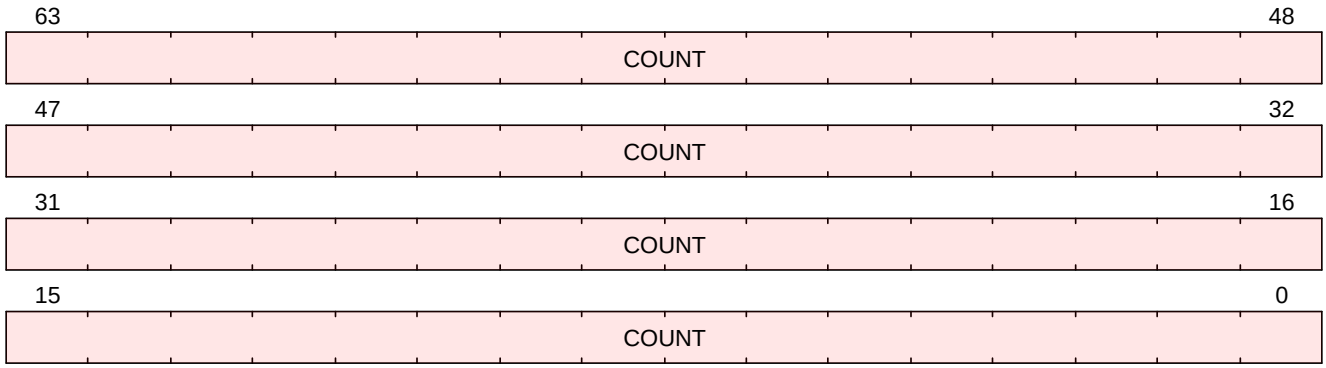


Figure 60. hpmcounter6h format

C.60. hpmcounter7

User-mode Hardware Performance Counter 4

Alias for M-mode CSR mhpmcounter7.

Privilege mode access is controlled with mcounteren.HPM7 <%- if ext?(:S) -%> , scounteren.HPM7 <%- if ext?(:H) -%> , and hcounteren.HPM7 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM7	scounteren.HPM7	hcounteren.HPM7	hpmcounter7 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM7	scounteren.HPM7	hpmcounter7 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM7	hpmcounter7 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.60.1. Attributes

CSR Address	0xc07

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.60.2. Format

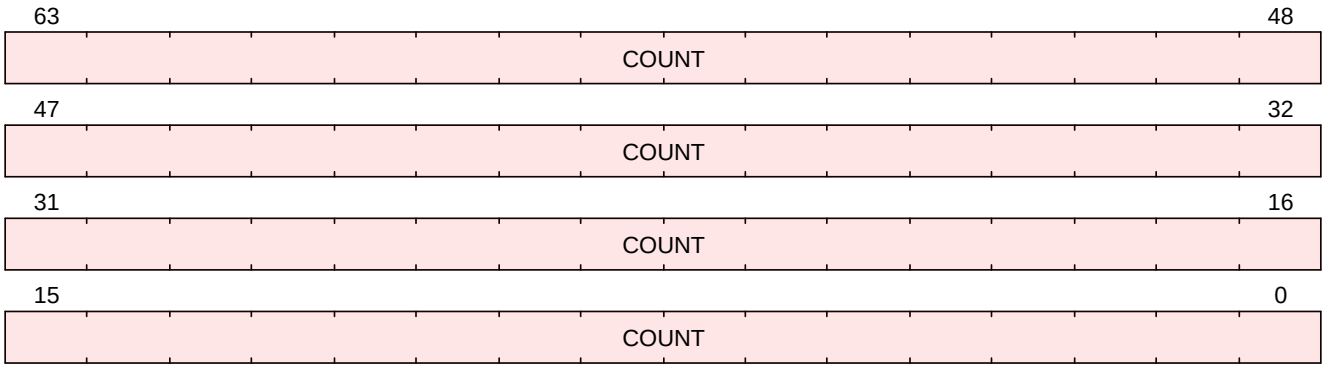


Figure 61. hpmcounter7 format

C.61. hpmcounter7h

User-mode Hardware Performance Counter 4, high half

Alias for M-mode CSR mhpmcounter7h.

Privilege mode access is controlled with mcounteren.HPM7, scounteren.HPM7, and hcounteren.HPM7 as follows:

mcounteren.HPM7	scounteren.HPM7	hcounteren.HPM7	hpmcounter7h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.61.1. Attributes

CSR Address	0xc87
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.61.2. Format

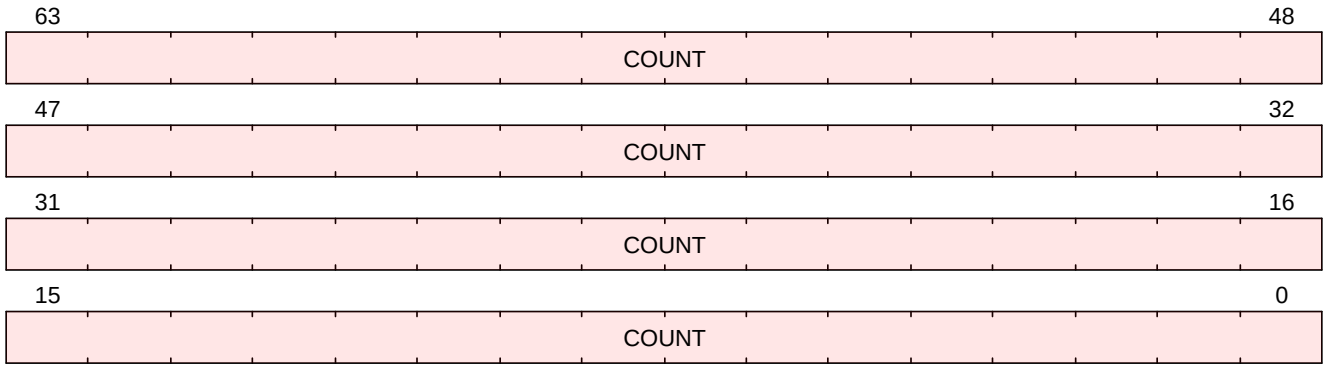


Figure 62. hpmcounter7h format

C.62. hpmcounter8

User-mode Hardware Performance Counter 5

Alias for M-mode CSR mhpmcounter8.

Privilege mode access is controlled with mcounteren.HPM8 <%- if ext?(:S) -%> , scounteren.HPM8 <%- if ext?(:H) -%> , and hcounteren.HPM8 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM8	scounteren.HPM8	hcounteren.HPM8	hpmcounter8 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM8	scounteren.HPM8	hpmcounter8 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM8	hpmcounter8 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.62.1. Attributes

CSR Address	0xc08

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.62.2. Format

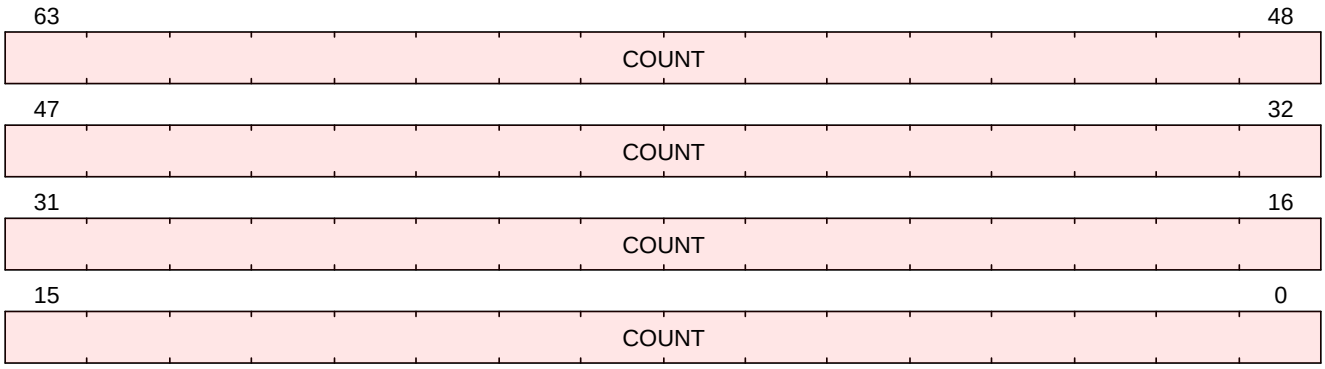


Figure 63. hpmcounter8 format

C.63. hpmcounter8h

User-mode Hardware Performance Counter 5, high half

Alias for M-mode CSR mhpmcounter8h.

Privilege mode access is controlled with mcounteren.HPM8, scounteren.HPM8, and hcounteren.HPM8 as follows:

mcounteren.HPM8	scounteren.HPM8	hcounteren.HPM8	hpmcounter8h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.63.1. Attributes

CSR Address	0xc88
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.63.2. Format

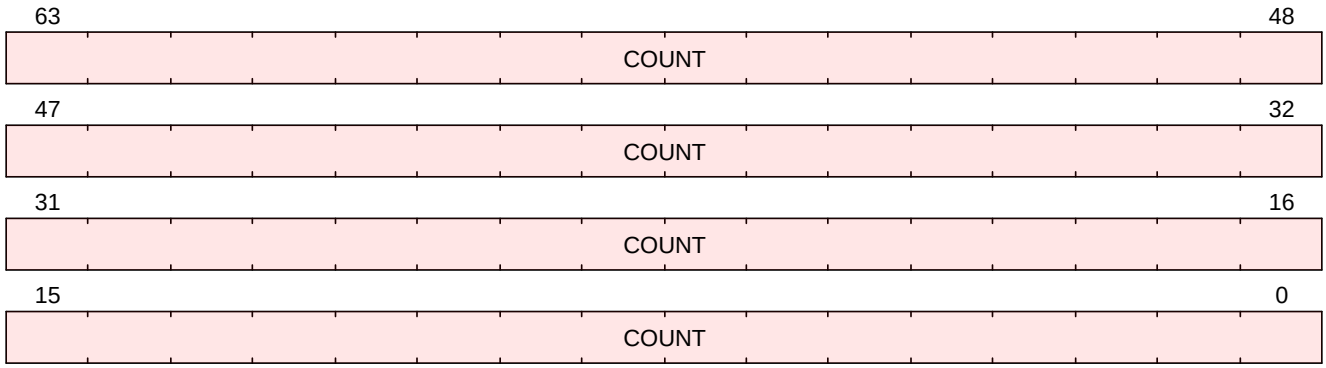


Figure 64. hpmcounter8h format

C.64. hpmcounter9

User-mode Hardware Performance Counter 6

Alias for M-mode CSR mhpmcounter9.

Privilege mode access is controlled with mcounteren.HPM9 <%- if ext?(:S) -%> , scounteren.HPM9 <%- if ext?(:H) -%> , and hcounteren.HPM9 <%- end -%> <%- end -%> as follows:

<%- if ext?(:H) -%>

mcounteren.HPM9	scounteren.HPM9	hcounteren.HPM9	hpmcounter9 behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

<%- elsif ext?(:S) -%>

mcounteren.HPM9	scounteren.HPM9	hpmcounter9 behavior	
		S-mode	U-mode
0	-	IllegalInstruction	IllegalInstruction
1	0	read-only	IllegalInstruction
1	1	read-only	read-only

<%- else -%>

mcounteren.HPM9	hpmcounter9 behavior
	U-mode
0	IllegalInstruction
1	read-only

<%- end -%>

C.64.1. Attributes

CSR Address	0xc09

Defining extension	• Zihpm, version >= 0
Length	64-bit
Privilege Mode	U

C.64.2. Format

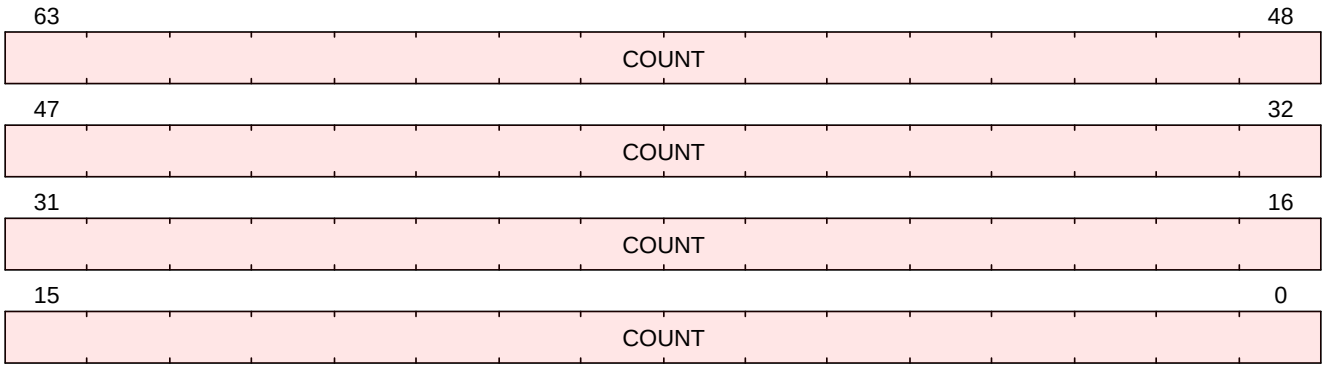


Figure 65. hpmcounter9 format

C.65. hpmcounter9h

User-mode Hardware Performance Counter 6, high half

Alias for M-mode CSR mhpmcounter9h.

Privilege mode access is controlled with mcounteren.HPM9, scounteren.HPM9, and hcounteren.HPM9 as follows:

mcounteren.HPM9	scounteren.HPM9	hcounteren.HPM9	hpmcounter9h behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.65.1. Attributes

CSR Address	0xc89
Defining extension	• Sscofpmf, version >= 0
Length	64-bit
Privilege Mode	U

C.65.2. Format

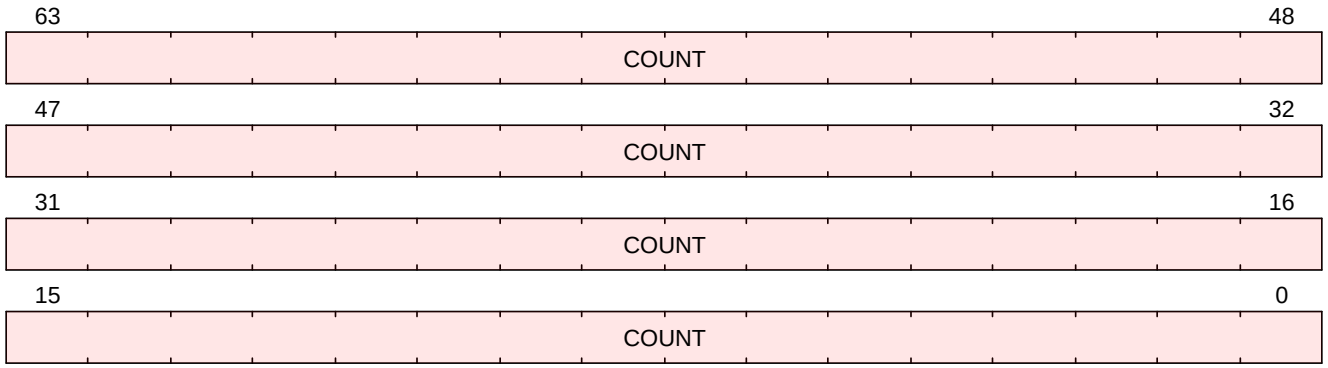


Figure 66. hpmcounter9h format

C.66. hstatus

Hypervisor Status

The hstatus register tracks and controls a VS-mode guest.

Unlike fields in sstatus, which are all aliases of fields mstatus, bits in hstatus are independent bits and do not have aliases.

C.66.1. Attributes

CSR Address	0x600
Defining extension	• H, version >= 0
Length	32 when CSR[mstatus].SXL == 0 64 when CSR[mstatus].SXL == 1
Privilege Mode	S

C.66.2. Format

This CSR format changes dynamically with XLEN.

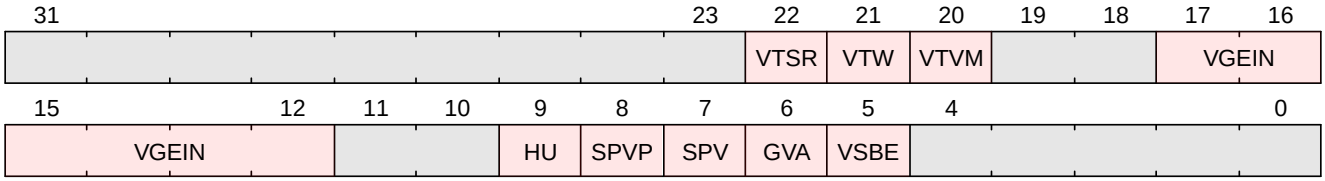


Figure 67. hstatus Format when CSR[mstatus].SXL == 0

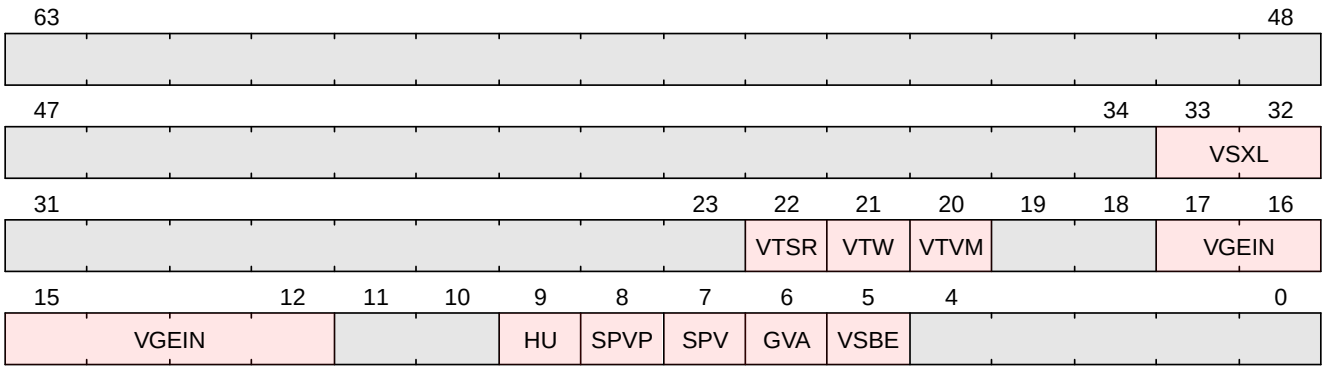


Figure 68. hstatus Format when CSR[mstatus].SXL == 1

C.67. htimedelta

Hypervisor time delta

The htimedelta CSR is a 64-bit read/write register that contains the delta between the value of the time CSR and the value returned in VS-mode or VU-mode. That is, reading the time CSR in VS or VU mode returns the sum of the contents of htimedelta and the actual value of time.

Because overflow is ignored when summing htimedelta and time, large values of htimedelta may be used to represent negative time offsets.

C.67.1. Attributes

CSR Address	0x605
Defining extension	• H, version >= 0
Length	64-bit
Privilege Mode	S

C.67.2. Format

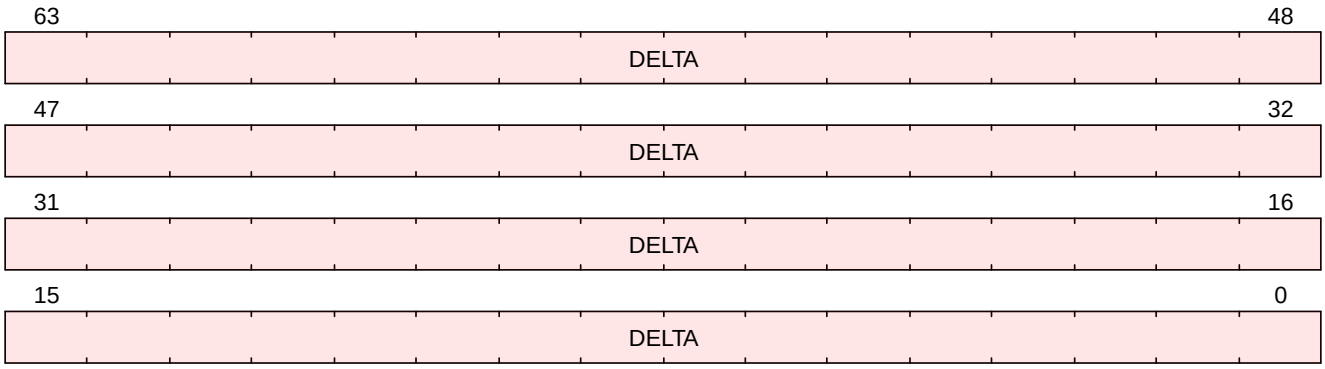


Figure 69. htimedelta format

C.68. htimedeltah

Hypervisor time delta, upper half

Upper half of the htimedelta CSR.

C.68.1. Attributes

CSR Address	0x615
Defining extension	• H, version >= 0
Length	32-bit
Privilege Mode	S

C.68.2. Format

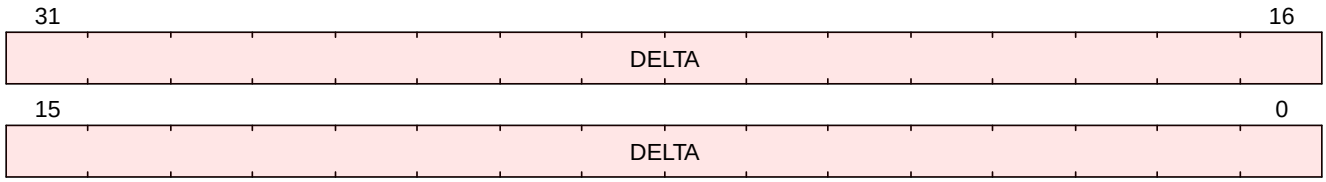


Figure 70. htimedeltah format

C.69. htinst

Hypervisor Trap Instruction Register

When a trap is taken into HS-mode, mtinst is written with a value that, if nonzero, provides information about the instruction that trapped, to assist software in handling the trap. The values that may be written to mtinst on a trap are documented in TODO.

htinst is a WARL register that need only be able to hold the values that the implementation may automatically write to it on a trap.

C.69.1. Attributes

CSR Address	0x64a
Defining extension	• H, version >= 0
Length	32 when CSR[mstatus].SXL == 0 64 when CSR[mstatus].SXL == 1
Privilege Mode	S

C.69.2. Format

This CSR format changes dynamically with XLEN.

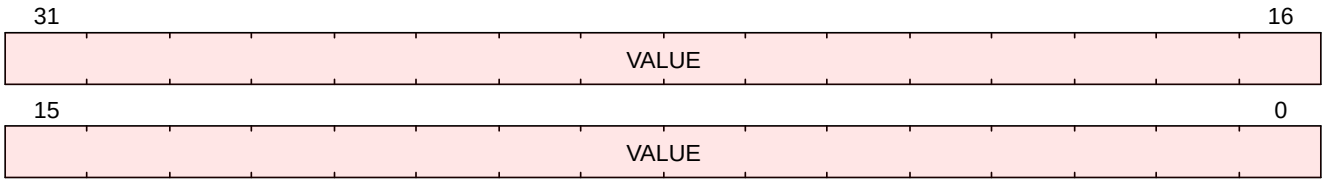


Figure 71. htinst Format when CSR[mstatus].SXL == 0

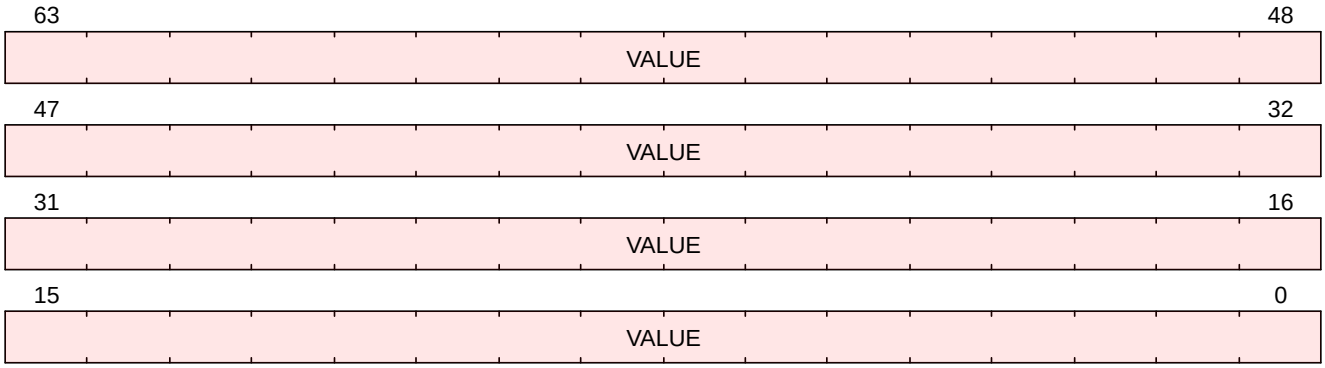


Figure 72. htinst Format when CSR[mstatus].SXL == 1

C.70. htval

Hypervisor Trap Value Register

When a trap is taken into HS-mode, htval is written with additional exception-specific information, alongside stval, to assist software in handling the trap.

When a guest-page-fault trap is taken into HS-mode, htval is written with either zero or the guest physical address that faulted, shifted right by 2 bits. For other traps, htval is set to zero, but a future standard or extension may redefine htval's setting for other traps.

A guest-page fault may arise due to an implicit memory access during first-stage (VS-stage) address translation, in which case a guest physical address written to htval is that of the implicit memory access that faulted-for example, the address of a VS-level page table entry that could not be read. (The guest physical address corresponding to the original virtual address is unknown when VS-stage translation fails to complete.) Additional information is provided in CSR htinst to disambiguate such situations.

Otherwise, for misaligned loads and stores that cause guest-page faults, a nonzero guest physical address in htval corresponds to the faulting portion of the access as indicated by the virtual address in stval. For instruction guest-page faults on systems with variable-length instructions, a nonzero htval corresponds to the faulting portion of the instruction as indicated by the virtual address in stval.

htval is a WARL register that must be able to hold zero and may be capable of holding only an arbitrary subset of other 2-bit-shifted guest physical addresses, if any.

C.70.1. Attributes

CSR Address	0x643
Defining extension	• H, version >= 0
Length	64-bit
Privilege Mode	M

C.70.2. Format

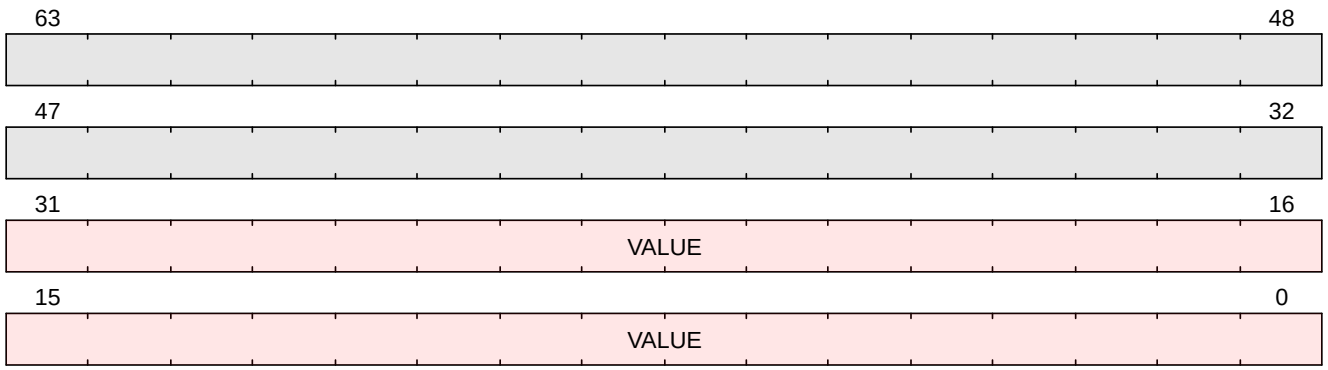


Figure 73. htval format

C.71. instret

Instructions retired counter for RDINSTRET Instruction

Alias for M-mode CSR minstret.

Privilege mode access is controlled with mcounteren.IR, scounteren.IR, and hcounteren.IR as follows:

mcounteren.IR	scounteren.IR	hcounteren.IR	instret behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	IllegalInstruction	IllegalInstruction	IllegalInstruction	IllegalInstruction
1	0	0	read-only	IllegalInstruction	VirtualInstruction	VirtualInstruction
1	1	0	read-only	read-only	VirtualInstruction	VirtualInstruction
1	0	1	read-only	IllegalInstruction	read-only	VirtualInstruction
1	1	1	read-only	read-only	read-only	read-only

C.71.1. Attributes

CSR Address	0xc02
Defining extension	• Zicntr, version >= 0
Length	64-bit
Privilege Mode	U

C.71.2. Format

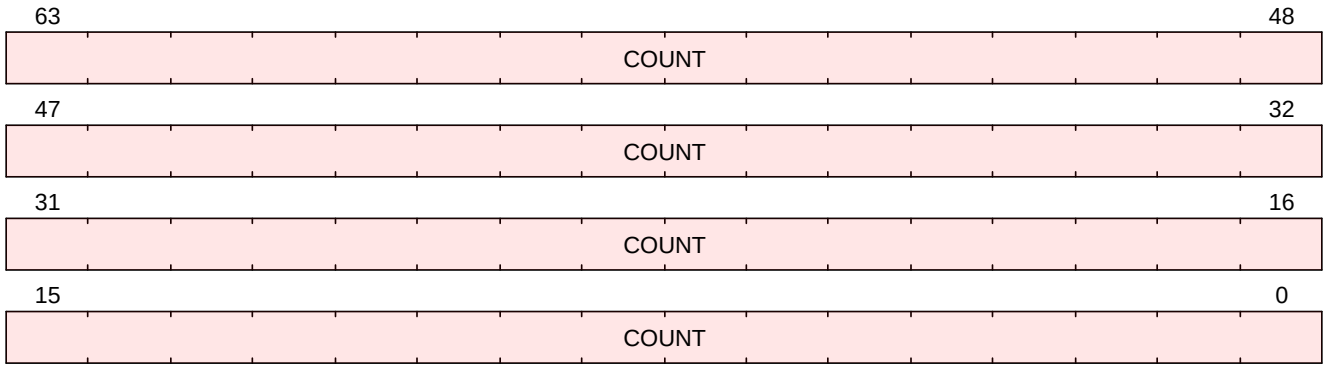


Figure 74. instret format

C.72. instreth

Instructions retired counter, high bits

instreth is only defined in RV32.

Alias for high bits of M-mode CSR minstret[63:32].

Privilege mode access is controlled with mcounteren.IR, scounteren.IR, and hcounteren.IR as follows:

mcounteren.IR	scounteren.IR	hcounteren.IR	instret behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	`IllegalInstruction`	`IllegalInstruction`	`IllegalInstruction`	`IllegalInstruction`
1	0	0	read-only	`IllegalInstruction`	`VirtualInstruction`	`VirtualInstruction`
1	1	0	read-only	read-only	`VirtualInstruction`	`VirtualInstruction`
1	0	1	read-only	`IllegalInstruction`	read-only	`VirtualInstruction`
1	1	1	read-only	read-only	read-only	read-only

C.72.1. Attributes

CSR Address	0xc82
Defining extension	• Zicntr, version >= 0
Length	32-bit
Privilege Mode	U

C.72.2. Format

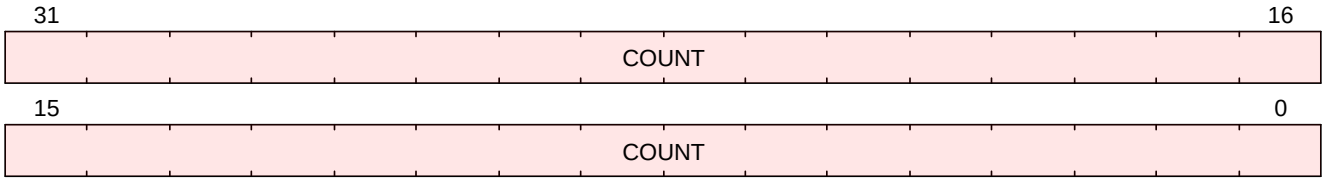


Figure 75. instreth format

C.73. mcounteren

Machine Counter Enable

The counter-enable mcounteren register is a 32-bit register that controls the availability of the hardware performance-monitoring counters to <%- if ext?(:S) -%> S-mode <%- elsif ext?(:U) -%> U-mode <%- else -%> the next-lower privileged mode <%- end -%> .

The settings in this register only control accessibility. The act of reading or writing this register does not affect the underlying counters, which continue to increment even when not accessible.

When the CY, TM, IR, or HPMn bit in the mcounteren register is clear, attempts to read the cycle, time, instret, or `hpmcountern` register while executing in <%- if ext?(:S) -%> S-mode <%- elsif ext?(:U) -%> U-mode <%- else -%> S-mode or U-mode <%- end -%> will cause an `IllegalInstruction` exception. When one of these bits is set, access to the corresponding register is permitted in <%- if ext?(:S) -%> S-mode <%- elsif ext?(:U) -%> U-mode <%- else -%> the next implemented privilege mode (S-mode if implemented, otherwise U-mode). <%- end -%>

The counter-enable bits support two common use cases with minimal hardware. For harts that do not need high-performance timers and counters, machine-mode software can trap accesses and implement all features in software. For harts that need high-performance timers and counters but are not concerned with obfuscating the underlying hardware counters, the counters can be directly exposed to lower privilege modes.

The cycle, instret, and `hpmcountern` CSRs are read-only shadows of mcycle, minstret, and `mhpmcountern`, respectively. The time CSR is a read-only shadow of the memory-mapped `mtime` register. <%- if possible_xlens.include?(32) -%> Analogously, on RV32I the cycleh, instreth and `hpmcounternh` CSRs are read-only shadows of mcycleh, minstreth and `mhpmcounternh`, respectively. On RV32I the timeh CSR is a read-only shadow of the upper 32 bits of the memory-mapped `mtime` register, while time shadows only the lower 32 bits of `mtime`. <%- end -%>

Implementations can convert reads of the time and timeh CSRs into loads to the memory-mapped `mtime` register, or emulate this functionality on behalf of less-privileged modes in M-mode software.

<%- if !ext?(:U) -%> In harts with U-mode, the mcounteren CSR must be implemented, but all fields are WARL and may be read-only zero, indicating reads to the corresponding counter will cause an `IllegalInstruction` exception when executing in a less-privileged mode. In harts without U-mode, the mcounteren register should not exist. <%- end -%>

<%- if ext?(:S) -%>

The cycle, instret, and `hpmcountern` CSRs can also be made available to U-mode through the scounteren CSR <%- if ext?(:H) -%> and to VS-mode and/or VU-mode through hcounteren <%- end -%> . <%- end -%>

C.73.1. Attributes

CSR Address	0x306
Defining extension	• U, version >= 0
Length	32-bit
Privilege Mode	M

C.73.2. Format

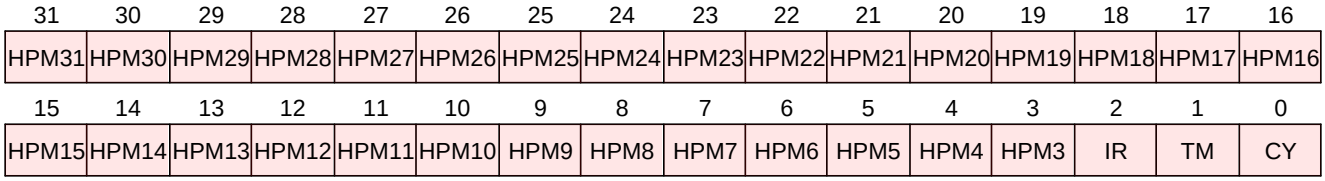


Figure 76. mcounteren format

C.74. mcycle

Machine Cycle Counter

Counts the number of clock cycles executed by the processor core on which the hart is running. The counter has 64-bit precision on all RV32 and RV64 harts.

The mcycle CSR may be shared between harts on the same core, in which case writes to mcycle will be visible to those harts. The platform should provide a mechanism to indicate which harts share an mcycle CSR.

C.74.1. Attributes

CSR Address	0xb00
Defining extension	• Zicntr, version >= 0
Length	64-bit
Privilege Mode	M

C.74.2. Format

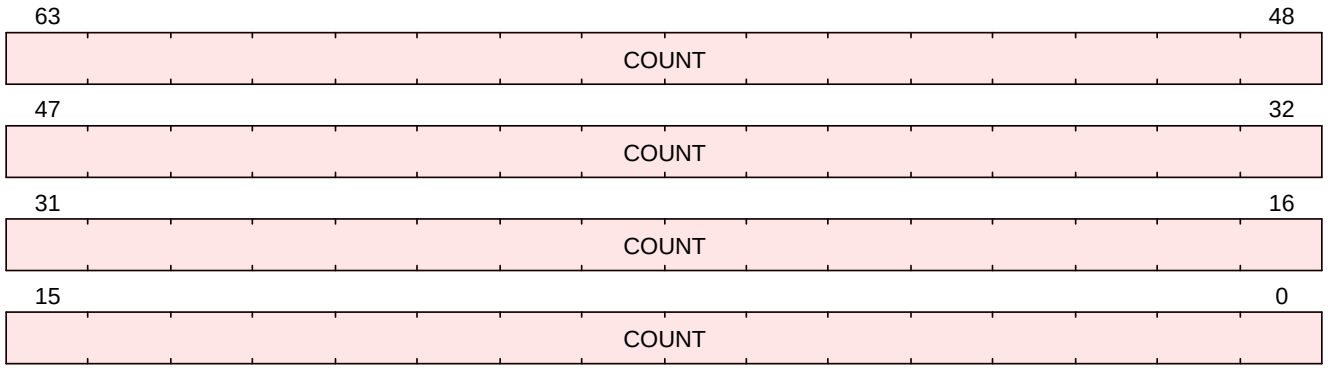


Figure 77. mcycle format

C.75. mcycleh

High-half machine Cycle Counter

mcycleh is only defined in RV32.

High-half alias of mcycle.

C.75.1. Attributes

CSR Address	0xb80
Defining extension	• Zicntr, version >= 0
Length	32-bit
Privilege Mode	M

C.75.2. Format

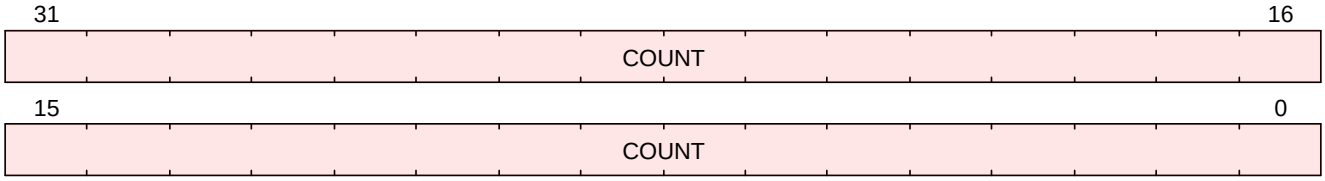


Figure 78. mcycleh format

C.76. medeleg

Machine Exception Delegation

Controls exception delegation from M-mode to (H)S-mode <%- if ext?(:H) -%> or, in conjunction with hedeleg, to VS-mode <%- end -%> .

An exception cause is delegated to (H)S-mode when all of the following hold:

- The corresponding field in medeleg is set.
- The current privilege level is not M-mode. <%- if ext?(:H) -%>
- The same field in hedeleg is clear. <%- end -%>

<%- if ext?(:H) -%> An exception cause is delegated to VS-mode when all of the following hold:

- The corresponding field in medeleg is set.
- The corresponding field in hedeleg is set.
- The current privilege level is not M-mode or HS-mode. <%- end -%>

Otherwise, an exception cause is handled by M-mode.

See interrupt documentation for more details.

C.76.1. Attributes

CSR Address	0x302
Defining extension	• S, version >= 0
Length	64-bit
Privilege Mode	M

C.76.2. Format

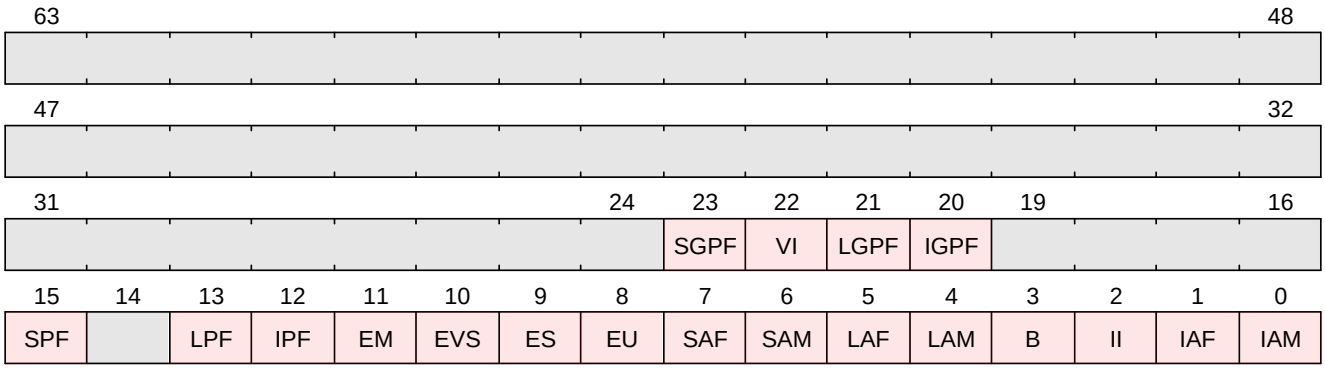


Figure 79. medeleg format

C.77. mhpmevent10h

Machine Hardware Performance Counter 10 Control, High half

mhpmevent10h is only defined in RV32.

Alias of mhpmevent10[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.77.1. Attributes

CSR Address	0x72a
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.77.2. Format

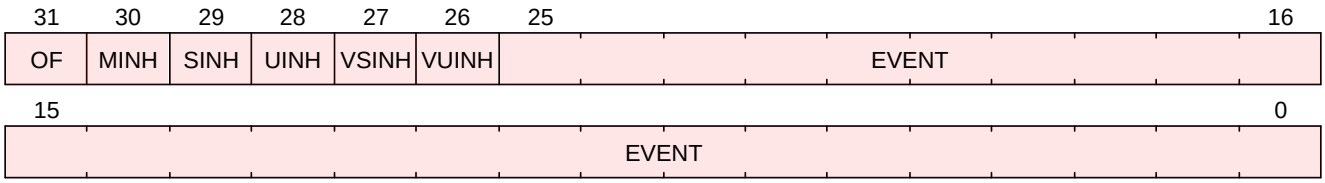


Figure 80. mhpmevent10h format

C.78. mhpmevent11h

Machine Hardware Performance Counter 11 Control, High half

mhpmevent11h is only defined in RV32.

Alias of mhpmevent11[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.78.1. Attributes

CSR Address	0x72b
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.78.2. Format

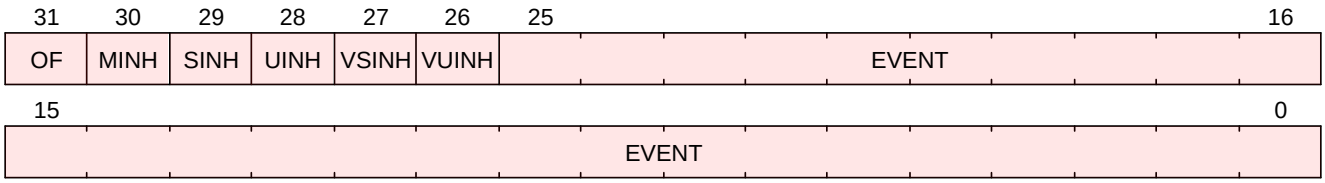


Figure 81. mhpmevent11h format

C.79. mhpmevent12h

Machine Hardware Performance Counter 12 Control, High half

mhpmevent12h is only defined in RV32.

Alias of mhpmevent12[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.79.1. Attributes

CSR Address	0x72c
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.79.2. Format

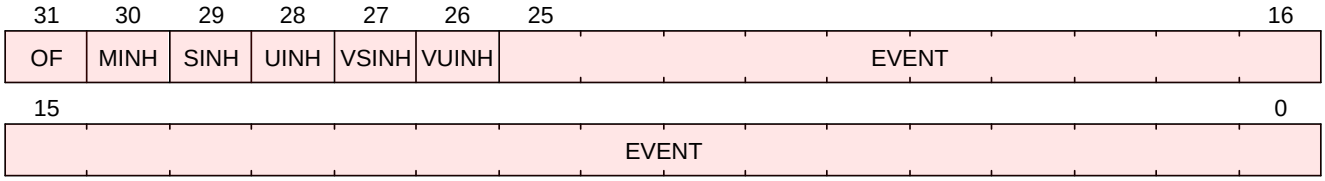


Figure 82. mhpmevent12h format

C.80. mhpmevent13h

Machine Hardware Performance Counter 13 Control, High half

mhpmevent13h is only defined in RV32.

Alias of mhpmevent13[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.80.1. Attributes

CSR Address	0x72d
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.80.2. Format

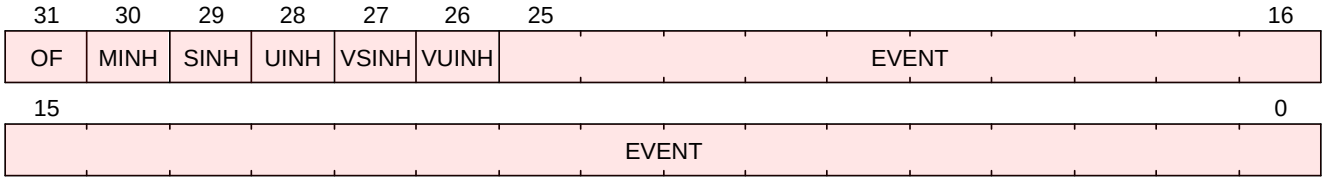


Figure 83. mhpmevent13h format

C.81. mhpmevent14h

Machine Hardware Performance Counter 14 Control, High half

mhpmevent14h is only defined in RV32.

Alias of mhpmevent14[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.81.1. Attributes

CSR Address	0x72e
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.81.2. Format

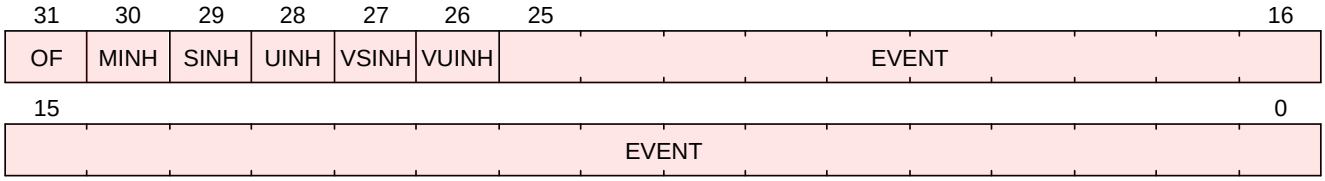


Figure 84. mhpmevent14h format

C.82. mhpmevent15h

Machine Hardware Performance Counter 15 Control, High half

mhpmevent15h is only defined in RV32.

Alias of mhpmevent15[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.82.1. Attributes

CSR Address	0x72f
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.82.2. Format

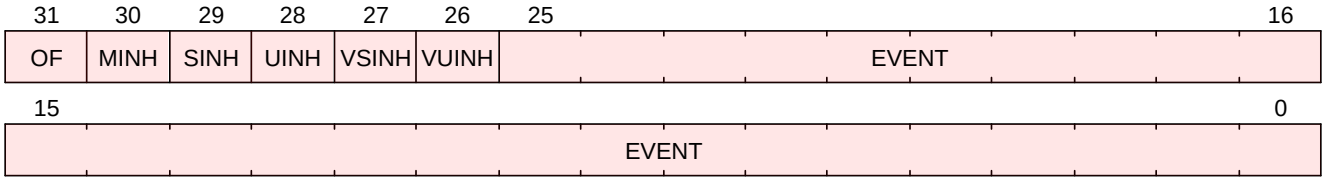


Figure 85. mhpmevent15h format

C.83. mhpmevent16h

Machine Hardware Performance Counter 16 Control, High half

mhpmevent16h is only defined in RV32.

Alias of mhpmevent16[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.83.1. Attributes

CSR Address	0x730
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.83.2. Format

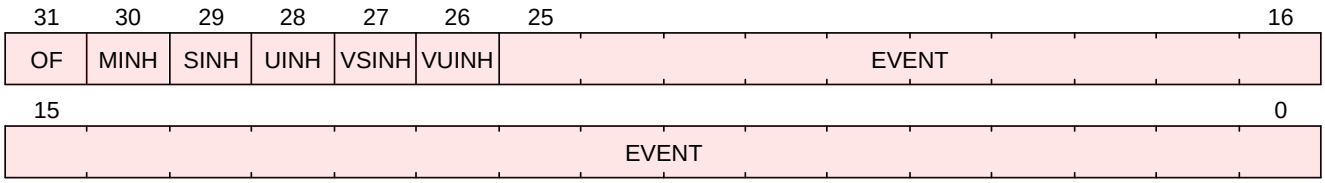


Figure 86. mhpmevent16h format

C.84. mhpmevent17h

Machine Hardware Performance Counter 17 Control, High half

mhpmevent17h is only defined in RV32.

Alias of mhpmevent17[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.84.1. Attributes

CSR Address	0x731
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.84.2. Format

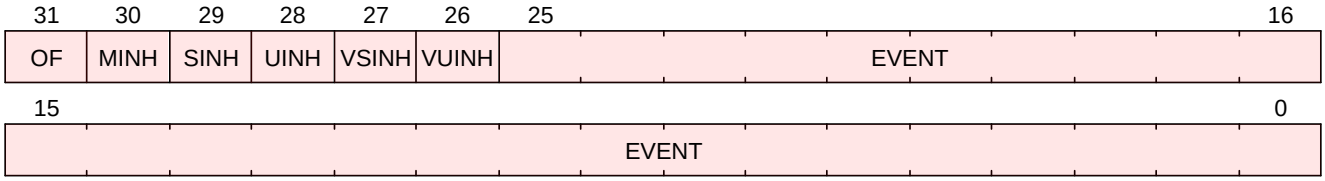


Figure 87. mhpmevent17h format

C.85. mhpmevent18h

Machine Hardware Performance Counter 18 Control, High half

mhpmevent18h is only defined in RV32.

Alias of mhpmevent18[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.85.1. Attributes

CSR Address	0x732
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.85.2. Format

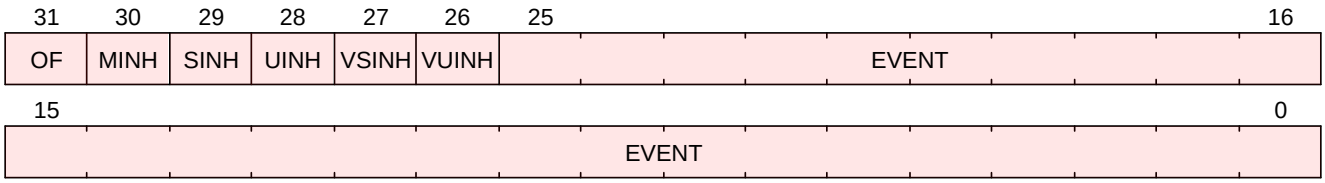


Figure 88. mhpmevent18h format

C.86. mhpmevent19h

Machine Hardware Performance Counter 19 Control, High half

mhpmevent19h is only defined in RV32.

Alias of mhpmevent19[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.86.1. Attributes

CSR Address	0x733
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.86.2. Format

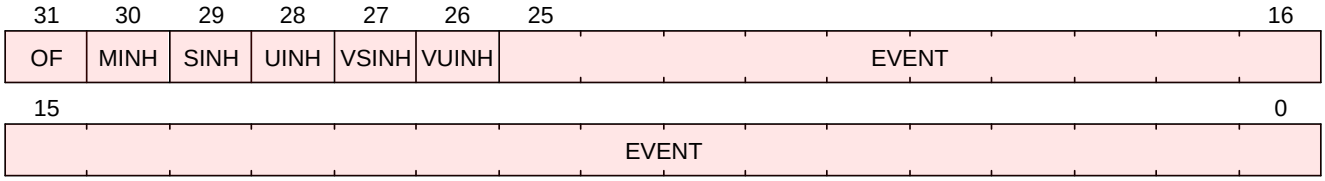


Figure 89. mhpmevent19h format

C.87. mhpmevent20h

Machine Hardware Performance Counter 20 Control, High half

mhpmevent20h is only defined in RV32.

Alias of mhpmevent20[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.87.1. Attributes

CSR Address	0x734
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.87.2. Format

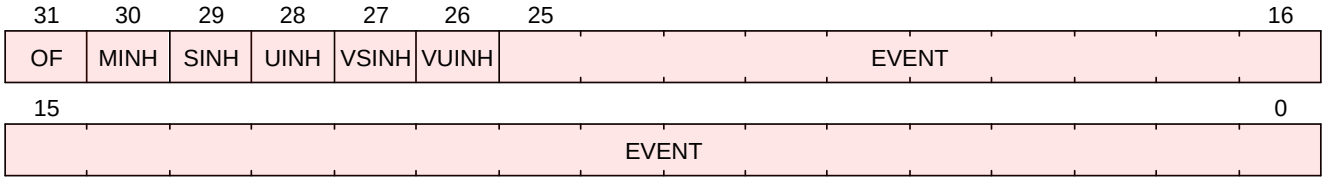


Figure 90. mhpmevent20h format

C.88. mhpmevent21h

Machine Hardware Performance Counter 21 Control, High half

mhpmevent21h is only defined in RV32.

Alias of mhpmevent21[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.88.1. Attributes

CSR Address	0x735
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.88.2. Format

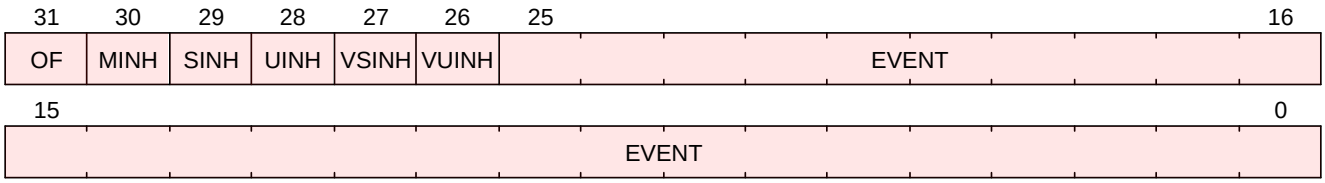


Figure 91. mhpmevent21h format

C.89. mhpmevent22h

Machine Hardware Performance Counter 22 Control, High half

mhpmevent22h is only defined in RV32.

Alias of mhpmevent22[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.89.1. Attributes

CSR Address	0x736
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.89.2. Format

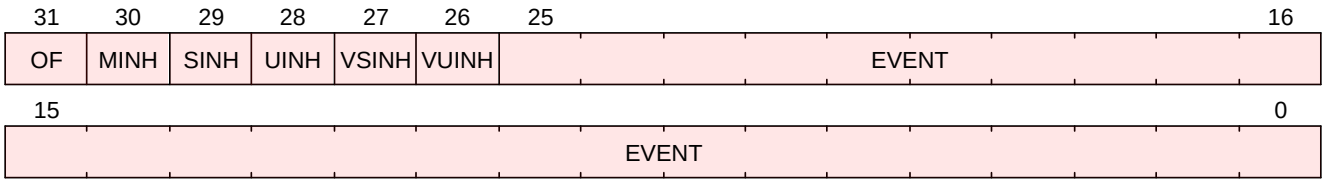


Figure 92. mhpmevent22h format

C.90. mhpmevent23h

Machine Hardware Performance Counter 23 Control, High half

mhpmevent23h is only defined in RV32.

Alias of mhpmevent23[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.90.1. Attributes

CSR Address	0x737
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.90.2. Format

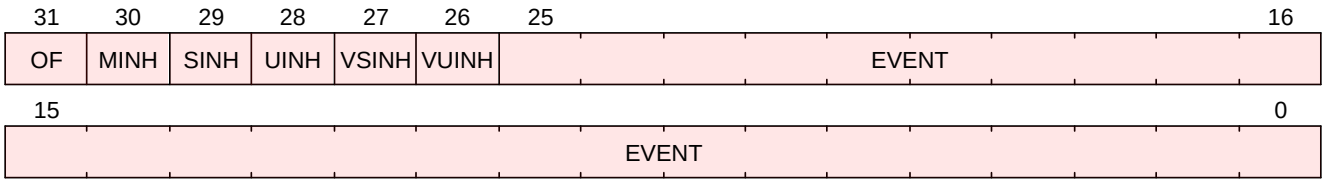


Figure 93. mhpmevent23h format

C.91. mhpmevent24h

Machine Hardware Performance Counter 24 Control, High half

mhpmevent24h is only defined in RV32.

Alias of mhpmevent24[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.91.1. Attributes

CSR Address	0x738
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.91.2. Format

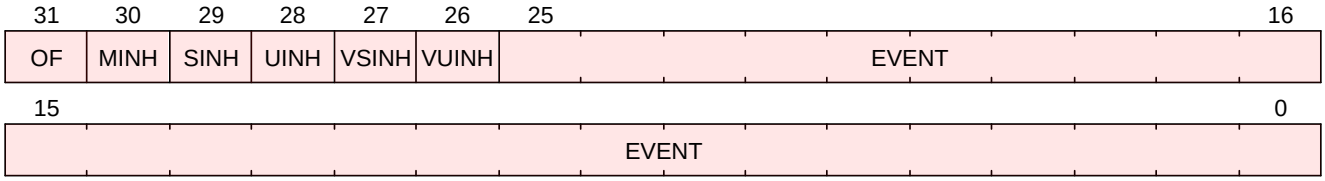


Figure 94. mhpmevent24h format

C.92. mhpmevent25h

Machine Hardware Performance Counter 25 Control, High half

mhpmevent25h is only defined in RV32.

Alias of mhpmevent25[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.92.1. Attributes

CSR Address	0x739
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.92.2. Format

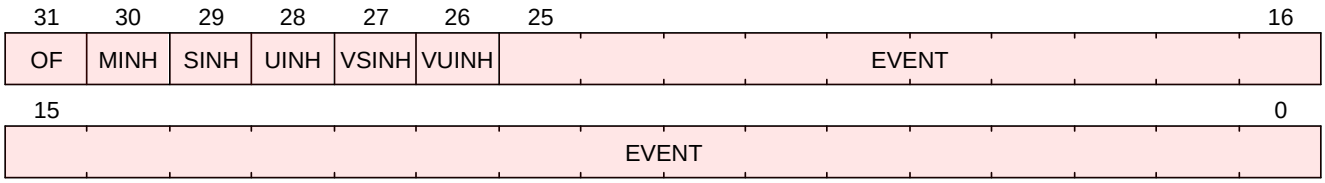


Figure 95. mhpmevent25h format

C.93. mhpmevent26h

Machine Hardware Performance Counter 26 Control, High half

mhpmevent26h is only defined in RV32.

Alias of mhpmevent26[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.93.1. Attributes

CSR Address	0x73a
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.93.2. Format

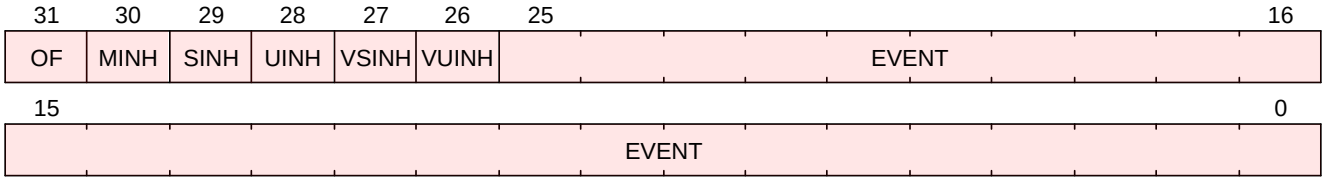


Figure 96. mhpmevent26h format

C.94. mhpmevent27h

Machine Hardware Performance Counter 27 Control, High half

mhpmevent27h is only defined in RV32.

Alias of mhpmevent27[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.94.1. Attributes

CSR Address	0x73b
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.94.2. Format

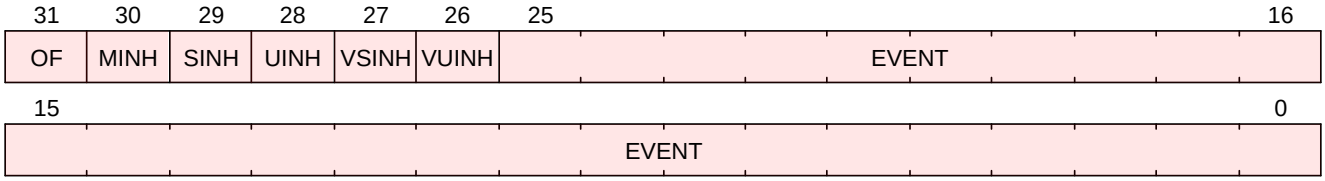


Figure 97. mhpmevent27h format

C.95. mhpmevent28h

Machine Hardware Performance Counter 28 Control, High half

mhpmevent28h is only defined in RV32.

Alias of mhpmevent28[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.95.1. Attributes

CSR Address	0x73c
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.95.2. Format

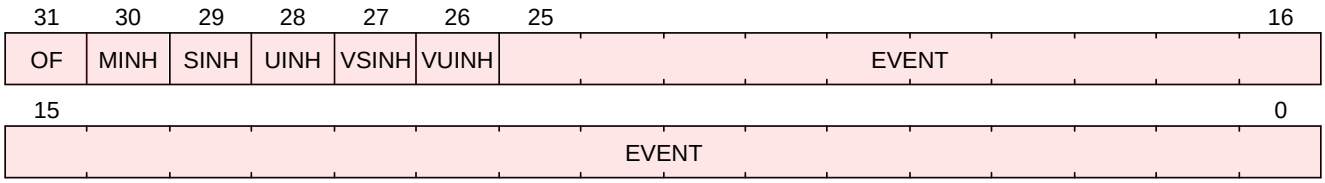


Figure 98. mhpmevent28h format

C.96. mhpmevent29h

Machine Hardware Performance Counter 29 Control, High half

mhpmevent29h is only defined in RV32.

Alias of mhpmevent29[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.96.1. Attributes

CSR Address	0x73d
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.96.2. Format

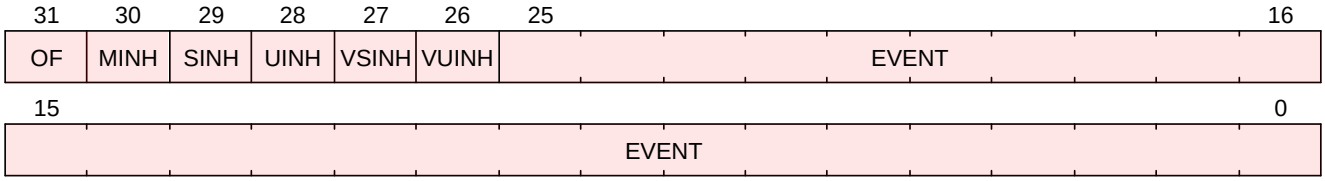


Figure 99. mhpmevent29h format

C.97. mhpmevent30h

Machine Hardware Performance Counter 30 Control, High half

mhpmevent30h is only defined in RV32.

Alias of mhpmevent30[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.97.1. Attributes

CSR Address	0x73e
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.97.2. Format

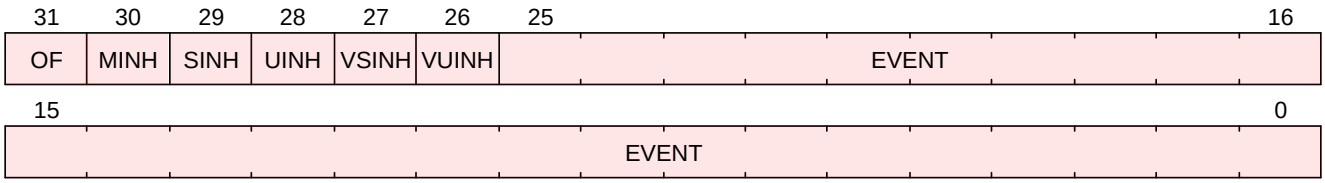


Figure 100. mhpmevent30h format

C.98. mhpmevent31h

Machine Hardware Performance Counter 31 Control, High half

mhpmevent31h is only defined in RV32.

Alias of mhpmevent31[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.98.1. Attributes

CSR Address	0x73f
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.98.2. Format

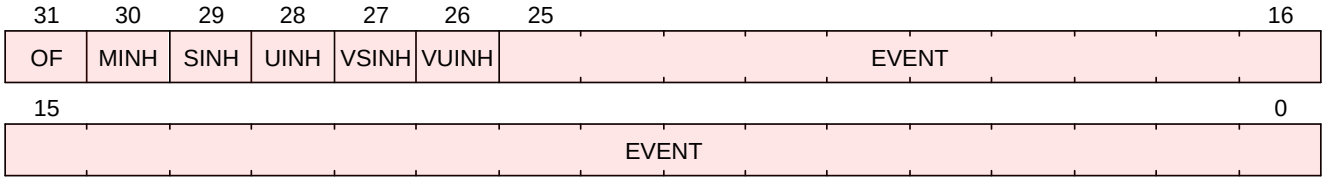


Figure 101. mhpmevent31h format

C.99. mhpmevent3h

Machine Hardware Performance Counter 3 Control, High half

mhpmevent3h is only defined in RV32.

Alias of mhpmevent3[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.99.1. Attributes

CSR Address	0x723
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.99.2. Format

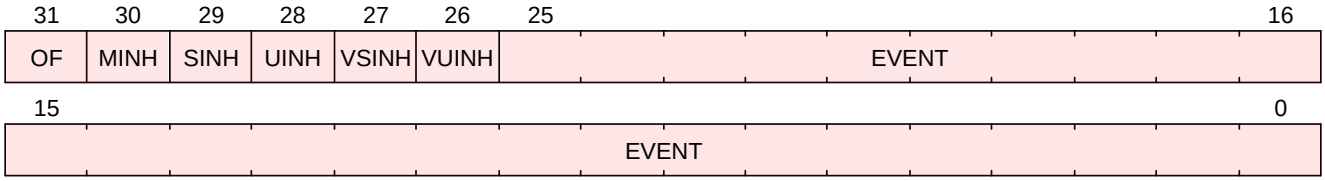


Figure 102. mhpmevent3h format

C.100. mhpmevent4h

Machine Hardware Performance Counter 4 Control, High half

mhpmevent4h is only defined in RV32.

Alias of mhpmevent4[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.100.1. Attributes

CSR Address	0x724
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.100.2. Format

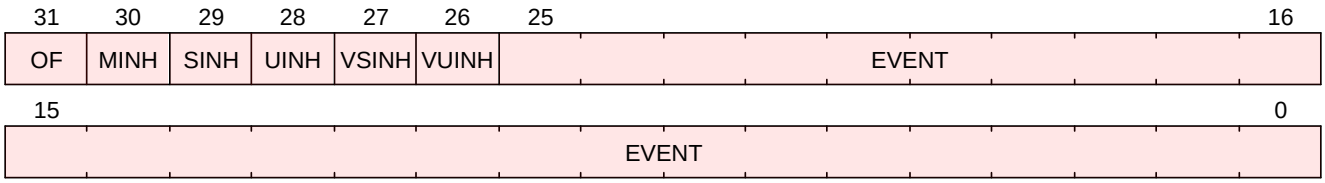


Figure 103. mhpmevent4h format

C.101. mhpmevent5h

Machine Hardware Performance Counter 5 Control, High half

mhpmevent5h is only defined in RV32.

Alias of mhpmevent5[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.101.1. Attributes

CSR Address	0x725
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.101.2. Format

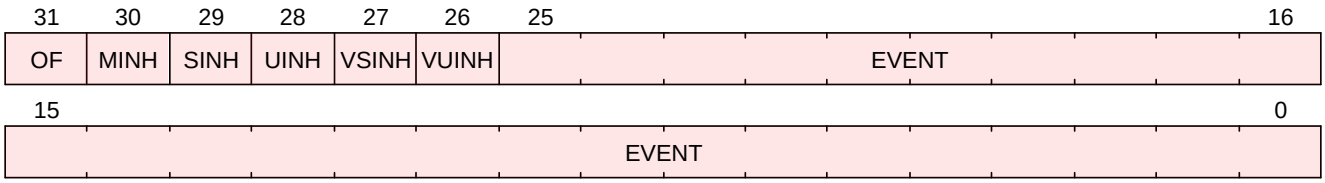


Figure 104. mhpmevent5h format

C.102. mhpmevent6h

Machine Hardware Performance Counter 6 Control, High half

mhpmevent6h is only defined in RV32.

Alias of mhpmevent6[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.102.1. Attributes

CSR Address	0x726
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.102.2. Format

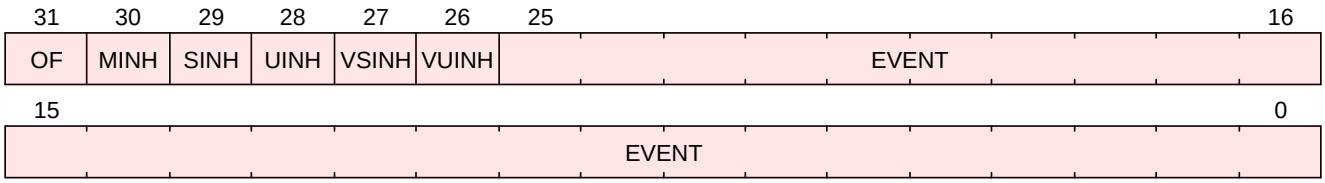


Figure 105. mhpmevent6h format

C.103. mhpmevent7h

Machine Hardware Performance Counter 7 Control, High half

mhpmevent7h is only defined in RV32.

Alias of mhpmevent7[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.103.1. Attributes

CSR Address	0x727
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.103.2. Format

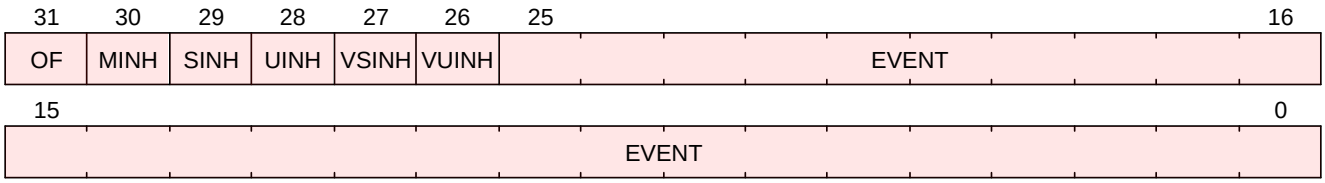


Figure 106. mhpmevent7h format

C.104. mhpmevent8h

Machine Hardware Performance Counter 8 Control, High half

mhpmevent8h is only defined in RV32.

Alias of mhpmevent8[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.104.1. Attributes

CSR Address	0x728
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.104.2. Format

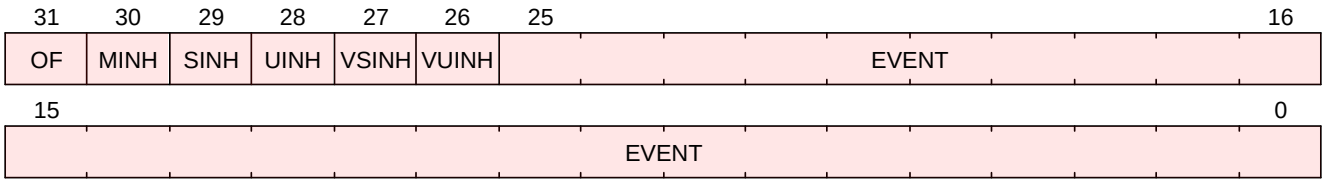


Figure 107. mhpmevent8h format

C.105. mhpmevent9h

Machine Hardware Performance Counter 9 Control, High half

mhpmevent9h is only defined in RV32.

Alias of mhpmevent9[63:32].

Introduced with the Sscofpmf extension. Prior to that, there was no way to access the upper 32-bits of `mhpmevent#{hpm_num}`.

C.105.1. Attributes

CSR Address	0x729
Defining extension	• Sscofpmf, version >= 0
Length	32-bit
Privilege Mode	M

C.105.2. Format

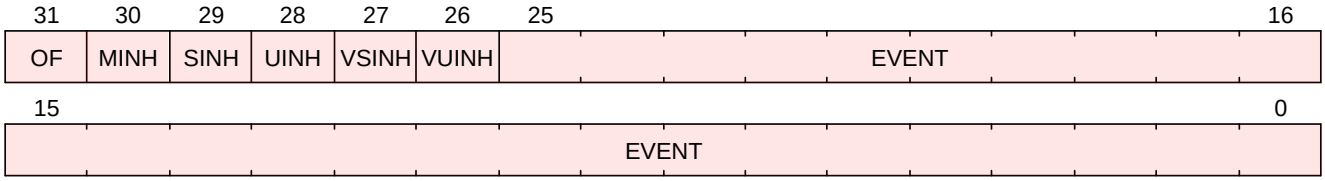


Figure 108. mhpmevent9h format

C.106. minstret

Machine Instructions Retired Counter

Counts the number of instructions retired by this hart from some arbitrary start point in the past.

Instructions that cause synchronous exceptions, including ecall and ebreak, are not considered to retire and hence do not increment the minstret CSR.

C.106.1. Attributes

CSR Address	0xb02
Defining extension	• Zicntr, version >= 0
Length	64-bit
Privilege Mode	M

C.106.2. Format

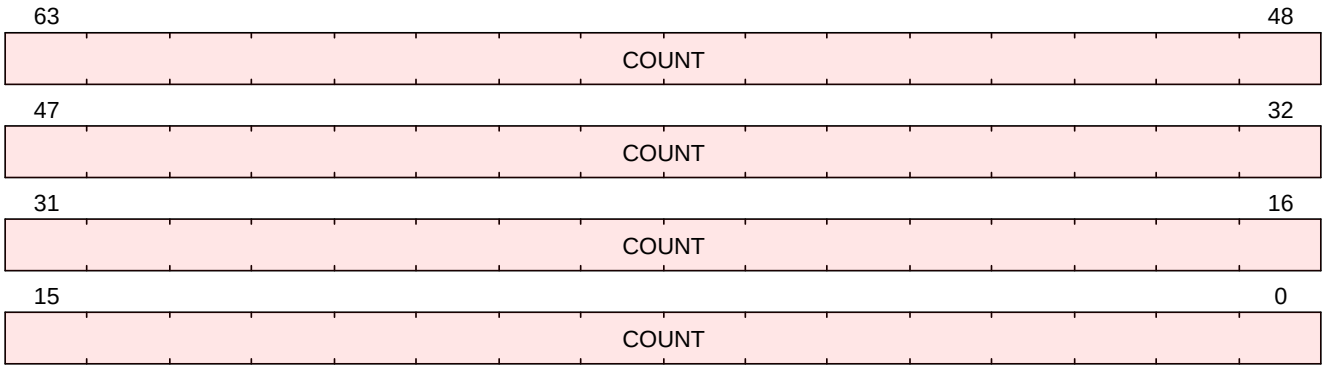


Figure 109. minstret format

C.107. minstreth

Machine Instructions Retired Counter

Upper half of 64-bit instructions retired counters.

See minstret for details.

C.107.1. Attributes

CSR Address	0xb02
Defining extension	• Zicntr, version >= 0
Length	32-bit
Privilege Mode	M

C.107.2. Format

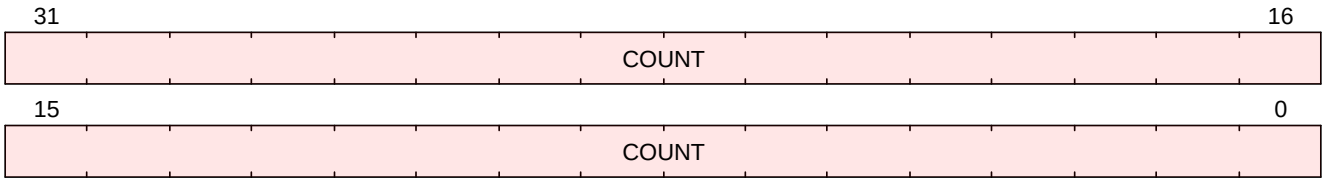


Figure 110. minstreth format

C.108. mtinst

Machine Trap Instruction Register

When a trap is taken into M-mode, mtinst is written with a value that, if nonzero, provides information about the instruction that trapped, to assist software in handling the trap. The values that may be written to mtinst on a trap are documented in TODO.

mtinst is a WARL register that need only be able to hold the values that the implementation may automatically write to it on a trap.

C.108.1. Attributes

CSR Address	0x34a
Defining extension	• H, version >= 0
Length	64-bit
Privilege Mode	M

C.108.2. Format

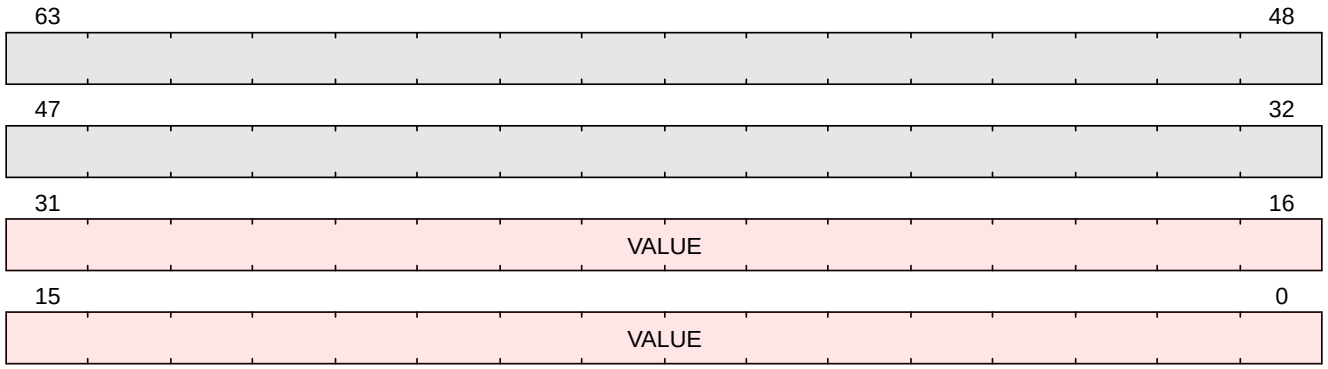


Figure 111. mtinst format

C.109. mtval2

Machine Second Trap Value Register

When a trap is taken into M-mode from a virtual mode, mtval2 is written with additional exception-specific information, alongside mtval, to assist software in handling the trap.

When a guest-page-fault trap is taken into M-mode, mtval2 is written with either zero or the guest physical address that faulted, shifted right by 2 bits. For other traps, mtval2 is set to zero, but a future standard or extension may redefine mtval2's setting for other traps.

If a guest-page fault is due to an implicit memory access during first-stage (VS-stage) address translation, a guest physical address written to mtval2 is that of the implicit memory access that faulted. Additional information is provided in CSR mtinst to disambiguate such situations.

Otherwise, for misaligned loads and stores that cause guest-page faults, a nonzero guest physical address in mtval2 corresponds to the faulting portion of the access as indicated by the virtual address in mtval. For instruction guest-page faults on systems with variable-length instructions, a nonzero mtval2 corresponds to the faulting portion of the instruction as indicated by the virtual address in mtval.

mtval2 is a WARL register that must be able to hold zero and may be capable of holding only an arbitrary subset of other 2-bit-shifted guest physical addresses, if any.

C.109.1. Attributes

CSR Address	0x34b
Defining extension	• H, version >= 0
Length	64-bit
Privilege Mode	M

C.109.2. Format

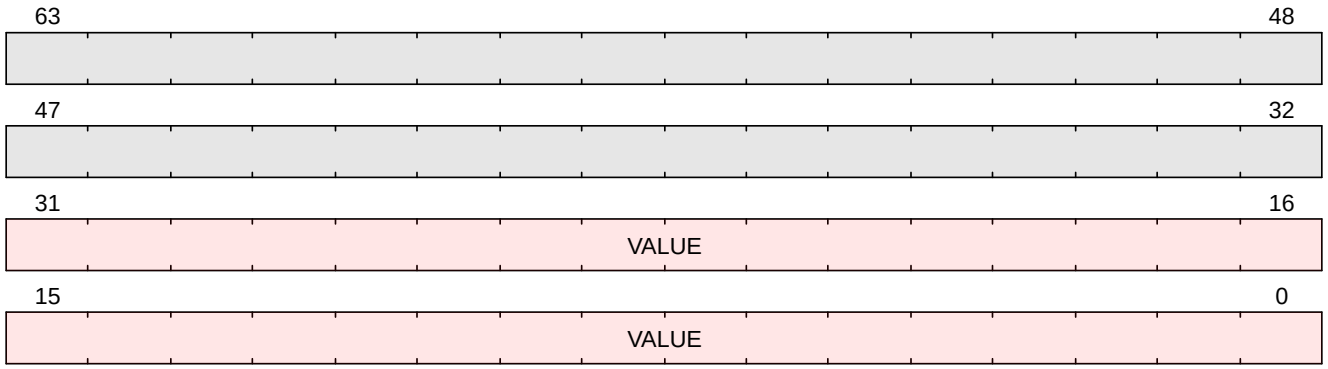


Figure 112. mtval2 format

C.110. satp

Supervisor Address Translation and Protection

Controls the translation mode in (H)S-mode and U-mode, and holds the current ASID and page table base pointer.

C.110.1. Attributes

CSR Address	0x180
Defining extension	• S, version >= 0
Length	32 when CSR[mstatus].SXL == 0 64 when CSR[mstatus].SXL == 1
Privilege Mode	S

C.110.2. Format

This CSR format changes dynamically with XLEN.

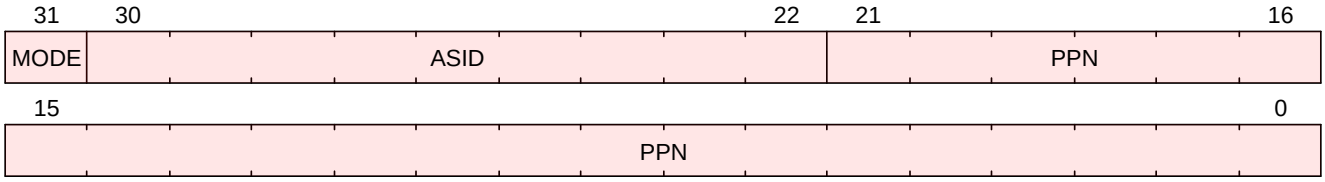


Figure 113. satp Format when CSR[mstatus].SXL == 0

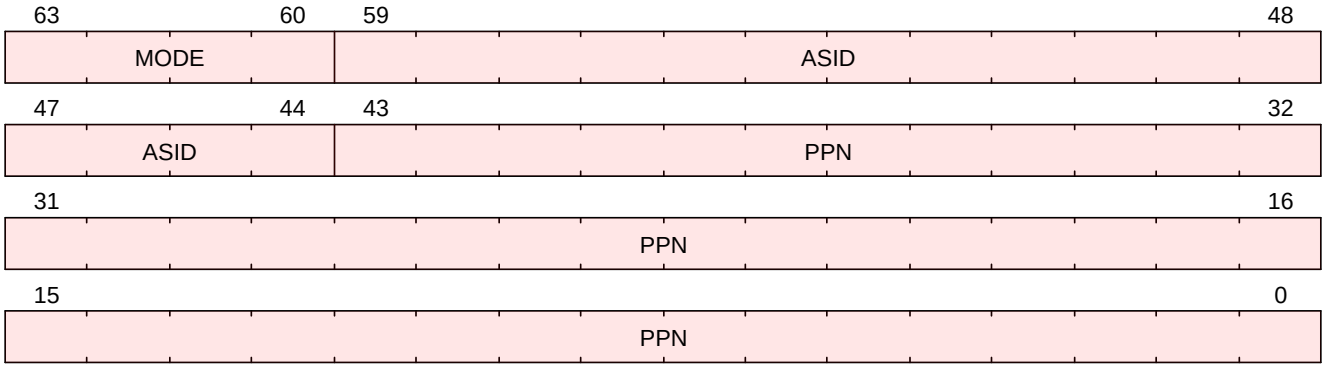


Figure 114. satp Format when CSR[mstatus].SXL == 1

C.111. scause

Supervisor Cause

Reports the cause of the latest exception.

C.111.1. Attributes

CSR Address	0x142
Defining extension	• S, version >= 0
Length	32 when CSR[mstatus].SXL == 0 64 when CSR[mstatus].SXL == 1
Privilege Mode	S

C.111.2. Format

This CSR format changes dynamically with XLEN.

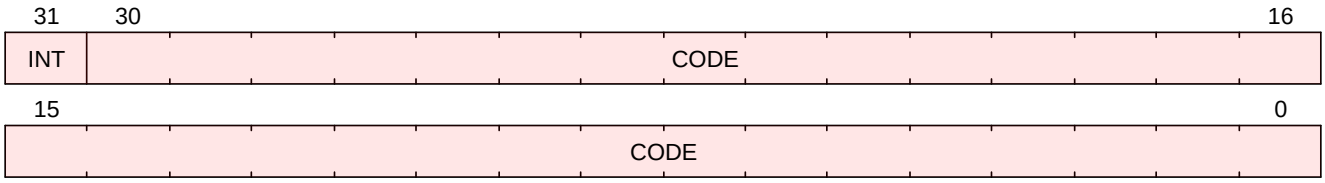


Figure 115. scause Format when CSR[mstatus].SXL == 0

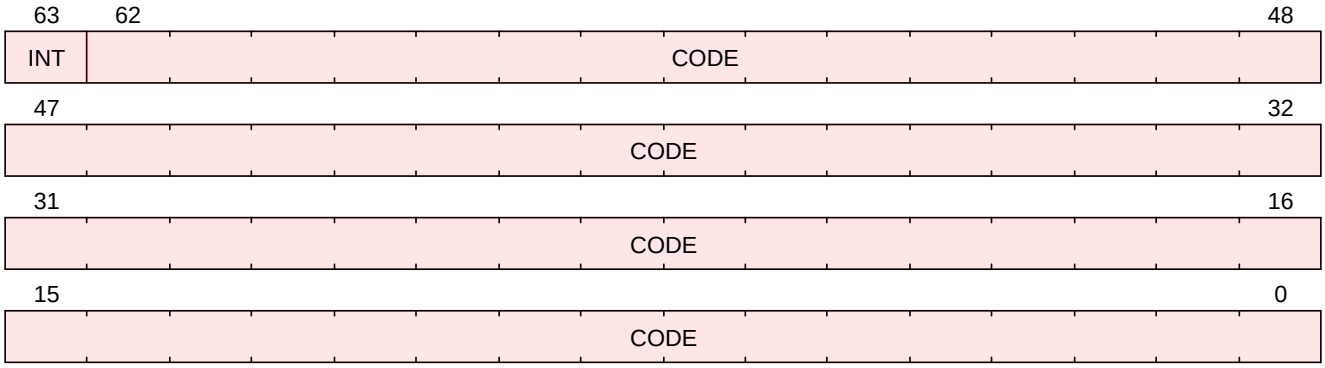


Figure 116. scause Format when CSR[mstatus].SXL == 1

C.112. scounteren

Supervisor Counter Enable

Delegates control of the hardware performance-monitoring counters to U-mode

C.112.1. Attributes

CSR Address	0x106
Defining extension	• S, version >= 0
Length	32-bit
Privilege Mode	S

C.112.2. Format

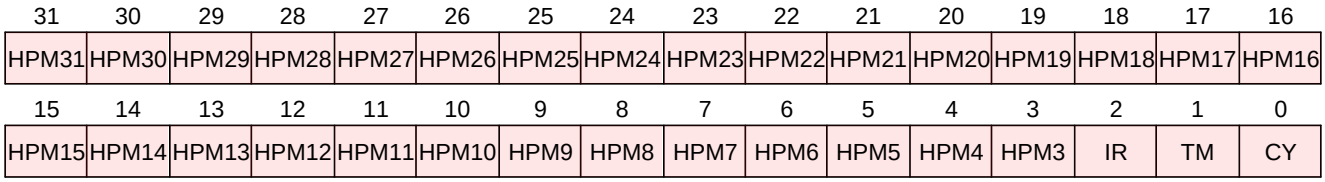


Figure 117. scounteren format

C.113. sepc

Supervisor Exception Program Counter

Written with the PC of an instruction on an exception or interrupt taken in (H)S-mode.

Also controls where the hart jumps on an exception return from (H)S-mode.

C.113.1. Attributes

CSR Address	0x141
Defining extension	• S, version >= 0
Length	64-bit
Privilege Mode	S

C.113.2. Format

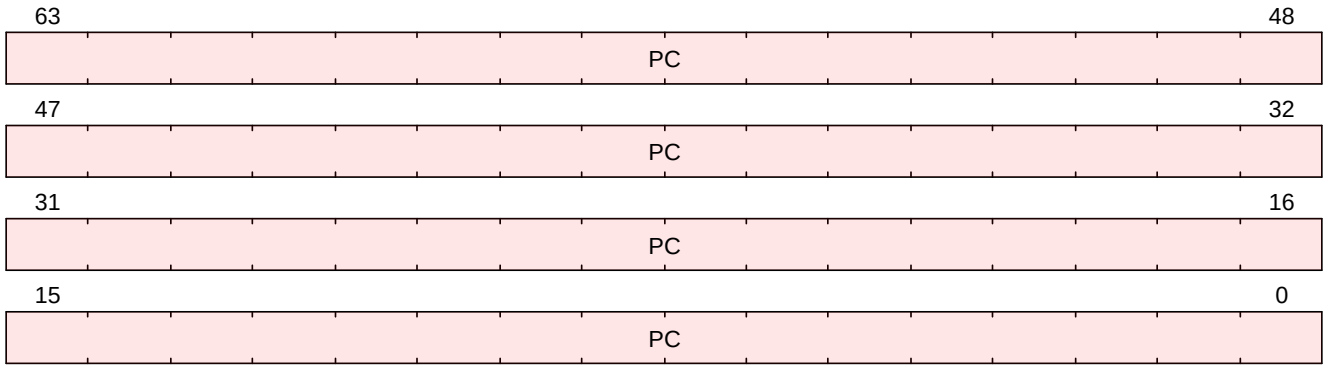


Figure 118. sepc format

C.114. sip

Supervisor Interrupt Pending

A restricted view of the interrupt pending bits in mip.

Hypervisor-related interrupts (VS-mode interrupts and Supervisor Guest interrupts) are not reflected in sip even though those interrupts can be taken in HS-mode. Instead, they are reported through `hip`.

C.114.1. Attributes

CSR Address	0x144
Defining extension	• S, version >= 0
Length	64-bit
Privilege Mode	S

C.114.2. Format

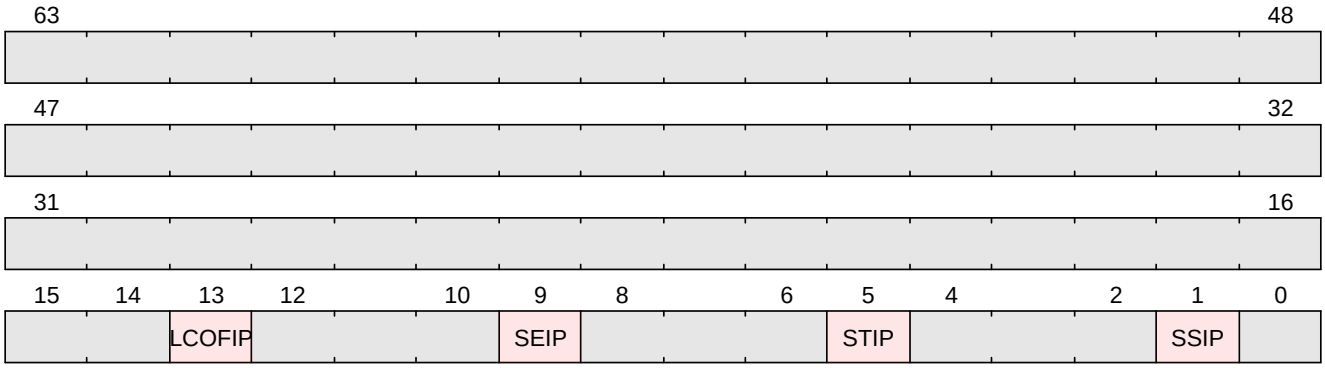


Figure 119. sip format

C.115. sscratch

Supervisor Scratch Register

Scratch register for software use. Bits are not interpreted by hardware.

C.115.1. Attributes

CSR Address	0x140
Defining extension	• S, version >= 0
Length	64-bit
Privilege Mode	S

C.115.2. Format

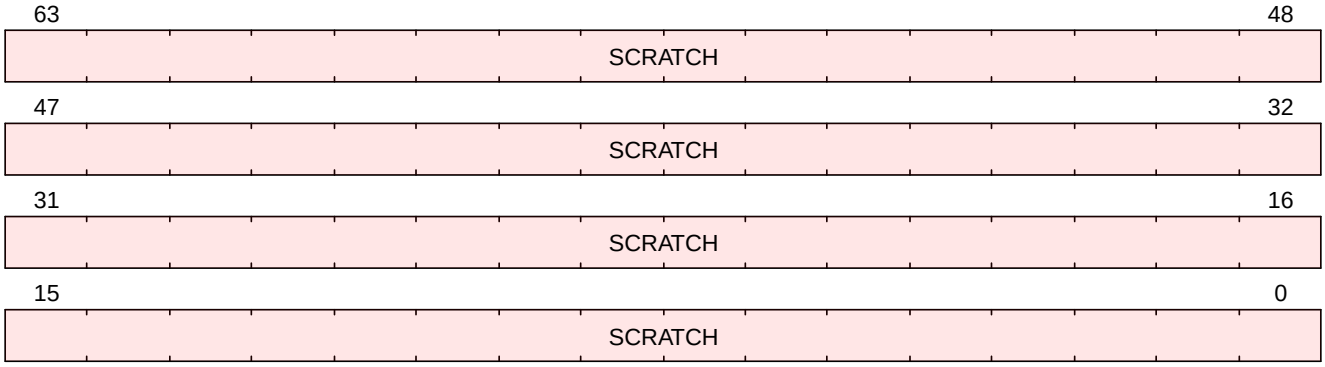


Figure 120. sscratch format

C.116. sstatus

Supervisor Status

The sstatus register tracks and controls the hart's current operating state.

All fields in sstatus are aliases of the same field in mstatus.

C.116.1. Attributes

CSR Address	0x100
Defining extension	• S, version >= 0
Length	32 when CSR[mstatus].SXL == 0 64 when CSR[mstatus].SXL == 1
Privilege Mode	S

C.116.2. Format

This CSR format changes dynamically with XLEN.

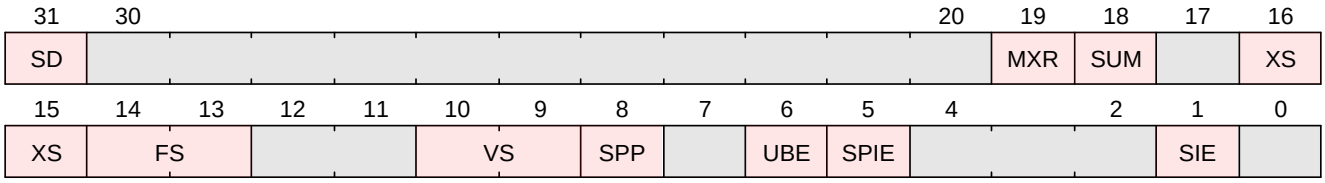


Figure 121. sstatus Format when CSR[mstatus].SXL == 0

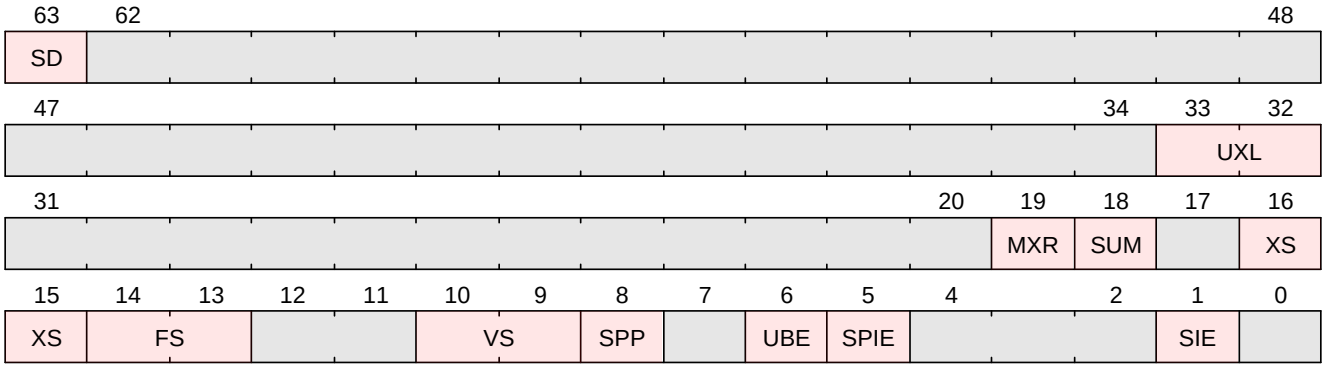


Figure 122. sstatus Format when CSR[mstatus].SXL == 1

C.117. stval

Supervisor Trap Value

Holds trap-specific information

C.117.1. Attributes

CSR Address	0x143
Defining extension	• S, version >= 0
Length	64-bit
Privilege Mode	S

C.117.2. Format

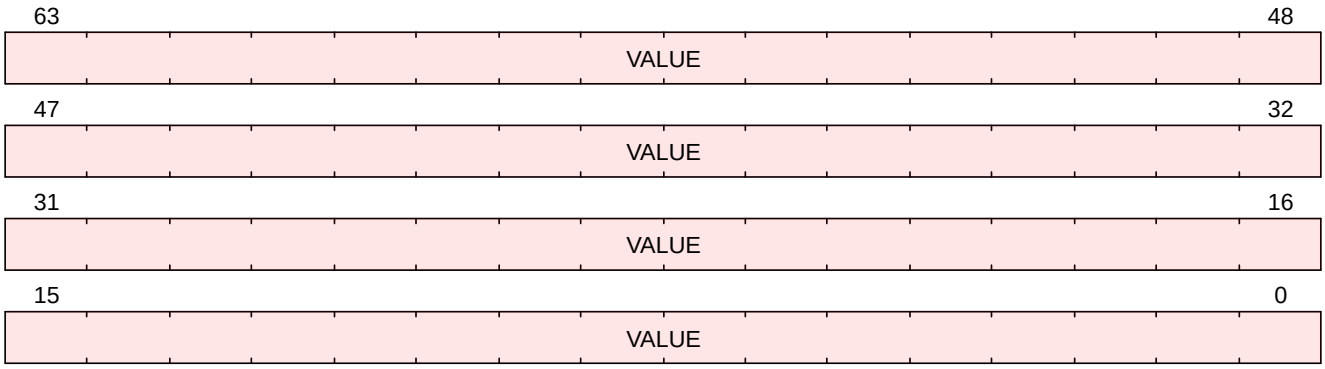


Figure 123. stval format

C.118. stvec

Supervisor Trap Vector

Controls where traps jump.

C.118.1. Attributes

CSR Address	0x105
Defining extension	• S, version >= 0
Length	64-bit
Privilege Mode	S

C.118.2. Format

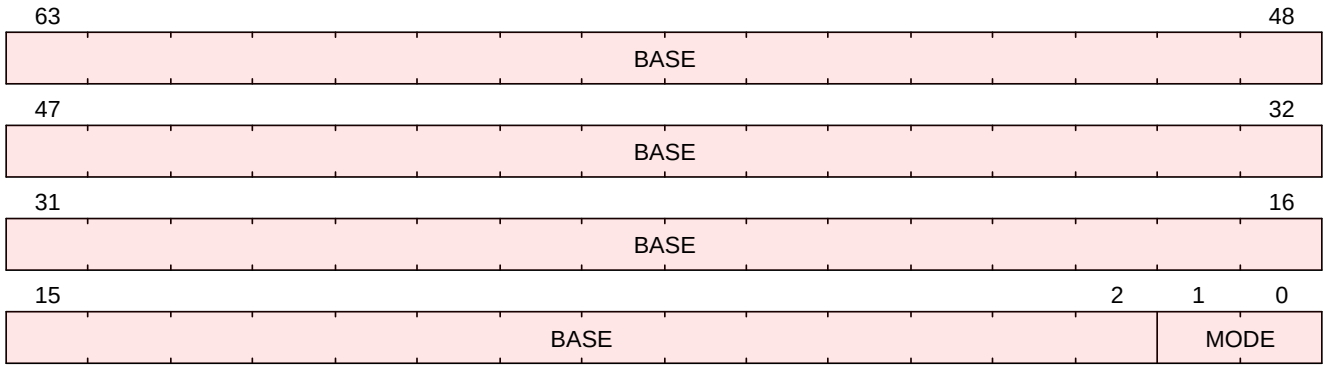


Figure 124. stvec format

C.119. time

Timer for RDTIME Instruction

This CSR does not exist, and access will cause an IllegalInstruction exception.

Shadow of the memory-mapped M-mode CSR `mtime`.

Privilege mode access is controlled with mcounteren.TM, scounteren.TM, and hcounteren.TM as follows:

mcounteren.TM	scounteren.TM	scounteren.TM	time behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	Illegal Instruction	Illegal Instruction	Illegal Instruction	Illegal Instruction
1	0	0	read-only	Illegal Instruction	Illegal Instruction	Illegal Instruction
1	1	0	read-only	read-only	Illegal Instruction	Illegal Instruction
1	0	1	read-only	Illegal Instruction	read-only	Illegal Instruction
1	1	1	read-only	read-only	read-only	read-only

C.119.1. Attributes

CSR Address	0xc01
Defining extension	• Zicntr, version >= 0
Length	64-bit
Privilege Mode	U

C.119.2. Format

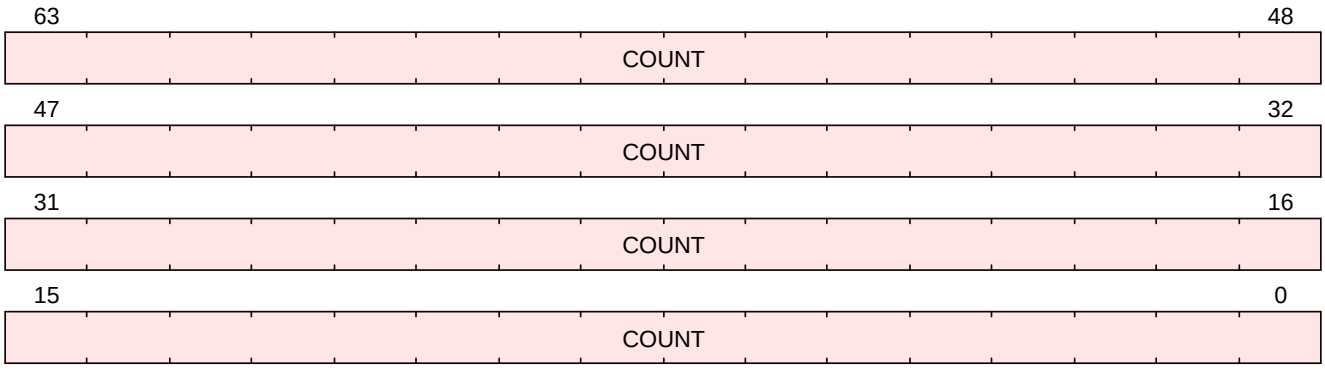


Figure 125. time format

C.120. timeh

High-half timer for RDTIME Instruction

This CSR does not exist, and access will cause an IllegalInstruction exception.

Shadow of the memory-mapped M-mode CSR `mtimeh`.

Privilege mode access is controlled with mcounteren.TM, scounteren.TM, and hcounteren.TM as follows:

mcounteren.TM	scounteren.TM	scounteren.TM	time behavior			
			S-mode	U-mode	VS-mode	VU-mode
0	-	-	Illegal Instruction	Illegal Instruction	Illegal Instruction	Illegal Instruction
1	0	0	read-only	Illegal Instruction	Illegal Instruction	Illegal Instruction
1	1	0	read-only	read-only	Illegal Instruction	Illegal Instruction
1	0	1	read-only	Illegal Instruction	read-only	Illegal Instruction
1	1	1	read-only	read-only	read-only	read-only

C.120.1. Attributes

CSR Address	0xc81
Defining extension	• Zicntr, version >= 0
Length	32-bit
Privilege Mode	U

C.120.2. Format

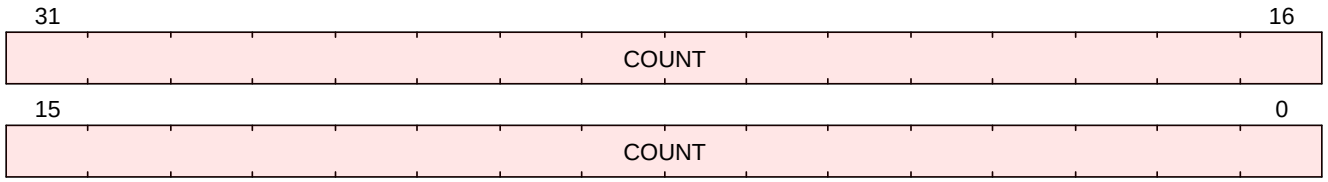


Figure 126. timeh format

C.121. vsatp

Virtual Supervisor Address Translation and Protection

The vsatp register is a VSXLEN-bit read/write register that is VS-mode's version of supervisor register satp. When V=1, vsatp substitutes for the usual satp, so instructions that normally read or modify satp actually access vsatp instead. vsatp controls VS-stage address translation, the first stage of two-stage translation for guest virtual addresses.

The vsatp register is considered active for the purposes of the address-translation algorithm unless the effective privilege mode is U and hstatus.HU=0. However, even when vsatp is active, VS-stage page-table entries' A bits must not be set as a result of speculative execution, unless the effective privilege mode is VS or VU.

In particular, virtual-machine load/store (`hlv`, 'hlvx', or 'hsv') instructions that are misspeculatively executed must not cause VS-stage A bits to be set.

When V=0, a write to vsatp with an unsupported MODE value is either ignored as it is for satp, or the fields of vsatp are treated as WARL in the normal way. However, when V=1, a write to satp with an unsupported MODE value is ignored and no write to vsatp is effected.

C.121.1. Attributes

CSR Address	0x280
Virtual CSR Address	0x180
Defining extension	• H, version >= 0
Length	32 when CSR[hstatus].VSXL == 0 64 when CSR[hstatus].VSXL == 1
Privilege Mode	VS

C.121.2. Format

This CSR format changes dynamically with XLEN.

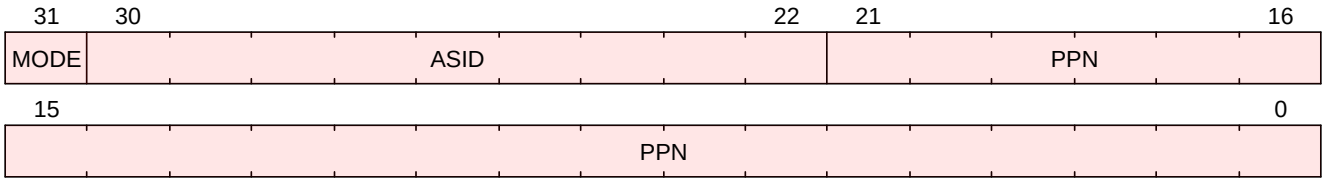


Figure 127. vsatp Format when CSR[hstatus].VSXL == 0

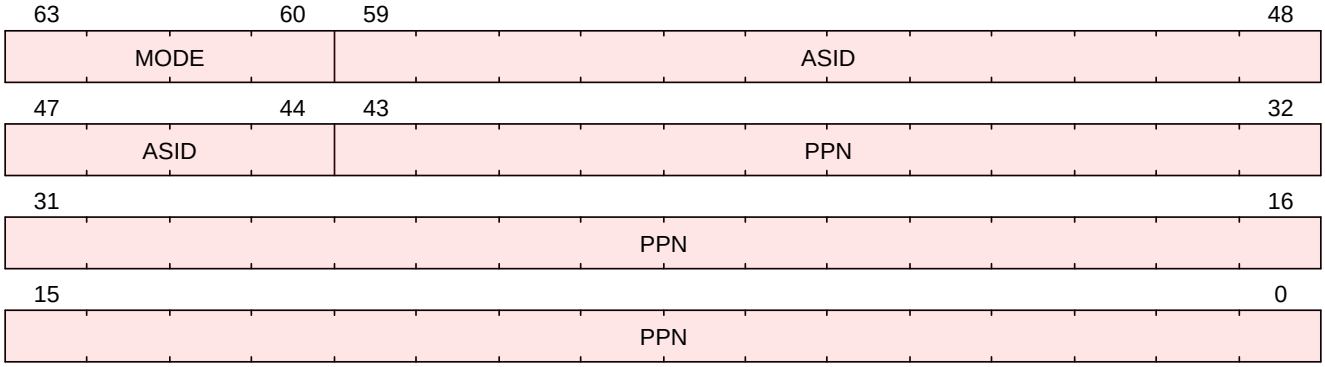


Figure 128. vsatp Format when CSR[hstatus].VSXL == 1

C.122. vscause

Virtual Supervisor Cause

Reports the cause of the latest exception taken in VS-mode.

C.122.1. Attributes

CSR Address	0x242
Virtual CSR Address	0x142
Defining extension	• H, version >= 0
Length	32 when CSR[hstatus].VSXL == 0 64 when CSR[hstatus].VSXL == 1
Privilege Mode	VS

C.122.2. Format

This CSR format changes dynamically with XLEN.

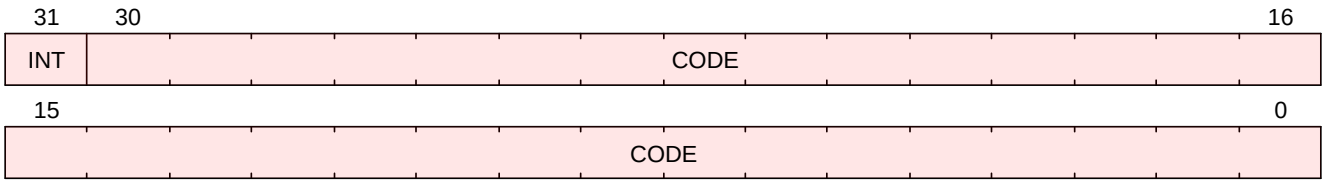


Figure 129. vscause Format when CSR[hstatus].VSXL == 0

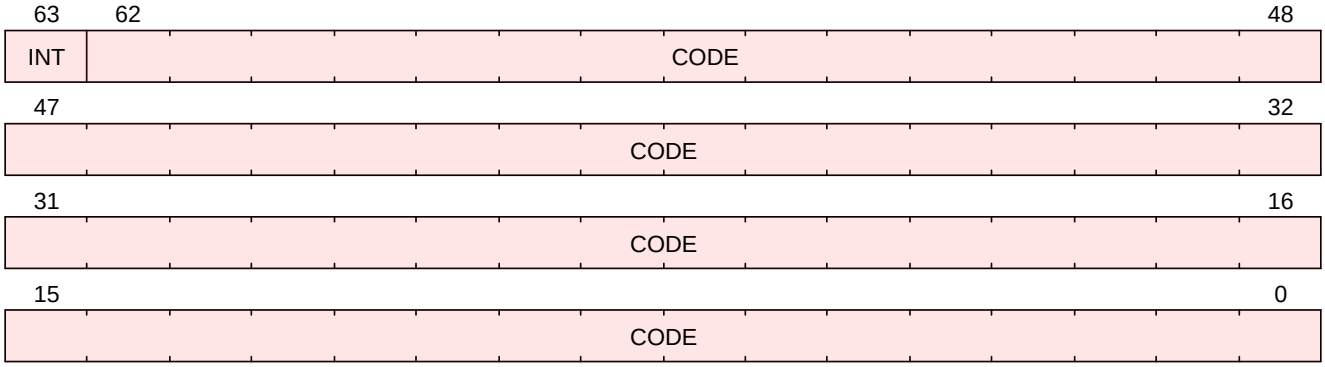


Figure 130. vscause Format when CSR[hstatus].VSXL == 1

C.123. vsepc

Virtual Supervisor Exception Program Counter

Written with the PC of an instruction on an exception or interrupt taken in VS-mode.

Also controls where the hart jumps on an exception return from VS-mode.

C.123.1. Attributes

CSR Address	0x241
Virtual CSR Address	0x141
Defining extension	• H, version >= 0
Length	64-bit
Privilege Mode	VS

C.123.2. Format

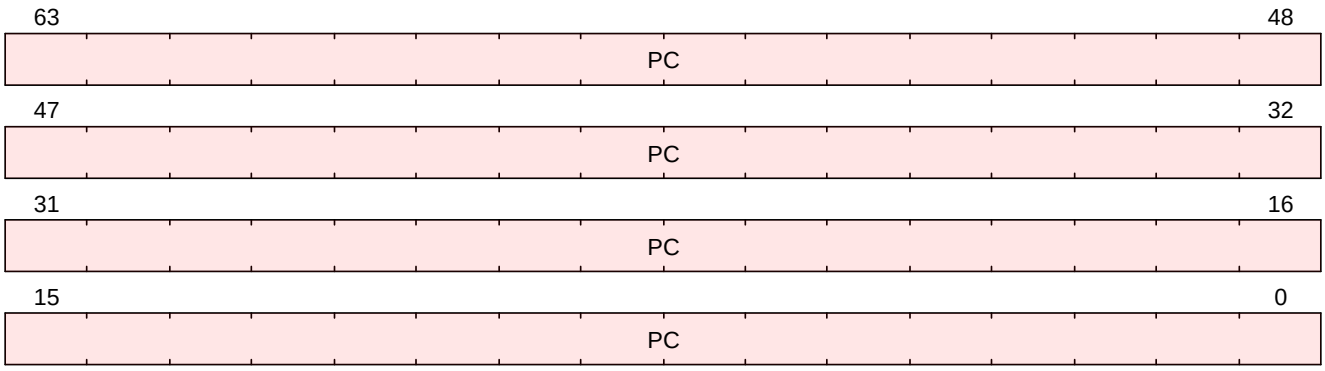


Figure 131. vsepc format

C.124. vsstatus

Virtual Supervisor Status

The vsstatus register tracks and controls the hart's current operating state.

It is VS-mode's version of sstatus, and substitutes for it when in VS-mode (*i.e.*, in VS-mode CSR address 0x100 is vsstatus, not sstatus).

Unlike the relationship between sstatus and mstatus, none of the bits in vsstatus are aliases of another field.

C.124.1. Attributes

CSR Address	0x200
Virtual CSR Address	0x100
Defining extension	• H, version >= 0
Length	32 when CSR[hstatus].VSXL == 0 64 when CSR[hstatus].VSXL == 1
Privilege Mode	VS

C.124.2. Format

This CSR format changes dynamically with XLEN.

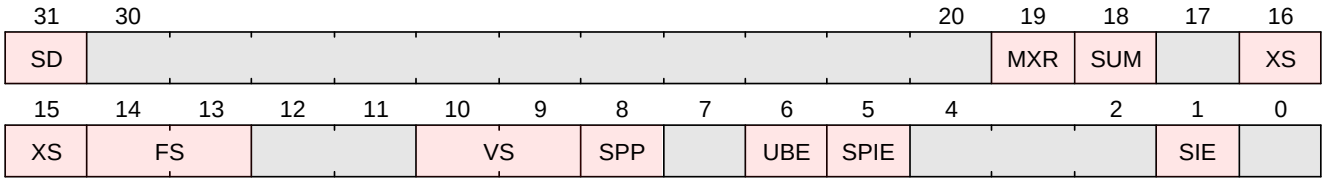


Figure 132. vsstatus Format when CSR[hstatus].VSXL == 0

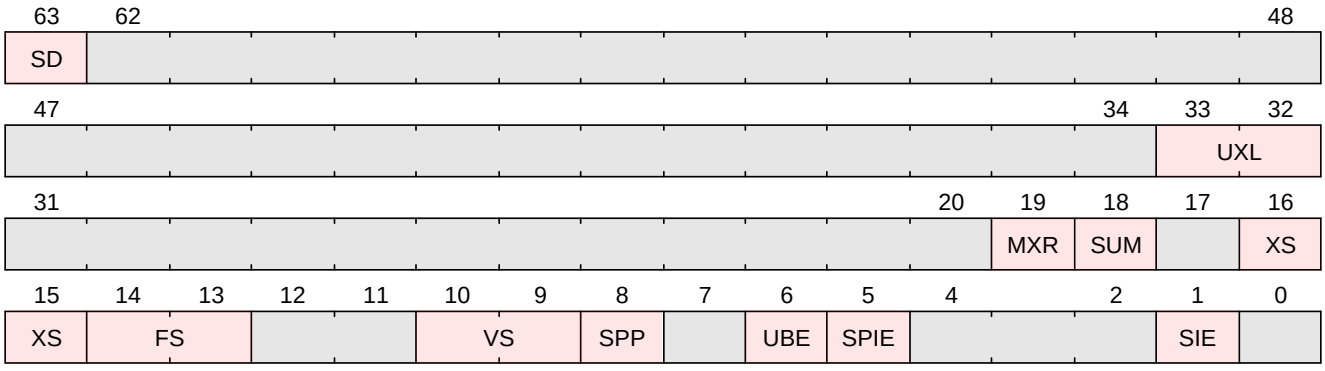


Figure 133. vsstatus Format when CSR[hstatus].VSXL == 1

C.125. vstval

Virtual supervisor Trap Value

Holds trap-specific information

C.125.1. Attributes

CSR Address	0x243
Defining extension	• H, version >= 0
Length	32 when CSR[hstatus].VSXL == 0 64 when CSR[hstatus].VSXL == 1
Privilege Mode	S

C.125.2. Format

This CSR format changes dynamically with XLEN.

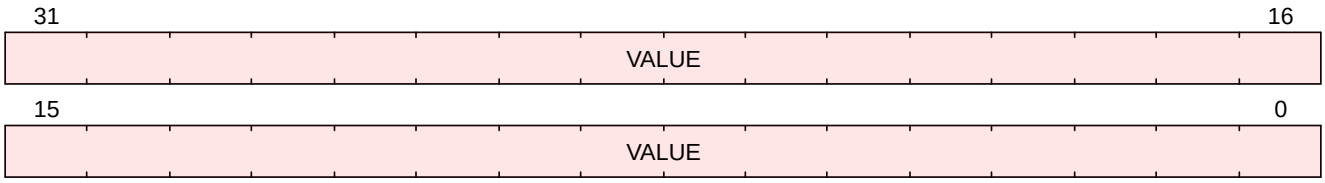


Figure 134. vstval Format when CSR[hstatus].VSXL == 0

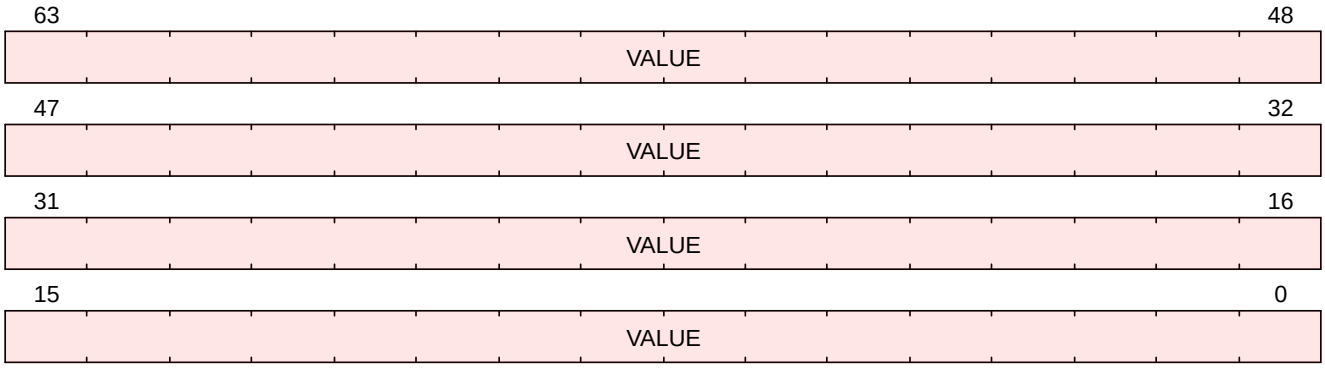


Figure 135. vstval Format when CSR[hstatus].VSXL == 1

C.126. vstvec

Supervisor Trap Vector

Controls where traps jump.

C.126.1. Attributes

CSR Address	0x205
Defining extension	• H, version >= 0
Length	64-bit
Privilege Mode	S

C.126.2. Format

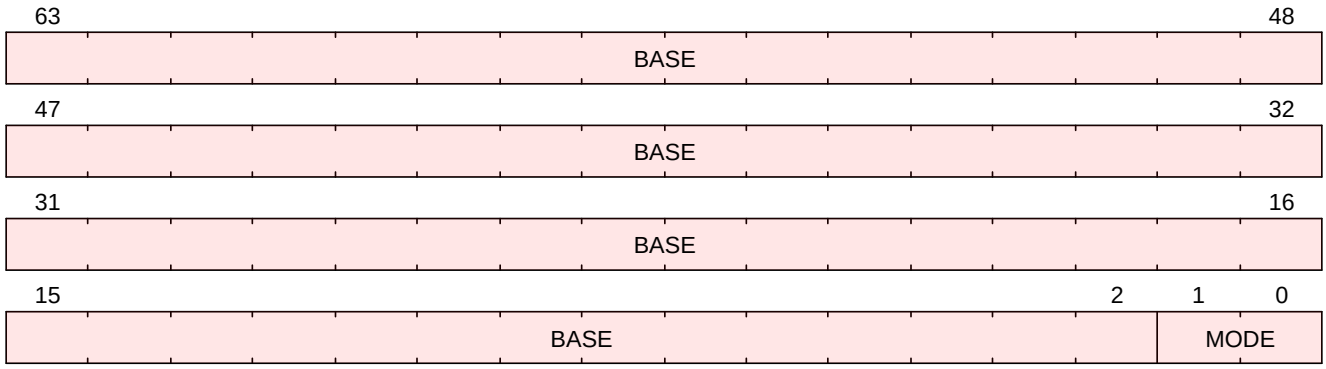


Figure 136. vstvec format