SYNOPSYS® 新思

RISC-V Unified Database

The Centralized Source of Truth

Afonso Oliveira

July 18th, 2025

Agenda

- Challenges with current RISC-V Specification
- RISC-V Unified Database (UDB)
- Get involved

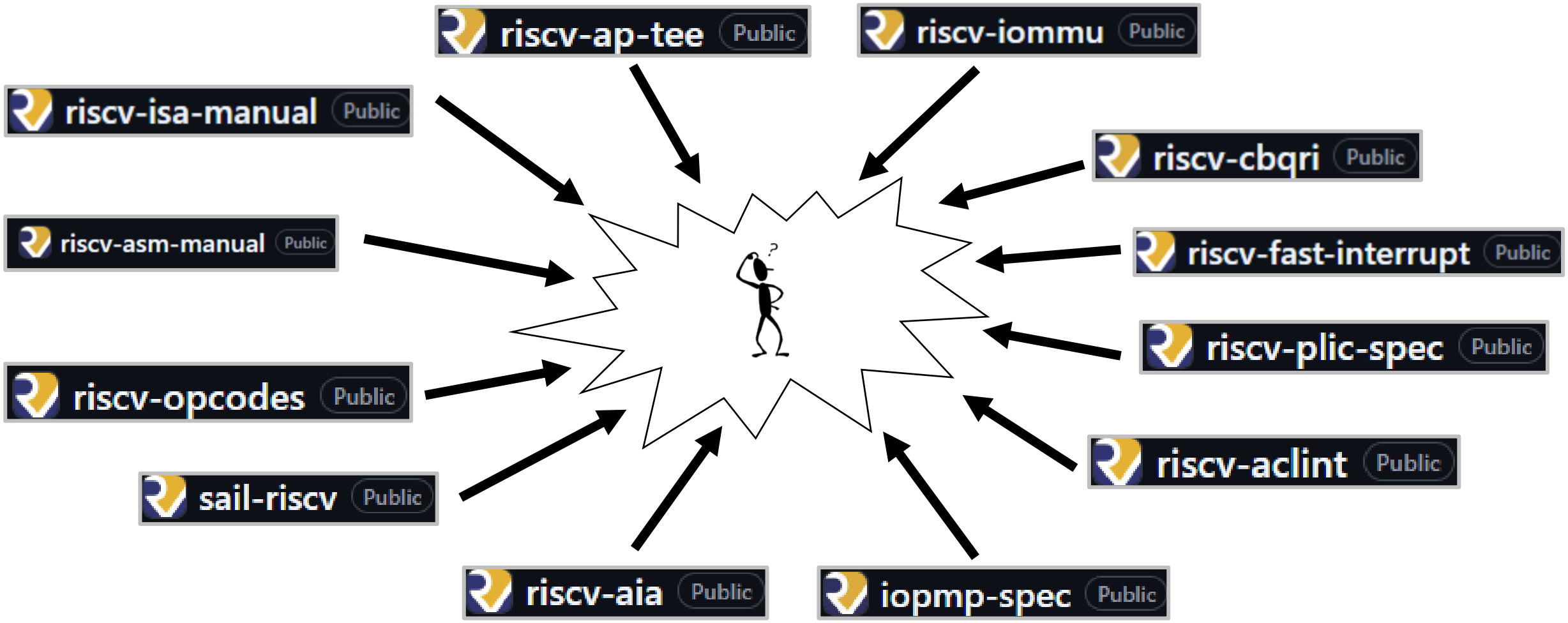
Growth Challenges with the RISC-V Specification

Why this matters

RISC-V ecosystem relies on multiple disconnected specifications

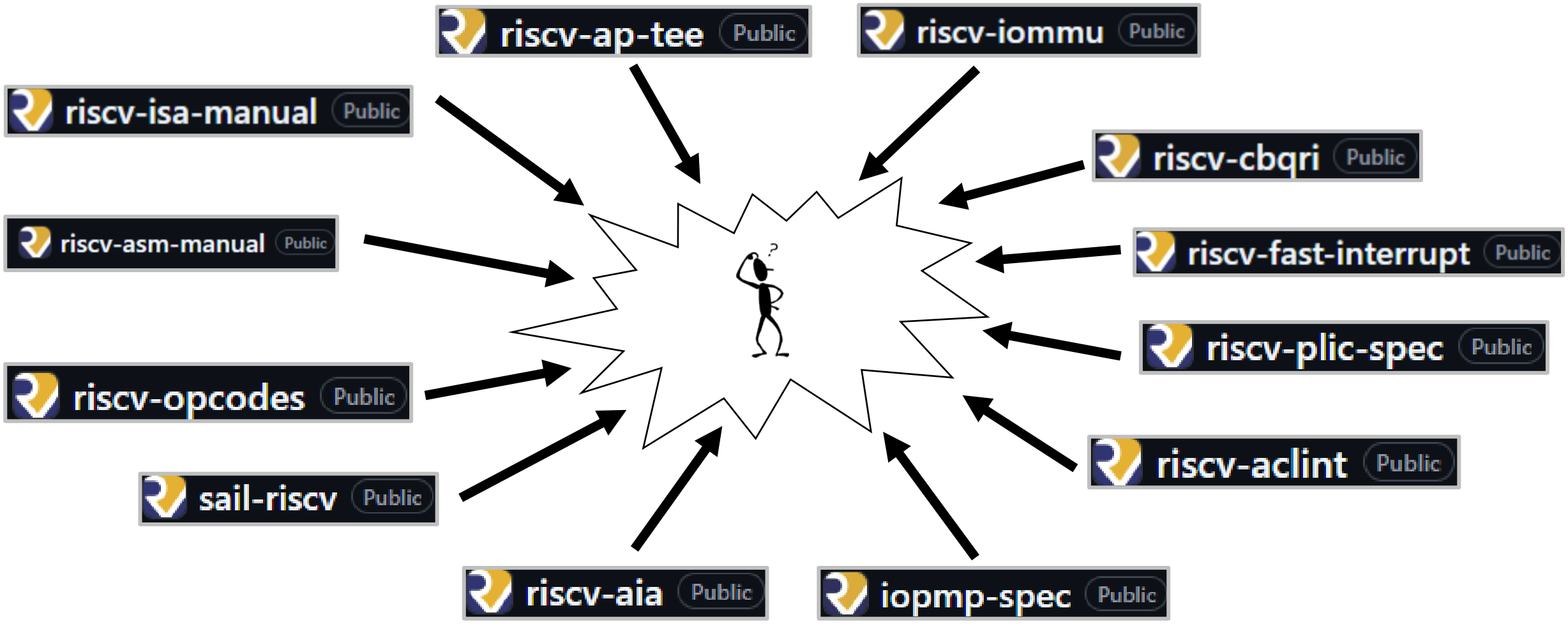
Why this matters

RISC-V ecosystem relies on multiple disconnected specifications



Why this matters

RISC-V ecosystem relies on multiple disconnected specifications



Why this matters

RISC-V ecosystem relies on multiple disconnected specifications

- Fully text - asciidoc
- How do we verify?
- How to diversify? PRMs, TRMs …

- Format doesn’t allow all pseudo-instructions
- Doesn’t allow to easily overlay implementation specific matters
- How many tests can you really run on a simplistic format?

https://github.com/riscv/riscv-isa-manual
https://github.com/riscv/riscv-opcodes

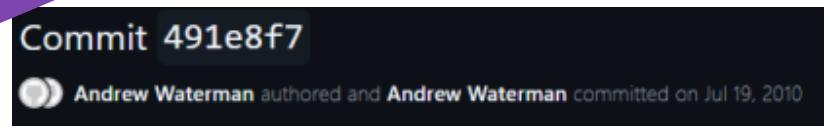
Why this matters

RISC-V ecosystem relies on multiple disconnected specifications

- First published in 2
- 10 yea
- 10 -

nsions

extensions



ored in 2022 – some previous problems stayed

- From 5 to 200 extensions

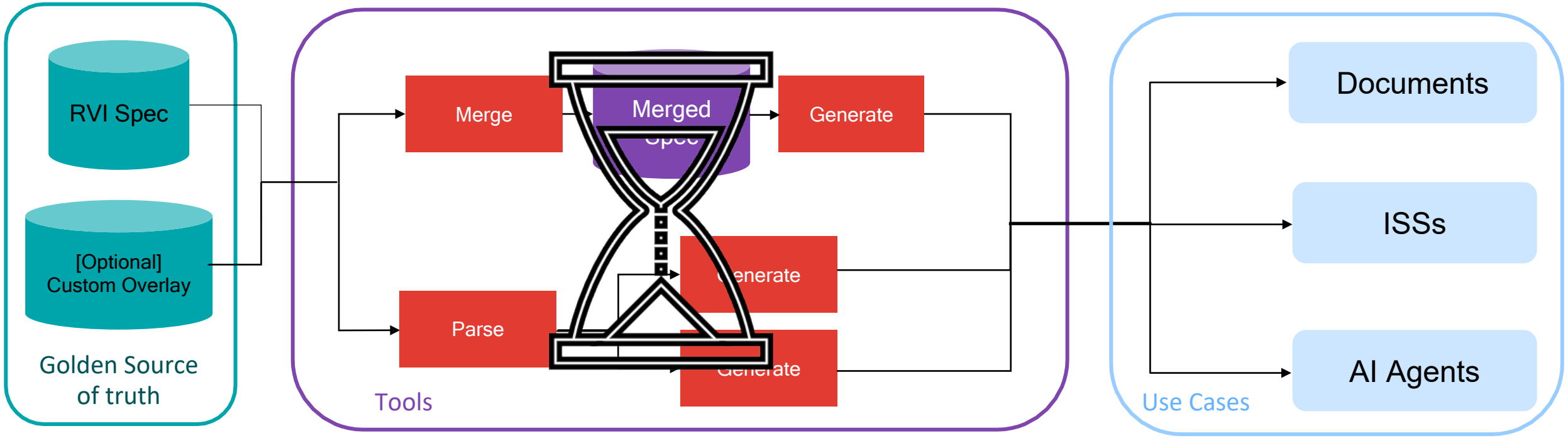
Not a RISC-V problem –
Infrastructure was outgrown

https://github.com/riscv/riscv-isa-manual
https://github.com/riscv/riscv-opcodes

RISC-V Unified Database (UDB)

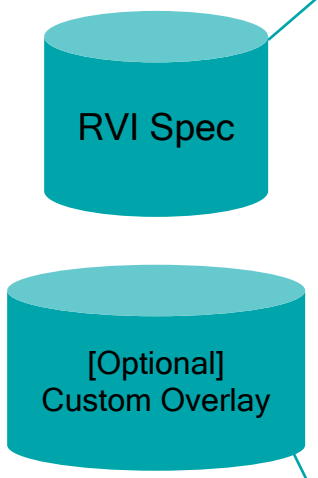
What is the UDB?

A centralized, machine readable, source of truth
An efficient tool to generate several outputs



Adapted from Derek Hower's ARC Presentation

Golden Source of Truth



	Instructions	CSRs	Profiles	Architectural Parameters
Definition	Operations executed by the processor	Special-purpose state management registers	Subsets of the ISA for specific use cases	Customizable implementation attributes
Purpose	Arithmetic, logic, memory, control	System config, monitoring, exceptions	Compatibility across implementations	Tailored design
Examples	ADD, SUB, LW	mstatus, mtvec, mip	RV32A, RV64G	32/64-bit, trap handling

Golden Source of Truth

Instructions

- Name
- Long name
- Description
- Defined by
  - Extensions
- Encoding
  - Match
  - variables
- Access Mode
- Formal Specification (Sail and IDL)

```yaml
kind: instruction
name: add
long_name: Integer add
description: |
  Add the value in rs1 to rs2, and store the
  result in rd. Any overflow is thrown away.
definedBy: I
assembly: xd, xs1, xs2
encoding:
  match: 0000000----------000-----0110011
  variables:
    - name: rs2
      location: 24-20
    - name: rs1
      location: 19-15
    - name: rd
      location: 11-7
access:
  s: always
  u: always
  vs: always
  vu: always
operation(): X[rd] = X[rs1] + X[rs2];

sail(): |
  {
    let rs1_val = X(rs1);
    let rs2_val = X(rs2);
    let result : xlenbits = match op {
      RISCV_ADD  => rs1_val + rs2_val,
    };
    X(rd) = result;
    RETIRE_SUCCESS
  }
```

https://github.com/riscv-software-src/riscv-unified-db/blob/main/arch/inst/I/add.yaml

Current use cases

Used by the Certification Steering Commitee SIG to create certification documents

Qualcomm created Xqci: an extension that is only available through the UDB

Synopsys is using the UDB to generate ARC-V PRM

Substitute RISC-V Opcodes

Target use-cases

Community endorsement

UDB is being collaboratively developed

- Several engineers from different companies are actively developing and meeting weekly
  - Akeana, Qualcomm, Renode, Red Hat, Rivos, Sifive, Synopsys, Ventana
- We had 7 RVI mentees working on UDB! – under official LFX/RVI mentorship platform
- UDB is a RISC-V official SIG!
  - UDB is engaging with other Working Groups – such as the Doc SIG, TSC, CSC

Join us!

- Check our github repository https://github.com/riscv-software-src/riscv-unified-db/
  - Several issues and discussions on going!
  - Ask questions!
- On our weekly meetings – under LFX/RVI calendar – click here
- Contact me Afonso.Oliveira@synopsys.com

SYNOPSYS® 新思

Thank you

SYNOPSYS® · 新思

Backup slides

Golden Source of Truth

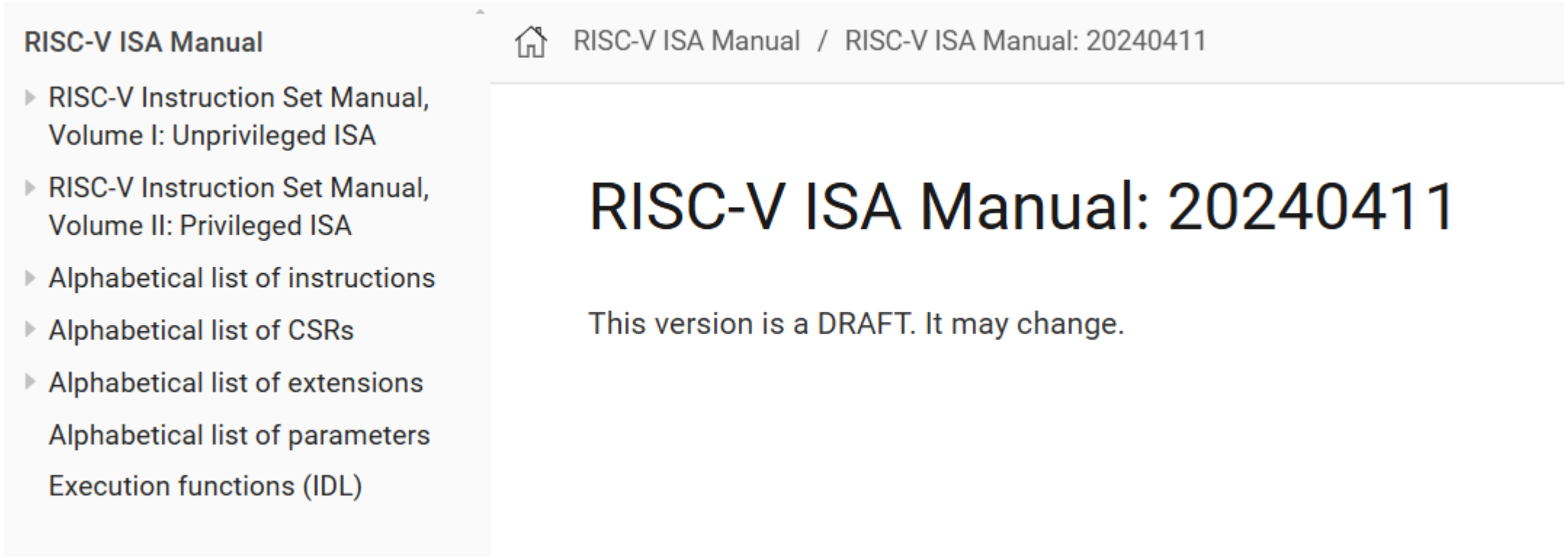
Architectural Parameters

- Name
- Mode
  - Privileged or unprivileged
- Base
  - 32bit vs 64bit
- Release
- Introduction
  - Plain text
- Extensions included
  - Optional or Mandatory
  - Versioned

https://github.com/riscv-software-src/riscv-unified-db/blob/main/arch/profile/RVI20U32.yaml

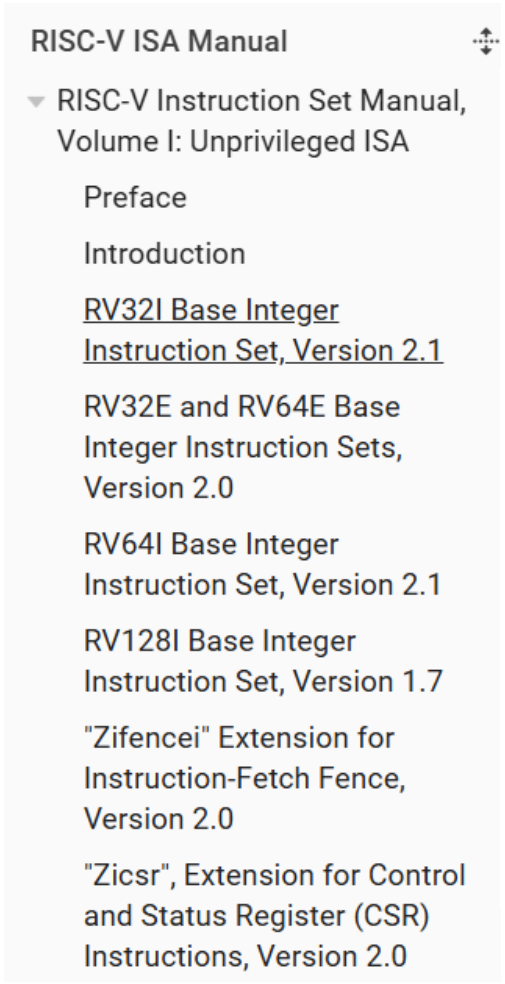
```yaml
kind: profile
name: RVI20U32
marketing_name: RVI20U32
mode: Unpriv
base: 32
release: { $ref: profile_release/RVI20.yaml# }
introduction: |
  This profile specifies the ISA features available to generic
  unprivileged execution environments.

extensions:
  I:
    presence: mandatory
    version: "~> 2.1"
    note: |
      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.
  A:
    presence: optional
    version: "= 2.1"
  C:
    presence: optional
    version: "= 2.2“
# list of extensions continues. See github for full list

recommendations:
  - text: |
      Implementations are strongly recommended to raise illegal-instruction
      exceptions on attempts to execute unimplemented opcodes.
```

ISA Manual



https://riscv-software-src.github.io/riscv-unified-db/manual/html/isa/20240411/index.html

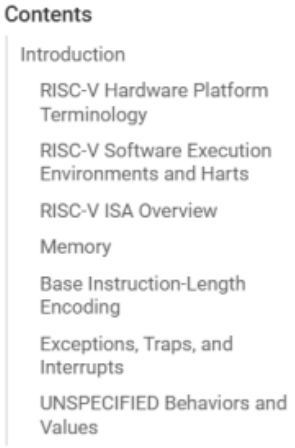
ISA Manual



Introduction

RISC-V (pronounced "risk-five") is a new instruction-set architecture (ISA) that was originally designed to support computer architecture research and education, but which we now hope will also become a standard free and open architecture for industry implementations. Our goals in defining RISC-V include:

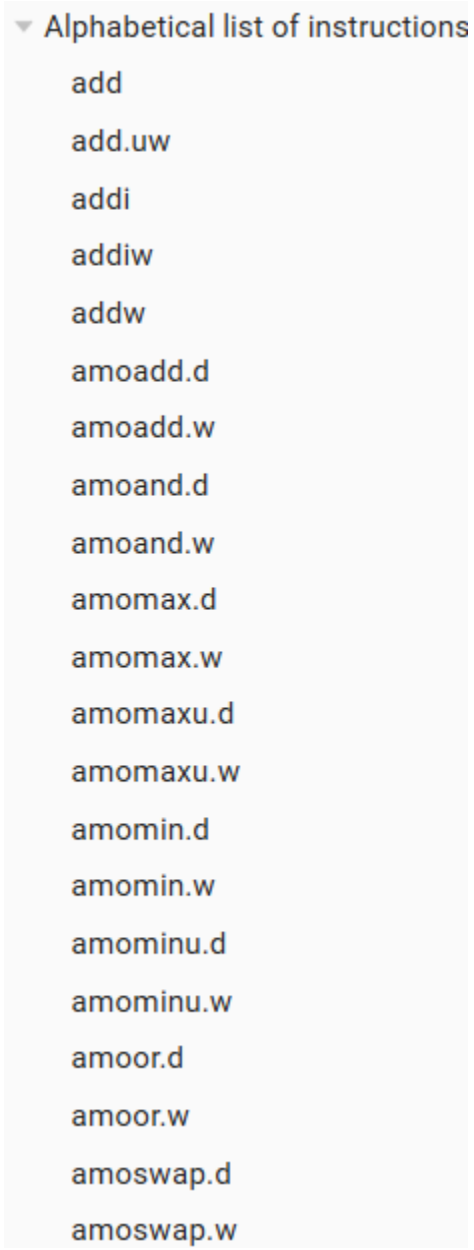
- A completely *open* ISA that is freely available to academia and industry.
- A *real* ISA suitable for direct native hardware implementation, not just simulation or binary translation.
- An ISA that avoids "over-architecting" for a particular microarchitecture style (e.g., microcoded, in-order, decoupled, out-of-order) or implementation technology (e.g., full-custom, ASIC, FPGA), but which allows efficient implementation in any of these.
- An ISA separated into a *small* base integer ISA, usable by itself as a base for customized accelerators or for educational purposes, and optional standard extensions, to support general-purpose software development.
- Support for the revised 2008 IEEE-754 floating-point standard. cite:[ieee754-2008]
- An ISA supporting extensive ISA extensions and specialized variants.
- Both 32-bit and 64-bit address space variants for applications, operating system kernels, and hardware implementations.

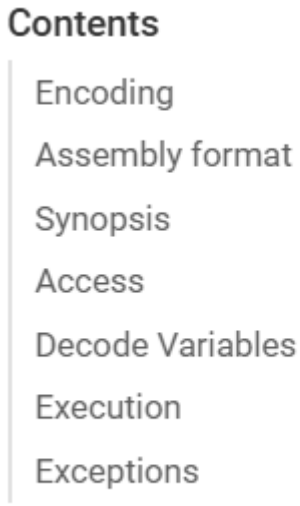


https://riscv-software-src.github.io/riscv-unified-db/manual/html/isa/20240411/index.html

List of Instructions

- **All** RISC-V instructions
  - 70% of pseudo-instructions (WiP)
- Versioned (Future work)
- Detailed descriptions
  - Encoding
  - Assembly format
  - Synopsis
  - Access
  - Decode Variables
  - Formal Description (Sail and IDL)
  - Exception





https://riscv-software-src.github.io/riscv-unified-db/manual/html/isa/20240411/insts/add.html

List of Instructions

add

Integer add

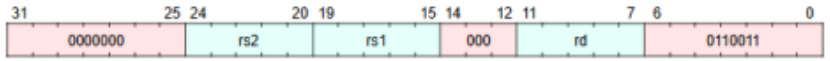
This instruction is defined by:

- I, version >= 0

This instruction is included in the following profiles:

- MockProfile 64-bit Unpriv (Mandatory)
- MockProfile 64-bit S-mode (Mandatory)
- RVA20U64 (Mandatory)
- RVA22U64 (Mandatory)
- RVI20U32 (Mandatory)
- RVI20U64 (Mandatory)

Encoding



Assembly format

```
add rd, rs1, rs2
```

Synopsis

IMPORTANT

This instruction must have data-independent timing when extension Zkt is enabled.

Access

M	HS	U	VS	VU
Always	Always	Always	Always	Always

Decode Variables

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

Execution

IDL | Sail

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

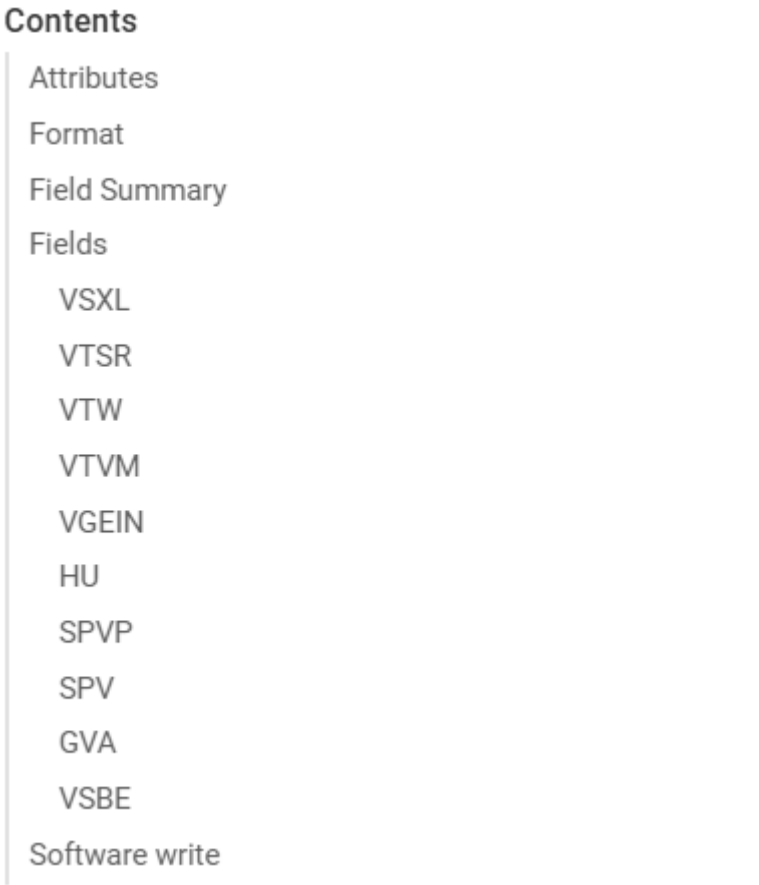
IDL | Sail

```
{
  let rs1_val = X(rs1);
  let rs2_val = X(rs2);
  let result : xlenbits = match op {
    RISCV_ADD  => rs1_val + rs2_val,
    RISCV_SLT  => zero_extend(bool_to_bits(rs1_val <_s rs2_val)),
    RISCV_SLTU => zero_extend(bool_to_bits(rs1_val <_u rs2_val)),
    RISCV_AND  => rs1_val & rs2_val,
    RISCV_OR   => rs1_val | rs2_val,
    RISCV_XOR  => rs1_val ^ rs2_val,
    RISCV_SLL  => if   sizeof(xlen) == 32
                  then rs1_val << (rs2_val[4..0])
                  else rs1_val << (rs2_val[5..0]),
    RISCV_SRL  => if   sizeof(xlen) == 32
                  then rs1_val >> (rs2_val[4..0])
                  else rs1_val >> (rs2_val[5..0]),
    RISCV_SUB  => rs1_val - rs2_val,
    RISCV_SRA  => if   sizeof(xlen) == 32
                  then shift_right_arith32(rs1_val, rs2_val[4..0])
                  else shift_right_arith64(rs1_val, rs2_val[5..0])
  };
  X(rd) = result;
  RETIRE_SUCCESS
}
```

https://riscv-software-src.github.io/riscv-unified-db/manual/html/isa/20240411/insts/add.html

List of CSRs

- Detailed descriptions
  - Acess Modes
  - Attibute List
  - Format
  - Field summary and descrption
  - Formal description (IDL)



https://riscv-software-src.github.io/riscv-unified-db/manual/html/isa/20240411/csrs/cycle.html

List of CSRs

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

Attributes

Defining Extension	• Zicntr, version >= 0
CSR Address	0xc00
Length	64-bit
Privilege Mode	U

Format

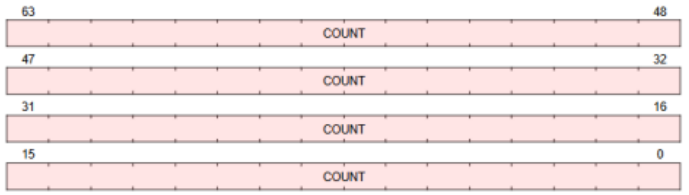


Figure 1. cycle format

Field Summary

Name	Location	Type	Reset Value
COUNT	63:0	RO-H	UNDEFINED_LEGAL

Fields

COUNT

Location
63:0

Description
Alias of mcycle.COUNT.

Type
RO-H

Reset value
UNDEFINED_LEGAL

Software read

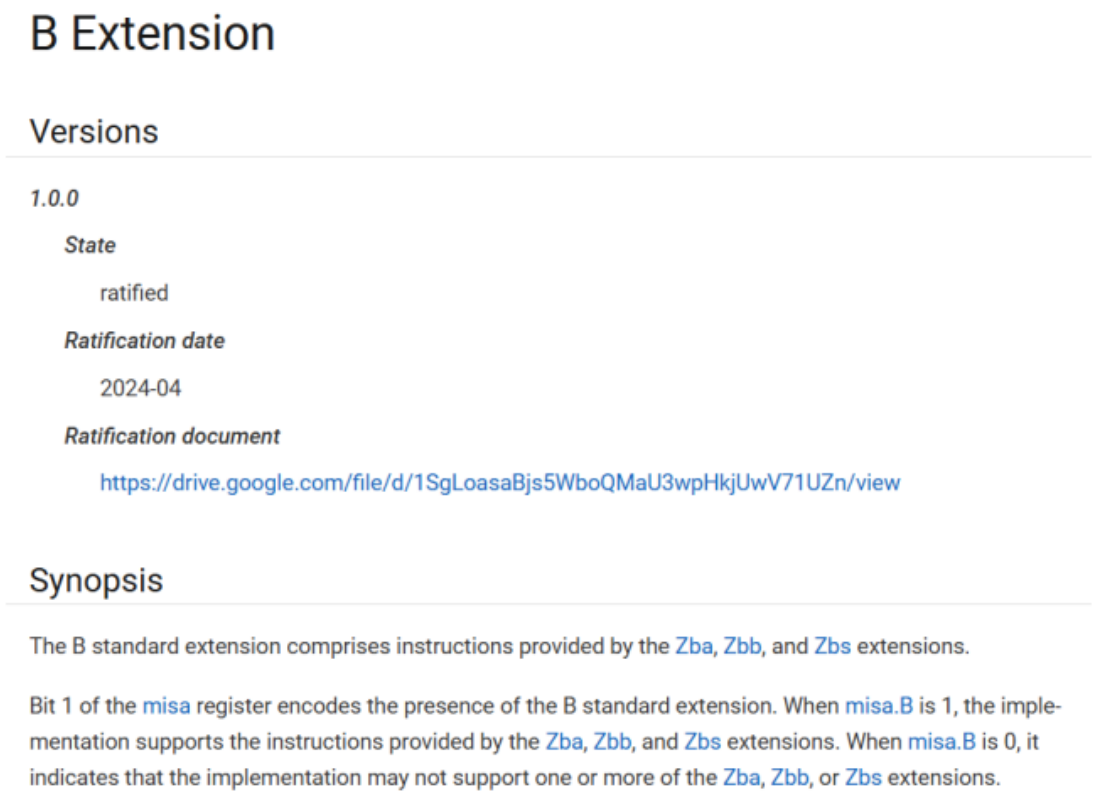
This CSR may return a value that is different from what is stored in hardware.

```
if (mode() == PrivilegeMode::S) {
  if (CSR[mcounteren].CY == 1'b0) {
    raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
  }
} else if (mode() == PrivilegeMode::U) {
  if (CSR[misa].S == 1'b1) {
    if ((CSR[mcounteren].CY & CSR[scounteren].CY) == 1'b0) {
      raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
    }
  } else if (CSR[mcounteren].CY == 1'b0) {
    raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
  }
} else if (mode() == PrivilegeMode::VS) {
  if (CSR[hcounteren].CY == 1'b0 && CSR[mcounteren] == 1'b1) {
    raise(ExceptionCode::VirtualInstruction, mode(), $encoding);
  } else if (CSR[mcounteren].CY == 1'b0) {
    raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
  }
} else if (mode() == PrivilegeMode::VU) {
  if (CSR[hcounteren].CY & CSR[scounteren].CY) == 1'b0) && (CSR[mcounteren].CY ==
    raise(ExceptionCode::VirtualInstruction, mode(), $encoding);
  } else if (CSR[mcounteren].CY == 1'b0) {
    raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
  }
}
return read_mcycle();
```

https://riscv-software-src.github.io/riscv-unified-db/manual/html/isa/20240411/csrs/cycle.html

List of Extensions

- Versions
- Synopsis
- List of Instructions
  - Another View
- List of Parameters



B Extension

Versions

1.0.0

State

ratified

Ratification date

2024-04

Ratification document

https://drive.google.com/file/d/1SgLoasaBjs5WboQMaU3wpHkjUwV71UZn/view

Synopsis

The B standard extension comprises instructions provided by the Zba, Zbb, and Zbs extensions.

Bit 1 of the misa register encodes the presence of the B standard extension. When misa.B is 1, the implementation supports the instructions provided by the Zba, Zbb, and Zbs extensions. When misa.B is 0, it indicates that the implementation may not support one or more of the Zba, Zbb, or Zbs extensions.

https://riscv-software-src.github.io/riscv-unified-db/manual/html/isa/20240411/exts/A.html

List of Extensions

Instructions

The following instructions are defined by this extension:

add.uw	Add unsigned word
andn	AND with inverted operand
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)
brev8	No synopsis available.
bset	Single-Bit set (Register)
bseti	Single-Bit set (Immediate)
clmul	Carry-less multiply (low-part)
clmulh	Carry-less multiply (high-part)
clmulr	Carry-less multiply (reversed)
clz	Count leading zero bits
clzw	Count leading zero bits in word
cpop	Count set bits
cpopw	Count set bits in word

Parameters

This extension has the following implementation options:

MUTABLE_MISA_B

Type	boolean
Valid Values	boolean
Description	Indicates whether or not the B extension can be disabled with the misa.B bit.

https://riscv-software-src.github.io/riscv-unified-db/manual/html/isa/20240411/exts/A.html

List of Parameters

- Table of ALL Parameters (WiP)
  - Name
  - Type
  - Extension
  - Description

Architectural Parameters

The following 144 parameters are defined in this manual:

Name	Type	Extension(s)	Description
ARCH_ID	64-bit integer	Sm	Vendor-specific architecture ID in marchid
ASID_WIDTH	0 to 16	S	Number of implemented ASID bits. Maximum is 16 for XLEN==64, and 9 for XLEN==32
CONFIG_PTR_ADDRESS	integer	Sm	Physical address of the unified discovery configuration data structure. This address is reported in the mconfig-ptr CSR.
COUNTINHIBIT_EN	32-element array where: [0] is boolean [1] is false [2] is boolean additional items are: boolean	Smhpm	Indicates which hardware performance monitor counters can be disabled from mcountinhibit. An unimplemented counter cannot be specified, i.e., if HPM_COUNTER_EN[3] is false, it would be illegal to set COUNTINHIBIT_EN[3] to true. COUNTINHIBIT_EN[1] can never be true, since it corresponds to mcountinhibit, which is always read-only-0. COUNTINHIBIT_EN[3:31] must all be false if Zihpm is not implemented.

https://riscv-software-src.github.io/riscv-unified-db/manual/html/isa/20240411/params/param_list.html

RISC-V Opcodes

- Over simplistic representation
  - Format only specifies encodings
- No versions
- Does not allow for swapped arguments
  - e.g. gt to lt pseudo instructions can not be described

```
                        blt   bimm12hi rs1 rs2 bimm12lo 14..12=4 6..2=0x18 1..0=3
$pseudo_op rv_i::blt    bgt   bimm12hi rs2 rs1 bimm12lo 14..12=4 6..2=0x18 1..0=3
```

Original Pseudo Variable Fields Fixed Fields

- Arguments with fixed values are not supported
  - Assigning variable fields is not possible

```
$pseudo_op rv_v::vmxor.mm   vmclr.m   31..26=0x1b 25=1  vs2=vd  vs1=vd  14..12=0x2 vd 6..0=0x57
```

SYNOPSYS® 新思

Thank you