RISC-V Unified Database

Streamlining the Ecosystem with a Centralized Source of Truth

Afonso Oliveira

February 1, 2025

About

- Master’s student in Universidade do Minho
- Software engineer at SYNOPSYS®
- Active contributor in UDB
  - Responsible for fetching instructions data
  - Responsible for data validation against outside sources

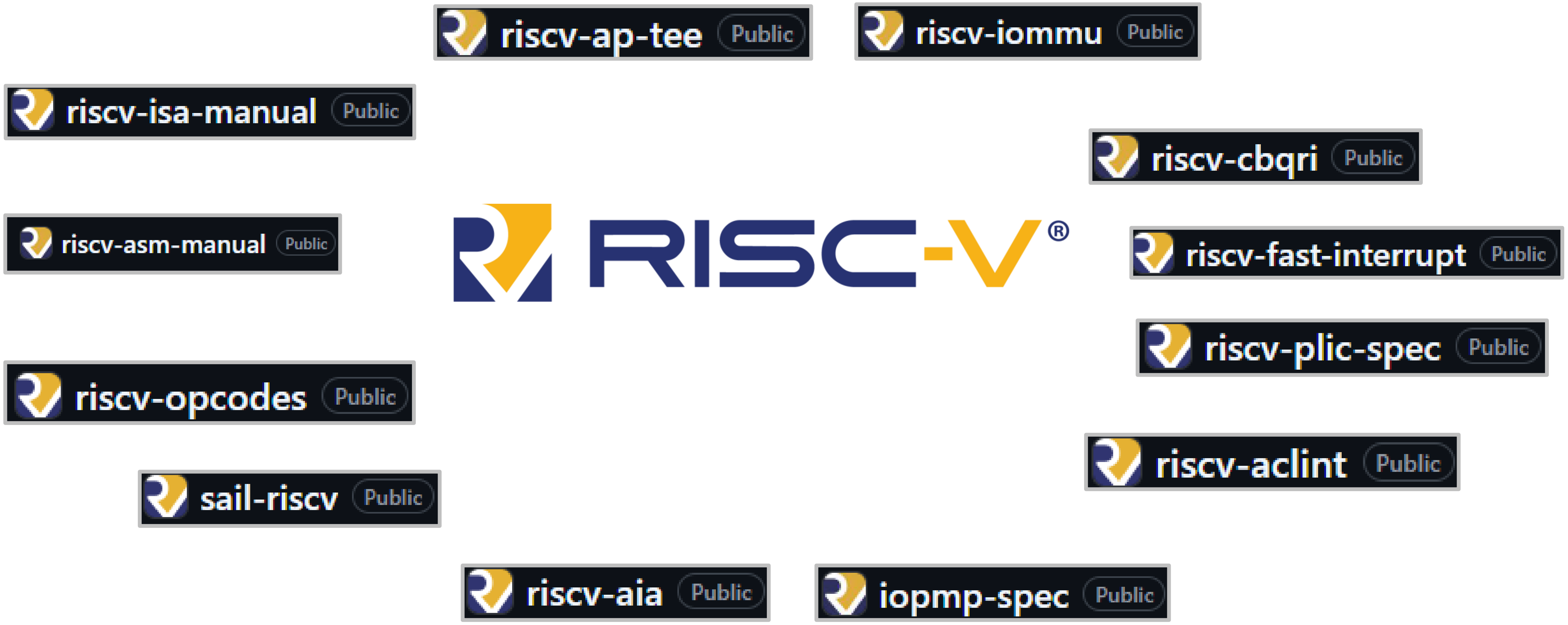
Agenda

- Challenges with current RISC-V Specification Ecosystem
- RISC-V Unified Database (UDB)
- Call for help

Challenges with the RISC-V Specification Ecosystem

Why this matters

RISC-V ecosystem relies on multiple disconnected specifications



riscv-ap-tee Public

riscv-iommu Public

riscv-isa-manual Public

riscv-cbqri Public

riscv-asm-manual Public

RISC-V®

riscv-fast-interrupt Public

riscv-plic-spec Public

riscv-opcodes Public

riscv-aclint Public

sail-riscv Public

riscv-aia Public

iopmp-spec Public

Technical Challenges with the RISC-V Specification Ecosystem

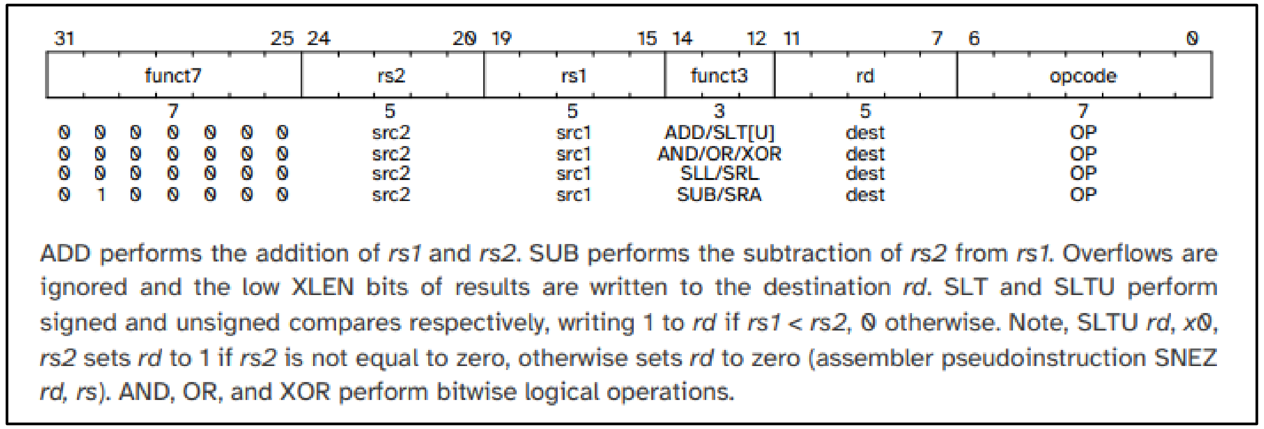
(some of)

- ISA Manuals (https://github.com/riscv/riscv-isa-manual)
  - Not machine readable or verifiable
  - Not versioned schema
- Assembly Manual (https://github.com/riscv/riscv-asm-manual)
  - Not machine readable or verifiable
  - Not versioned schema
  - Not a clear and concise index of ALL Instructions or Pseudoinstructions
- RISC-V Opcodes (https://github.com/riscv/riscv-opcodes)
  - Not versioned
  - Not a fully descriptive database, the format doesn’t allow to encapsulate necessary information
  - Not a description of ALL instructions and pseudo-instructions that exist
- Sail RISC-V (https://github.com/riscv/sail-riscv)
  - Not Pseudo-code
  - Hard to parse and insert in documentation

Usability Challenges with RISC-V Specification Ecosystem

For documentation **consumers**: developers of software & tools, scientists etc

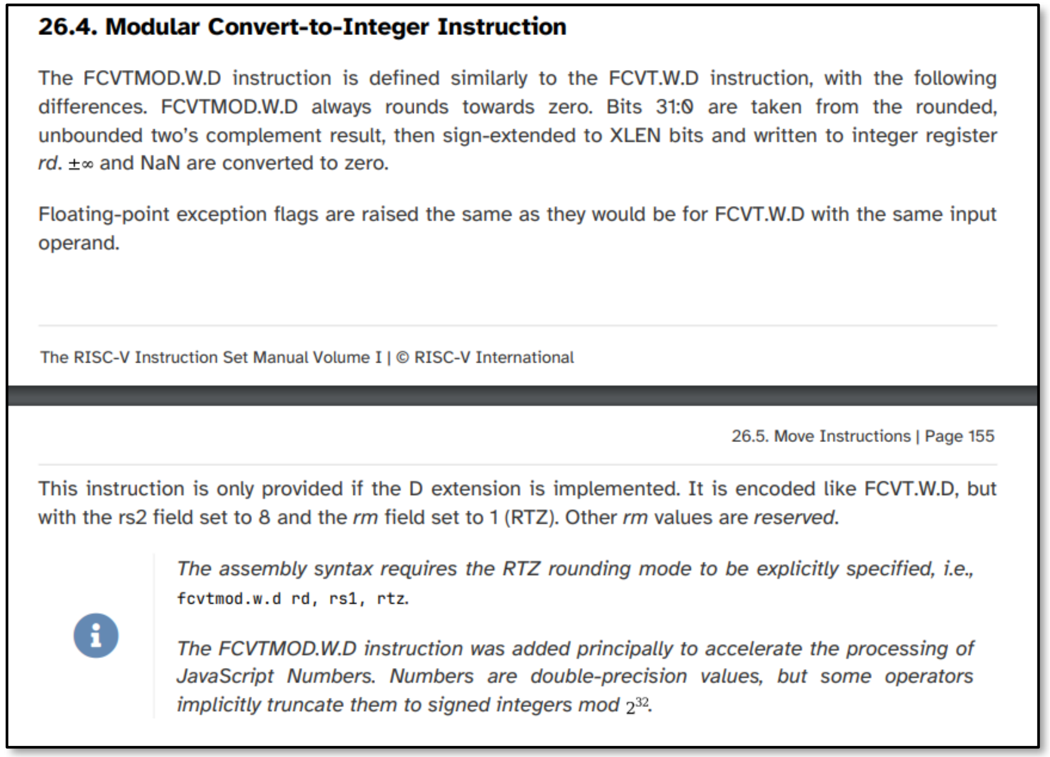
- Information is not cross-referenced between resources
  - E.g. ISA Manual describes an instruction, but doesn't specify or link to its encoding (riscv-opcodes) or pseudocode description (Sail)



31–25	24–20	19–15	14–12	11–7	6–0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
0 0 0 0 0 0 0	src2	src1	ADD/SLT[U]	dest	OP
0 0 0 0 0 0 0	src2	src1	AND/OR/XOR	dest	OP
0 0 0 0 0 0 0	src2	src1	SLL/SRL	dest	OP
0 1 0 0 0 0 0	src2	src1	SUB/SRA	dest	OP

ADD performs the addition of *rs1* and *rs2*. SUB performs the subtraction of *rs2* from *rs1*. Overflows are ignored and the low XLEN bits of results are written to the destination *rd*. SLT and SLTU perform signed and unsigned compares respectively, writing 1 to *rd* if *rs1* < *rs2*, 0 otherwise. Note, SLTU *rd*, *x0*, *rs2* sets *rd* to 1 if *rs2* is not equal to zero, otherwise sets *rd* to zero (assembler pseudoinstruction SNEZ *rd*, *rs*). AND, OR, and XOR perform bitwise logical operations.

Prose instruction description;
encodings but formal description/pseudocode



26.4. Modular Convert-to-Integer Instruction

The FCVTMOD.W.D instruction is defined similarly to the FCVT.W.D instruction, with the following differences. FCVTMOD.W.D always rounds towards zero. Bits 31:0 are taken from the rounded, unbounded two's complement result, then sign-extended to XLEN bits and written to integer register *rd*. ±∞ and NaN are converted to zero.

Floating-point exception flags are raised the same as they would be for FCVT.W.D with the same input operand.

The RISC-V Instruction Set Manual Volume I | © RISC-V International

26.5. Move Instructions | Page 155

This instruction is only provided if the D extension is implemented. It is encoded like FCVT.W.D, but with the rs2 field set to 8 and the *rm* field set to 1 (RTZ). Other *rm* values are *reserved*.

The assembly syntax requires the RTZ rounding mode to be explicitly specified, i.e., `fcvtmod.w.d rd, rs1, rtz`.

The FCVTMOD.W.D instruction was added principally to accelerate the processing of JavaScript Numbers. Numbers are double-precision values, but some operators implicitly truncate them to signed integers mod 2^{32}.

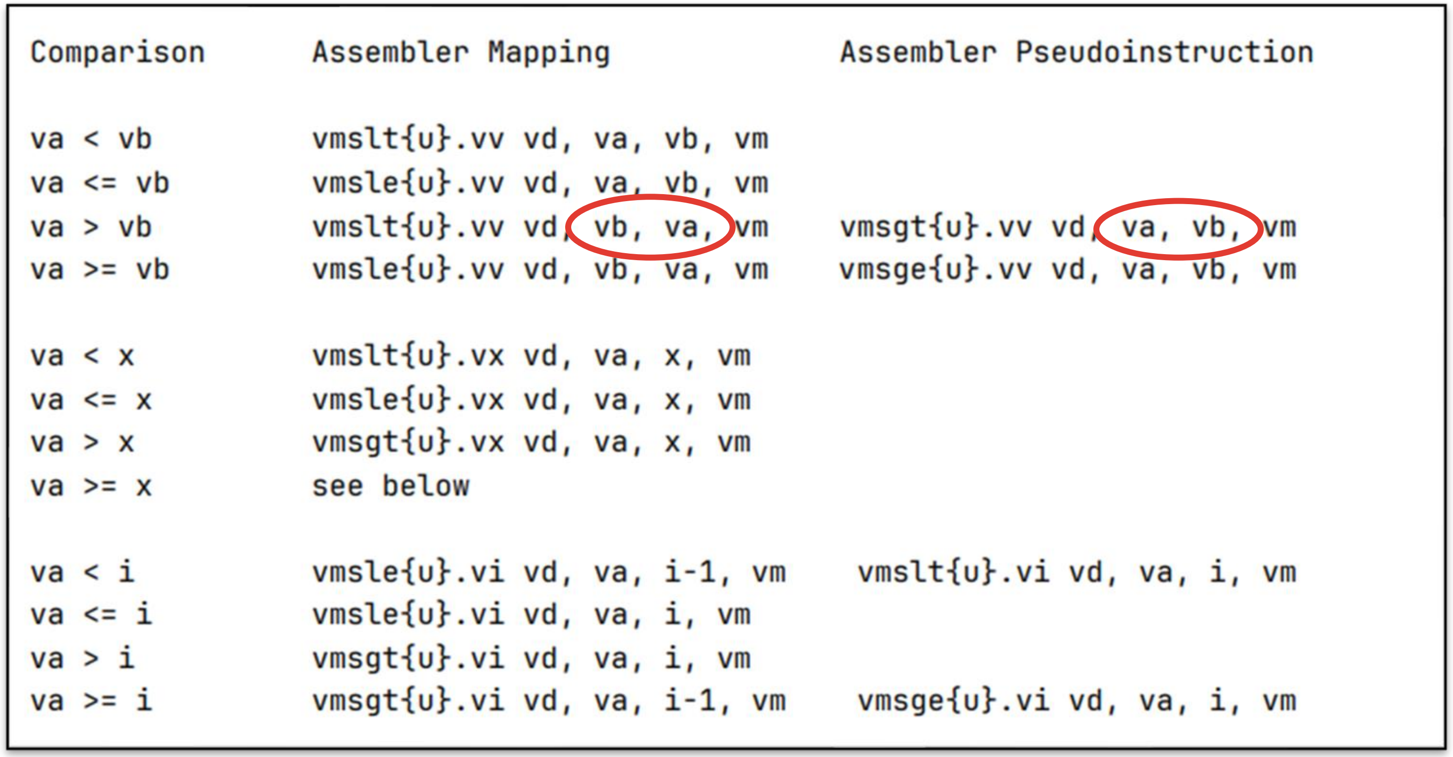
Prose-only instruction description;
missing encodings and formal description/pseudocode

Usability Challenges with RISC-V Specification Ecosystem

For documentation **consumers**: developers of software & tools, scientists etc

- Inconsistencies throughout different documents
  - Only some manuals are ratified, the rest of the documents are maintained randomly

- Defined on the ISA Manual
- Not defined on the ASM Manual (at the time)
- Not defined in RISC-V Opcodes (or supported)



Comparison	Assembler Mapping	Assembler Pseudoinstruction
va < vb	vmslt{u}.vv vd, va, vb, vm	
va <= vb	vmsle{u}.vv vd, va, vb, vm	
va > vb	vmslt{u}.vv vd, vb, va, vm	vmsgt{u}.vv vd, va, vb, vm
va >= vb	vmsle{u}.vv vd, vb, va, vm	vmsge{u}.vv vd, va, vb, vm
va < x	vmslt{u}.vx vd, va, x, vm	
va <= x	vmsle{u}.vx vd, va, x, vm	
va > x	vmsgt{u}.vx vd, va, x, vm	
va >= x	see below	
va < i	vmsle{u}.vi vd, va, i-1, vm	vmslt{u}.vi vd, va, i, vm
va <= i	vmsle{u}.vi vd, va, i, vm	
va > i	vmsgt{u}.vi vd, va, i, vm	
va >= i	vmsgt{u}.vi vd, va, i-1, vm	vmsge{u}.vi vd, va, i, vm

Usability Challenges with RISC-V Specification Ecosystem

For documentation **consumers**: developers of software & tools, scientists etc

RISC-V Opcodes Lack of Support

- Over simplistic representation
  - Format only specifies encodings and arguments

- 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

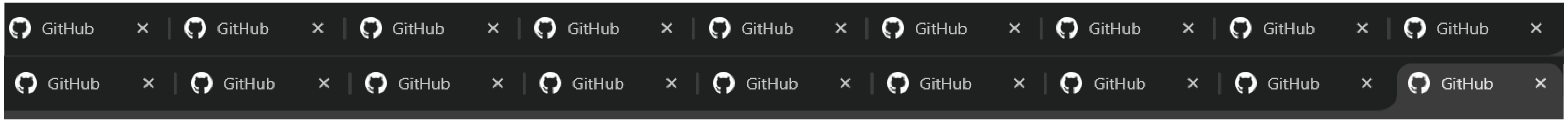
- Arguments with fixed values are not supported
  - Assigning variable fields is not possible

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

Usability Challenges with RISC-V Specification Ecosystem

For documentation **consumers**: developers of software & tools, scientists etc

- Developers have a *lot* of different places to search in
  - Several Manuals
  - It's sometimes not clear to what manuals the user should refer to
  - Results in relying in unofficial resources
    - Conference presentations, Reddit, Internet forums etc
  - Ends up looking like


GitHub ✕ GitHub ✕ GitHub ✕ GitHub ✕ GitHub ✕ GitHub ✕ GitHub ✕ GitHub ✕ GitHub ✕
GitHub ✕ GitHub ✕ GitHub ✕ GitHub ✕ GitHub ✕ GitHub ✕ GitHub ✕ GitHub ✕ GitHub ✕

Usability Challenges with RISC-V Specification Ecosystem

For documentation **producers**: specification writers, IP & SoC vendors

- Current information is written in AsciiDoc
  - Not machine readable
  - Not verifiable
- Creating any Manual from this is unreliable
  - Needs copy/paste or rewrite
  - It’s Error-Prone

ADD performs the addition of _rs1_ and _rs2_. SUB performs the subtraction of _rs2_ from _rs1_. Overflows are ignored and the low XLEN bits of results are written to the destination _rd_. SLT and SLTU perform signed and unsigned compares respectively, writing 1 to _rd_ if _rs1_ < _rs2_, 0 otherwise. Note, SLTU _rd_, _x0_, _rs2_ sets _rd_ to 1 if _rs2_ is not equal to zero, otherwise sets _rd_ to zero (assembler pseudoinstruction SNEZ _rd, rs_). AND, OR, and XOR perform bitwise logical operations.

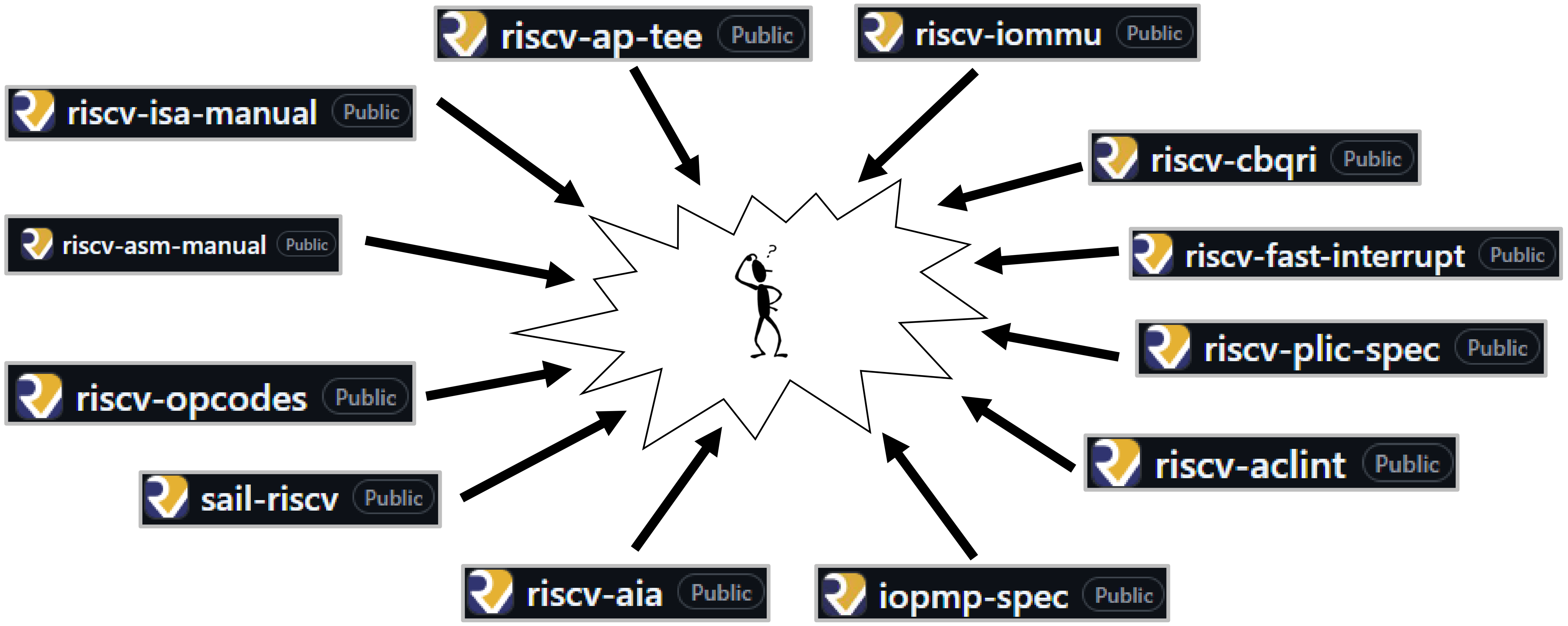
Usability Challenges with RISC-V Specification Ecosystem

For documentation **producers**: specification writers, IP & SoC vendors

- A given design may include a mix of arbitrary extensions and their versions
  - Without versions "rv32im_zba_zbb_zbs_zicsr_zca_zcb_zcmp“
  - With different versions “RV32IM_Zba1p0_Zbb1p0_Zbs1p0_Zicsr2p0_Zca1p0_Zcb1p0_Zcmp1p0”
- Documentation ends up being a really complex task
  - Hard to keep up with upstream
  - Expensive and time consuming
  - Can even impact Time to Market
- No reliable existing solution to create specific documentation

Why this matters

RISC-V ecosystem relies on multiple disconnected specifications

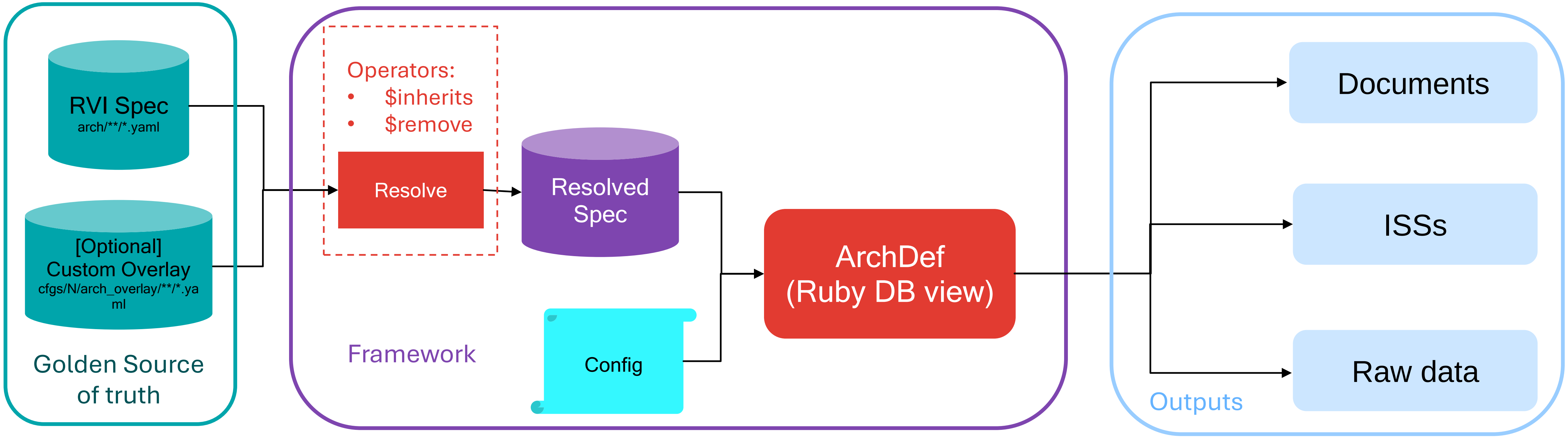


RISC-V Unified Database (UDB)

What is the UDB?

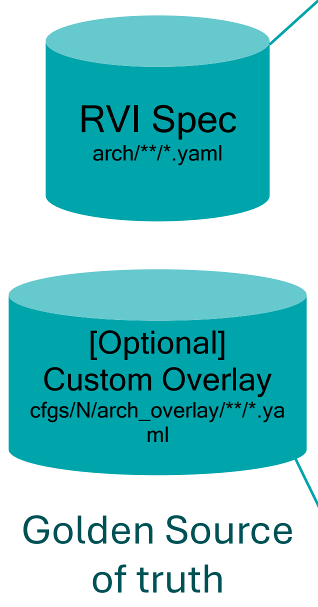
A centralized, machine readable, source of truth

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 performance and design
Examples	ADD, SUB, LW	mstatus, mtvec, mip	RV32A, RV64G	32/64-bit, pipeline depth, cache

Golden Source of Truth

Instructions

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

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

```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
data_independent_timing: true
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
  }
```

Golden Source of Truth

Control and State Registers - CSRs

- Name
- Long name
- Address
- Description
- Priv Mode
- Length
- Extension definition
- Fields
- Formal Specification

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

```
kind: csr
name: instret
long_name: Instructions retired counter for RDINSTRET Instruction
address: 0xC02
description: |
  Alias for M-mode CSR `minstret`.

  Privilege mode access is controlled with `mcounteren.IR`,
  `scounteren.IR`, and `hcounteren.IR`.
  Shortened for presentation purposes. Full description in github.

priv_mode: U
length: 64
definedBy: Zicntr
fields:
  COUNT:
    location: 63-0
    alias: minstret.COUNT
    description: Alias of `minstret.COUNT`.
    type: RO-H
    reset_value: 0
sw_read(): |

  # Shortened for presentation purposes. Full code on github.
  #access is determined by *counteren CSRs
  if (mode() == PrivilegeMode::S) {
    # S-mode is present ->
    #   mcounteren determines access in S-mode
    if (CSR[mcounteren].IR == 1'b0) {
      raise(ExceptionCode::IllegalInstruction, mode(), $encoding);
    }

  return CSR[minstret].COUNT;
```

Golden Source of Truth

Profiles

- Name
- Mode
- 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

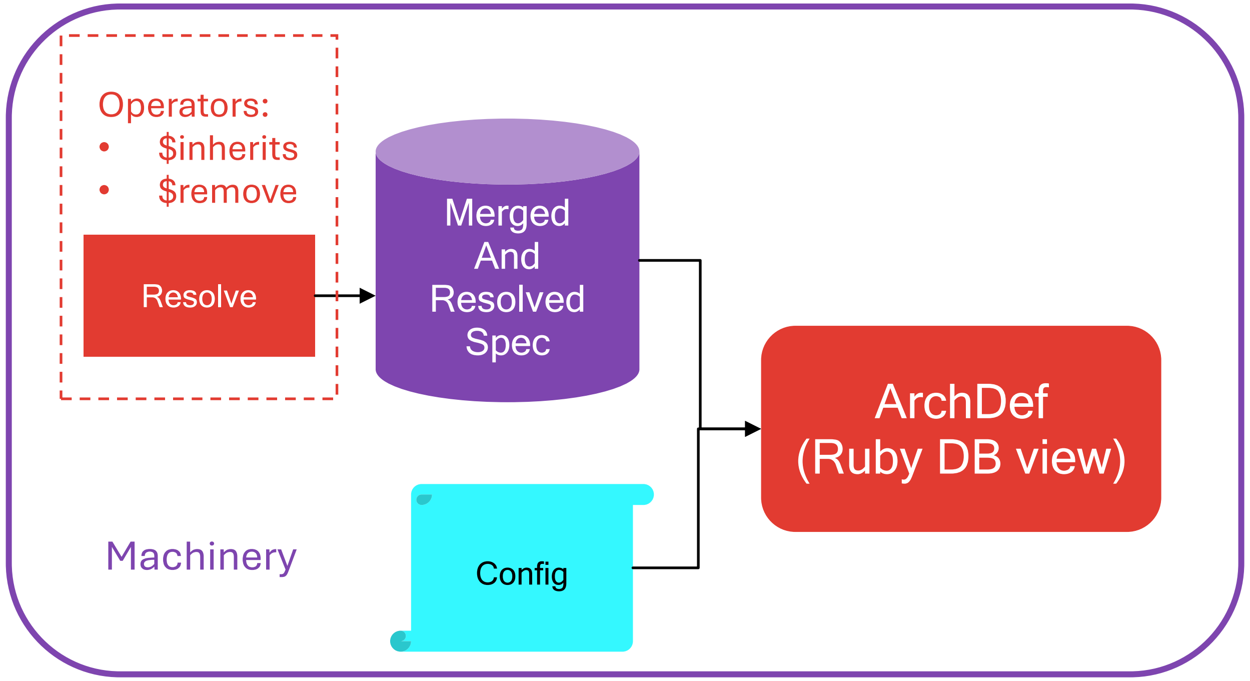
```yaml
kind: profile
name: RVI20U32
marketing_name: RVI20U32
mode: Unpriv
base: 32
release: { $ref: profile_release/RVI20.yaml# }
introduction: |
  This profile specifies the ISA features available to generic
  unprivileged execution environments.

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.
```

Framework

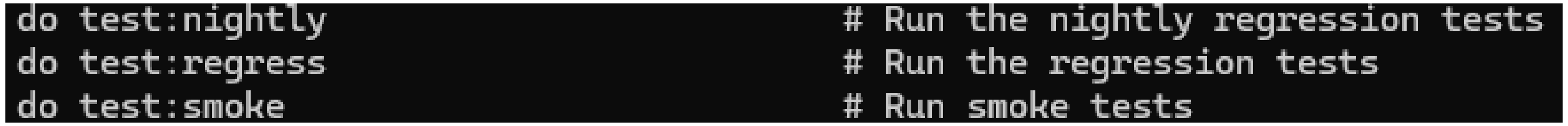
Simplistic view



- Merge different source files
- Resolve dependencies
- Configure Architectural parameters
- Generate RubyDB View
    - Where the outputs come from

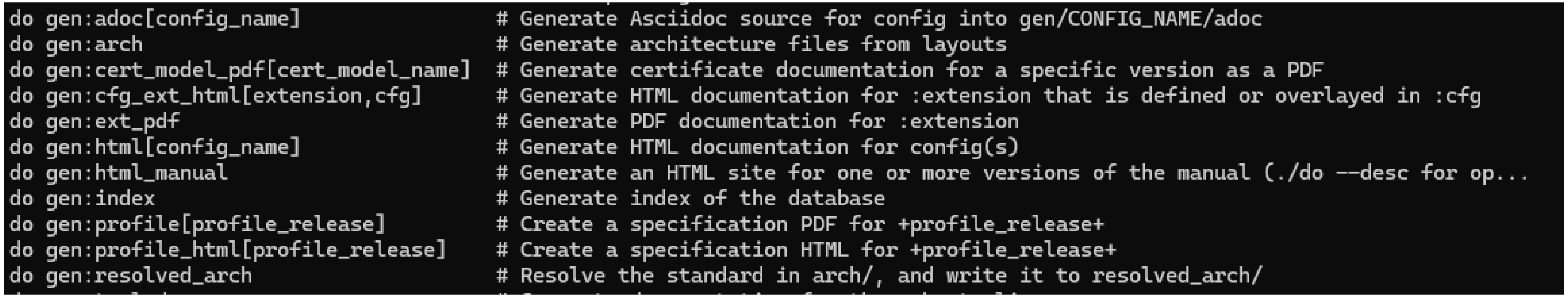
Framework

- Validation



```
do test:nightly                         # Run the nightly regression tests
do test:regress                         # Run the regression tests
do test:smoke                           # Run smoke tests
```

- Generation
  - Solving dependencies from source
  - Available outputs for tools to pick
  - Build documents (HTML/PDF)



```
do gen:adoc[config_name]                # Generate Asciidoc source for config into gen/CONFIG_NAME/adoc
do gen:arch                             # Generate architecture files from layouts
do gen:cert_model_pdf[cert_model_name]  # Generate certificate documentation for a specific version as a PDF
do gen:cfg_ext_html[extension,cfg]      # Generate HTML documentation for :extension that is defined or overlayed in :cfg
do gen:ext_pdf                          # Generate PDF documentation for :extension
do gen:html[config_name]                # Generate HTML documentation for config(s)
do gen:html_manual                      # Generate an HTML site for one or more versions of the manual (./do --desc for op...
do gen:index                            # Generate index of the database
do gen:profile[profile_release]         # Create a specification PDF for +profile_release+
do gen:profile_html[profile_release]    # Create a specification HTML for +profile_release+
do gen:resolved_arch                    # Resolve the standard in arch/, and write it to resolved_arch/
```

Current outputs

Documents

ISSs

Raw Data

Outputs

Manuals	ISA Manual Certification Documents Profile Manuals Extensions Manuals
Detailed Indexes	Instructions Extensions CSRs
RISC-V Opcodes outputs	Json Encodings.h Inst.go
Instruction Set Simulator	Configurable on UDB parameters

Demo

(for offline visualization, please go to backup slides)

RISC-V Opcodes Outputs

- .json middle-end
- Encodings.h for Spike
- inst.chisel
- inst.sverilog
- inst.go

```
"add": {
   "encoding": "0000000----------000-----0110011",
   "variable_fields": [
     "rd",
     "rs1",
     "rs2"
   ],
   "extension": [
     "rv_i"
   ],
   "match": "0x33",
   "mask": "0xfe00707f"
 },
 "sub": {
   "encoding": "0100000----------000-----0110011",
   "variable_fields": [
     "rd",
     "rs1",
     "rs2"
   ],
   "extension": [
     "rv_i"
   ],
   "match": "0x40000033",
   "mask": "0xfe00707f"
 },
```

How the UDB relates to current challenges

How the UDB **solves** current challenges

Challenge	UDB Solution
Inconsistencies throughout different documents	- Same source solves this inconsistency
Information is sometimes not referenced	- Build system allows tagging
Creating CPU documentation requires copy-pasting the ISA	- Automates documentation generation by consolidating data from all ISA versions into one centralized database.
Current information is not machine-readable	- YAML format ensures machine-readability.
No reliable solution to create tailored documentation	- Provides a unified platform for tailored, consistent documentation. - Outputs include instruction indexes, opcode definitions, and profiles.

How does the UDB **not solve** RISC-V specification ecosystem challenges?

Design Limitations
Not designed to describe full SoCs
There still is prose and that can not be verified

Current Roadblocks
Copy/Paste is Chicken/Egg problem
Information is not verified by a lot of people due to not super mature stage of the project
Information is missing in some of the new outputs of the UDB

Use cases

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 processors documentation
- Can substitute RISC-V Opcodes
- Provide content for new tools
- Building any documents **you** may want to

Future work

Future Steps

- Vendors and consumers use UDB to create their documentation
- New tools can rely on this information
- Specify all ISA and peripheral specification

The UDB end goal

- Become the main source of information for the entire RISC-V ecosystem

Call for help

UDB is being collaboratively developed

- Several engineers from different companies are actively developing and meeting weekly, meetings are open to everyone
  - Akeana, Qualcomm, Rivos, Sifive, Synopsys, Ventana
- SIG is being formally proposed in the 5th of February to the Technical Sterring Comitee

Join us!

- Check our github repository https://github.com/riscv-software-src/riscv-unified-db/
  - Several issues and discussions on going!
- On our weekly meetings (No formal SIG yet, but please e-mail me and I'll forward an invite!)
- Contact me Afonso.Oliveira@synopsys.com

We are hiring!

Join us if you're interested in making RISC-V documentation better

- Contact me if interested
- Check our job posting and apply!

SYNOPSYS®

Thank you

synopsys®

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

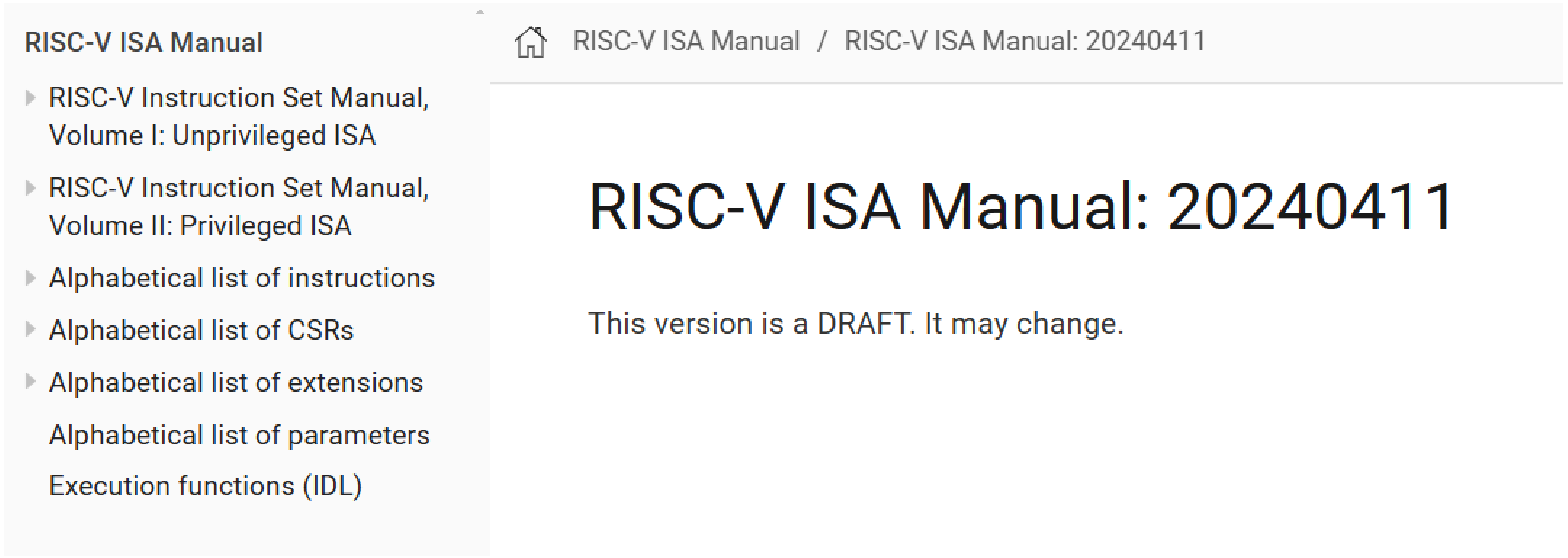
https://github.com/riscv-software-src/riscv-unified-db/blob/main/arch/profile/RVI20U32.yaml

```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



RISC-V ISA Manual

- RISC-V Instruction Set Manual, Volume I: Unprivileged ISA
- RISC-V Instruction Set Manual, Volume II: Privileged ISA
- Alphabetical list of instructions
- Alphabetical list of CSRs
- Alphabetical list of extensions
- Alphabetical list of parameters
- Execution functions (IDL)

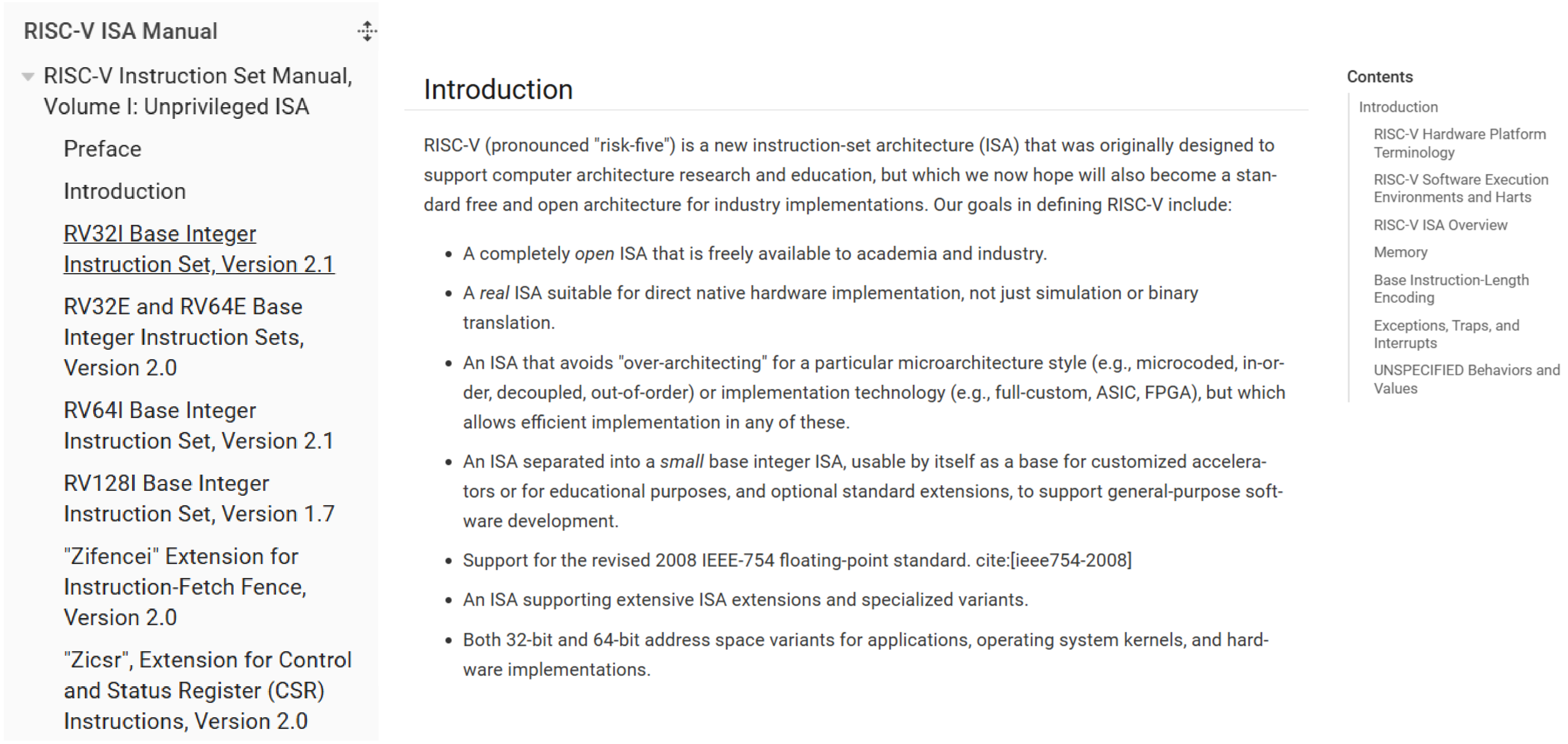
RISC-V ISA Manual / RISC-V ISA Manual: 20240411

RISC-V ISA Manual: 20240411

This version is a DRAFT. It may change.

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

ISA Manual



RISC-V 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:

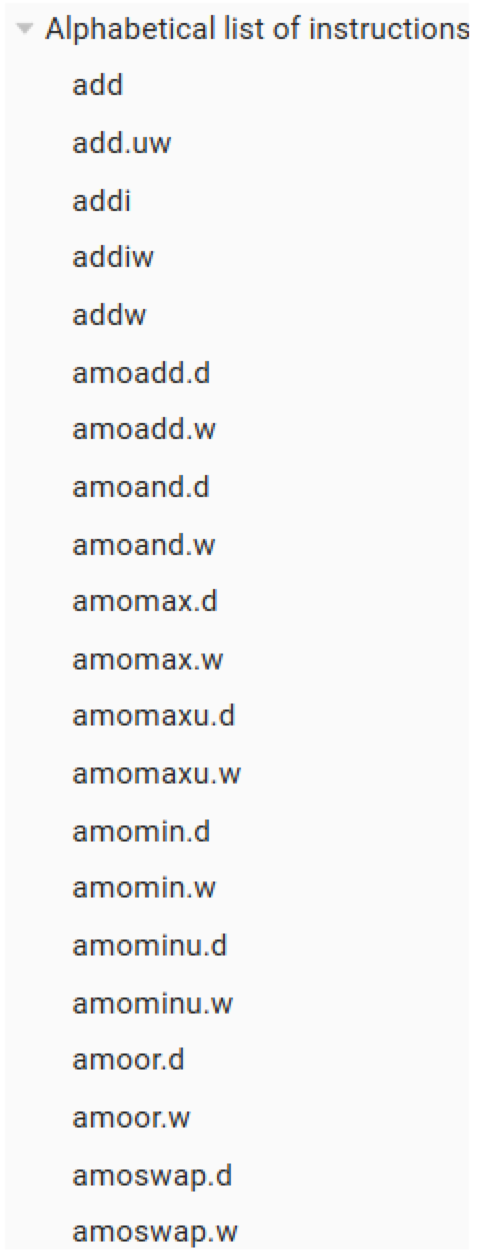
- A completely *open* ISA that is freely available to academia and industry.
- A *real* ISA suitable for direct native hardware implementation, not just simulation or binary translation.
- An ISA that avoids "over-architecting" for a particular microarchitecture style (e.g., microcoded, in-order, decoupled, 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.

Contents

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

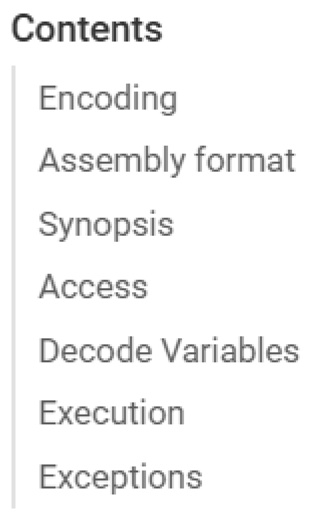
List of Instructions

- **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



Alphabetical list of instructions

add
add.uw
addi
addiw
addw
amoadd.d
amoadd.w
amoand.d
amoand.w
amomax.d
amomax.w
amomaxu.d
amomaxu.w
amomin.d
amomin.w
amominu.d
amominu.w
amoor.d
amoor.w
amoswap.d
amoswap.w

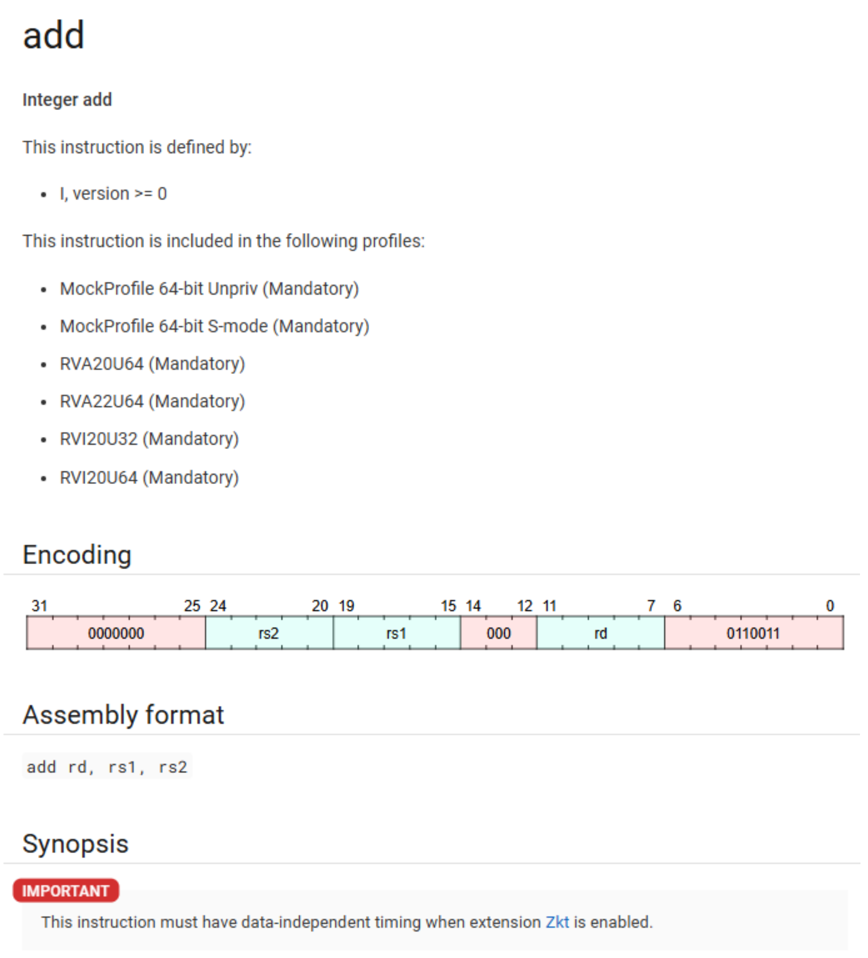


Contents

Encoding
Assembly format
Synopsis
Access
Decode Variables
Execution
Exceptions

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

31–25	24–20	19–15	14–12	11–7	6–0
0000000	rs2	rs1	000	rd	0110011

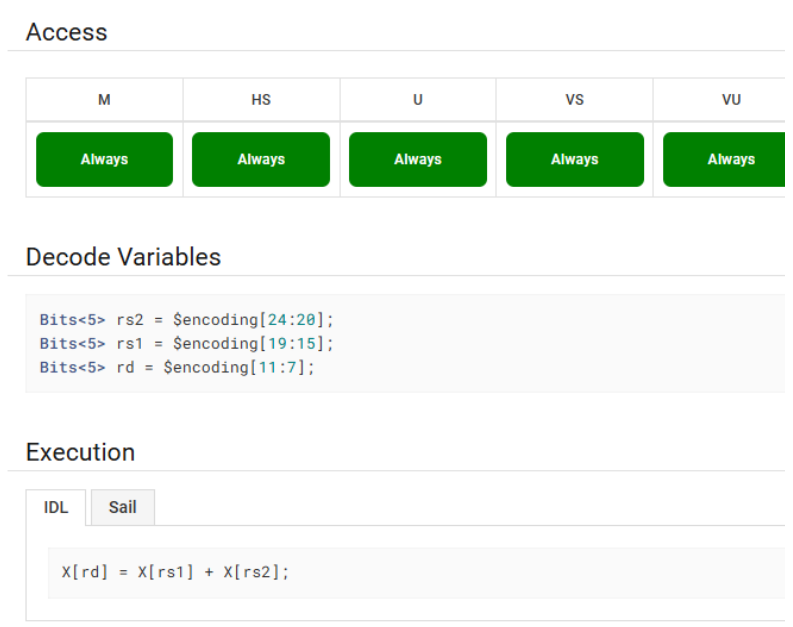
Assembly format

```
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

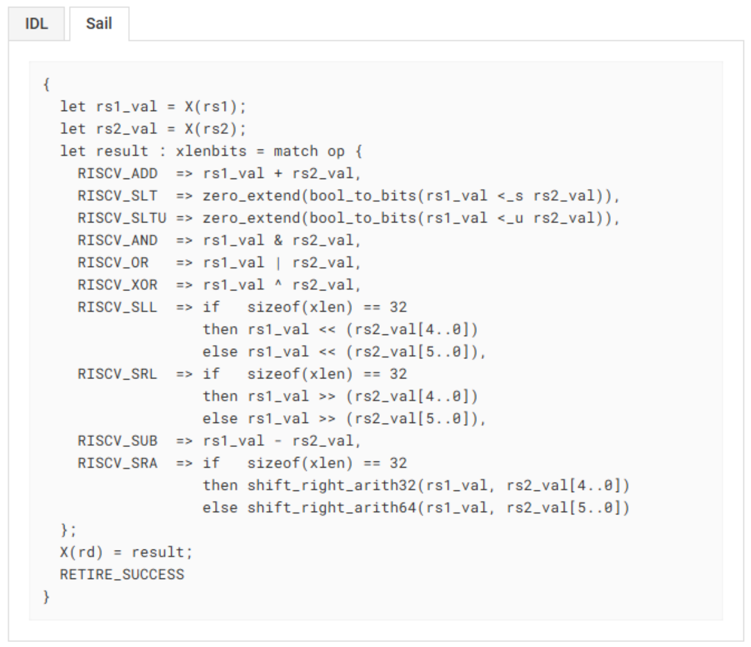
Decode Variables

```
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];
```



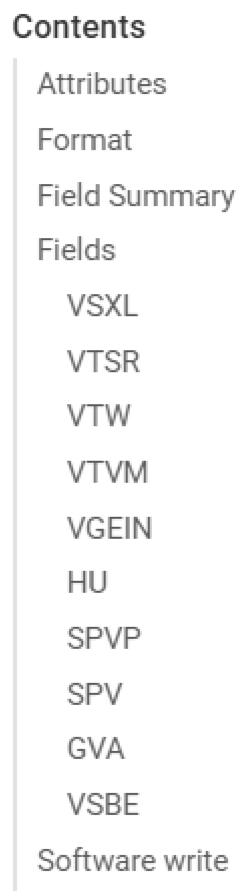
IDL Sail

```
{
  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

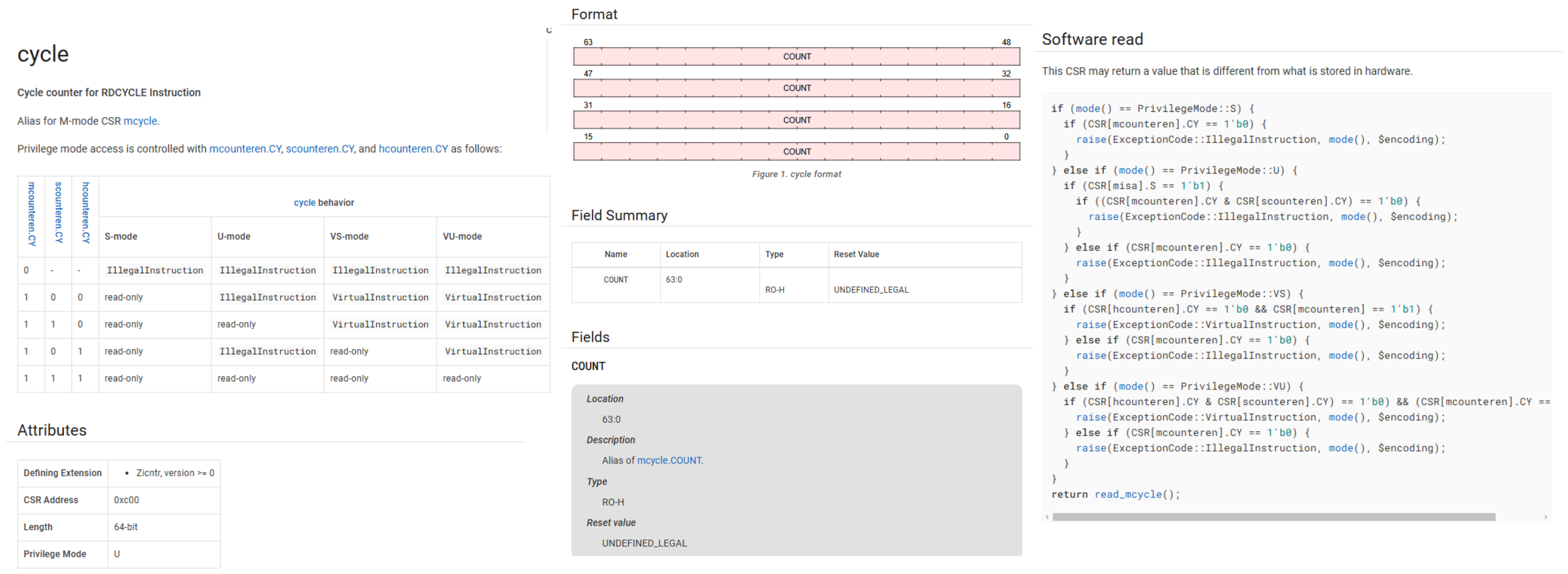
- Detailed descriptions
  - Acess Modes
  - Attibute List
  - Format
  - Field summary and descrption
  - Formal description (IDL)


Contents

- Attributes
- Format
- Field Summary
- Fields
  - VSXL
  - VTSR
  - VTW
  - VTVM
  - VGEIN
  - HU
  - SPVP
  - SPV
  - GVA
  - VSBE
- Software write

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

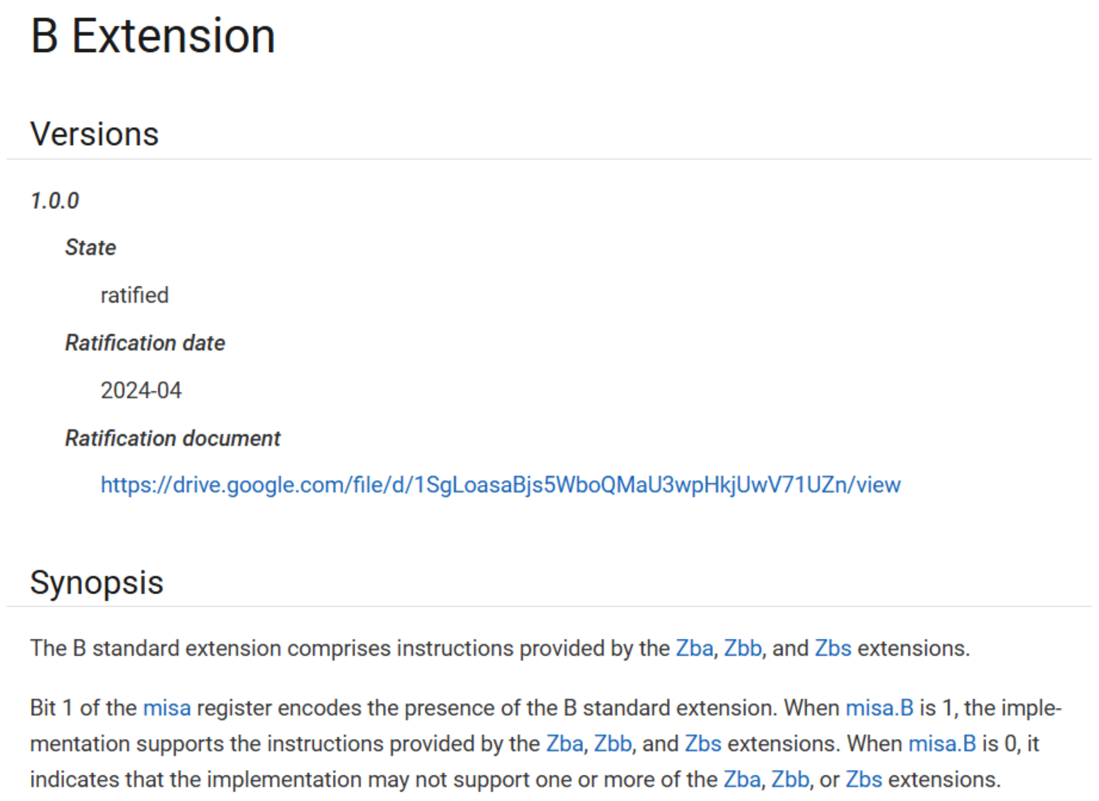
This CSR may return a value that is different from what is stored in hardware.

```
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

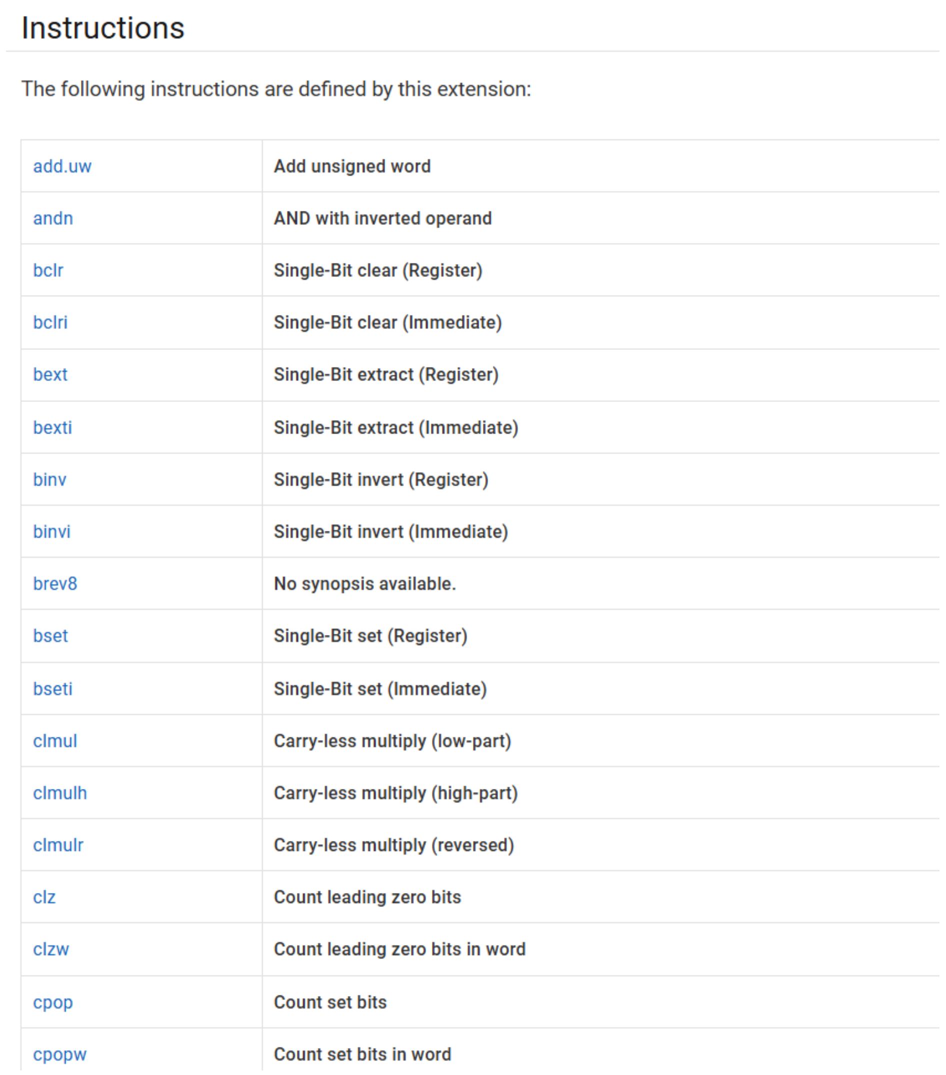
Synopsis

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

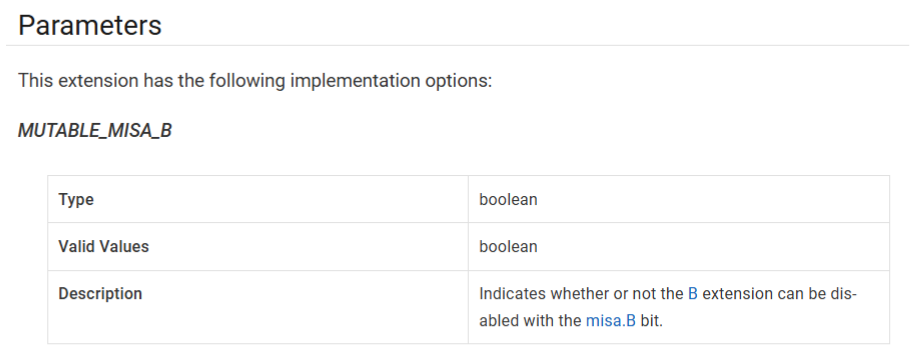
List of Extensions



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:

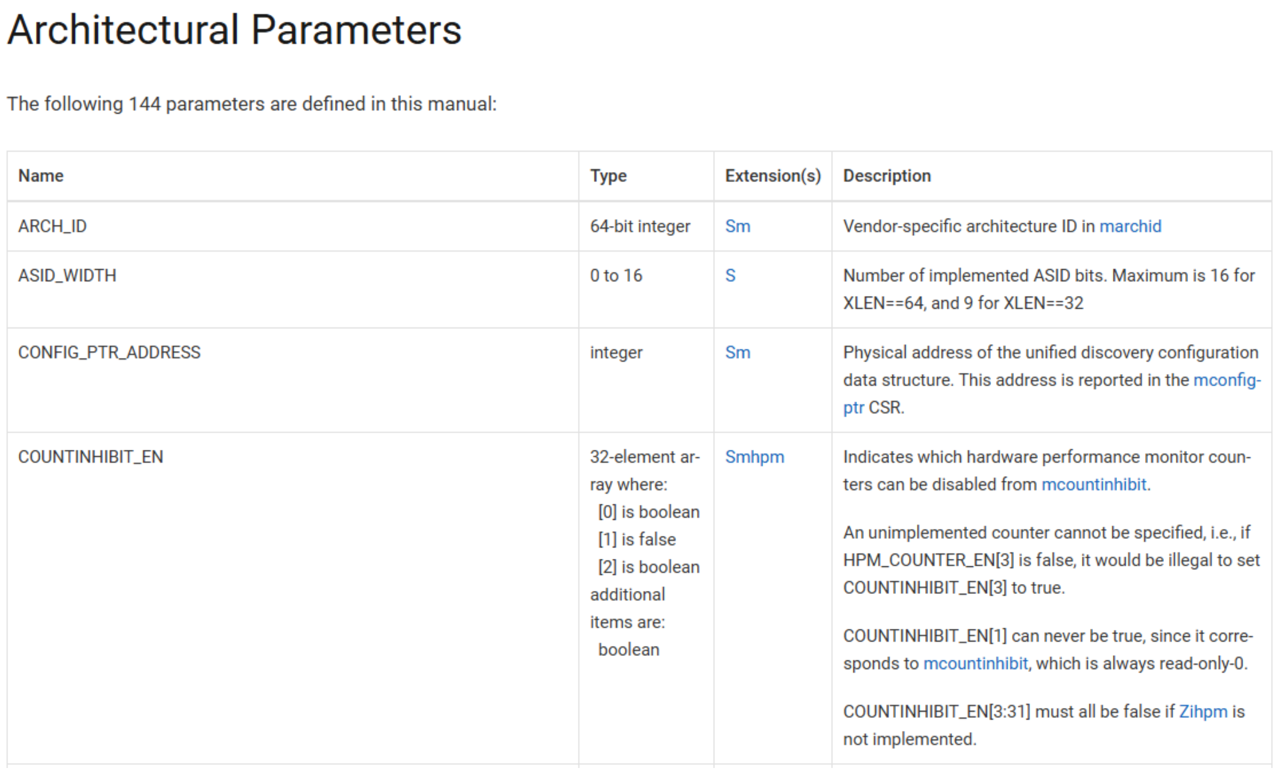
MUTABLE_MISA_B

Type	boolean
Valid Values	boolean
Description	Indicates whether or not the B extension can be dis-abled 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:

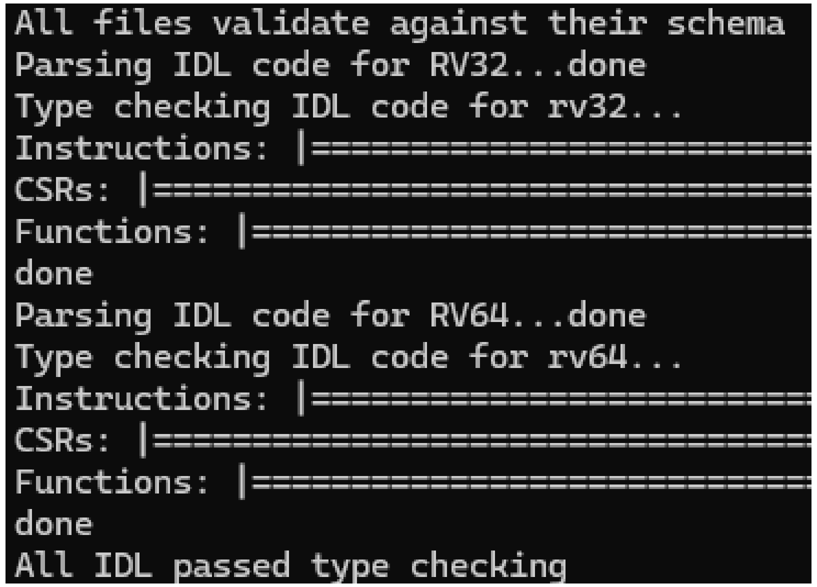
Name	Type	Extension(s)	Description
ARCH_ID	64-bit integer	Sm	Vendor-specific architecture ID in marchid
ASID_WIDTH	0 to 16	S	Number of implemented ASID bits. Maximum is 16 for XLEN==64, and 9 for XLEN==32
CONFIG_PTR_ADDRESS	integer	Sm	Physical address of the unified discovery configuration data structure. 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

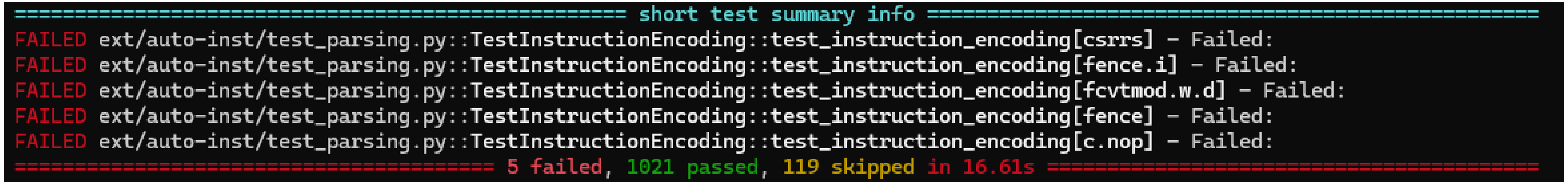
Framework

Validation methods

- Source of truth files
  - Schema Validation
- IDL Validation
- Instructions
  - LLVM
  - RISC-V Opcodes
  - Binutils – Licensing roadblock



```
All files validate against their schema
Parsing IDL code for RV32...done
Type checking IDL code for rv32...
Instructions: |========================
CSRs: |================================
Functions: |===========================
done
Parsing IDL code for RV64...done
Type checking IDL code for rv64...
Instructions: |========================
CSRs: |================================
Functions: |===========================
done
All IDL passed type checking
```



```
================================================== short test summary info ==================================================
FAILED ext/auto-inst/test_parsing.py::TestInstructionEncoding::test_instruction_encoding[csrrs] - Failed:
FAILED ext/auto-inst/test_parsing.py::TestInstructionEncoding::test_instruction_encoding[fence.i] - Failed:
FAILED ext/auto-inst/test_parsing.py::TestInstructionEncoding::test_instruction_encoding[fcvtmod.w.d] - Failed:
FAILED ext/auto-inst/test_parsing.py::TestInstructionEncoding::test_instruction_encoding[fence] - Failed:
FAILED ext/auto-inst/test_parsing.py::TestInstructionEncoding::test_instruction_encoding[c.nop] - Failed:
======================================== 5 failed, 1021 passed, 119 skipped in 16.61s ========================================
```