



# RISC-V Integer Conditional (Zicond) operations extension

Dr. Philipp Tomsich (VRULL GmbH)

Version 1.0.1, 2023-10-12: This document is frozen. See <http://riscv.org/spec-state> for details.

# Table of Contents

Preamble .....	1
Copyright and license information .....	2
Contributors .....	3
1. Introduction .....	4
1.1. Motivation and use cases .....	4
2. Zicond specification .....	5
3. Instructions (in alphabetical order) .....	6
3.1. czero.eqz .....	7
3.2. czero.nez .....	8
4. Usage examples .....	9
4.1. Instruction sequences .....	9

# Preamble



*This document is in the [Freeze state](#)*

*Change is extremely unlikely. A high threshold will be used, and a change will only occur because of some truly critical issue being identified during the public review cycle. Any other desired or needed changes can be the subject of a follow-on new extension.*

# Copyright and license information

This specification is licensed under the Creative Commons Attribution 4.0 International License (CC-BY 4.0). The full license text is available at [creativecommons.org/licenses/by/4.0/](https://creativecommons.org/licenses/by/4.0/).

Copyright 2022-2023 by RISC-V International.

# Contributors

This RISC-V specification has been contributed to directly or indirectly by:

- Dr. Philipp Tomsich <[philipp.tomsich@vrull.eu](mailto:philipp.tomsich@vrull.eu)>
- Ken Dockser <[kdockser@tenstorrent.com](mailto:kdockser@tenstorrent.com)>
- Brendan Sweeney <[brs@eecs.berkeley.edu](mailto:brs@eecs.berkeley.edu)>
- Andrew Waterman <[andrew@sifive.com](mailto:andrew@sifive.com)>

# Chapter 1. Introduction

The Zicnd extension defines a simple solution that provides most of the benefit and all of the flexibility one would desire to support conditional arithmetic and conditional-select/move operations, while remaining true to the RISC-V design philosophy. The instructions follow the format for R-type instructions with 3 operands (i.e., 2 source operands and 1 destination operand). Using these instructions, branchless sequences can be implemented (typically in two-instruction sequences) without the need for instruction fusion, special provisions during the decoding of architectural instructions, or other microarchitectural provisions.

## 1.1. Motivation and use cases

One of the shortcomings of RISC-V, compared to competing instruction set architectures, is the absence of conditional operations to support branchless code-generation: this includes conditional arithmetic, conditional select and conditional move operations. The design principles of RISC-V (e.g. the absence of an instruction-format that supports 3 source registers and an output register) make it unlikely that direct equivalents of the competing instructions will be introduced.

Yet, low-cost conditional instructions are a desirable feature as they allow the replacement of branches in a broad range of suitable situations (whether the branch turns out to be unpredictable or predictable) so as to reduce the capacity and aliasing pressures on BTBs and branch predictors, and to allow for longer basic blocks (for both the hardware and the compiler to work with).

## Chapter 2. Zicond specification

The "Conditional" operations extension provides a simple solution that provides most of the benefit and all of the flexibility one would desire to support conditional arithmetic and conditional-select/move operations, while remaining true to the RISC-V design philosophy. The instructions follow the format for R-type instructions with 3 operands (i.e., 2 source operands and 1 destination operand). Using these instructions, branchless sequences can be implemented (typically in two-instruction sequences) without the need for instruction fusion, special provisions during the decoding of architectural instructions, or other microarchitectural provisions.

The following instructions comprise the Zicond extension:

RV32	RV64	Mnemonic	Instruction
✓	✓	<code>czero.eqz rd, rs1, rs2</code>	Conditional zero, if condition is equal to zero
✓	✓	<code>czero.nez rd, rs1, rs2</code>	Conditional zero, if condition is nonzero



*Architecture Comment: defining additional comparisons, in addition to equal-to-zero and not-equal-to-zero, does not offer a benefit due to the lack of immediates or an additional register operand that the comparison takes place against.*

Based on these two instructions, synthetic instructions (i.e., short instruction sequences) for the following **conditional arithmetic** operations are supported:

- conditional add, if zero
- conditional add, if non-zero
- conditional subtract, if zero
- conditional subtract, if non-zero
- conditional bitwise-and, if zero
- conditional bitwise-and, if non-zero
- conditional bitwise-or, if zero
- conditional bitwise-or, if non-zero
- conditional bitwise-xor, if zero
- conditional bitwise-xor, if non-zero

Additionally, the following **conditional select** instructions are supported:

- conditional-select, if zero
- conditional-select, if non-zero

More complex conditions, such as comparisons against immediates, registers, single-bit tests, comparisons against ranges, etc. can be realized by composing these new instructions with existing instructions.

## Chapter 3. Instructions (in alphabetical order)



## 3.1. czero.eqz

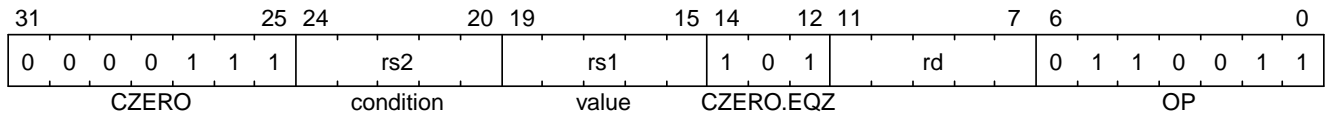
### Synopsis

Moves zero to a register *rd*, if the condition *rs2* is equal to zero, otherwise moves *rs1* to *rd*.

### Mnemonic

`czero.eqz rd, rs1, rs2`

### Encoding



### Description

If *rs2* contains the value zero, this instruction writes the value zero to *rd*. Otherwise, this instruction copies the contents of *rs1* to *rd*.

This instruction carries a syntactic dependency from both *rs1* and *rs2* to *rd*. Furthermore, if the Zkt extension is implemented, this instruction's timing is independent of the data values in *rs1* and *rs2*.

### SAIL code

```
let condition = X(rs2);
result : xlenbits = if (condition == zeros()) then zeros()
                    else X(rs1);
X(rd) = result;
```

## 3.2. czero.nez

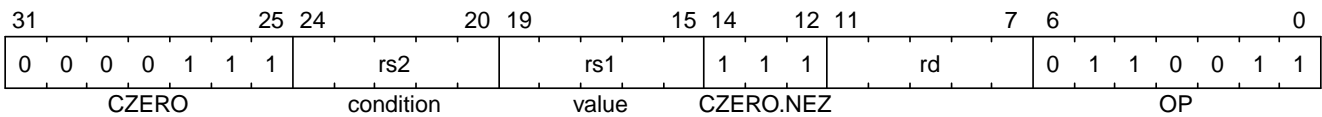
### Synopsis

Moves zero to a register *rd*, if the condition *rs2* is nonzero, otherwise moves *rs1* to *rd*.

### Mnemonic

*czero.nez rd, rs1, rs2*

### Encoding



### Description

If *rs2* contains a nonzero value, this instruction writes the value zero to *rd*. Otherwise, this instruction copies the contents of *rs1* to *rd*.

This instruction carries a syntactic dependency from both *rs1* and *rs2* to *rd*. Furthermore, if the Zkt extension is implemented, this instruction’s timing is independent of the data values in *rs1* and *rs2*.

### SAIL code

```
let condition = X(rs2);
result : xlenbits = if (condition != zeros()) then zeros()
                    else X(rs1);
X(rd) = result;
```

## Chapter 4. Usage examples

The instructions from this extension can be used to construct sequences that perform conditional-arithmetic, conditional-bitwise-logical, and conditional-select operations.

### 4.1. Instruction sequences

Operation	Instruction sequence	Length
Conditional add, if zero $rd = (rc == 0) ? (rs1 + rs2) : rs1$	<code>czero.nez rd, rs2, rc</code> <code>add rd, rs1, rd</code>	2 insns
Conditional add, if non-zero $rd = (rc != 0) ? (rs1 + rs2) : rs1$	<code>czero.eqz rd, rs2, rc</code> <code>add rd, rs1, rd</code>	
Conditional subtract, if zero $rd = (rc == 0) ? (rs1 - rs2) : rs1$	<code>czero.nez rd, rs2, rc</code> <code>sub rd, rs1, rd</code>	
Conditional subtract, if non-zero $rd = (rc != 0) ? (rs1 - rs2) : rs1$	<code>czero.eqz rd, rs2, rc</code> <code>sub rd, rs1, rd</code>	
Conditional bitwise-or, if zero $rd = (rc == 0) ? (rs1   rs2) : rs1$	<code>czero.nez rd, rs2, rc</code> <code>or rd, rs1, rd</code>	
Conditional bitwise-or, if non-zero $rd = (rc != 0) ? (rs1   rs2) : rs1$	<code>czero.eqz rd, rs2, rc</code> <code>or rd, rs1, rd</code>	
Conditional bitwise-xor, if zero $rd = (rc == 0) ? (rs1 ^ rs2) : rs1$	<code>czero.nez rd, rs2, rc</code> <code>xor rd, rs1, rd</code>	
Conditional bitwise-xor, if non-zero $rd = (rc != 0) ? (rs1 ^ rs2) : rs1$	<code>czero.eqz rd, rs2, rc</code> <code>xor rd, rs1, rd</code>	
Conditional bitwise-and, if zero $rd = (rc == 0) ? (rs1 \& rs2) : rs1$	<code>and rd, rs1, rs2</code> <code>czero.eqz rtmp, rs1, rc</code> <code>or rd, rd, rtmp</code>	3 insns (requires 1 temporary)
Conditional bitwise-and, if non-zero $rd = (rc != 0) ? (rs1 \& rs2) : rs1$	<code>and rd, rs1, rs2</code> <code>czero.nez rtmp, rs1, rc</code> <code>or rd, rd, rtmp</code>	
Conditional select, if zero $rd = (rc == 0) ? rs1 : rs2$	<code>czero.nez rd, rs1, rc</code> <code>czero.eqz rtmp, rs2, rc</code> <code>or rd, rd, rtmp</code>	
Conditional select, if non-zero $rd = (rc != 0) ? rs1 : rs2$	<code>czero.eqz rd, rs1, rc</code> <code>czero.nez rtmp, rs2, rc</code> <code>or rd, rd, rtmp</code>	