



# RISC-V Wait-on-Reservation-Set (Zawrs) extension

Ved Shanbhogue

Version 1.0, 6/2022: This document is in development. Assume everything can change. See <http://riscv.org/spec-state> for details.

# Table of Contents

Preamble.....	1
Copyright and license information .....	2
Contributors.....	2
Introduction .....	2
1. Zawrs .....	3

# Preamble



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

*Assume everything can change. This draft specification will change before being accepted as standard, so implementations made to this draft specification will likely not conform to the future standard.*

# 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 by RISC-V International.

## Contributors

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

Aaron Durbin, Abel Bernabeu, Allen Baum, Christoph Müllner, David Weaver, Greg Favor, Josh Scheid, Ken Dockser, Paul Donahue, Phil McCoy, Philipp Tomsich, Tariq Kurd, Ved Shanbhogue

## Introduction

The Zawrs extension defines a pair of instructions to be used in polling loops that allows a core to enter a low-power state and wait on a store to a memory location. Waiting for a memory location to be updated is a common pattern in many use cases such as:

1. Contenders for a lock waiting for the lock variable to be updated.
2. Consumers waiting on the tail of an empty queue for the producer to queue work/data. The producer may be code executing on a RISC-V hart, an accelerator device, an external I/O agent.
3. Code waiting on a flag to be set in memory indicative of an event occurring. For example, software on a RISC-V hart may wait on a "done" flag to be set in memory by an accelerator device indicating completion of a job previously submitted to the device.

Such use cases involve polling on memory locations, and such busy loops can be a wasteful expenditure of energy. To mitigate the wasteful looping in such usages, a **WRS.NTO** (WRS-with-no-timeout) instruction is provided. Instead of polling for a store to a specific memory location, software registers a reservation set that includes all the bytes of the memory location using the **LR** instruction. Then a subsequent **WRS.NTO** instruction would cause the hart to temporarily stall execution in a low-power state until a store occurs to the reservation set or an interrupt is observed.

Sometimes the program waiting on a memory update may also need to carry out a task at a future time or otherwise place an upper bound on the wait. To support such use cases a second instruction **WRS.STO** (WRS-with-short-timeout) is provided that works like **WRS.NTO** but bounds the stall duration to an implementation-defined short timeout such that the stall is terminated on the timeout if no other conditions have occurred to terminate the stall. The program using this instruction may then determine if its deadline has been reached.

