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

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

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

Chapter 1. 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-define short timeout such that the stall is terminated on the timeout if no other conditions have to occurred to terminate the stall. The program using this instruction may then determine if its deadline has been reached.

Chapter 2. Zawrs

The `WRS.NTO` and `WRS.STO` instructions cause the hart to temporarily stall execution in a low-power state till the reservation set is valid or an interrupt, even if disabled, is observed. For `WRS.STO` the stall duration is bounded by an implementation defined short timeout. These instructions are available in all privilege modes. These instructions are not supported in a constrained `LR`/`SC` loop.

Encoding:

31 – 20	19 – 15	14 – 12	11 – 7	6 – 0
funct12	rs1	funct3	rd	opcode
WRS.NTO(0x0d) WRS.STO(0x1d)	0	0	0	SYSTEM(0x73)

Operation:

```
Hart execution may be stalled while the following are all satisfied:
    a) The reservation set is valid
    b) If WRS.STO, a "short" duration since start of stall has not elapsed
    c) No pending interrupt is observed (see the rules below)
```

While stalled, an implementation is permitted to occasionally terminate the stall and complete execution for any reason.

`WRS.NTO` and `WRS.STO` instructions follow the rules of the `WFI` instruction for resuming execution on a pending interrupt.

When the `TW` (Timeout Wait) bit in `mstatus` is set and `WRS.NTO` is executed in S or U mode, and it does not complete within an implementation-specific bounded time limit, the `WRS.NTO` instruction will cause an illegal instruction exception.

When executing in VS or VU mode, if the `VTW` bit is set in `hstatus`, the `mstatus` `TW` bit is clear, and the `WRS.NTO` does not complete within an implementation-specific bounded time limit, the `WRS.NTO` instruction will cause a virtual instruction exception.

Since the `WRS.STO` and `WRS.NTO` instructions can complete execution for reasons other than stores to the reservation set, software will likely need a means of looping until the required stores have occurred.

The duration of a `WRS.STO` instruction's timeout may vary significantly within and among implementations. In a typical implementation this duration should be long enough to allow meaningful power reduction but short enough to avoid a program using the timeout to meet a deadline from missing it significantly. In typical implementations this duration should be roughly in the range of 10 to 100 times an on-chip cache miss latency or a cacheless access to main memory.