RISC-V Atomic Load-Acquire and Store-Release Extension (Zalasr)

Version 1.0, 2025-10-30: Ratified

Table of Contents

Preamble

This document is in the Ratified state

No changes are allowed. Any necessary or desired modifications must be addressed through a follow-on extension. Ratified extensions are never revised.

Copyright and license information

Contributors

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

- Brendan Sweeney <turtwig@utexas.edu>
- Hans Boehm <hboehm@google.com>
- Andrew Waterman <andrew@sifive.com>
- Andrea Parri <parri.andrea@gmail.com>

Chapter 1. Introduction

The Zalasr (Load-Acquire and Store-Release) extension provides load-acquire and store-release instructions in RISC-V. These can be important for high performance designs by enabling finer-grained synchronisation than is possible with fences alone, by providing a unidirectional fence. Load-acquire and store-release are widely used in language-level memory models: both the Java and C++ memory models make use of acquire-release semantics, and C++'s `atomic` provides primitives that are meant to map directly to load-acquire and store-release instructions.

The Zalasr extension builds on the atomic support provided by the Zaamo (Atomic Memory Operations), Zalrsc (Load-Reserved and Store-Conditional), and Zabha (Byte and Halfword Atomic Memory Operations) extensions by providing additional atomic operations (although it can be implemented independently of them). All of the AMO operations in Zaamo (and Zabha) are read-modify-write operations that both load and store. The Zalrsc extension provides operations that are only loads or stores. However, since it is designed to perform an atomic operation on a single memory word or doubleword, the loads and stores are designed to be paired. The load-reserved implies that a future store-conditional will follow while store-conditional requires that there was a previous load-reserved without other intervening loads or stores. Therefore, the Zalrsc extension does not provide a general atomic and ordered load or store.

Zalasr fills this gap by offering truly standalone atomic and ordered loads and stores. The Zalasr instructions are atomic loads and stores that support ordering annotations. With the combination of Zaamo, Zabha, and Zalasr all C++ atomic operations can be supported with single instructions.

Chapter 2. Instructions

The Zalasr instructions always sign-extend the value placed in *rd* and ignore the upper bits of the value of *rs2*. The instructions in the Zalasr extension require that the address held in *rs1* be naturally aligned to the size in bytes (2^{width}) of the operand. If the address is not naturally aligned, an address-misaligned exception or an access-fault exception will be generated. The access-fault exception can be generated for a memory access that would otherwise be able to complete except for the misalignment, if the misaligned access should not be emulated.

The misaligned atomicity granule PMA, defined in Volume II of this manual, optionally relaxes this alignment requirement. If all accessed bytes lie within the same misaligned atomicity granule, the instruction will not raise an exception for reasons of address alignment, and the instruction will give rise to only one memory operation for the purposes of RVWMO—i.e., it will execute atomically.

2.1. Load Acquire

Synopsis

The load-acquire instruction atomically loads a 2^{width}-byte value from the address in *rs1* and places the sign-extended value into the register *rd*, subject to the ordering annotations specified in the instruction.

Mnemonic

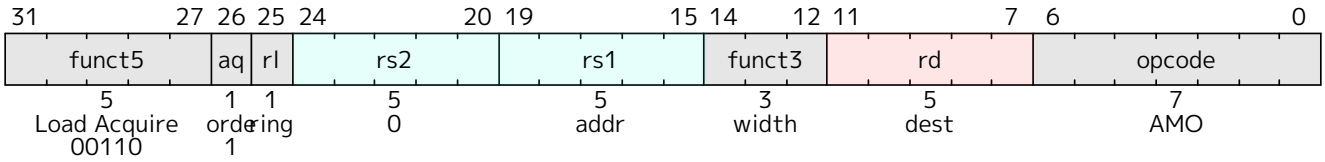
lb.{aq,aqrl} *rd*, (*rs1*)

lh.{aq,aqrl} *rd*, (*rs1*)

lw.{aq,aqrl} *rd*, (*rs1*)

ld.{aq,aqrl} *rd*, (*rs1*)

Encoding



Description

This instruction loads 2^{width} bytes of memory from rs1 atomically and writes the result into rd. If the size ($2^{width+3}$) is less than XLEN, it is sign-extended to fill the destination register. This load must have the ordering annotation *aq* and may have ordering annotation *rl* encoded in the instruction. The instruction always has an "acquire-RCsc" annotation, and if the bit *rl* is set the instruction has a "release-RCsc" annotation.

The versions without the *aq* bit set are RESERVED. LD.{AQ, AQRL} is RV64-only.

The aq bit is mandatory because the two encodings that would be produced are not seen as useful at this time. The version with neither the aq nor the rl bit set would correspond to a load with no ordering annotations that was guaranteed to be performed atomically. This can be achieved with ordinary load instructions by suitably aligning pointers. The version with only the rl bit would correspond to load-release. Load-release has theoretical applications in seqlocks, but is not supported in language-level memory models and so is not included.

2.2. Store Release

Synopsis

The store-release instruction atomically stores the 2^{width}-byte value from the low bits of register *rs2* to the address in *rs1*, subject to the ordering annotations specified in the instruction.

Mnemonic

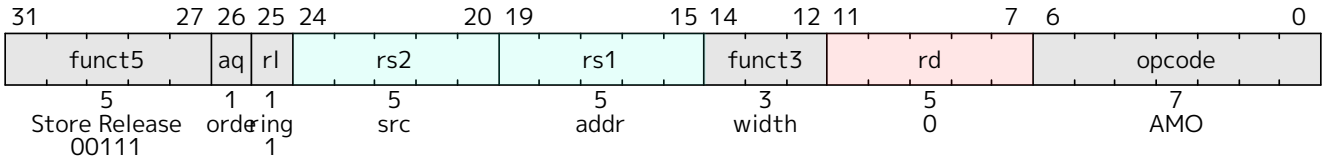
sb.{rl,aqrl} *rs2*, (*rs1*)

sh.{rl,aqrl} *rs2*, (*rs1*)

sw.{rl,aqrl} *rs2*, (*rs1*)

sd.{rl,aqrl} *rs2*, (*rs1*)

Encoding



Description

This instruction stores 2^{width} bytes of memory from rs1 atomically. This store must have ordering annotation *rl* and may have ordering annotation *aq* encoded in the instruction. The instruction always has an "release-RCsc" annotation, and if the bit *aq* is set the instruction has a "acquire-RCsc" annotation.

The versions without the *rl* bit set are RESERVED. SD.{RL, AQRL} is RV64-only.

The rl bit is mandatory because the two encodings that would be produced are not seen as useful at this time. The version with neither the aq nor the rl bit set would correspond to a store with no ordering annotations that was guaranteed to be performed atomically. This can be achieved with ordinary store instructions by suitably aligned pointers. The version with only the aq bit would correspond to store-acquire. Store-acquire has theoretical applications in seqlocks, but is not supported in language-level memory models and so is not included.

Bibliography