Atomic Compare-and-Swap (CAS) instructions (Zacas)

Version 1.0-rc1, 6/2023: This document is in stable state. See http://riscv.org/spec-state for details.

Table of Contents

Preamble

This document is in the Stable state

Assume anything could still change, but limited change should be expected. 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: Greg Favor, Ved Shanbhogue, Andrew Waterman

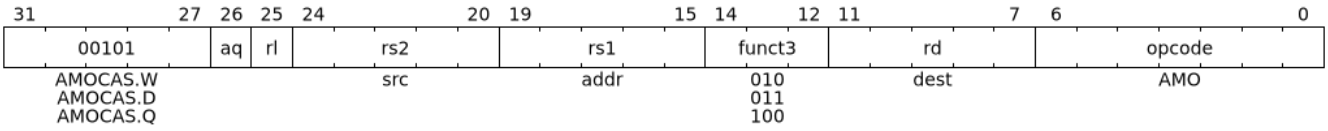
Chapter 1. Introduction

Compare-and-Swap (CAS) provides an easy and typically faster way to perform thread synchronization operations when supported as a hardware instruction. CAS is typically used by lock-free and wait-free algorithms. This extension proposes CAS instructions to operate on 32-bit, 64-bit, and 128-bit (RV64 only) data values. The CAS instruction supports the C++11 atomic compare and exchange operation.

While compare-and-swap for XLEN wide data may be accomplished using LR/SC, the CAS atomic instructions scale better to highly parallel systems than LR/SC. Many lock-free algorithms, such as a lock-free queue, require manipulation of pointer variables. A simple CAS operation may not be sufficient to guard against what is commonly referred to as the ABA problem in such algorithms that manipulate pointer variables. To avoid the ABA problem, the algorithms associate a reference counter with the pointer variable and perform updates using a quadword compare and swap (of both the pointer and the counter). The double and quadword CAS instructions support implementation of algorithms for ABA problem avoidance.

The Zacas extension depends upon the A extension.

Chapter 2. Word/Doubleword/Quadword CAS (AMOCAS.W/D/Q)



For RV32, `AMOCAS.W` atomically loads a 32-bit data value from address in `rs1`, compares the loaded value to the 32-bit value held in `rd` and if the comparison is bitwise equal, then stores the 32-bit value held in `rs2` to the original address in `rs1`. The value loaded from memory is placed into register `rd`. The operation performed by `AMOCAS.W` for RV32 is as follows:

```
temp = mem[X(rs1)]
if ( temp == X(rd) )
    mem[X(rs1)] = X(rs2)
endif
X(rd) = temp
```

For RV64, `AMOCAS.W` atomically loads a 32-bit data value from address in `rs1`, compares the loaded value to the lower 32 bits of the value held in `rd` and if the comparison is bitwise equal, then stores the lower 32 bits of the value held in `rs2` to the original address in `rs1`. The 32-bit value loaded from memory is sign extended and is placed into register `rd`. The operation performed by `AMOCAS.W` for RV64 is as follows:

```
temp[31:0] = mem[X(rs1)]
if ( temp[31:0] == X(rd)[31:0] )
    mem[X(rs1)] = X(rs2)[31:0]
endif
X(rd) = SignExtend(temp[31:0])
```

`AMOCAS.D` is similar to `AMOCAS.W` but operates on 64-bit data values.

For RV32, `AMOCAS.D` atomically loads 64-bits of a data value from address in `rs1`, compares the loaded value to a 64-bit value held in a register pair consisting of `rd` and `rd+1` and if the comparison is bitwise equal, then stores the 64-bit value held in the register pair `rs2` and `rs2+1` to the original address in `rs1`. The value loaded from memory is placed into the register pair `rd` and `rd+1`. The instruction requires the first register in the pair to be even numbered; encodings with odd numbered registers specified in `rs2` and `rd` are reserved. When the first register of a source register pair is `x0`, then both halves of the pair read as zero. When the first register of a destination register pair is `x0`, then the entire register result is discarded and neither destination register is written. The operation performed by `AMOCAS.D` for RV32 is as follows:

```
temp0 = mem[X(rs1)+0]
temp1 = mem[X(rs1)+4]
comp0 = (rd == x0)  ? 0 : X(rd)
```

```
comp1 = (rd == x0)  ? 0 : X(rd+1)
swap0 = (rs2 == x0) ? 0 : X(rs2)
swap1 = (rs2 == x0) ? 0 : X(rs2+1)
if ( temp0 == comp0 ) && ( temp1 == comp1 )
    mem[X(rs1)+0] = swap0
    mem[X(rs1)+4] = swap1
endif
if ( rd != x0 )
    X(rd)   = temp0
    X(rd+1) = temp1
endif
```

For RV64, `AMOCAS.D` atomically loads 64-bits of a data value from address in `rs1`, compares the loaded value to a 64-bit value held in `rd` and if the comparison is bitwise equal, then stores the 64-bit value held in `rs2` to the original address in `rs1`. The value loaded from memory is placed into register `rd`. The operation performed by `AMOCAS.D` for RV64 is as follows:

```
temp = mem[X(rs1)]
if ( temp == X(rd) )
    mem[X(rs1)] = X(rs2)
endif
X(rd) = temp
```

`AMOCAS.Q` (RV64 only) atomically loads 128-bits of a data value from address in `rs1`, compares the loaded value to a 128-bit value held in a register pair consisting of `rd` and `rd+1` and if the comparison is bitwise equal, then stores the 128-bit value held in the register pair `rs2` and `rs2+1` to the original address in `rs1`. The value loaded from memory is placed into the register pair `rd` and `rd+1`. The instruction requires the first register in the pair to be even numbered; encodings with odd numbered registers specified in `rs2` and `rd` are reserved. When the first register of a source register pair is `x0`, then both halves of the pair read as zero. When the first register of a destination register pair is `x0`, then the entire register result is discarded and neither destination register is written. The operation performed by `AMOCAS.Q` is as follows:

```
temp0 = mem[X(rs1)+0]
temp1 = mem[X(rs1)+8]
comp0 = (rd == x0)  ? 0 : X(rd)
comp1 = (rd == x0)  ? 0 : X(rd+1)
swap0 = (rs2 == x0) ? 0 : X(rs2)
swap1 = (rs2 == x0) ? 0 : X(rs2+1)
if ( temp0 == comp0 ) && ( temp1 == comp1 )
    mem[X(rs1)+0] = swap0
    mem[X(rs1)+8] = swap1
endif
if ( rd != x0 )
    X(rd)   = temp0
    X(rd+1) = temp1
endif
```

For a future RV128 extension, AMOCAS.Q would encode a single XLEN=128 register in rs2 and rd.

Just as for AMOs in the A extension, AMOCAS.W/D/Q requires that the address held in rs1 be naturally aligned to the size of the operand (i.e., 16-byte aligned for *quadwords*, eight-byte aligned for *doublewords*, and four-byte aligned for *words*). And the same exception options apply if the address is not naturally aligned.

Just as for AMOs in the A extension, the AMOCAS.W/D/Q optionally provide release consistency semantics, using the aq and rl bits, to help implement multiprocessor synchronization.

Some algorithms may load the previous data value of a memory location into the register used as the compare data value source by a Zacas instruction. When using a Zacas instruction that uses a register pair to source the compare value, the two registers may be loaded using two individual loads. The two individual loads may read an inconsistent pair of values but that is not an issue since the AMOCAS operation itself uses an atomic load-pair from memory to obtain the data value for its comparison.

The following code sequence illustrates a quadword compare-and-swap operation. The registers a6 and a7 that are used to source the compare data value are loaded using two individual loads. The registers a4 and a5 hold the swap data value. The register a0 holds the address of the memory location operated on by the instruction.

```
    # initialize the swap value
    li a4, new_value0
    li a5, new_value1

    # initialize the expected value
    li a2, expected_value0
    li a3, expected_value1
retry:
    # load the current value
    ld a6, 0(a0)
    ld a7, 8(a0)

    # retry if current value not the expected value
    bne a6, a2, retry
    bne a7, a3, retry

    # quadword compare and swap
    amocas.q.aqrl a6, a4, (a0)

    # retry if CAS failed
    bne a6, a2, retry
    bne a7, a3, retry
```

Chapter 3. Additional AMO PMAs

There are four levels of PMA support defined for AMOs in the A extension. Zacas defines three additional levels of support: `AMOCASW`, `AMOCASD`, and `AMOCASQ`.

`AMOCASW` indicates that in addition to instructions indicated by `AMOArithmetic` level support, the `AMOCAS.W` instruction is supported. `AMOCASD` indicates that in addition to instructions indicated by `AMOCASW` level support, the `AMOCAS.D` instruction is supported. `AMOCASQ` indicates that all RISC-V AMOs are supported.

`AMOCASW/D/Q` require `AMOArithmetic` level support as the `AMOCAS.W/D/Q` instructions require ability to perform an arithmetic comparison and a swap operation.