



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

Version 0.1, 4/2023: This document is in development state. See <http://riscv.org/spec-state> for details.

Table of Contents

Preamble..... 1

Copyright and license information..... 2

 Contributors 2

1. Introduction..... 3

2. Word/Doubleword/Quadword CAS (AMOCAS.W/D/Q)..... 4

3. AMO PMA..... 7

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/.

Copyright 2022 by RISC-V International.

Contributors

This RISC-V specification has been contributed to directly or indirectly by: Greg Favor, Ved Shanbhogue

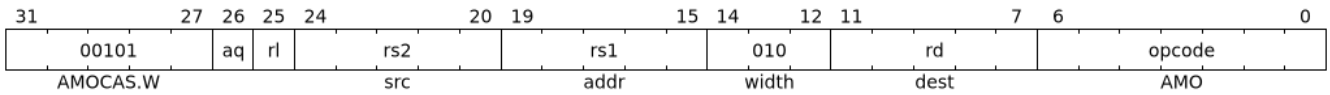
Chapter 1. Introduction

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.

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

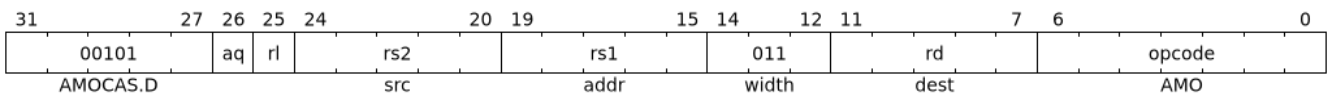
AMOCAS.W atomically loads 32-bits of a data value from address in **rs1**, compares the loaded value to a 32-bit value held in **rd** and if the comparison is bitwise equal, then stores the 32-bit value held in **rs2** to the original address in **rs1**. The value loaded from memory is placed into register **rd**. For RV64, **AMOCAS.W** always sign-extends the value placed in **rd**, and ignores the upper 32 bits of the original value in **rd** and **rs2**.



The operation performed by **AMOCAS.W** is as follows:

```
temp = *[rs1]
if temp == [rd]
    *[rs1] = [rs2]
endif
[rd] = temp
```

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-bits 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 = *([rs1]+0)
temp1 = *([rs1]+4)

comp0 = (rd == 0) ? 0 : [rd];
comp1 = (rd == 0) ? 0 : [rd+1];

swap0 = (rs2 == 0) ? 0 : [rs2];
swap1 = (rs2 == 0) ? 0 : [rs2+1];

If (temp0 == comp0) && (temp1 == comp1)
    *([rs1]+0) = swap0
```

```

        *([rs1]+4) = swap1
    endif

    if ( rd != x0 )
        [rd]    = temp0
        [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**.

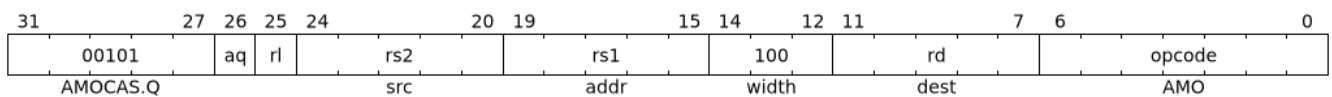
The operation performed by **AMOCAS.D** for RV64 is as follows:

```

temp = *[rs1]
if temp == [rd]
    *[rs1] = [rs2]
endif
[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-bits 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:

Listing 1. AMOCAS.Q operation

```

temp0 = *([rs1]+0)
temp1 = *([rs1]+8)

comp0 = (rd == 0) ? 0 : [rd];
comp1 = (rd == 0) ? 0 : [rd+1];

swap0 = (rs2 == 0) ? 0 : [rs2];
swap1 = (rs2 == 0) ? 0 : [rs2+1];

If (temp0 == comp0) && (temp1 == comp1)
    *([rs1]+0) = swap0
    *([rs1]+8) = swap1
endif

```

```

if ( rd != x0 )
    [rd] = temp0
    [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 128-bit words, eight-byte aligned for 64-bit words, and four-byte aligned for 32-bit words). 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. If the misaligned access could be emulated then an address-misaligned exception may be generated.

The draft “Zam” extension, described in Chapter 23, relaxes this requirement and specifies the semantics of misaligned AMOs.

Just as for AMOs in the A extension, the **amocas.w/d/q** optionally provides release consistency semantics to help implement multiprocessor synchronization. If the **aq** bit is set, then no later memory operations in this RISC-V hart can be observed to take place before the **amocas.w/d/q**. Conversely, if the **rl** bit is set, then other RISC-V harts will not observe the instruction before memory accesses preceding the instruction in this RISC-V hart. Setting both the **aq** and the **rl** bit on an AMO makes the sequence sequentially consistent, meaning that it cannot be reordered with earlier or later memory operations from the same hart.

Chapter 3. AMO PMA

There are four levels of support defined within 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` requires `AMOArithmetic` level support as the `amocas.w/d/q` instructions require ability to perform an arithmetic comparison and a swap operation.