Load/Store Pair for RV32 (Zilsd & Zclsd)

Version v1.0, 2025-02-21: Ratified

Table of Contents

Preamble

This document is Ratified

No changes are allowed. Any desired or needed changes can be the subject of a follow-on new extension. Ratified extensions are never revised

Copyright and license information

Contributors

This RISC-V specification has been contributed to directly or indirectly by: Christian Herber, Torbjørn Viem Ness, Tariq Kurd.

Chapter 1. Introduction

This specification contains two RV32-only extensions, which add load and store instructions using register pairs. It does so by reusing existing instruction encodings which are RV64-only. The specification defines 32-bit encodings (Zilsd extension) and 16-bit encodings (Zclsd).

Load and store instructions will use the same definition of even-odd pairs as defined by the Zdinx extension.

The extension improves static code density, by replacing two separate load or store instructions with a single one. In addition, it can provide a performance improvement for implementations that can make use of a wider than XLEN memory interface.

Chapter 2. Load/Store pair

The Zilsd & Zclsd extensions provide load/store pair instructions for RV32, reusing the existing RV64 doubleword load/store instruction encodings.

Operands containing `src` for store instructions and `dest` for load instructions are held in aligned `x` -register pairs, i.e., register numbers must be even. Use of misaligned (odd-numbered) registers for these operands is *reserved*.

Regardless of endianness, the lower-numbered register holds the low-order bits, and the higher-numbered register holds the high-order bits: e.g., bits 31:0 of an operand in Zilsd might be held in register `x14`, with bits 63:32 of that operand held in `x15`.

2.1. Load/Store pair instructions (Zilsd)

The Zilsd extension adds the following RV32-only instructions:

RV32	RV64	Mnemonic	Instruction
yes	no	ld rd, offset(rs1)	Load doubleword to register pair, 32-bit encoding
yes	no	sd rs2, offset(rs1)	Store doubleword from register pair, 32-bit encoding

As the access size is 64-bit, accesses are only considered naturally aligned for effective addresses that are a multiple of 8. In this case, these instructions are guaranteed to not raise an address-misaligned exception. Even if naturally aligned, the memory access might not be performed atomically.

If the effective address is a multiple of 4, then each word access is required to be performed atomically.

The following table summarizes the required behavior:

Alignment	Word accesses guaranteed atomic?	Can cause misaligned trap?
8B	yes	no
4B not 8B	yes	yes
else	no	yes

To ensure resumable trap handling is possible for the load instructions, the base register must have its original value if a trap is taken. The other register in the pair can have been updated. This affects x2 for the stack pointer relative instruction and rs1 otherwise.

If an implementation performs a doubleword load access atomically and the register file implements writeback for even/odd register pairs, the mentioned atomicity requirements are inherently fulfilled. Otherwise, an implementation either needs to delay the writeback until the write can be performed atomically, or order sequential writes to the registers to ensure the requirement above is satisfied.

2.2. Compressed Load/Store pair instructions (Zclsd)

Zclsd depends on Zilsd and Zca. It has overlapping encodings with Zcf and is thus incompatible with Zcf.

Zclsd adds the following RV32-only instructions:

RV32	RV64	Mnemonic	Instruction
yes	no	c.ldsp rd, offset(sp)	Stack-pointer based load doubleword to register pair, 16-bit encoding
yes	no	c.sdsp rs2, offset(sp)	Stack-pointer based store doubleword from register pair, 16-bit encoding
yes	no	c.ld rd', offset(rs1')	Load doubleword to register pair, 16-bit encoding
yes	no	c.sd rs2', offset(rs1')	Store doubleword from register pair, 16-bit encoding

2.3. Use of x0 as operand

LD instructions with destination `x0` are processed as any other load, but the result is discarded entirely and x1 is not written. For C.LDSP, usage of `x0` as the destination is reserved.

If using `x0` as `src` of SD or C.SDSP, the entire 64-bit operand is zero — i.e., register `x1` is not accessed.

C.LD and C.SD instructions can only use `x8-15`.

2.4. Exception Handling

For the purposes of RVWMO and exception handling, LD and SD instructions are considered to be misaligned loads and stores, with one additional constraint: an LD or SD instruction whose effective address is a multiple of 4 gives rise to two 4-byte memory operations.

This definition permits LD and SD instructions giving rise to exactly one memory access, regardless of alignment. If instructions with 4-byte-aligned effective address are decomposed into two 32b operations, there is no constraint on the order in which the operations are performed and each operation is guaranteed to be atomic. These decomposed sequences are interruptible. Exceptions might occur on subsequent operations, making the effects of previous operations within the same instruction visible.

Software should make no assumptions about the number or order of accesses these instructions might give rise to, beyond the 4-byte constraint mentioned above. For example, an interrupted store might overwrite the same bytes upon return from the interrupt handler.

2.5. Instructions

2.5.1. ld

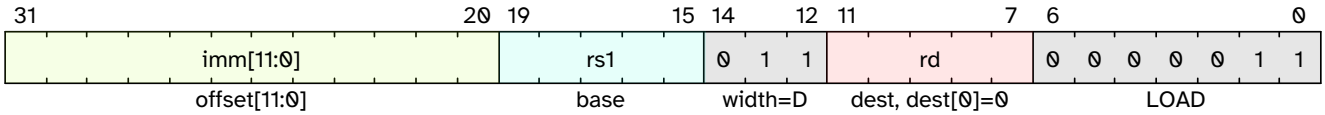
Synopsis

Load doubleword to even/odd register pair, 32-bit encoding

Mnemonic

ld rd, offset(rs1)

Encoding (RV32)



Description

Loads a 64-bit value into registers `rd` and `rd+1`. The effective address is obtained by adding register rs1 to the sign-extended 12-bit offset.

Included in: Zilsd

2.5.2. sd

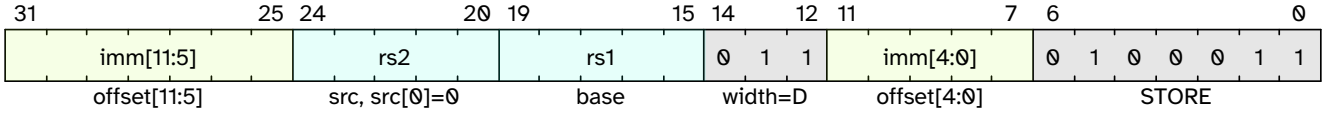
Synopsis

Store doubleword from even/odd register pair, 32-bit encoding

Mnemonic

sd rs2, offset(rs1)

Encoding (RV32)



Description

Stores a 64-bit value from registers `rs2` and `rs2+1`. The effective address is obtained by adding register rs1 to the sign-extended 12-bit offset.

Included in: Zilsd

2.5.3. c.ldsp

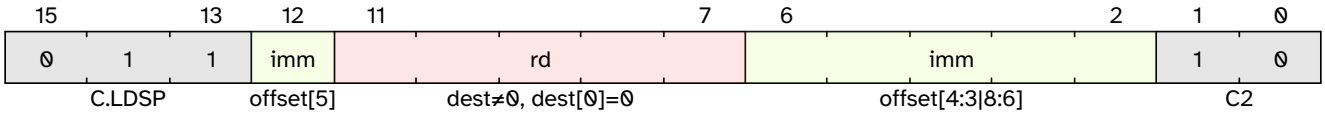
Synopsis

Stack-pointer based load doubleword to even/odd register pair, 16-bit encoding

Mnemonic

c.ldsp rd, offset(sp)

Encoding (RV32)



Description

Loads stack-pointer relative 64-bit value into registers `rd'` and `rd'+1`. It computes its effective address by adding the zero-extended offset, scaled by 8, to the stack pointer, `x2`. It expands to `ld rd, offset(x2)`. C.LDSP is only valid when *rd*≠x0; the code points with *rd*=x0 are reserved.

Included in: Zclsd

2.5.4. c.sdsp

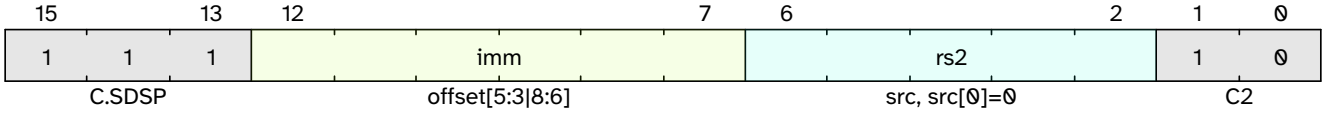
Synopsis

Stack-pointer based store doubleword from even/odd register pair, 16-bit encoding

Mnemonic

c.sdsp rs2, offset(sp)

Encoding (RV32)



Description

Stores a stack-pointer relative 64-bit value from registers `rs2'` and `rs2'+1`. It computes an effective address by adding the *zero*-extended offset, scaled by 8, to the stack pointer, `x2`. It expands to `sd rs2, offset(x2)`.

Included in: Zclsd

2.5.5. c.ld

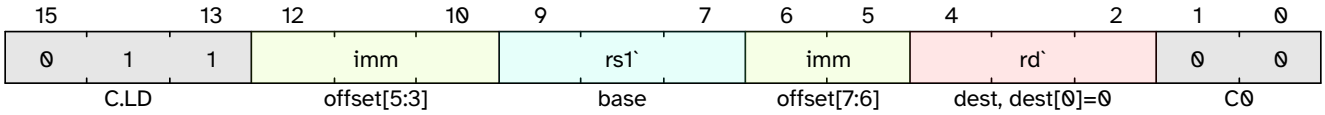
Synopsis

Load doubleword to even/odd register pair, 16-bit encoding

Mnemonic

c.ld rd', offset(rs1')

Encoding (RV32)



Description

Loads a 64-bit value into registers `rd'` and `rd'+1`. It computes an effective address by adding the zero-extended offset, scaled by 8, to the base address in register rs1'.

Included in: Zclsd

2.5.6. c.sd

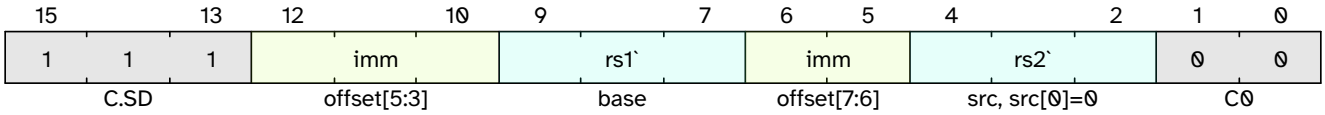
Synopsis

Store doubleword from even/odd register pair, 16-bit encoding

Mnemonic

c.sd rs2', offset(rs1')

Encoding (RV32)



Description

Stores a 64-bit value from registers `rs2'` and `rs2'+1`. It computes an effective address by adding the zero-extended offset, scaled by 8, to the base address in register rs1'. It expands to `sd rs2', offset(rs1')`.

Included in: Zclsd