Project Report

Lars Pfau

Measuring the Performance Overhead of RIOT Running in RISC-V User-Mode

Supervision: Prof. Dr. Thomas C. Schmidt
Submitted: February 15th, 2024

Faculty of Engineering and Computer Science
Department Computer Science

Contents

The RISC-V ISA allows for the implementation of trusted execution environments on microcontrollers with physical memory protection and user-mode. To get towards the implementation of such a TEE, this work investigates the feasibility of running RIOT OS in user-mode on a SiFive FE310 RISC-V microcontroller. A minimal M-mode firmware and RIOT port are developed and evaluated using several performance benchmarks. The results show a wide range of performance penalties from 0% in CoreMark to 90% in some microbenchmarks. It is concluded that TEE technology on RISC-V microcontrollers with user-mode and PMP is in general feasible.

Keywords: IoT Security, RISC-V

1 Introduction

With the increasing number of attacks on IoT devices, the security of Internet-connected embedded systems is becoming an area of growing concern. Trusted Execution Environments (TEE) is a technology that adresses growing cyber-security risks in IoT devices [13]. TEEs can be implemented by running the real time operating system (RTOS) or bare-metal application at a lower privilege mode so that a secure firmare in the highest privilege mode can restrict memory access from the application [9].

The RISC-V ISA specification [14] used to contain an extension for user-mode interrupts to support TEEs in RISC-V microcontrollers. This so-called N-extension allowed for efficient handling of interrupts on secure embedded systems that only support two privilege modes: machine (M-mode) and user (U-mode). However, with the recent deprecation of the N-extension, the RISC-V ISA no longer has a hardware mechanism to delegate interrupt handling to user-mode.

This work investigates the feasibility of running an RTOS in user-mode on RISC-V M+U-mode secure embedded systems without the N-extension. To achieve this, a prototype of a secure firmware is developed and RIOT OS [1] is modified to run in user-mode. The prototype is evaluated using the RIOT benchmark suite by measuring the performance overhead of the secure firmware.

2 Background + Related Work

2.1 RISC-V

RISC-V is an open instruction set architecture (ISA) originally developed and published by Waterman et. al. at UC Berkeley [14]. The RISC-V privileged architecture [15] defines multiple privilege modes such as machine-mode (M-mode), hypervisor-mode (HS-mode), supervisor-mode (S-mode), and user-mode (U-mode), which enable a wide range of implementations ranging from simple 32-bit microcontrollers to 64-bit data center CPUs.

For embedded systems, the RISC-V privileged specification recommends only supporting M-mode on simple embedded systems and M+U-mode on secure embedded systems. In addition, the RISC-V instruction set manual defines a mechanism named Physical Memory Protection (PMP), which allows M-mode software to restrict access of less privileged software to the physical address space [15].

2.2 RIOT

RIOT [1] is a real-time operating system (RTOS) designed for microcontrollers used in IoT applications. RIOT supports 32-bit RISC-V CPUs and currently has support for the following RISC-V microcontrollers:

- Espressif ESP32-C3 [3],
- GigaDevice GD32VF103 [4] and
- SiFive FE310 [12].

All listed microcontrollers feature M+U-mode and the FE310 also features PMP. The RIOT RISC-V port does not take advantage of the ability to run software in U-mode, instead running both the kernel and applications in M-mode. As an M-mode only operating system, RIOT heavily uses the *mstatus* control and status register (CSR) for globally disabling and re-enabling interrupts to achieve mutual exclusion – a CSR that can only be accessed from M-mode. RIOT also requires direct access to M-mode CSRs for interrupt handling and configuration.

2.3 RISC-V Traps in RIOT

The RISC-V privileged ISA defines two kinds of traps. The first type is exceptions, which are triggered by memory access faults or by the execution of environment calls (ECALL), breakpoints (EBREAK), and other illegal instructions. The second type is interrupts, which can be software interrupts, timer interrupts or external interrupts [15].

In the case of the SiFive FE310, interrupt configuration and handling is done using hardware external to the CPU core itself: A core-local interruptor (CLINT) [12] and a platform-level interrupt controller (PLIC) [8], which are controlled by memory-mapped registers.

M-mode CSRs are still required for interrupt handling on the ISA level. These CSRs include:

- *mtvec* – to set the address of the trap handler.
- *mcause* – to get the type of trap.
- *mepc* – to read/set the program counter when entering/exiting the trap handler.
- *mstatus* – to read/set the privilege mode when entering/exiting the trap handler.
- *mie* – to mask interrupts [15].

When an interrupt is triggered, a RISC-V CPU will perform the following steps:

1. Interrupts are disabled globally, and the previous interrupt enable state is saved in *mstatus*.
2. Privilege mode is set to M-mode, and the previous privilege mode is saved in *mstatus*.
3. *mcause* is set to the interrupt cause.
4. *mepc* is set to the current value of the program counter.
5. The program counter is set to the address of the trap handler specified in *mtvec* [12].

At this point, the control flow has been transferred to the trap handler of RIOT, which then pushes the general-purpose registers onto the stack of the interrupted process and switches to the interrupt-stack. Depending on the value of the *mcause*-CSR, the trap handler then dispatches the interrupt request to the responsible interrupt handler.

When the interrupt handler processed the request, it returns to the trap handler. If a context switch has been requested, the scheduler is invoked. Then, *mepc* is set to the address of the next process and the general-purpose registers of that process are restored. The trap handler terminates by executing an *mret* instruction, which causes the CPU to:

6. Restore the privilege mode to the previous privilege saved in *mstatus*.
7. Re-enable interrupts globally depending on the previous interrupt enable state saved in *mstatus*.
8. Jump to the address specified in *mepc* [12].

Interrupts can be enabled and disabled globally using the global interrupt enable bit in the *mstatus*-CSR or using individual interrupt enable bits in the *mie*-CSR. There is no mechanism to disable exceptions [15].

2.4 Related Work

In [6], Pinto et al. describe the method for handling interrupts in user-mode, which they implemented in MultiZone TEE. Similar to this work, MultiZone also supports M+U-mode RISC-V microcontrollers. The authors emphasize the importance of the N-extension, but also mention the possibility of using trap-and-emulate to implement it in software. However, in their work, they do not provide any data to show what the performance impact of emulating the N-extension is.

Henrik Karlsson developed a secure monitor in his masters thesis [5] that is API-compatible with MultiZone, targeting the SiFive FE310 microcontroller. He evaluated the implementation by measuring the performance overhead of the secure monitor switching between secure partitions. The results show a performance overhead of only a few hundred to a thousand clock cycles when the secure monitor is cached in the instruction cache. However, in his work, Karlsson does not measure the performance overhead of an application running in user-mode.

3 Problem Statement

Implementing a TEE on RISC-V M+U-mode secure embedded systems with only two privilege modes can only be achieved with the secure monitor running in M-mode and the RTOS running in U-mode. Because RIOT relies on access to M-mode CSRs for interrupt enable/disable and interrupt processing, access to CSRs has to be emulated by the firmware. This raises the question of how much performance overhead is caused by the emulation. The data can be used for future work to determine whether it is feasible to implement a secure monitor for M+U-mode RISC-V microcontrollers, or if M+S-mode systems and co-processors should be targeted instead.

4 Implementation

To measure the performance overhead, the RIOT target for the SiFive FE310 is modified to run in U-mode and a minimal M-mode firmware is developed. The M-mode firmware handles system startup, access to privileged instructions, and dispatching of interrupt requests to U-mode. Each of these functions is described in the following sections.

4.1 System Reset

The startup procedure of RIOT is modified to start the RTOS in U-mode. When the CPU is reset, the program counter jumps to a platform-defined address in M-mode. From there the initialization of the M-mode firmware begins. The firmware initializes the RAM by loading the data section from its load address (LMA) region to its virtual address (VMA) region and clearing the section for uninitialized data (BSS). The machine-level CSRs are initialized by configuring the interrupt handlers in the *mtvec*-CSR.

The PMP regions are configured to prevent U-mode software from accessing code and data of the M-mode firmware using the configuration shown in table 1. The settings grant read, write, and execute permissions for most of the address space to the U-mode software. However all access permissions are revoked for the RAM and ROM sections that belong to the M-mode firmware.

Table 1: PMP Region configuration

PMP Region	PMP Type	Privilege Mode	Description	M-Mode Sections	R	W	X
0	TOR	M+U			1	1	1
1	TOR	M	M-Mode ROM	.init .text .rodata .srodata	0	0	0
2	TOR	M+U			1	1	1
3	TOR	M	M-Mode RAM (LMA)	.data .sdata	0	0	0
4	TOR	M+U			1	1	1
5	TOR	M	M-Mode RAM (VMA)	.data .sdata .bss .sbss	0	0	0
6	TOR	M+U			1	1	1

The linker script is modified to group all code and data of the M-mode firmware together, and export the start and end addresses of all PMP regions.

4.2 Privileged Instructions

The main obstacle for RIOT to run in U-mode is that the OS loses the ability to execute privileged instructions. RIOT requires access to privileged instruction in the interrupt handler. RIOT also frequently accesses privileged CSRs to disable interrupts for mutual exclusion.

One way to overcome this is to use trap-and-emulate. This means the U-mode software remains unchanged such that privileged instructions cause illegal instruction exceptions, that are transparently handled by the M-mode firmware. The disadvantage of this approach is its complexity as CSR instructions have multiple instruction formats.

A much simpler approach is to use a supervisor binary interface (SBI), similar to [5, 6]. With this approach, the unprivileged U-mode software is modified to interface directly with the privileged M-mode firmware using environment calls (ECALLs).

The ECALL interface, that is used, directly maps to the RIOT API functions: irq_disable, irq_restore, and irq_is_enabled. It also contains a function to trigger software interrupts for the RIOT thread_yield_higher function. Other privileged instructions such as the optional *WFI* (wait for interrupt) remain unimplemented and are removed from the RIOT port.

4.3 Trap Handling

The M-mode firmware implements a simple strategy for trap handling, where all exceptions are handled by the firmware in M-mode and all interrupts are handled by the RTOS in U-mode. There are two challenges to overcome: First, the U-mode interrupt handler of the RTOS does not have access to the machine-level CSRs necessary for interrupt processing. This is overcome by mapping the required CSRs into memory and updating their values at interrupt entry and exit. As a result the U-mode environment is not transparent to the RTOS. However, this is expected to show much lower overhead compared to using trap-and-emulate for CSR instructions. The RIOT interrupt handler requires access to the following CSRs:

1. *mcause* (ro): To determine the interrupt cause.
2. *mie* (rw): To mask a pending interrupt cause.
3. *mhartid* (ro): To claim a pending external interrupt.
4. *mepc* (rw): To set the return address of the interrupt handler.

The second challenge is entering and exiting interrupts in U-mode. In the RISC-V ISA, a trap causes the CPU to switch to M-mode. Also, the *mret* instruction, used to return from traps, is a privileged instruction that can only be executed in M-mode. To overcome this, the M-mode firmware must dispatch interrupt requests to the U-mode RTOS and provide a method for the RTOS to return from its interrupt handler. This is accomplished with the sequence shown in Figure 1.

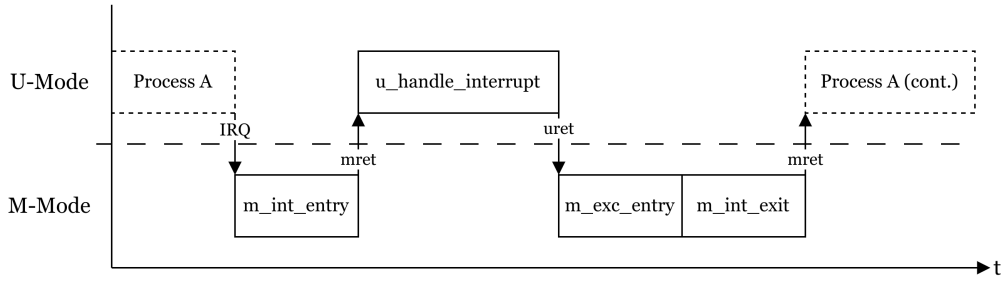

Figure 1: Interrupt dispatching and handling in U-mode

When an interrupt is triggered, the M-mode interrupt entry handler is called. This M-mode handler stores the CSRs for interrupt processing into shared memory and calls the U-mode interrupt handler of the RTOS. The RTOS then processes the interrupt, which may include modifying the *mepc* and *mie* CSR values in memory. When the U-mode handler returns, it executes an *uret*-instruciton which triggers an illegal instruction exception. This causes the exception handler of the M-mode firmware to run. The exception handler then calls the interrupt exit function which restores the CSR values from memory and executes the *mret*.

5 Evaluation

To evaluate the performance penalty caused by the M-mode firmware, a set of comparative benchmarks were performed. All measurements were carried out using a SiFive FE310 microcontroller on a SparkFun RED-V Thing Plus development board. The benchmarks used are CoreMark [2] and the benchmark suite provided by RIOT [7]. All benchmarks were run with both the default M-mode RISC-V target and the modified U-mode target. The benchmark results are compared to show the performance loss/gain.

Since the modified RIOT U-mode port does not support the *WFI* instruction, the default M-mode target has been modified to also not use *WFI* for fairness.

5.1 CoreMark

First, CoreMark is used to verify that the microcontroller used performs as expected and to show that the privilege mode has no impact on the raw computational performance of the CPU. The experiment consists of a total of 10 runs with RIOT and CoreMark

running with the default version of the RIOT hifive1b target in M-mode as well as the modified target of RIOT running in U-mode. The tests are run with both a performance optimized binary (-O3), recommended by the vendor [10], and a size optimized binary (-Os), which is the default for RIOT.

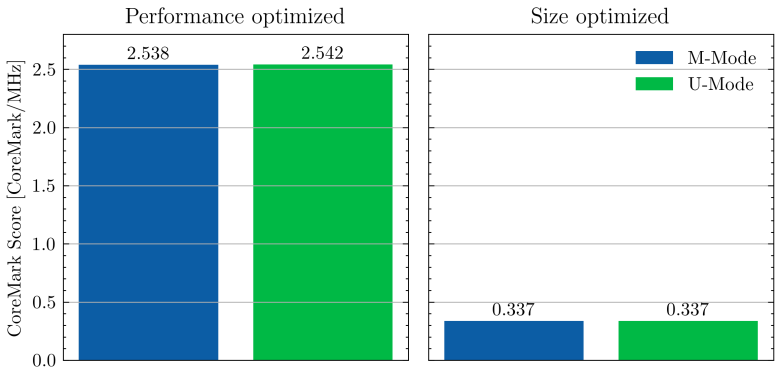

Figure 2: CoreMark Benchmark: Both the performance and size optimized compiler profiles show no significant difference in the CoreMark score.

As can be seen in Figure 2, the benchmark score when compiled with performance optimizations is within 10% of the 2.73 CoreMark/MHz performance claimed by SiFive [11], validating the correct operation of the experiment setup. The results also show that for both optimization profiles the performance is the same between M-mode and U-mode. This confirms that the raw compute performance is not impacted by the privilege mode the benchmark runs in. There is no significant impact caused by the M-mode firmware.

5.2 Microbenchmarks

Next, the performance overhead of the SBI is measured using API microbenchmarks that require access to privileged instructions. These functions include irq_disable, irq_restore, and irq_is_enabled, as well as some core API functions that use those functions. The measurement is done using a modified version of the RIOT runtime_coreapis benchmark, which measures the time to execute each functions 1 million times. The number of iterations has been adjusted to ensure that all benchmarks are run for at least one second. All measurements were repeated 10 times. The results showed a variance of about 0,1%. The expectation is for the througput of the functions to be reduced to a fraction of the initial performance. This is because the runtime of the irq_* functions

increases from a single CSR instruction to an ECALL of more than 20 instructions, as well as additional pipeline flushes.

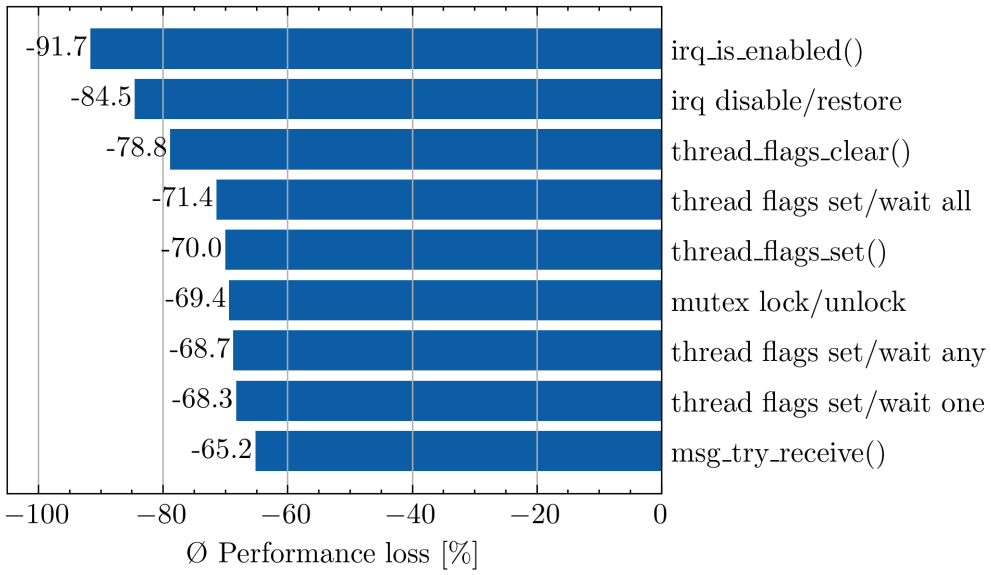

Figure 3: Microbenchmarks: Some API functions of RIOT show a significant performance loss between 92% and 65%.

Figure 3 shows the difference in runtime of going through the SBI compared to executing privileged instructions directly. The results show a performance loss between 91% and 65% depending on the function used. The interrupt processing API is affected the most, while IPC mechanisms such as thread flags, mutex, and messages are less affected.

5.3 Context-Switch Benchmarks

Finally, the performance overhead of context switches is measured, which is caused by the modified trap handling. This includes the overhead of yielding to another process, as well as interrupt entry, interrupt handling, and interrupt exit. To achieve this, the RIOT thread_yield_pingpong and msg_pingpong benchmarks are used.

The thread_yield_pingpong benchmark starts two threads that continuously yield to the other process. It measures the number of context switches that are completed in one second. The msg_pingpong benchmark is functionally similar, but uses the RIOT messaging IPC mechanism instead.

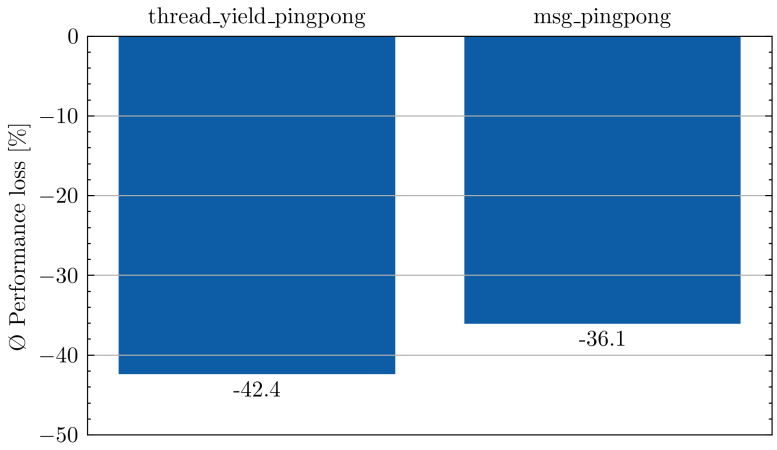

Figure 4: Context-switch benchmark: Performing a context switch, as a result of yielding or blocking IPC mechanisms, is slowed down by up to 42%

Figure 4 shows the difference in the benchmark score with and without the additional overhead of the M-mode firmware. The thread_yield_pingpong result shows a reduction of 42.4%, meaning that the throughput of context switches per second has decreased by 42.4%. The throughput of messages that can be sent per second has decreased by 36%.

6 Conclusion + Outlook

The deprecation of the N-extension from the RISC-V specification has raised the question of whether or not it is feasible to implement trusted execution environments on RISC-V secure embedded systems without user-mode interrupts. To answer this question, a minimal M-mode firmware was developed, enabling RIOT to run in U-mode.

A series of comparative benchmarks were performed to evaluate the performance penalty of RIOT running in U-mode. The results show that there is no performance loss in terms of raw computational performance and a performance loss of 36% - 91% for context switches, IPC-mechanisms, and other API functions that depend on the ability to globally disable interrupts.

Given these results, it is reasonable to expect that the performance loss of a real-world application will be a low double digit percentage. This is assuming the application is

a mix of compute heavy operations, represented by CoreMark, which showed no performance penalty, and IO operations represented by the IPC mechanisms and context switch benchmarks. However, the feasibility still depends on the exact software used.

In future work, the development of a TEE targeting RISC-V microcontrollers with U-mode and PMP can be pursued. Further research questions that should be answered is the energy consumption and memory footprint of such a TEE. In addition, a thorough security analysis of the firmware should be performed.

References

[1] Emmanuel Baccelli, Cenk Gündogan, Oliver Hahm, Peter Kietzmann, Martine Lenders, Hauke Petersen, Kaspar Schleiser, Thomas C. Schmidt, and Matthias Wählisch. RIOT: An Open Source Operating System for Low-End Embedded Devices in the IoT. *IEEE Internet of Things Journal*, 5(6):4428–4440, December 2018. URL http://doi.org/10.1109/JIOT.2018.2815038.

[2] EEMBC. Coremark®. URL https://www.eembc.org/coremark/. Accessed 2024-02-01.

[3] Espressif Systems. ESP32-C3 Series Datasheet v1.5, 2023. URL https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf. Accessed 2024-02-01.

[4] GigaDevice Semiconductor Inc. GD32VF103 RISC-V 32-bit MCU User Manual Revision 1.4, June 2022. URL https://www.gigadevice.com.cn/Public/Uploads/uploadfile/files/20230209/GD32VF103_User_Manual_EN_Rev1.4.pdf. Accessed 2024-02-01.

[5] Henrik Karlsson. OpenMZ: a C implementation of the MultiZone API. Master's thesis, KTH, School of Electrical Engineering and Computer Science (EECS), 2020. URL https://www.diva-portal.org/smash/get/diva2:1466978/FULLTEXT01.pdf. Accessed 2024-02-01.

[6] Sandro Pinto and Cesare Garlati. User Mode Interrupts: A Must for Securing Embedded Systems. *Embedded World 2019*, 2019. URL https://www.researchgate.net/publication/331529294_User_

Mode_Interrupts_A_Must_for_Securing_Embedded_Systems. Acessed 2024-02-01.

[7] RIOT OS. RIOT Benchmark Tests. URL https://github.com/RIOT-OS/RIOT/tree/master/tests/bench. Accessed 2024-02-01.

[8] RISC-V International. RISC-V Platform-Level Interrupt Controller Specification. URL https://github.com/riscv/riscv-plic-spec/releases/download/1.0.0/riscv-plic-1.0.0.pdf. Accessed 2024-02-01.

[9] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. The Dual-Execution-Environment Approach: Analysis and Comparative Evaluation. In *ICT Systems Security and Privacy Protection*, pages 557–570. Springer International Publishing, 2015. doi:10.1007/978-3-319-18467-8_37.

[10] SiFive Inc. SiFive Freedom E SDK. URL https://github.com/sifive/freedom-e-sdk. Accessed 2024-02-01.

[11] SiFive Inc. SiFive FE310-G002 Datasheet v1p2, March 2021. URL https://sifive.cdn.prismic.io/sifive/4999db8a-432f-45e4-bab2-57007eed0a43_fe310-g002-datasheet-v1p2.pdf. Accessed 2024-02-01.

[12] SiFive Inc. SiFive FE310-G002 Manual v1p5, September 2022. URL https://sifive.cdn.prismic.io/sifive/034760b5-ac6a-4b1c-911c-f4148bb2c4a5_fe310-g002-v1p5.pdf. Accessed 2024-02-01.

[13] Dalton Cezane Gomes Valadares, Newton Carlos Will, Jean Caminha, Mirko Barbosa Perkusich, Angelo Perkusich, and Kyller Costa Gorgonio. Systematic Literature Review on the Use of Trusted Execution Environments to Protect Cloud/Fog-Based Internet of Things Applications. *IEEE Access*, 9:80953–80969, 2021. doi:10.1109/access.2021.3085524.

[14] Andrew Waterman and Krste Asanović. The RISC-V Instruction Set Manual Volume I: Unprivileged ISA, Document Version 20191213, December 2019. URL https://github.com/riscv/riscv-isa-manual/releases/tag/Ratified-IMAFDQC. Accessed 2024-02-01.

[15] Andrew Waterman, Krste Asanović, and John Hauser. The RISC-V Instruction Set Manual Volume II: Privileged Architecture, Document Version 20211203, December

2021. URL https://github.com/riscv/riscv-isa-manual/releases/tag/Priv-v1.12. Accessed 2024-02-01.