Implementing Runtime-Configurable Endianness in RISC-V: Challenges and Solutions

Ben Dooks[1], Lawrence Hunter[1], and Roan Richmond[1]

[1]Codethink Ltd, Manchester, United Kingdom
{first}.{last}@codethink.co.uk

Abstract

The RISC-V Privileged Specification introduced dynamic endianness switching in version 1.12, though no commercial hardware yet supports it. This work extends QEMU to enable big-endian execution, allowing the booting of a big-endian Linux system with OpenSBI. Modifications were required across QEMU, OpenSBI, the Linux kernel, and supporting libraries to ensure correct memory operations, instruction encoding, and runtime patching. The project demonstrates the feasibility of big-endian support for RISC-V, providing a foundation for future hardware and software development.

Introduction

The RISC-V Privileged Specification includes endianness control, allowing runtime switching between big and little endian memory read and writes [1]. While introduced in privileged specification version 1.12 (Late 2021), no commercially available hardware implements it, unlike ARM [2, 3]. The benefit of endian control is primarily for specialised applications such as digital signal processing where big endian allows for optimised byte operations but little endian provides much wider software compatibility, and therefore being able to switch is a useful feature [4].

The popular system emulation tool QEMU does not implement this aspect of the specification. The benefit of QEMU and similar tools are that they allow for rapid software development even before Field Programmable Gate Arrays (FPGAs) are available. The work performed involved adding endianness control support to QEMU using a little endian target, which is then switched to big endian at runtime, in order to boot a big endian Linux with a big endian OpenSBI. Aside from QEMU the entire software stack must be modified to allow big endian booting and therefore it was required to modify OpenSBI, the Linux Kernel, libraries such as uclibc, and kvmtool. The status of upstreaming efforts is also reported.

QEMU

The QEMU implementation required the addition of the *{M,VS,S,U}BE {m,h,s,u}status* register flags and handling of the endianness switch (including the required *sfence.vma*), as well as modifications to load and store operations. Additionally, the atomic load/store operations added in the *Zacaz* extension required similar modifications.

Endianness control was achieved by modifying the CPU state to include a *BE_DATA* Translation Block (TB) flag. The specification permits modifications to this flag only via a write to the *mstatus* register. The *{h,s,u}status*'s *{VS,U,S}BE* serve as read-only clones of the *MBE*. The TB required further modifications to ensure the correct pointer endianness.

Page-table code was modified to follow the same endianness as the *S* or *HS* modes in the specification. Any code creating or using page table entries will now see the entries in the correct endian format.

OpenSBI

OpenSBI modifications were necessary to switch the endianness at boot and handle its effects. Compiler flags and Memory Mapped Input Output (MMIO) load and store operations were modified to correctly handle endianness. While the boot log did not require changes, it was updated to explicitly indicate that OpenSBI was a big endian build. Additionally, some initialisation code was modified since QEMU hands control over with data written in little endian.

Linux kernel

Similar to OpenSBI, the Linux kernel required modifications to compiler flags and endianness handling in load and store operations. The build configuration system was updated to allow the selection of big endian, disabling unsupported options such as the vector instructions, which gcc does not currently support compiling as big endian. It was decided that addressing such a limitation was not in scope.

Any code that performed multi-item memory operations, such as *memcpy*, *strlen*, and *uaccess*, were adjusted to accommodate endianness changes. This

was achieved either by disabling the optimised variants for later fixing or modifying header files to use the appropriate endian version.

Significant modifications were necessary for code accessing or generating instructions, as these are always represented in little endian format. Adjustments were made to the trap handler, vector handling, kprobes, static branch modifier code, and the Berkeley Packet Filter (BPF) to allow Just-In-Time (JIT) insertions to function correctly. The Linux kernel uses runtime instruction modification for features such as runtime variant patching and efficient runtime code selection. Without proper endianness handling, these mechanisms would fail, leading to kernel crashes.

Another area that required modification was the creation of custom instructions. In some cases, instructions such as a *pause* are generated directly by using the integer value of the instruction, as older toolchains - still supported by the build system - do not include these instructions natively. Identifying these instructions were challenging as sometimes an endian swapped integer could appear to be another valid instruction which performed a different action. This could bypass the illegal instruction trap leading to unpredictable failures later in execution. One such failure scenario involved unintended modifications to the frame pointer, which is a vital component in function call management and stack unwinding. If incorrectly altered, the frame pointer could lead to incorrect stack traces, misaligned memory access, and difficult to diagnose crashes.

Headers and libraries

The Linux kernel exports headers for endian conversion, such as network byte ordering macros like *htons* and *htonl*, which required updates to ensure correct endianness handling. Additional conversion macros, such as *cpu_to_{le,be}{16,32,64}* and their reverses, were also adjusted.

Some libraries, including uclibc, assumed a little endian system. Modifications were necessary to allow builds and ensure the correct header selection.

kvmtool

PCI read and write operations required modifications to handle endianness for the hypervisor, including the introduction of a new function to configure endianness. While kvmtool includes endian conversion for most little endian IO operations, issues persist in areas such as *virtio*. These issues can be bypassed using device trees instead of PCI for exporting devices into the VM, though future fixes are necessary..

Achievements

A big endian Linux system has been booted to userland under a runtime configurable RISC-V target switched to big endian. The kernel supports BPF, ftrace, and kprobes, and KVM can run in either big or little endian mode. Tooling such as GCC and binutils have been verified, although some bugs remain in GCC and glibc within buildroot.

Upstream

Feedback has been requested from the Linux, QEMU, and OpenSBI mailing lists. No response has been received from QEMU or OpenSBI. Kernel maintainers have expressed concerns regarding the maintenance burden but are open to inclusion if hardware support becomes available. The current work is available at https://gitlab.com/CodethinkLabs/riscv_bigendian..

Future

The project has demonstrated feasibility and provides a foundation for future development. When hardware capable of configurable endianness becomes available, further testing and issue resolution will be required. Limited resources were available to carry out extensive testing, therefore more bugs are expected to be found once real workloads are run on big endian systems.

References

[1] *GitHub - riscv/riscv-isa-manual: RISC-V Instruction Set Manual — github.com.* `https://github.com/riscv/riscv-isa-manual/`. [Accessed 04-02-2025].

[2] Five Embeddev. *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture — five-embeddev.com.* `https://five-embeddev.com/riscv-priv-isa-manual/Priv-v1.12/riscv-privileged.html`. [Accessed 04-02-2025].

[3] *Arm Developer: developer.arm.com.* `https://developer.arm.com/documentation/ddi0406/b/Application-Level-Architecture/Application-Level-Memory-Model/Endian-support/Control-of-the-endianness-mapping-scheme-in-ARMv7?lang=en`. [Accessed 05-02-2025].

[4] *What is Mixed-endian? - Definition from Amazing Algorithms — amazingalgorithms.com.* `https://amazingalgorithms.com/definitions/mixed-endian/`. [Accessed 05-02-2025].