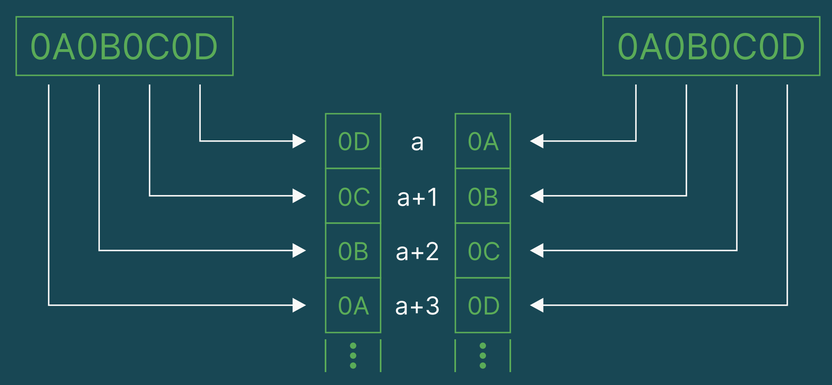
Implementing Runtime-Configurable Endianness in RISC-V

Why big-endian

While little-endian is almost ubiquitous nowadays, big-endian still exists as a legacy in many network protocols developed in the early days of the internet.

To communicate using these big-endian protocols, little-endian based computers must reverse the bytes into the big-endian format; this operation is not cheap and, if performed frequently, will produce significant overhead on non-optimised platforms.

Switching endianness on a modern system would allow normal operations to be performed in little-endian and then, when performing a network-intensive workload, switched to big-endian, removing the need for byte swapping during operation.



RISC-V Implementation

RISC-V defines a set of Control and Status Registers (CSRs). These registers contain information about the system, such as bits for what extensions are available.

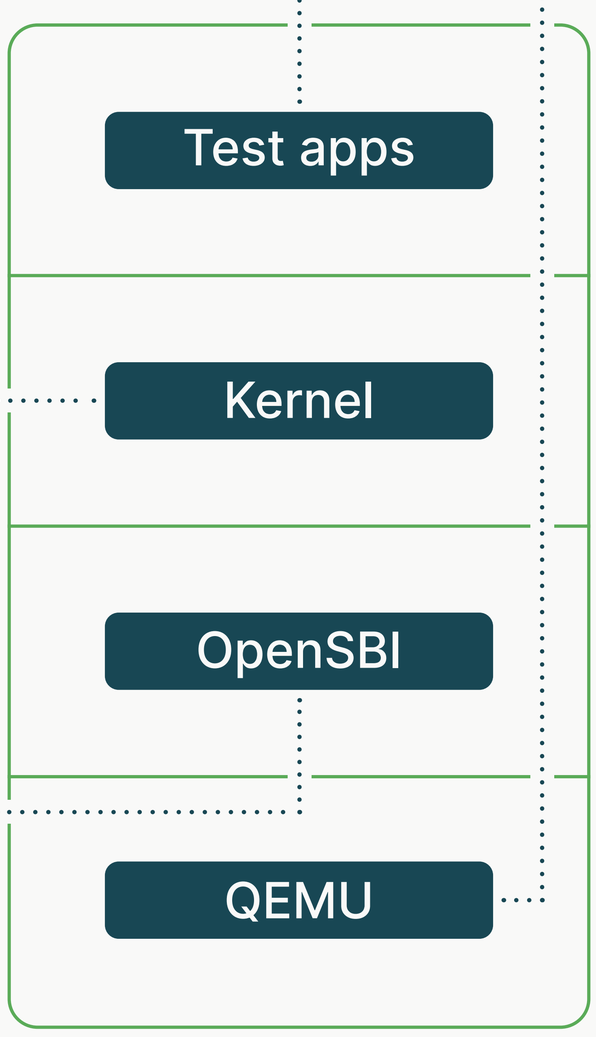
By setting a specific bit in one of these registers, the hardware signals that it operates in either big or little-endian mode.

Software Changes

Interacting with instructions must correctly use little endian data. This means changing self-modifying and self-introspecting code. Fixes in the kenrel include bpf-jit, exception handling, and runtime probes. There are also places where instructions are hand assembled for when the toolchain does not support the instruction.

The Software Stack

OpenSBI is an open implementation of the RISC-V Supervisor Binary Interface. It acts as a bridge between platform-specific firmware + bootloader and a hypervisor or a general-purpose OS.



Emulating for Hardware (QEMU)

Required the addition of the `{M,VS,S,U}BE {m,h,s,u}status` register flags, handling of the endianness switch, and modifications to load and store operations. Additionally, the atomic load/store operations added in the Zacaz extension required similar modifications.

There are CSR registers for each privilege level, `{M,H,S,U}status`, were already in QEMU, the team had to add an extra flag, `{M,VS,S,U}BE`, to allow the bit which signifies the endianness to be changed.

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.

The page-table code was modified to follow the same endianness as the `S` or `HS` modes in the specification.

Ongoing Issues

There will be further issues found, as shown with instruction endian-ness, MMIO accesses as software gets further tested. Having RISC-V sillicon will drive more work. The team will keep working on this so please stay tuned!

For more information, project source, patch submissions, and logs can be found on our GitLab. Scan the QR code to access.

gitlab.com/CodethinkLabs/riscv_bigendian

connect@codethink.co.uk