Date: 2020/8/11
Task Group: Fast Interrupts
Chair: Krste Asanovic
Co-Chair: Kevin Chen
Number of Attendees: ~10
Current issues on github:
https://github.com/riscv/riscv-fast-interrupt/blob/master/clic.adoc

In response to the recent mail thread on hardware stacking, we spent the entire meeting discussing how to provide support for a hardware stacking scheme.

Without clearance of liability using concrete expired implementations or documents, we cannot safely support all features available in commercial processors for now. Therefore, it is more feasible to break down the hardware features by levels and try to support them incrementally.

First, Krste defined some terms: Regular C ABI for ISRs (CAISR), versus “interrupt attribute” ISRs (IAISR)

From the simplest to the most complicated, hardware support can be roughly classified as the following levels:

- Software base only, with software trampoline, software support for stack switching, getting next interrupt

- Hardware vector table
    - For CAISR, leads to code size growth due to repeated software save/restore in each ISR
    - Even for IAISR can have high code size
    - Can put jump in table to common routine to save code, but adds to service time

- Selective hardware vectoring
    - Use common base plus trampoline for most routines to reduce code size
    - For few critical interrupts, assign customized save/restore code to reduce latency
    - Hardware cost of one flop per interrupt input, may be shared with mode/level/priority function

- GPR save/restore instructions (push/pop multiple) - could add options for mepc/mcause
    - Reduce power and code size in ISRs
    - Maybe faster (for cases with one single bus shared by both instruction and data)
    - Need to work with the new Code Size Reduction Task Group to define these new instructions

- Stack switching in hardware (swap sp with some CSR, e.g., mscratch or some new sp)
    - Reduce power and code size

- Hardware register save/restore including mcause/mepc etc.
    - Reduce power and code size in ISRs
    - Slightly faster because hardware can automatically start saving registers without waiting for the fetching of “save” instructions. Also, it can overlap the latency of saving register with the latency of fetching the first instructions from ISRs.

- Fully hardware-optimized fixed interrupt scheme
    - Further include automatic hardware features such as optimizing the selection and execution of multiple concurrent interrupts during prologue and epilogue phases

Currently, we have only defined the first 3 features in our spec. If we want to include more hardware features in our first ratification (hopefully at the end of this year), we need help from all members to provide useful old prior arts/documents in order to quickly come up with proposals.