The RISC-V Instruction Set Manual

Volume I: User-Level ISA

Document Version 2.2

Memory Consistency Model Addendum

Daniel Lustig
NVIDIA
`dlustig@nvidia.com`

May 12, 2017

The RISC-V Foundation is still working to choose and formalize a memory consistency model that meets the needs of all RISC-V implementers. In the meantime, until the memory model is settled, we recommend that implementers of processors be conservative in how they implement the memory model, and that system software writers be conservative in how they use it. Both parties can expect compatibility with the memory consistency model, provided they adhere to the following strictures:

- Hardware implementers should err on the side of caution by assuming that RISC-V may adopt a memory model as strong as Total Store Ordering (TSO). In particular:
  - Architects should pay careful attention to agressive memory access reordering, aggressive cache cache coherence protocols, and designs that share store buffers between threads.
  - Hardware should respect all same-address orderings (including load-load pairs) and any orderings established by address, control, and data dependencies.
- Assembly programmers should err on the side of caution and assume that RISC-V may adopt a weakly ordered memory model. We recommend using a full `fence` instruction where the corresponding code on other weakly ordered architectures employs any fence.
- Compiler writers should for now continue to use the intuitive mappings from language-level memory ordering to RISC-V operations. In particular,

C/C++ Construct	Base ISA Mapping	'A' Extension Mapping
Loads		
Non-atomic Load	`ld`	
`atomic_load(memory_order_relaxed)`	`ld`	
`atomic_load(memory_order_consume)`	`ld; fence r,rw`	
`atomic_load(memory_order_acquire)`	`ld; fence r,rw`	
`atomic_load(memory_order_seq_cst)`	`fence rw,rw; ld; fence r,rw`	
Stores		
Non-atomic Store	`sd`	
`atomic_store(memory_order_relaxed)`	`sd`	
`atomic_store(memory_order_release)`	`fence rw,w; sd`	`amoswap.rl`
`atomic_store(memory_order_seq_cst)`	`fence rw,rw; sd`	`fence rw,rw; amoswap`
Fences		
`atomic_thread_fence(memory_order_acquire)`	`fence r,rw`	
`atomic_thread_fence(memory_order_release)`	`fence rw,w`	
`atomic_thread_fence(memory_order_acq_rel)`	`fence rw,rw`	
`atomic_thread_fence(memory_order_seq_cst)`	`fence rw,rw`	

Furthermore, we recommend compiler writers avoid fences weaker than `fence r,rw`, `fence rw, w`, and `fence rw, rw` until the memory model clarifies their semantics. Additionally, while AMOs with both the `aq` and `rl` bits set do imply both `aq` and `rl` semantics, we recommend against their use until the memory model clarifies their combined semantics.

Undoubtedly, our recommendations either to hardware implementers or to software writers will prove to be overly conservative; possibly both. Once the Foundation has decided upon a memory consistency model, the conservative implementations can easily be weakened to improve performance.

Any questions or comments about the status of the memory consistency model or the above recommendations should be directed to the RISC-V Foundation's Memory Consistency Model Task Group.