Download Worksheet 11

Please mute yourself when not asking questions

CS 152: Discussion Section 11

Memory Consistency

Albert Ou, Yue Dai
04/24/2020

Administrivia

- PS5 due on Fri, May 1
- Lab 5 due Mon, May 4

Agenda

- Memory consistency models
  - Much of this material is taken from: Adve and Gharachorloo. Shared Memory Consistency Models: A Tutorial (1995)

Memory Consistency: Other half of the ISA

- The instruction set and architectural state do not completely define the behavior of the ISA
- Sequential ISA only specifies that each processor observes its own memory operations in program order
- Need a *memory consistency model*, which defines the legal set of values that loads can return across multiple hardware threads

Memory Consistency vs Cache Coherence

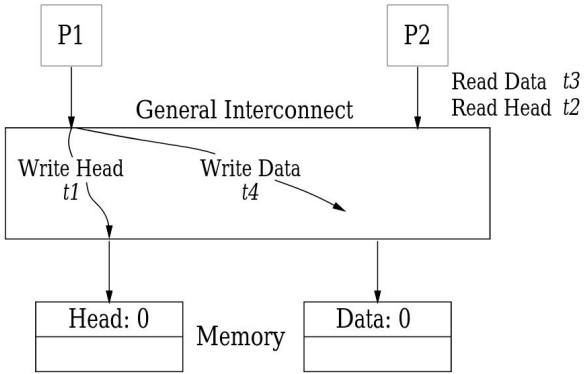
- *Coherence* describes the legal values that a *single* memory address should return
- *Consistency* describes properties across all memory address

Coherence alone does not imply any particular memory consistency model!

Memory Consistency and Caches

Q: Do memory consistency models apply only to systems with caches?

A: No — consider a cache-less system that allows concurrent write operations



P1	P2
Data = 2000	while (Head == 0) {;}
Head = 1	... = Data

RISC-V Memory Models

RISC-V currently has two:

1. Default: Weak memory ordering (RVWMO)
2. Optional extension: Total store order (RVTSO)

Sequential Consistency: Intuitive Starting Point

Definition: [A multiprocessor system is sequentially consistent if] the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.

Leslie Lamport. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs (1979)

Try worksheet Q1.1

Relaxing Memory Consistency Models

Strictness of SC suggests two major targets for relaxation:

1. *Program order requirement*: When can one thread's memory operations be reordered with respect to one another
2. *Write atomicity requirement*: When can a processor see the effect of a write relative to other processors in the system

Relaxing Program Order Requirement

Four types of ordering constraints:

1. Write → Read
2. Write → Write
3. Read → Write
4. Read → Read

Q: Can you think of a microarchitectural optimization made possible by relaxed ordering?

Relaxing Write Atomicity

Two flavors:

1. Can a processor see its own write before others?
2. Can processor A see some other processor B's writes before they are made visible to the rest of the memory system?

Q: Can you think of a microarchitectural optimization that would make (1) true? What about (2)?

Relaxation	W → R Order	W → W Order	R → RW Order	Read Others' Write Early	Read Own Write Early	Safety net
SC [16]					√	
IBM 370 [14]	√					serialization instructions
TSO [20]	√				√	RMW
PC [13, 12]	√			√	√	RMW
PSO [20]	√	√			√	RMW, STBAR
WO [5]	√	√	√		√	synchronization
RCsc [13, 12]	√	√	√		√	release, acquire, nsync, RMW
RCpc [13, 12]	√	√	√	√	√	release, acquire, nsync, RMW
Alpha [19]	√	√	√		√	MB, WMB
RMO [21]	√	√	√		√	various MEMBAR's
PowerPC [17, 4]	√	√	√	√	√	SYNC

Try worksheet Q1.2 - Q1.4

Using Fences to Constrain Memory Ordering

The RISC-V `FENCE` instruction comes in several fine-grained variants:

1. Write → Read `FENCE w,r`
2. Write → Write `FENCE w,w`
3. Read → Write `FENCE r,w`
4. Read → Read `FENCE r,r`

Can combine constraints: `FENCE r,rw == FENCE r,r; FENCE r,w`
`FENCE` without parameters is a full barrier: `FENCE rw,rw`

Try worksheet Q2