Memory Consistency and Cache Coherence

- Memory Consistency and Cache Coherence 定义
- TSO（Total Store Ordering）内存模型
- x86的多核宽松内存一致性模型

Memory Consistency and Cache Coherence ??

1. Memory Consistency???????

- [illegible]
  [illegible]
- [illegible] [illegible]
- [illegible]
  [illegible]
- [illegible] [illegible]
  - [illegible] **Sequential Consistency**[illegible]
    [illegible]
  - [illegible] **Weak Consistency**[illegible]
    [illegible]
- [illegible] [illegible]
  - [illegible]

---

2. Cache Coherence???????

- [illegible]
  [illegible]
- [illegible] [illegible]
- [illegible]
  [illegible]
- [illegible] [illegible]
  - **MESI**[illegible] [illegible] Modified[illegible] Exclusive[illegible] Shared[illegible] Invalid
    [illegible]
  - [illegible] **Write Invalidate**[illegible] [illegible]
  - [illegible] **Write Update**[illegible] [illegible]
- [illegible] [illegible]
  - [illegible] **A**[illegible] **B**
    [illegible]

---

3. ????

特性	**Memory Consistency**（内存一致性）	**Cache Coherence**（缓存一致性）
关注点	内存操作的全局顺序和可见性	多个缓存副本之间的数据一致性
范围	涉及所有内存操作（读 /写）的顺序	仅涉及缓存和内存之间的数据一致性
问题背景	多个处理器对共享内存的访问顺序是否一致	多个处理器的缓存中数据是否一致
解决方案	内存一致性模型（如顺序一致性、弱一致性）	缓存一致性协议（如 MESI 、写失效、写更新）

---

4. ??

- **Memory Consistency** 关注的是多处理器系统中内存操作的全局顺序和可见性。
- **Cache Coherence** 关注的是多处理器系统中缓存副本之间的数据一致性。

TSO?Total Store Ordering?????

1. TSO（ Total Store Ordering）是一个被广泛使用的内存模型
2. 并在 x86架构中使用， RISC-V也提供了 TSO扩展，即 RVTSO，人们普遍认为 x86内存模型等同于 TSO，然而 Intel和 AMD从来没有保证这一点
3. x86选择放弃 SC（顺序一致性 sequential consistency），以更好地支持基于 FIFO的 write buffer，用于加速性能
4. TSO和 SC最关键的区别就是 store可能被放入 write buffer中且允许 load的 bypass
   1. 对于单核视角来说，和 SC没有区别，执行顺序和程序顺序一致
   2. 对于多核来说，需要额外的手段来保证 memory consistency
5. 由于 TSO只允许 store-load的重排，因此需要使用 FENCE的场合不多， FENCE 的实现方式也不是很重要。一种简单的实现是，当 FENCE被执行时，清空 write buffer，并在 FENCE 提交之前不再执行后面的 load指令。

x86????????????

1. 被修饰的汇编指令成为 “原子的 ”
   1. 本身是原子指令，比如 “XCHG”和 “XADD”汇编指令
   2. 本身不是原子指令，但是被 `LOCK`指令前缀修饰后成为原子指令，比如 LOCK CMPXCHG
   3. 被修饰的汇编指令 A在执行期间，会在内存总线上声言一个 `#LOCK` 信号，该信号导致内存被锁住，此时内存不能再被其他汇编指令存取，直到 A 执行完成。经过分析可知， A的执行效果与 “暂停执行其他所有汇编指令直到 A 执行完成等价，因此此时 A是原子的
2. fence
   1. sfence: 在 sfence指令前的写操作当必须在 sfence指令后的写操作前完成
   2. lfence: 在 lfence指令前的读操作当必须在 lfence指令后的读操作前完成
   3. mfence: 在 mfence指令前的读写操作当必须在 mfence指令后的读写操作前完成
3. C++11中支持的内存模型
   1. Acquire-Release能保证不同线程之间的 Synchronizes-With 关系，同时也约束到同一个线程中前后语句的执行顺序。
   2. Release-Consume只约束有明确的 carry-a-dependency 关系的语句的执行顺序，同一个线程中的其他语句的执行先后顺序并不受这个内存模型的影响

```
enum memory_order {
    memory_order_relaxed, // Relaxed
    memory_order_consume, // Release-Consume
    memory_order_acquire, // Acquire-Release 用来修饰一个读操作，表示在本线程中，所有后续的关于此变量的内存操作都必须在本条原子操作完成后执行
    memory_order_release, // Acquire-Release 用来修饰一个写操作，表示在本线程中，所有之前的针对该变量的内存操作完成后才能执行本条原子操作
    memory_order_acq_rel, // Acquire-Release 同时包含memory_order_acquire和memory_order_release标志
    memory_order_seq_cst  // 顺序一致性模型
};
```