acquire release ???????

??

1. 在单线程场景中，CPU 通常会保证程序顺序（**Program Order**）的可见性，即单线程内的指令会按照代码编写的顺序执行（或看起来像是按顺序执行）存储器读写的结果也会符合单线程的预期
    1. 即使 CPU有乱序功能，也会通过 scoreboard等方式来处理 data hazard，address hazard 等，确保单线程内的内存访问都是保续的。即使现代的 CPU都是超标量处理器。
2. 但在 多线程或多处理器（**multi-hart**）场景中，要实现多线程同时正确的对一个内存操作就会遇到问题
    1. 乱序执行（Out-of-Order Execution）
    2. 缓存分层（Cache Hierarchy），
    3. 存储缓冲区（Store Buffer）**，**CPU有各级的存储器 cache，in-flight指令暂存器
    4. 其他线程观察到的实际操作顺序与当前线程的程序顺序不一定一致
    5. 希望实现一个功能，利用一片多核都可以访问到的存储器来实现同步，这个存储器实现 **“锁”**功能
        1. 处理器 1，通过读 “锁”，确定没人用，写 “锁”，确保占用，不被别人使用
        2. 在处理器，读 -写中间，处理器 2可能就会写 “锁”，从而造成错误
        3. 处理器 1，准备通过写 “锁”，来释放占用。但是写操作可能被乱序到当前线程的前面去执行，从而达不到保护的目的。
    6. 所以希望实现 “原子指令”，实现获取 “锁”，和释放 “锁”的中间不会被其他的线程干扰，产生错误。
3. 在多线程间建立 **“同步关系”**，通过限制 CPU 的重排序和缓存行为，确保
    1. 使用 `acquire` 和 `releasey` 语义，确保单线程内，和 LR.W 和 SC.W 指令之间是保序的。单线程内的指令实际执行顺序能被 LR.W 和 SC.W 控制，不收缓存和乱序的影响。
    2. 然后通过 LR.W 和 SC.W 指令，通过特殊的多线程都唯一的硬件，实现多线程之间的竞争。
4. 从而确保
    1. 一个线程的写操作结果能被另一个线程的读操作正确感知
    2. 跨线程的操作顺序符合预期（避免 “先写后读”变成 “先读后写”的错乱）

`LR.W`(Load-Reserved)??

??????

??????

1. 加载值：从内存地址 `rs1` 读取 32位数据到寄存器 `rd`。
2. 设置保留标记：
    - 在缓存子系统（如 L1 Cache）中标记该地址为 **Reserved**（保留状态）。
    - 硬件会记录当前核心（Hart）对该地址的独占访问权。

3. 隐式 **acquire**语义（若未显式指定）：
   - 后续内存操作不会被重排序到 `LR.W` 之前（保证加载操作的可见性）。

?????????

- 缓存行状态：目标地址对应的缓存行会被置为 **Exclusive** 或 **Modified** 状态（取决于一致性协议，如 MESI）。
- 保留标记的存储：
  - 通常由缓存控制器维护一个 **Reservation Register**（保留寄存器），记录保留地址和核心 ID。
  - 其他核心的写操作会清除该地址的保留标记（通过缓存一致性协议广播 Invalidate）。

?? `SC.W` (Store-Conditional)??

???????

???????

1. 检查保留标记：
   - 若地址 `rs1` 的保留标记仍被当前核心持有（未被其他核心修改）：
     - 执行存储：将 `rs2` 的值写入 `rs1`。
     - 清除保留标记。
     - 向 `rd` 写入 `0`（成功）。
   - 若保留标记已失效（其他核心修改了地址 `rs1`）：
     - 放弃存储。
     - 向 `rd` 写入 `非0值`（通常为 1，失败）。
2. 隐式 **release**语义（若未显式指定）：
   - 之前的内存操作不会被重排序到 `SC.W` 之后（保证存储操作的全局可见性）。

?????????

- 原子性保证：
  - 通过缓存一致性协议（如 MESI）锁定缓存行，确保“检查-存储”的原子性。
  - 若其他核心发起写请求，缓存控制器会清除保留标记并响应 Invalidate。
- 总线事务：
  - 成功时生成 **Atomic Write** 总线事务。
  - 失败时不发起存储请求。

???????????

```
# 加锁（使用aq/rl语义确保屏障）
lock:
  lr.w.aq t0, (a0)        # 带acquire的加载保留
  bnez    t0, lock        # 检查锁是否被占用（t0!=0则重试）
  li      t1, 1
  sc.w.rl t0, t1, (a0)    # 带release的条件存储
```

```
  bnez    t0, lock        # 若存储失败（t0!=0），重试


# 解锁
unlock:
  sw.rl   zero, (a0)      # 带release的存储，释放锁
```

?????????

```
Core 0                          Core 1
======                          ======
1. lr.w t0, (x)                 3. lr.w t0, (x)
   - 加载x=0，设置保留标记          - 加载x=0，设置保留标记
2. sc.w t1, 1, (x)              4. sc.w t1, 1, (x)
   - 保留有效，存储成功（x=1）      - 保留已失效（因Core 0修改x），存储失败
   - t1=0                           - t1=1
```

Revision #6
Created 2025-10-14 11:24:56 UTC by Colin
Updated 2026-04-29 07:34:51 UTC by Colin