The MIPS R4000, part 7: Memory access (atomic)

devblogs.microsoft.com/oldnewthing/20180410-00

April 10, 2018

Raymond Chen

Atomic memory access on the MIPS R4000 is performed with the load-linked and store-conditional instructions. This pattern shouldn't be much of a surprise because we already encountered it on the Alpha AXP.

```
    LL      rd, disp16(rs)  ; load linked
    SC      rs, disp16(rd)  ; store conditional
```

The `LL` instruction loads a value from memory and monitors the memory address to see if another processor writes to it. The `SC` instruction stores the value to memory, provided there have been no writes[1] to the monitored memory address[2] and no exceptions have occurred.[3] If the store succeeds, then *rs* is set to one; otherwise it is set to zero.

In both cases, the memory address must be word-aligned.

The intended usage pattern is

```
retry:
    LL      r1, disp16(r2)  ; load linked
    ADDIU   r1, r1, 1       ; increment
    SC      r1, disp16(r2)  ; store conditional
    BEQ     r1, 0, retry    ; if failed, then retry
    NOP                     ; (we'll learn about this later)
```

The state created by the `LL` is ephemeral, and the subsequent `SC` is permitted (but not required) to fail if any of the following occur prior to the `SC`:

- A memory access is performed.
- A branch is taken.
- More than 512 instructions are executed.

Furthermore, after the `SC` (either successful or unsuccessful), all subsequent `SC` instructions are required to fail until a new `LL` is executed.

If the `LL` from an address is followed by `SC` which does not write to the same address, then it is unspecified whether the `SC` succeeds. So don't do that.

It is legal to execute the `LL` instruction and not follow it with the `SC` instruction. This can happen if you want to perform a conditional atomic operation, and you discover that the condition is not met.

Before and after the `LL` / `SC` operation, you probably want to do a

```
SYNC            ; memory barrier
```

All memory operations that precede the `SYNC` must complete before any operations that follow the `SYNC` can begin.

Note that atomic operations are supported only on aligned words. For aligned sub-word objects, you can perform the atomic operation on the containing word. But if the object is not aligned, then you're out of luck.

Next time, we enter the exciting world of control transfer. That's where the `NOP` above gets its moment to shine.

[1] Note that if another processor writes the value that is already there back to the memory, or if there is an ABA condition where another processor changes the value, and then changes it back, then the conditional store will fail, even though the value in memory is the same value you started with. This is one cause for the mysterious case of the `compare_exchange_weak` spurious failure.

[2] The architecture permits implementations to be sloppy with the detection of a write. In particular, any modification on the same 4KB page as the locked address is permitted to cause the subsequent store conditional instruction to fail. Mind you, an implementation that was this sloppy would not be a very good implementation, but it is technically legal.

[3] This last clause is actually an operating system convention, not something inherent in the processor architecture. One of the things that kernel mode does before returning to user mode is execute the `SC` instruction with a scratch writable memory location. The `SC` might succeed, it might not, but it doesn't matter. The reason for the `SC` is to ensure that if the next atomic memory operation performed by user-mode code is `SC`, then that operation *definitely* fails. This is important in the case where the interrupt occurred after the user-mode code performed the `LL` but before it could execute the subsequent `SC`. Without it, the `SC` in user mode might succeed accidentally.

Raymond Chen

Follow