The AArch64 processor (aka arm64), part 13: Atomic access

devblogs.microsoft.com/oldnewthing/20220811-00

August 11, 2022

Raymond Chen

Atomic operations are performed by the traditional RISC-style *load locked/store conditional* pattern.

```
; load exclusive register byte
ldxrb   Rd/zr, [Xn/sp]

; load exclusive register halfword
ldxrh   Rd/zr, [Xn/sp]

; load exclusive register
ldxr    Rd/zr, [Xn/sp]

; load exclusive register pair
ldxp    Rd1/zr, Rd2/zr, [Xn/sp]
```

These instructions atomically load a byte, halfword, word, doubleword, or pair of registers from memory. The instruction also tells the processor to monitor the memory address to see if any other processor writes to that same address, or addresses in the same "exclusive reservation granule". (Implementations are allowed to have granules as large as 2KB.)

Note that the atomicity guarantee is only partial if you use `LDXP` to load a pair of 64-bit registers.[1] The entire 128-bit value is not loaded atomically; instead, each 64-bit portion is loaded atomically separately. You can still get tearing between the two registers.

The only supported addressing mode is register indirect. No offsets or indexes allowed.

After an exclusive load, you can attempt to store a value back to the same address:

```
; store exclusive register byte
stxrb   Rs/zr, Rt/zr, [Xn/sp]

; store exclusive register halfword
stxrh   Rs/zr, Rt/zr, [Xn/sp]

; store exclusive register
stxr    Rs/zr, Rt/zr, [Xn/sp]

; store exclusive register pair
stxp    Rs/zr, Rt1/zr, Rt2/zr, [Xn/sp]
```

If the reservation obtained by the previous `LDX` instruction is still valid, then the value in *Rt/zr* is stored to memory, and *Rs* is set to 0. Otherwise, no store is performed, and *Rs* is set to 1.

Whether the store succeeds or fails, the `STX` instructions clears the reservation.

For these exclusive load and store instructions, the address must be a multiple of the number of bytes being loaded. If not, then the behavior is undefined: There is no requirement that an exception be raised.

So don't do that.

It is also required that the `STX` match the `LDX` both in address and operand sizes. You cannot perform an `LDX` for one address and follow up with a `STX` to a different address. You also cannot perform a `LDXR` and follow up with a `STXRH` to the same address. You aren't even allowed to do a `LDXP` with two 32-bit registers and follow up with a `STXR` with a single 64-bit register. Again, the behavior is undefined if you break this rule.

The last instruction allows you to hit the reset button:

```
; clear exclusive
clrex
```

The `CLREX` discards any active reservation, and forces any subsequent `STX` to fail. This typically happens as part of interrupt handling or context switching to ensure that undefined behavior doesn't occur if the thread was interrupted while it was in the middle of a `LDX` / `STX` sequence.

These instructions are usually coupled with memory barriers, which we'll look at soon, but the next entry will be a little diversion.

Bonus chatter: There is an optional instruction set extension (mandatory starting in version 8.4) which includes a large set of atomic read-modify-write operations.

```
; atomic read-modify-write operation
; Rt = previous value of [Xr]
; [Xr] = Rt op Rs
ldadd   Rs/zr, Rt/zr, [Xr/sp]       ; add
ldclr   Rs/zr, Rt/zr, [Xr/sp]       ; and not
ldeor   Rs/zr, Rt/zr, [Xr/sp]       ; exclusive or
ldset   Rs/zr, Rt/zr, [Xr/sp]       ; or
ldsmax  Rs/zr, Rt/zr, [Xr/sp]       ; signed maximum
ldsmin  Rs/zr, Rt/zr, [Xr/sp]       ; signed minimum
ldumax  Rs/zr, Rt/zr, [Xr/sp]       ; unsigned maximum
ldumin  Rs/zr, Rt/zr, [Xr/sp]       ; unsigned minimum
```

By default, there is no memory ordering. You can add the suffix `a` to load with acquire, the suffix `l` to store with release, or the suffix `al` to get both. Note, however, that the acquire suffix is ignored if the destination register *Rt* is *zr*.

Furthermore, you can suffix `b` for byte memory access or `h` for halfword memory access.

The overall syntax is therefore

Prefix	Op	Acquire	Release	Size

`ld`	`add`
	`clr`
	`eor`
	`set`
	`smax`
	`smin`
	`umax`
	`umin`
		(none)
		`a`
			(none)
			`l`
				(none)
				`b`
				`h`

For example, the instruction `ldclrlh` means

- `ld` : Atomic load/modify/store
- `clr` : Clear bits
- (blank): No acquire on load
- `l` : Release on store
- `h` : Halfword size.

If you don't care about the previous value, then you can use a pseudo-instruction that uses *zr* as the destination.

```
; atomic read-modify-write operation
; [Xr] = [Xr] op Rs
stadd   Rs/zr, [Xr/sp]       ; add
stclr   Rs/zr, [Xr/sp]       ; and not
steor   Rs/zr, [Xr/sp]       ; exclusive or
stset   Rs/zr, [Xr/sp]       ; or
stsmax  Rs/zr, [Xr/sp]       ; signed maximum
stsmin  Rs/zr, [Xr/sp]       ; signed minimum
stumax  Rs/zr, [Xr/sp]       ; unsigned maximum
stumin  Rs/zr, [Xr/sp]       ; unsigned minimum
```

You can add the `l` suffix for store with release, and you can add `b` and `h` suffixes to operate on smaller sizes. You cannot request acquire on load for these instructions because the acquire is ignored due to the destination being *zr*.

The optional instruction set extension also provides for atomic exchanges:

```
; swap
; write Rs and return previous value in Rt (atomic)
swp     Rs/zr, Rt/zr, [Xn/sp]       ; word or doubleword
swpb    Ws/zr, Wt/zr, [Xn/sp]       ; byte
swph    Ws/zr, Wt/zr, [Xn/sp]       ; halfword

; compare and swap
; if value is Rs, then write Rt; Rs receives previous value
; (atomic)
cas     Rs/zr, Rt/zr, [Xn/sp]       ; word or doubleword
casb    Ws/zr, Wt/zr, [Xn/sp]       ; byte
cash    Ws/zr, Wt/zr, [Xn/sp]       ; halfword
casp    Rs/zr, Rt/zr, [Xn/sp]       ; register pair
                                    ; Rs,R(s+1) and Rt,R(t+1)

; also a, l, and al versions for acquire/release semantics
```

The memory order modifiers go between the `swp` / `cas` prefix and the size suffix, *except* that they go after the `p`. So you have `casab` (compare and swap with acquire, byte size) but `caspa` (compare and swap pair with acquire).

As with the `ld` instructions, requests to aquire on load are ignored if the destination register is *zr*.

The memory operand must be writable, even if the comparison fails. If no value is stored, then any requested release semantics are ignored.

Bonus reading: Atomics in AArch64.

[1] The load is required to be fully atomic starting with version 8.4 of the AArch64. On older processors, Windows uses `CASP` instead of `LDXP` / `STXP`.

Raymond Chen

Follow