The Alpha AXP, part 10: Atomic updates to byte and word memory units

devblogs.microsoft.com/oldnewthing/20170818-00

August 18, 2017

Raymond Chen

Today we're going to do a little exercise based on what we've learned so far. We learned how to perform byte and word loads and stores to memory. And we also learned how to perform atomic memory operations on longs and quads. But how about atomic memory operations on bytes and words?

We will have to put together what we've learned: Combine the byte and word access patterns with the atomic memory update pattern.

To recap: The sequence for reading an aligned word in memory goes like this:

```
    LDQ_U  t1, (t0)
    EXTWL  t1, t0, t1
```

The sequence for writing an aligned word in memory goes like this:

```
    LDQ_U   t5, (t0)                ; t5 = yyBA xxxx
    INSWL   t1, t0, t3              ; t3 = 00ba 0000
    MSKWL   t5, t0, t5              ; t5 = yy00 xxxx
    BIS     t5, t3, t5              ; t5 = yyba xxxx
    STQ_U   t5, (t0)

    ; Byte sequence is the same, except you use INSBL and MSKBL
```

And the sequence for an atomic quad update goes like this:

```
retry:
    LDQ_L   t1, (t0)        ; load locked
    ... calculate new value of t1 based on old value ...
    STQ_C   t1, (t0)        ; store conditional
                            ; t1 = 1 if store was successful
    BEQ     t1, failed      ; jump if store failed
    ... continue execution ...

failed:
    BR      zero, retry     ; try again
```

What we need to do is insert the byte or word extraction, calculation, and insertion code where it says "calculate new value of *t1* based on old value". The trick is that there is no `LDQ_LU` instruction. You can read for unaligned or you can read locked, but you can't read for unaligned locked.

Fortunately, this is easy to work around: We emulate the behavior of `LDQ_U` in software. Recall that `LDQ_U` is the same as `LDQ` except that it ignores the bottom 3 bits of the address. So let's mask out the bottom 3 bits of the address.

```
    ; atomically increment the word at the aligned address t0
    BIC     t3, #3, t0      ; force-align t0 to t3
retry:
    LDQ_L   t1, (t3)        ; load locked
    ... calculate new value of t1 based on old value ...
    STQ_C   t1, (t3)        ; store conditional
                            ; t1 = 1 if store was successful
    BEQ     t1, failed      ; jump if store failed
    ... continue execution ...

failed:
    BR      zero, retry     ; try again
```

Okay, we've successfully emulated the `LDQ_LU` and `STQ_LU` instructions. Now to do the extraction, calculation, and insertion:

```
    ; atomically increment the word at the aligned address t0
    BIC     t3, #3, t0      ; force-align t0 to t3
retry:
    LDQ_L   t1, (t3)        ; load locked
                            ; t1 = yyBA xxxx

    ; Extract
    EXTWL   t1, t3, t2      ; t2 = 0000 00BA (the word value)

    ; Calculate
    ADDL    t2, #1, t2      ; increment t2

    ; Insert
    INSWL   t2, t0, t2      ; t2 = 00ba 0000
    MSKWL   t1, t0, t1      ; t1 = yy00 xxxx
    BIS     t1, t2, t1      ; t1 = yyba xxxx

    STQ_C   t1, (t3)        ; store conditional
                            ; t1 = 1 if store was successful
    BEQ     t1, failed      ; jump if store failed
    ... continue execution ...

failed:
    BR      zero, retry     ; try again
```

Fortunately, our extraction, calculation, and insertion could be performed in under 20 instructions with no additional memory access, and no use of potentially-emulated instructions, so it all fits between the `LDQ_L` and `STQ_C`.

Exercise: What could we do if our calculation required additional memory access or required more than 20 instructions?

Raymond Chen

Follow