The Intel 80386, part 10: Atomic operations and memory alignment

devblogs.microsoft.com/oldnewthing/20190131-00

February 1, 2019

Raymond Chen

Memory access on the 80386 to misaligned locations are supported, although they will operate more slowly than their aligned counterparts. If a memory access straddles a page boundary, an access violation will be raised if either page does not support the desired operation (not readable or not writable), and the instruction will not appear to have started. The instruction does not partially-execute before the exception is raised.

If you are unlucky, and your misalignment straddles a page boundary, you can incur multiple page faults until pages on both sides of the boundary are simultaneously ready to accept the operation. If you are super-unlucky, this state may never be achieved and your program will just keep page faulting on that same instruction over and over again until the user terminates it.

Although 80386 does not support symmetric multiprocessor operations, it does support coprocessor operations as well as direct memory access (DMA), so you still need to be aware of atomicity.

Storing values to memory and reading values from memory are atomic operations.[1] If a competing processor writes to or reads the same memory, the result will be completely one value or the other, never a mix of the two.

This atomicity does not extend by default to read-modify-write operations, however.

```
    INC     [value]     ; may conflict with other processors
```

It's possible that another processor could write to the memory between the read and the write of the `INC` instruction.

To prevent another processor from accessing the memory during a read-modify-write memory operation, insert a `LOCK` prefix in front of the instruction. This causes the read-modify-write sequence to occur atomically.

```
    LOCK INC [value]    ; increment atomically
```

Any memory operation can be prefixed with a `LOCK`, and the processor will prevent any other processors from accessing the memory for the duration of that instruction. This works even for unaligned memory accesses!

The `LOCK` prefix is superfluous for simple reads and writes, since those are already atomic. It adds value only for read-modify-write instructions.

The `LOCK` prefix is also superfluous for the `XCHG` instruction, because the processor automatically locks the bus during an exchange. This automatic lock is for backward compatbility purposes, because `XCHG` was a common way to perform test-and-set operations on earlier versions of the processor.

Note that many atomic operations are not available in the form we have become accustomed to: Although you can perform an atomic increment or decrement, or atomic add or subtract, you don't receive the arithmetic result. The only atomic result from an arithmetic operation on memory is the flags. Therefore, the only information you got back from the `Interlocked-Increment` or `InterlockedDecrement` functions was the sign of the result. You could try to read the memory back to see what the result was, but that would be a separate instruction, outside the scope of the `LOCK`, and therefore is not part of the overall atomic operation.

The 80386 has no compare-exchange instruction, so there was no `InterlockedCompare-Exchange` available for the 80386. You did get a straight `InterlockedExchange`, though.

Okay, so that's atomic operations and memory alignment. Next time, we'll start looking at Windows software conventions.

[1] The operations are atomic, but not synchronized.

Raymond Chen

Follow