Memory Ordering Instructions in Intel x86_64 and ARMv8-A

Gordon Lichtstein
Massachusetts Institute of Technology

April 23, 2026

1 Overview

This document provides a compact, semi-formal summary and comparison of explicit memory ordering instructions in Intel's x86_64 and ARMv8-A+ architectures, relevant to inter-thread synchronization, timing, and IO. It builds on knowledge of the core x86 and ARM memory ordering models. I recommend this blog post for an overview, and the Intel / ARM manuals referenced in the sources section for a more detailed and formal treatment.

2 X86_64

Load instructions are considered globally visible once the value that will be loaded into the destination register is determined. This means that the memory access has been done, but the destination register isn't necessarily filled. I didn't find a good definition of globally visible stores, but it seems to mean that the store has been propagated throughout the store buffer and memory hierarchy. Locally completed

2.1 SFENCE

Waits for all previous stores to become globally visible before SFENCE completes, and before any store after SFENCE becomes globally visible. Ordered with respect to stores, SFENCE, MFENCE, and any serializing instructions. Not ordered with respect to loads or LFENCE.

2.2 LFENCE

Waits for all prior load instructions to complete locally and receive memory, and no later instruction executes until LFENCE is completed (not even speculatively, although they may be fetched and decoded). Blocking execution makes LFENCE useful for speculation hardening, but makes it slightly worse for measuring very short timescales (e.g. differentiating L1/L2 accesses), as it introduces longer and noisier latencies for instructions immediately following the LFENCE. LFENCE's pre-condition seems slightly stronger than global visibility, as it guarantees that prior load instructions have been completed, not just that we know the value that will be loaded into the destination register.

2.3 MFENCE

Waits for all prior loads and stores to be globally visible before executing and before all following loads/stores. Ordered with respect to all loads and stores, MFENCE, LFENCE, SFENCE, and serializing instructions, although it does not serialize the whole instruction stream.

2.4 RDTSCP (Not a memory ordering inst, but useful for timing)

Waits for all prior instructions to have been executed and for all prior loads to be globally visible. RDTSC is similar, but does not provide any such guarantee. According to the Intel Software Developer Manual, if

you want RDTSCP to only be run after previous stores are completed, add an MFENCE before it, and if you want RDTSCP to be executed before any subsequent instruction, add an LFENCE after it.

3 ARMv8-A

From the ARMv8-A architecture manual:

- A read/write RW1 is observed by a write W2 from iff W2 is coherence-after RW1.
- A write W1 is observed by a read R2 iff R2 reads from W1.
- A read R2 reads from a write W1 to the same location iff R2 takes its data from W1.

3.1 Data Memory Barrier (DMB)

All memory accesses and cache maintenance operations of the relevant type are ordered before and after the DMB operation. This means that all prior relevant memory accesses are observed by all relevant memory accesses after the DMB operation. It does not guarantee the completion of any memory accesses.

3.2 Data Synchronization Barrier / Data Write Barrier (DSB)

All prior memory accesses (of the relevant type), cache maintenance, and TLB operations are completed before the DSB is completed. No instructions after the DSB may execute except for being fetched and decoded, and reading registers that don't cause side effects (very limited).

3.3 Relevant Type (for both DSB and DMB)

- **SY / OSH / ISH:** Waits for prior stores and loads, and constrains loads and stores after the instruction. This is the default.
- **ST / OSHST / ISHST:** Waits for prior stores, and constrains stores after the instruction.
- **LD / OSHLD / ISHLD:** Waits for prior loads, and constrains loads and stores after the instruction. Only available on ARMv8+.

3.4 Sharability Domain (for both DSB and DMB)

- **Full System:** SY, ST, and LD referring to all memory devices. Slowest.
- **Outer Sharability Domain:** OSH, OSHST, and OSHLD wait for and constrain instructions in a set of inner sharability domains. Used for non-coherent DMA. Faster than full system.
- **Inner Sharability Domain:** ISH, ISHST, and ISHLD only wait for and constrain instructions in all devices coherent with the memory location. This is useful for DMA and MMIO, and is the fastest.

4 Comparison

	Restricts prior			Restricts future		
Instruction	Loads	Stores	Other	Loads	Stores	Other
MFENCE	Y	Y	N	Y	Y	N
LFENCE	Y (locally)	N	N	Y	Y	Y
SFENCE	N	Y	N	N	Y	N
DSB SY	Y	Y	N	Y	Y	Y
DSB ST	N	Y	N	Y	Y	Y
DSB LD	Y	N	N	Y	Y	Y
DMB SY	Y	Y	N	Y	Y	N
DMB ST	N	Y	N	N	Y	N
DMB LD	Y	N	N	Y	Y	N

5 Notes

- For x86_64, by default the restrictions mean that future instructions will become globally visible after the prior instructions become globally visible, not necessarily completed or executed.
- LFENCE provides a stronger guarantee - it waits for all loads to be locally completed and blocks all future instruction execution, but not fetch/decode.
- For DMB, the restriction means that prior instructions are observed by future instructions, not necessarily completed or executed.
- For DSB, the restriction means that prior memory operations are guaranteed to be completed before any future instruction executes.
- This table only holds for ARMv8-A. ARMv8-R relaxes some DMB restrictions, and previous ISA revisions describe a stricter DMB ST.

6 Rough ARM Analogues for x86 Operations

- MFENCE is most similar to DMB SY to enforce memory ordering, although DSB SY may be safer for timing-critical operations or speculation hardening as it blocks execution.
- SFENCE is most similar to DMB ST to enforce memory ordering, although DSB ST may be safer for timing-critical operations or speculation hardening as it blocks execution.
- LFENCE is most similar to DSB LD, but this is only available on ARMv8+.

7 Sources

- Intel® 64 and IA-32 Architectures Software Developer's Manual
- Intel® 64 and IA-32 Architectures Optimization Reference Manual
- ARM® Architecture Reference Manual - ARMv8, for ARMv8-A architecture profile
- `https://developer.arm.com/documentation/ddi0406/b/Application-Level-Architecture/Application-Level-Memory-Model/Memory-access-order/Memory-barriers`
- `https://www.felixcloutier.com/x86/RDTSCP`
- `https://www.felixcloutier.com/x86/MFENCE`
- `https://www.felixcloutier.com/x86/SFENCE`
- `https://www.felixcloutier.com/x86/LFENCE`