

# MICROARCHITECTURE CHEAT SHEET

## X86 CPUs & Performance

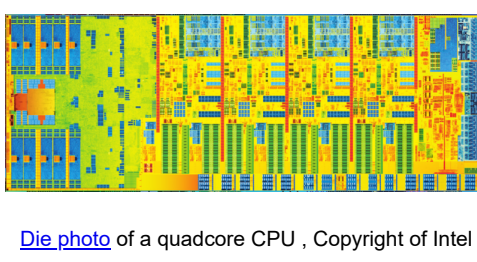


Photo of a quad-core CPU. Copyright of Intel

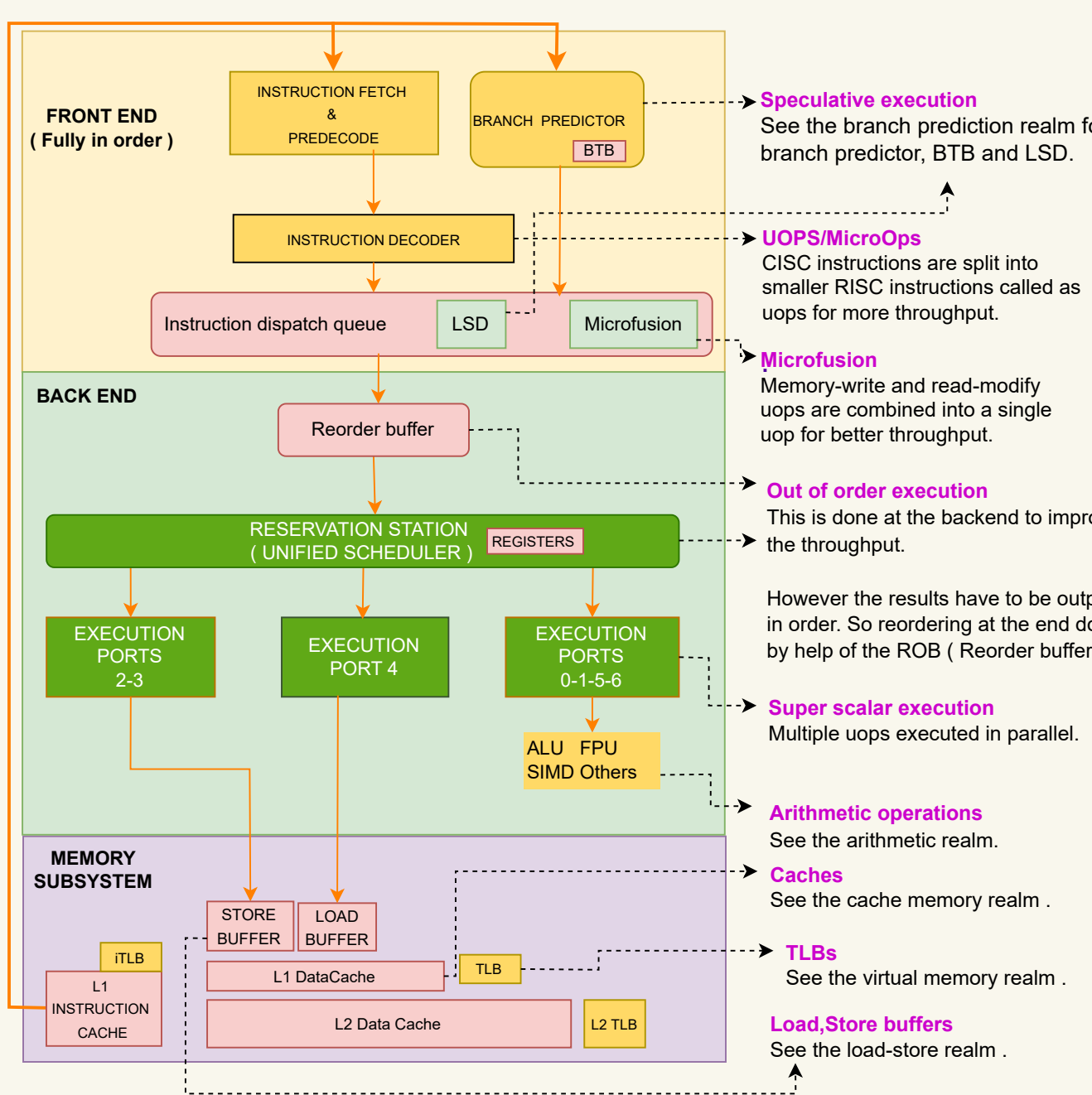
LAST UPDATE DATE : 20 OCT 2022

FOR LATEST VERSION: [www.github.com/akin/microarchitecture-cheatsheet](https://www.github.com/akin/microarchitecture-cheatsheet)

AUTHOR: AKIN OCAL [akin\\_ocal@hotmail.com](mailto:akin_ocal@hotmail.com)

## PIPELINE REALM : INSIDE AN INDIVIDUAL CORE

### A SIMPLIFIED OVERVIEW (BASED ON INTEL SKYLAKE)



Based on: [Skylake server on en.wikichip.org](https://en.wikichip.org/wiki/skylake_server)

### PIPELINE PARALLELISM & PERFORMANCE

**Pipeline diagrams** : The diagrams below in the following topics are outputs from an online microarchitecture analysis tool [UCA](https://en.wikichip.org/wiki/analysis_tool), and they represent parallel execution through cycles.

Rows are multiple instructions being executed at the same time.

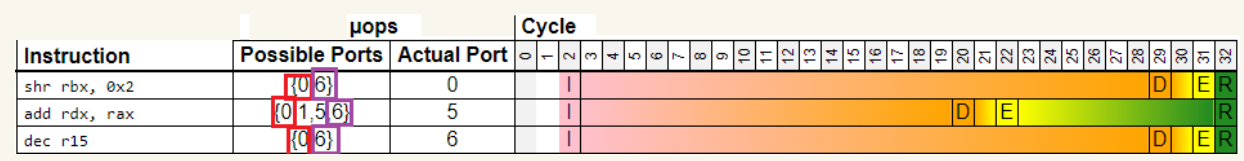
Columns display how instruction state changes through cycles.

**IPC** : As for pipeline performance, typically IPC is used. It stands for "Instructions per cycle". A higher IPC value usually means a better throughput.

You can measure IPC with perf: <https://perf.wiki.kernel.org/index.php/Tutorial>

**Rate of retired instructions** : Apart from IPC, number of retired instructions should be checked. Retired instructions are not committed/fused as they were wrongly speculated. On the other hand, executed instructions are the ones which were fused. Therefore a high rate of retired instructions indicates low branch prediction rate.

### CONTENTION FOR EXECUTION PORTS IN THE PIPELINE

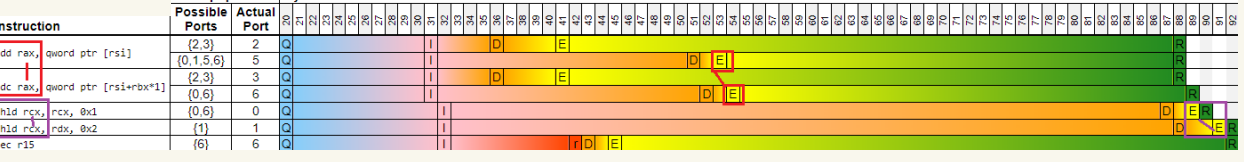


In the example above, all instructions are working on different registers, but SHR, ADD, DEC instructions are competing for ports 0 and 6. SHR and DEC are getting executed after ADD instruction.

Also notice that there is longer time between E[executed] and R[retired] states of instruction ADD as retirement has to be done in-order whereas execution is out-of-order.

Reference : [Denis Bashkatov's article](https://en.wikichip.org/wiki/skylake_server)

### INSTRUCTION STALLS DUE TO DATA DEPENDENCY



In the example above, there are 2 dependency chains, each marked with a different colour. In the first red coloured one, 2 instructions are competing for RAX register and notice that the second instruction gets executed after the first one.

Reference : [Denis Bashkatov's article](https://en.wikichip.org/wiki/skylake_server)

### RDTSCTP INSTRUCTION FOR MEASUREMENTS

RDTSCTP instruction can flush the pipeline to discard the instructions prior to the measurement and read the TSC value of the CPU.

TSC : timestamp counter

You can use CPUID and RDTSCTP combination in older systems that don't support RDTSCTP.

### ESTIMATING INSTRUCTION LATENCIES

Based on Agner Fog's [Instruction Latencies](https://www.agner-fog.com/instruction-latencies), RDTSCTP reciprocal throughput (clock cycle per instruction) is 32 on Skylake microarchitecture:

-> 1 cycle @4.5GHz = 0.22 nanoseconds  
-> 32\*0.22=7.04 nanoseconds

So its resolution estimate is about 7 nanoseconds on a 4.5 GHz Skylake microarchitecture. You have to recalculate it for different microarchitectures and clock speeds.

### HYPERTHREADING / SIMULTANEOUS MULTITHREADING

Based on [Intel Software Developer's Manual Volume3](https://en.wikichip.org/wiki/skylake_server), it is implemented by 2 virtual cores that share resources including cache memory, branch prediction resources and execution ports. And AMD seems to use the resources in the same way based on Agner Fog's [microarchitecture book](https://www.agner-fog.com/instruction-latencies).

For ex if you app is data-intensive, halved caches won't help. It can be disabled it via BIOS settings.

In general, it moves the control of resources from software to hardware and that is usually not desired for performance critical applications.

Note : Its generic name is simultaneous multithreading. Hyperthreading name used by Intel only.

### DYNAMIC CLOCK SPEEDS

Modern CPUs employ dynamic frequency scaling which means there is a min and max frequency per CPU core.

Also [ACPI](https://en.wikichip.org/wiki/skylake_server) defines multiple power states and modern CPUs implement those. P-State is for performance and C-States are for energy efficiency.

You can use Intel's [TurboBoost](https://en.wikichip.org/wiki/skylake_server) or AMD's [TurboCore](https://en.wikichip.org/wiki/skylake_server) to maintain the CPU usage.

Note that SSE usage may also introduce downclocking, therefore they should be used carefully : [Daniel Lemire's article](https://en.wikichip.org/wiki/skylake_server)

## LOAD STORE REALM

### LOAD & STORE BUFFERS

Load and store buffers allow CPU to do out-of-order execution on loads and stores by decoupling speculative execution and committing the results to the cache memory.

Reference : [https://en.wikipedia.org/wiki/Memory\\_disambiguation](https://en.wikipedia.org/wiki/Memory_disambiguation)

### STORE-TO-LOAD FORWARDING

Using buffers for stores and loads to support out of order execution leads to a data synchronization issue. That issue is described in [en.wikipedia.org/wiki/Memory\\_disambiguation#Store\\_to\\_load\\_forwarding](https://en.wikipedia.org/wiki/Memory_disambiguation#Store_to_load_forwarding).

As a solution, CPU can forward a memory store operation to a following load, if they are both operating on the same address.

An example store and load sequence :

```
mov [eax], STORE ; Write the value of ECX register to the memory ; address which is stored in EAX register
```

```
mov ecx, [eax] ; LOAD, Read the value from that memory address ; (which was just used) and write it to ECX register
```

### STORE-TO-LOAD FORWARDING & LHS & PERFORMANCE

Based on [Intel Optimization Manual 3.6.4](https://en.wikichip.org/wiki/skylake_server), store-to-load forwarding may improve combined latency of those 2 operations. The reason is not specified however it is potentially LHS (Load-Hit-Store) problem in which the penalty is a round trip to the cache memory.

<https://en.wikipedia.org/wiki/load-hit-store>

There are several conditions for the forwarding to happen. In case of a successful forwarding, the steps 2 and 3 (a roundtrip to the cache) will be bypassed.

Previous game consoles PlayStation3 and Xbox360 had PowerPC based processors which did [speculative execution](https://en.wikichip.org/wiki/skylake_server) rather than out-of-order execution. Therefore developers had to separately handle LHS by using [flushing](https://en.wikichip.org/wiki/skylake_server) keyword and other methods. [Alan Rusakov's article](https://en.wikichip.org/wiki/skylake_server)

The conditions for a successful forwarding and latency penalties in case of no-forwarding can be found in Agner Fog's [microarchitecture book](https://en.wikichip.org/wiki/skylake_server).

## ARITHMETIC REALM

### ARITHMETIC INSTRUCTION LATENCIES

You can see a set of arithmetic operations from fast to slow below.

The clock cycles are based on Agner Fog's [Instruction Latencies](https://en.wikichip.org/wiki/skylake_server) & Skylake architecture on 64 bit registers.

**Integer operations** : Integer address : 0.25 to 1 clock cycle

**Integer multiplication** : about 3 clock cycles

**Integer division** : about 30 clock cycles

**Integer division** : 24-30 clock cycles

### FLOATING POINTS

X86 uses [IEEE 754](https://en.wikichip.org/wiki/skylake_server) standard for floating points. A 32 bit floating point consists of 3 parts in the memory layout. Below you can see all bits of 32-bit IEEE 754 FP number. Used [https://en.wikichip.org/wiki/skylake\\_server](https://en.wikichip.org/wiki/skylake_server) as visualiser :

A floating point's value is calculated as :  $mantissa \times 2^{exponent}$

IEEE754 also defines [denormal numbers](https://en.wikichip.org/wiki/skylake_server). They are very small / near zero numbers.

As floating points are approximations, denormal numbers are needed to avoid an undefined case of  $0.0$ , but  $0.0$ .

If  $(a \neq b)$  :  $return 1$  if  $(a < b)$  ;  $return 0$  else

Based on Agner Fog's [microarchitecture book](https://en.wikichip.org/wiki/skylake_server), Intel CPUs have a penalty for denormal numbers, for ex : 120 clock cycles on Skylake. They also can be turned on of Intel CPUs.

As for AMD side, the recent Zen architecture CPUs seemingly don't have the same performance degradation.

### X86 EXTENSIONS

X86 extensions are specialised instructions. They have various categories from [cryptography](https://en.wikichip.org/wiki/skylake_server) to [vector instructions](https://en.wikichip.org/wiki/skylake_server).

[Intel Intrinsics Guide](https://en.wikichip.org/wiki/skylake_server) is a good page to explore those extensions.

**SSE** (Streaming SIMD Extensions) is one of the most important ones. **SIMD** stands for "single instruction multiple data". SIMD instructions use wider registers to execute more work in a single go :

Some typical application areas are 3D graphics and quantitative finance.

Apart from arithmetic operations, they can be utilised for string operations as well.

A SIMD based JSON parser : [https://en.wikichip.org/wiki/skylake\\_server](https://en.wikichip.org/wiki/skylake_server)

### X86 EXTENSIONS : SIMD DETAILS

The most recent SIMD instruction sets and their corresponding registers are :

AVX : 128 bits, XMM registers  
AVX2 : 256 bits, YMM registers  
AVX512 : 512 bits, ZMM registers

As for programming, there are also wider data types. The data type diagrams below are for 128 bit AVX :

$mm256$  : 4 x 32 bit floating points

$mm256$  : 2 x 64 bit doubles

$mm256$  : 4 x 32 bit ints

$mm256$  : 2 x 64 bit long longs

Note that SSE instructions require more power, therefore their usage may also introduce downclocking. They should be benchmarked : [Daniel Lemire's article](https://en.wikichip.org/wiki/skylake_server)

## BRANCH PREDICTION REALM

### BRANCH PREDICTION BASICS

Why : CPUs proactively fetch instructions of potentially upcoming branches to utilise the pipeline as much as possible.

**Cost if predicted correctly** : If the right branch was predicted that will increase the throughput as it completed fetching a set of instructions in advance.

**Penalty in case of misprediction** : If the prediction was wrong, that prefetch will be a waste and the cost will be flushing the pipeline.

**What are branch instructions** ? : Unconditional ones (jmp), conditional ones (eg: jne), call/ret.

How : There are auxiliary hardware buffers.

Branch target buffer stores target addresses / instruction pointers of branches. AMD uses multiple level of BTBs : L1 BTB, L2 BTB etc.

Pattern history tables track the history of results (whether it was taken or not) per branch.

A hypothetical pattern history table

BP METHODS : 2-LEVEL ADAPTIVE BRANCH PREDICTION

**Saturating counter** : A 2-bit saturating counter can store 4 strength states.

Whenever a branch is taken it goes stronger.

And whenever a branch is not taken it goes weaker.

**2 level adaptive predictor**

In this method, you store the history of last n occurrences in a history register which is n bits.

Also you create a table called "system history table" for that branch. That pattern history table keeps 2^n rows and each row has a saturating counter.

The branch history register will be used to choose which row will be used from the pattern history table.

Reference : Agner Fog's [microarchitecture book](https://en.wikichip.org/wiki/skylake_server)

### BP METHODS : AMD PERCEPTIONS

They are used in Zen architectures.

A **perception** is basically the simplest form of machine learning. They can be considered as a linear array of weights.

Agner Fog mentions that they are good at predicting very long branches compared to 2-level adaptive branch prediction in his [microarchitecture book](https://en.wikichip.org/wiki/skylake_server).

For details of perception based branch prediction, [Dynamic Branch Prediction with Perceptions by Daniel Lemire and Calvin Lin](https://en.wikichip.org/wiki/skylake_server)

### INTEL LSD (LOOP STREAM DETECTOR)

Intel LSD will detect a loop and store fetching instruction to improve frontend bandwidth. Several conditions mentioned in [Intel Optimization Manual](https://en.wikichip.org/wiki/skylake_server).

Loop body size up to 60 jumps, with up to 15 taken branches, and up to 15 64-byte fetch lines.

No CALL or RET.

No mismatched stack operations (e.g., more PUSH than POP).

More than 10 iterations.

Note that LSD is disabled on Skylake Server CPUs. Reference : [https://en.wikichip.org/wiki/skylake\\_server](https://en.wikichip.org/wiki/skylake_server)

You can consider disabling system patches for speculative execution related vulnerabilities such as Meltdown and Spectre for performance, if it is disabled in your system.

Kernel.org documentation : [https://en.wikichip.org/wiki/skylake\\_server](https://en.wikichip.org/wiki/skylake_server)

Meltdown paper : [https://en.wikichip.org/wiki/skylake\\_server](https://en.wikichip.org/wiki/skylake_server)

Spectre paper : [https://en.wikichip.org/wiki/skylake\\_server](https://en.wikichip.org/wiki/skylake_server)

### ESTIMATED LIMITS : HOW MANY IFs ARE TOO MANY ?

As for max number of writes in BTBs, there are estimations made by stress testing the BTB with conditions of branch instructions.

Intel Xeon Gold 6262 -> roughly 4K  
AMD EPYC 7713 -> roughly 3K

Reference : [Mark Malkov's article on Cloudflare blog](https://en.wikichip.org/wiki/skylake_server)

## CACHE MEMORY REALM

### CACHE BASICS

System memory is made of DRAM cells. Cache memory on the other hand are made of DRAM cells which is much faster than SRAM. On the other hand they are more expensive.

DRAM used in system memories

SRAM used in cache memories

Access time : 50-150 nanoseconds due to capacitor charge/discharge times and other steps

Cost : Cheaper in the price as it has less components

Reference : Ulrich Drepper's [What every programmer should know about memory](https://en.wikichip.org/wiki/skylake_server)

### CACHE ORGANISATION

Caches are organised in multiple levels. As you go upper in that hierarchy, the capacity increases. Therefore LLC term used to indicate the last level of cache.

3 level caches are currently the most common ones. Intel Broadwell architecture had a level caches in the past. It is expected that upcoming AMD CPUs may come with 4 level of caches.

A **cache line** is the smallest addressable unit in cache memories. It is typically 64 bytes.

All the mentioned caches still now have data caches. But there is also **instruction cache** (I-Cache) which store program instructions rather than data to improve throughput of CPU frontend.

In case of a cache hit, the latency is typically single digit nanoseconds. And in case of a cache miss, we need a round trip to the system memory and total latency becomes 3 digit nanoseconds.

The disadvantage is that the time needed to walk the page table increases due to a new level of indirection.

### CACHE ORGANISATION

Load & Store Buffers

L1 Data Cache

LLC : Last level cache

System memory

### N-WAY SET ASSOCIATIVITY

Cache capacities are much smaller than the system memory. Moreover, software can use various regions of their address space. So if there was one to one mapping of a fully sequential memory that would lead to cache misses most of the time. Therefore there is a need for efficient mapping between the cache memory and the system memory.

In N-Way set associativity, caches are divided to groups of sets. And each set will have N cache lines. The mapping information is stored in bits of addresses. A cache address has 3 parts :

TAG : used as unique identifier per cache line

SET : used to determine the set in the cache

OFFSET : used to determine the actual bytes in the target cache line

The pseudocode below shows steps for searching a single byte in the cache memory :

Get tag, set and offset from the address

For each line in the current set (which we just found out) if tag of the current line equals to tag (which we just found out) read and return data using offset of CACHE HIT

If there was no matching tag, it is a cache miss

### DIRECT CACHE ACCESS

Modern NICs come with a DMA (Direct Memory Access) engine and can transfer data directly to diverse ring buffers which reside on the system memory.

DMA mechanism doesn't require CPU involvement. Though mechanism initiated by CPU, therefore CPU support needed.

DCA bypasses the system memory and can transfer to directly LLC of CPUs that support this feature.

Intel refers to their technology as DIO (Direct IO).

Reference : [Intel documentation](https://en.wikichip.org/wiki/skylake_server)

## MULTICORE REALM

### COHERENCY

#### CACHE COHERENCY PROTOCOLS

As LLC is typically shared by multiple cores, cache coherency protocols are needed to avoid data hazards. Intel CPUs use MESI and AMD CPUs use MOESI, however both heavily depend on MESI protocol. There are 4 states for a cache line in MESI protocol :

**Modified** - cache line is present only in the current cache and has been modified from its value in system memory. Exclusive - cache line is present only in the current cache and matches its value in system memory. Shared - cache line is present here and the other cache line matches its value in system memory. Invalid - cache line is unused

Allowed state transitions in MESI protocol :

As for state transition costs, M and S states are the cheapest ones as they don't involve cross cache communication. Therefore it is useful to keep data in those states as long as possible. That can be achieved by using cache variables whenever it is applicable.

If those happen in higher rates and if cache lines transition between cores rapidly, that situation is called **cache ping-pong**.

Whenever a page table entry is modified by any of the cores, that particular TLB entry is invalidated in all cores via IPIs. This time is not by hardware but initiated by operating system.

IPI : Interprocessor interrupt

**TLB SHOOTDOWNS** : One of the cores modifies a table entry

**PAGE TABLE ON SYSTEM MEMORY** : One of the cores modifies a table entry

**OS** : One of the cores modifies a table entry

Reference : [https://en.wikichip.org/wiki/skylake\\_server](https://en.wikichip.org/wiki/skylake_server)

### MEMORY REORDERINGS & SYNCHRONISATION

#### MEMORY REORDERINGS

The term memory ordering refers to the order in which the processor issues reads (loads) and writes (stores). Based on [Intel Software Developer's Manual Volume3](https://en.wikichip.org/wiki/skylake_server) 8.2.3.4, there is only one kind of memory reordering that can happen. Loads can be reordered with earlier stores if they use different memory locations. That reordering will not happen if they use the same address.

**CORE1** : `x = 0 ; initially 0 ; mov [x], 1 ; STORE TO X ; mov [result1], x ; LOAD FROM Y`

**CORE2** : `x = 0 ; initially 0 ; mov [y], 1 ; STORE TO Y ; mov [result2], x ; LOAD FROM X`

In case of reordering, result1 and result2 above can both end up as zero in both cores. Note that, apart from CPUs, also compilers can do memory reordering.

[https://en.wikichip.org/wiki/skylake\\_server](https://en.wikichip.org/wiki/skylake_server)

#### INSTRUCTIONS TO AVOID REORDERINGS

Reorderings can be avoided by using serialising instructions such as SFENCE, LFENCE, and MFENCE : [Intel Software Developer's Manual Volume3](https://en.wikichip.org/wiki/skylake_server) 8.3 defines them as :

These instructions force the processor to complete all modifications to flags, registers, and memory by previous instructions and to drain all buffered writes to memory before the next instruction is fetched and executed.

There is also bus locking "LOCK" prefix ([Intel Software Developer's Manual Volume3](https://en.wikichip.org/wiki/skylake_server) 8.1.2) which can be used to avoid reorderings.

High level languages expose memory fence APIs which are omitted as one of those methods:

The image is taken from the online book [Compiler Explorer](https://en.wikichip.org/wiki/skylake_server).

#### ATOMIC OPERATIONS

An atomic operation means that there will be no other operations going on during the execution. From point of execution, an atomic operation is indivisible and nothing can affect its execution.

The most common type of atomic operations are RMW (read-modify-write) operations.

#### ATOMIC RMW OPERATIONS : COMPARE-AND-SWAP

CAS instruction (CMPXCHG) reads values of 2 operands. It then compares them and if they are equal, it swaps values. All the operations are atomic / uninterruptible. It can be used to implement lock free data structures.

#### ATOMIC RMW OPERATIONS : TEST-AND-SET

Test-and-set is an atomic operation which writes to a target memory and returns its old value. It is typically used to implement spin locks.

#### TRANSACTIONAL EXTENSIONS

Transactional memory areas are programmer specified critical sections. Reads and writes in those areas are done atomically. ([Intel Optimization Manual](https://en.wikichip.org/wiki/skylake_server) section 16)

However due to another hardware security issue, Intel disabled them from Skylake Coffee Lake CPUs : [https://en.wikichip.org/wiki/skylake\\_server](https://en.wikichip.org/wiki/skylake_server)

### LIMITING CONTENTION BETWEEN CORES

#### ALLOCATING A PARTITION OF LLC

You can allocate a partition of the shared CPU last level cache for your performance sensitive application to avoid evictions on Intel CPUs that support CAT feature.

CAT : Cache allocation tech, reference : [Intel CAT page](https://en.wikichip.org/wiki/skylake_server)

For AMD equivalent, AMD64 QOS Extensions introduced starting from Zen2: [https://en.wikichip.org/wiki/skylake\\_server](https://en.wikichip.org/wiki/skylake_server)