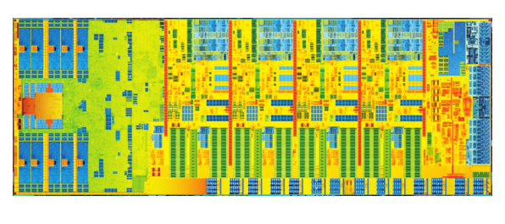
MICROARCHITECTURE CHEAT SHEET

X86 CPUs & Performance



Die photo of a quadcore CPU , Copyright of Intel

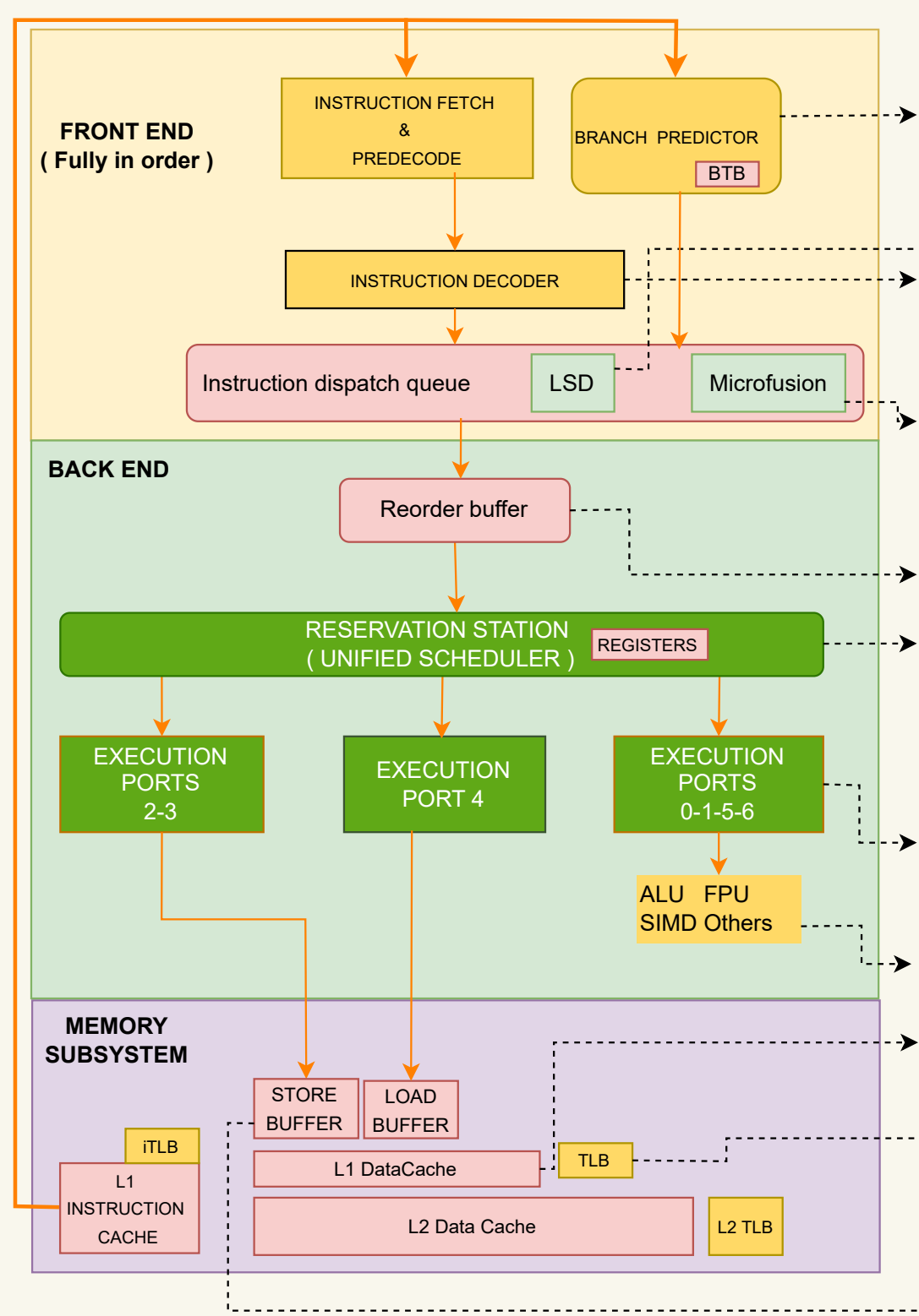
LAST UPDATE DATE : 31 OCT 2022

FOR LATEST VERSION: www.github.com/akhin/microarchitecture-cheatsheet

AUTHOR: AKIN OCAL akin_ocal@hotmail.com

PIPELINE REALM : INSIDE AN INDIVIDUAL CORE

A SIMPLIFIED OVERVIEW (BASED ON INTEL SKYLAKE)



Speculative execution
See the branch prediction realm for branch predictor, BTB and LSD.

UOPS/MicroOps
CISC instructions are split into smaller RISC instructions called as uops for more throughput.

Microfusion
Memory-write and read-modify uops are combined into a single uop for better throughput.

Out of order execution
This is done at the backend to improve the throughput.

However the results have to be output in order. So reordering at the end done by help of the ROB (Reorder buffer).

Super scalar execution
Multiple uops executed in parallel.

Arithmetic operations
See the arithmetic realm.

Caches
See the cache memory realm .

TLBs
See the virtual memory realm .

Load,Store buffers
See the load-store realm .

Based on: Skylake server on en.wikichip.org

PIPELINE PARALLELISM & PERFORMANCE

Pipeline diagrams : The diagrams below in the following topics are outputs from an online microarchitecture analysis tool UICA 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.

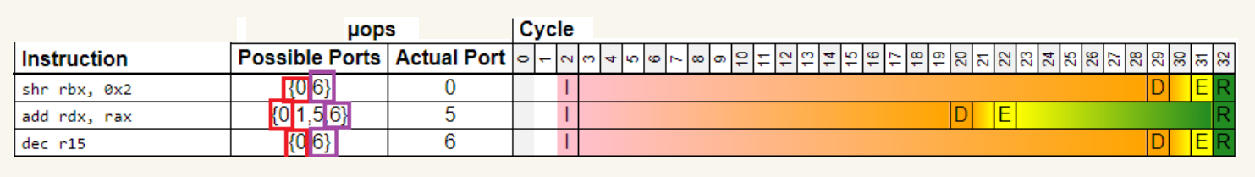
IPC : As for pipeline performance, typically IPC is used. It stands for "instructions per cyle".
A higher IPC value usually means a better throughput.
You can measure IPC with perf : https://perf.wiki.kernel.org/index.php/Tutorial

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

P	Predecoded
Q	Added to IDQ
I	Issued
r	Ready for dispatch
D	Dispatched
E	Executed
R	Retired

Instruction lifecycle states in UICA diagrams

CONTENTION FOR EXECUTION PORTS IN THE PIPELINE



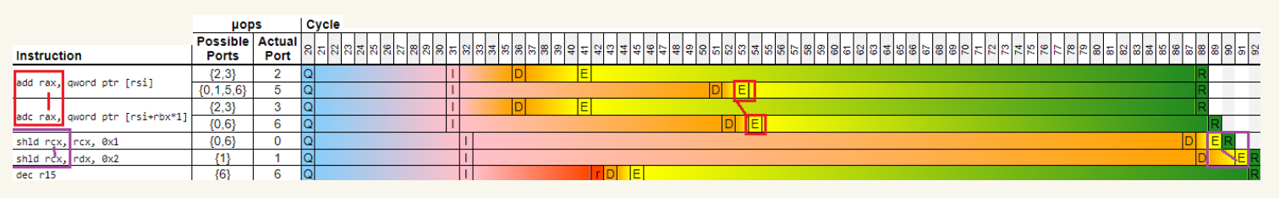
Instruction	µops Possible Ports	Actual Port
shr rbx, 0x2	{0,6}	0
add rdx, rax	{0,1,5,6}	5
dec r15	{0,6}	6

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 Bakhvalov`s article

INSTRUCTION STALLS DUE TO DATA DEPENDENCY



Instruction	µops Possible Ports	Actual Port
add rax, qword ptr [rsi]	{2,3}	2
	{0,1,5,6}	5
adc rax, qword ptr [rsi+rbx*1]	{2,3}	3
	{0,6}	6
shld rcx, rcx, 0x1	{0,6}	0
shld rcx, rdx, 0x2	{1}	1
dec r15	{6}	6

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 Bakhvalov`s article

RDTSCP INSTRUCTION FOR MEASUREMENTS

RDTSCP 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 RDTSC combination in older systems that don`t support RDTSCP.

ESTIMATING INSTRUCTION LATENCIES

Based on Agner Fog`s Instruction tables, RDTSCP reciprocal throughput (clock cycle per instruction) is 32 on Skylake microarchitecture:

-> 1 cycle @4.5GHZ is 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, 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.

For ex if your app is data-intensive , halved caches won`t help. It can be disabled it via BIOS settings.

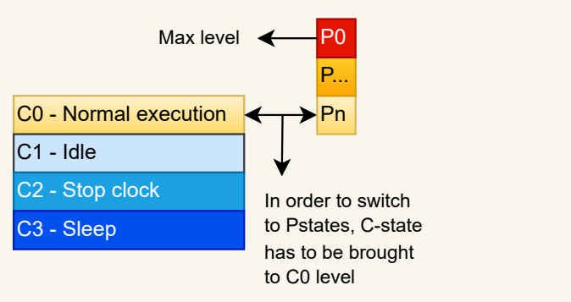
In general, it moves the control of resources from software to hardware and that is usually not desired for performance critical applications.

DYNAMIC FREQUENCIES

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

ACPI : ACPI defines multiple power states and modern CPUs implement those. P-State`s are for performance and C states are for energy efficiency.

You can use Intel`s Turboboost or AMD`s Turbocore to maximise the CPU usage.



Intel CPUs number of active cores & SIMD usage : Number of active cores can introduce downclocking regardless of SIMD usage . Therefore disabling unused cores can further improve frequency of your target cores.

Example Xeon Gold 5120 frequency table : https://en.wikichip.org/wiki/intel/xeon_gold/5120#Frequencies

Use of some SIMD instructions may introduce further downclockings. You shall refer to your CPU`s frequency table. Regarding SIMD and downclockings : Daniel Lemire`s article

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 commiting the results to the cache memory.

Reference : 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 syncronisation issue. That issue is described in 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 :

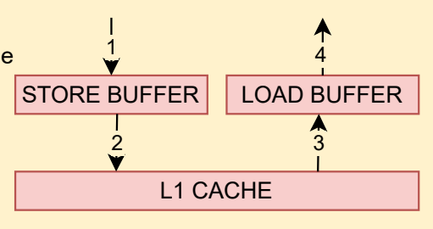
```
mov [eax],ecx  ; STORE , Write the value of ECX register to the memory
               ; address which is stored in EAX register
mov ecx,[eax]  ; LOAD, Read the value from that memory address
               ; ( which was just used) and write it to ECX register
```

STORE-TO-LOAD FORWARDING & LHS & PERFORMANCE

Based on Intel Optimization Manual 3.6.4, 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.



The conditions for a successful forwarding and latency penalties in case of no-forwarding can be found in Agner Fog`s microarchitecture book.

What would happen without forwarding ? : Previous game consoles PlayStation3 and Xbox360 had PowerPC based processors which did in-order-execution rather than out-of-order execution. Therefore developers had to separately handle LHS by using restrict keyword and other methods : Elan Ruskin`s article

ARITHMETIC REALM

ARITHMETIC INSTRUCTION LATENCIES

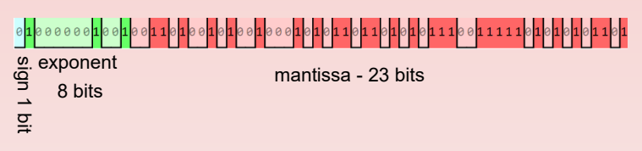
You can see a set of arithmetic opertions from fast to slow below.

The clock cycles are based on Agner Fog`s Instruction tables & Skylake architecture on 64 bit registers.

- Bitwise operations , integer add/sub : 0.25 to 1 clock cycle
- Floating point add : 3 clock cycles
- Floating point multiplication : about 5 clock cycles
- Floating point division: about 14-16 clock cycles
- Integer division : 24-90 clock cycles

FLOATING POINTS

X86 uses IEEE 754 standard for floating points. A 32 bit floating point consists of 3 parts in the memory layout. Below you can see all bits of 1234.5678 FP number. Used bartaz.github.io/ieee754-visualization as visualiser :



A floating point`s value is calculated as : ±mantissa × $2^{exponent}$

IEEE754 also defines denormal numbers. They are very small / near zero numbers.

As floating points are approximations, denormal numbers are needed to avoid an undesired case of : a!=b but a-b=0
Without denormals the code to the right would invoke a divide-by-zero exception.
Reference : Bruce Dawson`s article

```
float GetInverseOfDiff(float a, float b)
{
  if (a != b)
  return 1.0f / (a - b);
  return 0.0f;
}
```

Based on Agner Fog`s microarchitecture book, Intel CPUs have a penalty for denormal numbers, for ex: 129 clock cycles on Skylake. They also can be turned off on Intel CPUs.

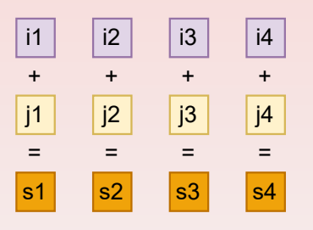
As for AMD side, the recent Zen architecture CPUs seemingly don`t have the same performance degradation.

X86 EXTENSIONS

x86 extensions are specialised instructions. They have various categories from cryptography to neural network operations.

Intel Intrinsics Guide 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 :



In the example above, an array 4 integers (i1 to i4) are added to another array of integers (j1 to j4). The result is also an array of sums (s1 to s4). In this example, 4 add operations are executed by a single instruction.

They play key role in compilers` vectorisation optimisation : GCC auto vectorisation

Apart from arithmetic operations, they can be utilised for string operations as well.
A SIMD based JSON parser : https://github.com/simdjson/simdjson

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 :

__m128 , 4 x 32 bit floating points	Float	Float	Float	Float
__m128d , 2 x 64 bit doubles	Double		Double	
__m128i , 4 x 32 bit ints	int	int	int	int
__m128l , 2 x 64 bit long longs	long long		long long	

Note that SSE instructions require more power, therefore their usage may also introduce downclocking. They should be benchmarked : Daniel Lemire`s article

BRANCH PREDICTION REALM

BRANCH PREDICTION BASICS

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

Gain 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.

	1	2	3	4	...
branch 1	T	T	NT	T	...
branch ...	NT	NT	NT	T	...
branch n	T	NT	NT	NT	...

A hypothetical pattern history table
T: taken, NT : not taken

CONDITIONAL MOVE INSTRUCTION

CMOV (Conditional move) instruction takes additional time in order to compute also the condition. Therefore they don`t introduce extra load to branch prediction. They can be used to eliminate branches.
Reference : Intel Optimisation Manual 3.4.1.1

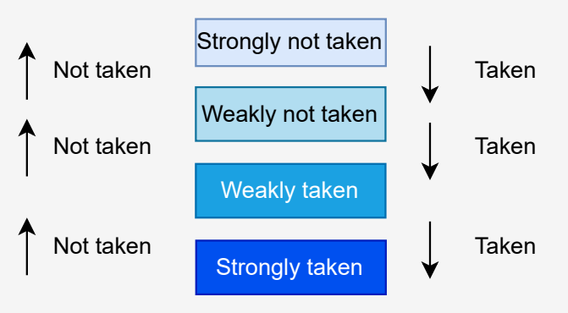
BP METHODS : 2-LEVEL ADAPTIVE BRANCH PREDICTION

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 occurences in a history register which is n bits.

Also you create a table called "pattern 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

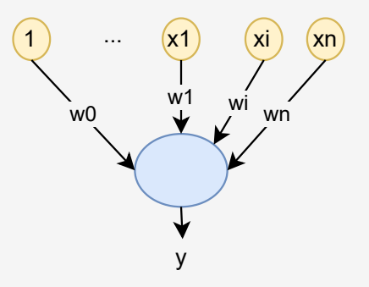
BP METHODS : AMD PERCEPTRONS

They are used in Zen arcihtectures.

A perceptron 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.

For details of perceptron based branch prediction: Dynamic Branch Prediction with Perceptrons by Daniel Jimenez and Calvin Lin



The output Y (in this case whether a branch taken or not) is calculated by dot product of the weight vector and the input vector.

INTEL LSD (LOOP STREAM DETECTOR)

Intel LSD will detect a loop and stop fetching instruction to improve frontend bandwidth. Several conditions mentioned in Intel Optimization Manual :

- *Loop body size up to 60 µops, 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 ~20 iterations.*

Note that LSD is disabled on Skylake Server CPUs. Reference : https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(server)#Front-end

DISABLING SPECULATIVE EXECUTION PATCHES

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

Kernel.org documentation : https://www.kernel.org/doc/html/latest/admin-guide/kernel-parameters.html

Red Hat Enterprise documentation : https://access.redhat.com/articles/3311301

Meltdown paper : https://meltdownattack.com/meltdown.pdf

Spectre paper : https://spectreattack.com/spectre.pdf

ESTIMATED LIMITS : HOW MANY IFs ARE TOO MANY ?

As for max number of entries in BTBs, there are estimations made by stress testing the BTB with sequences of branch instructions :

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

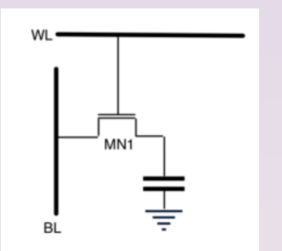
Reference : Marek Majkovski`s article on Cloudflare blog

CACHE MEMORY REALM

CACHE MEMORY VS SYSTEM MEMORY

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

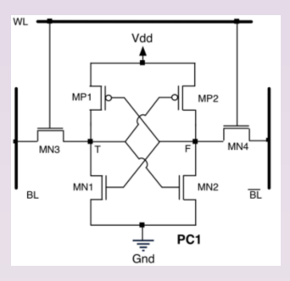
DRAM used in system memories



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

Cost : Cheaper in the price as it has less components

SRAM used in cache memories



Access time : Under 1 nanosecond

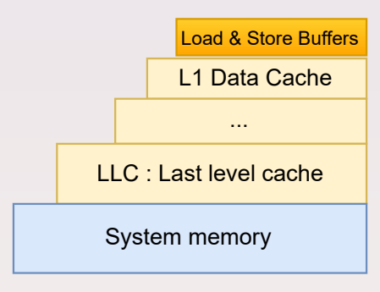
Cost : Expensive in the price due to 6 transistors

Reference : Ulrich Drepper`s What every programmer should know about memory

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 4 level caches in the past. Also upcoming AMD CPUs may come with 4 level of caches.



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

All the mentioned caches till now were data caches. But there is also instruction cache (iCache) 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.

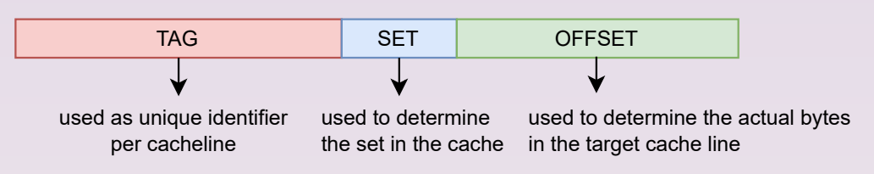
HARDWARE AND SOFTWARE PREFETCHING

Intel Optimisation Manual 3.7 describes prefetching. Hardware prefetchers prefetch data and instruction to cache lines automatically. Developers can also use instruction _mm_prefetch to prefetch data explicitly. That is called as software prefetching. However performance improvement by using software prefetcher is controversial : Daniel Lemire`s article

N-WAY SET ASSOCIATIVITY

Cache capacities are much smaller than the system memory. Moreover , softwares 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
SET
OFFSET
used as unique identifier per cacheline
used to determine the set in the cache
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 line equals to tag ( which we just found out )
        read and return data using offset // CACHE HIT
If there was no matching tag, it is a cache miss
```

The level of associtivity (the number of ways) is a trade off between the search time and the amount of system memory we can map.

DIRECT CACHE ACCESS

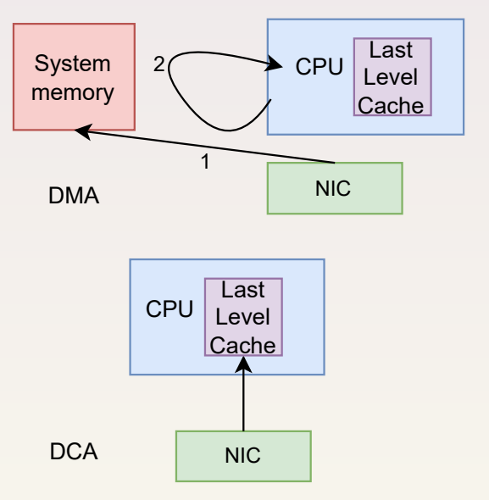
Modern NICs come with a DMA (Direct Memory Access) engine and can transfer data directly to drivers' ring buffers which reside on the system memory.

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 DDIO (Direct I/O).

Reference : Intel documentation



VIRTUAL MEMORY REALM

VIRTUAL MEMORY ORGANISATION

Why virtual memory ?
Because cumulative memory requirement of multiple processes in an OS can be more than the system capacity.

It is basically for sharing memory resources between multiple processes.

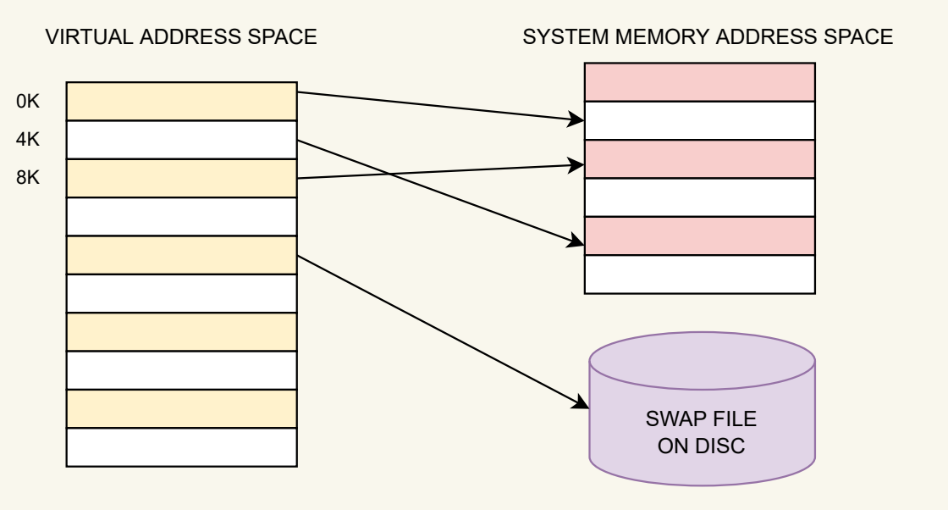
It also provides security by isolating processes.

Pages
- Minimum addressable virtual space that can be requsted from OS.

- Typically 4KB

Swapping
In case of out of memory, process memory will be evicted to the disc.

Page faults
Happens when the page is not on physical memory but on the swap file which is on the harddrive.



TLB PRESSURE & HUGE PAGES

TLB pressure

If each page is 4K , that increases the load on the TLB buffer.

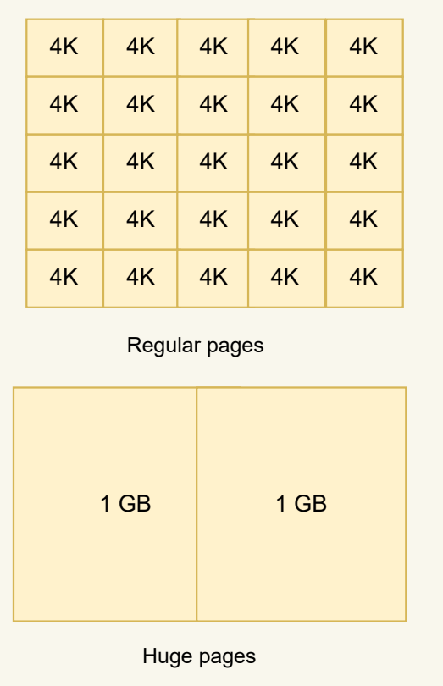
CPU support for larger pages

x86-x64 CPUs support huge pages from 2MB to 1GB to reduce the pressure on TLB.

OS support

Linux implementation refers to them as huge pages and Windows calls them as large pages.

You shall check your OS and CPU in combination to find out the supported sizes.



VIRTUAL MEMORY ADDRESS TRANSLATION

Address translation & Page table

CPUs work with virtual addresses and those addresses need to be converted to physical addresses.

Page table structures on system memory are used for this purpose.

TLB (Translation lookaside buffer)

TLBs are caches in CPUs to make the translation process faster. Modern CPUs have multiple levels of TLBs.

(Intel refers to L2 TLB as sTLB)

TLB Shootdowns

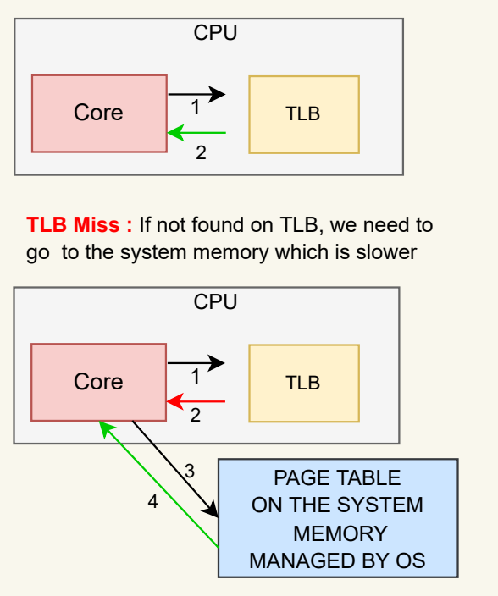
See the multicore realm below.

iTLB

Apart from data TLB , there is also iTLB for caching addresses of instructions on both Intel and AMD architectures.

TLB Hit No need to access page table

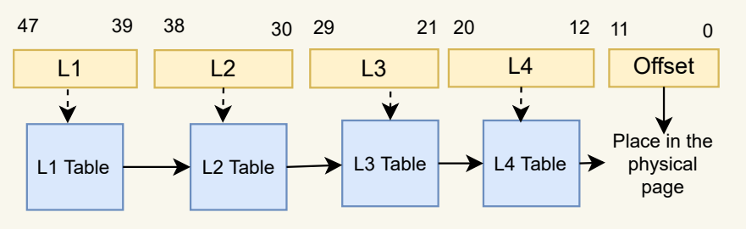
TLB Miss : If not found on TLB, we need to go to the system memory which is slower



PAGE TABLE WALKING

Even with pages which group addresses, having all pages in a page table would still need too much storage on 64 bit systems. Therefore page tables are implemented hierarchically.

Memory is divided into address spaces. And there is a tree data structure for each address space in the page table. Processes have to "walk the page table" level by level in the hierarchy to find out the actual address :



4 level page tables is the most common one. In the diagram above first 48 bits of a 64 bit address are used for page table walking. All of 48 bits have to be used in order to find out the final actual address. (For all details : Intel Software Developer`s Manual Volume3 4.5)

Intel CPUs started to support 5 level tables since Ice Lake.
The advantage of another level is that you can address even more space.

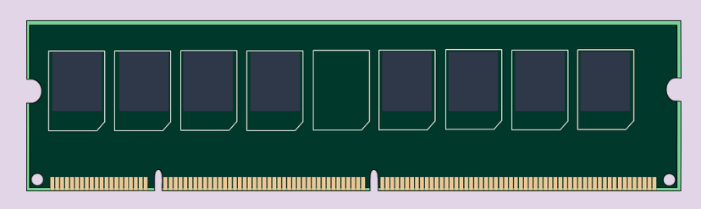
The disadvantage is that the time needed to walk the page table increases due to a new level of indirection.

SYSTEM MEMORY REALM

DDR RAMs

DDR RAMs are the most common commodity hardware as system memory.

They are found in forms of DIMMs (Dual inline memory module) / RAMSticks.

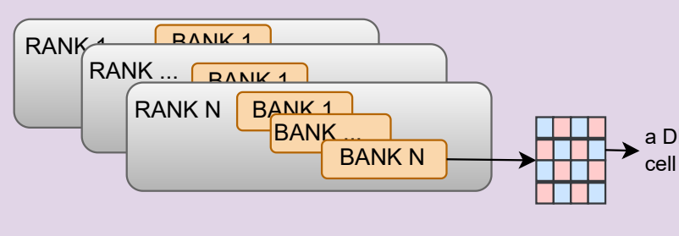


A DIMM. Click for image source

Organisation

System memory / RAM is organised as collection of ranks.

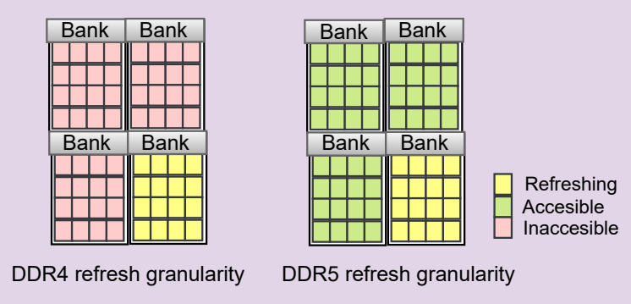
Each rank have banks which are collection of DRAM cells per bit.



DRAM refreshes

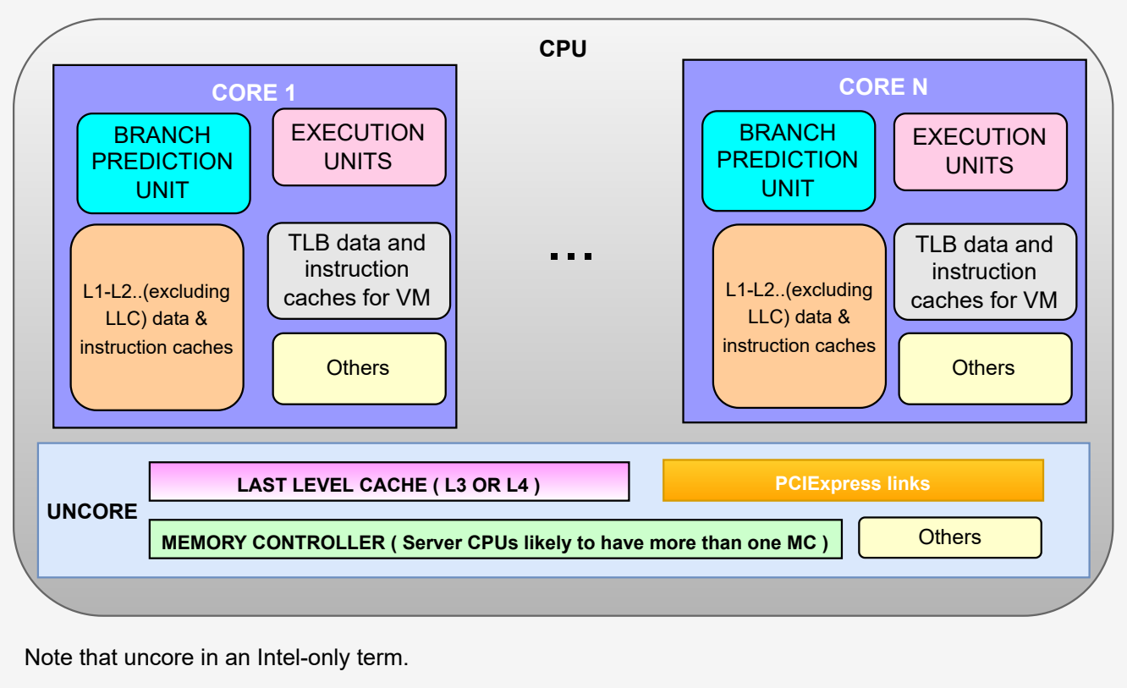
DRAM circuits use capacitors which lose their charge over time. (See the cache memory realm). So RAMs have to refresh their DRAM cells periodically.

As for DDR4, refreshing is rank-level which means the other banks in the same rank become inaccesible. DDR5 comes with same-bank-refresh feature which allows a more fine-grained bank-level refresh. Therefore it can offer a higher throughput.



MULTICORE REALM

TOPOLOGICAL OVERVIEW - INTEL CPUS


CPU
CORE 1
BRANCH PREDICTION UNIT
EXECUTION UNITS
L1-L2..(excluding LLC) data & instruction caches
TLB data and instruction caches for VM
Others
...
CORE N
BRANCH PREDICTION UNIT
EXECUTION UNITS
L1-L2..(excluding LLC) data & instruction caches
TLB data and instruction caches for VM
Others
UNCORE
LAST LEVEL CACHE (L3 OR L4)
PCIExpress links
MEMORY CONTROLLER (Server CPUs likely to have more than one MC)
Others

Note that uncore in an Intel-only term.

Exception of E-cores : An exception to the above diagram is Intel`s recent E-cores. E-cores are meant for power efficiency and paired with less resources. For ex: Alder Lake CPUs` E-cores also share L2 cache.

Reference : https://www.anandtech.com/show/16959/intel-innovation-alder-lake-november-4th

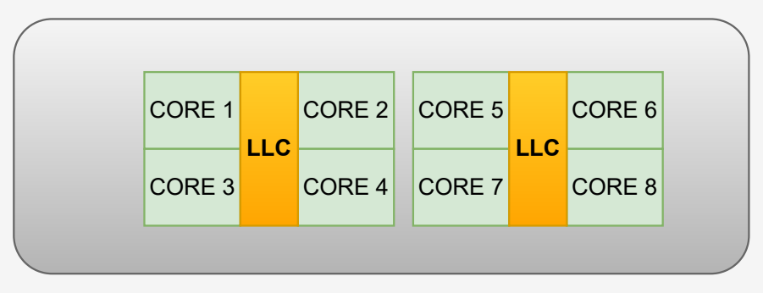
TOPOLOGICAL OVERVIEW - AMD CPUS

Most of AMD topology is similar to Intel. However starting from Zen microarchitecture, one key difference is CCXs :

AMD CPUs are designed as group of 4 cores which is called as CCX (Core complex) by AMD.
And there is one LLC per each CCX/quad core.

So practically the max number of cores competing for the LLC (without simultanenous multithreading) is 4 in recent AMD CPUs :

An example 8 core CPU with 2 CCXs :



AMD CCX structure :There will be multiple LLCs for each group of 4 cores

Reference : https://en.wikichip.org/wiki/amd/microarchitectures/zen#CPU_Complex_.28CCX.29

COHERENCY

CACHE COHERENCY : PROTOCOLS

As LLC is typically shared by multiple cores, cache coherency protocols are needed to avoid data hazards. Intel CPUs use MESIF 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 in other cache lines and matches its value in system memory
Invalid – cache line is unused

Allowed state transitions in MESI protocol :

	M	E	S	I
M	N	N	N	Y
E	N	N	N	Y
S	N	N	Y	Y
I	Y	Y	Y	Y

As for state transition costs, M and E 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 cached variables whereever it is applicable.

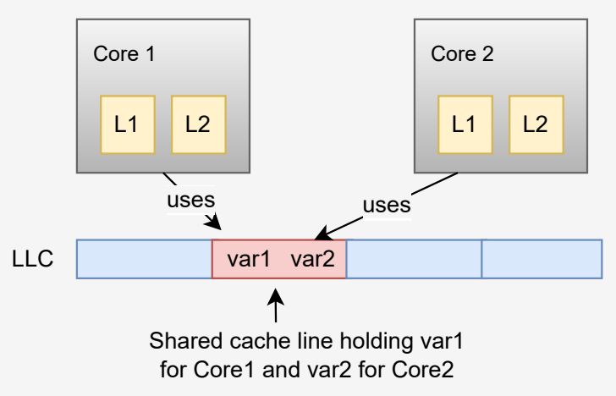
Intel MESIF : https://en.wikipedia.org/wiki/MESIF_protocol
AMD MOESI : https://en.wikipedia.org/wiki/MOESI_protocol

CACHE COHERENCY : FALSE SHARING AND CACHE PING-PONGING

If Core1 updates "var1" variable , that will be touching that shared cache line.

That change will then need to be propagated to all other cores by the cache coherency protocol , even though other cores are not using that variable. That situation is called false sharing.

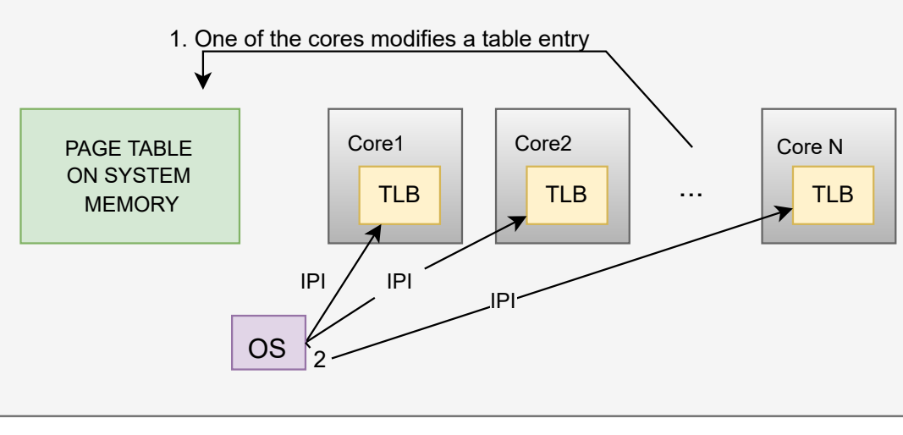
If those happen in higher rates and if cache lines transerred between cores rapidly , that situation is called as cache ping-pong.



VIRTUAL MEMORY PAGE TABLE COHERENCY : TLB SHOOTDOWNS

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

IPI : Interprocessor interrupt


1. One of the cores modifies a table entry
PAGE TABLE ON SYSTEM MEMORY
Core1
TLB
Core2
TLB
...
Core N
TLB
IPI
IPI
IPI
OS
2

MEMORY REORDERINGS & SYNCRONISATION

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 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                             CORE2

; x and y initially 0             ; x and y initially 0
mov [x], 1 ; STORE TO X           mov [y], 1 ; STORE TO Y
mov [result1], y ; LOAD FROM 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 : Jeff Preshing`s article : Memory Ordering at Compile Time

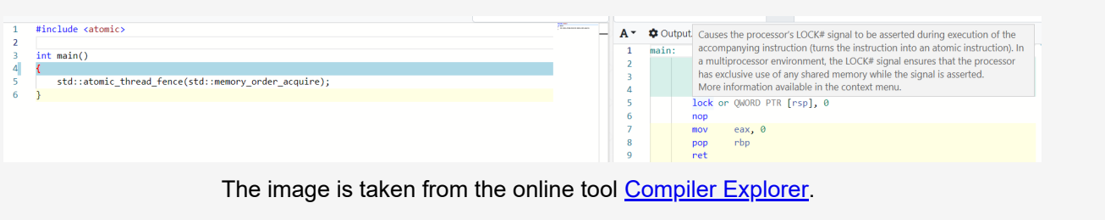
INSTRUCTIONS TO AVOID REORDERINGS

Reorderings can be avoided by using serialising instructions such as SFENCE, LFENCE, and MFENCE : Intel Software Developer`s Manual Volume3 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 8.1.2) which can be used as well to avoid reorderings.

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



```
#include <atomic>

int main()
{
    std::atomic_thread_fence(std::memory_order_acquire);
}
```

```
main:
        lock or QWORD PTR [rsp], 0
        nop
        mov     eax, 0
        pop     rbp
        ret
```

The image is taken from the online tool Compiler Explorer.

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 MEMORY

Transactional memory areas are programmer specified critical sections. Reads and writes in those areas are done atomically. (Intel Optimization Manual section 16)

However due to another hardware security issue, Intel disabled them from Skylake to Coffee Lake CPUs : https://www.theregister.com/2021/06/29/intel_tsx_disabled/

AMD equivalent is called as "Advanced Syncronisation Facility". According to Wikipedia article, no released AMD processors implemented it.

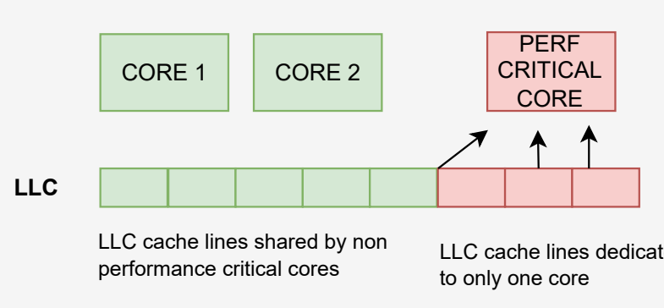
LIMITING CONTENTION BETWEEN CORES

ALLOCATING A PARTITION OF LLC (SERVER CLASS CPUS)

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

CDP (Code and data prioritisation) allows developers to allocate LLC on code basis : Intel`s CDP page on supported CPUs.


CORE 1
CORE 2
PERF CRITICAL CORE
LLC
LLC cache lines shared by non performance critical cores
LLC cache lines dedicated to only one core

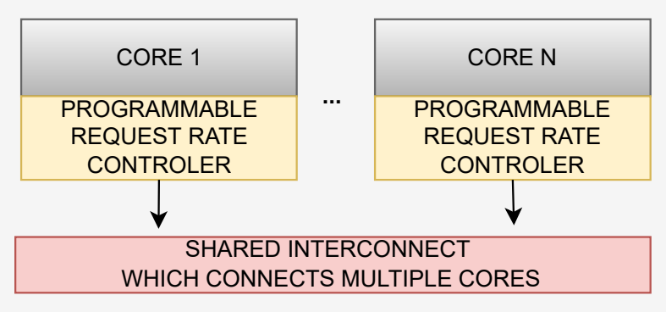
On AMD side, QOS Extensions were introduced starting from Zen2. Corresponding technologies are called as "Cache allocation enforcement" and "Code and data prioritisation": https://developer.amd.com/wp-content/resources/56375_1.00.pdf

MEMORY BANDWIDTH THROTTLING (SERVER CLASS CPUS)

You can throttle memory bandwidth per CPU core on Intel CPUs that support MBA. Each core can be throttled with their request rate controller units.

MBA : Memory bandwidth allocation , reference : Intel MBA page

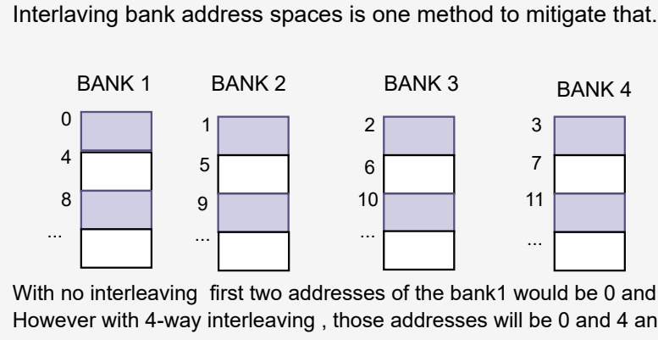
For AMD equivalent, QOS Extensions were introduced starting from Zen2: https://developer.amd.com/wp-content/resources/56375_1.00.pdf


CORE 1
PROGRAMMABLE REQUEST RATE CONTROLER
...
CORE N
PROGRAMMABLE REQUEST RATE CONTROLER
SHARED INTERCONNECT WHICH CONNECTS MULTIPLE CORES

INTERLEAVING FOR REDUCING CONTENTION ON SYSTEM MEMORY

System memory / RAM is organised as collection of ranks.
Each rank have banks which are collection of DRAM cells per bit.

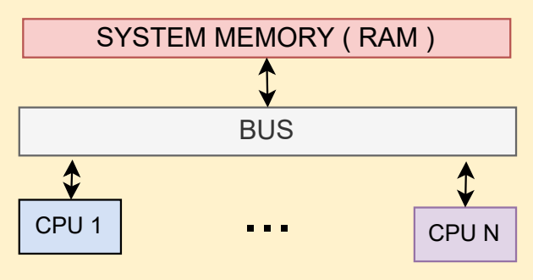
Read and write requests are done at bank level. Therefore if multiple cores try to access to the same bank , there will be a contention.
Interlaving bank address spaces is one method to mitigate that.



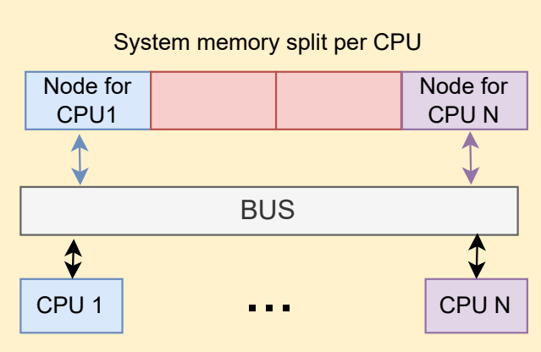
With no interleaving first two addresses of the bank1 would be 0 and 1.
However with 4-way interleaving , those addresses will be 0 and 4 and so on.
Therefore a 4-byte sequential read will be distributed to multiple banks.

MULTICPU REALM

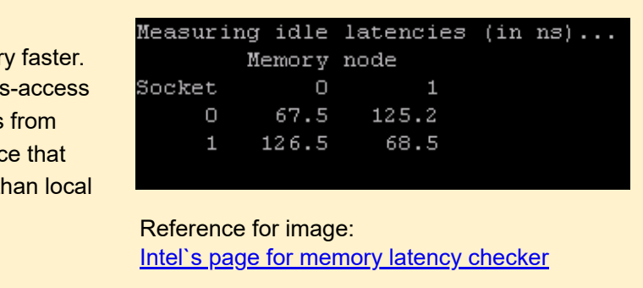
SMP (Symmetric multiprocessing)
All CPUs use a single bus to access the same system memory. CPUs may slow down each other as there may be a contention for access to the banks.



NUMA (Non uniform memory access)
The system memory is organised as nodes and each node is meant to be accessed by only one CPU, therefore memory access is split. So CPUs don`t compete with each other.



Cross NUMA accesses
A CPU can access to its local memory faster. However that also means that a cross-access will be slower. The side screenshot is from Intel`s memory latency checker. Notice that cross acceses are two times slower than local accesses.



```
Measuring idle latencies (in ns)...
        Memory node
Socket       0       1
     0    67.5   125.2
     1   126.5    68.5
```

Reference for image:
Intel`s page for memory latency checker

Sub NUMA Cluster mode (starting from Skylake , formerly COD)

Intel`s SNC splits LLC into disjoint clusters based on address range. You can bind each cluster to a set of memory controllers.

Scalable NUMA-aware apps can benefit from this extra LLC & MC based resource-partitioning.

The lstopo image below is from a 2 Intel Xeon CPU system, each with 28 cores. (Since hyperthreading is enabled it displays 112 cores instead of 56)

That configuration gives 2 more NUMA nodes which means further scalability to benefit from :

Reference for image : Intel® 64 and IA-32 Architectures Optimization Reference Manual Chapter10

ABOUT ALL REALMS

INTEL`S TOPDOWN MICROARCHITECTURE ANALYSIS METHOD

Intel`s Top Down analysis is hierarchical organization of microarchitecture events. It is documented in Intel Optimisation Manual Appendix B1.

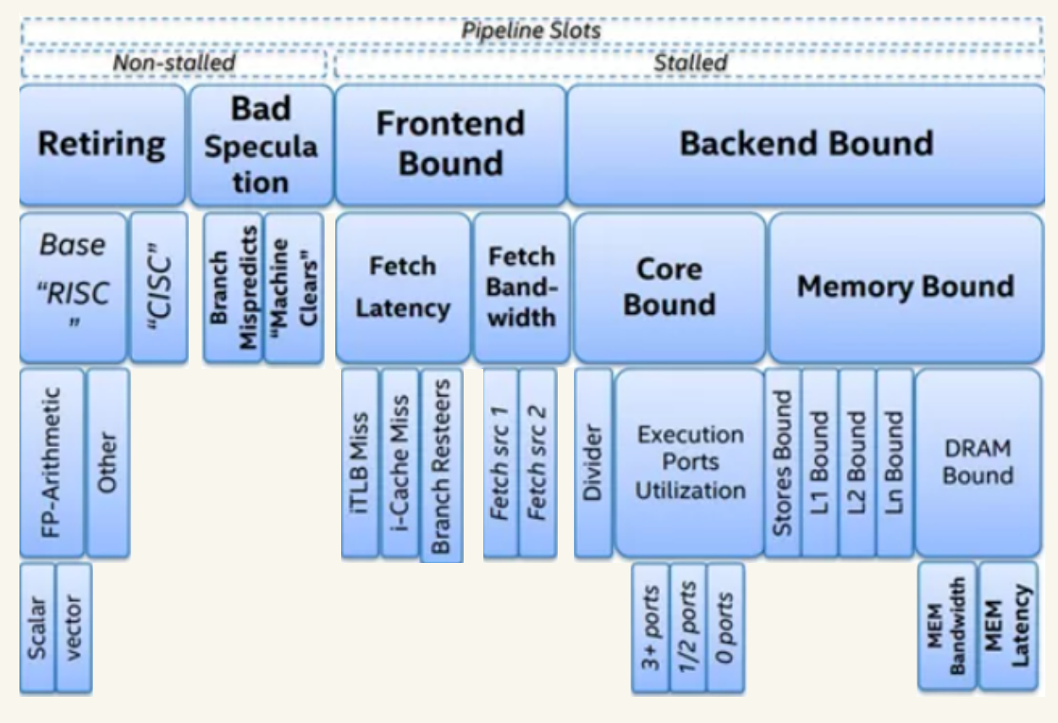
There are 2 main categories for stalls :

1. Frontend : A typical example is large code sizes leading to instruction cache misses.

2. Backend : Usually either memory bound or compute bound

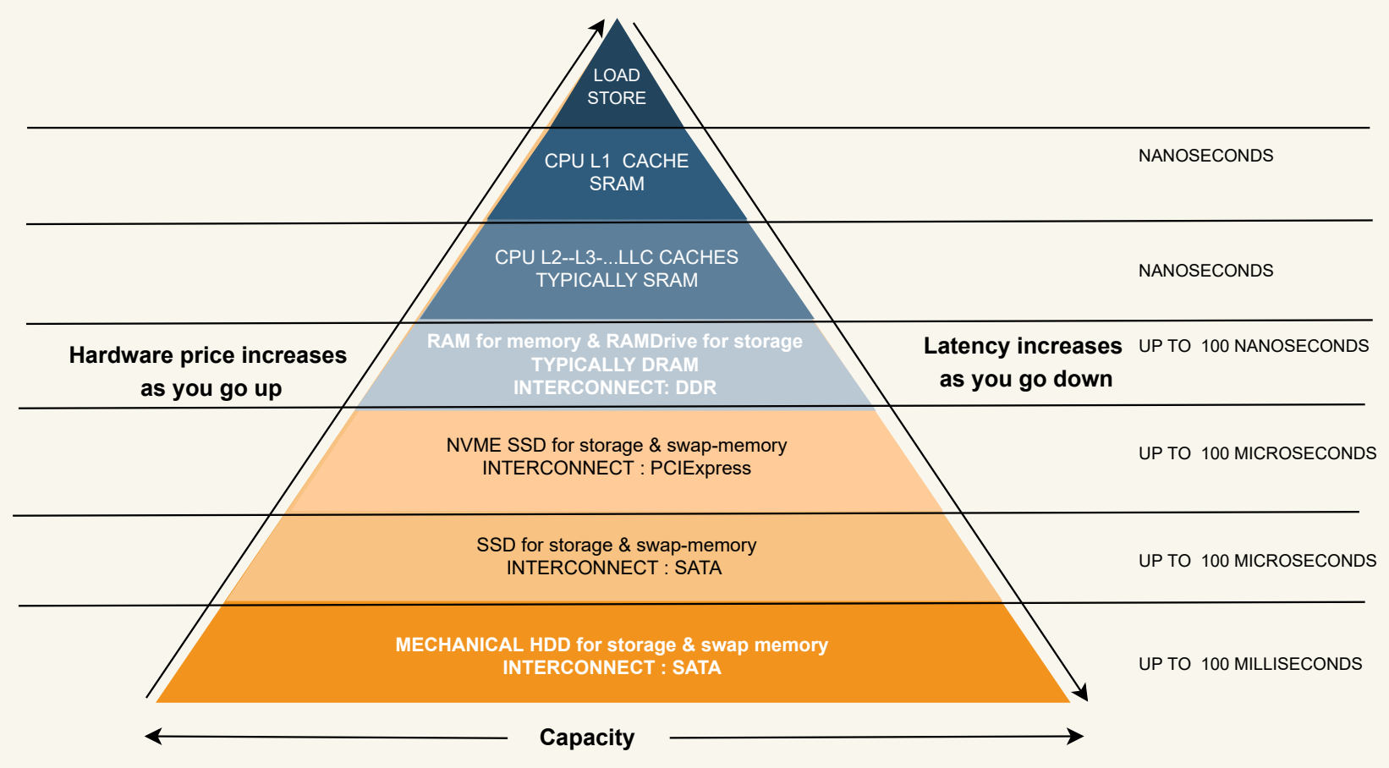
Branch mispredictions are categorised under "bad speculation".

You can use either Intel`s Vtune or Toplev tool to make a top down analysis. Both use Intel CPUs` PMCs (performance monitoring counter).



Reference for image : It is taken from Intel architect Ahmad Yasin`s presentation .

OVERALL MEMORY & STORAGE HIERARCHY


LOAD STORE
CPU L1 CACHE SRAM
CPU L2--L3-...LLC CACHES TYPICALLY SRAM
RAM for memory & RAMDrive for storage TYPICALLY DRAM INTERCONNECT: DDR
NVME SSD for storage & swap-memory INTERCONNECT : PCIExpress
SSD for storage & swap-memory INTERCONNECT : SATA
MECHANICAL HDD for storage & swap memory INTERCONNECT : SATA
NANOSECONDS
NANOSECONDS
UP TO 100 NANOSECONDS
UP TO 100 MICROSECONDS
UP TO 100 MICROSECONDS
UP TO 100 MILLISECONDS
Hardware price increases as you go up
Latency increases as you go down
Capacity

ESTIMATED LATENCY NUMBERS IN CLOCK CYCLES

Simple register operation (ADD OR etc)	less than 1 clock cycle	Floating point division	10-40 clock cycles
Branch prediction	1-2 clock cycles	Compare and swap atomic op.	15-30 clock cycles
Floating point addition	1-3 clock cycles	Integer division	15-40 clock cycles
Multiplication (int, floating point)	1-7 clock cycles	L3 data cache read	30-70 clock cycles
L1 data cache read	3-4 clock cycles	System memory read	100-150 clock cycles
TLB miss	7-21 clock cycles	Cross-NUMA L3 read	100-300 clock cycles
L2 data cache read	10-12 clock cycles	Cross-NUMA system memory read	300-500 clock cycles
Branch misprediction	10-20 clock cycles		

Reference for numbers : http://ithare.com/infographics-operation-costs-in-cpu-clock-cycles/

Note : The reference is a bit dated (2016) though it still is good to show proportions between different event.