Digital microsystems design

Marius Marcu

2021

Objectives

- Specific objectives
  - Advanced topics of microprocessor architectures

Outline

- Instruction level parallelism
- Advanced microarchitectures
- Hyper-threading
- Multi-core
- Cache coherency
- Multi-processor

Instruction level parallelism

- Pipelining achieve executing multiple instructions in parallel
- To increase ILP
  - Deeper pipeline
    - Less work per stage ⇒ shorter clock cycle
  - Multiple resources
    - Replicate pipeline stages ⇒ multiple pipelines
    - Start multiple instructions per clock cycle

Instruction level parallelism

- Pipeline evolution
  - Pipeline

P5 Microarchitecture

  - Super-pipeline

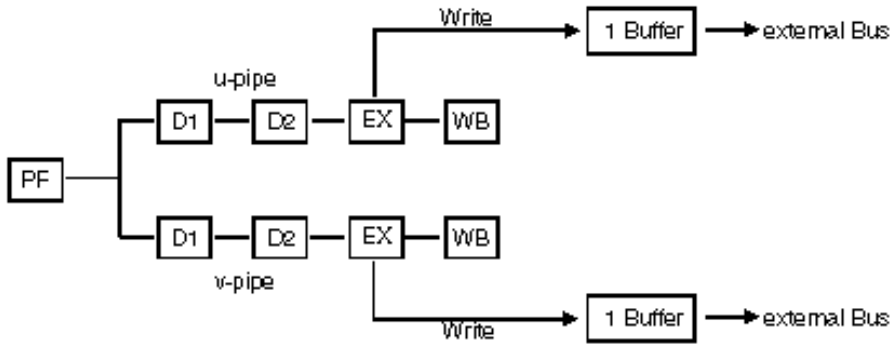
P6 Microarchitecture

  - Hyper-pipeline

NetBurst Microarchitecture

ILP

- Multiple pipelines (Pentium)
  - Prefetch (PF)
  - Decode (D1)
  - Decode (D2)
  - Execute (EX)
  - Writeback (WB)



ILP

- Dynamic execution
  - Sequential execution vs. Speculative execution
    - In-order / Out-of-order
  - Allows processing cores to continue execution in advance of instructions while it waits for preparing operands of previous instructions

ILP

- Example
  - Parallelizable

```
r1 = [r9]
r2 = 17
[r3+4] = r6
```

  - Not parallelizable

```
r1 = [r9]
r2 = r1 + 17
[r2+4] = r6
```

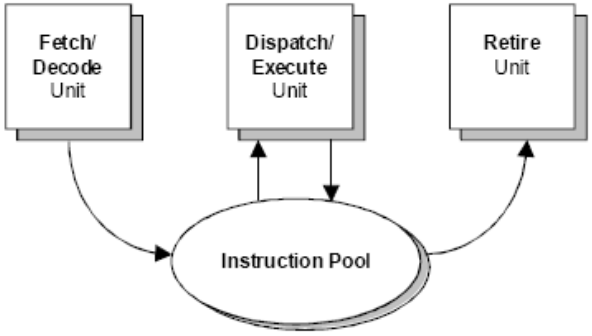
ILP

- Example
  - The third and fourth instructions can be executed before the processor receives data from memory for the first instruction
  - The second instruction has to wait for the result of previous instruction because it depends on its result

```
r1 <= mem [r0]   /* Instruction 1 */
r2 <= r1 + r2   /* Instruction 2 */
r5 <= r5 + 1   /* Instruction 3 */
r6 <= r6 - r3   /* Instruction 4 */
```

ILP

- Dynamic execution
  - Multiple/Deep branch prediction
  - Data flow analysis
    - Data oriented execution
    - Chose the best order of execution
  - Speculative executions
    - Execute instructions in any order if operands are available



Superscalar microarchitectures

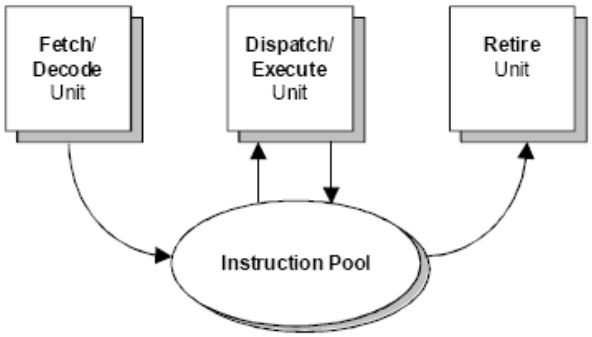
- Speculative execution
  - Example

```
mov eax, [ebx]      ; eax <= mem[ebx]
add esi, eax        ; esi <= esi + eax
inc ecx             ; ecx <= ecx + 1
sub [edi], ecx      ; mem[edi] <= mem[edi] – ecx

; rearranging the code at execution time
mov eax, [ebx]
inc ecx
sub [edi], ecx
add esi, eax
```

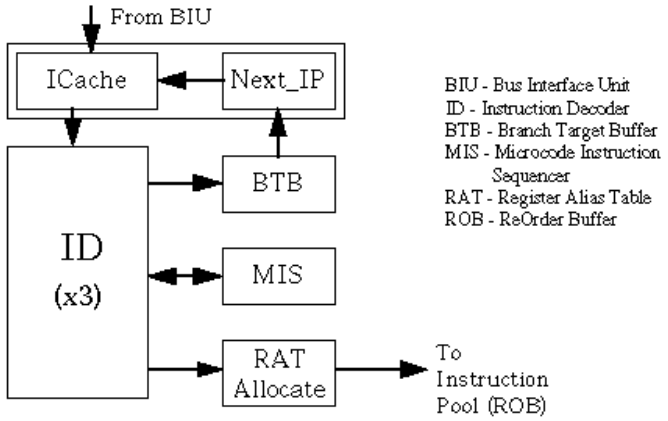
ILP

- Dynamic execution
  - Fetch and decode unit/ Front-end
    - Fetches the instructions
    - Decodes instructions into micro-operations
    - Inserts micro-operations into instruction pool (reorder buffer)
    - In-order execution



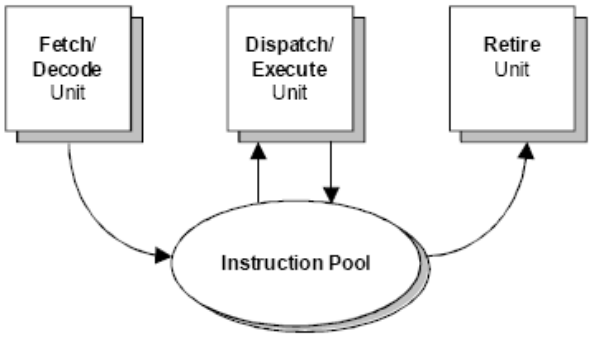
ILP

- Dynamic execution
  - Fetch and decode unit/ Front-end
    - Instruction cache
    - IP
    - BTB
    - MIS
    - n x IDs
    - RAT



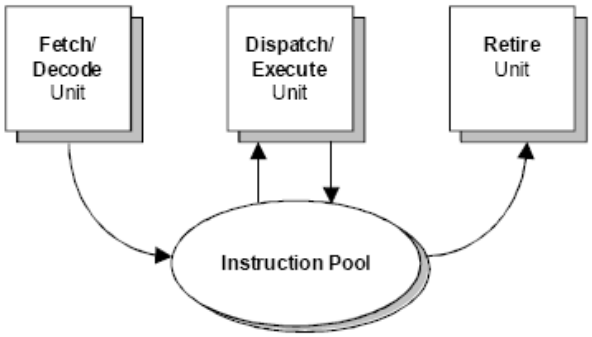
ILP

- Dynamic execution
  - Instruction pool / Reorder buffer
    - Stores temporarily micro-operations until they are executed and retired
    - Keeps micro-operation while waiting for operands to be available
    - Input/output for dispatch/execute unit
    - Input for retire unit



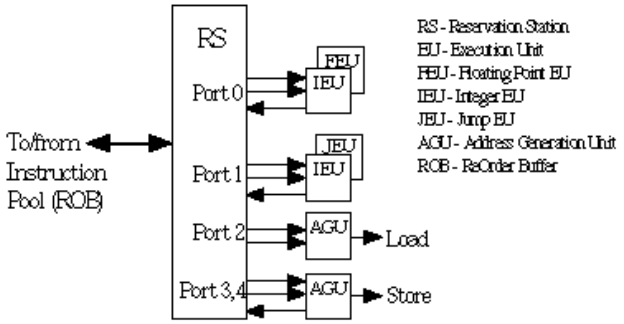
ILP

- Dynamic execution
  - Dispatch/execute unit
    - Speculative execution/ out-of-order
    - Micro-operations having all operands available
    - The result is deposited back into instruction pool



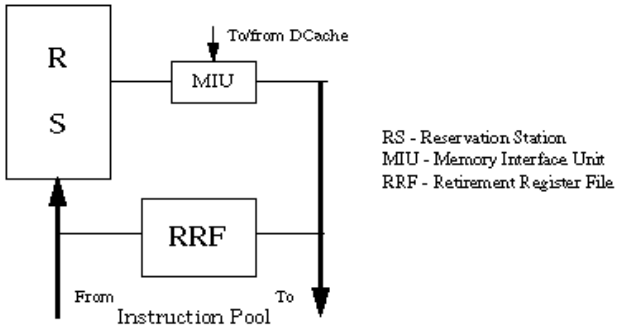
ILP

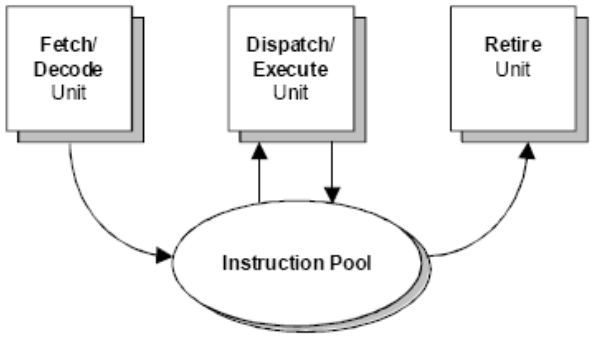
- Dynamic execution
  - Dispatch/execute unit
    - RS
    - n x EUs
      - branch
      - integer
      - floating point
      - MMX/SSE
      - read
      - Write
  - Performance (example)
    - Max 5 micro-ops/clock cycle
    - Avg 3 micro-ops/clock cycle



ILP

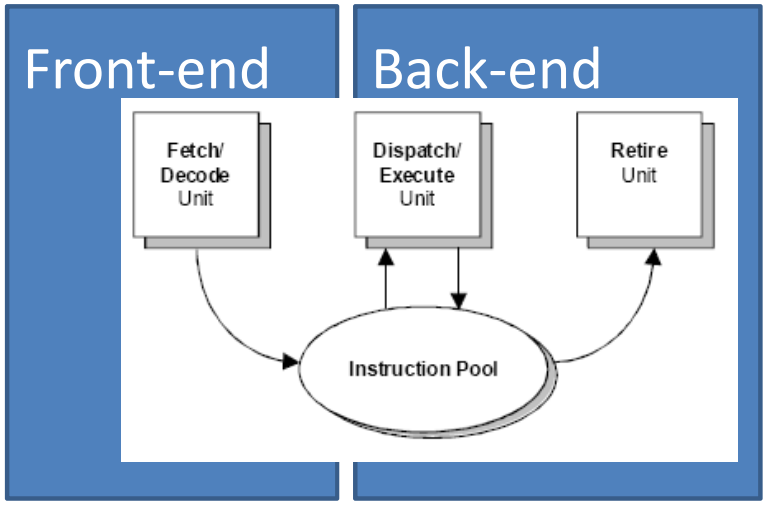
- Dynamic execution
  - Retirement unit / back-end
    - Extract finalized micro-ops
    - In-order
    - Write-back: caches/ registers





Superscalar microarchitectures

- Internal components of superscalar architectures are mapped to the pipeline stages (are used as a pipeline)
- Pipeline stages are grouped into two sections:
  - Front-end – instruction fetch and decoding
  - Back-end – instruction execution and retirement



ILP

- Register renaming
  - Logical x86 registers
    - EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP
    - Limited number of x86 registers (IA32 – 8 registers, AMD64 – 16 registers)
  - Physical registers
    - Register file
    - Larger number of physical registers
    - Logical registers are mapped onto physical registers at runtime, per micro-code

ILP

- Register renaming
  - Is an architectural concept implemented on super-scalar processors in order to avoid serialization of most accessed logical registers

```
// C
b = a+2;
d = c+4;

; eax is a resource          - parallelizable
mov eax, a                          mov eax, a
add eax, 2                          add eax, 2
mov b, eax                          mov b, eax
mov eax, c                          mov ebx, c
add eax, 4                          add ebx,4
mov c, eax                          mov d, ebx
; if multiple registers are used, they can be used in parallel
; however the logical registers are limited as number
```

ILP

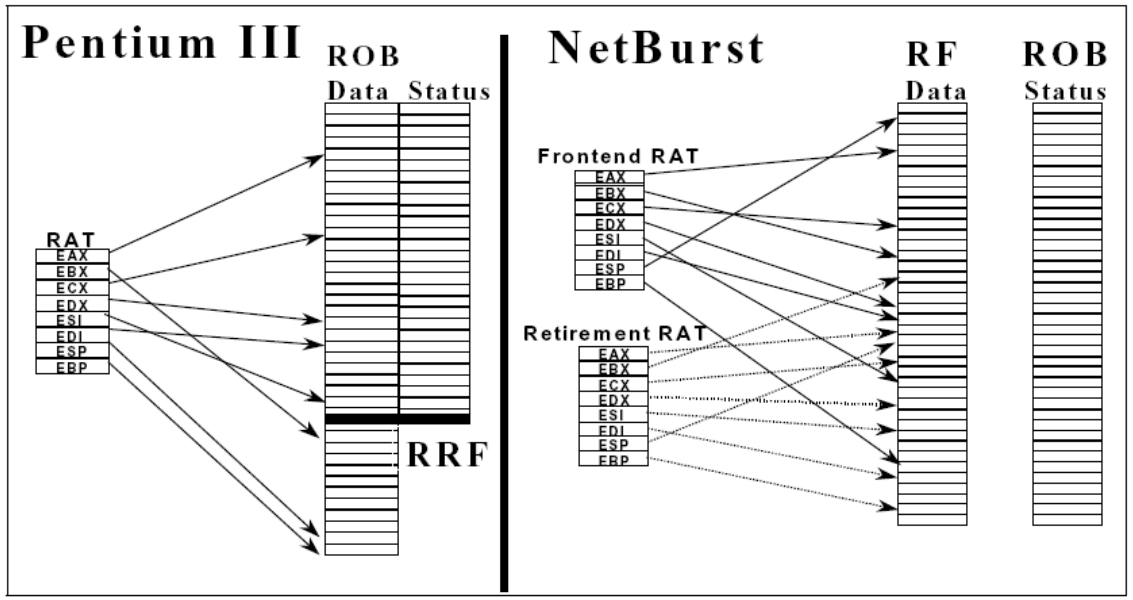
- Register renaming
  - The 8 IA-32 logical register are mapped at runtime to the N physical registers (ex. 40, 128)
  - Many instructions use by default or by convention specific register (e.g. EAX for arithmetic instructions)
    - Therefore these instruction cannot be executed in parallel
    - They have to wait for the shared resource (the register) to be released
  - Using register renaming multiple instances of the same logical register may coexist
    - Support for parallel execution of such instructions

Arhitectura IA-32

- Register renaming
  - Technique to avoid unnecessary serialization when accessing internal registers
  - Each micro-operation knows what physical instance has to use for its operands (logical register)
  - The current mapping is indicated by RAT (Register Alias Table) – mapping table between logical registers (aliases) to physical registers

ILP

- Register renaming
  - RAT – register alias table
    - Maps the IA32 registers onto physical register



TLP

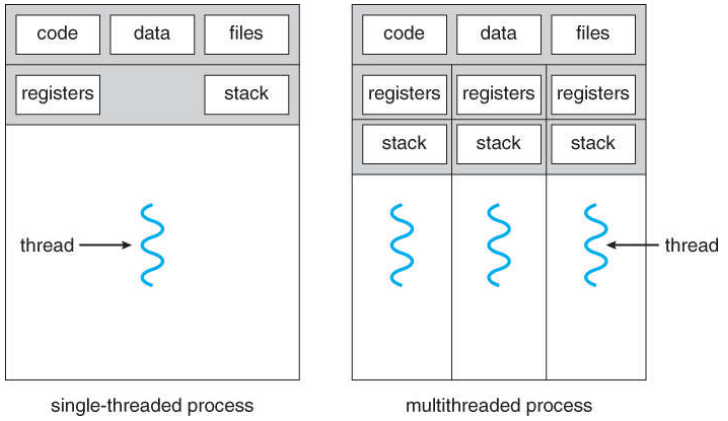
- How you can characterize pipelines' parallelism? - ILP
- How you can characterize applications' parallelism? - TLP

TLP

- ILP vs TLP
  - Micro-operațiile executate în paralel sunt ale instrucțiunilor dintr-un singur fir de execuție
  - Un thread este unitatea de fundamentală de execuție la nivelul sistemului de operare
  - Fiecare program (proces) are cel puțin un fir de execuție (main thread/ UI thread)
  - ILP duce la creșterea performanței execuției instrucțiunilor unui singur fir de execuție

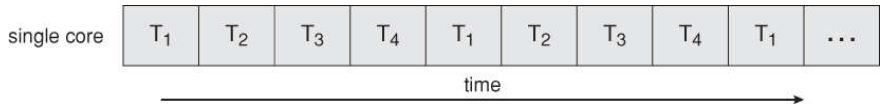
TLP

- ILP vs TLP
  - Un fir de execuție necesită exclusivitate pe
    - Pointer-ul de program (program counter)
    - Stivă
    - Setul de registre
  - Un core logic execută un singur thread la un moment dat

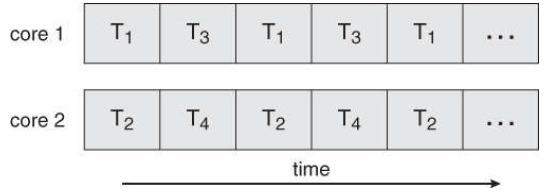


TLP

- ILP vs TLP
  - Multi-threading pe un singur core



  - Execuția concurentă pe mai multe core-uri



Hyper-threading technology

- Instruction Level Parallelism:
  - Execution of multiple micro-operations in parallel
  - ILP
- Application level parallelism:
  - Multi-threading - (TLP – Thread Level Parallelism)
  - Multi-tasking

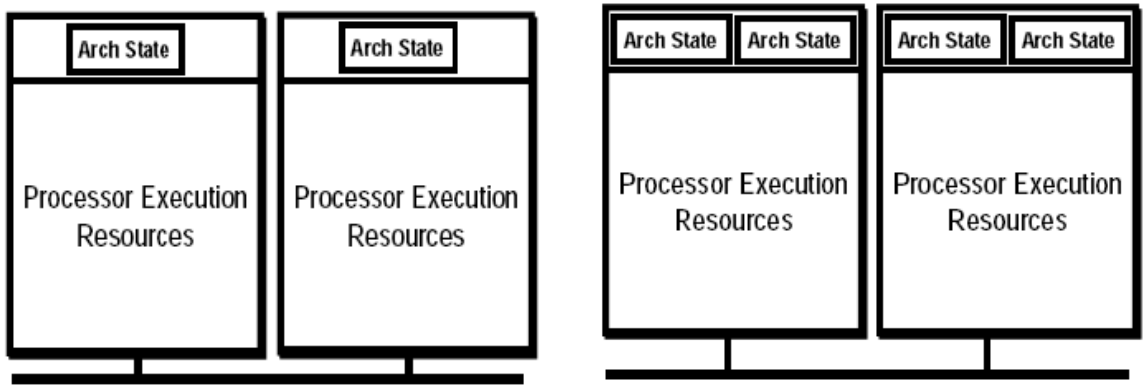
- Applications benefit from TLP – they require execution of multiple threads in parallel

Hyper-threading technology

- Technology proposed by Intel to increase applications' parallelism implementing the SMT (Simultaneous Multi-Threading) concept
- Implemented from Pentium 4 and Xeon processor families
- One physical core can host two logical cores
  - One logical core is allocated for one thread
- OS will identify and use logical cores as they are independent cores
- One physical core with HT can execute instructions from two threads simultaneously
- One logical core can execute instructions from one thread at a time

Hyper-threading technology

- For multi-core all core resources are fully replicated
  - Physical cores
- For HT only parts of the processor resources are duplicated
  - Logical cores
  - Most of the resources are used in common (execution units)
  - Duplicated resources are the ones that keep architectural state of IA-32 (register set)



Hyper-threading technology

- Replicated resources:
  - Register set
  - IP – Instruction Pointer
  - RAT – Register Alias Table (register renaming)
  - Branch prediction

Hyper-threading technology

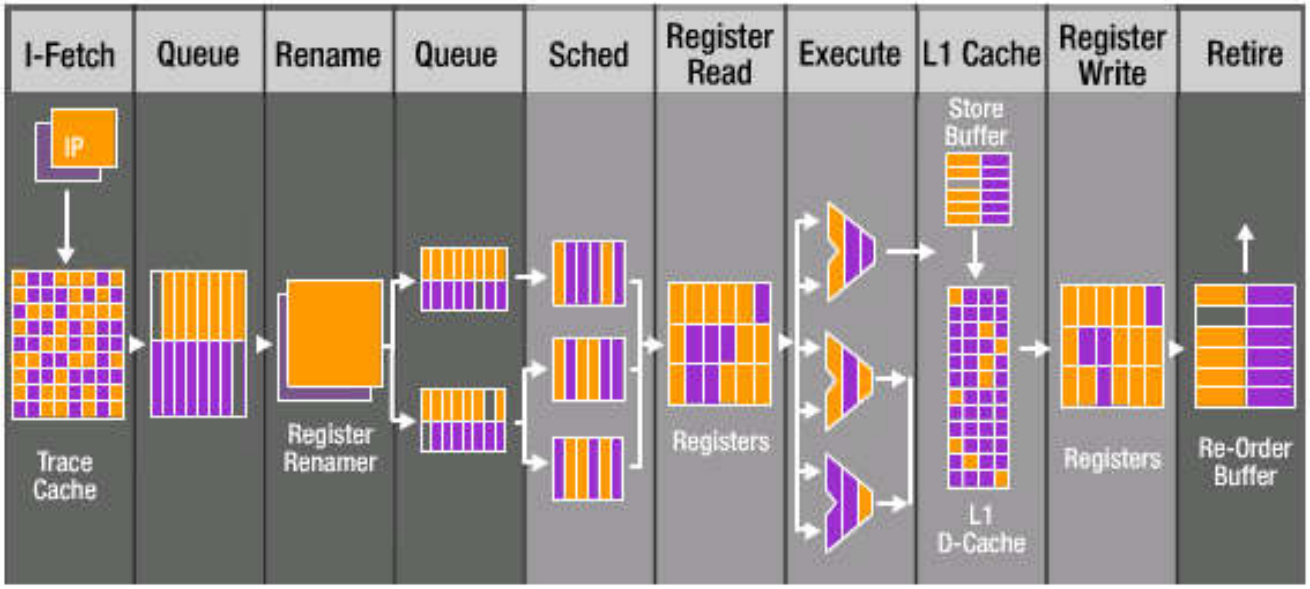
- Partitioned resources
  - Some of the processor resources are allocated equally by statically to each thread
    - Buffers between pipeline stages
  - Disadvantage
    - If the two threads executed are not balanced in using resources, some of the resources are not used (the cannot be used only by the thread that owns them)

Hyper-threading technology

- Shared resources
  - Most of the processor resources are allocated dynamically at runtime to the threads in execution as needed
    - Cache
    - Execution units

Hyper-threading technology

- Example
  - Two threads executed by two logical cores



Hyper-threading technology

- Example of HT inefficiency
  - One highly computational thread run by a logical core
    - Some of the partitioned resources are not used
    - The overall thread performance is lower compared to its execution on the physical core without HT, when all partitioned resources are allocated to the thread

Multi-Core

- Motivation
  - It is hard to further increase the operating frequency
    - Clock stability
    - Perturbations
  - It is hard to further increase the depth of the pipeline – number of stages (Hyper-pipelines)
    - Increased energy consumption
    - High temperatures
  - Low efficiency of high frequencies and deep pipelines

Multi-Core

- Motivation
  - It is difficult to increase the single-core performance and double the processing power:
    - From 3-6 instructions/ clock cycle to 6-12 instructions/ clock cycle
      - 3-4 memory accesses per cycle
      - More than 20 registers accesses per cycle
      - 12-24 instruction fetches per cycle

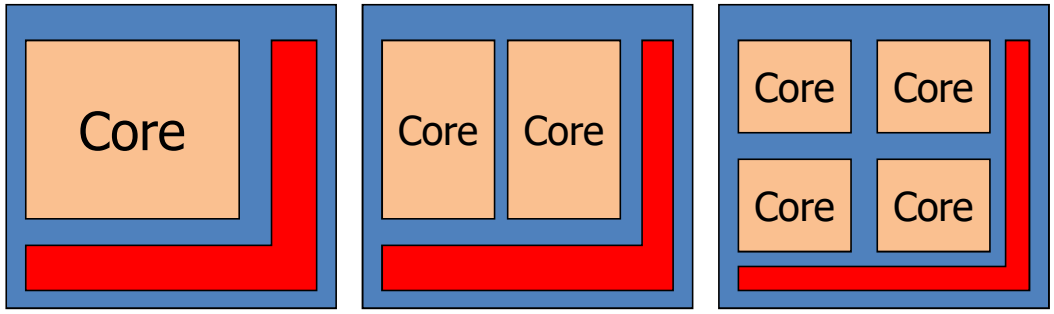
Multi-Core

- Motivation
  - OS multitasking – each application implements at least one thread – the main thread
  - Applications are implementing more than one thread
  - Parallel algorithms

Multi-Core

- Definition
  - Multi-core architectures replicate multiple processors on a single die
  - Each processor is called physical core
- CMP – Core Multi Processing

Multi-Core

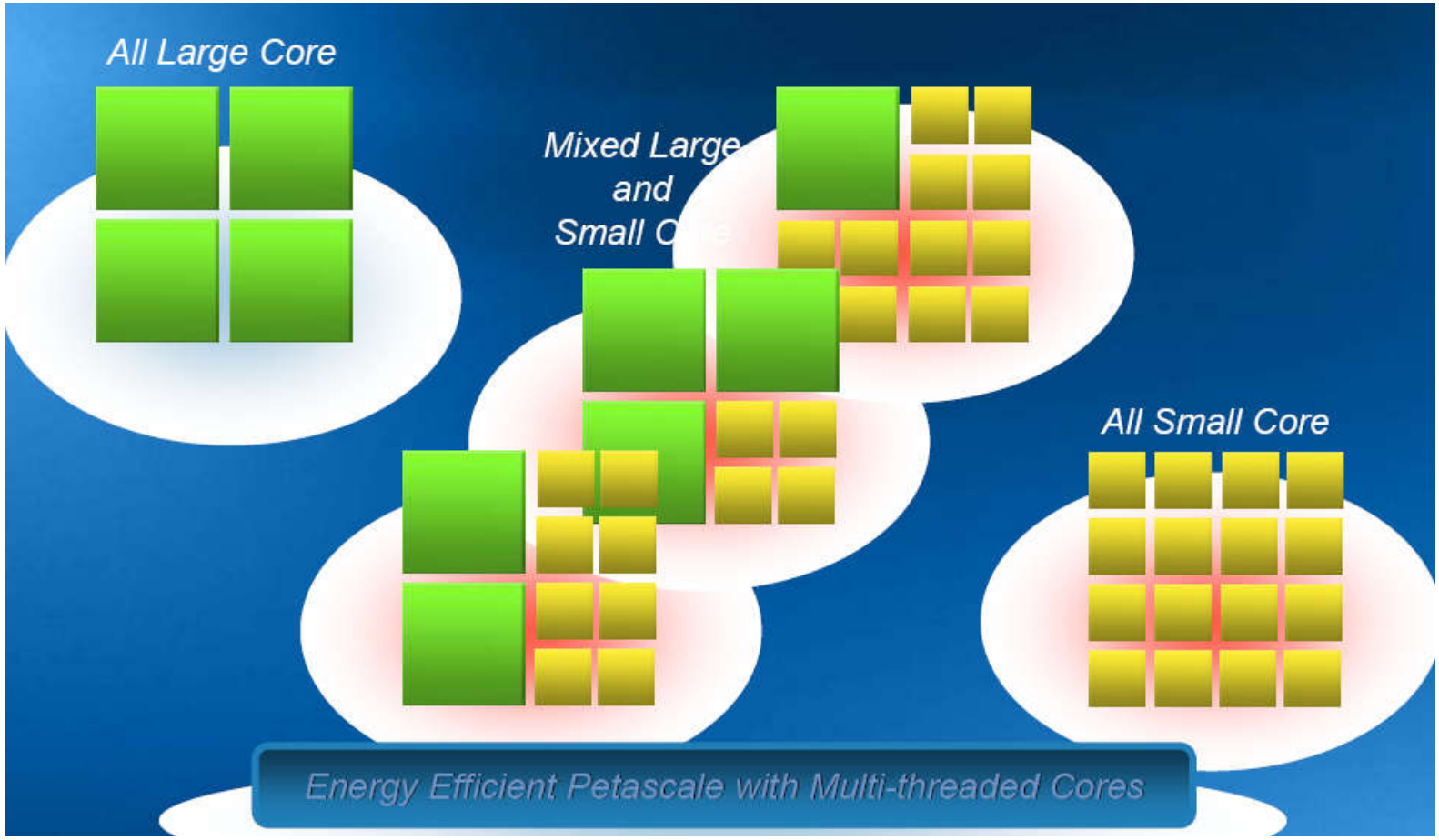


	Single Core	Dual Core	Quad Core
Core area	**A**	**~A/2**	**~A/4**
Core power	**W**	**~W/2**	**~W/4**
Chip power	**W + O**	**W + O'**	**W + O"**
Core performance	**P**	**0.9P**	**0.8P**
Chip performance	**P**	**1.8P**	**3.2P**

Multi-Core

- UnCore – terminology used by Intel to describe functions of a microprocessor that are not part of the physical processing cores
  - Memory controller
  - QPI controller
  - LLC (Last Level Cache – ex. L3 cache)

Multi-Core


All Large Core
Mixed Large and Small C
All Small Core
Energy Efficient Petascale with Multi-threaded Cores

Multi-Core

- Big-little processing cores architectures
  - Heterogeneous architectures including a small number of complex cores and a large number of simple cores
  - Big cores – complex processing architectures providing an extended instruction set, small number of instances
  - Little cores – simple processing architectures, providing a reduced instruction set, large number of instances

Multi-Core

- The main architectural model used by today processors
- The trend is to use this model for next processors
- Combines two or more replicated physical cores in the same circuit

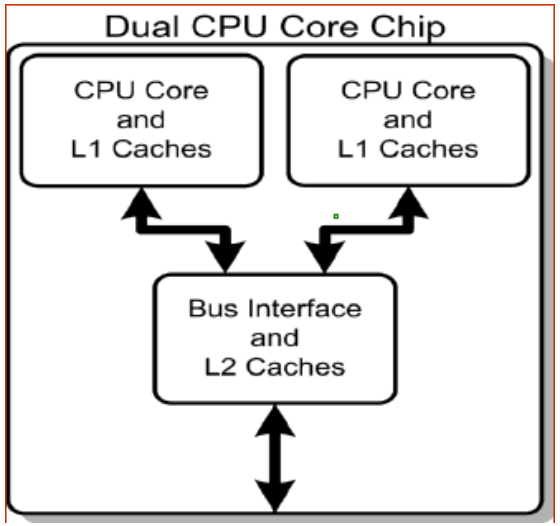
Multi-Core

- **Common ways to improve the performance of a CPU on the hardware level**

Technique	Advantages	Disadvantages
Frequency scaling	Nearly no design overhead, immediate scaling	Some manufacturing process overhead, leakage problems
Architectural changes	Increased versatility, performance	Huge design and manufacturing overhead, minor (20%) speedups
Simultaneous multi-threading	Medium design overhead, up to 30% performance improvement	Requires more memory, single thread performance hit (~10-15%)
Cloning chips (MCP)	Minimal design overhead	Requires parallel software & more memory, inter-chip communication difficult
Adding processing cores	Small design overhead, easy to scale, 50%+ performance improvement	Requires parallel software or more memory

Multi-Core

- CPU-local Level 1 caches,
- Shared, on-chip Level 2 caches
- Cache coherency!



Cache coherency

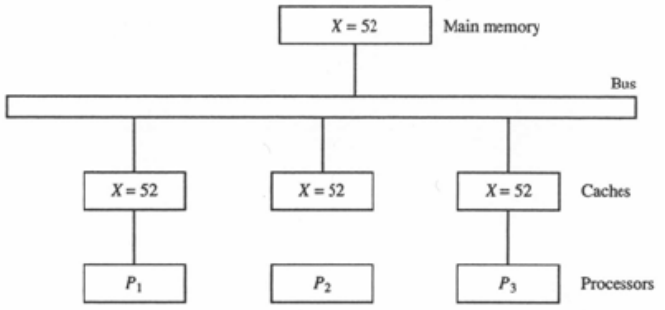
- How multi-core processors have impacted the memory hierarchy?

Cache coherency

- The value returned by a LOAD operation will give the value of the last STORE operation to the same location.

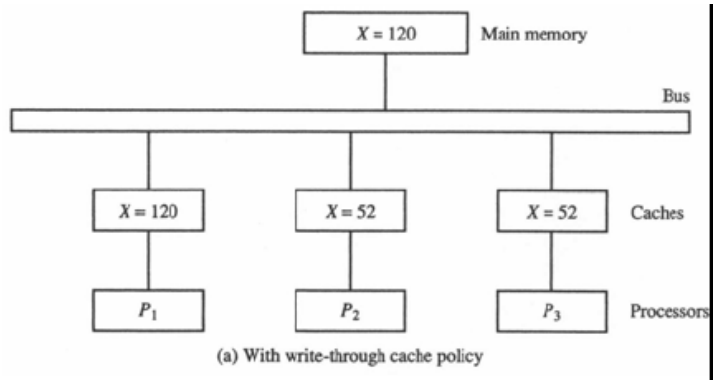
Cache coherency

- LOAD X
  - Two or more cores can read the same location



Cache coherency

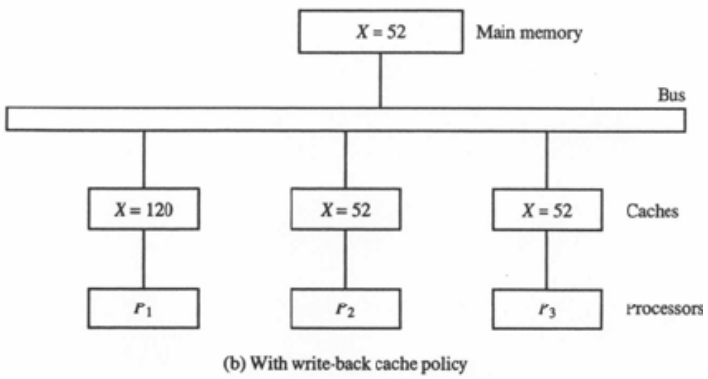
- Write one location (write-through)
  - Local caches are not updated automatically



(a) With write-through cache policy

Cache coherency

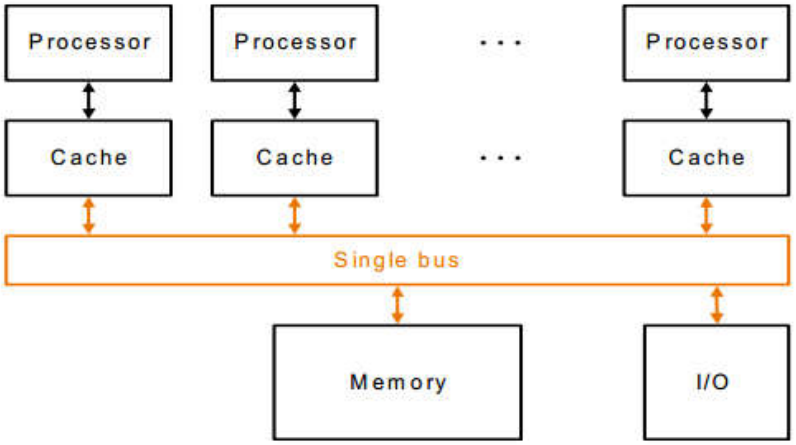
- Write one location (write-back)
  - Main memory and local caches are not updated



(b) With write-back cache policy

Cache coherency

- Different caches may contain different value for the same memory location.



Time	Event	Cache Contents for CPU A	Cache Contents for CPU B	Memory Contents for location X
0				1
1	CPU A Reads X	X = 1		1
2	CPU B reads X	X = 1	X = 1	1
3	CPU A stores 0 into X	X = 0	X = 1	0 if write through 1 if write back

Cache coherency

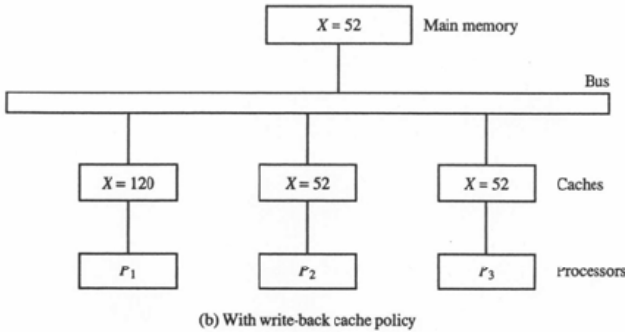
- Solutions
  - Software
  - Hardware
  - Heterogeneous

Cache coherency

- Solutions
  - Use local caches only for private data
  - Use local caches by shared data only for read
  - Cache line status: RO, RW
  - Cache coherency protocols and controllers
    - Snooping controller
    - Monitors cache writes
    - Invalidates shared locations if changed

Cache coherency

- Cache coherency protocols:
  - Write – invalidate: If a processor changes a cache line the rest of local caches will invalidate the line if they have it. Further reads will generate cache miss and load from main memory
  - Write – update: If a processor updates a location, the information is sent to all local caches to update the location



(b) With write-back cache policy

Cache coherency

- MESI protocol
  - Modified – Exclusive – Shared – Invalid
  - Strategy: write – invalidate

Cache coherency

- MESI protocol
  - Modified (M) – cache line is valid and exclusively used by local processor, it has been updated having the most recent value
  - Exclusive (E) – cache line is valid and exclusively used by local processor, its value is not changed
  - Shared (S) – cache line is valid but is shared with other processors
  - Invalid (I) – cache line is not valid

Cache coherency

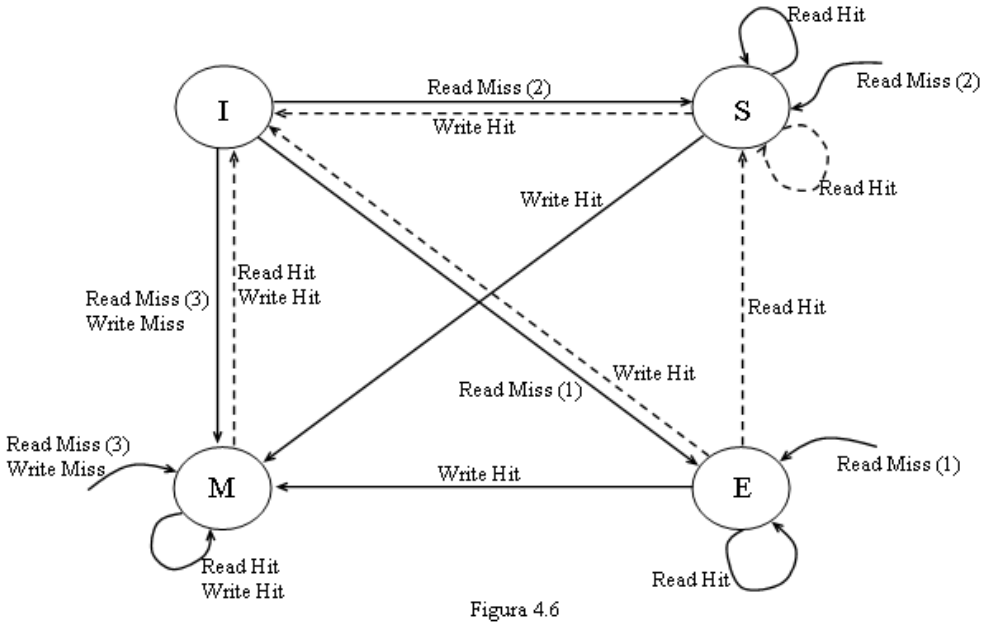
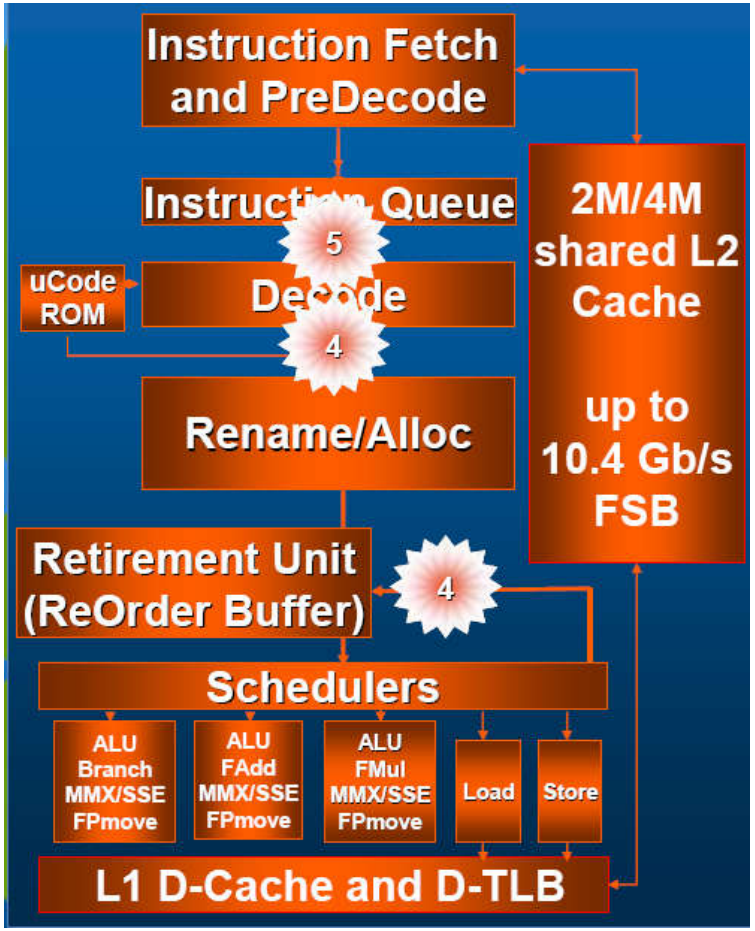


Figura 4.6

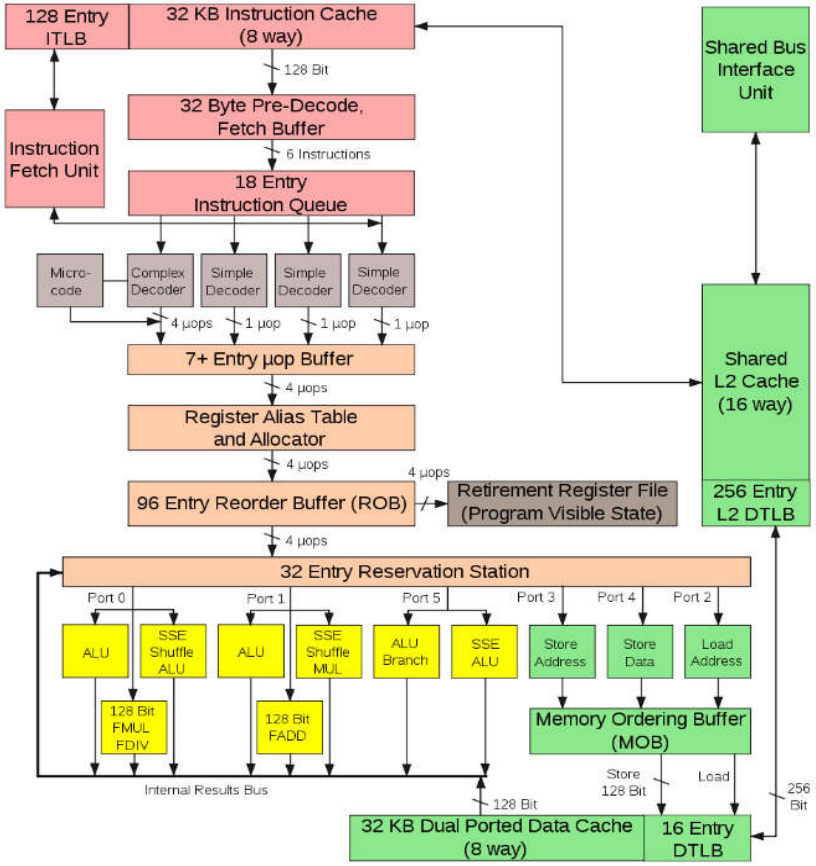
Read Miss (1) – nici un alt cache nu are copia acestei linii;
Read Miss (2) – linia cerută se găseşte în alte cache-uri în stările E sau S;
Read Miss (3) – linia cerută se găseşte în alte cache-uri în stare M.

Intel core microarchitecture



Intel core microarchitecture

- 14-stage Pipeline
- 4 wide decode
- 4 wide Retire
- Macro-fusion
- Enhanced ALUs
- Deeper Buffers

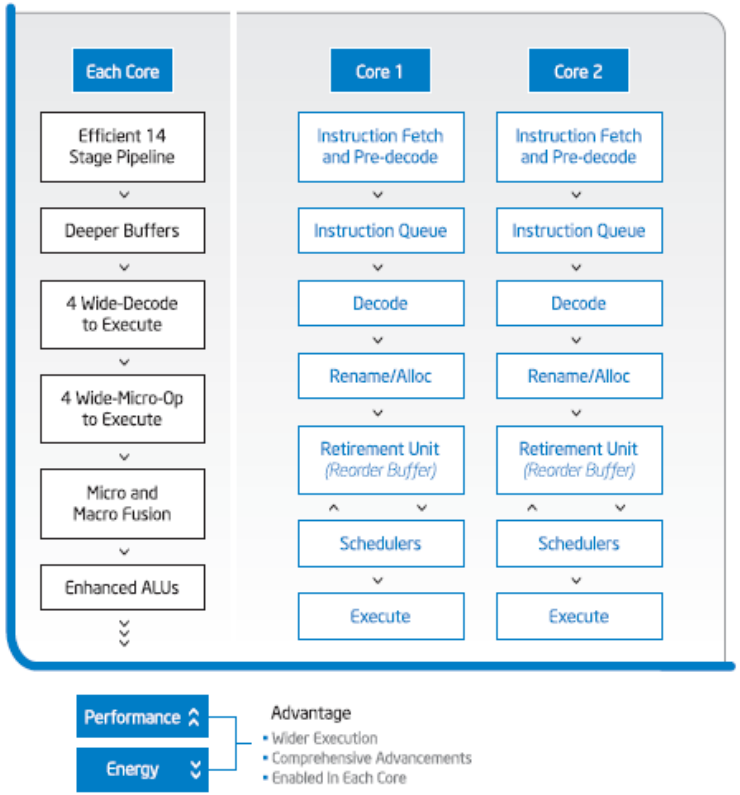


Intel core microarchitecture

- Intel core microarchitecture innovations
  - Wide Dynamic Execution
  - Smart Memory Access
  - Advanced Smart Cache
  - Intelligent Power Capability
  - Advanced Digital Media Boost

Intel core microarchitecture

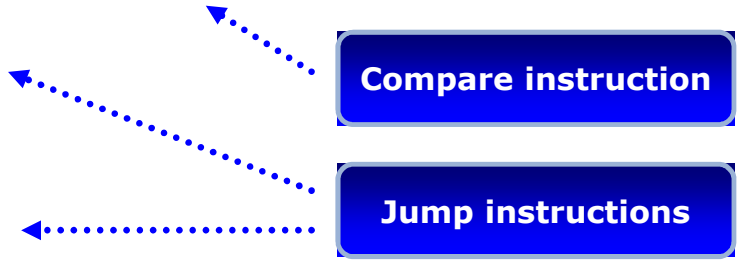
- Wide Dynamic Execution:
  - Each core can fetch, schedule, execute and retire till 4 instructions in parallel
  - Macro-fusion – permits combining multiple distinct instructions into one micro-operation
  - Micro-fusion – permits combining multiple micro-operations of a single instruction



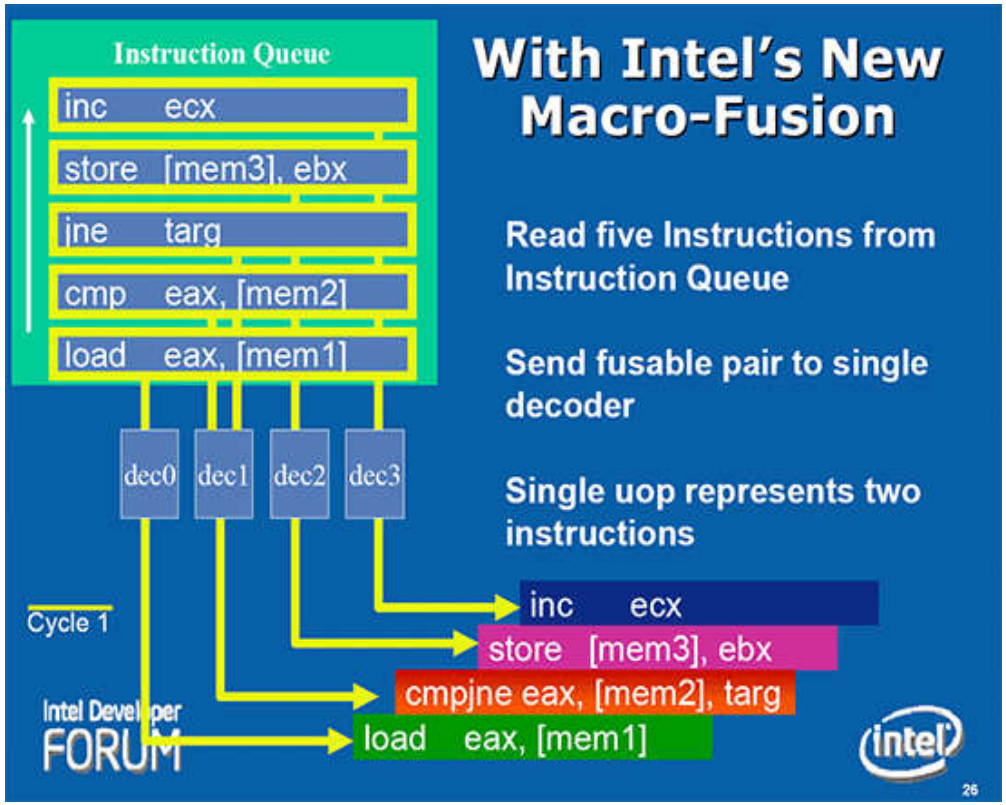
Intel core microarchitecture

- Wide Dynamic Execution
  - MacroFusion

```
If (myVariable == myConstant)
        doThis();
Else
        doThat();
```



Intel core microarchitecture



Intel core microarchitecture

- Wide Dynamic Execution
  - Micro-op Fusion
  - Example:

    Load the contents of [mem] into a register (MOV EBX, [mem])
    An ALU operation, ADD the two registers together (ADD EBX, EAX)
    Store the result back to memory (MOV [mem], EBX)

  - The micro-ops which are derived from the same macro-op are fused to reduce the number of micro-ops that need to be executed.
  - Better scheduling
  - Reduces the number of micro-ops which are handled by the out-of-order logic.

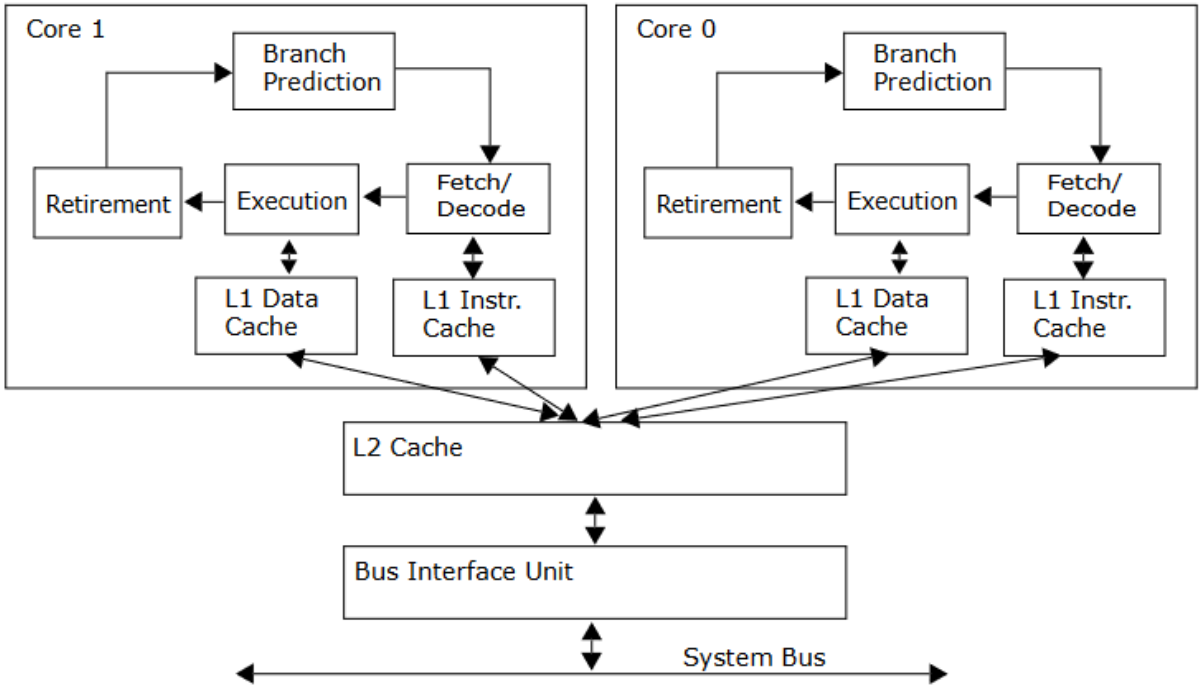
Intel core microarchitecture

- Smart Memory Access
  - Memory disambiguation
    - aggressive memory dependence speculation based on a load's- EIP-address-indexed hash table
    - watchdog mechanism
  - Instruction Pointer Based Prefetcher
    - L1 DCache:2 IP prefetchers/core
    - L1 ICache:1 traditional prefetcher
    - L2 Cache: 2 IP prefetchers;

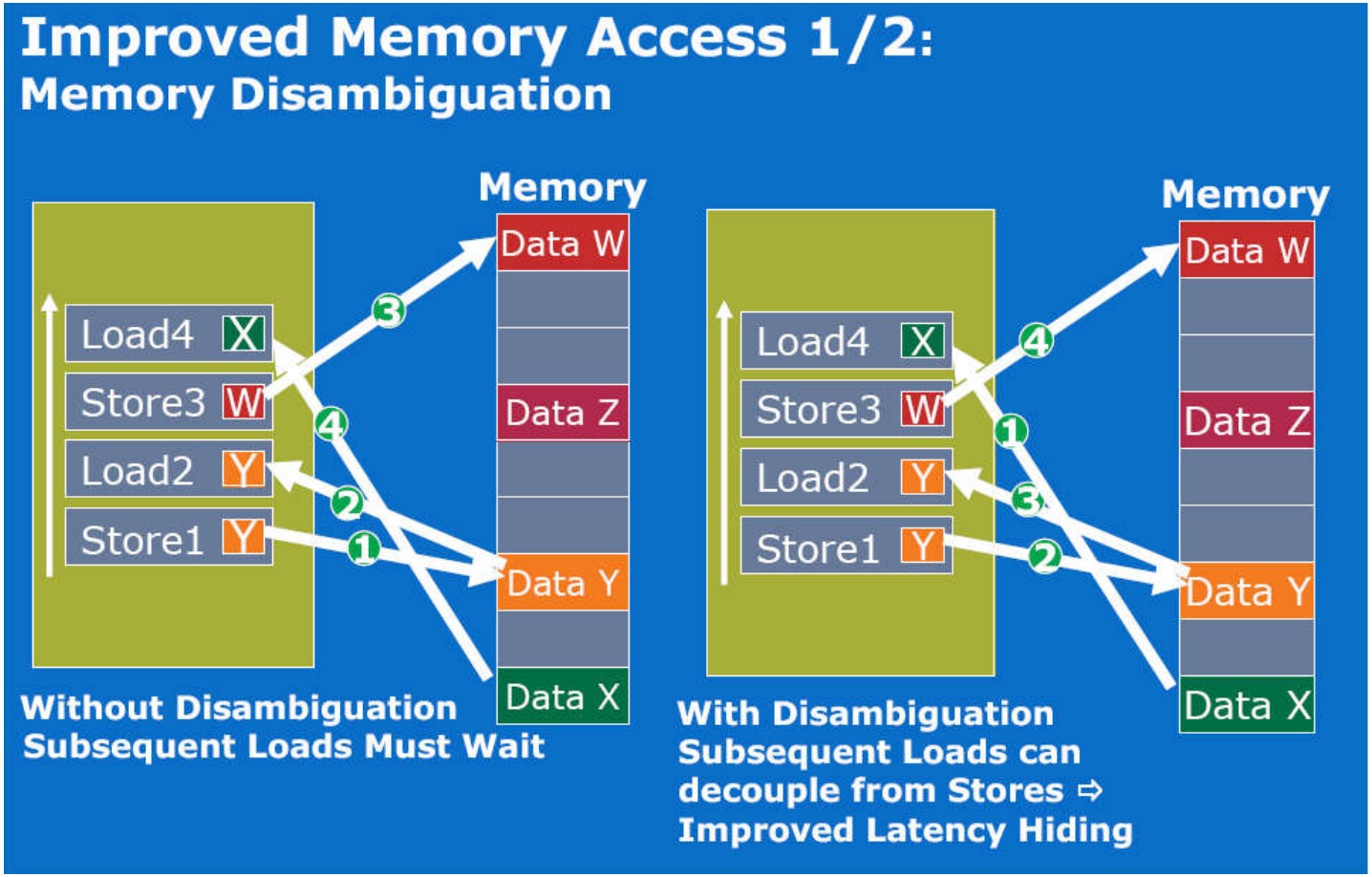
    - predict what memory address will be used and deliver in time
    - record every load's history using Instruction Pointer
    - IP history array
    - parameters for prefetch traffic control fine-tuned for different platforms
    - prefetch monitor

Intel core microarchitecture

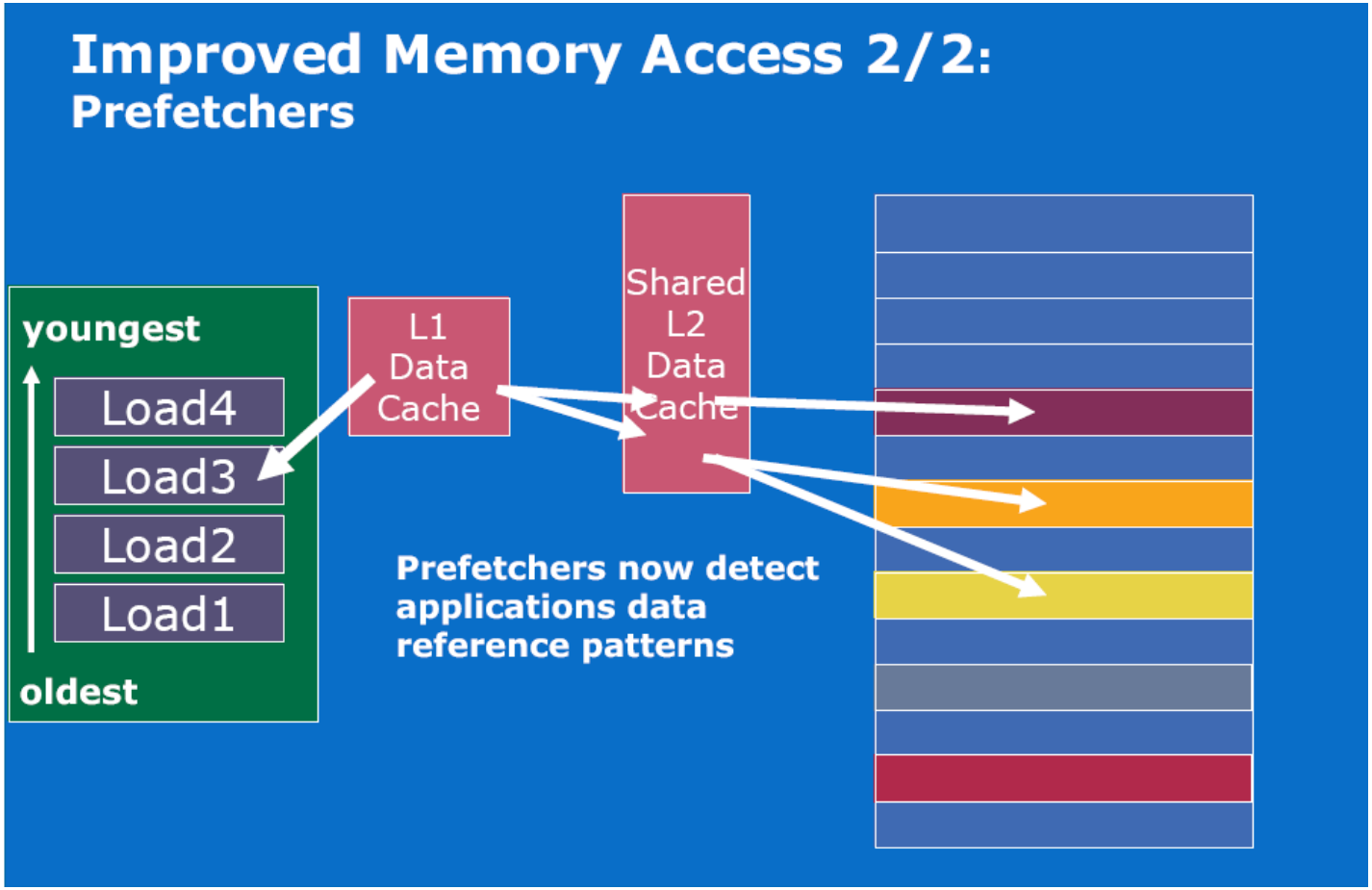
- Smart cache architecture



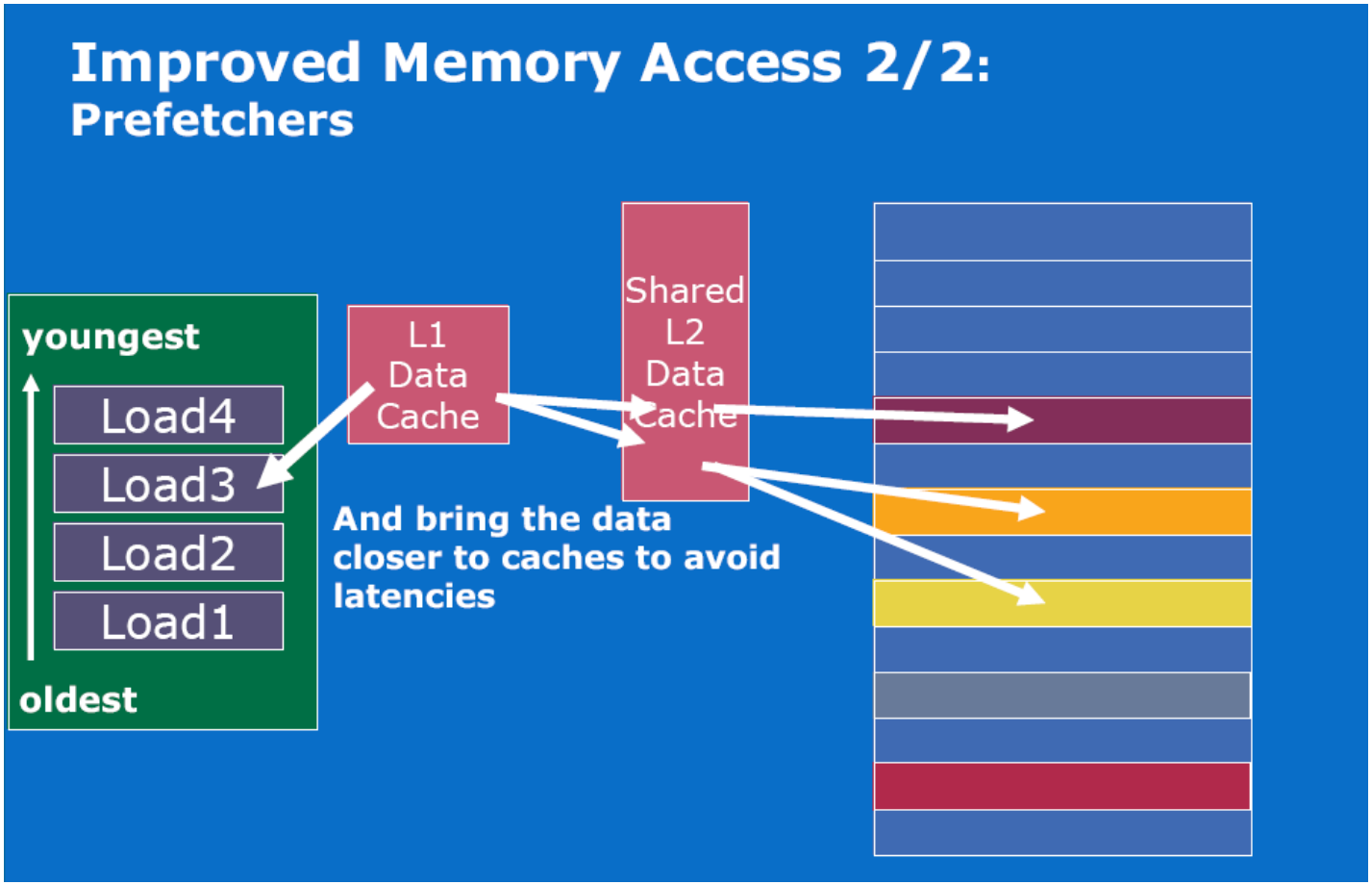
Intel core microarchitecture



Intel core microarchitecture



Intel core microarchitecture



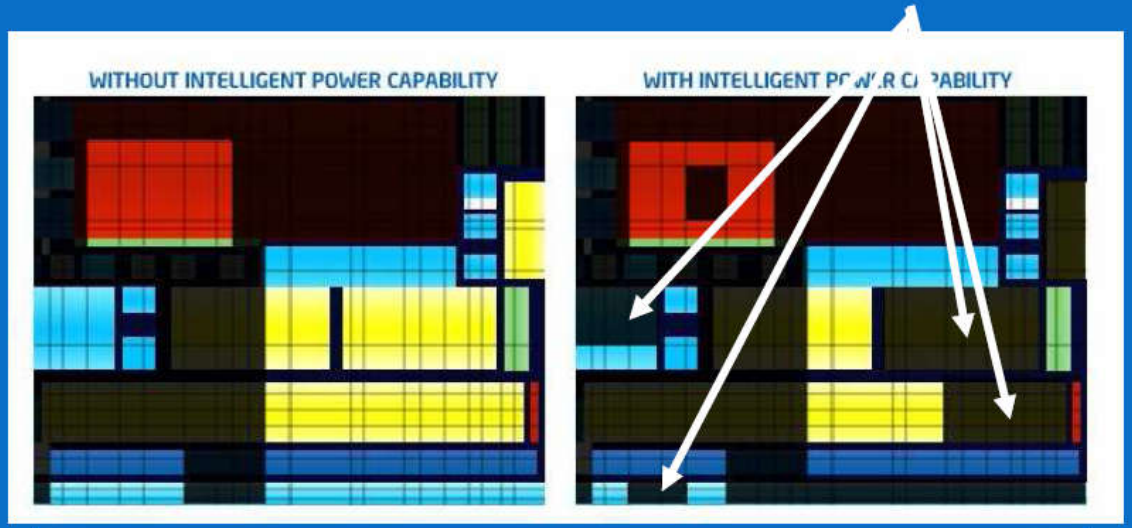
Intel core microarchitecture

- Intelligent Power Capability
  - A set of capabilities designed to reduce power consumption and design requirements.
  - This feature manages the runtime power consumption of all the processor's execution cores and allocates energy to the part which needs energy.

Intel core microarchitecture

Improved Power Saving

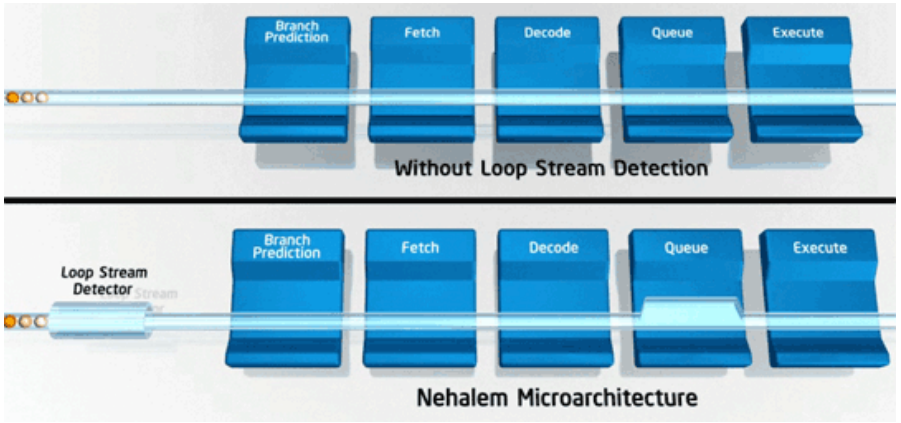
- Ultra fined grained power control
- Turn on only required logic systems
- Even during performance execution, parts can be **shut off.**



Improved Power Saving delivers more performance per watt

Intel core microarchitecture

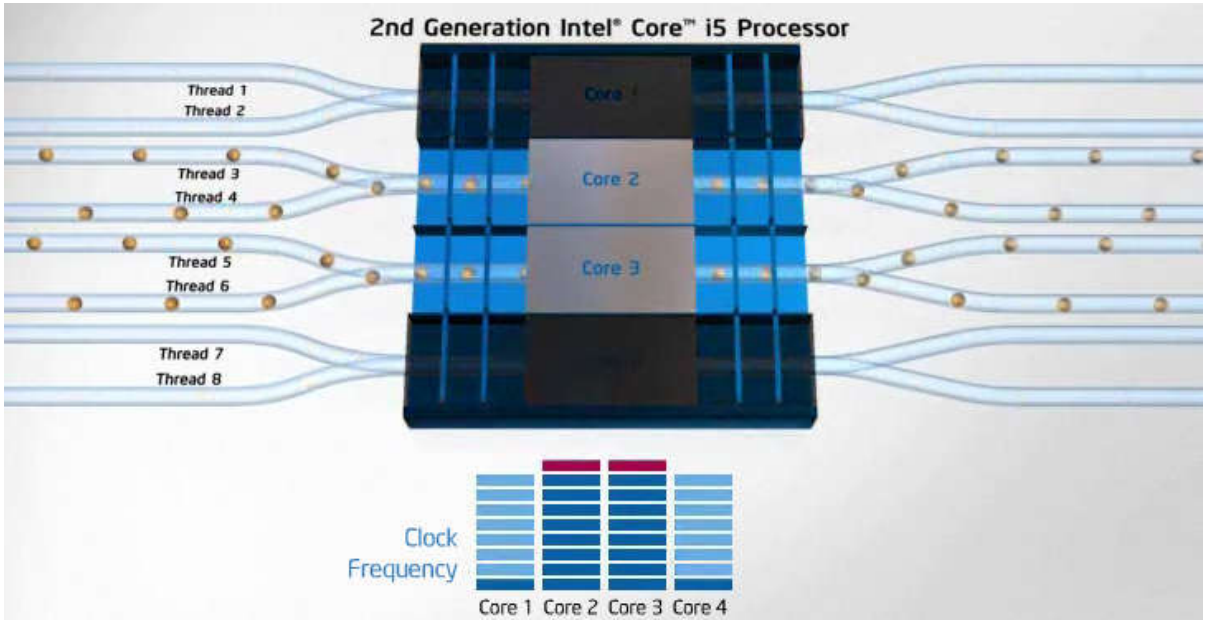
- Loop stream detector
  - Cache uops and execute small loops without decoding



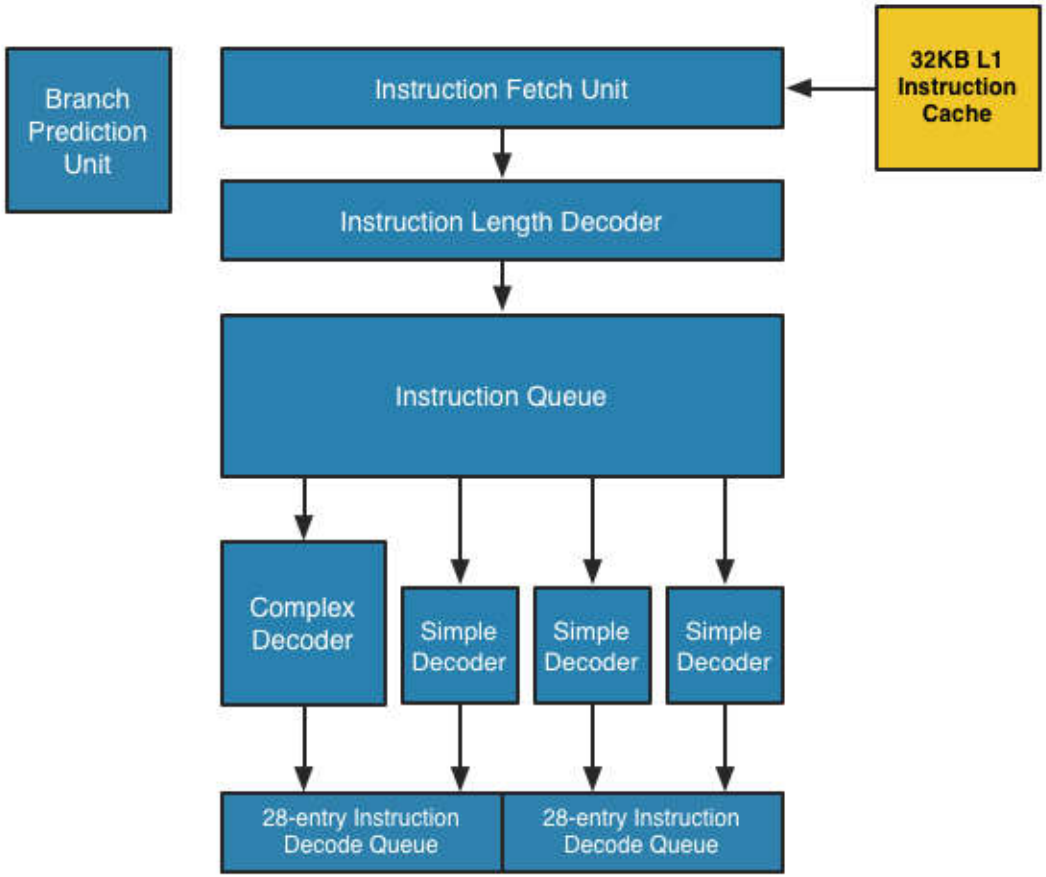
Intel core microarchitecture

- Turbo Boost Technology
  - Overclocking one core when needed and when not all cores are active
  - Is activated when the Operating System (OS) requests the highest processor performance state (P0)
  - The maximum frequency is dependent on the number of active cores
  - The amount of time the processor spends in the Turbo Boost state depends on the workload and operating environment

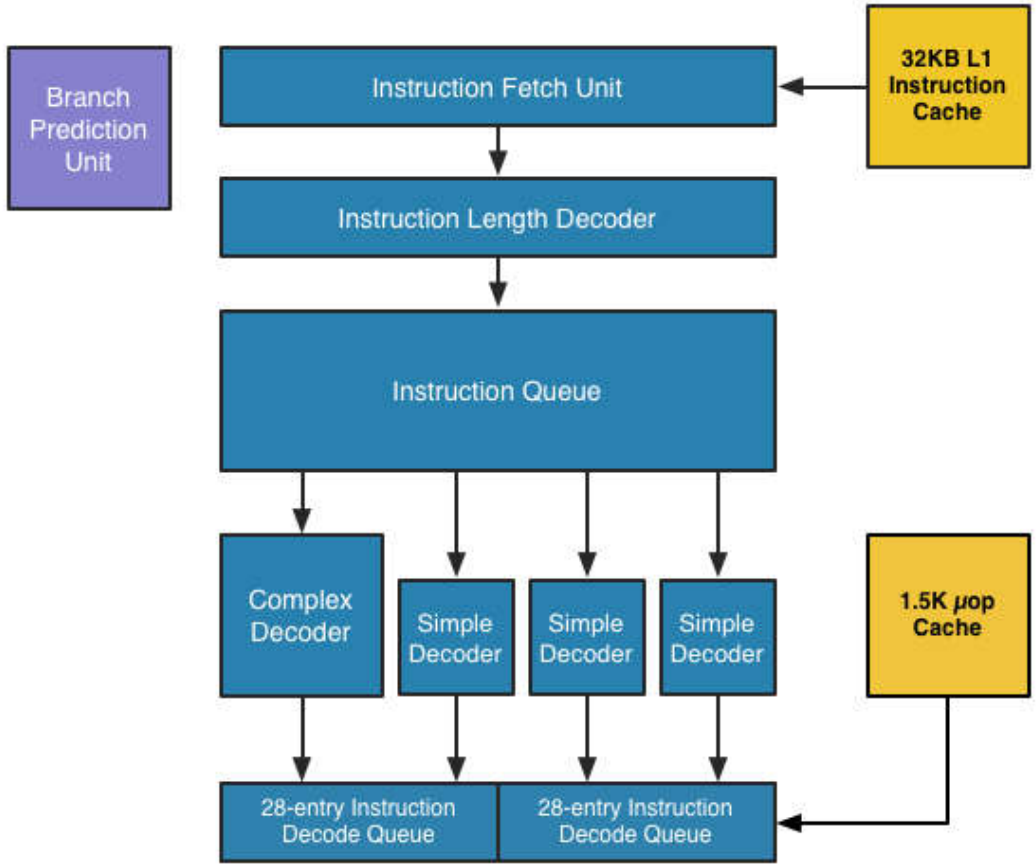
Intel core microarchitecture



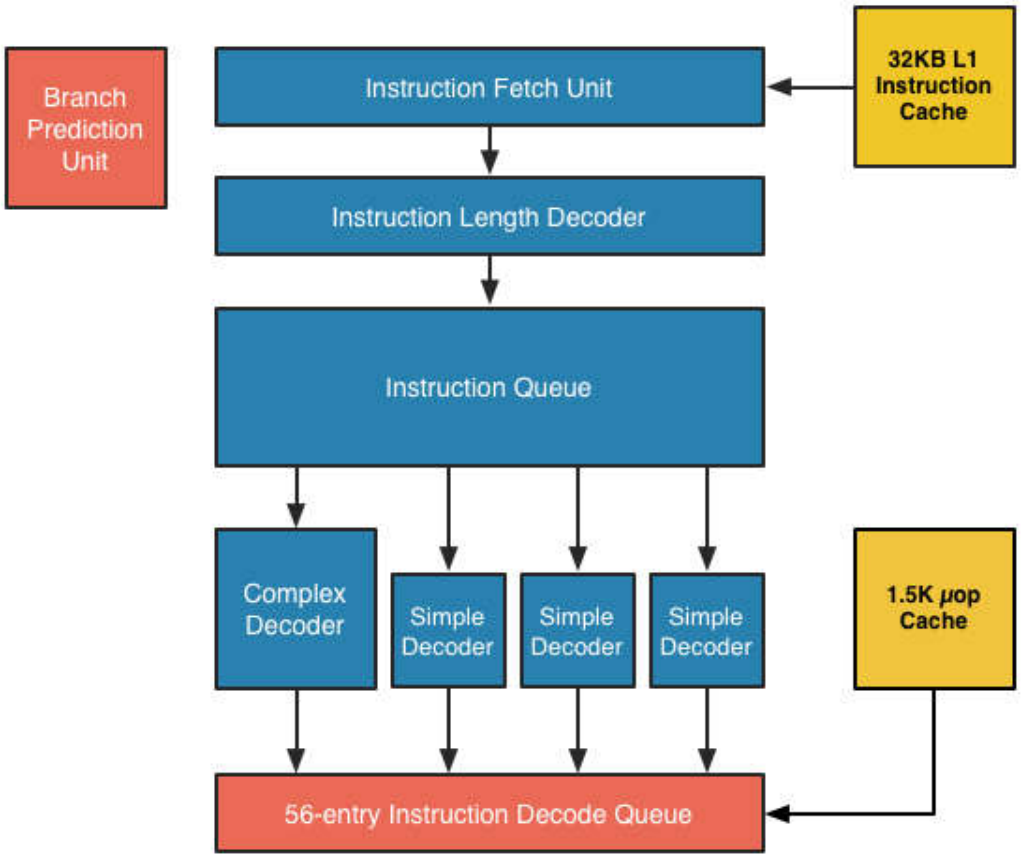
Intel Nehalem Front End



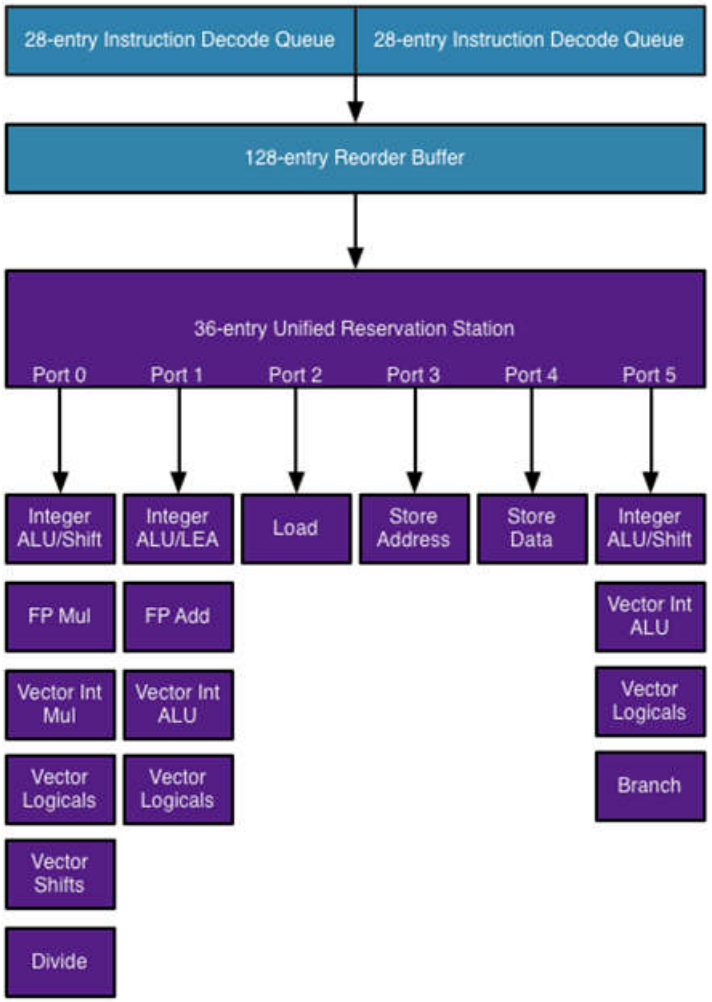
Intel Sandy Bridge Front End



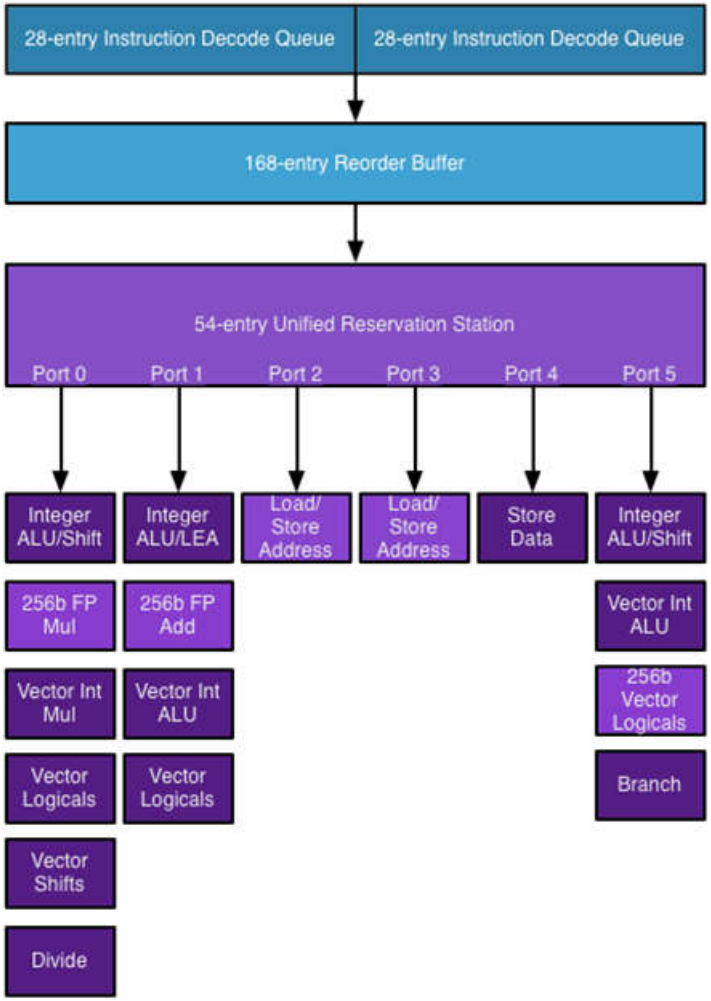
Intel Haswell Front End



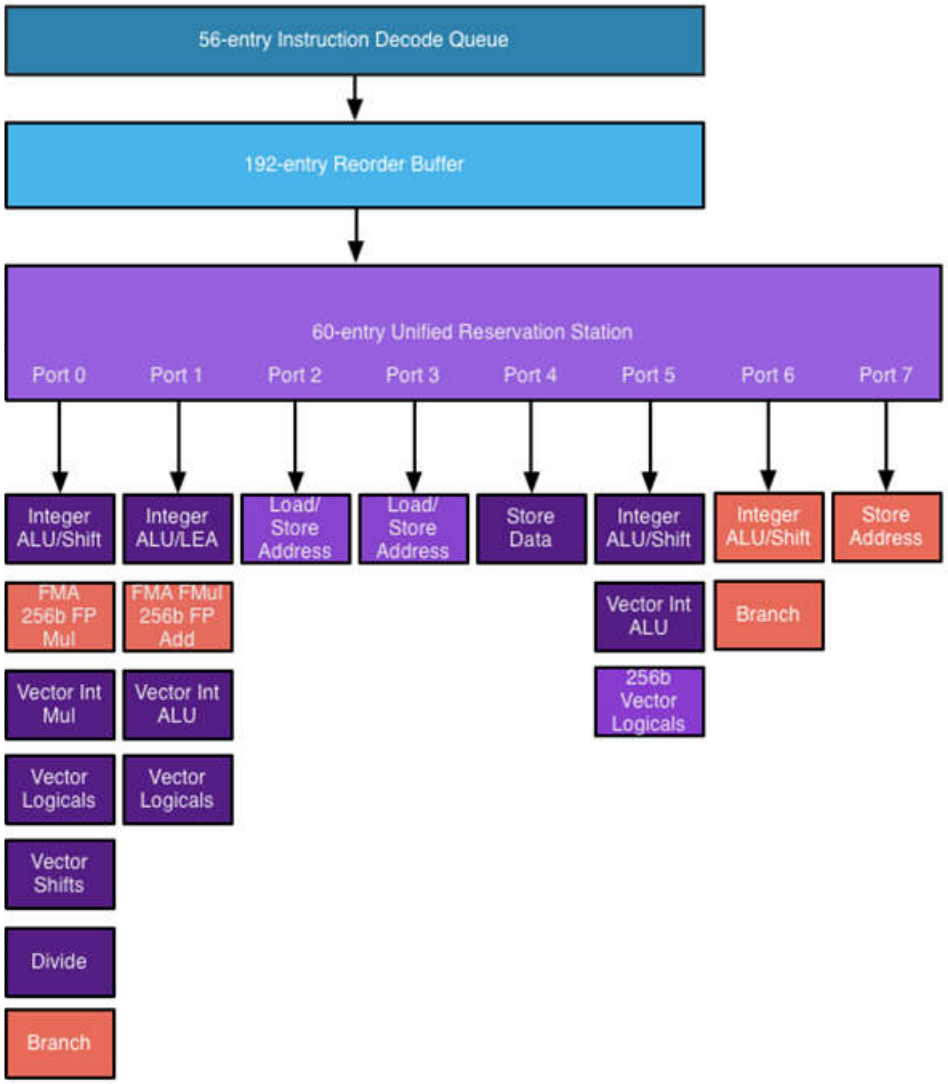
Intel Nehalem Execution Engine



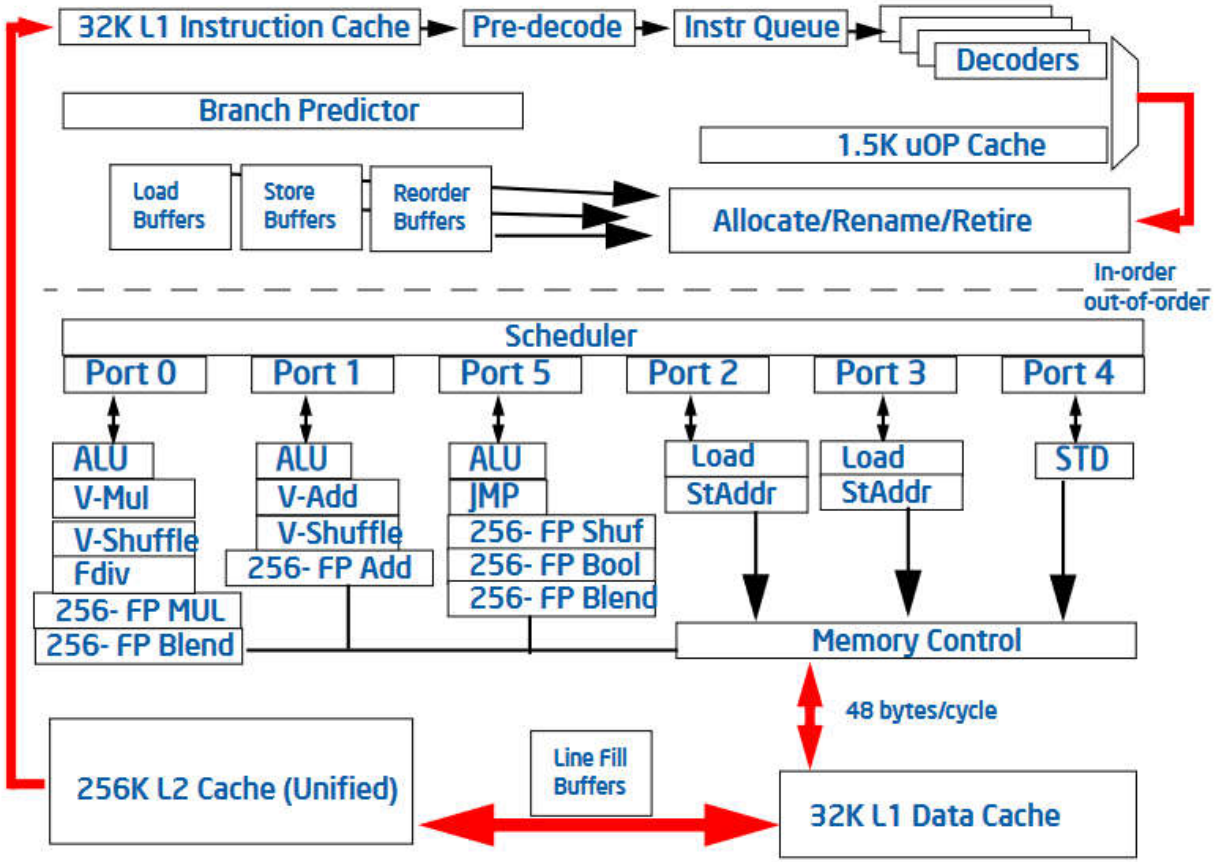
Intel Sandy Bridge Execution Engine



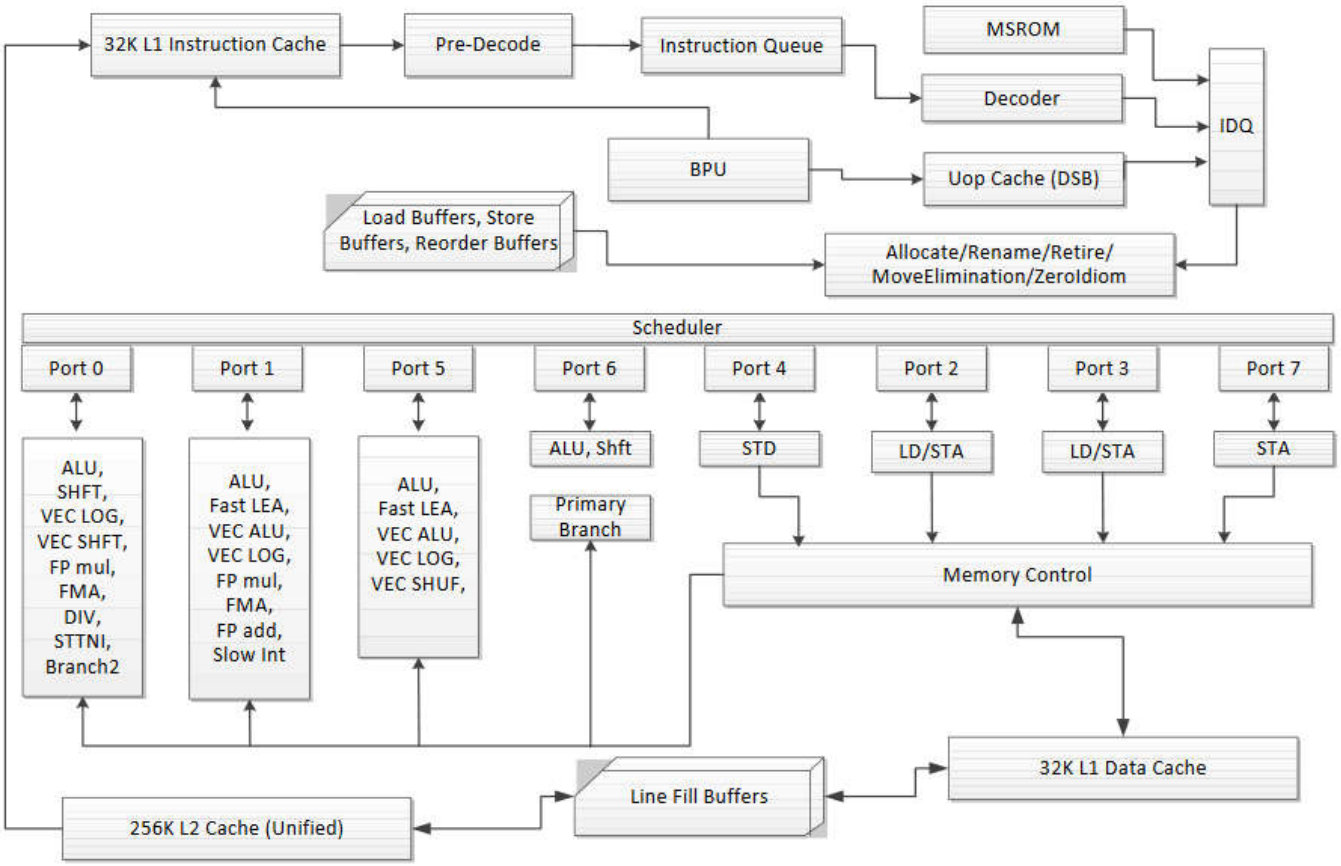
Intel Haswell Execution Engine



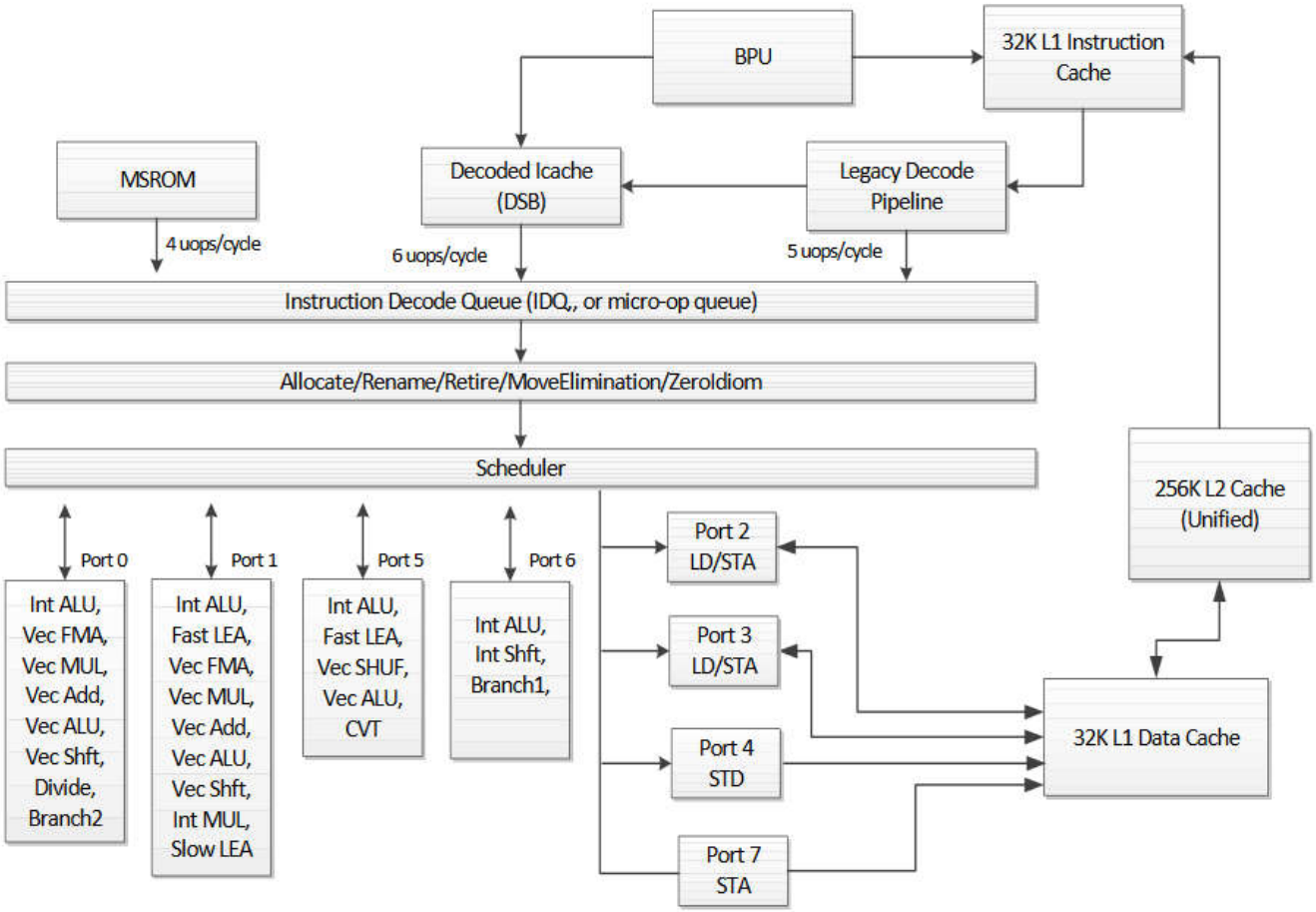
SandyBridge



Haswell


32K L1 Instruction Cache
Pre-Decode
Instruction Queue
MSROM
Decoder
IDQ
BPU
Uop Cache (DSB)
Load Buffers, Store Buffers, Reorder Buffers
Allocate/Rename/Retire/ MoveElimination/ZeroIdiom
Scheduler
Port 0
Port 1
Port 5
Port 6
Port 4
Port 2
Port 3
Port 7
ALU, SHFT, VEC LOG, VEC SHFT, FP mul, FMA, DIV, STTNI, Branch2
ALU, Fast LEA, VEC ALU, VEC LOG, FP mul, FMA, FP add, Slow Int
ALU, Fast LEA, VEC ALU, VEC LOG, VEC SHUF,
ALU, Shft
Primary Branch
STD
LD/STA
LD/STA
STA
Memory Control
32K L1 Data Cache
Line Fill Buffers
256K L2 Cache (Unified)

Skylake


BPU
32K L1 Instruction Cache
MSROM
Decoded Icache (DSB)
Legacy Decode Pipeline
4 uops/cycle
6 uops/cycle
5 uops/cycle
Instruction Decode Queue (IDQ,, or micro-op queue)
Allocate/Rename/Retire/MoveElimination/Zeroldiom
Scheduler
256K L2 Cache (Unified)
Port 0
Port 1
Port 5
Port 6
Int ALU, Vec FMA, Vec MUL, Vec Add, Vec ALU, Vec Shft, Divide, Branch2
Int ALU, Fast LEA, Vec FMA, Vec MUL, Vec Add, Vec ALU, Vec Shft, Int MUL, Slow LEA
Int ALU, Fast LEA, Vec SHUF, Vec ALU, CVT
Int ALU, Int Shft, Branch1,
Port 2 LD/STA
Port 3 LD/STA
Port 4 STD
Port 7 STA
32K L1 Data Cache

Examples

- Laptop Lenovo 320-15IKBN cu procesor Intel® Core™ i5-7200U 2.50 GHz, Kaby Lake, 15.6", 4GB, 1TB, DVD-RW, Intel HD Graphics, Free DOS, Black

PROCESOR

Producator procesor	Intel®
Tip procesor	i5
Model procesor	7200U
Arhitectura	Kaby Lake
Numar nuclee	2
Frecventa nominala	2500 MHz
Cache	3072 KB
Frecventa Turbo Boost	3100 MHz
Tehnologie procesor	14 nm
Procesor grafic integrat	Intel® HD Graphics 620

Examples

- ark.intel.com

Product Collection	7th Generation Intel® Core™ i5 Processors
Code Name	Products formerly Kaby Lake
Vertical Segment	Mobile
Processor Number	i5-7200U
Status	Launched
Launch Date	Q3'16
Lithography	14 nm
Recommended Customer Price	$281.00

Performance

# of Cores	2
# of Threads	4
Processor Base Frequency	2.50 GHz
Max Turbo Frequency	3.10 GHz
Cache	3 MB SmartCache
Bus Speed	4 GT/s OPI
TDP	15 W
Configurable TDP-up Frequency	2.70 GHz
Configurable TDP-up	25 W
Configurable TDP-down Frequency	800 MHz
Configurable TDP-down	7.5 W

Examples

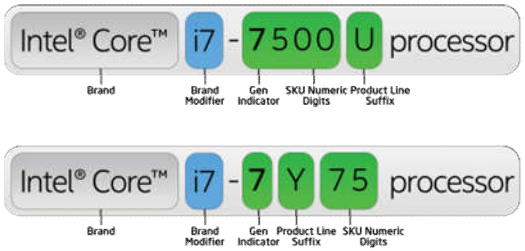
- Sistem Desktop PC ASUS K31CD-K-RO008D cu procesor Intel® Core™ i7-7700 pana la 4.20 GHz, Kaby Lake, 8GB, 256GB SSD, DVD-RW, Intel HD Graphics, Free DOS, Black, Mouse + Tastatura

PROCESOR

Producator procesor	Intel®
Tip procesor	i7
Model procesor	7700
Numar nuclee	4
Numar thread-uri	8
Arhitectura	Kaby Lake
Frecventa nominala	3600 MHz
Frecventa Turbo Boost	4200 MHz
Cache	8 MB
Tehnologie procesor	14 nm
Procesor grafic integrat	Intel® HD Graphics 630

CPU identification

- Brand
- Brand Identifier/Modifier
- Generation Indicator and SKU-Numeric Digits
- Letter and Product Line Suffix



CPU core generations

- 1: first generation – Nehalem - 45nm (architecture)
- 1: first generation – Westmere - 32nm (process)
- 2: second generation - Sandy Bridge - 32nm (architecture)
- 3: third generation - Ivy Bridge - 22nm (process)
- 4: fourth generation – Haswell - 22nm (architecture)
- 5: fifth generation – Broadwell - 14nm (process)
- 6: sixth generation – SkyLake - 14nm (architecture)
- 7: seventh generation – KabyLake - 14nm (optimization)
- 8: eighth generation – KabyLake R/ CoffeeLake - 14nm (optimization)
- 9: ninth generation – CoffeeLake R - 14nm (optimization)
- 10: tenth generation – Cannon Lake/Ice Lake - 10nm (process?)

Identificare procesoare

- H: High performance graphics (Generation 5, 6)
- HK: High performance graphics, unlocked (Generation 6)
- HQ: High performance graphics, quad core (Generation 4, 5, 6)
- M: Mobile (Generation 2, 3, 4)
- MQ: Quad core mobile (Generation 4)
- MX: Mobile extreme edition (Generation 4)
- QM: Quad core mobile (Generation 2, 3)
- U: Ultra low power (Generation 3, 4, 5, 6, 7, 8, 9, 10)
- Y: Extremely low power (Generation 3, 4, …, 10)
- K: Unlocked

Caracteristici procesor

- Lock vs. Unlocking processors
  - made with an unlocked clock multiplier. This means when paired with the proper chipset like the Z-170 and in the hands of a knowledgeable professional, unlocked processors can be Overclocked for faster than factory core speeds. Locked processors cannot be Overclocked.

Caracteristici procesor

- Overclocking vs. Turbo
  - You may have noticed that all current Intel i5 and i7 processors list two clock speeds in their specs – “3.3 GHz, up to 3.9 GHz,” for example. This is the Turbo feature included with these processors, and while similar in concept to Overclocking, is not the same thing as we refer to it here. Turbo mode is a sort of temporary overclock the processor does to itself automatically when in need of a little extra juice. The turbo process itself is totally seamless and something you wouldn’t even realize is happening within the depths of your PC. It’s not a feature that’s unique only to unlocked “k” processors.

Examples

- Desktop

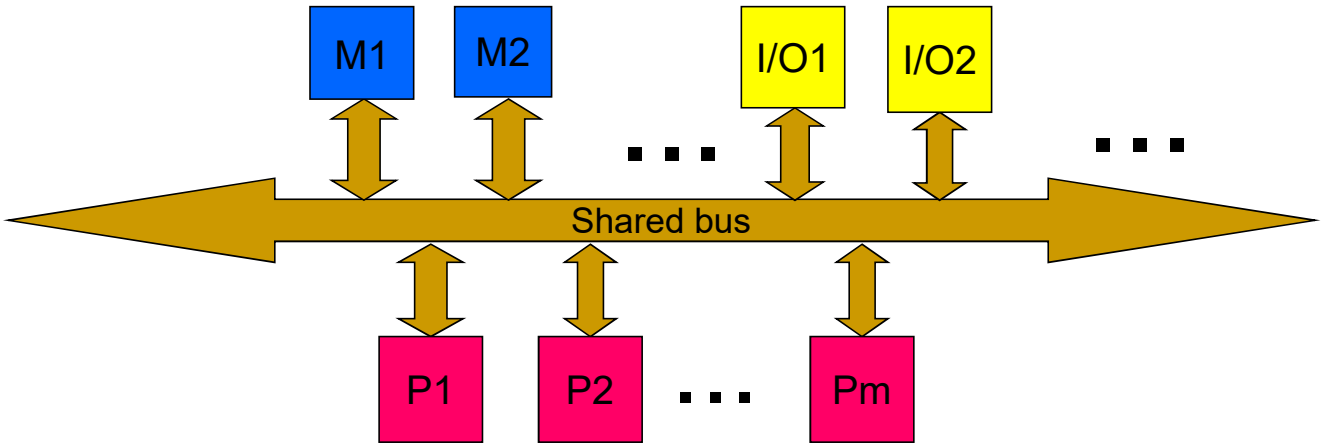
DESKTOP	Core i3	Core i5	Core i7
No. of cores	2	4	4
Frequency range	3.4-4.2GHz	2.4-3.8GHz	2.9-4.2GHz
Turbo Boost	No	Yes	Yes
Hyper-Threading	Yes	No	Yes
Cache	3-4MB	6MB	8MB

- Laptop

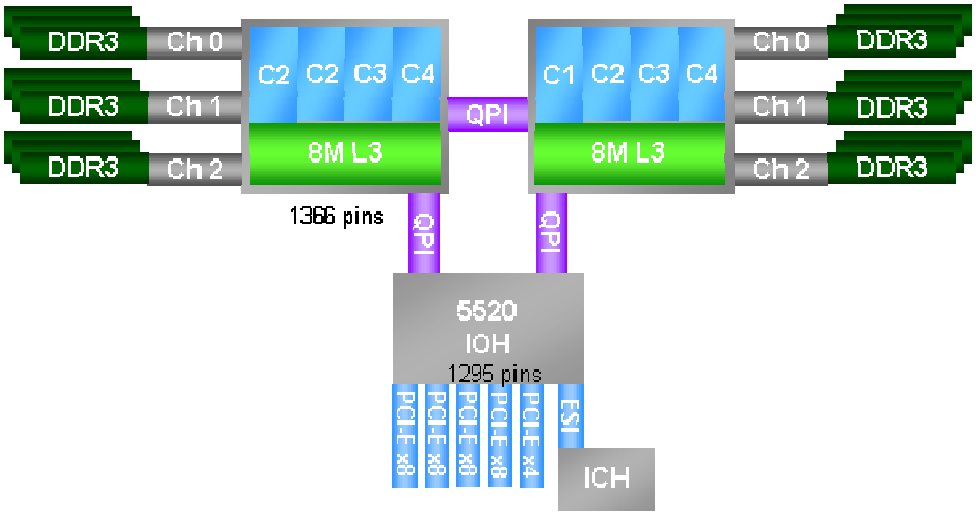
LAPTOP	Core i3/Core m3	Core i5	Core i7
No. of cores	2	02-Apr	02-Apr
Frequency range	1-3.5GHz	1.2-3.6GHz	1.3-3.5GHz
Turbo Boost	Y-Series only	Yes	Yes
Hyper-Threading	Yes	Dual-core chips only	Yes
Cache	3-4MB	3-6MB	4-8MB

Multiprocessor

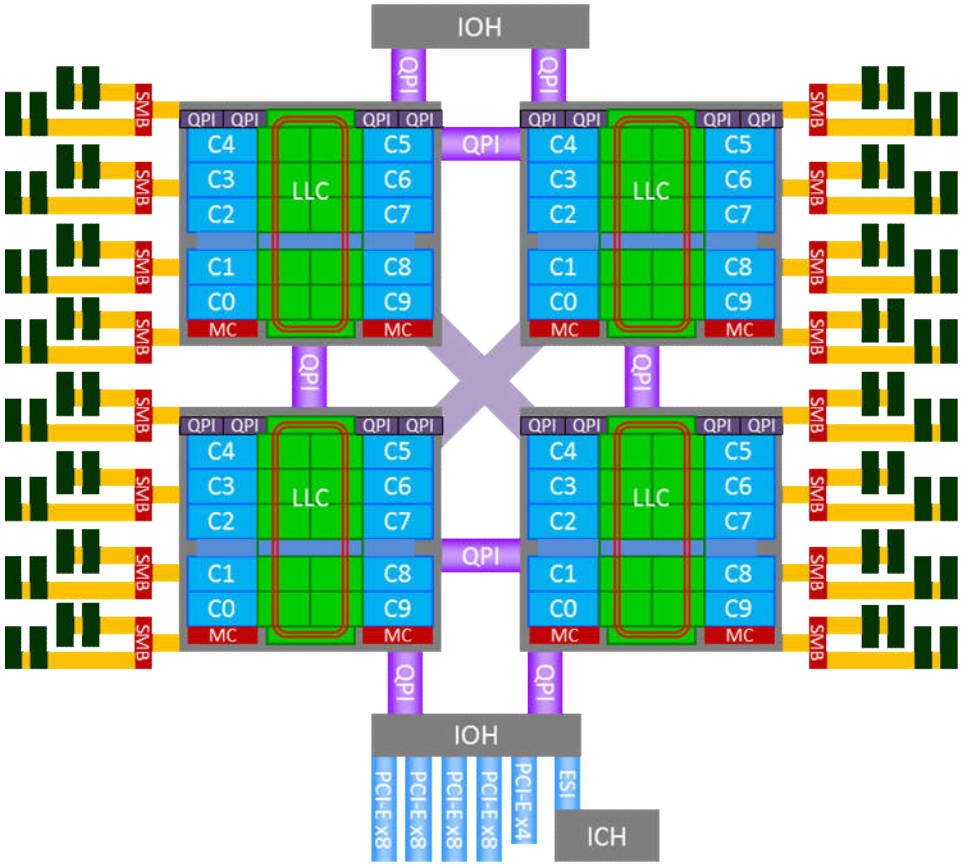
- Multiprocessor mode
  - Arbitration
  - Synchronization



Multi-processor



Multi-processor

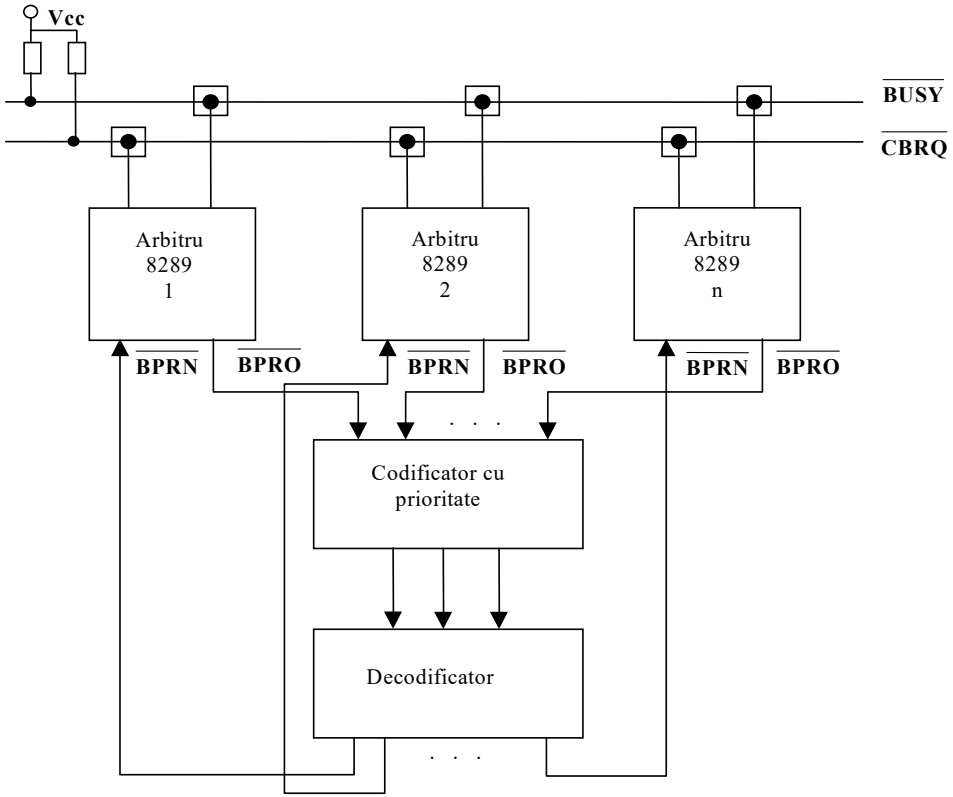


Multiprocessor

- Bus arbiter
  - Bus priorities – specify which processor will gain access when two or more processors will access the bus simultaneously
  - Signals
    - Bus request, bus grant
    - Priority in, priority out
  - Priority allocation
    - Parallel (centralized)
      - Fixed priority
      - Dynamic priorities (e.g. round robin)
    - Serial (decentralized)
      - Fixed priority

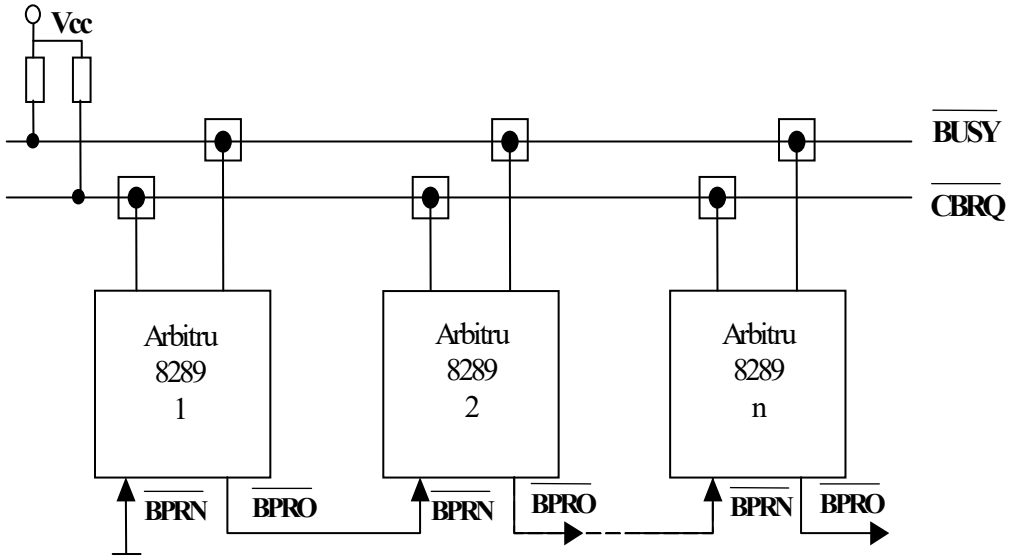
8086 microsystem

- Bus arbiter – parallel priority



8086 microsystem

- Bus arbiter – serial priority
  - Fixed priority



Multiprocessor

- Bus arbiter operation
  - The processor with highest priority will gain access to system bus in order to perform memory or I/O transfers
  - Lower priority processors will gain access only when processors with higher priority do not need access to system bus
  - Once the processor initiated a bus transfer it cannot be interrupted – bus transfer is an atomic operation
  - Dynamic priorities – centralized implementation of the arbiter that allows fair allocation of system bus

Multiprocessor

- Synchronization
  - Single bus transfer is atomic
  - Atomic transactions
    - more than one transfers on shared bus
    - uninterruptible
  - Required by (read-modify-write)
    - Critical section
    - Mutex
    - Semaphore

Multiprocessor

- Synchronization
  - What hardware mechanisms are required to implement synchronization?
  - How many bus transfers are needed to implement synchronization primitives?

Multiprocessor

- Synchronization
  - Single processor
    - Read-modify-write
    - Two transactions implemented atomic
  - Multi-processor
    - LOCK signal

```
          MOV AL, 1            ; initialize AL
WAIT:     LOCK XCHG [MEM], AL  ; exchange memory content
          TEST AL, 1           ; test if available
          JNZ    WAIT          ; wait if not available
```

Multiprocessor

- Shared bus multi-processor
  - Bus arbiter arbiters the multi-master access to shared bus based on the microprocessors' priorities
  - The processor having the highest priority will become bus master for the next bus cycle
  - A bus cycle is an atomic operation that cannot be interrupted by any processor
  - Processors having lower priority will announce their intention to transfer data using bus request signal
  - One processor can avoid loosing the bus by activating LOCK signal (e.g. implementation of synchronization primitives)

SIMD extensions

- Multimedia extension – MMX
- Streaming SIMD extensions
  - SSE, SSE2, SSE3, SSE4.1, SSE4.2
- Advanced Vector Extensions
  - AVX
- Usage:
  - Voice processing, filtering, recognition, coding
  - Video processing, filtering, recognition, coding
  - 3D graphics, CAD models
  - Security algorithms

SIMD extensions

- Identification of SIMD support

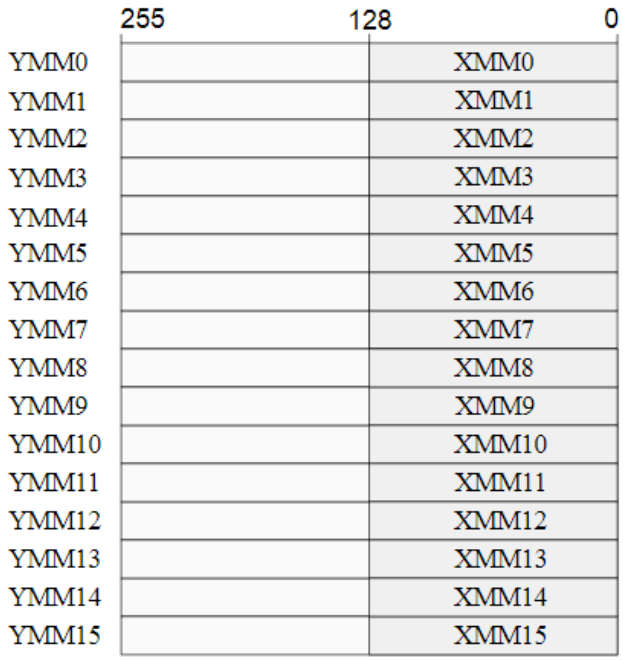
```
...
mov eax, 1
cpuid
test edx, 00800000h
jnz   mmx_found
...
```

SIMD extensions

- MMX and SSE registers set

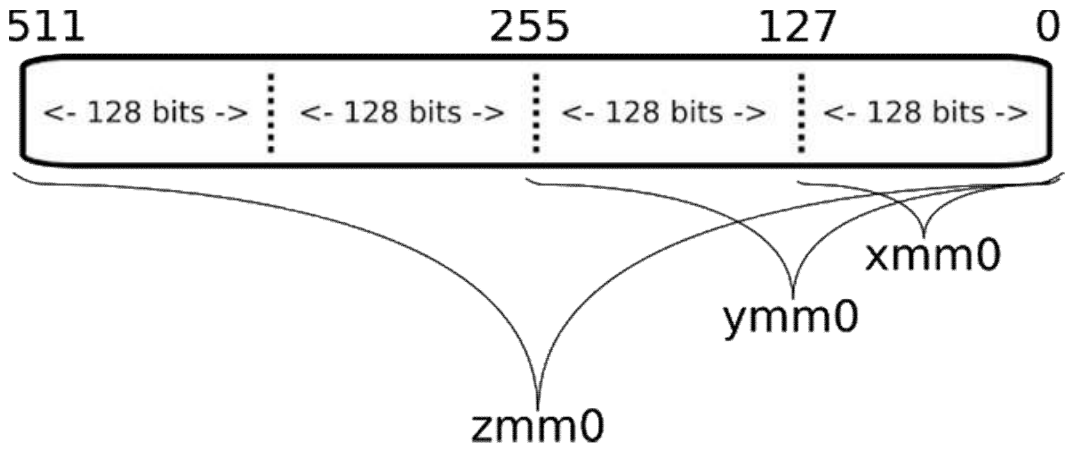
Extensii SIMD

- AVX (Advanced Vector Extension)
  - Registers 256 bits



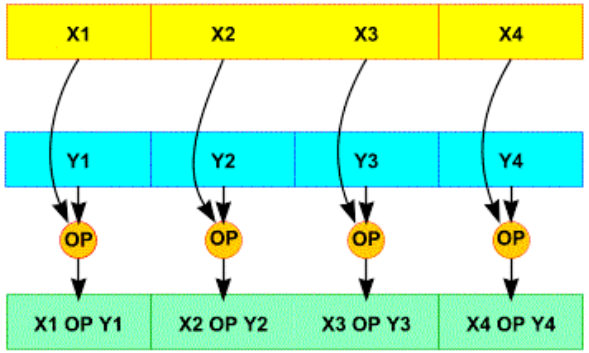
Extensii SIMD

- AVX 512



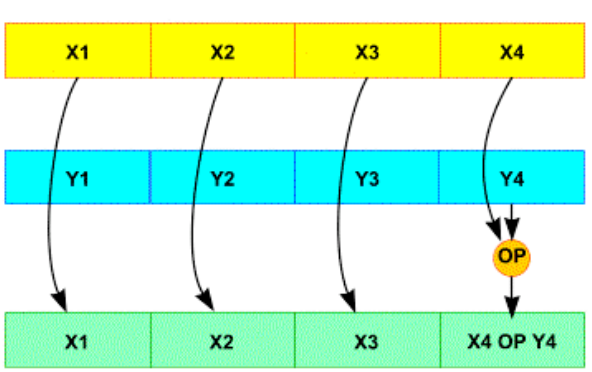
SIMD extensions

- SIMD operations and operands
  - Packed instructions



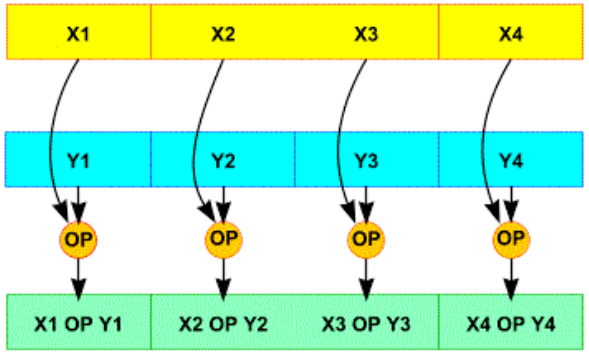
SIMD extensions

- SIMD operations and operands
  - Scalar instructions



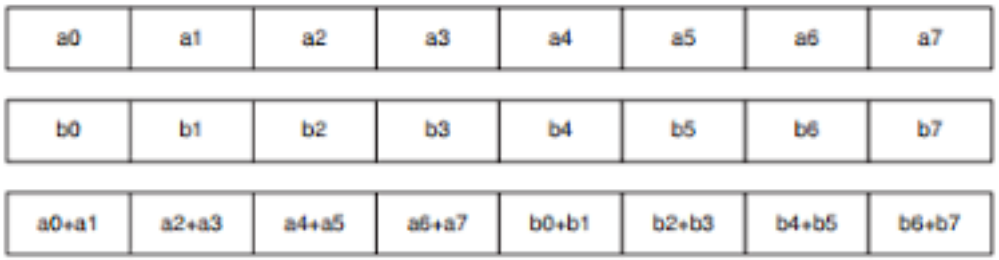
SIMD extensions

- SIMD operations and operands
  - Vertical operations



Extensii SIMD

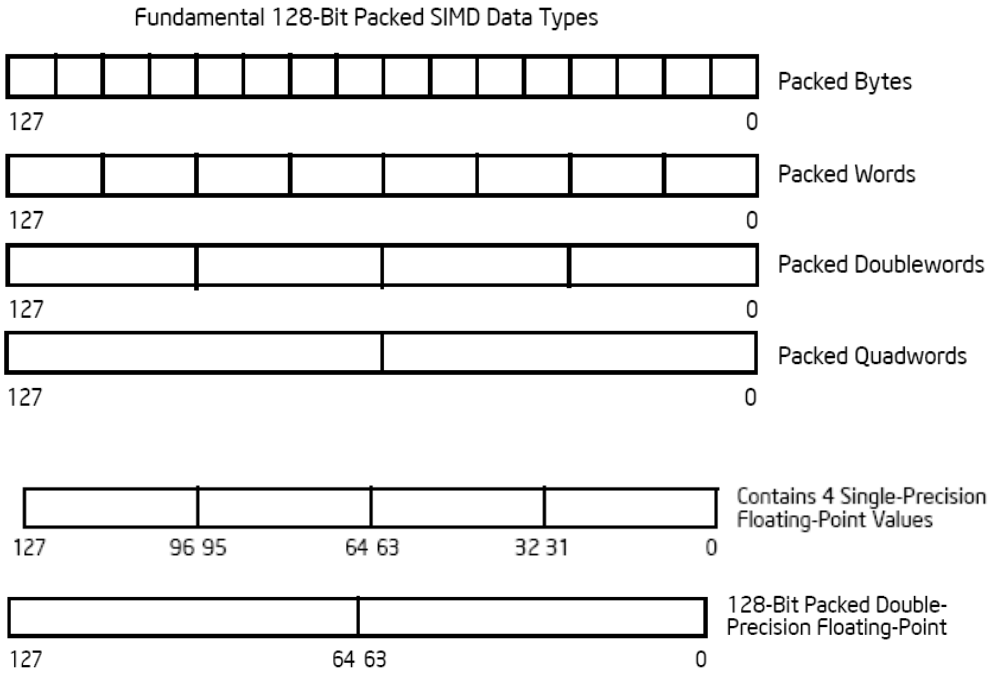
- Operaţii şi operanzi SIMD
  - Instrucţiunile operează orizontal asupra datelor



PHADDW instruction

SIMD extensions

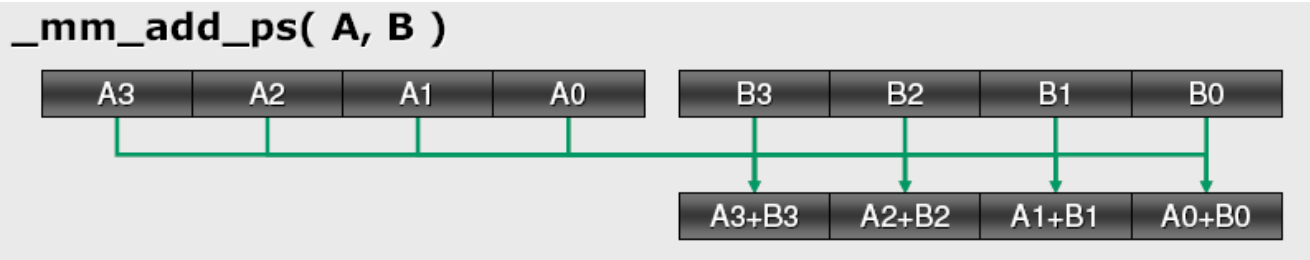
- SIMD operations and operands (SSE, SSE2)



SIMD extensions

```
int i;
for( i = 0; i < 4; i++ )
{
    c[i] = a[i] + b[i];
}

paddd xmm1, xmm2
```



SIMD extensions

- Overflow handling
  - Wraparound – remove most significant bits of the result
  - Signed saturation – limit the result to min/max signed value the data type can support
  - Unsigned saturation – limit the result to min/max unsigned value the data type can support

Data Type	Lower Limit		Upper Limit	
	Hexadecimal	Decimal	Hexadecimal	Decimal
Signed Byte	80H	-128	7FH	127
Signed Word	8000H	-32,768	7FFFH	32,767
Unsigned Byte	00H	0	FFH	255
Unsigned Word	0000H	0	FFFFH	65,535

SIMD extensions

- Arithmetic

Category		Wraparound	Signed Saturation	Unsigned Saturation
Arithmetic	Addition	PADDB, PADDW, PADDD	PADDSB, PADDSW	PADDUSB, PADDUSW
	Subtraction	PSUBB, PSUBW, PSUBD	PSUBSB, PSUBSW	PSUBUSB, PSUBUSW
	Multiplication	PMULL, PMULH		
	Multiply and Add	PMADD		

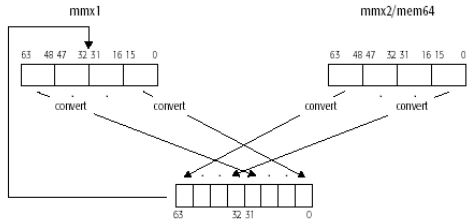
- Comparison

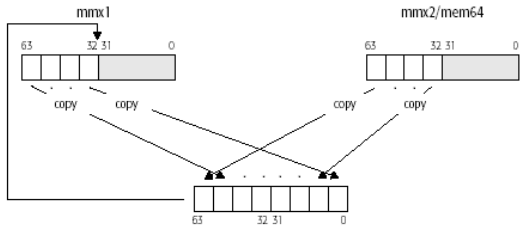
Comparison	Compare for Equal	PCMPEQB, PCMPEQW, PCMPEQD		
	Compare for Greater Than	PCMPGTPB, PCMPGTPW, PCMPGTPD		

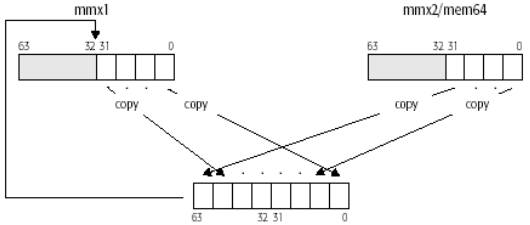
SIMD extensions

- Pack si Unpack

Conversion	Pack		PACKSSWB, PACKSSDW	PACKUSWB
Unpack	Unpack High	PUNPCKHBW, PUNPCKHWD, PUNPCKHDQ		
	Unpack Low	PUNPCKLBW, PUNPCKLWD, PUNPCKLDQ		





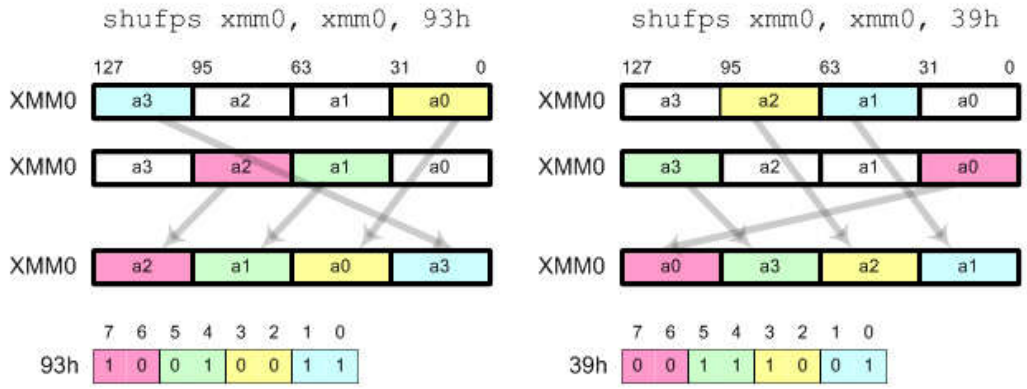


SIMD extensions

- Shuffle
  - SHUFPS op1, op2, mask

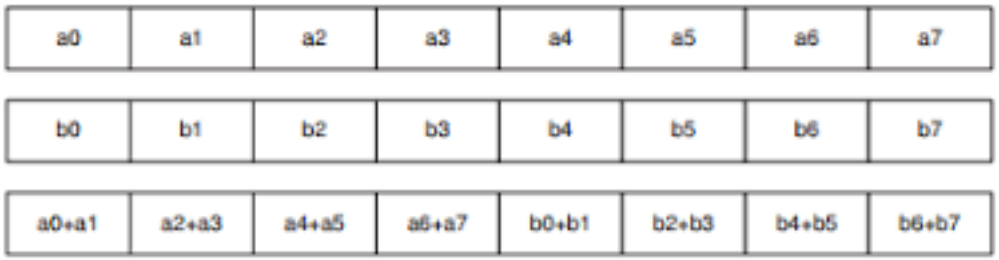
SIMD extensions

- Shuffle
  - SHUFPS op1, op2, mask



SIMD extensions

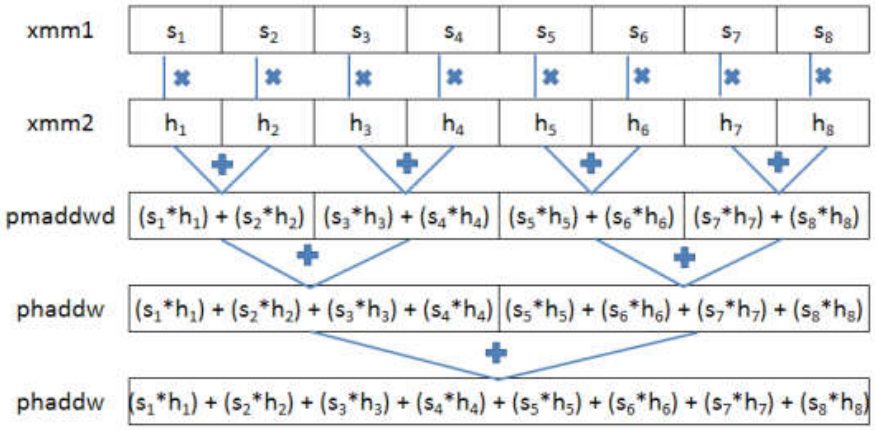
- PHADD – packet horizontal add
  - PHADDW, PHADDD



PHADDW instruction

SIMD extensions

- Pmaddwd – packet multiply add



SIMD extensions

- Memory transfers involving SIMD registers
  - Aligned transfer
    - movaps, movapd
  - Unaligned transfer
    - movups, movupd

```
  align 16
  v1:   dd 1.1, 2.2, 3.3, 4.4
; Four Single precision floats 32 bits each

   movaps xmm0, v1
```

SIMD extensions

- Memory transfers involving SIMD registers

```
__declspec( align( # ) ) declarator
```

```
# - multiplu de 2 – alinierea
declarator - variabila
```

SIMD extensions

- Packed operations with floating point numbers
  - addps, mulps, subps
  - addpd, mulpd, subpd
- Scalar operations
  - addss, ...
  - addsd

Summary

- Pipelining improves performance by increasing instruction throughput
  - Executes multiple instructions in parallel
  - Each instruction has the same latency
  - More efficient usage of resources
- Subject to hazards
  - Structure, data, control
- Instruction set design affects complexity of pipeline implementation

Summary

- Pipeline evolution
  - Pipeline

P5 Microarchitecture

  - Super-pipeline

P6 Microarchitecture

  - Hyper-pipeline

NetBurst Microarchitecture

ILP

- Multiple pipelines

Address	Instruction type	Pipeline Stages						
n	ALU/branch	IF	ID	EX	MEM	WB		
n + 4	Load/store	IF	ID	EX	MEM	WB		
n + 8	ALU/branch		IF	ID	EX	MEM	WB	
n + 12	Load/store		IF	ID	EX	MEM	WB	
n + 16	ALU/branch			IF	ID	EX	MEM	WB
n + 20	Load/store			IF	ID	EX	MEM	WB