Nehalem pt. 2

•••

Sandy Lee
Daniel Starikov

Single-Instruction Multiple-Data (SIMD)

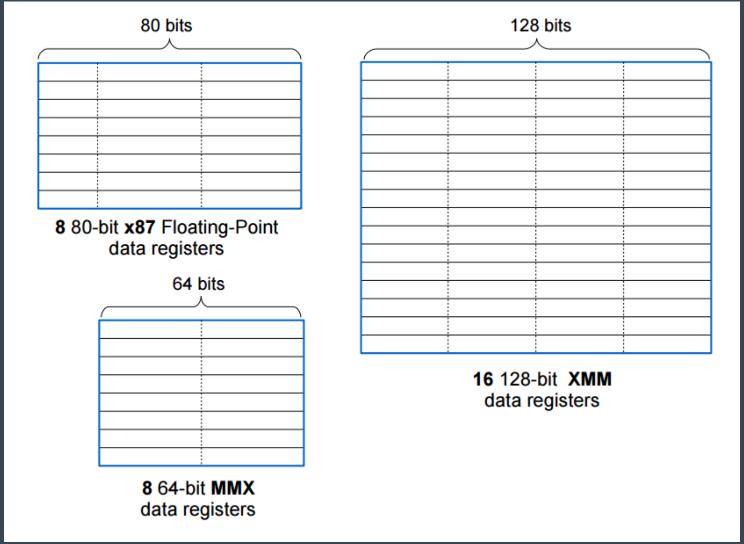
Same instruction is applied simultaneously to different sets of input operands and requires multiple ALUs

Floating point SIMD

- 2 double precision or 4 single precision
- Retires 4 instructions per cycle
- Supports ALU extensive code with SSE

Floating point registers

- Separate core registers XMM and MMX
- Certain memory areas can be treated as non-temporal, meaning that they can be used as buffers for vector data, which is more efficient that requesting them from the


80 bits
8 80-bit **x87** Floating-Point data registers
64 bits
8 64-bit **MMX** data registers
128 bits
16 128-bit **XMM** data registers

Floating-point Processing and Exception Handling

Compliant with IEEE standards which makes FP portable unlike before

Floating point exceptions (FPE) are thrown when an operation is considered invalid or precision is lost

Detectable exceptions:

- IEEE standard: NaN, ∞ - ∞
- Zero divide
- Numeric overflow
- Underflow
- Inexact

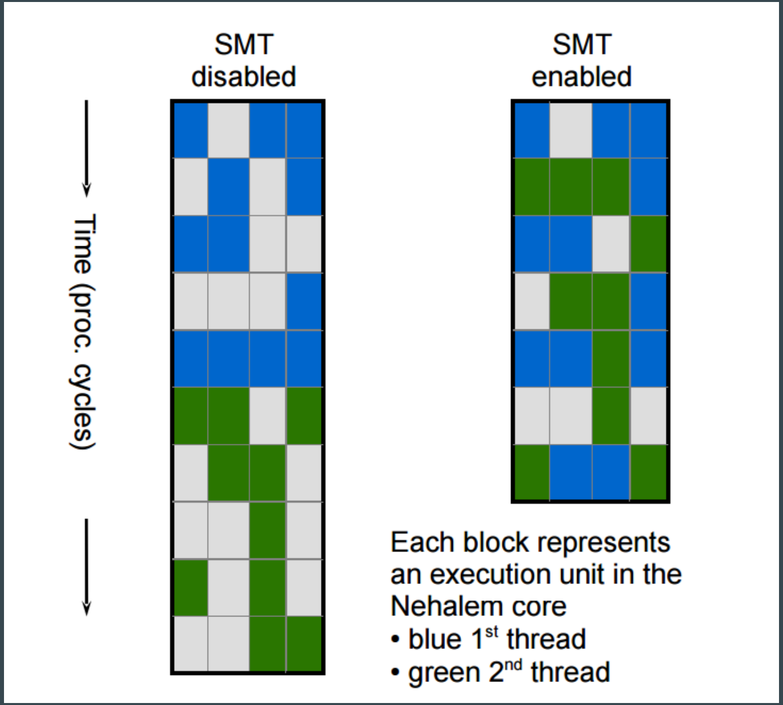
Simultaneous Multi-Threading (SMT)

A pipeline design that allows multiple hardware threads to execute simultaneously within each core with shared resources

In this case, two threads can be executed at the same time in each core

Increased utilization of functional units , overall increased throughput of instructions per clock cycle, and also preservation of energy consumed by idle states

More efficient in terms of power instead of another core



RISC vs CISC - Continued

Nehalem processors execute instructions in the same manner as RISC processors

The CPU accepts x86 CISC instructions and decodes them into RISC-like micro-ops

The rest of the CPU is indistinguishable from any other RISC-type processor.

This leaves the following questions about RISC vs CISC:

- Which one is more efficient for capturing higher-level semantics of applications
- Is it more efficient to use a RISC back-end with CISC front-end over a RISC front-end?
- What would the performance be like if the back end of the Nehalem was put into a RISC CPU and vice versa?

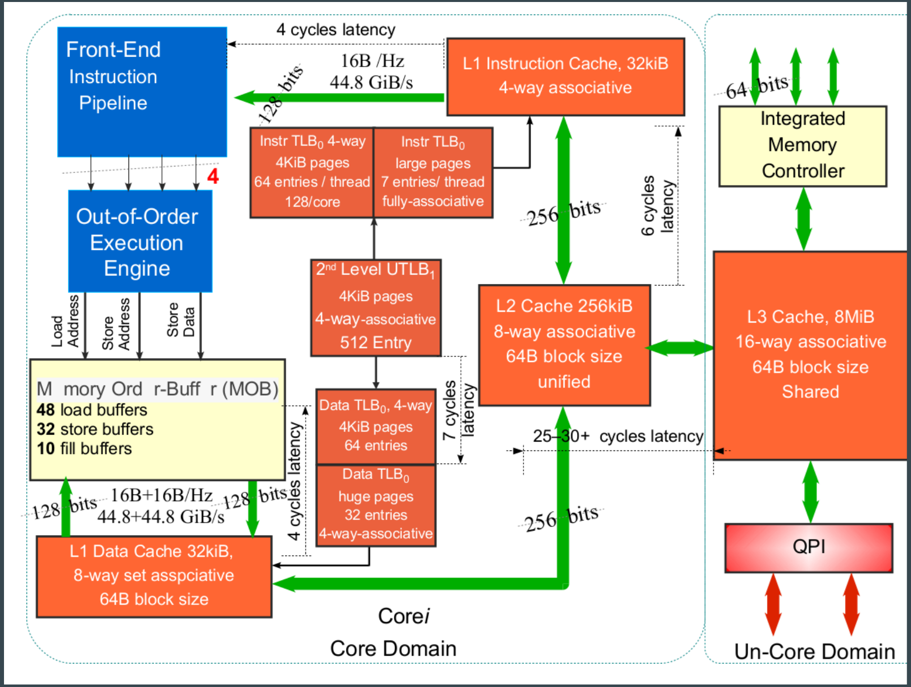
Memory Organization in Nehalem Processors

DRAM is cheap, more compact, and slow. SRAM is expensive and takes up lots of physical space.

- Nehalem's solution is to have three levels of cache between a core and main memory
  - L1 Cache - Level 1 is 32 KiB
  - L2 Cache - Level 2 is 256 KiB
  - L3 Cache - Level 3 is 8 MiB
- Other memory enhancements
  - Store Buffers
    - Store data being written to memory
  - Load and Store Enhancements

The L1 Cache

- Private
  - There is one for each core
  - Only the core that this cache is associated with can access it
- Harvard Style
  - Separate data and instruction caches
  - Both data and instruction caches are 32 KiB each with a 64 byte block size.
  - Instruction cache is 4-way set associative
  - Data cache is 8-way set associative
- The fastest



4 cycles latency
Front-End Instruction Pipeline
16B /Hz 44.8 GiB/s
128 bits
L1 Instruction Cache, 32kiB 4-way associative
64 bits
Integrated Memory Controller
4
Instr TLB0 4-way 4KiB pages 64 entries / thread 128/core
Instr TLB0 large pages 7 entries/ thread fully-associative
256 bits
6 cycles latency
Out-of-Order Execution Engine
2nd Level UTLB1 4KiB pages 4-way-associative 512 Entry
L2 Cache 256kiB 8-way associative 64B block size unified
L3 Cache, 8MiB 16-way associative 64B block size Shared
Load Address
Store Address
Store Data
M mory Ord r-Buff r (MOB)
48 load buffers
32 store buffers
10 fill buffers
Data TLB0, 4-way 4KiB pages 64 entries
7 cycles latency
25-30+ cycles latency
Data TLB0 huge pages 32 entries 4-way-associative
128 bits
16B+16B/Hz 44.8+44.8 GiB/s
128 bits
4 cycles latency
256 bits
QPI
L1 Data Cache 32kiB, 8-way set asspciative 64B block size
Corei
Core Domain
Un-Core Domain

The L2 Cache

- Private
  - There is one for each core
  - Only the core that this caches is associated with can access it
- Unified cache - Von Neumann Style
  - Data and instructions are stored together
  - 256 KiB
  - 8-way set associative
  - Block size is 64 bytes
- Fast
  - [illegible]

The L3 Cache

- Shared
  - There is only one L3 cache
  - All the cores can access this cache
- Unified cache - Von Neumann Style
  - Data and instructions are stored together
  - 8 MiB
  - 16-way set associative
  - Block size is 64 also bytes
- Not so fast
  - Access latency to data is 35-40+ clock cycles

Other Memory Enhancements

- Store buffers in between L1 cache and the core
    - Allow CPU to continue execution without waiting for write to memory/cache to complete
    - There are various safeguards in place to ensure that write operations occur in the proper order and that the buffered data is written to memory in the case of various instructions or exceptions.
- Data Load and Stores
    - Up to one 128-bit load and one 128-bit store per cycle.
    - Memory operations can be executed out of order
        - There is logic to perform speculative loads in the case of branches
        - Loads can be issues before stores if there is no address conflict - there is even speculative logic which can perform the load even if it is unsure if there will be a conflict.
    - There are 48 load buffers, 32 store buffers, and 10 fill buffers
    - Data pre-fetching for L1 caches