- Floating-point multiply instruction latency is improved from 5 cycles in prior generation to 3 cycle in the Broadwell microarchitecture. This applies to AVX, SSE and FP instruction sets.
- The throughput of Gather instructions has been improved significantly, see Table C-5.
- The PCLMULQDQ instruction implementation is a single uop in the Broadwell microarchitecture with improved latency and throughput.

The TLB hierarchy consists of dedicated level one TLB for instruction cache, TLB for L1D, plus unified TLB for L2.

2.2 INTEL® MICROARCHITECTURE CODE NAME SANDY BRIDGE

Intel® microarchitecture code name Sandy Bridge builds on the successes of Intel® Core™ microarchitecture and Intel microarchitecture code name Nehalem. It offers the following innovative features:

- Intel Advanced Vector Extensions (Intel AVX)
  - 256-bit floating-point instruction set extensions to the 128-bit Intel Streaming SIMD Extensions, providing up to 2X performance benefits relative to 128-bit code.
  - Non-destructive destination encoding offers more flexible coding techniques.
  - Supports flexible migration and co-existence

  - Improved prefetching.
  - High bandwidth low latency LLC architecture.
  - High bandwidth ring architecture of on-die interconnect.
- System-on-a-chip support
  - Integrated graphics and media engine in second generation Intel Core processors.
  - Integrated PCIE controller.
  - Integrated memory controller.
- Next generation Intel Turbo Boost Technology
  - Leverage TDP headroom to boost performance of CPU cores and integrated graphic unit.

2.2.1 Intel® Microarchitecture Code Name Sandy Bridge Pipeline Overview

Figure 2-4 depicts the pipeline and major components of a processor core that's based on Intel microarchitecture code name Sandy Bridge. The pipeline consists of

- An in-order issue front end that fetches instructions and decodes them into micro-ops (micro-operations). The front end feeds the next pipeline stages with a continuous stream of micro-ops from the most likely path that the program will execute.
- An out-of-order, superscalar execution engine that dispatches up to six micro-ops to execution, per cycle. The allocate/rename block reorders micro-ops to "dataflow" order so they can execute as soon as their sources are ready and execution resources are available.
- An in-order retirement unit that ensures that the results of execution of the micro-ops, including any exceptions they may have encountered, are visible according to the original program order.

The flow of an instruction in the pipeline can be summarized in the following progression:

1. The Branch Prediction Unit chooses the next block of code to execute from the program. The processor searches for the code in the following resources, in this order:
   a. Decoded ICache
   b. Instruction Cache, via activating the legacy decode pipeline
   c. L2 cache, last level cache (LLC) and memory, as necessary

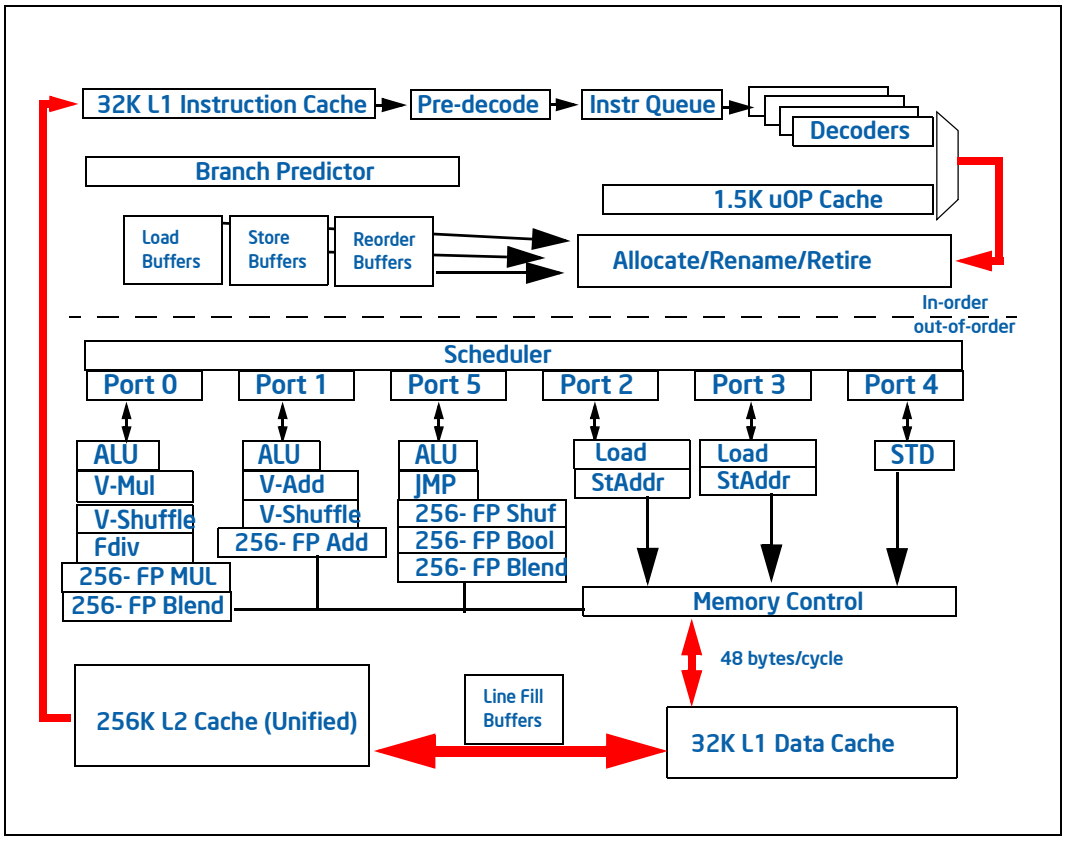

Figure 2-4. Intel Microarchitecture Code Name Sandy Bridge Pipeline Functionality

2. The micro-ops corresponding to this code are sent to the Rename/retirement block. They enter into the scheduler in program order, but execute and are de-allocated from the scheduler according to data-flow order. For simultaneously ready micro-ops, FIFO ordering is nearly always maintained.

   Micro-op execution is executed using execution resources arranged in three stacks. The execution units in each stack are associated with the data type of the instruction.

   Branch mispredictions are signaled at branch execution. It re-steers the front end which delivers micro-ops from the correct path. The processor can overlap work preceding the branch misprediction with work from the following corrected path.

3. Memory operations are managed and reordered to achieve parallelism and maximum performance. Misses to the L1 data cache go to the L2 cache. The data cache is non-blocking and can handle multiple simultaneous misses.

4. Exceptions (Faults, Traps) are signaled at retirement (or attempted retirement) of the faulting instruction.

Each processor core based on Intel microarchitecture code name Sandy Bridge can support two logical processor if Intel HyperThreading Technology is enabled.

2.2.2 The Front End

This section describes the key characteristics of the front end. Table 2-6 lists the components of the front end, their functions, and the problems they address.

Table 2-6. Components of the Front End of Intel Microarchitecture Code Name Sandy Bridge

Component	Functions	Performance Challenges
Instruction Cache	32-Kbyte backing store of instruction bytes	Fast access to hot code instruction bytes
Legacy Decode Pipeline	Decode instructions to micro-ops, delivered to the micro-op queue and the Decoded ICache.	Provides the same decode latency and bandwidth as prior Intel processors. Decoded ICache warm-up
Decoded ICache	Provide stream of micro-ops to the micro-op queue.	Provides higher micro-op bandwidth at lower latency and lower power than the legacy decode pipeline
MSROM	Complex instruction micro-op flow store, accessible from both Legacy Decode Pipeline and Decoded ICache	
Branch Prediction Unit (BPU)	Determine next block of code to be executed and drive lookup of Decoded ICache and legacy decode pipelines.	Improves performance and energy efficiency through reduced branch mispredictions.
Micro-op queue	Queues micro-ops from the Decoded ICache and the legacy decode pipeline.	Hide front end bubbles; provide execution micro-ops at a constant rate.

2.2.2.1 Legacy Decode Pipeline

The Legacy Decode Pipeline comprises the instruction translation lookaside buffer (ITLB), the instruction cache (ICache), instruction predecode, and instruction decode units.

Instruction Cache and ITLB

An instruction fetch is a 16-byte aligned lookup through the ITLB and into the instruction cache. The instruction cache can deliver every cycle 16 bytes to the instruction pre-decoder. Table 2-6 compares the ICache and ITLB with prior generation.

Table 2-7. ICache and ITLB of Intel Microarchitecture Code Name Sandy Bridge

Component	Intel microarchitecture code name Sandy Bridge	Intel microarchitecture code name Nehalem
ICache Size	32-Kbyte	32-Kbyte
ICache Ways	8	4
ITLB 4K page entries	128	128
ITLB large page (2M or 4M) entries	8	7

Upon ITLB miss there is a lookup to the Second level TLB (STLB) that is common to the DTLB and the ITLB. The penalty of an ITLB miss and a STLB hit is seven cycles.

Instruction PreDecode

The predecode unit accepts the 16 bytes from the instruction cache and determines the length of the instructions.

The following length changing prefixes (LCPs) imply instruction length that is different from the default length of instructions. Therefore they cause an additional penalty of three cycles per LCP during length decoding. Previous processors incur a six-cycle penalty for each 16-byte chunk that has one or more LCPs in it. Since usually there is no more than one LCP in a 16-byte chunk, in most cases, Intel microarchitecture code name Sandy Bridge introduces an improvement over previous processors.

- Operand Size Override (66H) preceding an instruction with a word/double immediate data. This prefix might appear when the code uses 16 bit data types, unicode processing, and image processing.
- Address Size Override (67H) preceding an instruction with a modr/m in real, big real, 16-bit protected or 32-bit protected modes. This prefix may appear in boot code sequences.
- The REX prefix (4xh) in the Intel® 64 instruction set can change the size of two classes of instructions: MOV offset and MOV immediate. Despite this capability, it does not cause an LCP penalty and hence is not considered an LCP.

Instruction Decode

There are four decoding units that decode instruction into micro-ops. The first can decode all IA-32 and Intel 64 instructions up to four micro-ops in size. The remaining three decoding units handle single-micro-op instructions. All four decoding units support the common cases of single micro-op flows including micro-fusion and macro-fusion.

Micro-ops emitted by the decoders are directed to the micro-op queue and to the Decoded ICache. Instructions longer than four micro-ops generate their micro-ops from the MSROM. The MSROM bandwidth is four micro-ops per cycle. Instructions whose micro-ops come from the MSROM can start from either the legacy decode pipeline or from the Decoded ICache.

MicroFusion

Micro-fusion fuses multiple micro-ops from the same instruction into a single complex micro-op. The complex micro-op is dispatched in the out-of-order execution core as many times as it would if it were not micro-fused.

Micro-fusion enables you to use memory-to-register operations, also known as the complex instruction set computer (CISC) instruction set, to express the actual program operation without worrying about a loss of decode bandwidth. Micro-fusion improves instruction bandwidth delivered from decode to retirement and saves power.

Coding an instruction sequence by using single-uop instructions will increases the code size, which can decrease fetch bandwidth from the legacy pipeline.

The following are examples of micro-fused micro-ops that can be handled by all decoders.

- All stores to memory, including store immediate. Stores execute internally as two separate functions, store-address and store-data.
- All instructions that combine load and computation operations (load+op), for example:
    - ADDPS XMM9, OWORD PTR [RSP+40]
    - FADD DOUBLE PTR [RDI+RSI*8]
    - XOR RAX, QWORD PTR [RBP+32]
- All instructions of the form "load and jump," for example:
    - JMP [RDI+200]
    - RET
- CMP and TEST with immediate operand and memory

An instruction with RIP relative addressing is not micro-fused in the following cases:

- An additional immediate is needed, for example:
    - CMP [RIP+400], 27
    - MOV [RIP+3000], 142
- The instruction is a control flow instruction with an indirect target specified using RIP-relative addressing, for example:
    - JMP [RIP+5000000]

In these cases, an instruction that can not be micro-fused will require decoder 0 to issue two micro-ops, resulting in a slight loss of decode bandwidth.

In 64-bit code, the usage of RIP Relative addressing is common for global data. Since there is no micro-fusion in these cases, performance may be reduced when porting 32-bit code to 64-bit code.

Macro-Fusion

Macro-fusion merges two instructions into a single micro-op. In Intel Core microarchitecture, this hardware optimization is limited to specific conditions specific to the first and second of the macro-fusable instruction pair.

- The first instruction of the macro-fused pair modifies the flags. The following instructions can be macro-fused:
  - In Intel microarchitecture code name Nehalem: CMP, TEST.
  - In Intel microarchitecture code name Sandy Bridge: CMP, TEST, ADD, SUB, AND, INC, DEC
  - These instructions can fuse if
    - The first source / destination operand is a register.
    - The second source operand (if exists) is one of: immediate, register, or non RIP-relative memory.
- The second instruction of the macro-fusable pair is a conditional branch. Table 3-1 describes, for each instruction, what branches it can fuse with.

Macro fusion does not happen if the first instruction ends on byte 63 of a cache line, and the second instruction is a conditional branch that starts at byte 0 of the next cache line.

Since these pairs are common in many types of applications, macro-fusion improves performance even on non-recompiled binaries.

Each macro-fused instruction executes with a single dispatch. This reduces latency and frees execution resources. You also gain increased rename and retire bandwidth, increased virtual storage, and power savings from representing more work in fewer bits.

2.2.2.2 Decoded ICache

The Decoded ICache is essentially an accelerator of the legacy decode pipeline. By storing decoded instructions, the Decoded ICache enables the following features:

- Reduced latency on branch mispredictions
- Increased micro-op delivery bandwidth to the out-of-order engine
- Reduced front end power consumption

The Decoded ICache caches the output of the instruction decoder. The next time the micro-ops are consumed for execution the decoded micro-ops are taken from the Decoded ICache. This enables skipping the fetch and decode stages for these micro-ops and reduces power and latency of the Front End. The Decoded ICache provides average hit rates of above 80% of the micro-ops; furthermore, "hot spots" typically have hit rates close to 100%.

Typical integer programs average less than four bytes per instruction, and the front end is able to race ahead of the back end, filling in a large window for the scheduler to find instruction level parallelism. However, for high performance code with a basic block consisting of many instructions, for example, Intel SSE media algorithms or excessively unrolled loops, the 16 instruction bytes per cycle is occasionally a limitation. The 32-byte orientation of the Decoded ICache helps such code to avoid this limitation.

The Decoded ICache automatically improves performance of programs with temporal and spatial locality. However, to fully utilize the Decoded ICache potential, you might need to understand its internal organization.

The Decoded ICache consists of 32 sets. Each set contains eight Ways. Each Way can hold up to six micro-ops. The Decoded ICache can ideally hold up to 1536 micro-ops.

The following are some of the rules how the Decoded ICache is filled with micro-ops:

- All micro-ops in a Way represent instructions which are statically contiguous in the code and have their EIPs within the same aligned 32-byte region.

- Up to three Ways may be dedicated to the same 32-byte aligned chunk, allowing a total of 18 micro-ops to be cached per 32-byte region of the original IA program.
- A multi micro-op instruction cannot be split across Ways.
- Up to two branches are allowed per Way.
- An instruction which turns on the MSROM consumes an entire Way.
- A non-conditional branch is the last micro-op in a Way.
- Micro-fused micro-ops (load+op and stores) are kept as one micro-op.
- A pair of macro-fused instructions is kept as one micro-op.
- Instructions with 64-bit immediate require two slots to hold the immediate.

When micro-ops cannot be stored in the Decoded ICache due to these restrictions, they are delivered from the legacy decode pipeline. Once micro-ops are delivered from the legacy pipeline, fetching micro-ops from the Decoded ICache can resume only after the next branch micro-op. Frequent switches can incur a penalty.

The Decoded ICache is virtually included in the Instruction cache and ITLB. That is, any instruction with micro-ops in the Decoded ICache has its original instruction bytes present in the instruction cache. Instruction cache evictions must also be evicted from the Decoded ICache, which evicts only the necessary lines.

There are cases where the entire Decoded ICache is flushed. One reason for this can be an ITLB entry eviction. Other reasons are not usually visible to the application programmer, as they occur when important controls are changed, for example, mapping in CR3, or feature and mode enabling in CR0 and CR4. There are also cases where the Decoded ICache is disabled, for instance, when the CS base address is NOT set to zero.

2.2.2.3 Branch Prediction

Branch prediction predicts the branch target and enables the processor to begin executing instructions long before the branch true execution path is known. All branches utilize the branch prediction unit (BPU) for prediction. This unit predicts the target address not only based on the EIP of the branch but also based on the execution path through which execution reached this EIP. The BPU can efficiently predict the following branch types:

- Conditional branches
- direct calls and jumps
- indirect calls and jumps
- returns

2.2.2.4 Micro-op Queue and the Loop Stream Detector (LSD)

The micro-op queue decouples the front end and the out-of order engine. It stays between the micro-op generation and the renamer as shown in Figure 2-4. This queue helps to hide bubbles which are introduced between the various sources of micro-ops in the front end and ensures that four micro-ops are delivered for execution, each cycle.

The micro-op queue provides post-decode functionality for certain instructions types. In particular, loads combined with computational operations and all stores, when used with indexed addressing, are represented as a single micro-op in the decoder or Decoded ICache. In the micro-op queue they are fragmented into two micro-ops through a process called un-lamination, one does the load and the other does the operation. A typical example is the following "load plus operation" instruction:

```
ADD                    RAX, [RBP+RSI] ; rax := rax + LD( RBP+RSI )
```

Similarly, the following store instruction has three register sources and is broken into "generate store address" and "generate store data" sub-components.

```
MOV                    [ESP+ECX*4+12345678], AL
```

The additional micro-ops generated by unlamination use the rename and retirement bandwidth. However, it has an overall power benefit. For code that is dominated by indexed addressing (as often happens with array processing), recoding algorithms to use base (or base+displacement) addressing can sometimes improve performance by keeping the load plus operation and store instructions fused.

The Loop Stream Detector (LSD)

The Loop Stream Detector was introduced in Intel® Core microarchitectures. The LSD detects small loops that fit in the micro-op queue and locks them down. The loop streams from the micro-op queue, with no more fetching, decoding, or reading micro-ops from any of the caches, until a branch miss-prediction inevitably ends it.

The loops with the following attributes qualify for LSD/micro-op queue replay:

- Up to eight chunk fetches of 32-instruction-bytes
- Up to 28 micro-ops (~28 instructions)
- All micro-ops are also resident in the Decoded ICache
- Can contain no more than eight taken branches and none of them can be a CALL or RET
- Cannot have mismatched stack operations. For example, more PUSH than POP instructions.

Many calculation-intensive loops, searches and software string moves match these characteristics.

Use the loop cache functionality opportunistically. For high performance code, loop unrolling is generally preferable for performance even when it overflows the LSD capability.

2.2.3 The Out-of-Order Engine

The Out-of-Order engine provides improved performance over prior generations with excellent power characteristics. It detects dependency chains and sends them to execution out-of-order while maintaining the correct data flow. When a dependency chain is waiting for a resource, such as a second-level data cache line, it sends micro-ops from another chain to the execution core. This increases the overall rate of instructions executed per cycle (IPC).

The out-of-order engine consists of two blocks, shown in Figure 2-4: Core Functional Diagram, the Rename/retirement block, and the Scheduler.

The Out-of-Order engine contains the following major components:

Renamer. The Renamer component moves micro-ops from the front end to the execution core. It eliminates false dependencies among micro-ops, thereby enabling out-of-order execution of micro-ops.

Scheduler. The Scheduler component queues micro-ops until all source operands are ready. Schedules and dispatches ready micro-ops to the available execution units in as close to a first in first out (FIFO) order as possible.

Retirement. The Retirement component retires instructions and micro-ops in order and handles faults and exceptions.

2.2.3.1 Renamer

The Renamer is the bridge between the in-order part in Figure 2-4, and the dataflow world of the Scheduler. It moves up to four micro-ops every cycle from the micro-op queue to the out-of-order engine. Although the renamer can send up to 4 micro-ops (unfused, micro-fused, or macro-fused) per cycle, this is equivalent to the issue port can dispatch six micro-ops per cycle. In this process, the out-of-order core carries out the following steps:

- Renames architectural sources and destinations of the micro-ops to micro-architectural sources and destinations.
- Allocates resources to the micro-ops. For example, load or store buffers.
- Binds the micro-op to an appropriate dispatch port.

Some micro-ops can execute to completion during rename and are removed from the pipeline at that point, effectively costing no execution bandwidth. These include:

- Zero idioms (dependency breaking idioms)
- NOP
- VZEROUPPER
- FXCHG

The renamer can allocate two branches each cycle, compared to one branch each cycle in the previous microarchitecture. This can eliminate some bubbles in execution.

Micro-fused load and store operations that use an index register are decomposed to two micro-ops, hence consume two out of the four slots the Renamer can use every cycle.

Dependency Breaking Idioms

Instruction parallelism can be improved by using common instructions to clear register contents to zero. The renamer can detect them on the zero evaluation of the destination register.

Use one of these dependency breaking idioms to clear a register when possible.

- XOR REG,REG
- SUB REG,REG
- PXOR/VPXOR XMMREG,XMMREG
- PSUBB/W/D/Q XMMREG,XMMREG
- VPSUBB/W/D/Q XMMREG,XMMREG
- XORPS/PD XMMREG,XMMREG
- VXORPS/PD YMMREG, YMMREG

Since zero idioms are detected and removed by the renamer, they have no execution latency.

There is another dependency breaking idiom - the "ones idiom".

- CMPEQ XMM1, XMM1; "ones idiom" set all elements to all "ones"

In this case, the micro-op must execute, however, since it is known that regardless of the input data the output data is always "all ones" the micro-op dependency upon its sources does not exist as with the zero idiom and it can execute as soon as it finds a free execution port.

2.2.3.2 Scheduler

The scheduler controls the dispatch of micro-ops onto their execution ports. In order to do this, it must identify which micro-ops are ready and where its sources come from: a register file entry, or a bypass directly from an execution unit. Depending on the availability of dispatch ports and writeback buses, and the priority of ready micro-ops, the scheduler selects which micro-ops are dispatched every cycle.

2.2.4 The Execution Core

The execution core is superscalar and can process instructions out of order. The execution core optimizes overall performance by handling the most common operations efficiently, while minimizing potential delays.

The out-of-order execution core improves execution unit organization over prior generation in the following ways:

- Reduction in read port stalls
- Reduction in writeback conflicts and delays
- Reduction in power
- Reduction of SIMD FP assists dealing with denormal inputs and underflowed outputs

Some high precision FP algorithms need to operate with FTZ=0 and DAZ=0, i.e. permitting underflowed intermediate results and denormal inputs to achieve higher numerical precision at the expense of reduced performance on prior generation microarchitectures due to SIMD FP assists. The reduction of SIMD FP assists in Intel microarchitecture code name Sandy Bridge applies to the following SSE instructions (and AVX variants): ADDPD/ADDPS, MULPD/MULPS, DIVPD/DIVPS, and CVTPD2PS.

The out-of-order core consist of three execution stacks, where each stack encapsulates a certain type of data. The execution core contains the following execution stacks:

- General purpose integer
- SIMD integer and floating-point
- X87

The execution core also contains connections to and from the cache hierarchy. The loaded data is fetched from the caches and written back into one of the stacks.

The scheduler can dispatch up to six micro-ops every cycle, one on each port. The following table summarizes which operations can be dispatched on which port.

Table 2-8. Dispatch Port and Execution Stacks

	Port 0	Port 1	Port 2	Port 3	Port 4	Port 5
Integer	ALU, Shift	ALU, Fast LEA, Slow LEA, MUL	Load_Addr, Store_addr	Load_Addr Store_addr	Store_data	ALU, Shift, Branch, Fast LEA
SSE-Int, AVX-Int, MMX	Mul, Shift, STTNI, Int-Div, 128b-Mov	ALU, Shuf, Blend, 128b-Mov			Store_data	ALU, Shuf, Shift, Blend, 128b-Mov
SSE-FP, AVX-FP_low	Mul, Div, Blend, 256b-Mov	Add, CVT			Store_data	Shuf, Blend, 256b-Mov
X87, AVX-FP_High	Mul, Div, Blend, 256b-Mov	Add, CVT			Store_data	Shuf, Blend, 256b-Mov

After execution, the data is written back on a writeback bus corresponding to the dispatch port and the data type of the result. Micro-ops that are dispatched on the same port but have different latencies may need the write back bus at the same cycle. In these cases the execution of one of the micro-ops is delayed until the writeback bus is available. For example, MULPS (five cycles) and BLENDPS (one cycle) may collide if both are ready for execution on port 0: first the MULPS and four cycles later the BLENDPS. Intel microarchitecture code name Sandy Bridge eliminates such collisions as long as the micro-ops write the results to different stacks. For example, integer ADD (one cycle) can be dispatched four cycles after MULPS (five cycles) since the integer ADD uses the integer stack while the MULPS uses the FP stack.

When a source of a micro-op executed in one stack comes from a micro-op executed in another stack, a one- or two-cycle delay can occur. The delay occurs also for transitions between Intel SSE integer and Intel SSE floating-point operations. In some of the cases the data transition is done using a micro-op that is added to the instruction flow. The following table describes how data, written back after execution, can bypass to micro-op execution in the following cycles.

Table 2-9. Execution Core Writeback Latency (cycles)

	Integer	SSE-Int, AVX-Int, MMX	SSE-FP, AVX-FP_low	X87, AVX-FP_High
Integer	0	micro-op (port 0)	micro-op (port 0)	micro-op (port 0) + 1 cycle
SSE-Int, AVX-Int, MMX	micro-op (port 5) or micro-op (port 5) +1 cycle	0	1 cycle delay	0
SSE-FP, AVX-FP_low	micro-op (port 5) or micro-op (port 5) +1 cycle	1 cycle delay	0	micro-op (port 5) +1 cycle
X87, AVX-FP_High	micro-op (port 5) +1 cycle	0	micro-op (port 5) +1 cycle	0
Load	0	1 cycle delay	1 cycle delay	2 cycle delay

2.2.5 Cache Hierarchy

The cache hierarchy contains a first level instruction cache, a first level data cache (L1 DCache) and a second level (L2) cache, in each core. The L1D cache may be shared by two logical processors if the processor support Intel HyperThreading Technology. The L2 cache is shared by instructions and data. All cores in a physical processor package connect to a shared last level cache (LLC) via a ring connection.

The caches use the services of the Instruction Translation Lookaside Buffer (ITLB), Data Translation Lookaside Buffer (DTLB) and Shared Translation Lookaside Buffer (STLB) to translate linear addresses to physical address. Data coherency in all cache levels is maintained using the MESI protocol. For more information, see the Intel® 64 IA-32 Architectures Software Developer's Manual, Volume 3. Cache hierarchy details can be obtained at run-time using the CPUID instruction. see the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*.

Table 2-10. Cache Parameters

Level	Capacity	Associativity (ways)	Line Size (bytes)	Write Update Policy	Inclusive
L1 Data	32 KB	8	64	Writeback	-
Instruction	32 KB	8	N/A	N/A	-
L2 (Unified)	256 KB	8	64	Writeback	No
Third Level (LLC)	Varies, query CPUID leaf 4	Varies with cache size	64	Writeback	Yes

2.2.5.1 Load and Store Operation Overview

This section provides an overview of the load and store operations.

Loads

When an instruction reads data from a memory location that has write-back (WB) type, the processor looks for it in the caches and memory. Table 2-11 shows the access lookup order and best case latency. The actual latency can vary depending on the cache queue occupancy, LLC ring occupancy, memory components, and their parameters.