Inside Nehalem: Intel's Future Processor and System

Introduction

During the early part of the decade, the paths for Intel and AMD's CPU architecture diverged rather remarkably. Intel focused on the Pentium 4, a high clock speed design that relied on proper coding practices by programmers to achieve high performance. AMD in contrast, focused on extending the K7 architecture by overhauling the system and memory interfaces and refining the microarchitecture. Along the way, Intel discovered that high frequency designs are an anathema to mobile devices and evolved the venerable Pentium Pro (P6) design into a line of extremely successful notebook oriented CPUs that culminated with the 65nm Core Duo. Over the last two years, Intel's CPU line-up has experienced a remarkable renaissance, as the seeds of the infamous right hand turn began to bear fruit and designs based on the Core 2 Duo displaced the last remnants of the Pentium 4.

In many ways, the Core 2 is the clear descendent of a mobile CPU – for instance the focus on dual-core implementations (at 45nm) or the simple clock distribution. As it turns out, the Core 2 is a well balanced design that is excellent for desktops and servers, but the feature set focuses on mobile. This reflects the strengths of Intel's Haifa design team, which has specialized in mobile designs since the Timna project, but only recently began working with server designs. In comparison, Intel's Hillsboro design group has a long history of server design and validation, dating back to the original P6.

Currently, almost all of Intel's product portfolio is based around the microarchitecture of the Core 2 and uses the aging front-side bus as a system interface. In fact, this is one of the largest technical distinctions between Intel and AMD's offerings. AMD's Barcelona, unlike the Core 2, is clearly aimed at servers first and foremost and takes full advantage of the integrated memory controller, HyperTransport and integrates four cores in a single die. Unfortunately, Barcelona has not met expectations yet – it was released later than expected, at lower frequencies and suffered from a functional bug in the translation look-aside buffer which required a microcode workaround. The latest release of Barcelona, the B3 stepping last month, fixes the most serious of these problems (the TLB bug) and will likely herald frequency increases that will bring Barcelona's performance into a range that is competitive with Intel's current generation of Core 2 based designs.

At this IDF, Intel is announcing the details of Nehalem, a second generation 45nm microprocessor and the next step in the evolution of their flagship line. Nehalem differs from the previous generation in that it was explicitly designed not only to scale across all the different product lines, but to be optimized for all the different product segments, from mobile to MP server. This implies a level of flexibility above and beyond the Core 2. Nehalem refines almost every aspect of the microprocessor, although the most substantial changes were to the system architecture and the memory hierarchy. This article describes in detail the architecture and pipeline of Nehalem, a quad-core, eight threaded, 64 bit, 4 issue super-scalar, out-of-order MPU with a 16 stage pipeline, 48 bit virtual and 40 bit

physical addressing, implemented in Intel's high performance 45nm process which uses high-K gate dielectrics and metal gate stacks [1].

Cores, Caches and Interconnects

Intel's current front-side bus based architecture is quite workable for notebook and desktop systems, but is far from ideal for modern servers and workstations. Notebook systems use one to two cores and are cost and power sensitive, and need low latency for a single task. Desktop requirements are similar, but more sensitive to cost and less sensitive to power; but they must also be able to handle the extreme bandwidth required by high performance discrete graphics solutions. In contrast, servers can use an unlimited number of cores and require massive bandwidth and low latency for many different tasks operating simultaneously while keeping power within a reasonable envelope. At the higher end of the spectrum, reliability and availability becomes a key concern for MP servers. Given that one of the goals for Nehalem was to define a flexible architecture that could be optimized for (and not just shoehorned into) all different market segments, one of the key changes is to adopt a system architecture that could comfortably address all of these competing goals and requirements. Nehalem represents a complete and total overhaul of Intel's system infrastructure, one of the most dramatic changes since the introduction of the P6 in the latter part of the last decade. Figure 1 below shows the system architecture for Nehalem, Harpertown and Barcelona devices.

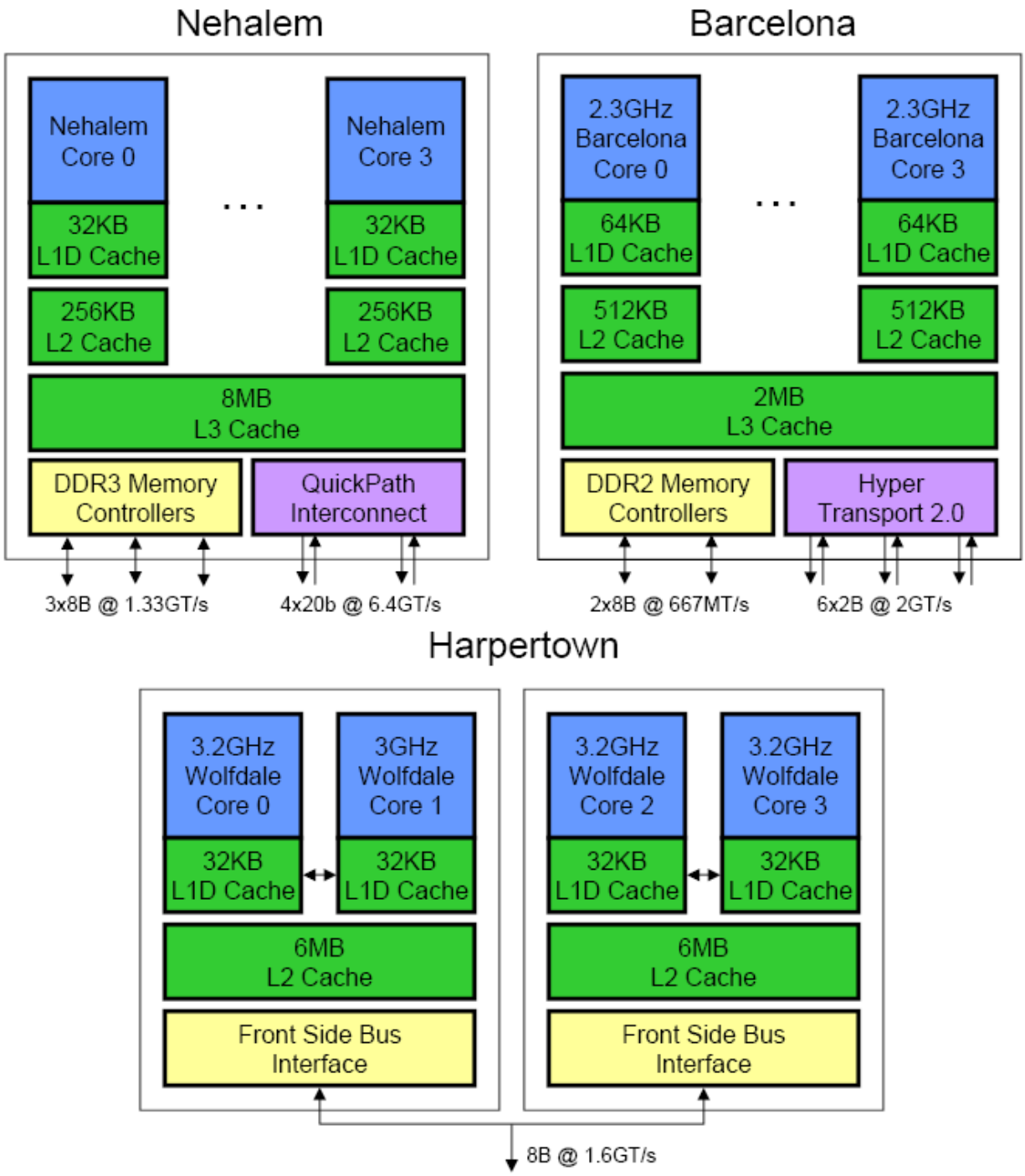

Figure 1 – System Architecture Comparison, B indicates Bytes, b indicates bits

Nehalem is a fully integrated quad-core device, with an inclusive and shared last level cache. A central queue acts as a crossbar and arbiter between the four cores and the 'uncore' region of Nehalem, which includes the L3 cache, integrated memory controller and QPI links. From a performance perspective, the inclusive L3 cache is the ideal configuration since it keeps the most cache coherency transactions on-die. On-die communication has the benefit of both lower latency and lower power. Additionally, a shared last level cache can reduce replication. AMD made this

transition with Barcelona at 65nm, leading to a massive 283mm^2 die size as a result. Intel's 45nm Harpertown was actually two dual-core devices in a package, which is advantageous due to the smaller die size and more flexible binning options. Depending on the workload, there was probably a fair bit of data replicated between the two caches in Harpertown (certainly the working set for the instruction caches at least). By eliminating this duplication, a unified cache can actually be smaller yet cache the same amount of data.

While an integrated quad-core has performance advantages, as Intel itself has pointed out previously there are costs to this degree of integration. Greater die size means lower percentage yields (and absolute yields) per wafer, which can be problematic early in the lifecycle of a manufacturing process. Greater integration can reduce the number of top bin devices, since a device runs at the speed of the slowest component, and increase vulnerability to point defects.

Nehalem also replaces the front-side bus, with an integrated memory controller and on-die dedicated interprocessor interconnects [2]. The QuickPath Interconnect (previously known as the Common System Interface or CSI) has been extensively described in an earlier article. To summarize, QPI is a packet-based, high bandwidth, low latency point to point interconnect that operates at up to 6.4GT/s in 45nm (4.8GT/s in 65nm). Each full width link is implemented as a 20 bit wide interface using high speed differential signaling and dedicated clock lanes (with failover between clock lanes). QPI packets (or flits) are 80 bits in length and transmitted in 4-16 cycles, depending on link width. Although flits are 80 bits long, only 64 bits is available for data, with the remainder used for flow control, CRC and other purposes. This means that each link provides 16 bits or 2 bytes of data per transfer for a total of 12.8GB/s in each direction – the remaining bits are used for CRC. Since links are bi-directional, there is actually 25.6GB/s total for a full-width link. Implementations of Nehalem will scale the number of QPI links based upon the target market and system complexity, with client systems having the fewest (as few as one or a half width link), DP servers will have two links and MP servers likely three and a half or four.

Memory Controllers, SSE4.2 and SMT

The integrated memory controller for Nehalem features up to 3 channels of DDR3 memory operating at 1.33GT/s at launch, for a total of 32GB/s peak bandwidth. The memory controller supports both registered and un-registered DDR3, but no FB-DIMMs for the mainstream implementations. FB-DIMM support will likely come with Beckton or Nehalem EX. Each channel of memory can operate independently and the controller services requests out-of-order to minimize latency. To take advantage of this 4x increase in memory bandwidth, each core supports up to 10 data cache misses and 16 total outstanding misses. In comparison, the Core 2 could have 8 data cache misses and 14 total misses in-flight.

The integrated memory controller substantially improves memory latency (especially relative to FB-DIMM based solutions). The local memory latency for Nehalem is about 60% of the latency for a

Harpertown system using the 1.6GT/s front-side bus, which implies the absolute latency is on the order of 60ns (Harpertown is just slightly under 100ns). For two socket implementations of Nehalem, the remote latency is higher, since the memory request and response must go through a QPI link. Remote latency is roughly 95% of a Harpertown system – so even in the worse case, latency will improve. An interesting question is where four socket servers based on Nehalem will fall. It seems likely that these systems will use FB-DIMMs, which implies a latency penalty, but the remote latency should still be about the same – roughly 30ns slower than local.

As with other systems that use an integrated memory controller and on-die interconnects (EV7, K8, Barcelona, etc.), the memory latency is non-uniform (NUMA). For optimal performance, the operating system must be aware of the differences in latencies and schedule processes that share data on the same socket. While Linux and proprietary operating systems have been NUMA-aware for a long time, Windows Vista is the first client operating system from Microsoft to make any NUMA optimizations.

The difference in local versus remote memory latency is roughly 1.5X for Nehalem. Measurements for the K8 show that the NUMA factor (i.e. remote latency divided by local) is roughly the same for two socket systems [4]. However, for four socket systems, Intel will have an advantage since all memory will be either be local (no hops over QPI) or remote (one hop over QPI), while current four socket K8 and Barcelona systems have some memory that is two hops away over HyperTransport. In general, the larger the NUMA factor, the more important it is for software to take memory locality into account. For reference, one of the first processors with an integrated memory controller and on-die interconnects (the EV7) had NUMA factors from 1.86-5.21 (1-8 hops away) in a 64P system [2].

Core Wide Changes

Nehalem also includes an instruction set extension which spans the entire core pipeline since they impact microcode. Nehalem includes the SSE4.2 instructions, which include several instructions for string manipulations, a CRC instruction and a popcount. The string instructions are all microcoded and will only show a small performance gain. The CRC instruction is used for calculating checksums which is useful for storage and networking and provides fairly substantial benefits – in the range of 6-18X for the code snippets that Intel demonstrated. Of course, the overall speedup will be much smaller, since Intel's examples just deal with the tightest inner loop.

The last major change to the system architecture for Nehalem is the return of Simultaneous Multi-Threading (SMT), which was first discussed in the context of the EV8, but first appeared for the 130nm P4. While SMT is not strictly speaking a system level change, but a core change, the implications span all major aspects of Nehalem so it is best to mention it upfront. Additionally, SMT has implications for system architecture. Given two identical microprocessors, one with SMT and one without, an SMT-enabled CPU will sustain more outstanding misses to memory and use more bandwidth. As a result, Nehalem is very likely designed with the requirements of SMT in-mind. For instance, variants of Nehalem which do not use SMT (notebook and desktop processors most likely) may not really need the full three channels of memory.

One interesting issue is why the Core 2 didn't use SMT. Certainly it was possible, as Nehalem shows. SMT increases performance in a very power efficient way, which is a huge win, and the software infrastructure was already there. There are two possible answers. First, Core 2 might not have had enough memory and inter-processor bandwidth to really take advantage of SMT for some workloads. In general, SMT substantially increases the amount of memory level parallelism (MLP) in a system, but that could be problematic when the system is already bottlenecked on memory bandwidth.

A much more plausible explanation is that while designing a SMT processor is relatively easy – the validation is extremely difficult. Supposedly Willamette, the 180nm P4, actually had all the necessary circuitry for SMT present, but it was disabled due to the difficulty of validating SMT until the tail end of the Northwood 130nm generation. More importantly, almost all of the experience with designing, validating and debugging SMT processors resides with Intel's Hillsboro design team, rather than the group in Haifa. Thus a decision to avoid SMT for the Core 2 makes a lot of sense from a risk management perspective.

The Front-end: Fetch Phase

Nehalem's fetch phase has been fairly substantially modified, although Figure 2 does not show some of these details. The instruction fetch unit in the diagram below contains the relative instruction point (RIP), which is replicated, one for each thread context.

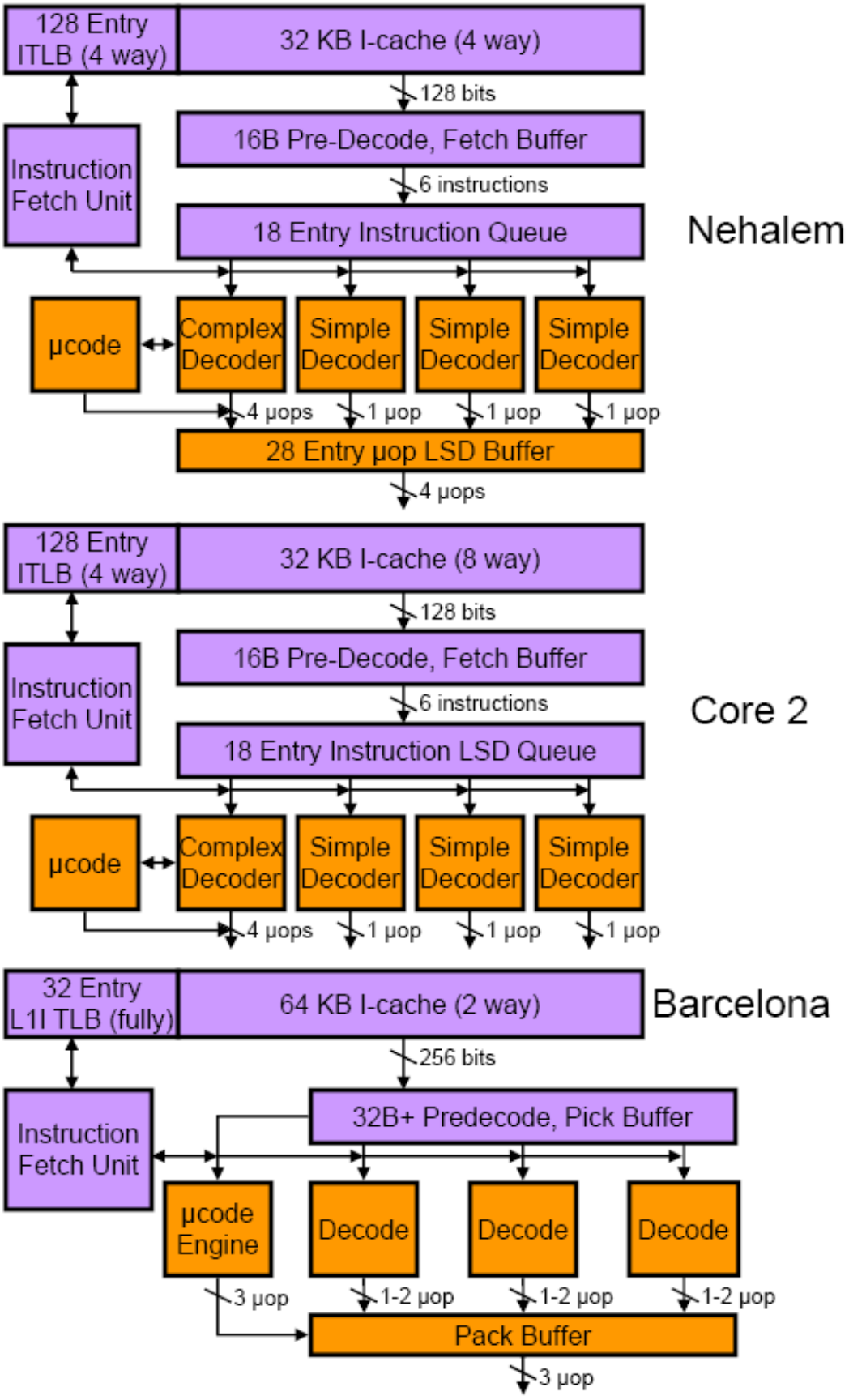
128 Entry ITLB (4 way)
32 KB I-cache (4 way)
128 bits
16B Pre-Decode, Fetch Buffer
Instruction Fetch Unit
6 instructions
18 Entry Instruction Queue
Nehalem
µcode
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
4 µops
1 µop
1 µop
1 µop
28 Entry µop LSD Buffer
4 µops
128 Entry ITLB (4 way)
32 KB I-cache (8 way)
128 bits
16B Pre-Decode, Fetch Buffer
Instruction Fetch Unit
6 instructions
18 Entry Instruction LSD Queue
Core 2
µcode
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
4 µops
1 µop
1 µop
1 µop
32 Entry L1I TLB (fully)
64 KB I-cache (2 way)
Barcelona
256 bits
Instruction Fetch Unit
32B+ Predecode, Pick Buffer
µcode Engine
Decode
Decode
Decode
3 µop
1-2 µop
1-2 µop
1-2 µop
Pack Buffer
3 µop

Figure 2 – Front-end Microarchitecture Comparison

The instruction fetch unit also contains the branch predictor, which is responsible for predicting the RIP of the next instructions to be fetched. Nehalem's branch predictors are not shown in detail, partially because some of the details are unknown – Intel simply states that they use "best in class" branch predictors and that their predictors are tuned to work with SMT. Intel did confirm that Nehalem continues to use all of the special predictors from the previous generations, such as the loop detector, indirect predictor, etc.

Once the branch predictor has determined that a branch is taken, the branch target buffer (BTB) is responsible for predicting the target address. Nehalem augments the previous generation of branch prediction by using a two level BTB scheme. For reference, Barcelona uses a 2K entry BTB for direct branches, and a 512 entry indirect branch target array.

Nehalem's two level BTB is designed to increase performance and power efficiency by improving branch prediction accuracy for workloads with larger instruction footprints (such as databases, ERP and other commercial applications). At this time, Intel is not describing the exact internal arrangement, but it is very possible to make an educated guess or two.

At a high level, there are two possibilities. The first possibility is that the two BTBs could use the same predictor algorithm, but one accesses a smaller history file that contains the most recently used branch RIPs and target RIPs. If that were the case, then the relationship between the BTBs would be the same as the relationship of an L1 cache to an L2 cache (remember that branch targets have fairly good locality). For example, the first level BTB could hold 256-512 entries, while a larger second level BTB could hold 2-8K entries. If a branch RIP is not in the first level BTB, then it would check the second level to make a target prediction. This approach has the benefit of using relatively little power, except when the instruction footprint is very big (i.e. does not fit in the L1 BTB). However, in that situation the extra power saved by fewer branch mispredictions will more than compensate for the extra power used by the L2 BTB.

The second alternative (which is much less likely) is that the first and second level BTBs actually use different prediction algorithms AND different history files. For example, the first level BTB could be configured to use a very simple and fast algorithm with a relatively small history table, while the second level would use a slower and more accurate algorithm and the second level BTB would be configured as an over-riding predictor. If the second level predictor disagrees with the first predictor, then it overrides the first level and has to fix up the pipeline by getting rid of any erroneously fetched instructions and starting to fetch from the newly predicted RIP. This sort of scheme is relatively unlikely since it is very power inefficient. In a two level over-riding predictor, the common case is that the L1 BTB and L2 BTB both independently come up with the correct branch target – which means that most of the time the L2 BTB is just wasting power. The over-riding predictor is only truly energy efficient when the L1 BTB is incorrect, and the L2 BTB is correct – which is a very small percentage of the time. While we do not believe Intel used this approach, it is worth mentioning since it is a possible option.

Another improved branch target prediction mechanism in Nehalem is the return stack buffer (RSB). When a function is called, the RSB records the address, so that when the function is returned, it will just pick up where it left off instead of ending up at the wrong address. A RSB can overflow if too many functions are called recursively and it can also get corrupted and produce bad return addresses if the branch predictor speculates down a wrong path. Nehalem actually renames the RSB, which avoids return stack overflows, and ensures that most misspeculation does not corrupt the RSB. There is a dedicated RSB for each thread to avoid any cross-contamination.
The fetch unit takes the next predicted address for each thread (which is usually just the next address) and then indexes into the ITLB and L1I cache. The ITLB is statically partitioned between both threads and has 128 entries for 4KB pages arranged with four way associativity. Each thread has 7 fully associative and dedicated entries for large pages (2M/4M) in addition to the shared small page entries. The instruction cache is 32KB and 4 way associative, with competitive sharing between threads. Each fetch into the cache grabs 16B of instructions which go into the pre-decode and fetch buffer. Then up to 6 instructions are sent from the buffer into the 18 entry instruction queue. The instruction queue in Core 2 is used as a loop cache (Loop Stream Detector or LSD), so that the instruction fetch unit can actually shut down for small loops. The instruction queue for Nehalem merely acts as a buffer for instructions before they are decoded, because the loop cache is located later in the pipeline in the decode stage.

The Front-end: Decode Phase

Once x86 macro-instructions have been fetched into the instruction queue, they are ready for decoding into micro-ops (uops), the internal RISC-like instructions supported by Intel’s architecture. Core 2 and Nehalem both have four decoders, one complex and three simple. A simple decoder can handle any x86 instructions which decode into a single uop, which includes most SSE instructions now. The complex decoder can decode instructions which maps to 1-4 uops and the microcode sequencer handles anything more complicated than that.

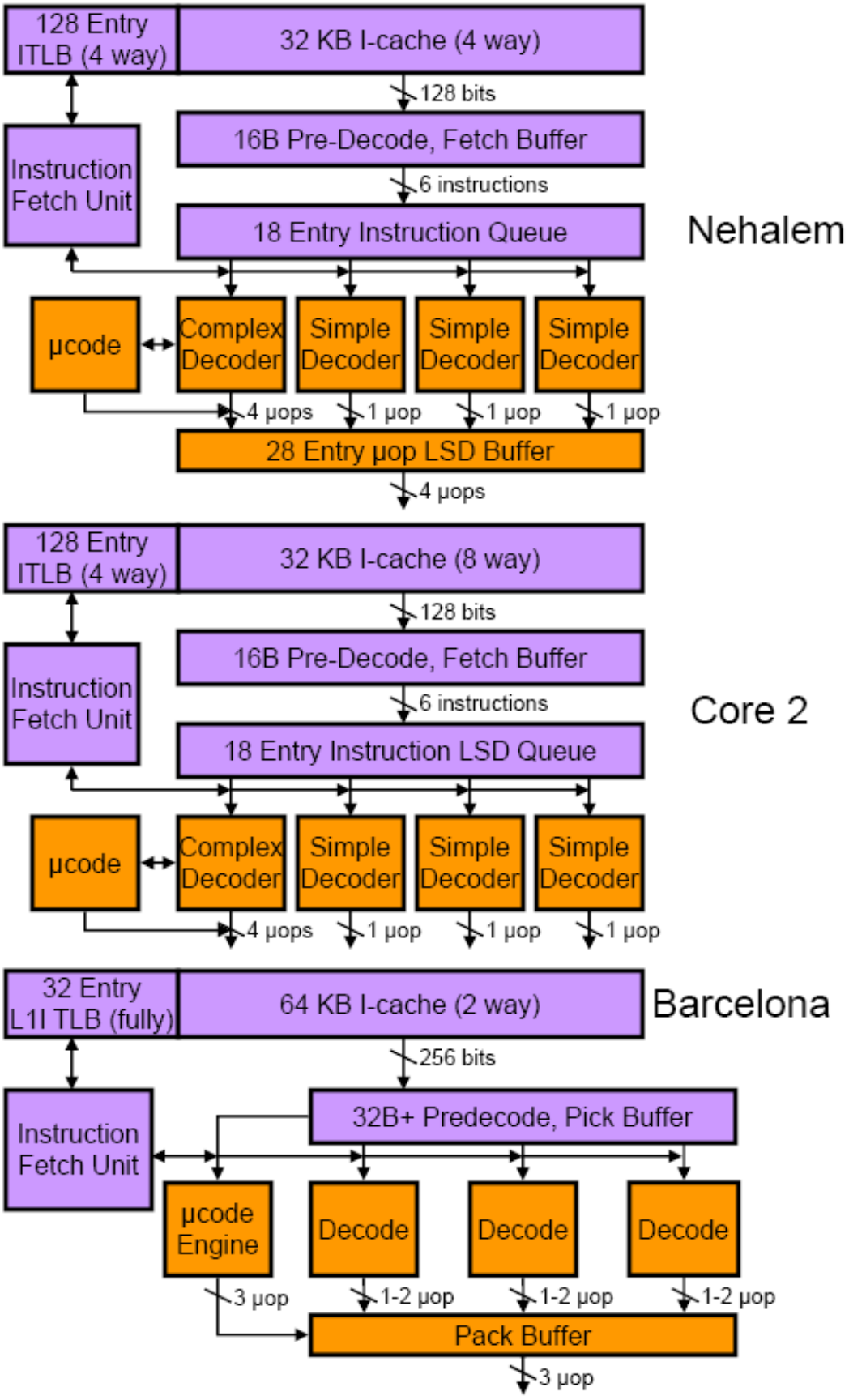
128 Entry ITLB (4 way)
32 KB I-cache (4 way)
128 bits
16B Pre-Decode, Fetch Buffer
Instruction Fetch Unit
6 instructions
18 Entry Instruction Queue
Nehalem
µcode
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
4 µops
1 µop
1 µop
1 µop
28 Entry µop LSD Buffer
4 µops
128 Entry ITLB (4 way)
32 KB I-cache (8 way)
128 bits
16B Pre-Decode, Fetch Buffer
Instruction Fetch Unit
6 instructions
18 Entry Instruction LSD Queue
Core 2
µcode
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
4 µops
1 µop
1 µop
1 µop
32 Entry L1I TLB (fully)
64 KB I-cache (2 way)
Barcelona
256 bits
Instruction Fetch Unit
32B+ Predecode, Pick Buffer
µcode Engine
Decode
Decode
Decode
3 µop
1-2 µop
1-2 µop
1-2 µop
Pack Buffer
3 µop

Figure 2 – Front-end Microarchitecture Comparison

Nehalem refines and improves the macro-op fusion already found in the previous generation. In 32 bit mode, the Core 2 could decode comparisons (CMP) or tests (TEST) and conditional branches (Jcc) into a single uop, CMP+JCC. This increased the decode bandwidth of the Core 2 and reduced the uop count, making the machine effectively wider. Macro-op fusion in Nehalem works with a wider variety of branch conditions, including JL/JNGE, JGE/JNL, JLE/JNG, JG/JNLE, so any of those, in addition to the previously handled cases will decode into a single CMP+JMP uop. Best of all, Nehalem's macro-op fusion operates in both 32 bit and 64 bit mode. This is essential, since the majority of servers and workstations are running 64 bit operating systems. Even modern desktops are getting close to the point where 64 bits makes a lot of sense, given the memory requirements of modern operating systems and current DIMM capacities and DRAM density. In addition to fusing x86 macro-instructions, the decoding logic can also also fuse uops, a technique first demonstrated with the Pentium M.

Once x86 instructions are decoded into uops, they go into a 28 entry uop buffer. As alluded to earlier, Core 2 had a Loop Stream Detector in the 18 entry instruction queue that acted as a cache for the instruction fetch unit. Nehalem improves on this concept by moving the LSD further down the pipeline across the decode stage into the new 28 entry uop buffer.

If a loop is less than 28 uops, then Nehalem can cache it in the LSD and issue into the out-of-order engine without using the instruction fetch unit or the decoders. This saves even more power than the Core 2 when using the LSD, by avoiding the decoders and more loops can be cached. Nehalem's 28 entry uop buffer can hold the equivalent of about 21-23 x86 macro-instructions based on our measurements from several games. The ratio of macro-ops/uops depends heavily on the workload, but in general Nehalem's buffer is 'larger' than that found in the Core 2.

One of the most interesting things to note about Nehalem is that the LSD is conceptually very similar to a trace cache. The goal of the trace cache was to store decoded uops in dynamic program order, instead of the static compiler ordered x86 instructions stored in the instruction cache, thereby removing the decoder and branch predictor from the critical path and enabling multiple basic blocks to be fetched at once. The problem with the trace cache in the P4 was that it was extremely fragile; when the trace cache missed, it would decode instructions one by one. The hit rate for a normal instruction cache is well above 90%. The trace cache hit rate was extraordinarily low by those standards, rarely exceeding 80% and easily getting as low as 50-60%. In other words, 40-50% of the time, the P4 was behaving exactly like a single issue microprocessor, rather than taking full advantage of it's execution resources. The LSD buffer achieves almost all the same goals as a trace cache, and when it doesn't work (i.e. the loop is too big) there are no extremely painful downsides as there were with the P4's trace cache.

After the LSD buffer, the last step in decode is the dedicated stack engine, which removes all the stack modifying uops. The stack modifying uops are all executed by a dedicated adder that writes to a speculative delta register in the front end, which is then occasionally synchronized with a renamed architectural register that contains the non-speculative value of the stack. After the stack

manipulating uops have been excised, the remaining uops head down into the out-of-order machine to be renamed, issued, dispatched and executed.

Out-of-Order Engine and Execution Units

The out-of-order engine for Nehalem was significantly augmented, both for general performance reasons, but also specifically to accommodate SMT. This is probably one of the largest architectural changes to the out-of-order engine since it was designed, since SMT requires that these resources be shared.

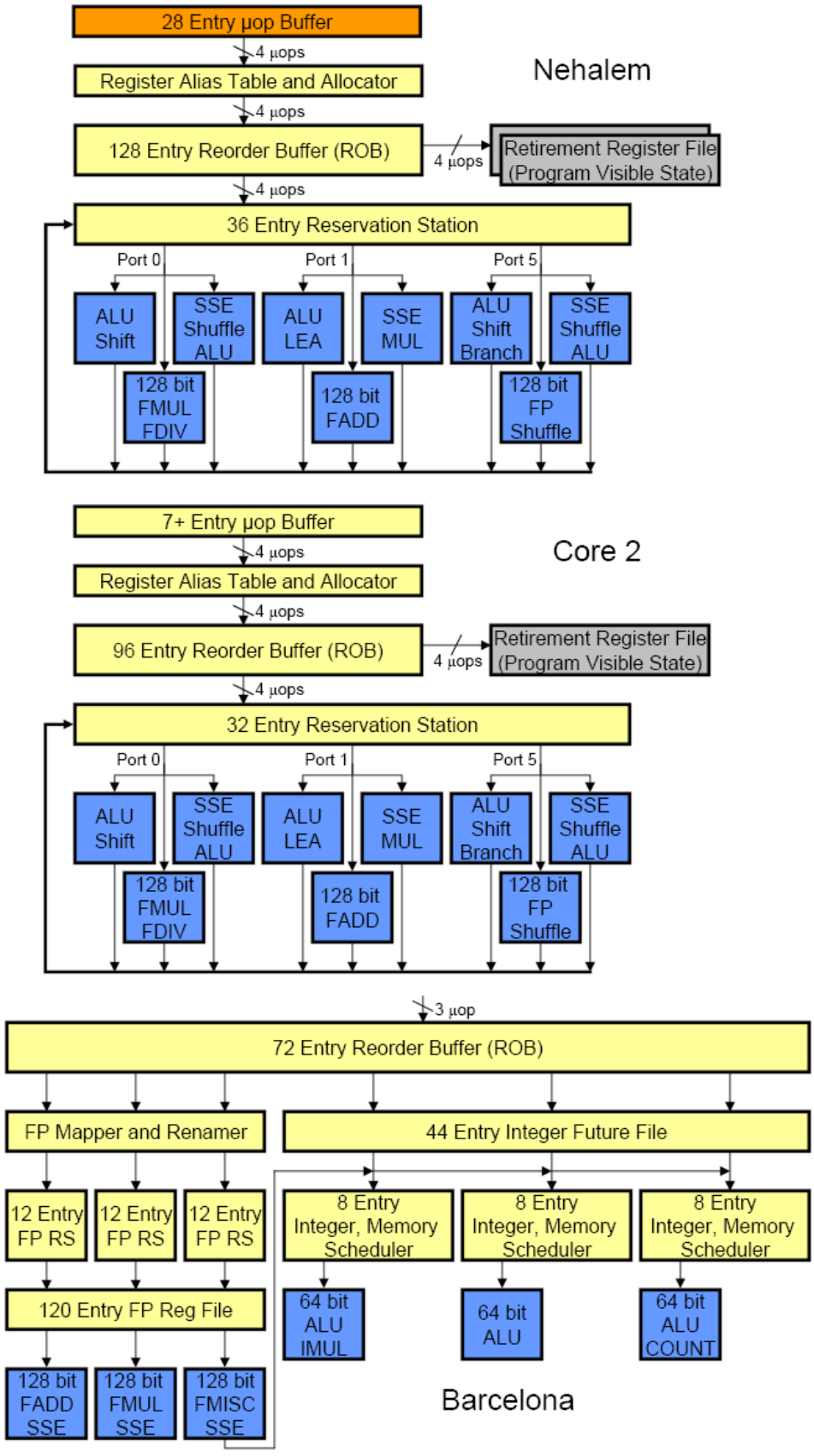
28 Entry µop Buffer
4 µops
Register Alias Table and Allocator
4 µops
128 Entry Reorder Buffer (ROB)
4 µops
Retirement Register File (Program Visible State)
4 µops
36 Entry Reservation Station
Port 0
Port 1
Port 5
ALU Shift
SSE Shuffle ALU
ALU LEA
SSE MUL
ALU Shift Branch
SSE Shuffle ALU
128 bit FMUL FDIV
128 bit FADD
128 bit FP Shuffle
Nehalem
7+ Entry µop Buffer
4 µops
Register Alias Table and Allocator
4 µops
96 Entry Reorder Buffer (ROB)
4 µops
Retirement Register File (Program Visible State)
4 µops
32 Entry Reservation Station
Port 0
Port 1
Port 5
ALU Shift
SSE Shuffle ALU
ALU LEA
SSE MUL
ALU Shift Branch
SSE Shuffle ALU
128 bit FMUL FDIV
128 bit FADD
128 bit FP Shuffle
Core 2
3 µop
72 Entry Reorder Buffer (ROB)
FP Mapper and Renamer
44 Entry Integer Future File
12 Entry FP RS
12 Entry FP RS
12 Entry FP RS
8 Entry Integer, Memory Scheduler
8 Entry Integer, Memory Scheduler
8 Entry Integer, Memory Scheduler
120 Entry FP Reg File
64 bit ALU IMUL
64 bit ALU
64 bit ALU COUNT
128 bit FADD SSE
128 bit FMUL SSE
128 bit FMISC SSE
Barcelona

Figure 3 – Out-of-Order Engine and Execution Unit Comparison

As with Core 2, the register alias table (RAT) points each architectural register into either the Re-Order Buffer (ROB) or the Retirement Register File (RRF) and holds the most recent speculative state (whereas the RRF holds the most recent non-speculative and committed state). The RAT can rename up to 4 uops each cycle, giving each one a destination register in the ROB. The renamed instructions then read their source operands and issue into the unified Reservation Station (RS), which is used by all instruction types.

The ROB was enlarged from 96 to 128 entries with Nehalem, and the RS grew from 32 to 36 entries. Both the ROB and RS are shared across the two threads, but using different policies. The ROB is statically partitioned between both threads, which allows each thread to speculate equally far through the instruction stream. In contrast, the RS is competitively shared, based on demand because often times a thread may stall while waiting for an operand from memory and use relatively few RS entries as a result – in which case it is better for that thread to yield up RS entries to the more active thread. Any instructions in the RS which have all their operands ready are dispatched to the execution units, which are largely unchanged from the Core 2 and unaffected by SMT, except for an increase in utilization.

A Cache Extravaganza

While almost every aspect of Nehalem has been enhanced, the memory subsystem received the most dramatic overhaul – largely because it is very closely coupled with the overall system architecture. Almost every part of the memory system has been refined, improved or otherwise changed to support greater parallelism as shown in Figure 5 below.

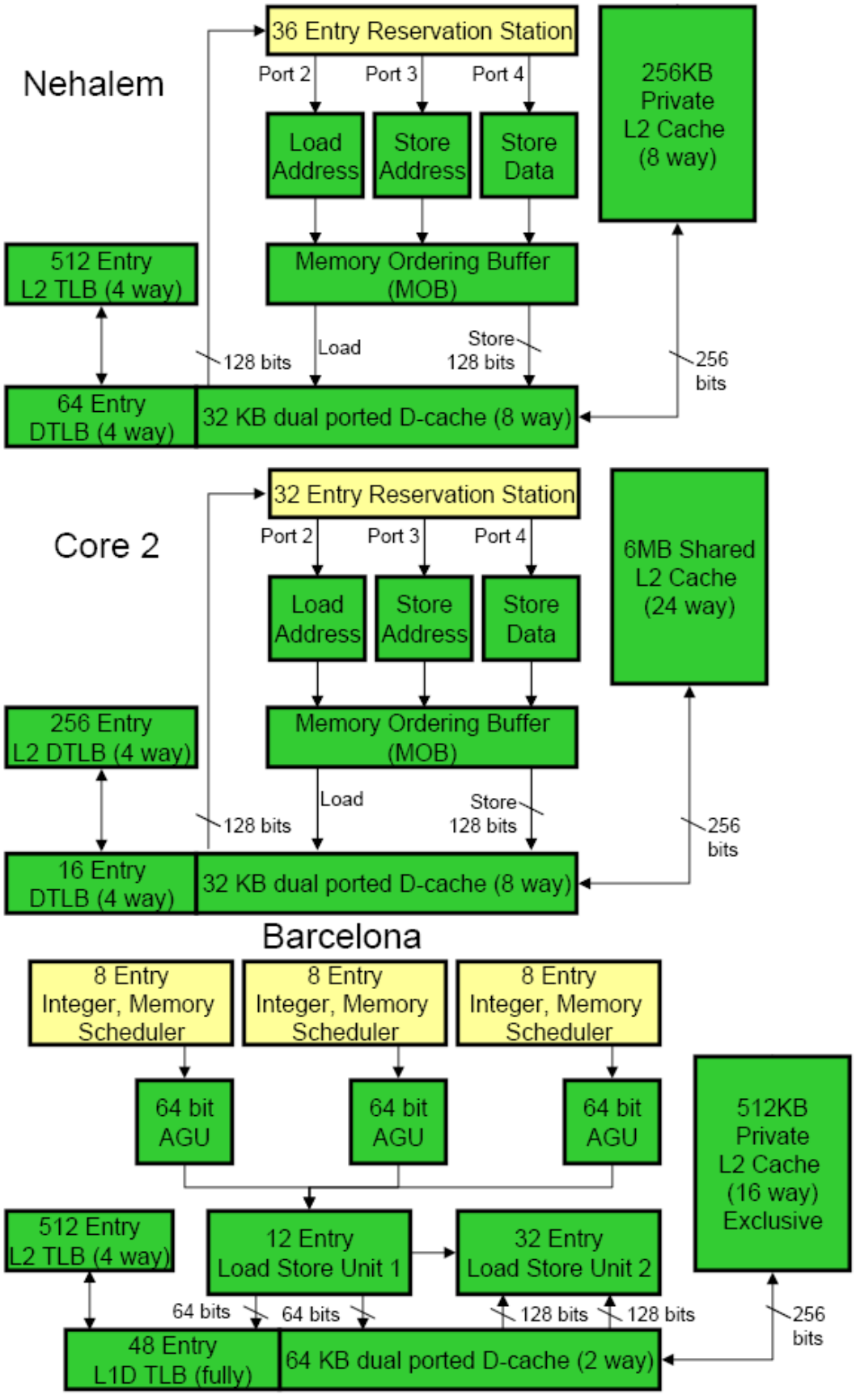
Nehalem
36 Entry Reservation Station
Port 2
Port 3
Port 4
Load Address
Store Address
Store Data
256KB Private L2 Cache (8 way)
512 Entry L2 TLB (4 way)
Memory Ordering Buffer (MOB)
128 bits
Load
Store 128 bits
256 bits
64 Entry DTLB (4 way)
32 KB dual ported D-cache (8 way)
Core 2
32 Entry Reservation Station
Port 2
Port 3
Port 4
Load Address
Store Address
Store Data
6MB Shared L2 Cache (24 way)
256 Entry L2 DTLB (4 way)
Memory Ordering Buffer (MOB)
128 bits
Load
Store 128 bits
256 bits
16 Entry DTLB (4 way)
32 KB dual ported D-cache (8 way)
Barcelona
8 Entry Integer, Memory Scheduler
8 Entry Integer, Memory Scheduler
8 Entry Integer, Memory Scheduler
64 bit AGU
64 bit AGU
64 bit AGU
512KB Private L2 Cache (16 way) Exclusive
512 Entry L2 TLB (4 way)
12 Entry Load Store Unit 1
32 Entry Load Store Unit 2
64 bits
64 bits
128 bits
128 bits
256 bits
48 Entry L1D TLB (fully)
64 KB dual ported D-cache (2 way)

Figure 4 – Memory Subsystem Comparison

The changes start early on, as Nehalem increases the number of in-flight loads and stores by 50%. As shown above, the load buffer now holds 48 entries up from 32 and the store buffer increased slightly more to 32 entries from 20 in the Core 2. One of the reasons to enlarge the load and store buffers is that they must be shared by both threads; in this case they are statically partitioned.

From the load and store buffers, memory operations proceed to access the cache hierarchy, which has been totally redone from top to bottom. As with the P4, both the caches and TLBs are dynamically shared between threads based upon observed behavior. Nehalem's L1D cache has retained the same size and associativity as the previous generation, but the latency increased from 3 to 4 cycles to accommodate timing constraints. As previously mentioned, each L1D cache can support more outstanding misses (up to 10) to take advantage of the extra memory bandwidth.

The remainder of the cache hierarchy in Nehalem is a substantial departure from the design found in Core 2. The last level cache in Core 2 was the L2, which was shared between two cores to reduce coherency traffic and weighed in at 6MB with 24 way associativity and a very low load to use latency of 14-15 cycles. Nehalem's cache hierarchy has been extended to three levels, with the first two levels staying relatively small and private to each core, while the L3 cache is much larger and shared between all cores.

Each core in Nehalem has a private unified 256KB L2 cache that is 8 way associative and provides extremely fast access to data and instructions. The load to use latency was not precisely disclosed, but Intel architect Ronak Singhal indicated that it was less than 12 cycles. The L2 cache is neither inclusive nor exclusive with respect to the L1D cache and can sustain 16 misses in-flight. Like the Core 2, Nehalem can transfer data between the private caches of two cores, although not at full transfer rates.

Nehalem is the first mainstream Intel processor to pack in a giant shared L3 cache. Plenty of Intel server designs have featured L3 caches; every Itanium and many Xeon MP's since 2003 (in fact, Potomac in 2004 featured an identically sized L3 cache). Neither AMD nor Intel ever used a three level design for mainstream products until the advent of monolithic quad-core designs, starting with Barcelona last year.

Nehalem's 8MB and 16 way associative L3 cache is inclusive of all lower levels of the cache hierarchy and shared between all four cores. Although Intel has not discussed the physical design of Nehalem at all, it appears that the L3 cache sits on a separate power plane than the cores and operates at an independent frequency. This makes sense from both a power saving and a reliability perspective, since large caches are more susceptible to soft errors at low voltage. As a result, the load to use latency for Nehalem varies depending on the relative frequency and phase alignment of the cores and the L3 itself and the latency of arbitration for access to the L3. In the best case, i.e. phase aligned operation and frequencies that differ by an integer multiple, Nehalem's L3 load to use latency is somewhere in the range of 30-40 cycles according to Intel architects.The advantage of an inclusive cache is that it can handle almost all coherency traffic without disturbing the private caches for each individual-core. If a cache access misses in the L3, it cannot be present in any of the L2 or

L1 caches of the cores. On the other hand, Nehalem's L3 also acts like a snoop filter for cache hits. Each cache line in the L3 contains four "core valid" bits denoting which cores may have a copy of that line in their private caches. If a "core valid" bit is set to 0, then that core cannot possibly have a copy of the cache line – while a "core valid" bit set to 1 indicates it is possible (but not guaranteed) that the core in question could have a private copy of the line. Since Nehalem uses the MESIF cache coherency protocol, as discussed previously, if two cores have valid bits, then the cache line is guaranteed to be clean (i.e. not modified). The combination of these two techniques lets the L3 cache insulate each of the cores from as much coherency traffic as possible, leaving more bandwidth available for actual data in the caches.

To be fair, inclusive caches are not the only way of doing this, and they also have some overhead. A non-inclusive cache can achieve the same benefits by simply replicating the tag files for all private caches with the last level cache and simultaneously checking both the last level tags and the private cache tags on any access. Inclusive caches are forced by design to replicate data, which implies certain relationships between the sizes of the various levels of the cache. In the case of Nehalem, each core contains 64KB of data in the L1 caches and 256KB in the L2 cache (there may or may not be data that is in both the L1 and L2 caches). This means that 1-1.25MB of the 8MB L3 cache in Nehalem is filled with data that is also in other caches. What this means is that inclusive caches should only really be used where there is a fairly substantial size difference between the two levels. Nehalem has about an 8X difference between the sum of the four L2 caches and the L3, while Barcelona's L3 cache is the same size as the total of the L2 caches.

Nehalem's cache hierarchy has also been made more flexible by increasing support for unaligned accesses. Previous generations of Intel chips always had two instructions for 16B SSE loads and stores – one for data that is guaranteed to be aligned to cache line (16B) boundaries and one that is for unaligned. The latter was avoided by compilers because it ran slower with less throughput (even if the data was actually aligned). Nehalem changes the situation by making aligned accesses and unaligned accesses that touch aligned data the same latency and throughput, and also improving the performance for unaligned accesses which touch unaligned data. As a result, an unaligned SSE load or store will always have the same latency as an aligned memory access, so there is no particular reason to use aligned SSE memory accesses.

TLBs, Page Tables and Synchronization

Along with a new cache hierarchy, Nehalem also features some important changes to the TLB hierarchy, which caches the virtual to physical address mappings. Core 2 had a very interesting TLB arrangement. The L1 DTLB (which Intel referred to as the micro-TLB sometimes) was extremely small and only used for loads; it featured 16 entries for 4KB page and 16 entries for larger page, and each was 4 way associative. The L2 DTLB was larger and serviced all memory accesses (loads that misses in the L1 DTLB and all stores). It offered 256 entries for 4KB pages and 32 entries for larger

(2M/4M) pages, and again, both were 4 way associative. Since the cache hierarchy contains much more data for Nehalem, the TLBs also needed to be enlarged.

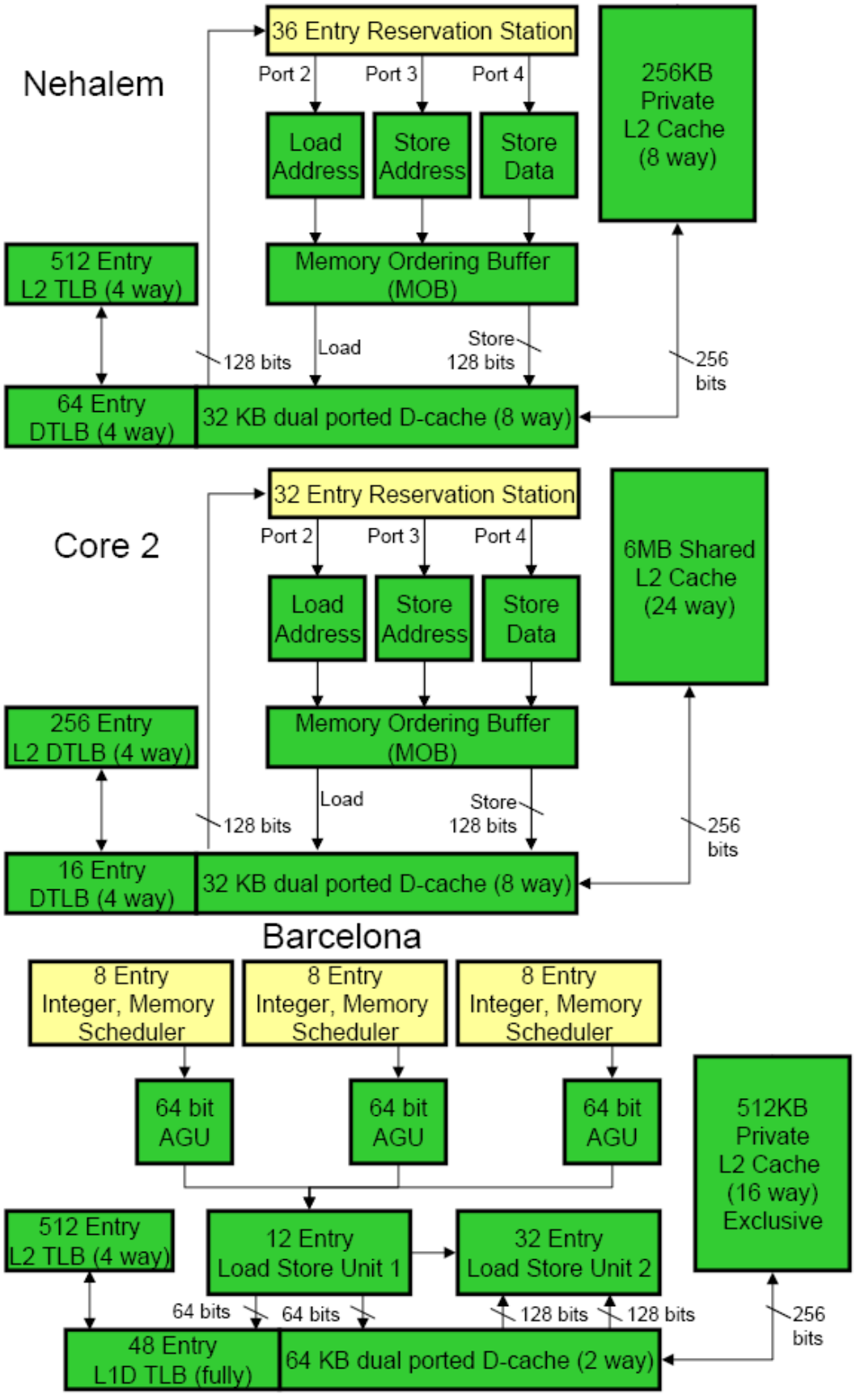
Nehalem
36 Entry Reservation Station
Port 2
Port 3
Port 4
Load Address
Store Address
Store Data
256KB Private L2 Cache (8 way)
512 Entry L2 TLB (4 way)
Memory Ordering Buffer (MOB)
128 bits
Load
Store 128 bits
256 bits
64 Entry DTLB (4 way)
32 KB dual ported D-cache (8 way)
Core 2
32 Entry Reservation Station
Port 2
Port 3
Port 4
Load Address
Store Address
Store Data
6MB Shared L2 Cache (24 way)
256 Entry L2 DTLB (4 way)
Memory Ordering Buffer (MOB)
128 bits
Load
Store 128 bits
256 bits
16 Entry DTLB (4 way)
32 KB dual ported D-cache (8 way)
Barcelona
8 Entry Integer, Memory Scheduler
8 Entry Integer, Memory Scheduler
8 Entry Integer, Memory Scheduler
64 bit AGU
64 bit AGU
64 bit AGU
512KB Private L2 Cache (16 way) Exclusive
512 Entry L2 TLB (4 way)
12 Entry Load Store Unit 1
32 Entry Load Store Unit 2
64 bits
64 bits
128 bits
128 bits
256 bits
48 Entry L1D TLB (fully)
64 KB dual ported D-cache (2 way)

Figure 4 – Memory Subsystem Comparison

Nehalem replaces the pre-existing TLBs in Core 2 with a true two level TLB hierarchy that is dynamically allocated between the active thread contexts for SMT. The first level DTLB in Nehalem services all memory accesses and contains 64 entries for 4KB pages and 32 entries for larger 2M/4M pages and keeps the 4 way associativity. Nehalem also features a new second level unified (i.e. both instructions and data) TLB that contains 512 entries for small pages only, and is again 4 way associative.

One of the stark differences between Nehalem and the Core 2 TLBs is the degree to which they cover the caches. In Core 2, there was 6MB of cache and the TLBs could translate 2176KB of memory using the smaller 4KB pages (most applications do not use large pages), effectively covering half or a third of the full L2 cache (depending on whether we are discussing Merom or Penryn). In contrast, each Nehalem core has 576 entries for the small pages and 2304 for the whole chip. This many TLB entries can translate 9216KB – more than enough to contain the whole 8MB L3 cache using small pages alone.

Nehalem's TLB entries have also changed subtly by introducing a "Virtual Processor ID" or VPID. Every TLB entry caches a virtual to physical address translation for a page in memory, and that translation is specific to a given process and virtual machine. Intel's older CPUs would flush the TLBs whenever the processor switched between the virtualized guest and the host instance, to ensure that processes only accessed memory they were allowed to touch. The VPID tracks which VM a given translation entry in the TLB is associated with, so that when a VM exit and re-entry occurs, the TLBs do not have to be flushed for safety. Instead if a process tries to access a translation that it is not associated with, it will simply miss in the TLB, rather than making an illegal access to the page tables. The VPID is helpful for virtualization performance by lowering the overhead of VM transitions; Intel estimates that the latency of a round trip VM transition in Nehalem is 40% compared to Merom (i.e. the 65nm Core 2) and about a third lower than the 45nm Penryn.

Another virtualization tweak in Nehalem is the Extended Page Tables, which actually eliminates many VM transitions (rather than just lowering the latency as the VPID does). The normal page tables map guest virtual addresses to guest physical addresses; however, for a virtualized system, there is also a translation from guest physical to host physical addresses. The EPT manages those mappings from guest physical to host physical. When a page fault happens on the guest physical to host physical mapping, Nehalem will simply walk the EPTs, whereas earlier Intel designs (and AMD designs before Barcelona) would need the hypervisor to service the page fault. This eliminates a lot of unnecessary VM exits.

Nehalem also lowers the latency for synchronization primitives such as LOCK, XCHG and CMPXCHG, which are necessary for multi-threaded programming. Intel claims that the latency for LOCK CMPXCHG instructions (which serializes the whole pipeline) is 20% of what it was for the P4 (which was absolutely horrible) and about 60% of the Core 2. While the latency is lower, the behavior is still similar to prior generaitons; Lock instructions are not pipelined, although younger operations can execute ahead of a LOCK instruction.

Flexible Productization

While Nehalem packs in an awful lot of improvements, the reality is that not all market segments really need everything. Take the integrated memory controller – it is an absolute necessity for servers, but might not even make sense for a low-end desktop. This is where the extreme modularity of Nehalem comes into play. The Core 2 was really a take it or leave it proposition; Merom (mobile), Conroe (desktop) and Woodcrest (server) and even Tigerton (MP server) were all more or less the same. For Nehalem, there will be a variety of different proliferations that are specifically tuned across a wide range of parameters to fit each market segment, just as Intel's chipsets were previously used to differentiate across the market.

Unsurprisingly then, most of the differentiation for proliferations of Nehalem will focus on changing the "un-core" features of Nehalem. Some of the un-core elements that are flexible are:

- Core count
- Simultaneous Multi-Threading
- L3 cache size
- Number of QPI links
- Advanced routing features like directories
- Integrated or discrete memory controller
- Type of memory
- Number of memory channels
- Integrated or discrete graphics
- Power and clocking
- Virtual and physical addressing

Of course, throughout this article, we have referred to Nehalem as if it was a single implementation, but that is not the case. Currently, there are six variants of Nehalem slated to be released in the future. Gainestown is a Xeon DP product that is more or less the baseline for all Nehalem implementations as described in this article. It is a quad-core, 8MB L3 cache design with 2 QPI links and a triple channel DDR3 integrated memory controller. Bloomfield is more or less the same device, but targeted at the high-end desktop market, rather than servers. Both Gainestown and Bloomfield will come out towards the end of this year, likely in Q4 – although Bloomfield may not have SMT enabled.

Next up are Auburndale, Havendale and Clarksfield, which are a dual-core desktop, a dual-core mobile and a quad-core mobile processor. These three products are where the bulk of the differentiation will be needed. Power and clocking features will be used to distinguish desktop and mobile processors, and may even be used within the mobile market to stratify ultra-mobile parts versus mainstream parts. In general, the dual-core products (note all dual-core codenames end in "dale", while quad-cores end in "field" or "town") will continue to be the mainstream for both desktop and especially notebook systems (for power reasons). These mainstream CPUs will ship with smaller caches, perhaps 2-4MB, fewer memory channels (probably 2) and SMT disabled. There will also be a lot of options with respect to the chipset and graphics. Intel will aggressively push options

that integrate the chipset and graphics in the same package – but OEMs are very likely to want some CPUs to use with third party chipsets from NVIDIA, SiS and the like (although how NVIDIA's QPI license plays out is unclear right now), as well as some with discrete graphics. The quad-core Clarksfield is a little harder to pin down, as that is probably aimed at mobile gaming or workstations – where integrated graphics makes little sense, but an integrated memory controller and large L3 cache would be a boon.

The last Nehalem based product to come to market will be Beckton (or Nehalem EX), the Xeon MP variant, which is slated for 2009. At this point in time, very little has been said publicly about the product. However, it is possible to speculate with some degree of accuracy. Considering Dunnington is a 6-core design, Beckton will be at least 6 cores, but more likely 8. Since Beckton will contain at least as many cores, it will probably need at least as much cache as Dunnington. Beckton will have at least 3 and a half QPI links, so that the CPUs in a four socket system will be fully connected, but they may very well include more links to encourage further scalability. Beckton may also raise the virtual and physical addressing limits, based upon DRAM density, DIMM capacity and market demand for large memory configurations.

Performance, Conclusions and References

While Intel has not disclosed the target frequency or TDP for Nehalem, it is reasonable to expect a slight loss in frequency, and a larger power envelop. The thermal envelop for Nehalem will increase, since the power budget allocated for the northbridge in a traditional system will become available for use. However, the package level thermal limit will probably continue to be under 150W, since that is the limit for affordable cooling solutions. Judging by Tukwila, incorporating a triple channel memory controller and 2 QPI links will probably cost around 12-20W of power [5]. Within those limits, Nehalem will not be able to increase frequency and may actually decrease frequency slightly. Nehalem's performance will come from refining the already impressive Core 2 microarchitecture and coupling it with a high performance memory and I/O system with minimal frequency impact. In workloads that are not particularly bandwidth dependent, such as general integer applications, Nehalem will provide a moderate boost over the previous generation. The performance gains will largely come from the integrated memory controller, microarchitectural innovation and circuit techniques (the latter was not discussed by Intel at IDF and will probably be saved for later disclosure). For floating point and HPC workloads that are typically bandwidth bound, Nehalem will be nothing short of a miracle – with performance gains of 2X or better. Commercial server workloads such as OLTP databases, decision support and virtualization will certainly benefit from more bandwidth and lower latency as well, but not to the same extent as HPC or floating point applications. Of course, when thinking about performance it is essential to also keep time in mind – Beckton will be about a year behind Gainestown and Bloomfield.

When the first implementation of Nehalem comes to the market, it will be competing primarily against AMD's 65nm Barcelona, both of which are shown below in Figure 5. AMD will also have the 45nm Shanghai and Griffin a 65nm mobile device in their product porftolio and Intel will have older 45nm Core 2 based products as well. Core 2 already has an IPC edge over most alternatives from AMD, and Nehalem will widen this gap. That being said, this is the first time in history that AMD has been able to muster a complete platform under one roof. Certainly in the desktop and mobile market, that will help AMD create a more attractive offering relative to Intel, without getting into a performance race – one that they would likely lose. Or perhaps more to the point, AMD will try and focus on graphics performance rather than CPU performance. While this may work for the destkop and mobile markets, that is decidedly not the case for servers and workstations, where the platform matters, but having the fastest CPU probably matters more.

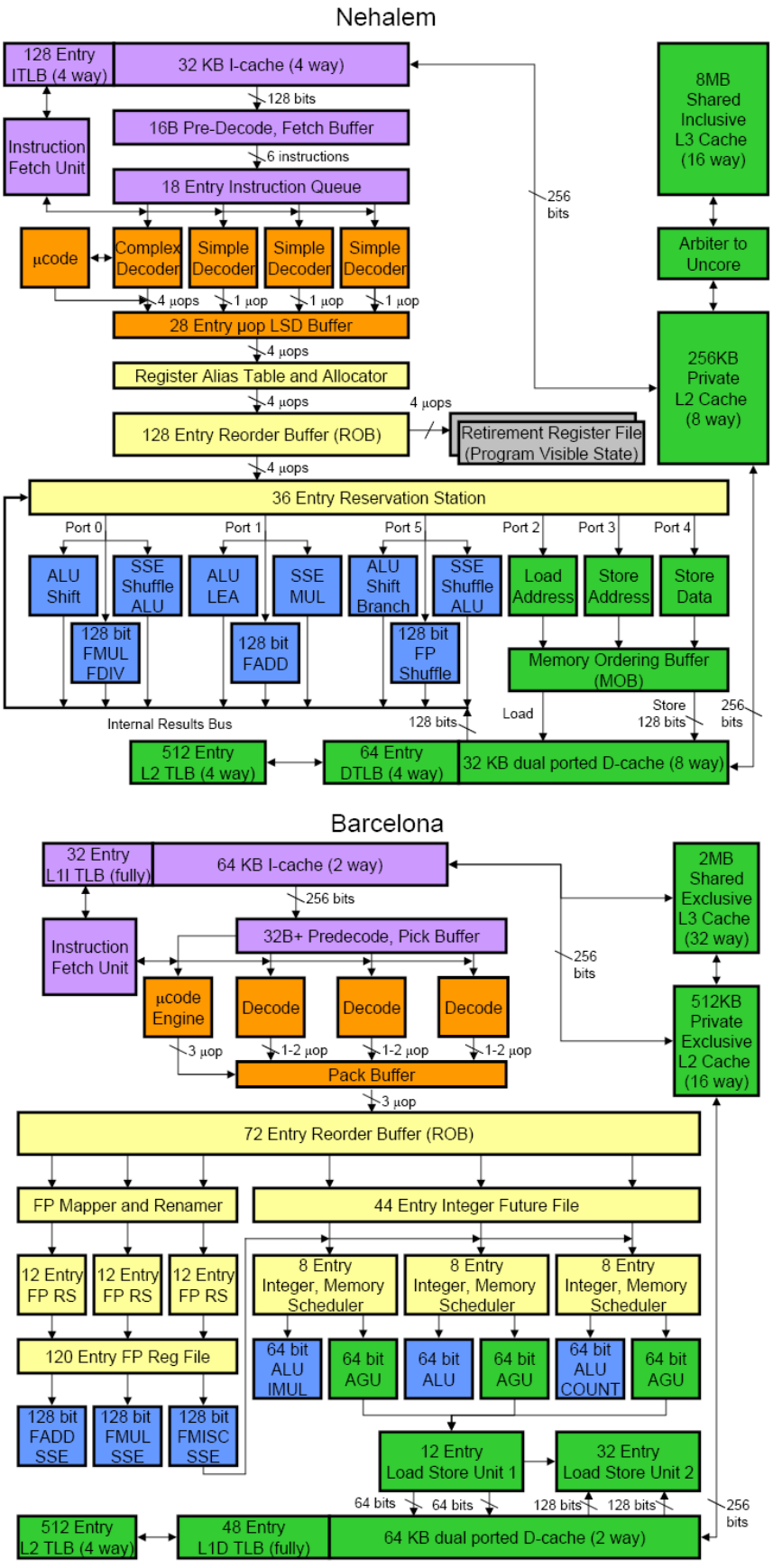
Nehalem
128 Entry ITLB (4 way)
32 KB I-cache (4 way)
128 bits
16B Pre-Decode, Fetch Buffer
6 instructions
Instruction Fetch Unit
18 Entry Instruction Queue
256 bits
8MB Shared Inclusive L3 Cache (16 way)
Arbiter to Uncore
μcode
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
4 μops
1 μop
1 μop
1 μop
28 Entry μop LSD Buffer
4 μops
Register Alias Table and Allocator
4 μops
4 μops
128 Entry Reorder Buffer (ROB)
Retirement Register File (Program Visible State)
256KB Private L2 Cache (8 way)
4 μops
36 Entry Reservation Station
Port 0
Port 1
Port 5
Port 2
Port 3
Port 4
ALU Shift
SSE Shuffle ALU
ALU LEA
SSE MUL
ALU Shift Branch
SSE Shuffle ALU
Load Address
Store Address
Store Data
128 bit FMUL FDIV
128 bit FADD
128 bit FP Shuffle
Memory Ordering Buffer (MOB)
Internal Results Bus
128 bits
Load
Store 128 bits
256 bits
512 Entry L2 TLB (4 way)
64 Entry DTLB (4 way)
32 KB dual ported D-cache (8 way)
Barcelona
32 Entry L1I TLB (fully)
64 KB I-cache (2 way)
256 bits
2MB Shared Exclusive L3 Cache (32 way)
Instruction Fetch Unit
32B+ Predecode, Pick Buffer
256 bits
μcode Engine
Decode
Decode
Decode
3 μop
1-2 μop
1-2 μop
1-2 μop
Pack Buffer
512KB Private Exclusive L2 Cache (16 way)
3 μop
72 Entry Reorder Buffer (ROB)
FP Mapper and Renamer
44 Entry Integer Future File
12 Entry FP RS
12 Entry FP RS
12 Entry FP RS
8 Entry Integer, Memory Scheduler
8 Entry Integer, Memory Scheduler
8 Entry Integer, Memory Scheduler
120 Entry FP Reg File
64 bit ALU IMUL
64 bit AGU
64 bit ALU
64 bit AGU
64 bit ALU COUNT
64 bit AGU
128 bit FADD SSE
128 bit FMUL SSE
128 bit FMISC SSE
12 Entry Load Store Unit 1
32 Entry Load Store Unit 2
64 bits
64 bits
128 bits
128 bits
256 bits
512 Entry L2 TLB (4 way)
48 Entry L1D TLB (fully)
64 KB dual ported D-cache (2 way)

Figure 5 – Nehalem and Barcelona Comparison

Conclusions

Nehalem is a once in a decade event for Intel – the opportunity to redefine their system architecture so that it will continue to scale for the next ten years (or more). This is an extremely significant change for Intel and the industry as a whole as systems will need to be redesigned across the board. Intel's integrated memory controller and on-die routing are relatively late compared to the first efforts by the Alpha team at DEC and AMD's K8. This is likely partially by choice, but also simply due to sheer inertia. AMD had no presence in the server market before they made the shift, which meant that relatively few vendors were bothered by it. In contrast, Intel was held back by the millions of existing FSB based products they were shipping every year and by industry players who would rather defer the transition a little longer. While this posed some competitive problems for Intel since 2003, there are some advantages – by the time Intel started down the path of greater integration their design teams had a chance to learn and improve upon the mistakes of others. This bodes well for Intel's new system architecture as it appears to be well thought out – but the proof will come later this year along with the first Nehalem based products.

Nehalem is also the first opportunity for a major overhaul of the Core 2 microarchitecture that first shipped in June, 2006. Intel's Hillsboro design team took great care to create a flexible and scalable processor that can be adapted and customized to all of the market segments that it will serve – which is a far different approach than Intel's Haifa design team took with Merom. The Merom team was under an incredible amount of pressure from Intel's management and the extra flexibility would not have been nearly as valuable as the timely delivery of the product. Moreover, the extra flexibility and customization is needed for Nehalem because of the system level integration of the microprocessor itself.

It seems as though almost every part of Nehalem's pipeline has been tweaked, extended or somehow refined, except for the functional units. The bulk of the changes were made to the memory pipeline, to complement the changes in system architecture. However, the single biggest change and performance gain (made in the core) is simultaneous multi-threading which could improve server workloads anywhere in the range of 10-40%. The other techniques that Intel's architects have implemented are much harder to peg down precisely, but will tend to apply more evenly across all workloads. Ultimately, it will be very interesting to see how each technique will impact performance, but that information will not be forthcoming till later this year (or perhaps next year).

While most of Nehalem has been discussed by Intel at this IDF in Shanghai, there is still a great deal of information to look forward to – and not just the performance numbers. Intel has not discussed any of the circuit level techniques used in Nehalem, probably both for competitive reasons, but also to get a little more mindshare at ISSCC or Fall IDF. When AMD went to a quad-core, they radically changed their circuit design going from a single clock distribution network to five, with three different voltage planes and dynamic FIFOs between the clock domains. It is guaranteed that Intel will take similar measures to control power and increase performance. The most interesting question is what

sort of dynamic clocking techniques will Intel use to improve single threaded performance. It's no secret that Penryn already can dynamically increase frequency and voltage on one core when the other is idle, and it is clear that this would be even more advantageous in a quad-core design which would have much more thermal headroom when running single threaded code. All these questions and more will give us something to look forward to later in the year when Gainestown arrives and Intel goes into greater detail on the circuit techniques used to deliver Nehalem.

References

[1] Singhal, Ronak. Inside Intel Next Generation Nehalem Microarchitecture. Intel Developers Forum, April 1, 2008.

[2] Kanter, David. The Common System Interface: Intel's Future Interconnect. httpp://www.realworldtech.com/page.cfm?ArticleID=RWT082807020032. August 28, 2007.

[3] Bannon, Peter. Alpha 21364 (EV7). http://www.eecs.umich.edu/vlsi_seminar/f01/slides/bannon.pdf

[4] Pase, D. and Eckl, M. Performance of the AMD Opteron LS21 for IBM BladeCenter. ftp://ftp.software.ibm.com/eserver/benchmarks/wp_ls21_081506.pdf. August 2006.

[5] DeMone, Paul. Alpha EV8 (Part 2): Simultaneous Multi-Threat. http://www.realworldtech.com/page.cfm?ArticleID=RWT122600000000. December 26, 2000.

[6] Stackhouse, B. et al. A 65nm 2-Billion-Transistor Quad-Core Itanium Processor. ISSCC Digest of Technical Papers, p92-93, February 2008.

Specal thanks for this article go to George Alfs, Nick Knupffer, Ronak Singhal, Bob Valentine, the rest of the Nehalem team and everyone else who helped me along with this article.