Intel’s Merom Unveiled

Introduction

Intel’s Developer Forum (IDF) for Spring 2006 has a rather obvious focus: the announcement of Intel’s next generation microarchitecture. Intel’s new core will almost certainly leapfrog arch-rival AMD for integer and commercial computing performance. That alone should make this IDF one of the most exciting in the last several years.

The most recent x86 microarchitecture was the Yonah core out of Israel, which is loosely derived from the venerable Pentium Pro (P6) design. Unfortunately, Intel opted to disclose fewer details for Yonah than for the 180nm Pentium 4, which was announced with much fanfare and thorough documentation. While Intel will withhold some details about the Next Generation Microarchitecture for later publication, this will be the first opportunity for the industry to get a detailed look at a new x86 microarchitecture.

Intel’s new core is known by a variety of names. The family of processors was previously described as the Next-Generation Micro-Architecture (NGMA), but is being productized as the “Core Microarchitecture”. As Intel roadmaps have indicated, Merom is the laptop variant, Conroe is for the desktop and Woodcrest is aimed at the single and dual socket server market. For the purposes of this article we will refer to the new microarchitecture as Merom.

This article will provide an overview of the Merom microarchitecture, and a detailed analysis of key sections and features in the context of Intel’s existing P4 and Yonah MPUs.

Rounding the Dual Core (Wood) Crest

While Yonah was truly the first ground up dual core design, featuring a shared L2 cache, some of the benefits were lost on a mobile platform. The true benefit of a good Chip Multi-Processing (CMP) design is that it significantly reduces cache coherency snooping, which is a major issue for workstations and servers, but less so for mobile and desktop devices. In that regard, Woodcrest will substantially improve over prior server chips, which as noted in an earlier article, do not share any cache. Furthermore, Intel has hinted that Merom can transfer data directly between the two L1D caches. The architects declined to disclose any further details on this feature, other than to say that it exists. Figure 1 below shows a high level system comparison between Yonah, Merom, and Dempsey.

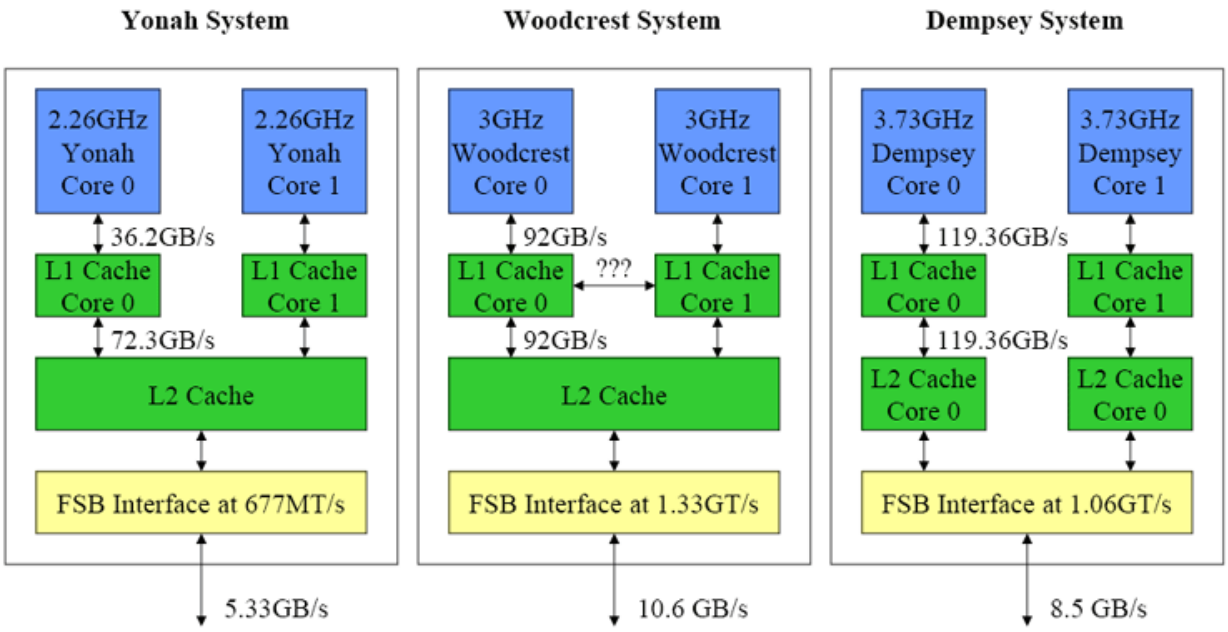


Figure 1 – Architectural Comparison of Intel MPUs

From a high level perspective, Merom has inherited the CMP hierarchy of Yonah, using a shared last level cache and bus interface. At the same time, the bandwidth for Merom is closer to that of Dempsey than Yonah; the on-chip caches run substantially faster and the front-side bus runs at twice the bandwidth. As noted above, these improvements help multi-socket systems more than desktop and mobile machines.

Intel will release Woodcrest products first, in early Q3, followed by Conroe later in Q3 and Merom in Q4 (or possibly Q1 of ’07). These release dates are more of a reflection of Intel’s competitive position in each market and their margins, than anything else. The mobile market has the highest margins for Intel, although the least competition, thanks to Intel’s aggressive marketing of the excellent Centrino platform using Yonah. This is not expected to change in the near future, and Yonah was released very recently in January. The server market has the next highest margins, and Intel’s offerings are lagging behind AMD by a fair bit. The desktop market is the least lucrative of all three, and is moderately close in terms of performance. Looking at all these variables, it is easy to see how and why Intel picked their release schedules. The server offerings need the most improvement, then the desktop market and last the mobile market. Moreover, a short product life cycle for Yonah would be financially and strategically undesirable.

Where Ever I Merom

Intel’s Merom is a dual core, 64 bit, 4 issue superscalar, moderately pipelined, out-of-order MPU, implemented in a 65nm high performance bulk process. The processor can address 36 bits of physical memory and 48 bits of virtual memory and supports all of Intel’s *Ts. Each CPU is fed by a 32KB L1I cache, a dual ported 32KB L1D cache, with a shared 4MB L2 cache. Merom variants

currently clock at up to 3.0GHz, but will probably scale to 3.33GHz. Each product family has a max TDP, Merom at 35W, Conroe 65W and Woodcrest at 80W. However, lower power parts will be available based on customer demand and usage. For example, LV Woodcrest products are targeted for blade systems, and sacrifice clockspeed to achieve a 40W TDP. Figure 3 below shows the Merom microarchitecture.

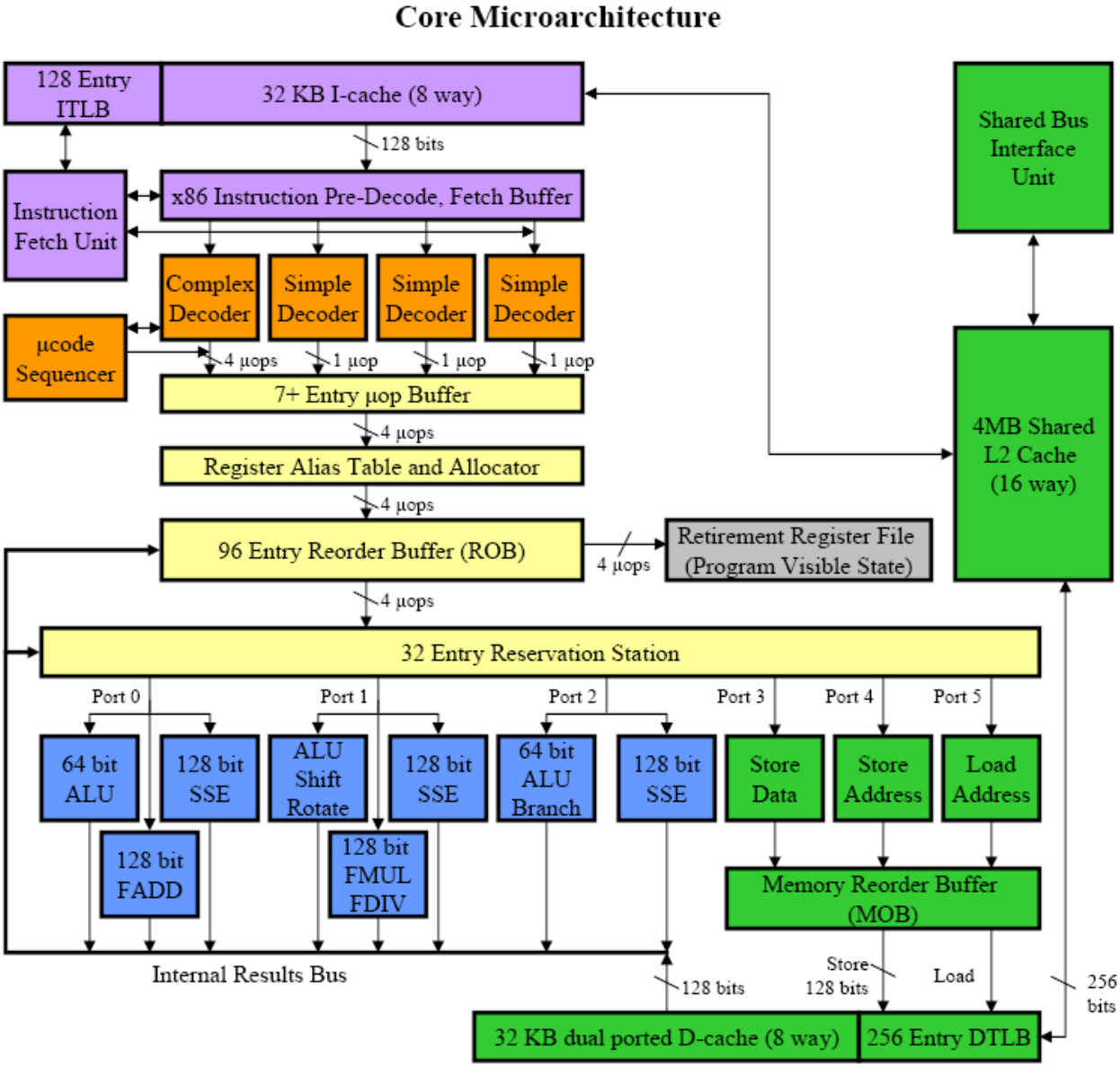
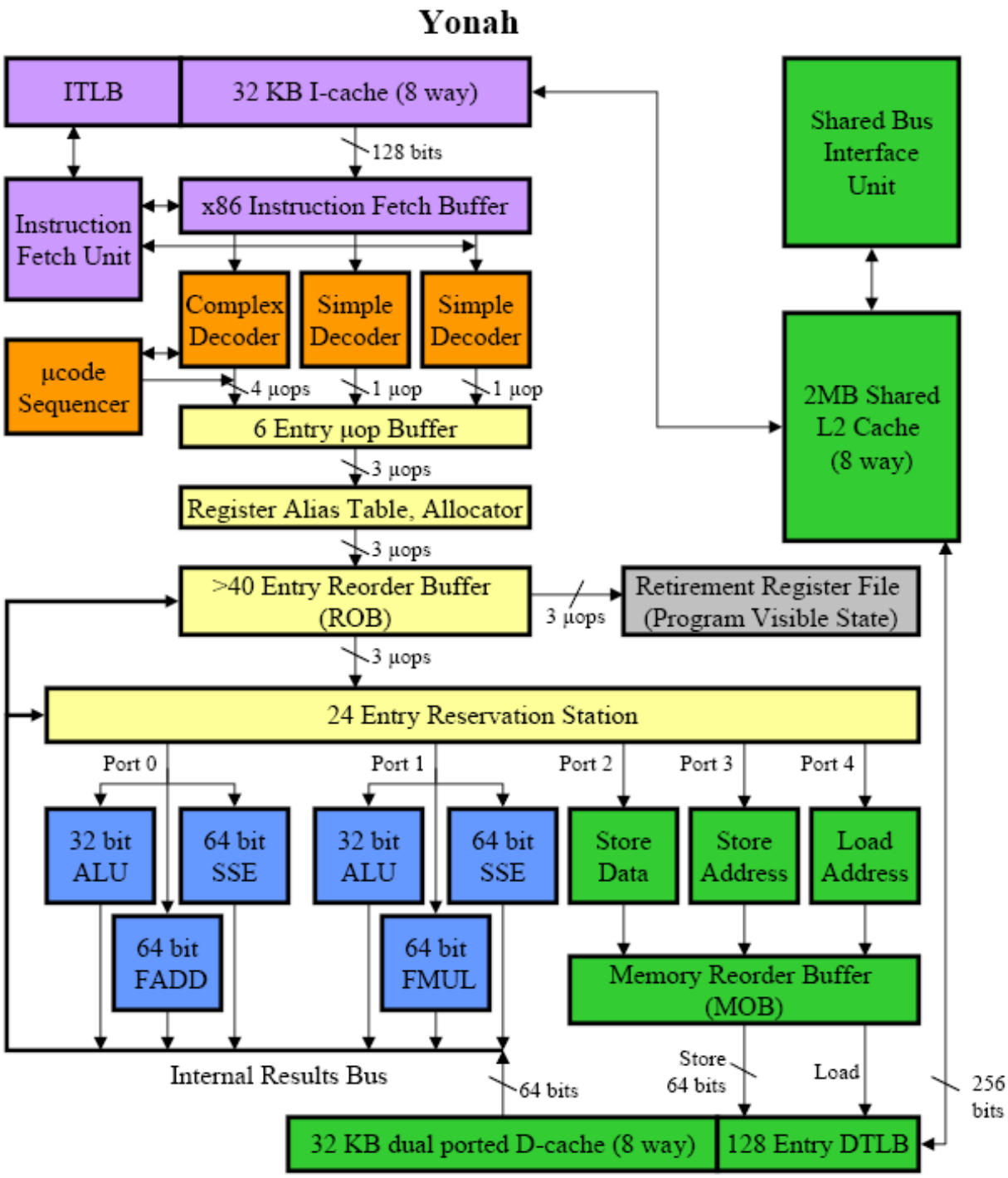


Figure 2 – Merom Microarchitecture

In Figure 2, the sub-blocks are color coded. Purple units are responsible for fetching instructions and predicting branches in the front end, while the orange blocks decode the x86 instructions into uops. The sections in tan are internal buffers for uops and the scheduling and out-of-order blocks. The functional units are all in blue, and the memory system is in green. Lastly, the actual x86-64 register state is in grey. For comparison, the Yonah and Pentium 4 microarchitectures are shown below in slightly less detail.


Yonah
ITLB
32 KB I-cache (8 way)
128 bits
Instruction Fetch Unit
x86 Instruction Fetch Buffer
Complex Decoder
Simple Decoder
Simple Decoder
µcode Sequencer
4 µops
1 µop
1 µop
6 Entry µop Buffer
3 µops
Register Alias Table, Allocator
3 µops
>40 Entry Reorder Buffer (ROB)
3 µops
Retirement Register File (Program Visible State)
3 µops
24 Entry Reservation Station
Port 0
Port 1
Port 2
Port 3
Port 4
32 bit ALU
64 bit SSE
32 bit ALU
64 bit SSE
Store Data
Store Address
Load Address
64 bit FADD
64 bit FMUL
Memory Reorder Buffer (MOB)
Internal Results Bus
64 bits
Store 64 bits
Load
256 bits
32 KB dual ported D-cache (8 way)
128 Entry DTLB
Shared Bus Interface Unit
2MB Shared L2 Cache (8 way)

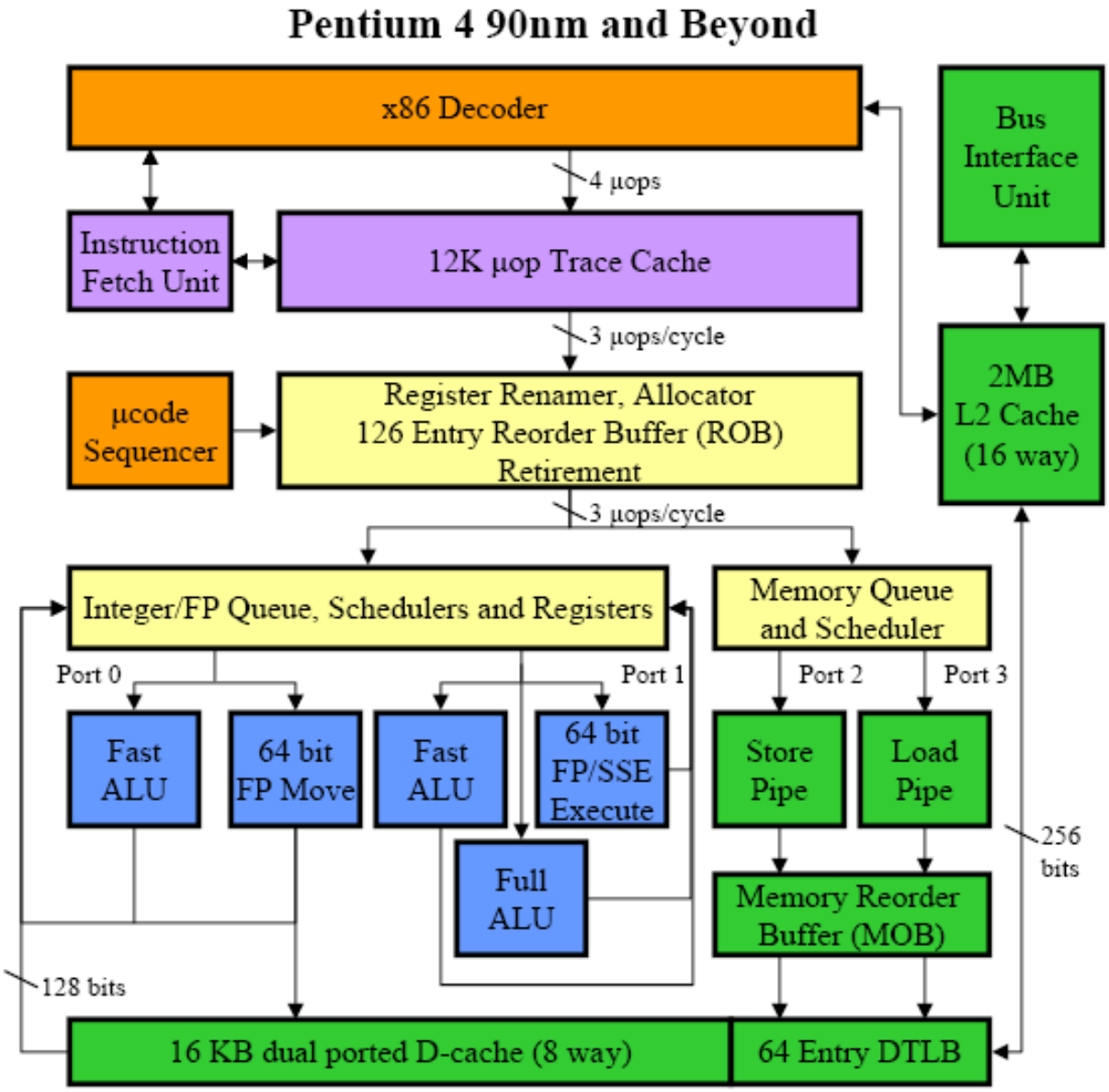


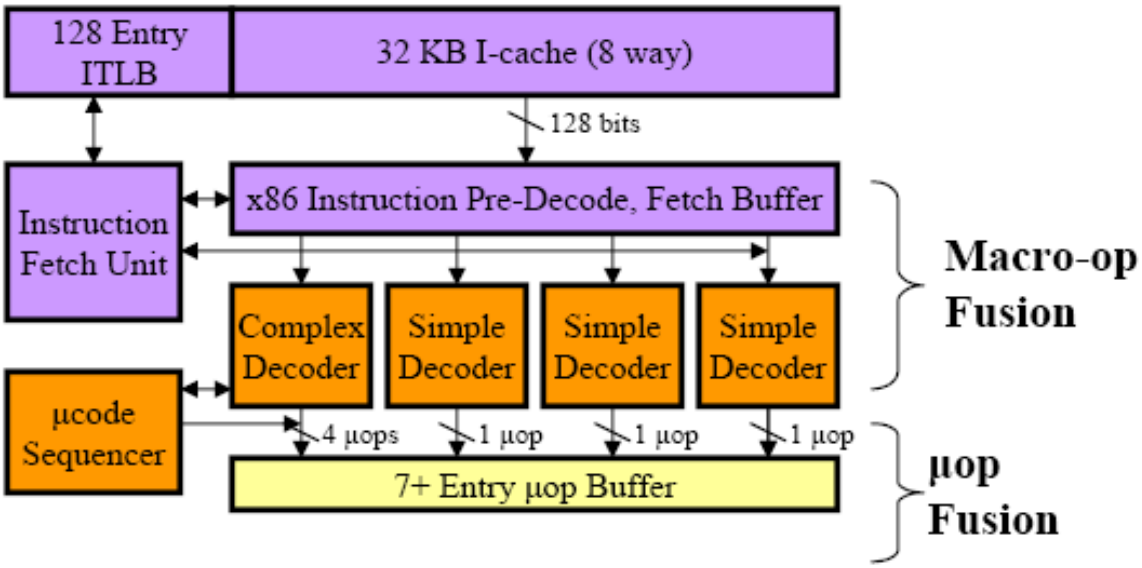
Figure 3 – Microarchitectures of Yonah and P4

Merom fetchs 128 bits of instructions, decodes 4+1 x86 instructions, issues 7 uops, reorders and renames 4 uops, dispatches 6 uops to execution units and retires up to 4 uops each cycle. In every regard, this is much wider and more aggressive than the P4 and Yonah. In the next few pages, we will dive into the details and explore each section of Merom.

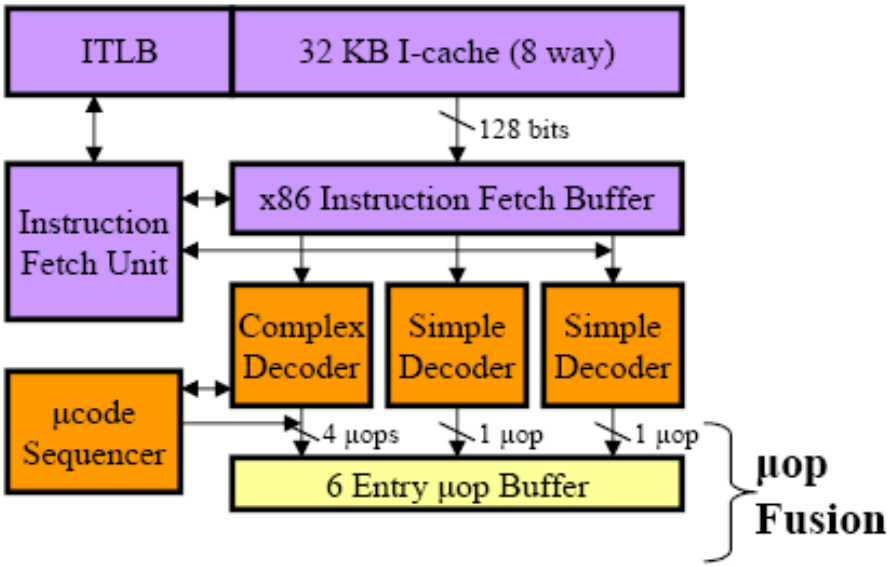
The Front-end

As promised, Merom is far wider than the P6 or P4. Figure 4 below shows a detailed comparison of the fetch and decode sections for Intel's microarchitectures.

Core Front End



Yonah Front End



P4 Front End

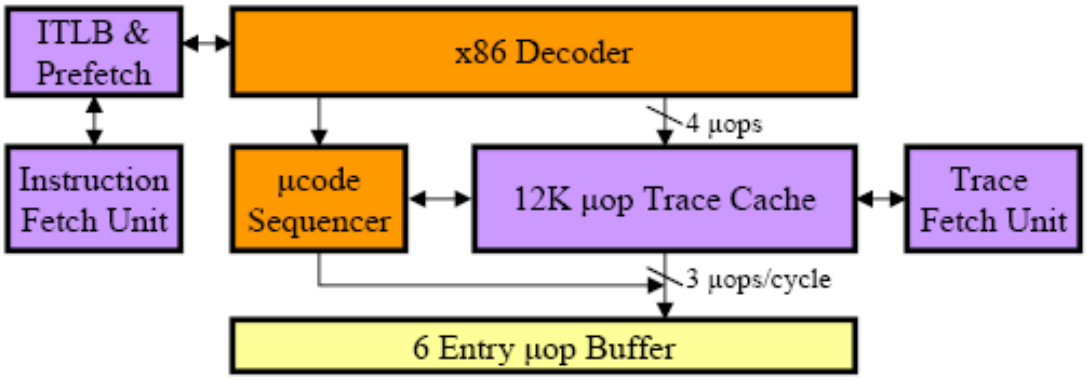


Figure 4 – Front-end Microarchitecture Compared

Intel did not initially disclose the fetch bandwidth, but the average x86 instruction is ~32 bits and it was later revealed that the instruction fetch remained at 128 bits. These instructions go into the pre-decode and fetch buffer, which also stores information about instruction length and decode boundaries. The pre-decode and fetch buffer is 32 bytes, as with the original P6, but feeds into an 18 entry instruction queue (IQ – not depicted above). When a loop is entirely contained within the IQ, the instruction fetcher will shut down to save power, using the IQ as a fetch cache.

The somewhat disappointing trace cache is gone, replaced by four x86 decoders, which pull instructions from the IQ. Each of the three simple decoders deal with x86 instructions that map to a single uop, while the complex decoder handles instructions that produce 1-4 uops (so the decode pattern is 4-1-1-1). The microcode sequencer is responsible for decoding or assisting instructions that produce more than 4 uops, as with prior designs. As with Yonah, all SSE instructions can be handled by the simple decoders, producing a single uop. effectively.

Additionally, as Figure 4 indicates, the Merom front-end introduces a new feature that is referred to as macro-op fusion. Within Intel, x86 instructions are called macro-ops, while the internal instructions are called uops. Macro-op fusion lets the decoders combine two macro instructions into a single uop. Specifically, x86 compare or test instructions are fused with x86 jumps to produce a single uop and any decoder can perform this optimization. Only one macro-op fusion can be performed each cycle, so the maximum decode bandwidth is really 4+1 x86 instructions per cycle. Macro-op fusion maps particularly well to the familiar if-then-else statement, which is a very common programming construct. Although Intel declined to comment, some estimates indicate that macro-op fusion can reduce the number of uops by 10%.

The benefits of macro-op fusion are readily apparent. Reducing the number of uops improves performance in two ways. The first is that fewer instructions are executed, which directly increases performance. Secondly, out-of-order execution becomes more effective since the out-of-order scheduling window can effectively examine more of the program at once and find more instruction level parallelism (ILP). Of course, these benefits are very similar to those from uop fusion, but improving a different class of instructions. Perhaps the most ironic part is that in some ways, macro-op and uop fusion are really making x86 MPUs interally more CISC-like, and less RISC-like.

Branch prediction occurs inside the Instruction Fetch Unit using many familiar predictors. Pentium M based designs featured the P4's traditional Branch Target Buffer (BTB), a Branch Address Calculator (BAC) and the Return Address Stack (RAS) but also two new predictors. The Loop Detector (LD) correctly predicts loop exits, and the Indirect Branch Predictor (IBP) picks targets based on global history, which helps for branches to a calculated address. Merom uses all of these predictors, and has added a new feature. In prior designs, taken branches always introduced a single cycle bubble into the pipeline. By adding a queue between the branch target predictors and the instruction fetch, most of these bubbles can be hidden.

The Out-of-Order Engine – Renaming and Scheduling

Merom's microarchitecture out-of-order engine generally resembles Yonah but with more resources. As Figure 5 below shows, they share many of the same structures, including the Register Alias Table, the Allocator, and Reorder Buffer. However, all of these structures have been enlarged to support more in-flight instructions and better exploit ILP.

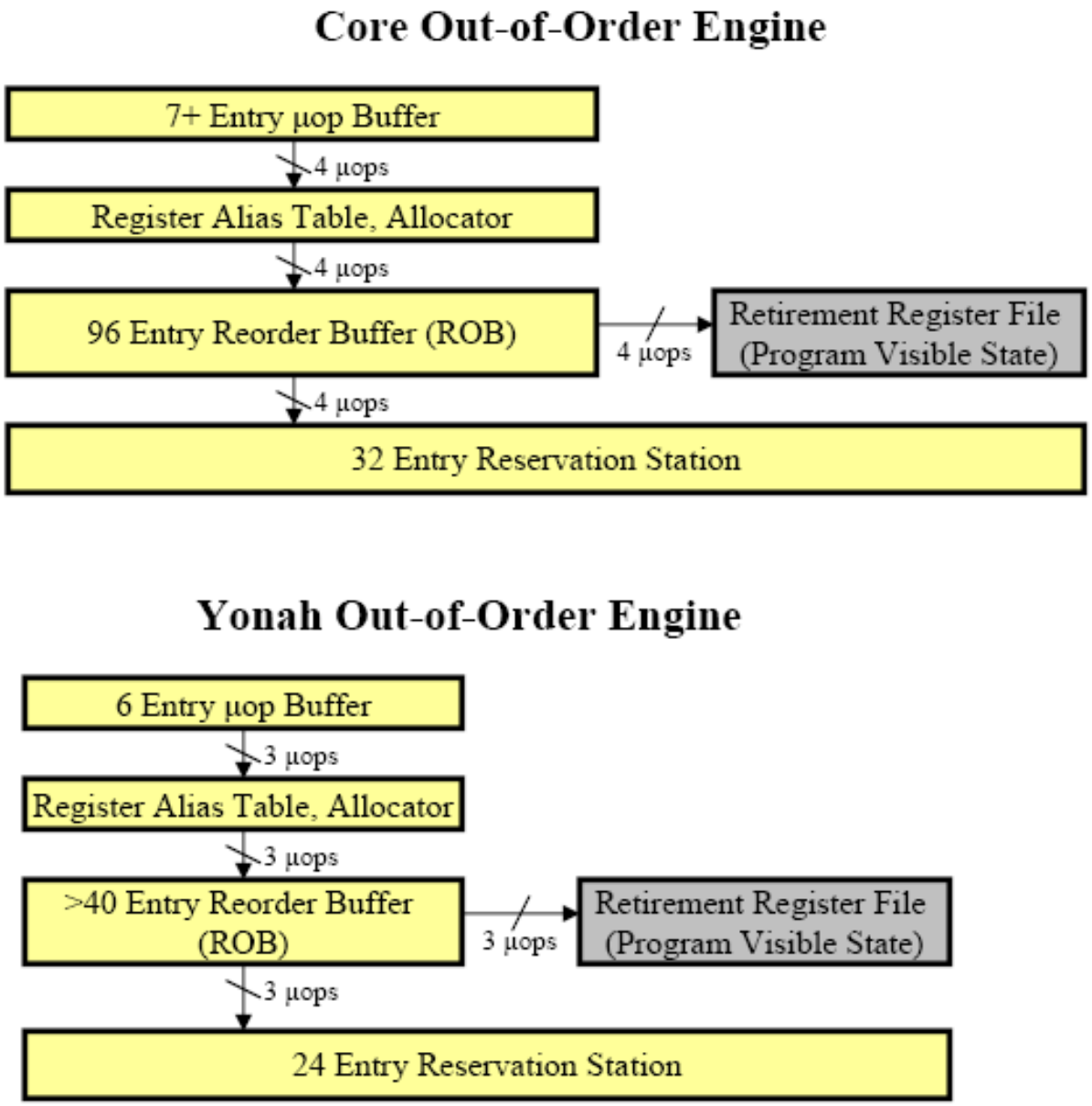


Figure 5 – Out-of-Order Engine Comparison

Both the P4 and Yonah have a maximum throughput of 3 uops/cycle, while Merom is designed to handle 4 uops/cycle, as Figure 6 shows. As a point of reference, the P4 ROB is 126 entries, Dothan (and probably Yonah as well) has >40 entries, compared to 96 entries for Merom. In fact, Merom has

more out-of-order resources relative to all other recent designs. The ratio of ROB entries to peak instructions in flight is around 1.7 for Merom, which compares favorably to the 90nm P4 (1.4 ROB/instructions), and the appropriate figure for Yonah, which I have been asked not to share. The reservation stations are also much larger for Merom than the Pentium M; 32 entries instead of 24. A comparison to the P4 is a little more difficult, since it uses distributed schedulers rather than reservation stations. The P4 has a total of 46 scheduler slots, 8 for the memory units, and 38 for the ALUs and FPUs.

The Out-of-Order Engine – Execution Units

While Intel was a little shy about the mechanics of renaming and scheduling, they were positively enthusiastic about the dispatch and execution in Merom. Merom has three execution dispatch ports, which feed a total of three 128 bit SSE units, two 128 bit floating point units, and three 64 bit integer units. The integer unit on dispatch 1 also handles 128 bit shifts and rotates and all of the ports can perform FP Moves. The FPUs and SSE units also share hardware where appropriate. The execution subsystems of Yonah and especially the P4 are paltry in comparison, as shown below in Figure 6.

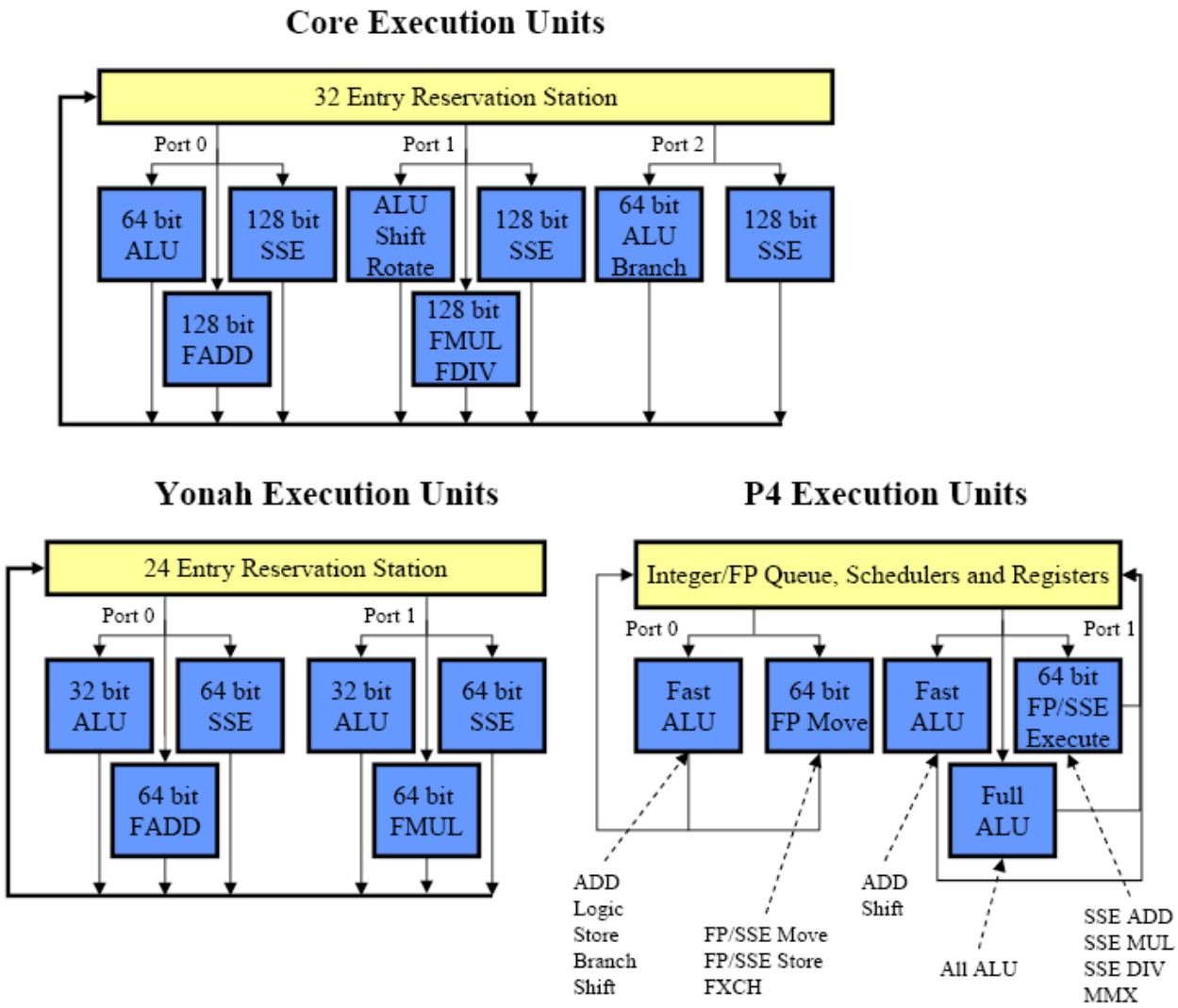


Figure 6 – Execution Unit Comparison

The first thing to notice is that Merom has an extra dispatch port compared to the P4 or Yonah, so it can consistently execute up to three instructions each cycle. The P4 can dispatch and execute 4 instructions each cycle, but that is relatively rare. To dispatch that many instructions, they all must be simple ALU operations, and there is a latency penalty for 64 bit operations. More importantly, Merom has a relatively balanced arrangement of functional units; in the P4, many operations go to dispatch 1, which causes contention. For integer operations, Merom will execute 3 operations per cycle much more consistently than the P4.

Merom also substantially improves on the floating point and SSE capabilities of its predecessors. Although Merom's 3 SSE units are not fully symmetric, the differences are relatively minor (shifting and multiplication resources). The SSE units are fully pipelined and each one can execute the appropriate 128 bit SSE operation in a single cycle. In comparison, the P4's SSE resources are somewhat scanty; the two 64 bit SSE units use two cycles to execute 128 bit operations, and therefore are only partially pipelined. Similarly, Yonah only has 64 bit data paths. In comparison,

Merom can perform 4 DP FLOPS/cycle and then some: a 128 bit multiply, a 128 bit add, a 128 bit load, a 128 bit store and then perhaps an ALU or fused compare and jump instruction in the last dispatch port.

The Memory System

Merom has extraordinary execution resources, and the memory system has been improved in tandem with the rest of the design. As Figure 7 shows, the memory system really looks like Yonah, but with the bandwidth of the P4. Note that the P4 uses the fast ALUs to calculate store addresses, which is why there is no store address unit.

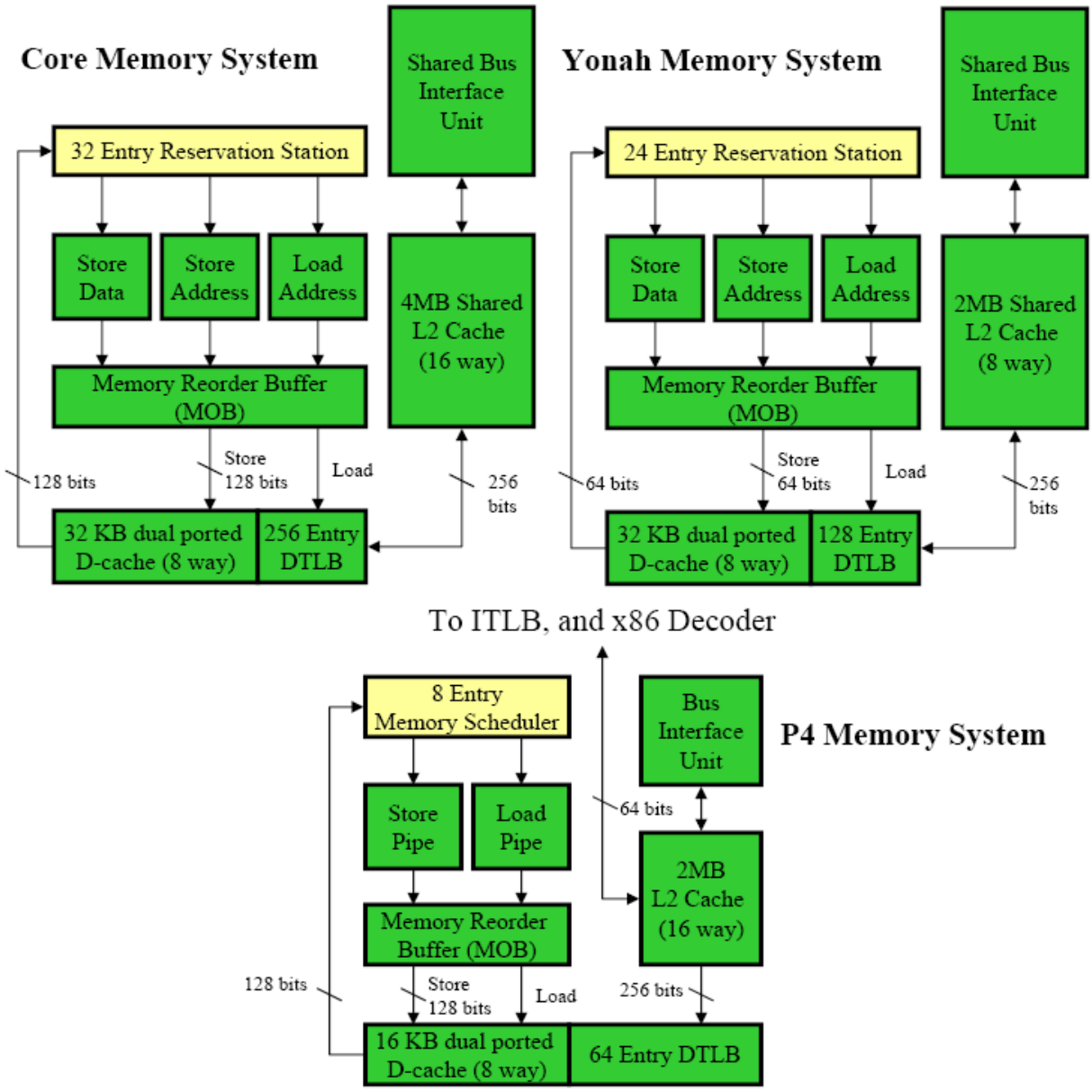


Figure 7 – Memory System Comparison

The caches in Merom and Yonah are both write back, and use 64 byte cache lines. The L1D cache for the P4 is write through with 64 byte lines and the L2 is write back with 128 byte lines, broken into two sectors.

The shared L2 cache for Merom is a non-inclusive, non-exclusive design. Latency numbers were not disclosed, but it is very likely that the L1D cache latency is 2-3 cycles, most likely 2. As previously mentioned, Merom can transfer directly between the L1D caches in some variants. However, it is currently unknown how often this transfer can occur, how much data is transferred (probably a cache line) and whether such a transaction would replace an L2 cache access. The Merom memory subsystem also implements new prefetchers designs to work effectively with shared caches. Each L1D cache has several prefetchers, and the L2 prefetchers dynamically allocate bandwidth between the two CPUs based on the data access patterns and intensity using a modified round-robin algorithm. The front-side bus is similarly arbitrated for fairness.

The memory system also implements a new technique for memory disambiguation in the MOB, which is described in the next two sections.

The Memory Aliasing Problem

Memory aliasing is a relatively easy to understand problem. Out-of-order CPUs usually have many load and store operations in-flight at the same time, but the CPU has to preserve the program order of loads and stores. Intuitively this problem is very similar to the cache coherency problem. Figure 8 shows the problem with aliasing; because they share the same address, instruction 9 cannot move before instruction 2 (broken red arrow), or it would read the wrong data. Unfortunately, the CPU does not know what address instruction 5 is storing to, so it is unclear whether instruction 9 can move around number 5 (solid black arrow). However, the CPU can obviously move instruction 9 before instruction 8 and after instruction 5, since there are no aliasing stores (solid blue arrow).

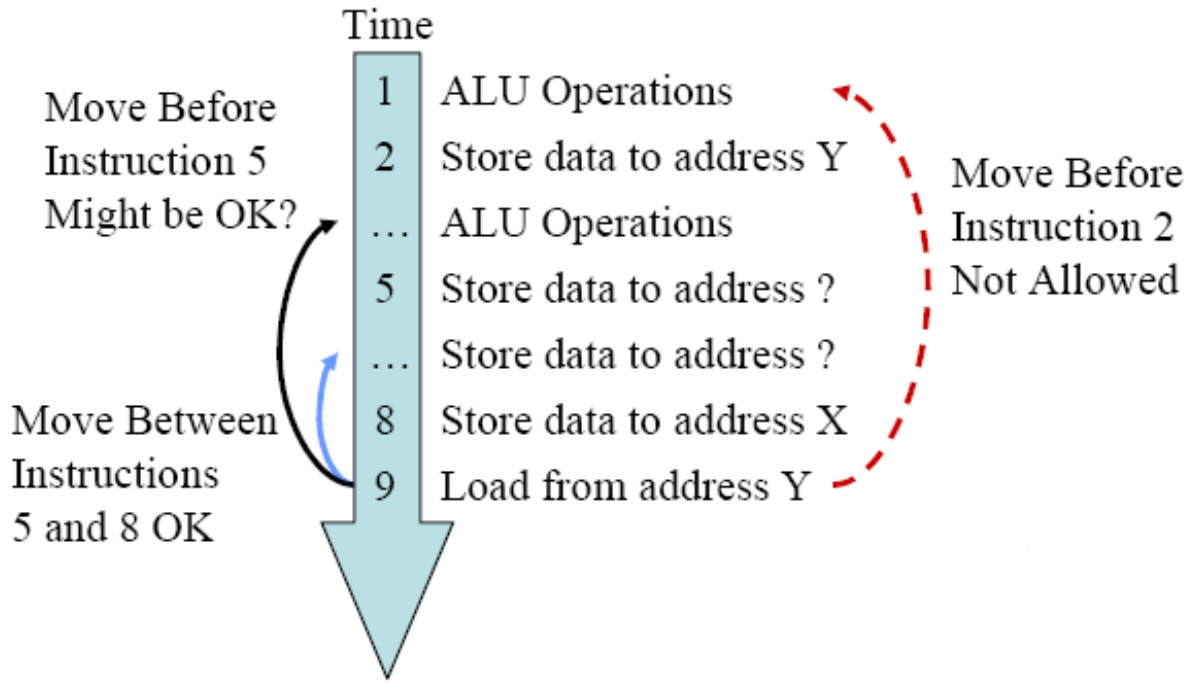


Figure 8 – Memory Aliasing Example

Memory disambiguation is the process of determining whether a pair of memory instructions (usually a load and a store) alias, or share the same address. If they use different addresses, then they can be moved around each other. The problem is that to disambiguate a load, the memory system has to search the addresses of all in-flight store operations, which is hideously expensive. The P6 makes this a lot easier by splitting up store instructions into two different uops, one to calculate the address, and one to actually store the data. This way the store address is known in advance, and can be easily checked for alias problems. In the P6, the Memory Reorder Buffer (MOB) uses the following rules to avoid aliasing:

1. All loads are delayed if a store is in-flight with an unknown address
2. Loads cannot proceed ahead of an aliased store data uop
3. A store cannot be moved in front of another store

The problem with this approach is that rule 1 is pessimistic and creates some false aliasing (i.e. it assumes aliasing, which is not always the case). Academic studies have shown that for an EV6-like processor with 512 in-flight instructions, roughly 97+% of loads and stores do not alias [1]. For a more modest instruction window (no shipping design has more than about 200 instructions in flight), there would be even less aliasing. Therefore, it makes sense to remove rule 1, and simply assume that all load/store pairs do not alias, but ensure correct recovery when a mistake is made.

Memory Disambiguation, the Solution

In Merom, loads can be speculatively moved around store instructions with an unknown address. In effect, the conservative assumptions in the P6 and P4 have been relaxed. Unfortunately, when a

load is moved illegally, there is a pipeline stall. To address this problem, Intel has implemented a dynamic alias predictor which predicts when a load cannot be moved around a store. This prediction is based on historical behavior and is said to achieve > 90% accuracy, although the architects would not comment on the matter.

Figure 9 below shows an example of how speculative memory disambiguation can increase performance compared to the conservative methods in other designs.

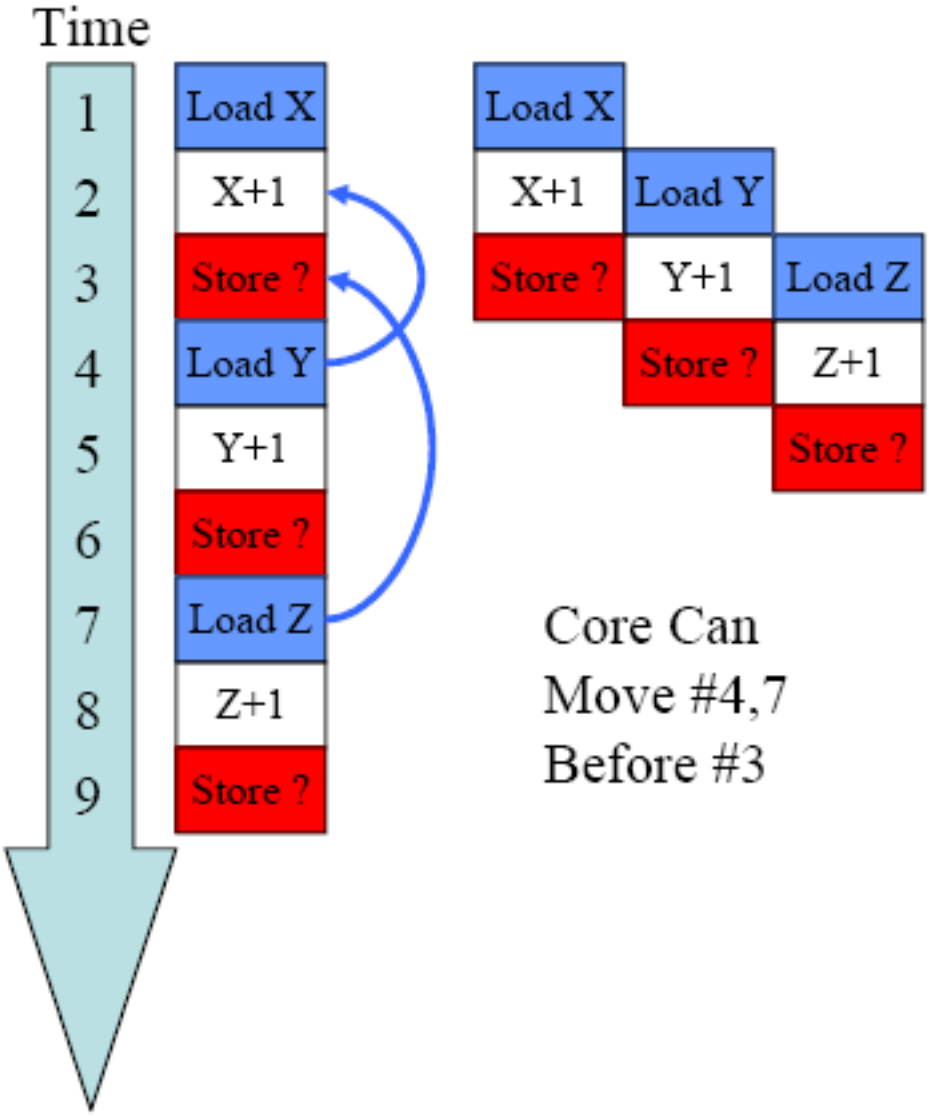


Figure 9 – The Benefits of Memory Disambiguation

This illustration is a little contrived, but it explains some of the importance of speculative disambiguation. Imagine a program that goes through a list of numbers and adds 1 to each entry; since this is the year 2006, the program was written poorly and the store addresses are unknown. Merom can speculatively move the blue load instructions before the red stores, and can make full use of multiple execution units. Other x86 CPUs would wait for each store address to be determined

before issuing the load instructions. In this hypothetical example, Merom would be about two times faster (5 cycles versus 9).

To really understand why disambiguation is important, it helps to think about a typical situation, rather than a contrived example. For x86, my own performance analysis for common workstation benchmarks has shown that roughly 10-25% of instructions are stores, and 30-45% are loads. Merom supports an out-of-order window of 96 instructions, which means that on average as many as 43 loads and 22 stores could be in flight. Now imagine there was no memory disambiguation. All it takes is for one of those stores to have an unknown address, and between a quarter and a half of the instructions in flight are forced to stall in the Reorder Buffer, using up valuable resources. Memory disambiguation solves this problem, by eliminating most false aliasing.

There is also room for improvement in memory disambiguation. Some of the original work on memory disambiguation was done at the Digital Equipment Corporation and was intended for use in the ambitious but ill-fated EV8. The techniques that were evaluated at DEC could be used to speculatively re-order stores, as well as loads [2].

Digital Thermal Sensor, SSE4 and other Improvements

While each of the subsystems have been substantially improved in Merom, there are also some new technologies that improve the entire MPU, rather than one particular section.

Merom includes an on-die digital thermal sensor, which has some rather interesting applications. The thermal sensor is used for safety and reliability features, as thermal diodes were previously. However, it is also rumored that it may be used to increase the frequency of the MPU when the sensor determines that there is thermal headroom. This is reminiscent of, but technically distinct from Foxton, which is much more tightly integrated into the MPU and measures current rather than heat. If this technology is productized, it will probably be for Conroe and Woodcrest only. The notion of using more power and producing more heat in exchange for better performance runs counter to the goals for mobile MPUs; therefore it is unlikely to end up being enabled for Merom.

Most of the MPU is heavily clock gated for more efficient operation. Each of the two cores is managed independently, and many entire blocks can be put to sleep, such as the microcode sequencer. Most internal buses are also gated for power savings. So if a bus is not sending out a full data load each cycle, part of the bus can be put to sleep. For example, if the FPU bypass network was mostly working with 64 bit operands, the part of the network that sends and receives bits 64-127 could be turned off to save power. In 99.9% of all cases, this gating has no impact on performance. This ability to shut off parts of the chip was integral to the entire design philosophy. Normally, increasing the IPC of a design means higher power consumption, whether the extra resources are used or not. With the extensive clock gating, the designers only pay for what is used, which makes a high IPC design much more attractive.

Merom also includes several new instructions, which were originally planned for Tejas. Tejas was a troubled super-pipelined design that would succeed Prescott, but was cancelled after the power consumption and heat dissipation problems became known. These new SSE4 instructions are not terribly interesting; there will be some performance gains, but nothing like the improvements from SSE2. One of the reasons why the performance gains will not be substantial is that most of the instructions are special purpose. Also, there is an architectural mismatch; Tejas was much more like the P4 than Merom. Naturally, it makes sense that instructions designed for Tejas might not benefit Merom as much. However, Intel is working on extensions for Penryn, the 45nm follow on to Merom, and it is rumored that these new instructions will be much more significant.

Performance

At this point in time, detailed performance numbers are difficult to find. Intel is claiming a 20% improvement for Merom over Yonah, a 40% improvement for Conroe on the desktop and an 80% improvement for Woodcrest over Paxville DP. Unfortunately, these are all relative and subject to change, as Merom is improved and tweaked over the coming months. However, the bottom line is that we expect the Merom family to provide a 20-40% performance boost over the prior generation products, and more in certain cases. At the same time, power consumption will drop dramatically for the desktop and server devices, in the range of 30-40% and possibly more. As a result, the performance/watt will improve substantially for Intel. With any luck, more solid details on performance, performance/watt and other quantitative metrics will be available in the near future.

Conclusions

Technically, this IDF has been one of the most rewarding in recent history. This is the first time in many years that a new x86 microarchitecture has been fully disclosed, and that is an exciting prospect regardless of the broader industry implications. Merom is an ambitious design that improves on its predecessors in nearly every way, especially the fanatical focus on power efficiency. Just as important as the technical innovations in Merom, this microarchitecture will have a profound impact on the industry. In the last two years, Intel has suffered a bit from the tail end of the infamous “Right Hand Turn”. Now that this turn is nearly complete, the MPU market will become far more competitive and hopefully, a new round of intense innovation will arise. Naturally, we will look forward to previews of Woodcrest and more details in the future.

References

[1] S. Sethumadhavan, et al. Scalable Hardware Memory Disambiguation for High ILP Processors. In 36th Annual International Symposium on Microarchitecture, 2003.
[2] G. Chrysos and J. Emer. Memory dependence prediction using store sets. In 25th Annual International Symposium on Computer Architecture, 1998.
[3] IA32 Intel Architecture Optimization Reference Manual.

Acknowledgements

We would like to thank several individuals for their time and energy:

- George Alfs
- Jack Doweck
- Bob Valentine
- Joel Emer
- Ofri Wechsler
- Anyone else who we may have forgotten or neglected to mention