Recent History of Intel Architecture -- a Refresher
Introduction

Pasted from <http://www.intel.com/cd/ids/developer/asmo-na/eng/44015.htm?prn=Y>

By Sara Sarmiento

To introduce a history of Intel Architecture, we could warm up with a discussion of one of the the first computers, the *Analytical Engine* by Charles Babbage, and take the history of computers from first inception to present day. Though an interesting read, it would be a long drawn out paper which would start with the development of computers that filled large rooms and contained thousands of transistors and conclude with the .13 micron processors that we have today. What is likely more relevant to developers working today, however, is modern architecture, so let's start with the Intel P5 Microarchitecture and discuss the evolution of the Intel family of 32 bit processors up to those available today. One theme we can borrow from the *Analytical Engine,* is that of using hardware to make computing data faster and more efficient.

Why Change the Architecture?

In any type of business, there is always room for improvement. In the restaurant business, for example, changes to better meet customer demand and improve the quality of service can build loyalty among customers. Adopting state of the art appliances and employing a more experienced staff are examples of changes that can bring this to pass. The same concept reflects in technology; software developers constantly strive for improvement. Developers begin revisions to create version X + 1 of a software product to add functionality, improve performance, and fix bugs, often before the completion of version X. Software architects consider customer feedback when defining these changes, as well as lessons learned from previous releases. Feature changes such as adding more graphics, changing the background colors, and improving the overall look and feel, can greatly improve the user experience. Rearranging data, unrolling loops, and/or using new instructions are some examples of active measures programmers can take in order to deliver performance improvements.

Technology advances in 32 bit Intel processor microarchitecture follow a similar process. The overall goal is to make the processor smaller, faster, and more robust to deliver a more powerful processor at a price the end users are willing to accept. However, unlike software, in which changes to the look and feel improve the user experience, hardware designers work to maintain the front end of the processor. In essence the goal is to provide a consistent look and feel to the software developer who will 'use' the processor to create products for his or her customers. Meanwhile Intel processor designers also make improvements to the back end to improve overall performance and maintain x86 compatibility, all the time working to make these transparent to the software developer. The fewer changes that the software developers must make to their code in order to take full advantage of the performance of the processor, the better. Before examining the details of architectural changes over generations of Intel processors, let us look at where performance comes from.

Where Does Performance Come From?

A processor sends instructions through a *pipeline,* which is a set of hardware components that acts on each instruction before it is written back to memory. An example of a hardware unit would be the *arithmetic logic unit* which executes simple integer instructions like add or subtract.

For a given set of instructions, the number of instructions processed over time is ultimately the metric that determines processor performance. Higher frequencies increase the speed at which data is processed. Improved branch prediction algorithms and data access (i.e. cache hits) reduce latencies and enhance number of instructions processed over time. Another way to improve performance is to increase instruction execution speed. Faster instruction computations and retirement raises the number of instructions that can be sent through the processor. All of these factors help increase the total throughput, and contribute to the overall performance of the processor.

Software developers can help increase the overall throughput of the processor by a number of coding practices. By increasing the number of instructions processed concurrently, the developer can reduce the amount of time that an application will spend in certain areas of code, where many cycles are spent processing data. A traditional way of doing this is to manually unroll a loop, and thereby increase the amount of data processed per loop iteration. Use of special instructions and larger data registers, initially made available in the P5 microarchitecture, are more alternatives to performing this task. This technology, know as SIMD (Single Instruction, Multiple Data) was introduced with MMX™ technology and allows computations to be performed using what are known as the 64-bit MMX technology registers. The MMX technology instruction set gives programmers the ability to execute instructions on multiple data elements loaded into MMX technology registers. Streaming SIMD Extensions (SSE), introduced in the P6 microarchitecture for the Intel Pentium III processor, extends MMX technology and allows SIMD computations to be performed on four packed single-precision floating-point data elements simultaneously using 128-bit registers (named XMM0-XMM7). Streaming SIMD Extensions 2 (SSE2) was introduced in the Intel ® NetBurst™ microarchitecture and extends SSE (and further extends MMX). SSE2 provides the ability to perform more computations in parallel by extending those instructions introduced in MMX technology and SSE to support 128-bit integer and packed double-precision floating-point data types. Figure 1 shows how an operation executes and data is stored in SIMD format within the XMM registers for 128-bit computations.

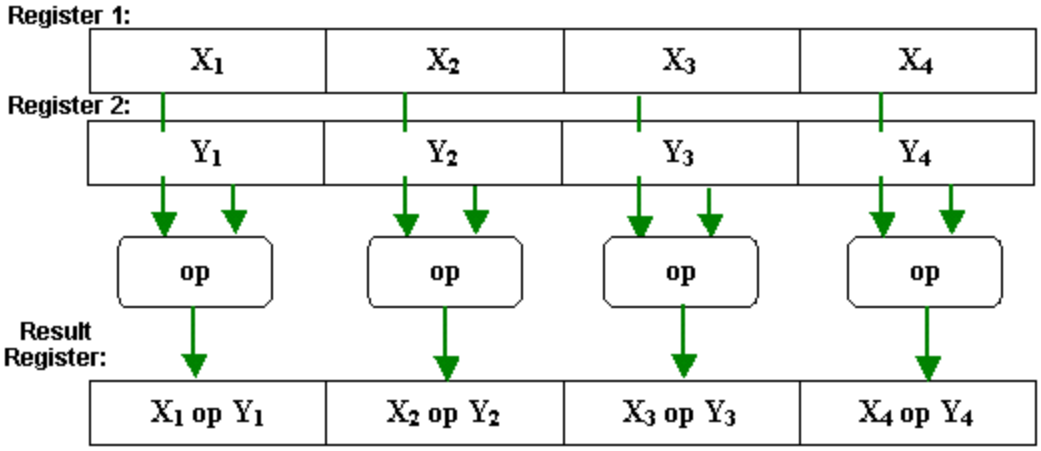

Figure 1. Xi and Yi (where i is between 0 and 4), are data sets, each contained within a register (Register1 and Register2 respectively). The "Result Register" contains the values of the instruction (denoted by "op" in the illustration) performed on the two data sets.

Evolution of Intel Architecture

The addition of SIMD instructions demonstrates just one way developers can influence the overall instruction throughput. During the evolution of the Intel 32 bit processor family, there has been continuous effort from a microarchitecture standpoint to increase the number of instructions processed at a given moment in time.

P5 Microarchitecture

The P5 Microarchitecture revolutionized the way that we did computing in previous x86 and x87 processors. First established in the Intel ® Pentium ® processor, it allowed faster computing, reaching three times the core frequency of the Intel486™ SX processor, its nearest predecessor. It achieved this with *instruction level parallelism* .

Introduction of Instruction Level Parallelism

" *Instruction level parallelism* " is a common theme of Intel processors. This is where independent instructions (i.e. instructions not dependent on the outcome of one another) execute concurrently to utilize more of the available hardware resources and increase instruction throughput. In the P5 microarchitecture, instructions move through the pipeline in order. However, there are cases where instructions pair up in such a way that they are allocated to two different pipes (pipe1 and pipe2), executed simultaneously, and then retired in order.

To help understand this concept, let's return to our restaurant analogy. This restaurant is equipped with a standard kitchen (i.e. staff, appliances, utensils, ingredients, etc.), a standard-sized table and a single smaller table, two waiters, and a long line of patrons waiting outside. The first waiter ushers parties in one by one; if two parties do not have conflicting food orders and can both be accommodated by

the size of tables, they can both be seated together, in which case the second waiter brings the other party to the smaller single table. For instance, party 1 orders hamburgers off the grill, and party 2 orders lasagna. Since their orders use different appliances and ingredients, they can be served together and more ingredients and appliances are used at once. Regardless of the size of the meal that each patron ordered, or the time it takes to prepare the meal, the cashier will only process bills in the order that the patrons entered the restaurant. The ability for the restaurant to serve their patrons two at a time (pending contending orders), improves the efficiency of the business as compared to serving one patron at a time.

The kitchen represents the available hardware resources, such as, the load and store units, arithmetic logic units, and the floating-point units. The table sizes represent the control unit; the two waiters represent the two pipelines that are available for the instructions to follow for execution, and the smaller table can only seat certain sized parties. The tables represent the limitations for the types of instructions it can execute. The long line of patrons represents the code cache, where all the instructions are waiting to be pre-fetched and decoded. The order in which the patrons pay their check is the retirement of the instructions in order. This example depicts how instructions are handled within the pipelines of the P5 microarchitecture.

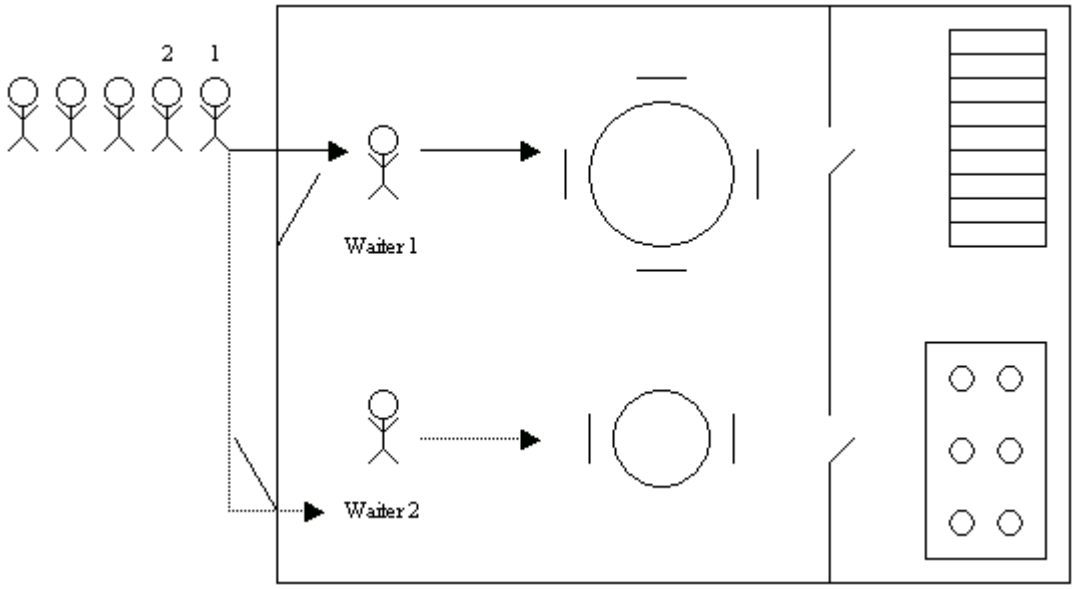

Figure 2. This is a diagram of the restaurant with only two waiters, and two variably sized tables. The parties of patrons outside are waiting to enter, and will leave the restaurant in the order that they came in. If party 1 and party 2 do not have conflicting food orders, then both waiter 1 and waiter 2 can seat each. The smaller table is limited to the number of people per party.

Like the restaurant set up to serve multiple patrons at the same time, the Pentium® processor executed some instructions concurrently. It was a superscalar processor that implemented the P5 microarchitecture with a 5-stage pipeline. It incorporated two general-purpose integer pipelines, and a pipelined floating-point unit. This essentially allowed the Pentium processor to execute two integer instructions in parallel. The main pipe (U) has five stages: pre-fetch (PF), Decode stage 1(D1), Decode stage 2 (D2), Execute (E), and Write back (WB). The secondary pipe (V)

shared characteristics of the main one but had some limitations on the instructions it could execute.

The Pentium processor issued up to two instructions every cycle. During execution, it checked the next two instructions and, if possible, issued pathing so that the first one executed in the U-pipe, and the second in the V-pipe. In cases where two instructions could not be issued, then the next instruction was issued to the U-pipe and no instruction is issued to the V-pipe. These instructions then retired in order (i.e. U-pipe instruction and then V-pipe instruction). Figure 3 below shows the P5 microarchitecture, and how instructions flow through the pipeline.

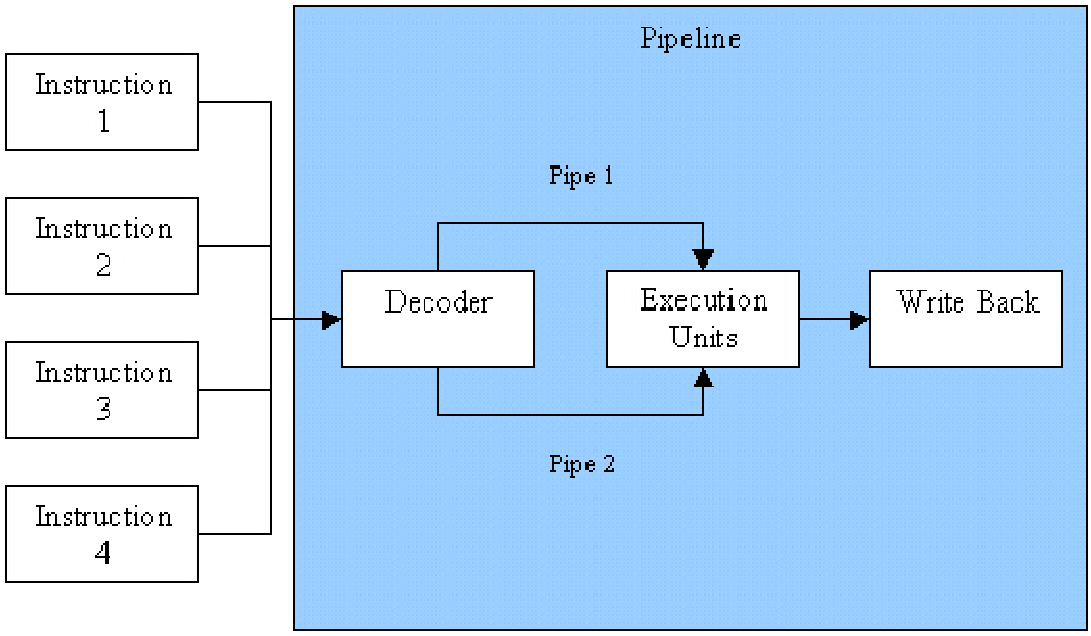

Figure 3. This diagram simply shows instruction throughput within the P5 Microarchitecture. Instructions are sent through the pipeline in order, and dependencies among instructions and the types of instructions used will determine if they can be executed concurrently.

The ability for the processor to work on more than one instruction at a time helped improve the cycles per instruction and increased the frequency from previous architectures. However, with these microarchitectural improvements came associated software development costs, as well as physical limitations. The parings for the two pipelines were particular, with rules specified in the optimization guidelines that programmers interested in performance had to learn. The V-pipe was limited in the types of instructions that it could process, and floating-point computations in this pipe required more processing as compared to the U-pipe. Architecturally, the five pipeline stages limited the frequency the processor could reach, and the Pentium processor's maximum speed peaked at 233 MHz. Consequently, processor architects started looking for ways to increase the number of instructions executed per clock and also increase the overall frequency of the processor.

P6 Microarchitecture

The P6 microarchitecture (Pentium® Pro, Pentium® II and Pentium® III processors) grew out of a desire to increase the number of instructions executed per clock, and improve the hardware utilization compared to P5 microarchitecture. The idea of *out-of-order execution* , or executing independent program instructions out of program order to achieve a higher level of hardware utilization, was first implemented in the P6 microarchitecture. Instructions are executed through an out-of-order 10 stage pipeline in program order. The *scheduler* takes care of resolving data dependencies, and sends the instructions to their appropriate execution unit. The *re - order buffer* takes care of putting the instructions back in order before writing back to memory.

Utilizing more hardware resources

By executing instructions out of order, the P6 Microarchitecture increased the hardware utilization over the P5 Microarchitecture. The Pentium Pro, Pentium II and Pentium III processors, all based off the P6 Microarchitecture, also significantly increased performance by increasing the number of instructions handled concurrently in flight, and increased the number of pipeline stages. The P6 Microarchitecture has three sections, the front end, which handles decoding the instructions, the out of order execution engine, which handles the scheduling of instructions based on data dependencies and available resources, and the in order retirement unit, which retires the instructions back to memory in order. Let's look at the P6 microarchitecture in a bit more detail, and take a look at a a simple diagram of the pipeline, shown in Figure 4.

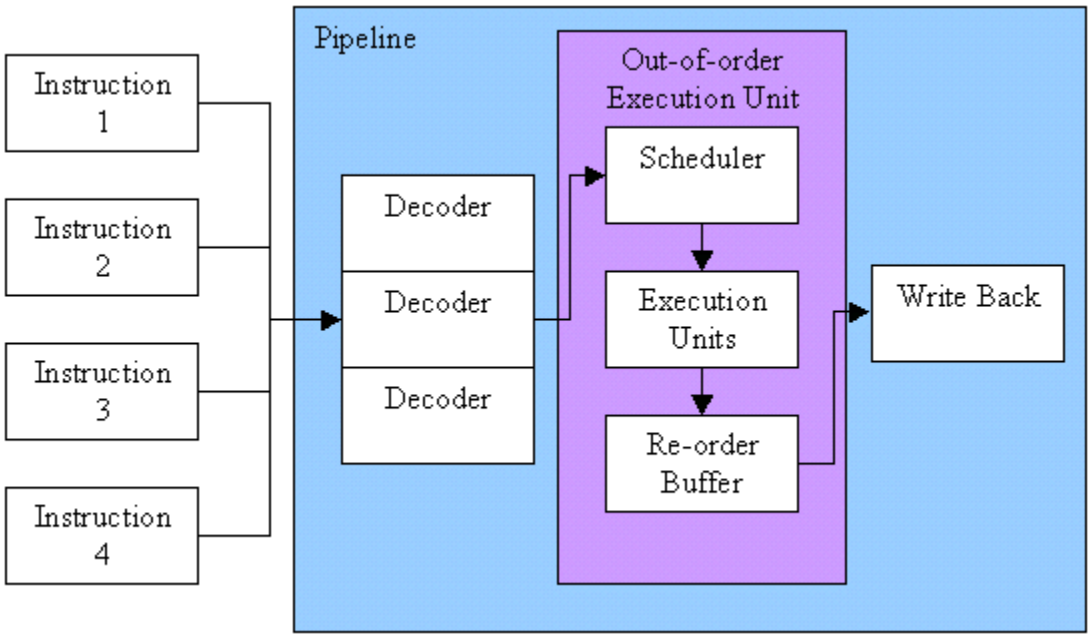

Figure 4. This figure simply shows the instruction flow through a P6 Microarchitecture. There are 2 more decoders as well as the out-order-execution engine. Up to three instructions can be retired concurrently to memory. The ROB is responsible for ensuring that instructions are written back in order.

The front end supplies instructions in program order to the out-of-order core. It fetches and decodes Intel Architecture-based processor macroinstructions, and breaks them down into simple operations called micro-ops (µops). It can issue multiple µops per cycle, in original program order, to the out-of-order core. The pipeline has three decoders in the decode stage which allows the front end to decode in a 4-1-1 fashion, meaning one complex instruction (4+ µops), and two simple instructions (1 µop each). Going back to the restaurant analogy, imagine that the same restaurant made improvements to the way it conducts business. The restaurant has now hired more waiters, and increased the number of tables within the restaurant. Instead of the waiters greeting and serving the customers, a maitre' d determines if patrons' orders conflict with each other, then seats them accordingly.

The out of order execution engine executes instructions out of order to exploit parallelism. This feature enables the processor to reorder instructions so that if one µop is delayed while waiting for data or a contended resource, other µops that are later in program order may proceed around it. This allows for a more efficient use of the processor's hardware resources. The core is designed to facilitate parallel execution assuming there are no data dependencies. Load and store instructions may be issued simultaneously. Most simple operations, such as integer operations, floating-point add, and floating-point multiply, can be pipelined with a throughput of one or two operations per clock cycle. Long latency operations can proceed in parallel with short latency operations. In the restaurant example, there are now three waiters instead of just two, to send orders to the kitchen. Since the matire-d is taking care of determining when the parties will be served next, all waiters need to do is bring the order to the kitchen. The matire-d can seat parties with large orders and parties with small orders, just as long as they are not requesting items that will require an appliance or ingredient necessary to make the other orders.

When a µop completes and writes its result, it is retired. Up to three µops may be retired per cycle. The unit in the processor, which buffers completed µops, is the *reorder buffer (ROB)* . The ROB updates the architectural state in order, that is, updates the state of instructions and registers in the program semantics order. The ROB also manages the ordering of exceptions. The ROB is analogous to the cashier of the restaurant, in that it ensures in order retirement of instructions. Upgrades to the kitchen appliances help to get orders through a little faster. However, even if the patrons towards the back of the line can enter before those in front of them, as for instructions in the P5 the microarchitecture, they still need to wait to pay until those originally in front of them in line 'pay their bill'. It is the cashier's job to make sure that this happens efficiently. This way the restaurant can serve more customers at once.

The ability to execute more instructions concurrently, improved branch prediction algorithms, and longer pipelines have their advantages, but they introduce coding challenges to the developer. Instead of focusing on instruction pairs, developers had to become aware of how the data transferred between registers. Because of the new branch prediction algorithm, conditional statements and loops had to be arranged in such a way that it would increase the number of correct branch predictions. Furthermore, developers had to consider accesses to data because cache misses created longer pipelines latencies, costing time. In the end then, even with the ability to execute the instructions out of order, the processor was limited in instruction throughput and frequency. Hardware designers continued looking for a way to increase the number of instruction processed and improve hardware utilization.

Intel NetBurst™ Microarchitecture

The concept behind the Intel® NetBurst™ microarchitecture (Pentium® 4 processor, Intel Xeon™ processor), was to improve the throughput, improve the efficiency of the *out-of-order execution* engine, and to create a processor that can reach much higher frequencies with higher performance relative to the P5 and P6 microarchitectures, while maintaining backward compatibility.

Smaller, Faster, More Robust

The Intel NetBurst microarchitecture took a new twist on the out-of-order execution engine. Initially launched in the Pentium ® 4 processors in late 2000, this new architecture offered the ability to process more instructions per clock and reach even higher frequencies than its predecessors (over 2X and rising) by significantly increasing the number of pipeline stages to 20. The die size of the Pentium 4 processor was reduced, currently at .13 micron, allowing for faster data transfer through the pipeline. It has faster arithmetic-logic units and floating-point units, improved branch prediction algorithms and new features that will help increase the efficiency of the hardware utilization and increase the overall throughput of the processor. Figure 5 shows a diagram of this new microarchitecture. The next few paragraphs discuss the NetBurst microarchitecture in more detail.

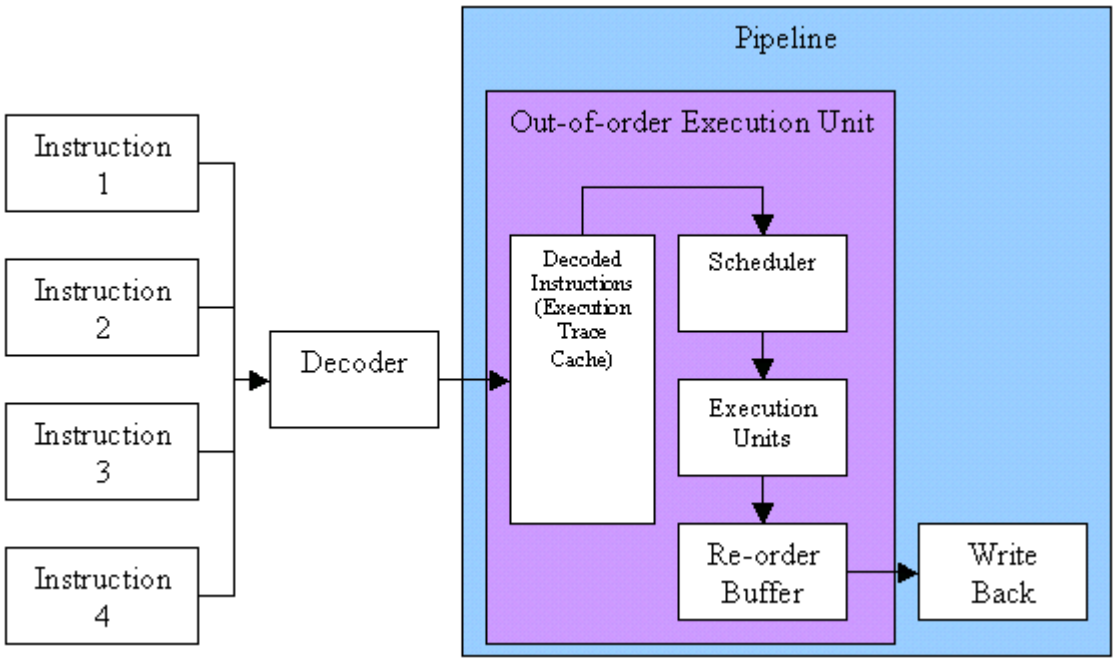

Figure 5. This figure is a simple diagram of the Intel NetBurst microarchitecture. There is only one decoder (as opposed to the three in the P6 microarchitecture), and the out of order execution unit now has the execution trace cache that stores decoded µops.

The Intel NetBurst microarchitecture addressed some of the common problems found in high-speed, pipelined microprocessors. Limiting factors for processor performance were delays from pre-fetch and decoding of the instructions to µops, the efficiency of

the branch prediction algorithm, and cache misses. The *execution trace cache* addresses these problems by storing decoded IA-32 instructions. Instructions are fetched and decoded by a translation engine, which builds the decoded instruction into sequences of µops called *traces* , which are then stored in the trace cache. The execution trace cache stores these µops in the path of predicted program execution flow, where the results of branches in the code are integrated into the same cache line. This increases the instruction flow from the cache and makes better use of the overall cache storage space since the cache no longer stores instructions that are branched over and never executed. The trace cache delivers up to three µops per clock to the core. In our restaurant analogy, this would correspond to the restaurant becoming more efficient by upgrading the appliances and further training their wait staff. However, our restaurant owner discovered that the biggest bang for the buck came from introducing a preferred customer list, analogous to the trace cache in Intel NetBurst microarchitecture The restaurant handles regular patrons everyday who order the same thing every time. So instead of the maitre' d needing to find out what their orders are each time they come in, there is a list of their names and their orders, and they are simply brought to the table.

The execution trace cache and the translation engine do something similar with their cooperating branch prediction hardware. Branch targets are predicted based on their linear address using branch prediction logic and fetched as soon as possible. Branch targets are fetched from the execution trace cache if they are cached there; otherwise, they are fetched from the memory hierarchy. The translation engine's branch prediction information is used to form traces along the most likely paths.

The core's ability to execute instructions out of order remains a key factor in enabling parallelism. The processor employs several buffers to smooth the flow of µops. This implies that when one portion of the entire processor pipeline experiences a delay, that delay may be covered by other operations executing in parallel (for example, in the core) or by the execution of µops which were previously queued up in a buffer (for example, in the front end).

The NetBurst microarchitecture adds further improvements to the execution units over that of the P6 microarchitecture. For example, the arithmetic logic units operate twice as fast as previous microarchitectures. This would be like the kitchen in our analogy getting an upgrade to a state-of-the-art kitchen in which no opportunity for efficiency was missed. They can fricassee or roast a duck in half the time! This helps the restaurant get more customers through the restaurant to increase revenue, just as the faster execution units help to get more instructions through the pipeline of a Pentium 4 or Intel Xeon processor.

As with the previous implementations, the retirement section receives the results of the executed µops from the execution core and processes the results so that the proper architectural state is updated according to the original program order. For semantically correct execution, the results of IA-32 instructions must be committed in original program order before they are retired. Exceptions may be raised as instructions are retired. Thus, exceptions cannot occur speculatively, they occur in the correct order, and the machine can be correctly restarted after an exception. When a µop completes and writes its result to the destination, it is retired. Up to three µops may be retired per cycle. Again, the ROB is the unit in the processor which buffers completed µops, updates the architectural state in order, and manages the ordering of exceptions. The retirement section also keeps track of branches and sends updated branch target information to the branch target buffer (BTB) to update

branch history. In the restaurant analogy, the cashiers in the restaurant represent the reorder buffer. The customers still need to pay their bill in the order that they came in, but now there are three cashiers instead of one to make sure that they leave at the best time. The retirement section of the pipeline, which handles updating branch history, would be analogous to the cashier sending the waiter the check back because a patron ordered another meal.

Longer pipelines and the improved out-of-order execution engine allow the processor to achieve higher frequencies, and improve throughput. Again, these microarchitecture improvements incur some costs and tradeoffs. Despite the improvements on the branch prediction algorithms, mispredicted branches are more costly and incur a significantly larger penalty as opposed to previous architectures due to the longer pipeline. The same is true for cache misses during data accesses. Developers who know the restrictions for manipulating data within the Intel NetBurst architecture will likely adjust their coding with this in mind and avoid these types of performance penalties.

Conclusions and Implications

Microarchitectures vary generation to generation as seen in the previous sections. Methods to optimize for one may not be suitable, and can possibly degrade an application's performance, for another. For example, instruction pairing for the P5 Microarchitecture is not beneficial to the P6 or NetBurst microarchitectures. Branch prediction algorithms for each processor differ among microarchitectures, and optimization recommendations should be taken into consideration when creating loops and conditional statements. These are only a few examples of optimization guidelines to keep in mind when developing an application. Reference manuals that discuss the basic microarchitecture, instruction set, and the system programming are available with each generation of Intel processors. An optimization manual is also available per microarchitecture. It gives details of what developers can do to their code to improve performance for each generation of processor. Developers should be aware of how instructions are executed, and how to write their code to get the full performance benefits for the latest processor family.
Updates to the Intel desktop processor family are continually being made. Modifications will center around the topics that we have discussed so far, and perhaps in other areas as well. Engineers will find new approaches to increasing the frequency, and the overall throughput for the next generation processors. Detailed information regarding the topics discussed and other architectural specific features can be found in the resources listed in the section below.

References

Intel Architecture Optimization Manual,
http://developer.intel.com/design/pentium/manuals/242816.htm

Intel Architecture Optimization Reference Manual (Pentium II and Pentium III),
http://developer.intel.com/design/PentiumII/manuals/245127.htm

IA-32 Intel® Architecture Software Developer's Manuals,
http://developer.intel.com/design/pentium4/manuals/index_new.htm

IA-32 Intel® Architecture Software Developer's Manual, Volume 1: Basic Architecture, http://developer.intel.com/design/pentium4/manuals/index_new.htm

IA-32 Intel® Architecture Software Developer's Manual, Volume 2: Instruction Set Reference, http://developer.intel.com/design/pentium4/manuals/index_new.htm

IA-32 Intel® Architecture Software Developer's Manual, Volume 3: System Programming Guide, http://developer.intel.com/design/pentium4/manuals/index_new.htm

About the Author

Sara Sarmiento is a Technical Marketing Engineer in the Software Solutions Group at Intel Corporation. A graduate from Oregon State University, Sara holds a Bachelors of Science degree in Computer Science. She joined Intel in October 2000, and is currently working on software enabling for current and future generation client desktop processors.