An Introduction to the Sandy Bridge Microarchitecture

J. Riedel, J. Wei

I. Introduction

For a period of two weeks, our team researched Intel's Sandy Bridge generation Central Possessing Unit. To provide a context for which we, and others, could understand, we have compared it to MIPS, along with several of Intel's earlier CPUs. The goal of this project is to determine the major differences between what we have learned in class and what is used in modern desktop processors in the world today. Through the research that we have conducted, we believe that we can explain some of the intricacies that differentiate MIPS and Sandy Bridge. The complexity of modern processors is almost incomprehensible, and we have therefore elected to focus on a couple of the optimizations in Sandy Bridge.

II. History and Future

According to Moore's Law, the number of transistors on the same space will be doubled every two years. The earliest commercial processor is Intel's 4004 4-bit unit in 1971. Intel's 8008 was released the next year with an 8-bit instruction bus. By 1978, Intel's 8086 contained a 16-bit bus and was 10x faster than any previous processor. The Intel 386, first produced in 1985, is known for its ability to run multiple programs at once. Then in 1993, Intel released the first Pentium processor.

With the release of the Pentium 4 in 2001, Intel began its Tick-Tock design policy. Each tick phase represents a new die shrink and an architecture refinement. The tock is on the same die size as the tick but has a new architecture. The Core architecture tock was released in 2006 with die of 65nm and the tick shrink reduced it to 45nm. The Nehalem tock followed in 2009 and the shrink tick brought it down to 32nm. Sandy Bridge started in 2011 and shrunk to 22nm. Haswell followed in 2013 and shrunk to 14nm. The current architecture is Skylake, released in August of this year. In 2013, the 10nm shrink was originally planned for 2015 as well but has since been delayed for 2 years until 2017.

III. Out-of-Order Execution

Out-of-Order execution (OoO), also known as dynamic execution, is a design used in many modern, high-performing microprocessors to reorder instructions and hide memory latencies by making use of idle time like cache misses. It works by decoding and retiring instructions in the original order, while execution is done out-of-order for independent instructions. The specification of only executing independent instructions out-of-order is to ensure that executed commands do not change the intent of the original code.

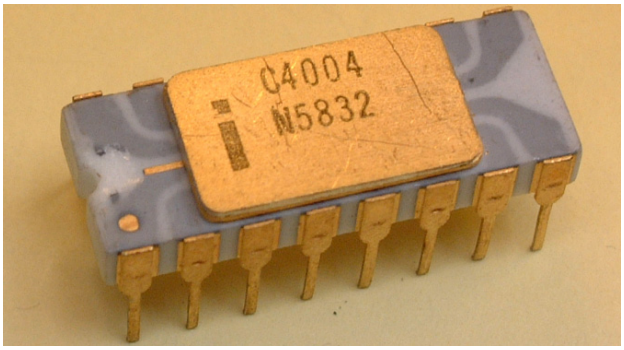

Fig. 1. The Intel 4004 processor was released in 1971 and was one of the earliest commercial processor.

```
lw $t0, 20($s2)
addu $t1, $t0, $t2
sub $s4, $s4, $t3
slt $t5, $s2, 20
```

TABLE I
A set of example instructions.

For instance, given the code in Table I, the runtime can be optimized using OoO since the third and fourth instruction (sub and slt) are independent while the second instruction (addu) is dependent on the first instruction (lw) and has to wait for `$t0` to load.

As shown in Figure 2, OoO operates like a dynamic pipeline with three main units: an instruction fetch and decode unit, functional units, and a commit unit, where the first and third are in-order while the functional units (the execution) are out-of-order.

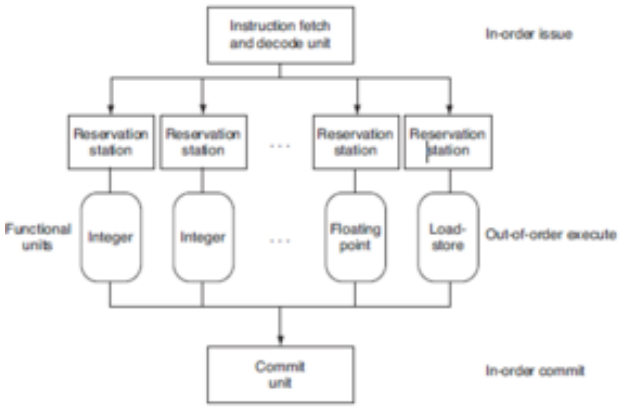

Fig. 2. Out-of-Order Execution Flow Diagram consists of three main units: an instruction fetch and decode unit, functional units, and a commit unit.

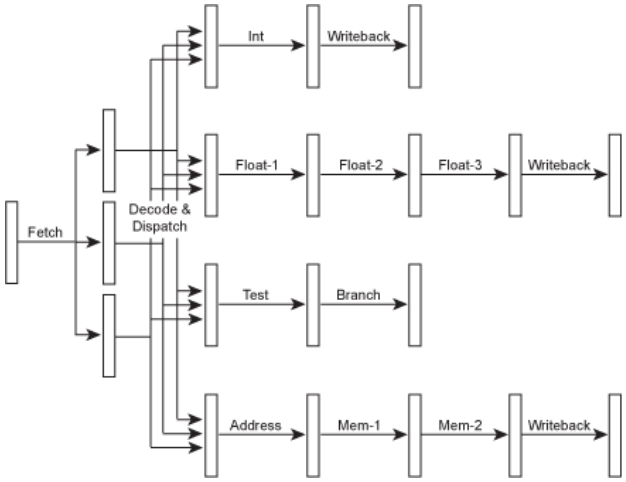

Fig. 3. A simple implementation of Superscalar Architecture showing several instructions being dispatched and executed.

The functional units are able to operate out-of-order with the help of buffers (also referred to as reservation stations) that hold operands and operations. Once both are obtained and ready to execute, results are calculated and sent to the commit unit as well as any waiting buffers. In the commit unit lies a reorder buffer that buffers the results until it's safe to put the results into a register file or to store it in memory. With the reorder buffer and the reservation stations, the OoO pipeline CPU is able to essentially implement register renaming.

IV. Superscalar Architecture

Superscalar Architecture refers to the inclusion of multiple execution units inside of a single core. These units are connected to a scheduler or dispatcher. This unit controls what instruction is to be sent to each of the execution units. Often, the execution units will be specialized to increase the speed of different tasks. This superscalar architecture increases the speed of the processor in several ways. By dispatching specific instructions to an optimized execution unit, the instruction will complete faster and spend less time in the execution cycle. If an instruction does take a long time to complete in the execution cycle, like a `LOAD`, the processor can continue to execute other instructions before and after the `LOAD` on the other available units. The dispatch will simple select a different execution unit as the target for the next instruction. Both of these optimizations increase the instructions per cycle of the processor by either reducing the cycles need to complete an instruction, or increasing the number of instructions being executed each cycle.

In 1993, Intel released its P5 microarchitecture with the Pentium processor. It was Intel's fifth-generation and first superscalar IA-32 microarchitecture. Since then, every desktop processor released by Intel have included superscalar architecture. The Pentium M and the Core execution architecture are fairly similar. Pentium M contained five dispatch ports, three of which were connected to memory units. the other two were connected to execution units. Each memory unit had one connection to the scheduler. These units were the

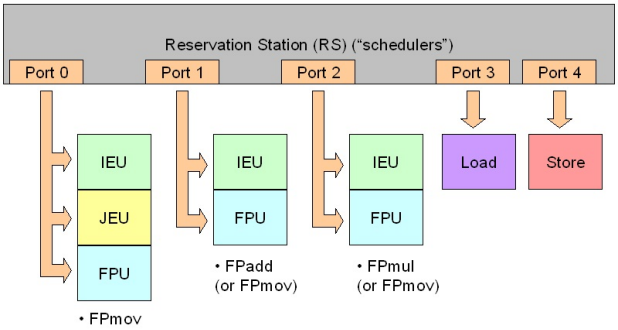

Fig. 4. Core's Superscalar Architecture with two memory ports and three execution ports. Although Pentium M and Core have the same number of dispatch ports, Core is able to process more integer instructions because of the swapped execution unit.

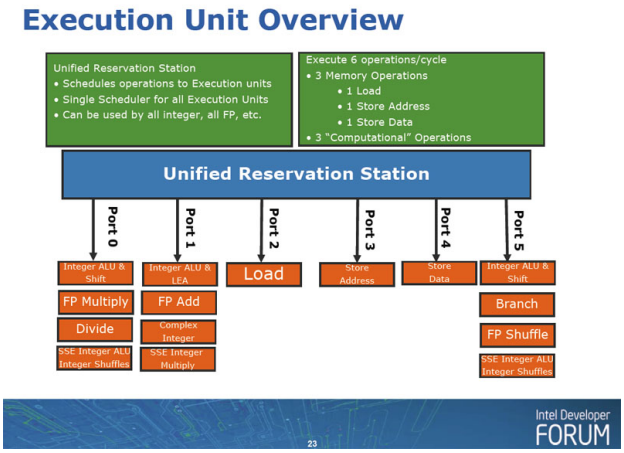

Fig. 5. Nehalem's Superscalar Architecture with an additional dispatch port since Core and the revival of the `STORE ADDRESS` port. The number of execution units per port also increased in this iteration.

`LOAD`, `STORE DATA`, and `STORE ADDRESS`. The execution ports have two or three units per port. These units were the `Integer Excution Unit (IEU)`, `Floating Point Unit (FPU)` and the `Jump Execution Unit (JEU)` (Only found on one of the ports). Core Architecture didn't change much in this unit. One large difference is the inclusion of embedded address generation units. With this, rather than having a `STORE ADDRESS` unit, Core only needed a `STORE DATA` or simply `STORE` to save data to RAM. The dispatch port connected to the `STORE ADDRESS` unit in Pentium M was freed up in Core and replaced with another execution port. This port was connected to both an `IEU` and `FPU`. This meant that Core could handle three integer instructions per clock cycle compared to Pentium M's two. See Figure 4 for a graphic of Core Execution.

Nehalem had far more changes from its predecessor then Core, but most of them were fairly small. Nehalem added another dispatch port hooked up to a memory unit, `STORE ADDRESS`, adding this unit back into the execution phase. To increase the number of operations, each of the three execution ports received at least one more unit, bring the count per port up to four, and the total count of execution units to twelve. Figure 5 is an image of Nehalem's superscalar architecture.

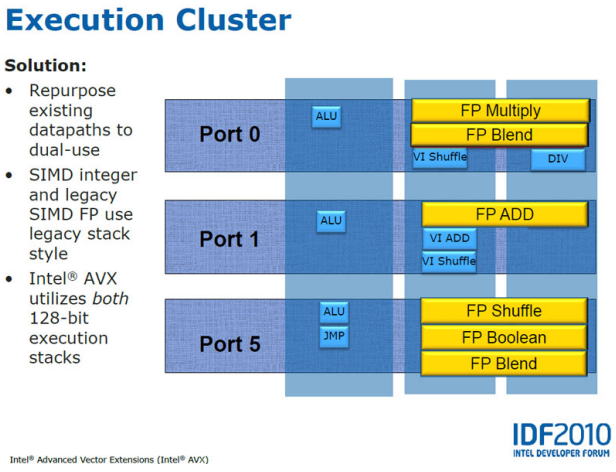

Fig. 6. Sandy Bridge's execution units. Originally, Intel faced a problem with the addition of 256-bit instructions. The above image displays their solution.

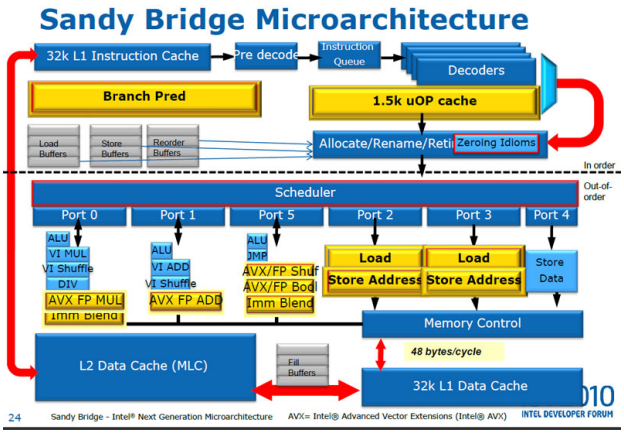

Fig. 7. Sandy Bridge's data pathway. Here, focus on the Scheduler and the dispatch ports connected. In the first two memory ports, both `LOAD` and `STORE ADDRESS` can be executed. Combining these results in a 256-bit bus space for instruction execution.

Sandy Bridge nearly reworked the entire execution process. To handle the new 256-bit instruction introduced in Advanced Vector Extensions (AVX), the 128-bit ports needed to be re-imagined. Rather than simply increasing the bus width on each of the ports and the units, Intel decided to take a modular approach. To handle the 256-bit instructions in execution, the structure of each unit was slightly modified where units were structured into stacks. Each port now contained four to six units sorted into three stacks. The last two stacks could be merged to form a 256-bit instruction space out of two 128-bit execution units, allowing for execution of these instructions. See Figure 6 for clarification of this. Similarly, to load or store these instructions in RAM, Sandy Bridge contained both `LOAD` and `STORE ADDRESS` units on two ports. This provided a two-fold benefit. For 128-bit commands, two `LOAD` or `STORE` operations could be executed at once, decreasing any down from waiting for these units to finish. Alternatively, a 256-bit instruction or address could be loaded or stored by combining these two blocks into one unit, just as in the execution ports. See Figure 7 for a visual representation of this.

V. Conclusion

Since the introduction of MIPS and single-cycle CPUs, commercial CPUs have improved greatly over the past 50+ years. From the Intel 4004 in 1971 to Sandy Bridge in 2011 and Skylake this year, CPUs have shrunk in size and grown in performance and complexity. This report touches on a couple of the many enhancements used in the Sandy Bridge microarchitecture.

A. Possible Next Steps

Future work on this research could involve a more in-depth pursuit of any of the following areas/topics:

- Intel Tick-Tock
- Micro-operation Cache (uop cache)
- Branch Prediction
- Advanced Vector Extensions (AVX)
- Buffers
- Register renaming

Acknowledgment

The authors would like to thank the teaching team for being helpful and supportive throughout the semester, aiding us in our pursuit to understanding computer architecture. We would like to give a special thank you to Kyle Flores, the NINJA who suggested this interesting research topic.

References

[1] Patterson, and Hennessy. *Computer Organization and Design: The Hardware/Software Interface*, N.p.: Elsevier Science, n.d. Web.
[2] http://stackoverflow.com/questions/454071/what-should-i-know-when-switching-from-mips-to-x86-assembly
[3] https://courses.cs.washington.edu/courses/cse378/09au/lectures/cse378au09-07.pdf
[4] http://cs.iupui.edu/~durresi/CSC3501_07/8_3501_07_2.pdf
[5] http://www.realworldtech.com/sandy-bridge/
[6] http://www.anandtech.com/show/1647/7
[7] http://www.hardwaresecrets.com/inside-the-intel-sandy-bridge-microarchitecture/