

## Q&A

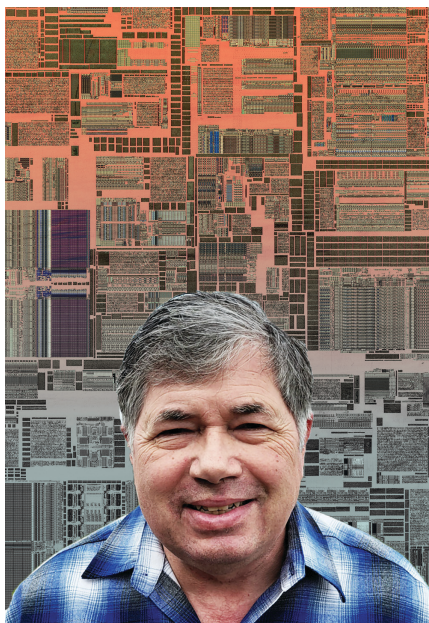
# Achievement in Microarchitecture

*David Papworth, a 30-year veteran of Intel, on what led to the P6 microprocessor and how that changed the microarchitectural paradigm.*

**D**AVID PAPWORTH, RECIPIENT of the 2022 ACM Charles P. “Chuck” Thacker Breakthrough in Computing Award, accepted a big job in 1990 when he joined Intel’s P6 microprocessor team as lead designer. The P6—commercialized as the Pentium Pro—was intended to leapfrog microprocessor design, and it did. Thanks to Papworth’s broad understanding of the hardware-software interface and adroit leadership of more than 500 architects, designers, validators, and engineers, the P6 introduced a new microarchitectural paradigm that is still in use today. Here, Papworth recalls how it all went down.

**In the 1980s, before joining Intel, you worked at a startup called Multiflow, which pioneered Very Long Instruction Word (VLIW) architecture. VLIW exploits instruction-level parallelism by enabling the compiler to schedule pipelines of instructions across different functional units—a technique known as superscalar processing. How did VLIW influence your work on Intel’s P6 microprocessor?**

The main thing Multiflow did that was carried forward into the P6 was the idea of a very wide superscalar. But Multiflow was all about scheduling things in software and doing as little as possible in the hardware. By contrast, the predecessors of the Pentium Pro were more of the mindset that “We can build



this, and the software will follow.”

There was a group of us—Bob Colwell (<https://bit.ly/3sEzgwc>) and myself, in particular—who had experienced how effective it can be for hardware and software to work together. We had a pretty good sense of what software can do, and what it expects from the hardware. We also had ideas about how hardware could exploit parallelism and use runtime information to improve scheduling. We worked through the challenges of trading off between hardware and software while still maintaining compatibility with the PC software base.

**One of the main innovations the P6**

**introduced is the paradigm of decomposing instructions into sequences of micro-operations. Can you explain how that works?**

The x86 instruction set has some very complex instructions. Imagine a “hyperbolic arc tangent instruction.” It’s easy to express the software intent as an instruction, but the required set of actions is way more than any practically realizable hardware can do in one step. That means it’s going to be a sequence of simpler things, whether you like it or not.

So, you have a complex instruction that does a load from memory and some sort of calculation: “Add to the BX Register the contents of this memory location over here.” In order to execute it, both of those values have to be available. That was no problem for the older Intel 386 and 486 pipelines, which were designed to execute everything in order.

What we did as part of P6 was to add out-of-order execution, which means we’ll do what we find in any order we feel like so long as the values are there. If they’re not there, we will just put it aside, move on to the next thing, and try to do that.

**So you’re not just converting X86 to RISC instructions and executing them in sequence.**

Not at all. The essence of micro-operations is twofold. One is to decompose complex instructions into what the hardware can actually do. The other is

to split them up into what, in software, are called data precedence arcs. So, you have the add operation, which is simple, and most machines can do that directly. There's also a load that goes with it: "Get this value from memory and prepare it to be put on the other side of the adder."

Instead of executing those two operations at the same time, we broke it up. We'll do the load when we can, and oftentimes that's well before the other side of the add is ready. And sometimes it's not. Either way, you don't want to sit there twiddling your thumbs. You can look at ahead, find the next instructions, and do them.

**And there are no paired pipelines for all these instructions, just a bunch of functional units to which operations are scheduled based on their availability.**

Right. Things can execute when their operands are ready and there's a functional unit ready to handle them. This is controlled by the process of register renaming, which takes the data precedence graph expressed by the software and encountered at runtime and maps that onto resources capable of containing that result for as long as it's needed.

**You also introduced some important validation and testing protocols.**

When Intel launches a successful microprocessor, a couple years later, it will be selling a hundred million microprocessors a year. Let's imagine you have to recall two years of production. That's 200 million microprocessors, each of which costs on the order of \$100 to service and replace. That's \$20 billion!

**"We worked through the challenges of trading off between hardware and software while still maintaining compatibility with the PC software base."**

Now, you can't sit paralyzed and not launch the thing. But unless you're Google or Apple and can make the software work around your microprocessor, you're just deathly afraid of that conundrum, so you do as much as you can to validate it pre-production.

**Building something as complex as a microprocessor requires a lot of juggling when it comes to requirements and constraints. Can you talk about some of the design trade-offs you made?**

I think the simplest example is 16-bit performance. The Intel 8088 was one of the company's most influential microprocessors. It was a 16-bit computation machine, and it had lots of quirks. For example, it would clear the upper byte of a register, then load something into the low byte of that register and read it as a composite thing. That causes horrible violence to the way we built and executed our register rename table, and there's really no reason to do it.

So we decided to deprecate it—to say, "We'll make it work, but it doesn't have to be fast." Our reasoning was that the workstations that used the Pentium Pro would be set up to run with modern software, but their compilers could deal with lower performance in that area and still be compatible with a 20-year-old version of Lotus 1-2-3. We thought it would be fine to make that performance trade-off, but the market taught us it wasn't entirely fine. Because the first thing they did with the Pentium Pro is run all of these old DOS benchmarks, and some of them didn't look very flattering.

**Is that something you did differently in subsequent iterations of the Pentium?**


Yes. As an architect, you have to design machines that can run a lot of software. Perhaps you'd like to do floating point really, really well. Do the people running on a PC or even a workstation really care? Some do. It sells computers. You can say, "Hey, LINPACK gets this great number." But at the end of the day, you pick, as best you can, a bunch of performance benchmarks, tailor the pipeline to do that, and then see how it works out.

**After the Pentium Pro, you worked on the Pentium 2 and 3 and 4—at which point you launched a second career**

**"I found I could be extremely helpful to the lawyers in explaining complicated technology, and I liked using the other side of my brain."**

**helping Intel's legal department defend against patent cases. What was that like?**

The legal work wasn't as stressful as building microprocessors. I found I could be extremely helpful to the lawyers in explaining complicated technology, and I liked using the other side of my brain. I was also good at staring down a plaintiff's attorney during depositions and answering questions truthfully without giving them the sound bites they were looking for. In federal case depositions, it's a nine-hour day, and you spend seven hours on the record. The lawyers are trying to catch just three extra words that they can take out of context and put in a brief. I was skilled at conveying complex technology in legally artful terms, and I had a steady stream of this work for many years.

However, the patent litigation landscape slowly changed over this time to be less favorable to plaintiffs and more favorable to defendants, particularly in difficult districts such as Marshall, TX (<https://bit.ly/45xnTVg>). By 2019, the number of high-profile cases had fallen off dramatically, and then in 2020, COVID hit. At that point, I was 64 years old, and with the quarantines, it wasn't a good time to get back into big microprocessor design projects. So I retired from Intel, and now I spend my days on my farm looking out over the fields and raising my grandson with my wife Katie. 

**Leah Hoffmann** is a technology writer based in Piermont, NY, USA.

© 2024 ACM 0001-0782/24/01