OBLIQUE PERSPECTIVE

Why RISC Is Doomed

After Ten Years of Philosophical Debate, Reductionist Arguments Falter

By John Wharton, Applications Research

"RISC: any computer announced after 1985"
– Steven Przybylski, a designer of the Stanford MIPS, as quoted in *Computer Architecture: A Quantitative Approach* (Hennessy and Patterson, 1990)

You can tell a value system is in trouble when its most zealous defenders start arguing semantics in order to prove they'd been right all along. So the "R" in RISC didn't *really* stand for "reduced," as in decreased in number, according to John Mashey's recent *Viewpoint* column (μ*PR* 3/25/92, p.21). What it now appears to have meant was "restricted," as in having a memory architecture sufficiently restrictive so as not to require trap handlers to fix two page faults at once. No doubt that distinction would be important to me—if I had to write my own trap handlers.

But at least now we now have absolute, objective, incontrovertible proof that CISCs aren't RISCs, and that they're not converging either. Fine—except that I never heard any credible claims that they were. I *was* a little surprised last year to hear one of the high priests of RISC tell a microcomputer workshop that he never felt the 386 architecture was all that complex to begin with, but that's beside the point. The real issue is whether computer *implementations* will converge, and the evidence is growing that they will.

And we now have a set of formal rules we can apply to CPU architectures to see which are among the most pure—a veritable catechism of architectural orthodoxy. To count as RISC it now seems an architecture must be less than six years old, restrict the flexibility of its addressing modes, limit the number of memory references per instruction, and define at least nine floating-point registers separate and distinct from those for integers. No doubt this would all matter, too, if philosophical purity were more important than results.

When the line defining RISC as "any architecture marketed after 1985" first surfaced, most of us thought it was meant as a joke. But now RISC proponents seriously seem to be claiming that creation date is a measure of architectural style. Was the first computer to use your instruction set introduced after 1986? If so, that's evidence your machine is RISC. Imagine! Guaranteed immortality of your ideas, all at the stroke of a pen. It's too bad our founding fathers didn't think to define "Democracy" as "any political system introduced after 1776." Had they done so, Communism might never have posed a threat to world order.

Shifting Ideological Tides

Several things bother me about Dr. Mashey's *Viewpoint*. First are the small factual errors used to strengthen its case, like the claim that one x86 instruction can safely modify the next. It can't. The 8086 architecture has always required a branch instruction to commit modified code to the instruction stream. No x86 processor ever built would execute code as the *Viewpoint* describes, so "software that uses this [immediate modification] capability" does not exist.

My second complaint is that the comparison table used to derive its architectural distinction rules omitted CRISP, the 960, Clipper, and ARM—four CPUs that would otherwise have disproven its case. CRISP, which the *Viewpoint* elsewhere calls a "well-documented ... RISC CPU," fails to qualify as RISC on at least seven of the eleven rules defined in the *Viewpoint*. The Intel 960 is a RISC architecture according to any conventional criteria, but it fails at least six rules—by providing *more* than the minimum RISC requirements. Since various members of the 960 family are in turn faster, cheaper, and sell in higher volume than any of the workstation processors that *were* listed, students of computer architecture would do well not to ignore its existence.

Third, the column focuses just on the philosophical issues of whether RISC architectures are *different* from CISC, not whether they're *better*. By limiting its scope as it does, the column diverts attention from the performance issues that matter most to users. Sure, one could build elaborate tables comparing color, shape, texture, and hardness to prove apples aren't oranges, and that they're not converging, either. None of that would matter, though, if your goal is to bake better pie.

And fourth, I get the discomfiting feeling that the criteria used to define RISC philosophy keep changing. When RISC was introduced I seem to recall a lot of emphasis on such issues as fixed-length encodings, decoupled memory accesses, delayed branches, and so on. These fundamental architectural differences were supposed to imbue RISC CPUs with some significant, inherent performance advantages: they'd run faster, be easier to design, move more quickly to new technologies, and allow more cost effective systems.

What follows is *not* meant to be another RISC/CISC

debate. Established CISC architectures are clearly crippled by design decisions that would never be made today. The issue is RISC vs. Reality. Ten years of theoretical debate should be plenty; what matters is practice, not theory. Whatever inherent advantages RISC may have should by now have been demonstrated, eliminating any need for philosophical debate. It's time for a sanity check to see whether the absolute, objective, incontrovertible arguments originally put forth on behalf of RISC have been proven true. The evidence—as measured by current CPU implementations and production systems—strongly suggests otherwise.

Canard 1: RISC Is Inherently Faster

The problem with CISC CPUs, the theory went, was that complexity adversely affects a CPU's maximum clock rate and average cycles per instruction—two key factors in the now-famous performance equation. But practice suggests theory was wrong.

Theory: Complex microcoded control logic precludes single-cycle execution.

Practice: Microcode and hardwired control logic are not incompatible. RISC devices like the 960 implement many single-cycle instructions, and escape to microcode only for non-trivial operations. The 486 and 68040 bypass microcode to do simple stuff in one cycle.

Theory: Complex instructions need complex ALUs, which limit the clock speed. RISC thus omits seldom-used operations like rotate and decimal add.

Practice: ALU complexity would limit clock rates only if ALU logic was part of a critical timing path. Apparently it's not. In cranking up the R4000 clock, it was the address translation and cache control logic, not the ALU, that had to be superpipelined. The R4000 designers found they could actually double the ALU width to 64 bits—and still finish operations in one cycle.

Theory: Complex instructions are slow to decode.

Practice: Possibly true, if all decoding is done at once. But CISC CPUs can "crack" instructions in two steps by introducing a new predecode pipeline stage.

Theory: Memory accesses require multiple chip crossings, consuming multiple CPU cycles. Data loads should therefore be isolated from data usage.

Practice: On-chip caches accessed early in the execution pipeline can return data to the ALU before it's needed. Several RISC vendors are now starting to brag that their latest implementations can load cache data and use it immediately. The 486 was actually the first CPU to load data during one clock cycle and use it on the next—comparable to a RISC with no load-delays, and without requiring separate instructions. The 68040 can load data, modify it, and store the result back to cache during a single one-cycle instruction—three times faster and more efficiently than standard RISCs.

Theory: Branches insert pipeline bubbles, which delayed branches can conveniently exploit.

Practice: With on-chip cache and good pipeline design, branches need not insert bubbles, as the i960CA proved. Superscalar designs actually make delay slots inefficient, since they keep at most one execution unit busy after the branch. Newer RISCs like Alpha leave delayed branches out. The Intel P5 uses a destination-address cache in order to process correctly predicted branches in a single cycle—with no delay slot.

Theory: Register windows reduce memory traffic.

Practice: Maybe so, maybe not. But the oversize register files needed for windowing *do* cause trouble as ports are added for superscalar implementations, and may even delay execution if the register file lies in a critical timing path. MIPS, the 88K, and essentially all RISCs introduced post-SPARC have decided register windowing isn't worth the trouble.

Theory: Fixed-length instruction formats are necessary for superscalar designs. With the right combination of specialized execution units, RISC CPUs could compute an operand address, access memory, and perform a register operation, all at the same time.

Variable-length instruction formats (it's said) preclude superscalar execution. 386 instructions vary from one to 15 bytes. In order to dispatch two x86 instructions with every possible alignment during every cycle, decode logic would need be to replicated 16 times.

Practice: Never mind that individual 68040 instructions can *already* combine address calculations, memory references, and register operations, and still complete in a single cycle—the equivalent of an order-three superscalar RISC. Never mind that they do so without the convoluted control logic needed for parallel decoding. Superscalar CISCs are still quite practical.

Decode logic need only handle the most common combinations of instruction lengths to get much of the benefit of parallel execution. Moreover, instructions can be predecoded as each is fetched from main memory, and then saved in cache in a RISCier format—expanded and aligned, perhaps, with "tag" bits identifying the operations to follow. The RISCy CRISP, i960CA, and Swordfish CPUs have used such tricks for years. The order-two superscalar P5 is believed to do so as well.

Theory: CPUs determine system performance.

Practice: High-end system throughput is largely determined by such issues as the memory hierarchy design, mass-storage latency, the video interface, and network speed. Unless all elements of a system are designed for balanced execution, speeding up just the CPU may not affect system throughput.

Canard 2: RISC is Easier to Design

Then there's the purported RISC design advantages. Don't simpler architectures inherently lead to a simpler design?

Theory: RISC CPUs use fewer gates, which makes them easier to design and faster to debug.

Practice: This claim was no doubt true back when microprocessors contained just a CPU; a handful of grad students could never have designed a 386 in one semester. But these days the CPU is one small part of a far more complex device. Both RISC and CISC microprocessors now contain full floating-point units, on-chip caches with messy coherency logic, complex bus interfaces with increasingly arcane protocols, and so forth. This is where the bugs appear. Getting Load, Add, and Branch to work is comparatively easy.

Theory: Uncommon instructions can be ignored.

Practice: The AMD 29000 and SPARC architectures omitted integer multiply and divide instructions in favor of simpler "step" primitives; the i860 left out even these. But time breeds wisdom; both the 29K and SPARC instruction sets have now been augmented to include integer multiply and divide. The i860 has not.

Theory: Since RISCs have faster design cycles, they'll be the first out the door with any new technology.

Practice: RISC's track record with new technology is not good. But suppose RISC *did* take a year less time to design. If competing design teams began at the same time, wouldn't RISC enjoy a one year head start?

Maybe—but this isn't the Olympics. Nobody fires a starter's pistol to make sure competing design teams begin together. If you're captive to an outside foundry it may be wise (if not mandatory) to wait for a new process to be proven before you stake your corporate future on it, but large IC vendors are not subject to the uncertainties of another company's technology.

Moore's law—that transistor counts double every two years—has proven quite dependable. Technologists thus know well in advance when it will become practical to manufacture CPUs containing 2 million transistors, or 5, or 20. With foresight and sufficient resources, there's no reason design teams can't begin years in advance to define the architectures and implementation techniques for the CPUs of tomorrow. This is what the P5 and P6 design teams apparently did. The ultimate rate-limiting step is thus production technology itself, and it has little to do with design time.

Canard 3: RISC is More Cost-Effective

And then, of course, there's the cost issue.

Theory: RISC CPUs use fewer transistors, opening up chip area for system enhancements like cache.

Practice: Right—but growing die sizes open up chip area on an ongoing basis, not as a one-shot deal. It's the '90s now, and transistor budgets are wide open. Whatever edge RISC once had is now lost in the noise.

Theory: Code density doesn't matter anymore.

Practice: Maybe not, if memory and cache are sufficiently large. But while microprocessors already devote more transistors to cache than any other single function, cache is still the most critical resource. Programs for RISC architectures typically expand by 20% to 90%, requiring proportionally more cache for a given instruction hit rate. Complex instructions and variable-length encoding would let smaller caches work as though they were at least 20% larger—and being able to trim a million-transistor cache by even 20% would make up for a lot of decoder and controller complexity.

Theory: Since RISC CPUs are smaller, they cost less to build and sell for less money.

Practice: Simple, scalar RISC chips undoubtedly *are* smaller than equivalent CISC devices. Just look at the integer core in a die photo of the i860XP (see μ*PR*, 6/12/91, p.7) Not the bottom half of the die—that's cache. And don't look at the FPU logic and graphics support in the top right quadrant, or the bus control, paging, and TLB logic in the top left. Just check out the integer core, that little block of the logic in the middle that consumes less than 5% of the total chip area. Had the i860 architects chosen not to cripple the part quite as severely as they did, that tiny little block might have been larger. On the other hand, the chip might then have sold better, and a full 1.5 million transistors might not then have been needed for cache arrays.

But price in today's market isn't determined by what chips cost to build, or even the defrayed cost of design and equipment. Price is determined by value. Most system costs are fixed. A CPU that multiplies system throughput will be worth quite a lot. The 486DX2 isn't cheap, selling for many times what it costs to build, but to those who buy it, it's a bargain. If RISC CPUs sell for less than some of their CISC cousins, it's in part because they have less value.

The Proof of the Pudding

But enough about theory. It's time for dessert. Proof of technological superiority should appear in the form of faster/better/cheaper production systems. Here RISC again fails to deliver on its initial promise.

Do RISC clocks run faster than CISCs? Not always. Intel had 100-MHz 486s running in the lab before 100-MHz R4000s were working. The fastest R4000 system made still has a 100-MHz internal clock, just 50% faster than a mainstream PC retrofitted with a 486DX2-66.

Can high-end RISCs significantly outperform a 486? On floating-point applications, yes—but FPU throughput depends more on implementation complexity than any underlying architecture. A 50-MHz 486 has SPECint92 ratings that are actually better than a SPARCstation 2, and comparable to mid-range workstations using the R4000 and HP Snakes. A 486DX2-66 delivers at least two-thirds the integer throughput of the fastest systems from Sun, MIPS, HP, and IBM. Joy's law—that workstation throughput will double each

year—started falling off track in 1989. If anything, the performance gap between RISCs and CISCs may be closing; Intel says the P5 should match the performance of an R4000.

Do RISC designs always go as smoothly as their proponents had expected? Apparently not. SuperSPARC was at least a year or two late and still can't meet its speed goals. HyperSPARC is long overdue, and SPARC Lightning was canceled outright.

Does RISC always move quickly to new technology? Hardly. The much-publicized ECL implementation of SPARC never was used in a meaningful system; the ECL R6000 was significantly delayed. Computer startup Prisma tried to build a gallium-arsenide version of SPARC. They failed and the company went bust.

And what about price? RISC chips generally do cost less than high-end 486s, but not because they cost less to make, and 486 prices may soon start falling like a rock. Not too surprisingly, first-tier system vendors charge about the same amount for fully-equipped workstations with comparable features and performance, whatever the CPU inside. If you're willing to chance a no-name clone, though, 486 systems can be assembled at price points RISC can't touch.

The Death of the RISC Chip Market

But what about the biggest promise of RISC, that its inherent technological superiority would overcome the head start of older architectures and soon displace established merchant-market microprocessors? RISC was once seen as the start of Intel's downfall. While RISC certainly does now dominate workstation markets once owned by the 68K, Motorola's marketing and production skills may deserve equal credit for this. In terms of displacing other merchant-market micros in computers, RISC has utterly failed.

The Intel 960 and AMD 29K each initially targeted a range of applications including UNIX workstations, but each was repositioned for embedded applications before release.

Sun initially positioned SPARC as a merchant-market processor and promoted the parts for open system designs. So far, though, Solbourne and everyone else who used SPARC to compete with Sun in an established market has failed to make a dent.

Motorola was the first major IC vendor to endorse the RISC concept. The 88K has since been all but abandoned, but not before the confusion it stirred up helped ensure the demise of the 68K. The Intel i860 raised reductionism to a new low. No one bought it, though, and the part was withdrawn from the workstation market.

The Precision Architecture is still essentially captive to HP. DEC has had trouble finding a second source for Alpha. IBM abandoned the RT, and the original RS/6000 architecture will be used only in proprietary multi-chip systems. The success of PowerPC in the open systems market can't be gauged until working devices exist.

The best and brightest hope for RISC, many thought, lay with MIPS. The MIPS architecture garnered early acceptance in systems built or planned by DEC, SGI, Olivetti, Bull, Prime, Compaq, and Acer. Alas, as one OEM after another backed away from the part, its prospects of success grew more dim. And with MIPS' acquisition by SGI, the architecture may have lost the neutrality system vendors demand of their suppliers, and its potential for outside success is now slim.

In a way, the failure of these RISC processors to break out of their early niches was inevitable. One of the fundamental precepts of RISC is that CPU architecture and implementation should *not* be decoupled, that in order to deliver the absolute maximum throughput, an architecture should map closely onto its intended implementation. Thus it was deemed advantageous to throw out any instructions and features that do not fit cleanly into the intended execution pipeline.

But design techniques will keep changing as long as technology continues to evolve. Whatever implementation techniques made the best use of yesterday's technology will likely *not* be the best for tomorrow's. RISC architectures therefore guarantee their own obsolescence. MIPS and SPARC begat POWER, which begat Alpha, which will beget something new—DEC's promise of a 25-year future for Alpha notwithstanding. None will dominate the open-systems market long enough for the critical mass of software needed for long-term success to develop.

At this point, then, if RISC survives, it will do so only as a multiplicity of incompatible proprietary products maintained by independent system vendors. This is a far cry from its promise. The future of RISC microprocessors in the open systems market is dead.

Bad Reviews

It's been ten years since RISC burst onto the scene. While the 32-bit desktop market has grown from zero to about 30 million units per year during that time, RISC has so far managed to penetrate less than 2% of the market. The i486DX2—introduced in just March of this year—is already shipping at about twice the rate of all workstation RISCs put together, with a cumulative total equal to one third of all the RISC workstations ever sold.

Film reviewers have a simple rule for sizing up new movies: "If nothing happens in the first ten minutes," they say, "nothing's going to happen." It's the same with computers. If an architecture fails to take over the mainstream computer market in its first ten years, it's not likely ever to do so. And an ideology with no future is doomed. ♦