Byte Addressing: From Laboratory to Practice

"Do you want software on this chip? Or not?"

John Mashey, a senior software engineer at Convergent Technologies, put the matter as starkly as he could. John Hennessy, an assistant professor of electrical engineering at Stanford and Chief Scientist at the startup MIPS Computer Systems, had just finished making a presentation on MIPS' planned products to the engineering staff at Convergent, a potential customer. Their plans, based on some very promising prototyping at Stanford, were intriguing, but Mashey saw what he thought was a critical technical flaw in the architecture of the first MIPS microprocessor chip. He pressed Hennessy on this, and became increasingly frustrated at the resistance he encountered. Why couldn't this bright young academic appreciate the problem that Mashey saw so clearly?

Company Background

Hennessy had led a group of Stanford graduate students and faculty in the architecture and design of one of the earliest "RISC" microprocessors. The chip was called MIPS (for Microprocessor without Interlocked Pipeline Stages) and had been carried through to fabrication at a local semiconductor company. (An academic paper describing the Stanford work is attached. It is "Hardware/Software Tradeoffs for Increased Performance," by Hennessy, Jouppi, Baskett, Gross, and Gill, from *Proc. Symposium on Architectural Support for Programming Languages and Operating Systems*, 1982.)

Following longstanding Silicon Valley tradition, Hennessy, with several members of his team, started a company to commercialize the chip and related technologies. Founded in 1984, MIPS Computer Systems planned to develop, market, and sell chips, workstations, and compilers, all based on the Stanford work. A critical step in the early going would be to try to get some potential customers, like Convergent Technologies, excited about the planned products, which offered great cost and performance advantages over contemporaneous alternatives.

The RISC Idea

In the early 1980's the idea of Reduced Instruction Set Computers was receiving a great deal of attention, both from university researchers (especially at Berkeley and Stanford) and from the computer industry. The key idea was that complex individual computer instructions, then very common in commercial computer architectures, were not necessary: not only were they rarely used in practice, but when they were needed, the compiler could use an equivalent sequence of simpler instructions instead. A computer designed to execute only the simple instructions could run at a higher clock rate (a lower cycle time).

Byte Addressing

The issue of memory addressing granularity was a perfect example of the RISC idea. Empirical evidence from real programs (see Table 4-1 in the attachment) showed that a large majority of memory references were to words rather than individual bytes. Commercial architectures of the time included instructions for referencing bytes as well as words. The Stanford group argued that a byte could just as well be extracted by first loading the enclosing memory word into a register, and then shifting and masking appropriately. (A slightly more elaborate sequence is required to store a byte into memory.) They further argued that eliminating the load-byte and store-byte instructions would shorten a long combinational logic path inside the processor (see Section 4.1 of the attachment), allowing a faster cycle time and therefore better overall performance.

Since a guiding principle of the RISC approach was that hardware should be streamlined for the common case, the Stanford MIPS team decided to support memory addressing only at word granularity, and to rely on compilers to supply appropriate instruction sequences when a program required access to individual bytes.

Mashey's View

Hennessy had argued forcefully in favor of this decision in his presentation at Convergent Technologies. But Mashey was skeptical. He pointed out that Hennessy's statistics were from user-level code only; Mashey suspected that byte references would be considerably more common in operating-system code. He also believed that there might well be hidden byte references and obscure pointer manipulations that would be difficult to ferret out. His experience with Unix on other architectures led him to the firm view that Hennessy's claimed performance benefit would not be worth the problems Mashey foresaw in trying to port Unix to a word-addressed machine.

The stakes were very high, as both men realized. MIPS Computer Systems depended on a rapid port of Unix for company success. Customers would not be interested in a machine that lacked solid operating system support. Writers of application programs would be unlikely to write for this new workstation unless it had robust Unix. But a performance sacrifice of up to 20 percent would be too high a price to pay for a feature if it turned out to make the Unix implementation only slightly easier.

Hennessy persisted in his view that the compiler could take care of the byte accesses: "Word-addressing is faster, saves some gate delays, and we can easily fix the software."

Mashey replied, "I guess. But would you like a reasonable Unix port, done quickly? Easy moving of the utilities, without extra maintenance forever? Some chance of getting third-party software without making them clean up code they won't do for a small company?"

Unconvinced, Hennessy responded, "Yes, but..."

"Do you want software on this chip? Or not?"