Comments on "The Case for the Reduced Instruction Set Computer," by Patterson and Ditzel

Douglas W. Clark and William D. Strecker

VAX Systems Architecture
Digital Equipment Corporation
1925 Andover Street
Tewksbury, MA Ø1876

September 198Ø

Patterson and Ditzel's paper [3] argues that a Reduced Instruction Set Computer (RISC) can be as cost-effective as a Complex Instruction Set Computer (CISC). In this note we suggest that several of their points are misleading, and present some evidence on the other side of the argument. We rely heavily, as did they, on the VAX-11 architecture [5] for examples.

The superiority of a RISC over a corresponding CISC will be very difficult to prove. Casual evaluation of cost and performance will not be sufficient unless the differences between a CISC and a RISC are extreme, which is unlikely. Paper designs will not be enough. A careful comparison between a RISC and a CISC would seem to us to require a complete design of the hardware and microcode for both, construction or simulation of the processors, the writing of compilers and possibly an operating system, and performance measurement across a variety of applications. Without this level of effort, claims of increased cost-effectiveness for a RISC are hard to support.

This is not to say, however, that this subject is unworthy of discussion or argument until such a comprehensive experiment is performed. The issues raised in the Patterson-Ditzel paper are interesting and important and architects of either style of computer can profit from a discussion of them.

Complexity vs. Size

The paper contraposes "Reduced" and "Complex". This is a false dichotomy. Can instruction set complexity be measured merely by instruction count? How about instruction count divided by the

number of data types? Does completeness (e.g. orthogonality of operator and data type) increase or decrease complexity? Our most serious criticism of the paper is that it contains no formal definition of a RISC or a CISC. In the absence of a definition of complexity the statement "...we shall argue that in many cases the complex instruction sets are more detrimental than useful" is meaningless.

Code Density

Patterson and Ditzel claim that code compaction, normally thought of as an advantage of a CISC such as VAX [1], is as easily achieved on a RISC, and that in any case dense code is not as important as it once was, thanks to cheap memory. But their case for the RISC boils down to the sentence "We suspect that code compaction can be as easily achieved by cleaning up the original [simple] instruction set," which is not convincing without supporting evidence. And while the cost of memory decreases over time, it will remain true that a small amount of cheap memory costs less than a large amount. Furthermore, for a given computer model memory is a per-system cost, while microcode development (for a CISC) is a one-time cost.

Dense code, of course, offers other advantages as well. Cache performance and paging performance will be better if there are more instructions per cache block and per page.

Different Languages Use Different Instructions

The paper makes the point (citing Shustek [4] and others) that "'very few opcodes account for most of a program's execution.'" True enough. But what about different languages?

The top 20 instructions for, say, Fortran may not make the COBOL hit-parade at all. The well-known Fortran benchmark Whetstone, for example, displays the usual kind of instruction frequency distribution on the VAX-11/780: the top 10 instructions account for 60% of all instruction executions, the top 20 get over 75%, the top 40 get over 90%. But those same top 10 account for a mere 8% of the instruction executions in a comparable synthetic COBOL benchmark. Whetstone's top 20 get 21% (as opposed to 75%) of the COBOL benchmark's executions. More telling still, the top 20 of Whetstone account for only 4% of the time taken by the COBOL benchmark.

So a multiplicity of instructions can help to support a multiplicity of languages. As a recent paper [2] points out,

> "For most programming environments, a system must be able to effectively support multiple languages. . . A single instruction set tailored to one particular language is constrictive, as it can make implementation of other languages difficult and inefficient."

Time is of the Essence

As Shustek argues in his thesis [4], the amount of time spent executing an instruction is more important for performance than its frequency of execution. He gives examples of rarely-executed instructions that consume a large amount of execution time. Replacing such an instruction by a multi-instruction sequence can make this much worse, and optimizing only those instructions that are executed frequently has the obvious hazard.

Here is an example from the VAX-11/780: in one time-sharing benchmark the instruction MOVC3 (a character-move instruction) accounts for less than 0.4% of the instruction executions, but for 13% of the time; it is 60th in the frequency ranking, 1st in the time ranking.

Ease of Compiler-Writing

A large instruction set can be justified by the desire to keep operators and data types orthogonal. Thus the VAX architecture includes, for example, six different exclusive-OR instructions: two- and three-operand versions for bytes, words, and longwords. Some of these are undoubtedly little used. But code generation in VAX compilers is simplified by having them all (this is attested to by VAX compiler-writers). Furthermore, once you have microcode for some of them, the others can be implemented very cheaply.

Microcode Size

The paper's PDP-11/40 - VAX-11/780 microcode size comparisons are specious. The amount of microcode in the 11/60 is nearly ten times the 11/40, even though the instruction sets are the same. The increase in microcode reflects (a) increased performance, (b) replacement of hardware by microcode, and (c) more elaborate diagnostic and console functions. The 11/780 also supports three instruction sets (PDP-11, VAX-11, and EDIT) and has much more complex memory management.

Increased Design Time

The paper's PDP-1 - VAX-11/780 design time comparisons are also specious. Ignoring the instruction set altogether, the VAX-11/780 hardware system and the VMS software are enormously more complex than the PDP-1 processor. Furthermore, there are numerous time-consuming processes in a large company designing products for high-volume manufacturing that do not exist in a small company designing products for low-volume manufacturing.

Increased Design Errors

It is unarguable that there will be more microcode errors in a large amount of microcode than in a small amount, all other things being equal. But if there is a small amount (RISC), somebody has to implement the complex functions somewhere. In a RISC, the compiler and run-time system would bear the burden formerly borne by microcode, and implementation errors would presumably turn up in this software.

Now one might argue that software is easier to write than microcode, and easier to change. And one might counter-argue that microprogram development tools such as microcode compilers will change this situation. But a complex function demanded by a user program must be implemented somehow, and errors will occur.

The detection of design errors depends in large measure on formal and informal test processes. This is as true for compilers as it is for processors. Such processes can readily be developed for processors. (How would Patterson and Ditzel compare the complexity of the VAX-11/780 microcode to that of, say, an optimizing compiler?)

Interface Level

One of the advantages of a higher level hardware-software interface (CISC) over a lower level interface (RISC) is that there is more opportunity to use specialized hardware to achieve improved cost/performance. For example, consider a CISC with a multiply instruction and a RISC without one. The multiply function would be performed on the RISC with a sequence of moves, branches, shifts, and adds. To speed up the multiply function on the RISC would require a speed-up of the whole processor while speeding up the multiply instruction on the CISC could be accomplished by adding specialized data paths and control. For some technologies and performance levels, the latter may be far less expensive than the former.

RISC and VLSI

In the absence of metrics, this section of the Patterson-Ditzel paper is unconvincing. Surely a simple instruction set can be implemented in less silicon than a complex one. This doesn't mean, however, that system cost-effectiveness is increased by reducing instruction set complexity.

The VAX INDEX Instruction

Anecdotal accounts of irrational implementations are certainly interesting. Is it typical, however, that composite instructions run more slowly than equivalent sequences of simple instructions? The paper reports that a sequence of several simple instructions can replace the VAX INDEX instruction with a 45% speed gain on the 780. This is a problem of implementation, not architecture. Fundamentally, after all, the implementation of the INDEX function with more than one instruction simply cannot take less time than the one-instruction version, assuming equal hardware in both cases. The explanation of this anomaly is that the 780's Floating Point Accelerator speeds up the multiply in the multi-instruction implementation, but doesn't see INDEX at all.

A Final VAX Fact

Patterson and Ditzel suggest that marketing strategy can increase the size or complexity of an instruction set. We can state from first-hand knowledge that this is not true for the VAX architecture.

References

[1] Dietz, W.B., and Szewerenko, L. Computer Family Architecture Selection, Phase IV Final Report: Comparative Evaluation of the Candidate Computer Architectures. Tech. Report, Computer Science Dept., Carnegie-Mellon University, Nov. 1979.

[2] Ditzel, D.R., and Patterson, D.A. Retrospective on High-Level Language Computer Architecture. Seventh International Symposium on Computer Architecture, La Baule, France, May 1980.

[3] Patterson, D.A., and Ditzel, D.R. The Case for the Reduced Instruction Computer. ACM SIGARCH Computer Architecture News, this issue.

[4] Shustek, L.J. Analysis and Performance of Computer Instruction Sets. Ph.D. thesis, Stanford Linear Accelerator Report 205, Stanford University, May 1978.

[5] Strecker, W.D. VAX-11/780: A Virtual Address Extension to the DEC PDP-11 Family. Proc. NCC, June 1978, pp. 967-980.