software impact. Instruction sets are becoming more orthogonal and supportive of high-level languages. The Intel iAPX 432 and Motorola 68000 are examples. It is not apparent to me that the newer minicomputers or microprocessors show careful consideration of software impacts.

I think what has happened is that we have packed more active devices on a chip and then covered our tracks by spraying the terrain with more instructions. If this were not so, the MC68000 would not have 55.3 percent of its instruction set used less than 0.1 percent of the time. The MC68000 instructions are far from being orthogonal—just try to use the address registers and data registers with the same instructions.

Smith wants the instruction set to be able to handle high-level-language requirements. This is a reasonable requirement and I think my design does just that! Without specific arguments to the contrary, it is not possible for me to address these areas.

The bit layouts of the instruction format were not included in the article, as Smith noted, but will be included in one of my follow-up articles.

Smith makes the point that with my design some type of modifier field is required for the different types of addressing modes. He is right. The modifiers are not instruction modifiers; however, they are addressing modifiers and will be discussed in a follow-up article on addressing modes and modifiers.

He also points out that my instruction set is not orthogonal for the MOVEM instruction, since it does not have a condition-code field. I’m afraid he has exposed a weakness in the design here. I could not see any possible use for a move multiple instruction on condition. There may exist such an animal but I could not justify creating it.

My instruction set will handle all of the fixed-point arithmetic that a user may want to implement (and it will handle it very efficiently). It does not provide floating-point instructions, these would be best provided by a floating-point or arithmetic-processing chip rather than by specialized instructions. The latter would be an unnecessary burden on the instruction set—an external chip is a better solution to the problem.

“The area to which I take the strongest objection is Fairclough’s ‘scientific’ approach to designing an instruction set . . . .” Smith states. He goes on to say that my approach ignores application requirements and language considerations. I strongly contest this criticism, for I used a large number of high-level-language programs, including applications programs, in generating the instruction groups presented in the article. I did not ignore these areas at all! I feel that my instruction set adequately addresses the applications and high-level-language problems.

Smith concludes by recommending that we should “first design machines to solve problems, then optimize.” I agree with this statement for the most part, but have one slight criticism of it. It has been my experience that we never get around to optimizing. I feel that if we have the data that tells us where and what to optimize (and we do), we should optimize first. My article attempted to provide data showing where to optimize an original design. There is a saying many readers may be familiar with—“We don’t have time to do it right, just time to do it over again.” In computer architecture, we don’t even have the time to do it over again. As a result, it is programmers who end up compensating for our design misjudgments—over and over again.

Dennis A. Fairclough

August 1982