Instruction set design clarified

Editor:

The article "Compilers and Computer Architecture" (July 1981 Computer, pp. 41-47) evoked some discussion between me and my colleagues. We are either involved in the selection of instruction sets for microcomputers or involved in compiler design and language-related areas.

I am personally in agreement with Dr. Wulf's general conclusions about regularity, orthogonality, and composability. There are, however, certain points which I feel require some expansion.

Is Dr. Wulf suggesting that instructions like FOR and CASE be deleted from machine architectures, or perhaps refined? Since Dr. Wulf has created an audience, I wonder if he would share with us his ideas on what machine features would provide good support for high-level data structures, string manipulation, and exception-handling. Perhaps we can avoid mistakes in the future.

Harold Massey
Senior System Analyst
Fairchild Microprocessor Division

Author's reply:

I acknowledge the implied criticism in Mr. Massey's letter that my article was rather nonspecific. In particular, I avoided making positive proposals for the design of an instruction set. The problem, of course, is that such suggestions depend upon many aspects of the instruction set design. I don't believe there is a single "right" way to design the instructions of a machine.

Perhaps I can amplify my point, however, with just one of the issues raised by Mr. Massey—namely, the FOR instruction. Similar remarks to the following apply to each of the other issues as well.

Clearly, loops are very important. The speed of a program often can be substantially improved by reducing the overhead for loop control. The ability to reduce this overhead is, in turn, a function of the instruction set design. But, should there be an instruction that "does" the FOR? I think not. I would vastly prefer a collection of instructions from which I can synthesize efficient loop control. To understand why, consider some of the differences between FOR statements as they appear in common languages.

- Some languages perform the loop body "at least once"; others do not.
- Some languages infer the termination condition from the sign of the step size; others have explicit syntax to signal whether the loop is counting up or down.
- Some languages compute the step size and the final value once (before loop entry); others require that these be recomputed at each point that they are used.
- Some languages restrict the iteration variable to being an integer type; others permit it to be of any type defined by the language.
- Some languages specify the value of the loop variable on exit to be that during the last iteration; others specify it to be the last value plus/minus the step; still others leave the value unspecified.

This list is not complete, but hopefully it illustrates the degree of variability in language semantics. Any single instruction that tried to accommodate all of the variations would be grossly inefficient. Moreover, a collection of instructions that span the alternatives—one for each variation—would be enormous. A few instructions that handled "the most common cases" would cause the compiler to perform surprisingly complex analysis to determine precisely when those special cases apply—especially when the instructions do not correspond to the natural semantics of the language being compiled.

The FOR statement is, of course, a very prosaic example. We all understand that the essence of this kind of iteration control is incrementing and testing the iteration variable. It's not hard to see how to provide a modest number of instructions that combine these operations—and perhaps even perform the test before or after the variable is updated.

This is the approach that I prefer. In most cases I believe it is possible to determine the "essence" of a construct appearing in several languages and to provide a set of instructions that, by doing the essential things well, can be used to synthesize more complex features. Again, I appreciate that these remarks are nonspecific, but I hope they clarify what I was trying to say.

William A. Wulf
Carnegie-Mellon University
Pittsburgh, Pennsylvania

---

**LETTERS TO THE EDITOR**

Draft standard defended

Editor:

I write with regard to the correspondence concerning the proposed microprocessor assembly language standard (IEEE Task P1694).* First, let me state that I participated actively in the development of the draft. I did so not because I believe in the use of assembly language except in isolated instances (I don't) but because I recognize the reality, confirmed by repeated surveys, that well over half of the code produced for microprocessors is written in assembly language.

There are two possible responses to this state of affairs—pretend that it does not exist or make the best of it. I submit that Messrs. Duncan, Rogers and Muhlbacher, and Deardoff prefer the former, while the IEEE Microprocessor Standards Committee is working towards the latter.

Laudable as the effort, which I strongly support, to reduce the use of assembly language may be, polemical attacks on proposals to facilitate its use appear to me to be inappropriate.

Andrew Allison,
Consultant
27360 Natoma Road
Los Altos Hills, California