Forget algebraic notation?

Editor:

After reading F. G. Duncan’s “‘Level-Independent Notation for Microcomputer Programs’ (IEEE Micro, May 1981, pp. 47-52) and the letters on the same subject in Computer (“‘Forget mnemonics?’” May 1981, pp. 8-9), it is apparent to me that the authors are confusing two notions. The first notion concerns the ability to properly express an algorithm. The second concerns the ability to express an algorithm in a form that may be directly converted into machine code for a variety of machines. Confusion of these notions has resulted, I believe, in a notation that satisfies neither criterion.

The problem of properly expressing an algorithm at various levels of abstraction is essentially no different for a microprocessor than for any other machine. I do not feel that the algebraic representation offers any new insight into the problem. In fact, it is more likely that one would intertwine levels of abstraction using the algebraic representation. The language elements of the notation are more closely akin to the physical machine than to the expression of the problem in its fullest abstraction. Structured languages such as Pascal go much further in their expressive ability than does the algebraic system.

The problem of expressing an algorithm in a form directly convertible into machine code for a variety of machines is precisely the problem being addressed by the assembly language standard group. Again, I feel that the algebraic representation falls short of solving the problem. Much of the expressive power one can obtain from an assembly language is lost in the algebraic notation. For example, the ability to reference addresses or subroutines by name is not provided. It is also not clear that the algebraic representation can be put in one-to-one correspondence with the machine code. This can lead to tremendous difficulties in the areas of standardization and validation. The fact that one can express internal machine operations such as stack pointer movement and program counter increments is certainly an interesting feature of the notation. This fact, however, seems totally irrelevant to the problem at hand. We could, of course, act upon Duncan’s suggestion and use three- or four-letter identifiers to represent the more formidable sequences. This could even lead to a method of defining a rudimentary assembly language.

It is my feeling that Duncan’s rather strong statements regarding the assembly language standard were a bit out of focus. The algebraic notation does not appear to be a viable replacement for assembly language. Elimination of ad hoc representations such as this should be an important function of organizations like the IEEE. In this regard, I believe we are meeting our obligation.

T. S. Berman
TRW, Inc.