FROM THE EDITORS

Counting on Today's Students

By Isabel Beichl, Editor in Chief

AS ALWAYS HAPPENS, I'M LEARNING A LOT FROM THE BRIGHT SUMMER STUDENTS WHO HAVE BEEN INVITED TO SPEND A COUPLE OF MONTHS AT OUR LAB. AND I'M ALSO LEARNING *ABOUT* THEM. THIS YEAR'S SURPRISE IS HOW MANY STUDENTS MAJORING IN MATHEMATICS OR COMPUTER engineering aren't required to take—early in their training—any courses introducing what I consider to be essential computer fundamentals. Examples include basic ideas regarding stored program instructions, arithmetic logic unit (ALU) registers, computer arithmetic, and storage. Nor is there any requirement to learn about what I think of as "real" programming in C, C++, Fortran, or CUDA.

Why does this matter? After all, a significant fraction of all science and engineering problems can be attacked using only wonderfully expressive software such as Mathematica and Matlab. And do we really need to know how a computer works to use one? All of this is true, of course, but I can think of two counterarguments—one cultural and the other quite practical.

On the cultural side, we have the fact that the stored-program computer is one of the most important inventions in the history of the human race. The notion that computation is an essentially mechanical activity is, of course, as old as civilization itself, if not older. But the idea that the instructions *themselves* are data that could and should be processed mechanically is a quite recent realization, dating only from the late 1930s. This is the concept for which we honor Alan Turing's memory, and it's this concept that really gave birth to the information age. No educated person, least of all one trained in science or engineering, should be unacquainted with the idea of the stored-program computer.

As for more practical issues, consider the fact that the fundamentals I'm talking about explain in a general way how a computer actually works. A rudimentary understanding of this is still essential to making efficient use of computational resources. Because of decades of the development of things such as optimizing compilers and scripting languages that find and correct our most egregious programming errors, we've become pretty lax about good programming. But, if there's to be any hope for bench scientists to make good use of multicore, highly parallel, hierarchical-memory, heterogeneous architectures, our casual attitude must change. Simply put, unless the atomic theory of matter is false, switching gate speeds won't get much better; so unless we're prepared to pay the electric bill for a large city, we must learn how to do parallel computing. And a fundamental prerequisite for writing better programs is at least a schematic notion of how a computer does what it does.

CiSE would be delighted to hear from our readers on this important matter. Those who are truly sincere can send me an email written using a text editor that you design and develop yourself, coded in assembler.