The Book Computer Structures: *Thoughts After 40 Years*

Gordon Bell, with Daniel P. Siewiorek

Editor: Dave Walden

> Sometimes I think the only universal in the computing field is the fetch-execute cycle.
>
> —Alan Perlis

I am always gratified when a computer scientist or engineer tells me of their recollections of *Computer Structures: Readings and Examples* that Allen Newell and I introduced in 1971 and *Computer Structures: Principles and Examples* that Dan Siewiorek created with Allen and I in 1982.[1,2]

Authors themselves often receive by far the most benefit from writing technical books, and so it was (and is) with *Computer Structures*. The indirect effects of my thinking about computers with Allen and Dan the way we did when developing these books had lasting imprints on subsequent computers at Carnegie Mellon University, Digital Equipment's PDP-11 and VAX computers, Bell's law describing computer classes,[3] and even the computer artifacts and their classification at the Computer History Museum.

The books posited the essential computer structure classes based on existing mainframes, minicomputers, and networks by studying every existing computer beginning with Babbage. These structures have remained constant through the subsequent introduction of newer computer classes of supercomputers, personal computers, workstations, cellphone-sized devices, the Internet, scalable clusters such as those that constitute clouds, and single-chip computers and multicore microprocessors. Bell's law came directly from studying how new kinds of computers came into existence and evolved in price and performance.

While writing the books we introduced two notations to help describe how just a small set of information processing component primatives are interconnected to form computers and information processing systems generally: PMS (for processors, memories, switches plus controllers, links, transducers, and so on) and the instruction-set processor, or ISP (for the processor's behavior). ISP was a forerunner of register transfer languages for simulation, verification, and design generally, although disappointingly, the computer architecture community did not adopt PMS for describing computers. Both were used at CMU for 20 years. Now on the 40th anniversary of PMS, I am firmly convinced that PMS does posit the correct, orthogonal, limited set of primitive information processing elements for all systems. No new elements or elements have been discovered!

Finally, the six years at CMU forever influenced how I think about information processing systems.

Writing Computer Structures

I cannot recall how or when the notion or structure of the book occurred and got started, but the manuscript was completed in 1971 prior to Intel's 4004. In 1966, I had just arrived at Carnegie Tech having spent the past six years at Digital Equipment Corporation (DEC) working on its first minicomputers and the PDP-6 multiprocessor computer for time sharing. In retrospect, I might have been burned out, but the book's basic motivation was that I came to CMU because I wanted to reflect on architecture and computer design.

My 1966 Carnegie Tech engineering notebooks included notes on a ''Case Book of Computer Systems' Structures'' for teaching computer architecture as the starting point. Many computers in *Computer Structures* were on that list. They also showed that I thought in terms of P, M, S, and I/O (K/Controllers for T/Transducers) primitives. This was not surprising because the DEC computers were modular and built to connect with other systems. I especially focused on the communication aspects with other computers and humans.

PMS

I started making PMS diagrams using my IBM Selectric for understanding and comparison. The typewriter constrained PMS block diagrams to a single letter with no lines around it, making them similar to chemical structures. A single diagram had to precisely show every possible attribute of a computer or component in an open-ended fashion, including data types, speed, scalability, memories, connections, as well as name, models, cost, size, and hardware technology. This was done by our open-ended use of `attribute:value` pairs. For example, `M(#0:7; function:primary/p; technology: core; cycle-time:1.5us; word length:12+1 bits)` is reduced to `Mp(core;1.5us)` when other attributes (such as word length) are implied.

The other requirements were that every PMS component expressed an identifiable and unique function, could be decomposed to subcomponents (ultimately to gates and bits that are of course just primitive PMS components), and could be aggregated to get networks and as yet unnamed things. These requirements made controllers (FSMs), data operators, transducers (I/O), and links for moving information.

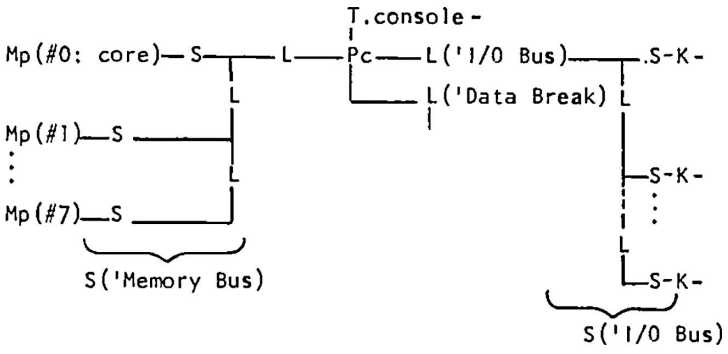


Figure 1. A simplified PMS diagram of the DEC PDP-8 from *Computer Structures* (1971).[2] Note the components without boxes. PMS diagrams can reveal exceptional detail, including all the linking busses, controllers, and so forth, and especially all the component specifications.

Using the following components we could build a complex structure such as a processor:

- `L/Link...` is the interconnection of just two components with properties such as simplex or duplex transmission, delay, data-rate, and technology. Examples include Ethernet, WiFi, and USB.
- `S/Switch...` is the component that allows multiple units to be interconnected through its input or output ports and is often just implied by multiple lines from a component—for example, multiplexors, busses, and cross-point switches.
- `D/Data...` is the unit to transform data values, such as arithmetic-logic unit (ALU), and floating-point multiply.
- `T/Transducer...` is the unit to transform the form of data (e.g. bits to pixels, key strokes to bits, and electronic bits to/from magnetic flux change bits in a device such as disk or tape).
- `K/Control...` is the component that evokes action in other components. Control is the program or hardware within a processor that defines an information processing system's behavior (e.g. a processor, computer, or network). All software is control[4] and likewise can be classified by its PMS function.[5]

The `P/Processor` (sometimes called a central processor, Pc) is a complex element consisting of M's including processor state, D's, S's, and K's that carry out the fetch-execute operation on a stored program.

And a computer is defined as a processor connected to a primary memory: `C/Computer := Pc—Mp`.

Happily, PMS is also compatible with the way our forefathers described their first computers using a four or five element diagram: memory, control, arithmetic, and I/O. Figure 1 shows a simple computer from the first edition.

The first six chapters stood the test of time. In the second edition, we rearranged a basic PMS diagram (Figure 2a) to abstract the essential performance parameters that are then used to create a 6D Kiviat diagram (Figure 2b) that let us easily compare the various computers throughout the book. For example, first-order performance models based on memory and execution times characterize performance differences between models of three major computer families collectively representing more than 30 implementations.

Foreseeing and Naming Future Computer Structures

In the second edition, Chapter 1 summarized the entire computer space through computer classes and their evolution with plots of fixed cost and function (represented by address space). We introduced a new class (monolithic microcomputers) with three members and anticipated a new class called monolithic systems (systems on a chip). The basic ways computers interconnect and operate have not changed in 40 years; they merely recur at different scales. We were able to partition, describe, and name the wide range of computers that would occur including multiprocessors and multicomputers (now called computer clusters). For example, through the mid-1980s, multiprocessors computers were formed by interconnecting cabinets or boxes containing modules of processor and memories; in the mid-1980s to 2005, multiple microprocessors were introduced that accessed memory via a common bus, which I called *multis*;[6] and in the mid-2000s, single chips were implemented that contain multiple processors sharing a common cache memory with access to an external memory or complete computers with processors and primary memory.

Multicomputers took a similar path with clusters introduced at Tandem and Digital in the 1970s. Today, nearly all high-performance computers, including supercomputers and cloud computers, are multicomputers. These building-sized multicomputers can be composed of almost a million interconnected computers. They are called by various names: clusters, constellations, and massively parallel computers.

ISP

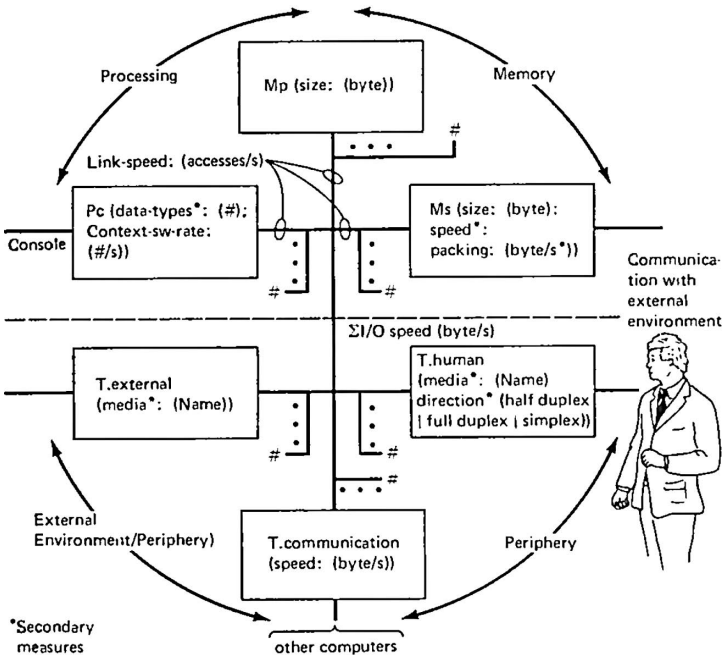
Although functional block diagrams give a general idea of what an information processing system does, we needed a notation to define processor behavior—its fetch-execute cycle if the goal is to understand or program a computer. The goal was to be able to describe a complete instruction set in just a few pages, including every bit of a complex operation. We did not want the diversion of creating an interpreter for these descriptions; the intent was to be able to simulate computers described in ISP.

Thus, the goal was for a student to easily understand, program, and compare instruction-set architectures (ISAs). The language was inherently parallel just like hardware—that is, every action occurred simultaneously, unless preceded by the word next, such as subsequent register transfer languages that it might have influenced. Many of the first edition's computers were described in ISP to show design variations in the computer space: CDC 6600; DEC PDP-8 (the simplest and most straightforward) and 338 display processor; IBM 1401 (the most foreign to me), 1800, and 7094; LGP-30; Olivetti Programma 101, an early programmable desk calculator; SDS 940; and an 8-bit pedagogical computer that I felt compelled to design. When the DEC PDP-11 was introduced, I wrote the ISP description that was included in the manual.

ISP was used in the computer family architecture (CFA) selection process that sought a standard computer for the Army and Navy.[7] All the final candidates were evaluated via the ISP simulator. This gave momentum to writing ISP descriptions, and every ISP in the second edition was simulated. Dan wrote the first large-scale ISP (PDP-11) that helped evolve the ISP simulator. Indeed, the PDP-11/70 ISP written for CFA executed all the PDP-11/70 diagnostic programs, and the ISP description was deemed more accurate than the programming manual.

Figure 3 shows a ISP description of the Manchester Mark I and illustrates the conciseness of ISP for describing instruction sets.

Whereas the first edition (668 pages) described 39 computers and took two years to write, the second edition (926 pages) reflected the explosion of computer types introducing new classes such as the personal computer and microprocessor (half of these descriptions written originally for the book, often by the architects themselves) took five years to write. We had at least three editor



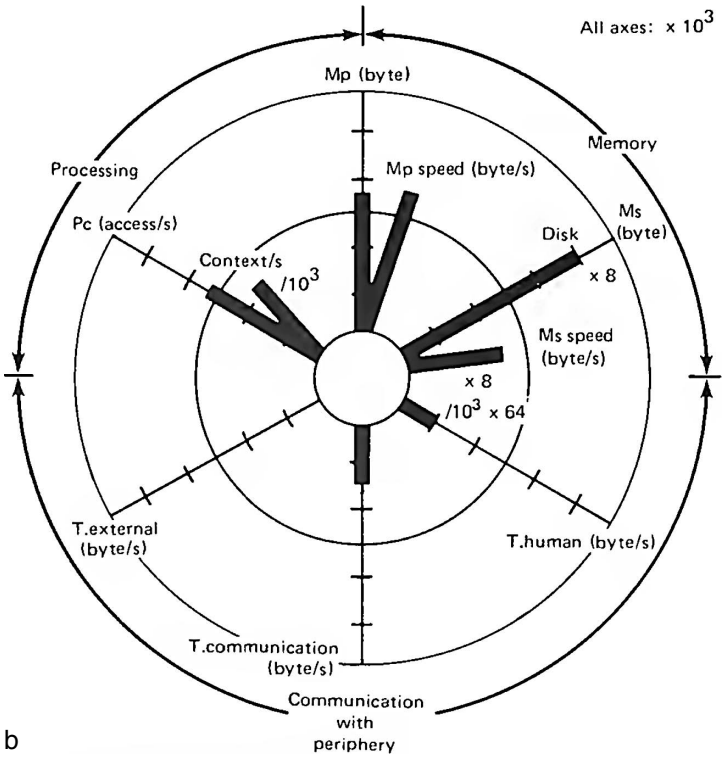


Figure 2. Transformation of structure summary into performance summary. (a) PMS diagram redrawn. Note components as boxes. (b) Major components of PMS diagram form the six major axes of performance: processing, primary and secondary memory, communication to humans or other computers (e.g. a network), and the computer's peripherals from *Computer Structures* (1982).[2]

```
MARK1 :=
    begin

! The Manchester Mark-I architecture is described.

! The Mark-I was an early (circa 1948) computer.

**MP.State**

    M[0:8191]<0:31>,

**PC.State**

    PI\Present.Instruction<0:15>,
        f\function<0:2> := PI<0:2>,
        s<0:12> := PI<3:15>,
    CR\Control.Register<0:12>,
    ACC\Accumulator<0:31>,

**Instruction.Execution**{tc}

    icycle\instruction.cycle{main} :=
        begin
        REPEAT
            begin
            PI ← M[CR]<0:15> next
            DECODE f =>
                begin
                #0 := CR ← M[s],
                #1 := CR ← CR + M[s],
                #2 := ACC ← - M[s],
                #3 := M[s] ← ACC,
             #4:#5 := ACC ← ACC - M[s],
                #6 := IF ACC lss 0 => CR ← CR + 1,
                #7 := STOP()
                end next
            CR ← CR + 1
            end
        end
    end
```

Figure 3. ISP description of the Manchester Mark I, or Baby, the first operational stored-program computer.

changes, and with each one, we had to fight to keep the price down (a large book in an oversized format) to about the same price as books half the size.

PMS and ISP: Beyond Computer Structures

PMS was used to communicate and compare computers in the books as intended. However, it failed to become a universal notation that other authors, such as John Hennessy and David Patterson,[8] would adopt, extend, and use. A text by R.W. Hockney and C.R. Jesshope did adopt and modify PMS so that a computer could be described in a linear text string to focus on parallel computers by adding a number of specialized processing components (e.g. pipelined processors using more capital letter primitives) versus iterating the basic primitives.[9]

At CMU, we thought in terms of PMS primitives. During the spring of 1971, an architecture seminar created the top-level specs, including the ISP of a large 16-processor, shared-memory multiprocessor designed for AI applications, C.ai.[10] The seminar stimulated the design and construction of a 16-processor multiprocessor computer, C.mmp. Tom McWilliams and Lowell Wood ultimately built the S-1 at Livermore along the lines of C.ai. Computer Modules, Cm*,[11] also came from this line of work, which included an extremely productive period of research from 1973 to 1986.[12]

Two useful "ah ha" moments occurred while writing *Computer Structures* and explaining the Unibus[13] that was used in the DEC PDP-11 and subsequent multis as a general structure for interconnecting all the PMS components as well as explaining the generality of general registers[14] implemented in the PDP-11 and other computers.

We had a virtual table of contents for the books that allowed students to refer to computers by their attributes. Thus, they could study technology, addressing, multiple processors, or multiple computers just by accessing those sections. Whether the idea was used anywhere else, we can only hope—especially for anthologies.

Computer Structures created a desire to actually collect and understand computers that resulted in the preservation activities I engaged in when I returned to DEC in 1972. Ken Olsen, the CEO of DEC, wanted to preserve MIT's Whirlwind, the memory-test computer used to test the first core memory, and the Lincoln Laboratory TX-0. I collected artifacts to trace technological change. We opened a DEC lobby "closet" exhibit in 1975, followed in 1978 by a DEC building lobby, and The Computer Museum in Boston opened in 1983. In 1995, all the artifacts were moved to Mountain View, California, as the basis for the Computer History Museum collection.

Gwen Bell was the museum founder and its curator through the movement to Silicon Valley. Throughout the museum's early collecting period, the *Computer Structures* PMS taxonomy with all types of computers, processors, memories, switches, links, and transducers served as the basis for the categorical naming of the artifacts and the identification of key artifacts to be collected. Although not every computer (such as the IBM 7090) or technology (such as rope memory) described by a chapter in the two books is in the

museum collection, a large number are.[15] Requiring everything to have a functional name is useful, aside from satisfying my desire for order and requirement for naming items. Where would biology or a natural history museum be without the life, domain, kingdom, phylum, class, family, genus, and species taxonomy?

Mario Barbacci took on the understanding, extension, and application of the ISP language for his PhD dissertation and created an interpreter that resulted in a workbook describing numerous computers.[16] However, the main impact of ISP was Barbacci's use to create a compiler that would generate actual chips based on compiling to register transfer primitives, not unlike the register transfer modules we had created with Digital in 1970.[17] This later system was a clear forerunner of modern silicon compilers because it enabled the simulation and compilation of an ISP description onto a single chip.[18] Barbacci and others even extended the ISP capability and combined it with PMS to program at a higher abstraction level.[19] Steven Rubin used PMS and ISP in his VLSI design book.[20]

Bell's Law

In writing the first *Computer Structures*, there was a strong focus on price performance, scalability, and work to disprove Grosch's law,[21] which to a certain extent was a pricing formula. The chapter on the IBM System/360 examined hypothetical multiprocessor computers I concocted to attempt to show that a scalable computer made from fewer component types would be more cost effective.

In 1971, we saw that computers existed in particular price bands and that these prices remained constant over time.[22] Bell's law didn't crystallize until the publication of *Computer Engineering*[23] and clearly seeing the two paths of constant price, increasing performance and constant performance, decreasing price that established a new class.

In 1975, the story was completed when computer price and performance were defined entirely by Moore's law.[24] The result is that new classes form every 10 years, including those (such as cell phones called smart phones) where a complete system is on a chip.

Allen Newell

Luckily for *Computer Structures*, I was able to convince Allen Newell to be part of our projects. The focus of the book became defining the basic information processing components, showing how these components were connected at every level from registers to computer networks to form systems, and defining a computer's behavior. Having these notations allowed computers to be discussed and compared uniformly.

Allen and I shared a love of taxonomies, and he was mainly responsible for the grammars of the PMS and ISP notations.

Aside from rewriting and translating every draft I wrote into readable prose, Allen and I shared a love of taxonomies, and he was mainly responsible for the grammars of the PMS and ISP notations that were the bedrock for describing computer structures and their behaviors. The collaboration with Allen took the form of weekly face-to-face meetings, many weekly phone calls, the creation of the ISP and PMS grammars used in the book's appendix, and the painful rewriting of my first drafts. Dan was the main author of the second book, and we collaborated similarly, but less intensive because my day job was heading R&D at DEC.

Herb Simon wrote about Allen's life and about our collaboration:

> It is perhaps not surprising that someone deeply concerned with program organization would become interested in computer hardware architectures, and Allen did. Nevertheless he regarded his work on this topic, which began with Gordon Bell's invitation in about 1968 to collaborate on a book on computer systems, as a diversion from his main objective. The strategy of simulating human thinking did not rest on any assumption of similarity between computer architectures and the architecture of the brain beyond the very general assumptions that both were physical symbol systems and that therefore the computer could be programmed to behave like the mind. Nevertheless, there are fundamental architectural problems common to all computing that reveal themselves in hardware and software at

> every level, for example, how to organize systems so that they can operate in parallel on multiple tasks with due respect for priorities and precedence constraints between processes.[25]

I sometimes regret the diversion and can only hope that he felt the resulting value of the book was worth his considerable effort. Fred Brooks' positive review of *Computer Structures* went a long way toward making us feel that our effort was worthwhile.[26]

Conclusion

Just what effect *Computer Structures* had on readers and students is hard to determine. It was the main text until Hennessy and Patterson introduced *Computer Architecture: A Quantitative Approach* in 1990.[8] Having a zoo of many computers in one place has been useful to the computer science community for access, comparison, and reference—though useful, is not quite enough. The main impact was the understanding that we gained in the writing that went into the subsequent computers we were part of, the taxonomy, Bell's law, and the various lines of research it stimulated.

Reference and Notes

1. This article was adapted from a talk, ''Recollecting *Computer Structures*,'' that I gave at Carnegie Mellon University's School of Computer Science when CMU granted me a Doctor of Science and Technology in May 2010.
2. C.G. Bell and A. Newell, *Computer Structures: Readings and Examples,* McGraw Hill, 1971; D. Siewiorek, C. Bell, and A. Newell, *Computer Structures: Principles and Examples,* McGraw Hill, 1982.
3. C.G. Bell, ''Bell's Law for the Birth and Death of Computer Classes,'' *Comm. ACM,* vol. 51, no. 1, 2008, pp. 86–94.
4. Until 2005, the most vexing question I've had since developing the PMS system has been the identity of software as a PMS component. To have a complete, functional taxonomy where the largest system component does not have a name within the framework is untenable. In 2005, my ''ah ha'' moment came while working on my own computing timeline of the important historical events. Software is the control element in a computer system! Control is the element that defines a system's behavior. Control exists at the lowest level as a finite-state machine; as a microprogram it defines the behavior of a processor; as an operating system to define the machine, including language transformations, for applications; and to define the function and behavior of the computer for use (behaving as a game, data store, communications link, calculator, word processor, etc.).
5. I believe the vast software world is nicely classified in terms of PMs—e.g. database and file systems are M's, compilers are T's, and operating systems or languages are just different computers that are hosted by a lower level computer.
6. C.G. Bell, ''Multis: A New Class of Multiprocessor Computers,'' *Science,* vol. 228, no. 4698, 26 Apr. 1985, pp. 462–467.
7. W.E. Burr, A.H. Coleman, and W.R. Smith, ''Overview of the Military Computer Family Architecture Selection,'' *Proc. Nat'l Computer Conf.,* Am. Federation of Information Processing Societies (AFIPS), 1977, p. 131.
8. J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach,* Morgan Kaufmann, 1990.
9. R.W. Hockney and C.R. Jesshope, *Parallel Computers: Architecture, Programming, and Algorithms,* Adam-Hilger, 1981.
10. C.G. Bell and P. Freeman, ''C.ai—A Computer for AI Research,'' *Proc. AFIPS Fall Joint Computer Conf.,* Am. Federation of Information Processing Societies (AFIPS), 1972, pp. 779–790.
11. C.G. Bell et al., ''The Architecture and Applications of Computer Modules: A Set of Components for Digital Design,'' *Proc. IEEE Int'l Computer Conf.* (CompCon), IEEE Press, 1973, pp. 177–180.
12. One monograph, 27 papers, 38 master's theses, and PhD dissertations. The cost was about $5 million for 13 years.
13. This came up during a discussion with T. Codd of IBM about a general model of communication among PMS components.
14. This came up during lunch with A. Perlis, a pioneer, one of the department's founders, and a member of the Algol Committee.
15. P. Pierce of Portland, Oregon, has used *Computer Structures* to guide his own collection of classic computers.
16. M. Barbacci and D. Siewiorek, *The Design and Analysis of Instruction Set Processors,* McGraw Hill, 1982.
17. C.G. Bell et al., ''The Description and Use of Register Transfer Modules (RTMs),'' *IEEE Trans. Computers,* vol. 21 no. 5, 1972, pp. 495–500.
18. D. Siewiorek and M. Barbacci, ''The CMU RT-CAD System: An Innovative Approach to Computer Aided Design,'' *Proc. AFIPS Nat'l Computer Conf. and Exposition,* Am. Federation

of Information Processing Societies (AFIPS), 1976, pp. 643–655.
19. M.R. Barbacci, C.B. Weinstock, and J.M. Wing, ''Programming at the Processor-Memory-Switch Level,'' *Proc. 10th Int'l Conf. Software Eng.* (ICSE), IEEE CS Press, 1988, pp. 19-29.
20. S.M. Rubin, *Computer Aids for VLSI,* R.L. Ranch Press, 1987, revised 1994.
21. Grosch conjectured that a computer's performance increases as the square of the price.
22. C.G. Bell, R. Chen, and S. Rege, ''The Effect of Technology on Near Term Computer Structures,'' *Computer,* vol. 5, no. 2, 1972, pp. 29–38.
23. C.G. Bell, J. McNamara, and J.C. Mudge, *Computer Engineering: A DEC View of Computer Design,* Digital Press, 1978.
24. G.E. Moore, ''Progress in Digital Integrated Electronics,'' *Proc. Int'l Electron Devices Meeting,* IEEE Press, 1975, pp. 11–13.
25. H. Simon, *Allen Newell, 1927–1992,* Nat'l Academies Press, 1997, pp. 157–158; http://www.nap.edu/html/biomems/anewell.pdf.
26. F. Brooks, ''Computer Structures: Readings and Examples,'' book review, *Computing Rev.,* May 1971, pp. 245–248.

Gordon Bell *is a principal researcher in the Microsoft Research Silicon Valley Research Group. Contact him at gbell@microsoft.com.*

Daniel P. Siewiorek *is the Buhl University Professor of electrical and computer engineering and computer science at Carnegie Mellon University. Contact him at dps@cs.cmu.edu.*