In a frank, often droll look at standards, DEC's former vice president of engineering concludes that standards really are the best tools for technology evolution. It's how we use them that causes problems.

Special Feature

Standards Can Help Us

C. Gordon Bell, Encore Computer Corporation

Standards are constraints that ensure the evolution—not revolution—of computing. They save design time by narrowing the search for new products and processes and permit us to build on past work rather than start each new design from silicon. Standards provide a real intellectual discipline that pushes computing developments further and faster.

The lack of standards impedes technological progress and lowers productivity. Redundancy in product development ties up critical resources in the reinvention of trivia, and solutions to hard, important problems such as those involving speech and video communication, intelligent programs revolutionary machines, and fully automatic production get short shrift. I have to wonder: Is it a shortage of engineers or a shortage of leadership that creates disconnected, overlapping, low-technology products?

As an official IEEE-CS Computer Pioneer (recognized for Computer Design, Compcon Spring 82), I urge the engineering establishment to see that key standards are set for the evolution to the next generation information era. These targets are critical for productivity and maintaining a healthy information processing industry.

I see the role of standards in today's computer evolution as critical, particularly in silicon chips and wafers, microprocessors and their buses, LANs, Unix, and Lisp because these standards areas constitute barriers that are impeding technological progress. What I'd really like to see is a Compcon devoted solely to promoting the understanding of various standards and the standards process. I hope this article at least makes a start in that direction.

Bell presented a preliminary version of this article as the keynote speech at Compcon 84, February 28, in San Francisco.

Standards: The basis of today's evolutionary generation

The first two computer generations were characterized by complete vertical integration. Each computer company or division designed and manufactured circuits, peripherals, hardware systems, operating systems, languages, and applications and created unique, proprietary standards. Today, standards provide clear constraints for building products within a given strata and segment. For example, a spreadsheet industry has evolved common data format standards for programs built by different software companies—to allow not only competitive programs but also the ability to interface with plotting and database programs. Thus, two other industries can form.

Today's generation is characterized by a large set of product-segmented industries that are organized by levels of integration that form strata, which in turn are formed by standards.[1] Entrepreneurial energy drives industries and venture capital releases their energy.

Unlike the previous generations, when processors and memories constituted a large fraction of system cost, microprocessors are comparatively small and standard. Paradoxically, then, many more creative computing structures are possible because of standardization. But the structure is creative not the building blocks, standards allow us to use bricks, not a collection of designer-created "pet rocks."

Today, the following eight levels of integration form the industrial strata. The bottom four are hardware and the first four software and applications. Each level has many product-segmented industries. A given organization usually excels in only a few strata/product segments or types of systems, which are a collection of many strata.

0018-9162/84/0600-0071$01.00 © 1984 IEEE

- *Discipline and Profession-Specific Application.* CAD (for logic design)
- *Generic Application.* word processing, electronic mail, spreadsheets
- *Third-Generation Procedural Programming Languages.* Fortran, Basic, C
- *Operating System.* base, communication gateways, databases/CP/M, MS/DOS, Unix.
- *Electromechanical.* disks, monitors, power supplies, enclosures/eight-inch, five-inch, three-inch(?) floppy disks; five-inch Winchester disk.
- *Printed Circuit Board.* buses that are synchronized to micro and memory/S-100, Multibus, PC Bus, Multibus II, and VME
- *Standard Chip.* micros, microperipherals, and memories; evolution of Intel and Motorola architectures that are synchronized to the evolution of memory chip sizes/8080 (4K), Z80 (16K), 8086 and 68,000 (64K), 286, 68020, and NS32032 (256K)
- *Silicon Wafer.* bipolar and evolving CMOS technologies (proprietary, corporate process standards . . .require formalization to realize a silicon foundry-based industry)

Some standard dos and don'ts

Before I go on to discuss specific standards, In think we need to look at ways to make the setting and adoption of standards easier. The following "rules" are suggestions about what we might observe when setting and using standards. The list isn't complete or necessarily consistent; it merely gives my observations and personal prejudices.

1. Either make the standard or follow the standard. The IBM PC emerged immediately as a standard. It came at a propitious time—concurrent with a processor capable of accessing almost a megabyte of memory, the 64K chip, and wide-scale availability of five-inch floppy disks and just prior to five-inch Winchester disks. PC software demand dramatically increased because people could work on useful applications instead of reinventing and transferring old operating systems for hardware-idiosyncratic PCs. The standard is fine for at least five years; *Byte*'s recent editorial about the compatibility craze[2] was wrong.

2. Be prepared to react quickly and follow when the de facto standard changes. Those who follow IBM standards might remember the early 70's when Amdahl, among others, had to scrap its 360 design as IBM evolved and introduced the 370. History may repeat when future IBM PCs make current products obsolete by providing a fully upward-compatible product with more capability, such as virtual memory.

On the other hand, Apple's Lisa and Macintosh designers are to be congratulated for *not* following the IBM standard.[3] People and organizations who, for the sake of progress, deviate from simple evolution are essential, even though such deviations like those made by Amdahl at Trilogy are risky and sometimes fail. Meaningful deviations don't merely repackage old ideas or provide the same simple function; they are large-scale, well-thought-out projects that provide much more capability and take us in a different direction.

3. Change the standard when it's wrong. IBM has finally adopted ASCII codes after years of using Extended Binary Coded Decimal Interchange Code,[4] and evolutionary extension and remnant of the card era. This adaptation means much mainframe translation work as the PC and mainframe communicate.

4. Make somebody [person(s) or organization(s)] responsible for defining, implementing, and maintaining a standard. Ethernet is a good example. Xerox and DEC needed it as the backbone of their product strategies, and Intel needed it to sell chips. Rarely has an interconnection standard been as important as the LAN. How it is implemented should be moot—the modulation (broadband versus baseband) and topology (buses, rings, trees, or centralized switches) have only minor impacts on cost and performance of the total system. Yet the 802 series, which addresses topological differences,[5,6] became a focus for ideological differences that lead to time-consuming arguments. The work building real systems is deferred by at least five years, with billions of dollars lost through redundancy, lack of productivity, and poor communications.

5. Minimize the number of organizations responsible for a standard. Our industry lives in a chaotic world of de facto standards that emerge from particular companies (the S-100 Bus, dialects of Basic), industry standards—which are often simply IBM's standards—and "vanity" standards, those developed by individual companies to box users into particular systems. Vanity machines— personal computers, processors, operating systems, and languages generated by every engineer or company trying to do its own thing—make no real contribution to the state of the art and usually wind up confusing the marketplace. In addition, many conflicting government and professional organizations that reflect multiple, overlapping disciplines are involved in standards setting.

Perhaps the greatest problem in standards today is that too many groups are competing to set too few standards. We must have a way of selecting the best organizations and people to work on a standard, and to reduce the number of bodies that delay work and introduce noise. While a standards process does have to recognize and respond to great ideas, these can generally come from outside the "official" body.

6. Remember that almost any standard is far more important than a highly refined optimum. To make progress, we often have to regress. Models such as the seven-layer open systems interconnection are often valuable as a base for building new standards. Unless a model is followed by the detailed definition of the layers, it is only a template for creating a multitude of vanity standards and for writing advertising copy. Since implementation must come before standardization, the seven layers might better be only four, or even nine. Unfortunately, every real implementation that says it uses the seven levels uses them the way an engineer uses a metric ruler to draw on quarter-

inch quadrille graph paper. The lines serve only as references for the infinity of figures possible; about every 2.5 inches the two scales line up pretty well. Unix is an excellent example of an arbitrary standard that is below the state of the art and will require much evolution. (I describe this standard's desirability and its necessary evolution on p. 76).

7. Provide and plan for evolution; it's often the fastest way. The evolution of a real standard usually beats the ideal that never gets completed. Algol, for example, was a good idea that never got completed or backed with appropriate implementation. Instead, it became a model for designers of succeeding languages to build on.

With exponential change in virtually every dimension of computing, the domain of a standard should be specified a priori to understand when it should be extended. Many standards, such as Fortran, survive longer than the sponsor thought or intended. One reason is that standards organizations can't arbitrarily pronounce a standard dead and ignore it if many people are using it. Otherwise, new products would make ad hoc extensions and no one would be responsible.

8. Base the standard on experience, not on a committee design; if you haven't lived with a proposed standard, don't adopt it. The best way to ensure reality is to implement several alternative interfaces before setting a given standard. The digital communications standard ISDN is an excellent example of a complex committee design, with no real test use, for what should have been a simple, clean interface. No wonder it took a decade to design. And even if it is to be widely used in the next 10 years, it will need a great deal of expensive redesign.

Another example is the Ethernet standard (802.3), which took almost 10 years after a full-scale working model to develop.[7] The upgrade over Xerox's first Ethernet improved performance by almost a factor of four, but if the original had been used to get real experience, all LANs could have been realized earlier and not still be "several years away."[5]

9. Make the standard precise, understandable, applicable, and useful at several levels of detail. You don't always have to be IBM to set a standard. In 1969, neither IBM nor any official group was interested in a standard for interconnecting computer components to form minicomputers. DEC's PDP-11 Unibus originally set the standard. Eight years after hundreds of engineers had designed hardware to attach to Unibuses, a really complete Unibus specification was finally written. The original specification provided a way to interconnect different kinds of parts, not just a pair, and showed the way for this generation of buses and the future generation of micros.

Had a standards committee been involved in the original Unibus, it is doubtful whether the design would have been completed.

10. Remember that only one or a few standards are needed for the same function; a standard should aim toward unifying a set of alternatives. Ideally, a standard should define the interface between sets of parts, not just two parts. Having too many standards is like having no standards at all. The current plethora of 802 LAN standards, including digital communication switching, which is also a LAN, is good example of too many, with no basis of experimentation. Every company, consortium, or committee tries to get one more standard bus and LAN. In turn, members add features to ride every new bus and LAN specification. There are simply too many buses, LANs, and riders. We need bus and LAN birth control.

Critical standards for the next generation

Silicon chip and wafer for custom systems: The silicon foundry. The silicon wafer is an important level of integration that requires wide-scale standardization. Since semiconductor processes have traditionally been the corporate jewels of semiconductor companies, the wafer and chip are not well publicized or documented. Yet, we can safely predict that the silicon wafer or custom chip is likely to be the basis of the next computer generation. Some computer systems will be a single chip with one to 10 million transistors. Of course, most chips will continue to come from manufacturers as a "standard" or combinations of "standards" such as microcomputers, peripherals, and memories.

Creative new products will come from the silicon foundry industry that Carver Mead advocates[8]—requiring substantial standardization. Weitek[9] is an example of this new kind of company that takes algorithms and embeds them in silicon—"VLSIzation." Another example is the workstation product, Iris, from Silicon Graphics.[10] Iris uses a dozen 75,000-transistor chips, which Jim Clark calls the Geometry Engine, and computes at a speed of 10 Mflops—roughly equivalent to a CDC 7600. In this way, Iris outperforms by a factor of several hundred the other 150 workstations. We can envision radically new, special chip-based systems that operate on pictures, voice, and mechanisms.

Because foundries and CAD systems lack standards, we are far from being able to realize the scenario of a silicon foundry industry. Standards are essential for all user-specific gate array, standard cell, or fully custom chips. It's distressing that we still have no standards for specifying gate arrays; custom PLAs and ROMs took too long to standardize. A few interfaces for this industrial structure are

- specifications of structure and behavior, including simulation and timing at all levels;
- physical information at all levels including those for processing wafer masks (CIF);
- control of foundry processes, especially if processing steps become optional;
- chip test, including automatic generation of test data; and
- chip assembly and packaging, including bonding and multichip interconnection.

For CAD, the development of standard interfaces to languages and databases that are communicable through networks must be targeted. It might be desirable to standardize the specification languages; I can't identify any

benefits of observing the differences. Agreeing on interfaces doesn't limit the competitiveness or creativity of any CAD company or foundry; it simply means that users don't have to learn many systems and languages for the same function or to convert data formats. Standards would let users mix and match different CAD systems in a completely flexible fashion. Syntactically idiosyncratic editors, timing verifiers, simulators, design rule checkers, etc., don't really increase user power. Use would expand much more rapidly because buyers wouldn't be forced to make critical long-term decisions with no way to exchange data to other systems. As an analogy, look at the pre-Cobol/pre-Fortran era in the late 50's when all the users rebelled at every manufacturer providing a unique language. The rebels designed Cobol, the first, oldest, and most widely used of the standard languages, because there was no reason for different languages.

In CAM, the user is also faced with a fuzzy and perplexing interface to the process from masks to tested components.

The foundries, CAD companies, and users, such as the Microelectronics and Computer Corporation, could effect change now so we can have the next generation. A whole Compcon could be devoted to describing alternatives and defining interfaces. I'll get back to this idea later.

Standard chip: micros, microperipherals, memories. The semicomputer manufacturers are responsible for standards resulting from the instruction-set architecture. This lowest level of integration for computers is the input to a very high gain "work amplifier" because it forces a range of unique buses, boards, and systems, including operating systems and languages, to be created.

Rebels designed Cobol, the first, oldest, and most widely used of the standard languages, because there was no reason for different languages.

Microarchitecture, like its mini- and mainframe predecessors, is the root of most of our redundant work. A micro's life is incredibly predictable, following a time-worn path with respect to its ability to access memory. Frailey[11] suggested that there are about 20 measures of word length. I believe only one counts—the amount of directly addressable memory to a process—because it determines a computer's programmability and therefore its longevity. Of course, when considering performance, a few embellishments like data types and implementation word lengths are worth noting.

Unlike participants in the semiconductor evolution, micro users are dragged along as an architecture evolves and can relive history. For example, we were dragged through the evolution of the stack, which started out in a Datapoint terminal, went on to become the 8008, the 8080, the Z80 (when another company provided us with an almost useful PC), then on to the 8086,[12] 186, 286, and so on. As a user of these parts, I have been able to relive computer evolution for a third time. The good news is that the 286 may be the fastest micro and is evidence that evolution does work because there is finally a large, state-of-the-art address. It also illustrates the difficulty of design for compatibility.

In the late 50's, a system that allowed users to treat primary and secondary memory as one was developed at the University of Manchester using Ferranti's version of the university's second machine, Mercury. By 1962, the university had an operating breadboard with a 27-bit virtual address for Atlas.[13] (Atlas also had a number of other ideas, such as Extracodes that Bhujade recently rediscovered.[14]) It was a university machine in the UK described in nearly 10 papers, and Ferranti built only a few. The critical paper was republished in 1971 in a work by Bell and Newell,[15] and again in 1982 in a work by Siewiorek, Bell, and Newell.[16] But if engineers read about it, they neither remembered or learned.

Having known Atlas, I went on to design two minicomputers with 12- and 13-bit addresses because I felt they were special and wouldn't evolve to general-purpose use. Both had to be extended to 16-bit addresses almost before they were shipped. In 1964, the PDP-6, the forerunner of DEC system 10 and the 360 were introduced, and both could access about a megabyte. The DEC system 10/20 and the 370 eventually ended up with 32 bits of address, complete with paging, just like Atlas, but about 15 years later.

In 1970, the PDP-11 came out with a 16-bit address to solve the minicomputer addressing problem.[17] The first customer demanded a physical address extension to 18 bits. The virtual and physical addresses evolved to 17 and 22 bits. For several years, DEC engineering spent thousands of hours trying to figure out how to address more memory. Users spent much time encoding programs in small memories. In 1975, the Vax project was started to provide a 32-bit address with an embedded PDP-11 for compatibility. This cycle took about eight years and was well documented.[18,19] Other East Coast minicomputers followed the same path for the second time around.

In 1971, the micro was born on the West Coast with the 4004 (12 bits of address) and 8008 (14 bits of address). Because the models were established two times, the evolution was clear. In 1978, the 8086 was extended to 20 bits and most recently to 24 bits of physical and 30 bits of virtual address. The cycle based on the 8086 has taken six to 12 years. It is ironic that information on addressing didn't travel from California to Oregon, where Intel's 432 was developed.

Motorola's saga is similar. National took the high road and copied Vax without violating its patents to supply Vax-like chips—since DEC is a minicomputer, not a semicomputer company.* Unfortunately, National didn't copy enough of Vax to make software automatically transportable from Vax.

The National architecture is certainly an interesting alternative to Vax, permitting transfer of Vax programs with minimal effort. If an exact copy of Vax could have been made, many billion dollars of software could have been made available, and many resources could have been freed for doing creative and productive work. With the micro, the cycle has been repeated three times. The saga is not yet ended, since we now understand the ramifications

*A semicomputer company is a semiconductor company that builds computers.

of address spaces greater than 32 bits. Stay tuned for further evolution.

The story surrounding the Computer Family Architecture, or CFA, specifically the Department of Defense's version of Vax, called Nebula, is far worse. An exact copy of Vax could have been made saving 10 years and many billions of dollars.

New architectures, especially those that have gone along well-traveled evolutionary paths, have cost computing at least half of its resources, have provided little or no benefit, and in some cases have delayed progress. Using C and Unix to obtain machine independence appears simple but don't be fooled. A compiler for C or a compiler written in C is only a starting point for a product . . . not the end. An architecture prevades virtually every part of a system and its database. Even if C and Unix can be standardized to a greater degree, the instruction set is still pervasive. Determining when an architecture should be copied, evolved, thrown out, or completely redesigned is fundamental to the notion of standards because of the tremendous user program and data investments.

Revolutionary research architectures based on parallelism are another issue. A recent taxonomy listed 55 new, evolutionary, and radical computer system designs: 25 can be built, 15 may be built, perhaps 10 are worthwhile building, and we have resources to build and evaluate five at most. At a time when meaningful research requires large team efforts, only a few experiments can be performed. We need the results of a few critical experiments, not more half-done, toy projects.

Board: buses for various performance, applications, etc. The board level is similar to the instruction set architecture story, except that buses last longer. The various species of the IBM channel buses are now 20 years old and will survive in their current forms for another 20 years, even though many functions that a peripheral might perform could be handled with the same amount of hardware as that required to interface and drive the bus.

The IEEE sanctions these buses. The politics is hard to understand. Is a bus designed and sanctioned independently of whether there are any riders? How many more buses do we need or can we afford beyond the existing ones? Why aren't the on-board signals standardized to mix and match processor and peripheral chips?

LANs and LANCs: another kind of switch. While riding buses, let's look at our most critical bus, the Local Area Network used for interconnecting computers and terminals in a local area. We need to keep the number of standards to a minimum so that we can get on with building clusters—or LANCs—which few organizations understand experientially. The motivation behind a LANC is the certain evolution of three types of clusters:

- a single, shared mini or large computer that will gradually be decomposed into functional server components;
- a collection of large computers that must behave as a single system with a common database; and
- the proliferation of PCs that require intercommunication to form a single integrated system by aggregation.

LANs are especially difficult to design and standardize because they cross from the computer industry into traditional communications and cable television industries involving more disciplines and organizations. A sorry parallel can be seen in the slow formation of videodisc standards—slow because both computer and television engineers are involved.

The IEEE 802 standards program is essential to LANs.[5,6] While it must be strengthened to include PABXs, a new set of numbers will be required just for all the new LANs. 802.3 was allocated for the CSMA/CD bus, Ethernet. But instead of spending energy to reduce the cost of this LAN, groups took the basic idea and built incompatible, lower cost, nonstandard versions. With IBM's recent announcement of yet another LAN, one for PCs based on CSMA/CA, we have one more standard and number.

802.3 can be transmitted on standard orange or yellow Ethernet cable. For those who like a simpler installation and lower cost and will give up distance, RGU 58 can be used—call it "Cheapernet." 3COM and Bob Metcalfe, Ethernet's inventor, call it "Thin Ethernet." Codenol has a fiber optic transmission system using the same basic electronics. For those who like cable television technology, a modem permits the same controller to transfer Ethernet's baseband information on broadband. The purpose of all these media is to build and use LANs and not to wait for what is really quite an arbitrary choice of media that only delays use. The controller/transceiver interconnect for Ethernet is becoming an important standard. With it, maybe any topology (including high-speed PABXs), modulation scheme, and medium can be used.

We need to keep the number of LAN standards to a minimum so that we can build clusters, or LANCs—something few organizations have practical experience with.

802.4 denotes LANs carried on broadband with three incompatible data-rate versions. With the 5M-bit-per-single-channel-pair version in operation, we can hope that the other two will not materialize. General Motors is using its market power to insist that various computers demonstrate their ability to communicate at NCC 84. If this works, maybe GM should take over some of the standards role from IEEE, ANSI, ECMA, and NBS.

The ring came out of early work at Bell Labs and Cambridge University. Cambridge and its alumni invent about one new ring each year. Prime uses a ring, and since Apollo was founded largely by Prime alumni, they too use *another* ring. Perhaps because rings can be built with large central controllers, IBM grabbed the ring, hence 802.5.

The 802.6 deals with metropolitan areas. Very high speed PABXs could provide the same function as the LAN and hence should come under the 802 purview. It is imperative to have conformity at the higher levels. Is this 802.6 or do we need yet another standard? Since we can obviously use fiber optics to build LANs, we require a fiber standard, 802.8. Since several already exist, can't we

either make one official or embed the medium as an option within another standard?

The multiplicity of standards to switch information at modern computer data rates causes us to avoid the essential problem of building networks and clusters. Almost every week a new incompatible LAN is announced. This is crazy!

The glib answer to the panopoly or lack of standards is gateways.[20] Virtually nothing is known about gateways except that building them is a craft—and roughly equivalent to conversions between high-level languages such as from Fortran to Pascal. Building a gateway is about as easy as designing a train that can travel on different gauge tracks. It may be fine if you can reach steady state, but the transition from track to track is tricky.

Electromechanical assembly: disks, I/O, power, enclosures. The evolution of small disks and tapes has been impressive and demonstrates the strongest case for standards. When Al Shugart started Seagate, his greatest concern was making sure a competitive second-source industry with a common interface and form factor was available. He used the same formula to create the original 5¼-inch floppy disk from factors, standards, and industries. We might increase our understanding of the standardization process by studying this industry.

Operating systems and Unix. In 1966, a user could have a 300-baud Teletype using a phone line. By 1980, the speed had been raised to 1200 baud for a performance improvement of less than 10 percent per year, while the connection cost rose. This performance is roughly equal to the improvement in horsepower and cost increases for sports cars, not for computers. By adopting Unix, a large part of our future systems development has been entrusted to some part of AT&T (call it Unixco). Given the simplicity of Unix, and the need for much more rapid evolution, either a strong Unix-compatible company will emerge, or IBM will take the responsibility for Unix. In other words, it is unlikely that Unixco will fulfill its role.

The Unix phenomenon illustrates rule 6: Almost any standard is far more important than the unobtainable ideal. Like many systems, the people who love Unix are its many parents and those who grew up with it. The final clause of rule 6 also typifies Unix: To make progress, we often have to regress. For example, look at the way Unix evolved. Its development was the result of Thompson and Ritchie's reaction to Multics, a very large, joint MIT and Bell Labs project conducted in the late 60's. The idea for Unix coincided with the time the book, *Small is Beautiful,* was popular. Thompson and Ritchie began with a discarded DEC PDP-9 and went on to use the PDP-11 in the early 70's. Since DEC didn't give away operating systems to universities, the university used Unix, which was essentially free. No manufacturer provided source code to users. Unix did. Unix, by most measures a very simple operating system, could do useful work with database access, special communications, and extra programs. Students and faculty could understand all facets of its internals and use because of its simplicity and availability. It was written in a very elegant, structured, high-level assembly language, C, and as such could be modified. It was an excellent pedagogical tool. Universities embraced it and trained many students with it providing a large, future market.

Unix was used on other computers because it was portable. As long as a C language compiler was available, a team of people could move Unix to another computer system. Other early high-level languages were never quite entirely portable because of incompatible extensions to access the operating and file system. Unix created the notion that someday we would have a complete system that was machine and manufacturer independent. Users liked this idea.

Chip makers with very small programming groups needed software and were used to adhering to standards. Small system manufacturers wanted system software and access to the DEC user base. IBM appeared to view Unix as a way into DEC's technical market. Thus, a standard was created that has almost everyone's support.

Much work is required to have a system that supports computing concepts in the 80's. Unixco must take the responsibility commensurate with its marketing. The notion of a standard is great, but is must be developed more rapidly than any single manufacturer is capable of doing. The standard can evolve if there is parallel development among many organizations. If Unixco is the only company doing and blessing all the extensions, we have simply substituted multiple competitive companies with a single, behemoth. Unix has to be evolved in a reasonable, not ad hoc, fashion. This potential bottleneck may be the most serious problem we have in extending computing today.

Some of the extensions of most concern are

- higher reliability, greater performance, and greater security;
- virtual memory (Berkeley, version 4.1 with virtual memory has been available nearly five years);
- special functions for real-time and transaction processing (Unix is being extended and adapted in incompatible ways by diverse companies. A clearinghouse to ensure portability and compatibility of applications is required);
- a modern human interface that is competitive with the PC or new PCs (Unix was developed in the time-sharing era using "glass Teletypes." As a result, interaction is via one-dimensional, cryptic messages. Helpful, less cryptic interaction and multiple windows with fast interaction are critical today.);
- multiprocessing (with the micro, multiprocessors are feasible, desirable and available);
- networks (given Unix's origin, we should demand modern communications capabilities for wide-area networks); and
- fully distributed processing across a LAN to form LANCs (The Universities of Newcastle and Berkeley and others have implemented incompatible systems for fully distributed processing. Berkeley 4.2 is a good starting point).

Languages including extensions to applications languages. The concern for Unix is paralleled by C, the heart of applications portability, and must be standardized. A language for artificial intelligence is of great concern

because of next generation applications. Lisp has proven useful for these applications.

Lisp, designed about 1960 by John McCarthy, impressed me so much that I included the critical data access primitives in the architecture of the DEC system 10 in 1965 (still about the fastest Lisp computer). Lisp branched. One path went west via Bolt, Beranek and Newman alumni to Xerox, creating Interlisp and its dialects. Many dialects evolved from the original MIT Lisp: Maclisp, Zetalisp, NIL, Scheme, T-Lisp, Portable Standard Lisp, and Common Lisp. The last two vie for standards status. Franz Lisp, Glisp, Nist are other dialects and extensions. Virtually everyone who works with a Lisp compiler or interpreter creates a private language or extension. Because these languages are incompatible with one another, we can't benchmark or extend the language in a compatible fashion using bootstrapping. Much work surrounding Lisp is to make applications development easier, but given the number of dialects and extensions to ease development, is anyone even working on applications?

To get on with the business of applying AI, we need some way of sharing information across the various different languages called Lisp. A serious standards activity is long overdue.

In fact, the Japanese were so confused about Lisp that they totally gave up and went on to Prolog.

A Compcon on standards

Unfortunately, we can't go off and simply make rules for standards; instead a better understanding of the whole standards process is needed. I think we need a Compcon devoted entirely to standards that would

- examine and prioritize critical standards [Certain standards such as LANs and electronic mail are relatively arbitrary and simply need to be frozen. On the other hand, some care is needed to avoid constraining future creativity.];
- establish responsibility and territoriality [Often too many groups are involved in setting goals and constraints, definition, review, test, and implementation. Having fewer designers, but extremely competent ones, always yields a far better system.];
- establish goals and constraints [In many cases, efforts immediately digress to bit encoding without agreement that a standard is necessary.];
- understand the timing in the past (origin), present, and future of a standard [With the invention of new phenomena, it is pointless to discuss standardization until a breadboard has been made to demonstrate utility. On the other hand, several organizations have extended Fortran in incompatible ways to handle vectors because the standards group has considered Fortran a dead language; a standard is still long overdue.];
- understand the effects and desirability of standards [Although most effects appear to be beneficial to both suppliers and consumers, the perception of almost every producer is that the "ideal state" is a monopoly.]; and
- develop arbitrary "standards" for use in future radical research to aid rapid progress [For example, every data-flow computer has a unique, higher level language. Almost any data-flow language would let us encode algorithms and measure parallelism without having to build any special hardware.].

Models for the next generation of computers will continue to rely heavily on standards. The traditional levels of integration are now well defined through various industry and traditional standards. Following certain rules might improve the standards process, and a Compcon devoted to particular standards and the standards process would help us develop standards more intelligently.

Postscript

I've extolled standards now for some time, but there is a down side. A standard provides an interface, or target, by which similar systems can be compared—and anyone can use it. For example, the focus and adoption of US standards has permitted Japan to become number one in computing.

In early 1984, kernel benchmark codes developed at the Lawrence Livermore National Laboratory were run in Japan on the Fujitsu VT100 and VT200 and the Hitachi 810/820 at a rate of over two times what a one-processor Cray XMP could achieve. The Japanese machines are evolved versions of a 20-year-old architecture, the IBM 370, implemented with evolved 25-year-old ECL circuitry, highly evolved 25-year-old ICs, and expressed in 25-year-old Fortran. The Japanese used standards to increase productivity; they did not start by inventing a new architecture and the associated reprogramming. They built on the vectorizing compilers derived from the 15-year-old Illinois Illiac IV project.

This example illustrates the value of using standard interfaces, understanding the old, and seeing that it envolves immediately to increase output and free resources with higher productivity. It lays down the gauntlet for a new revolution. *

Acknowledgments

I dedicate this article to all engineers who devote their energy to working on standards in an effort to create a more orderly environment.

References

1. M. A. Harris, "Computer Makers Bet on Standards," *Electronics*, Vol. 57, No. 3, Feb. 9, 1984, pp. 110-111.
2. P. Lemmons, "The Standards Craze," *Byte*, Vol. 9, No. 1, Jan. 1984, p. 5.
3. P. Lemmons, "An Interview: The Macintosh Design Team," *Byte*, Vol. 9, No. 2, Feb. 1984, pp. 58-80.
4. G. A. Brooks and F. P. Brooks, "The Structure of the System/360," *IBM Systems J.*, Vol. 3, No. 2, Feb. 1964, pp. 119-135 (reprinted in Chapter 43 of *Computer Structures*, see ref. 15).

5. J. Nelson, "802: A Progress Report," *Datamation*, Vol. 29, No. 8, Aug. 1983, pp. 136-152.
6. R. P. Blanc, "Local Area Network Standards," *Proc. Compcon Spring 84*, IEEE-CS Press, Los Alamitos, Calif., pp. 252-254.
7. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," *Comm. ACM*, Vol. 19, No. 7, July 1976, pp. 395-404 (reprinted in Chapter 26 of *Computer Structures: Principles and Examples*, see ref. 16).
8. Carver Mead, lectures and private communication.
9. *High-Speed IEEE Floating Point Processors*, Weitek Corp., Santa Clara, Calif., 1984.
10. *IRIS Product Reference Manual*, Silicon Graphics, Mountain View, Calif., 1983.
11. D. J. Frailey, "Word Length of a Computer Architecture: Definitions and Applications," *Computer Architecture News*, Vol. 11, No. 2, June 1983, pp. 20-26.
12. S. P. Morse et al., "Intel Microprocessors—8008 to 8086," *Computer*, Vol. 13, No. 10, Oct. 1980, pp. 42-60.
13. T. Kilburn et al., "One-level Storage System," Institute Radio Engineers Trans. Electronic Computers, EC-11, Vol. 2, Apr. 1962, pp. 223-235.
14. M. R. Bhujade, "On the Design of Always-Compatible Instruction Set Architecture (ACISA)," *Computer Architecture News*, Vol. 11, No. 5, Dec. 1983, pp. 28-30.
15. C. G. Bell and A. Newell, *Computer Structures: Readings and Examples*, McGraw-Hill, New York, 1971.
16. D. P. Siewiorek, C. G. Bell, and A. Newell, *Computer Structures: Principles and Examples*, McGraw-Hill, New York, 1982.
17. C. G. Bell et al., "A New Architecture for Minicomputers—The DEC PDP-11," *Conf. Proc. AFIPS SJCC*, Vol. 36, 1970, pp. 657-675 (reprinted in Chapter 9 of *A DEC View of Hardware Systems Design*, see ref. 19).
18. C. G. Bell and W. D. Strecker, "Computer Structures: What Have We Learned from the PDP-11?" *Proc Conf. Third Annual Symp. Comp. Arch.*, IEEE and ACM, 1976.
19. C. G. Bell, J. C. Mudge, and J. E. McNamara, *Computer Engineering: A DEC View of Hardware Systems Design*, Digital Press, Bedford, Mass., 1978.
20. W. Stallings, "Beyond Local Networks," *Datamation*, Vol. 29, No. 8, Aug. 1983, pp. 167-176.

C. Gordon Bell is chief technical officer for Encore Computer Corporation, where he is responsible for the overall product strategy. Before joining Encore Computer, he was vice president of engineering for Digital Equipment Corporation, responsible for R&D activities in computer hardware, software, and systems.

Bell was also manager of computer design at DEC, responsible for the PDP-4, -5, and -6 computers. He was also on the faculty of Carnegie-Mellon University from 1966 to 1972.

Bell's accomplishments are many. He led the team that conceived the Vax architecture, established Digital Computing Architecture, and was one of the principal architects of C.mmp (16 processors) and Cm* (50 processors) at Carnegie-Mellon University. He is a widely published author on computer architecture, and computer design.

Bell earned his BS and MS degrees in Electrical Engineering at Massachusetts Institute of Technology in 1956 and 1957 respectively. He holds several patents in computer and logical design.

Among Bell's professional affiliations are the National Academy of Engineering, the IEEE (fellow), the American Association for the Advancement of Science (fellow), and the ACM. He is listed in *American Men of Science* and *Who's Who*.