Technology and Business: Forces Driving Microprocessor Evolution

NICK TREDENNICK

Invited Paper

High-end microprocessors are the epitome of integrated circuit technology. But it is low end microprocessors which fuel progress in microprocessors. More than 98% of the four billion microprocessors forecast to ship worldwide in 1995 will end up in low end embedded applications. The microprocessor industry spends most of its research dollars developing high-end microprocessors for CPU applications. Current high-end microprocessor implementations exploit most of the inherent parallelism in their instruction streams. Where does microprocessor development go from here? I give an overview of the development of the microprocessor industry and I show that to predict where the microprocessor is going one must first understand the industry's illogical behavior.

I. Introduction

Any engineer can predict the future by assuming it's a logical extrapolation of the present. I'm going to show you how to predict the future as an *illogical* extrapolation of the present. A fresh look at the history of the microprocessor and at the corresponding development of semiconductor process technology can give us tools to see the future. As I look at the events that shaped the history of the microprocessor, I see that an outsider would have predicted vastly different results. The outsider tends to focus on events and on the progress of relevant technologies, but the truth is more complex. The social environment and the personalities of key participants significantly influence the course of events.

First, I pose why the computer itself has been successful. Then, I emphasize the development of microprocessor applications and the development of the market rather than emphasizing the history of the microprocessor itself. I show how these things shaped the development of the microprocessor. I attempt to identify conceptual breakthroughs and their effects. With this foundation, I launch into speculation about where microprocessor development may be headed and why.

Manuscript received April 3, 1995; revised August 15, 1995.
The author is with Tredennick, Inc., Los Gatos, CA 95030 USA.
IEEE Log Number 9415185.

II. Invention of the Computer

The computer was invented to solve a single problem: the calculation of artillery firing tables for the US Ordnance Department. Predictions that the world's need could be satisfied by fewer than a hundred computers have proved spectacularly wrong. Today's installed base of computers is millions of times the original predictions. It would be difficult to do worse on a prediction. *The miss was caused by a focus on what the computer did (fast numerical computation) rather than on what it was (a breakthrough method for solving problems).*

Prior to the invention of the computer, problem-solving engines solved their problems directly. The true breakthrough for the computer was the idea of solving a problem indirectly. The computer is able to solve a greater range of problems than a direct hardware implementation because expensive computing resources can be amortized over a range of problems. A collection of problems can share an expensive, problem-solving resource the cost of which no single problem could justify.

Further, the computer can iterate to solve a single large problem, amortizing the cost of solving the problem across a much smaller set of resources than hardware for a direct solution would employ. The computer can solve large problems which are too expensive to solve with a direct hardware implementation. Of course, large problems might take a long time to solve.

The computer solves problems more economically than the direct hardware solution because it divorces the problem solving algorithm from the physical resources used to solve the problem. The algorithm (manifested as a program), stored on cheap, plentiful resource (paper tape, punched cards, magnetic tape, magnetic disk), manipulates the scarce, expensive hardware resources.

The idea of a memory hierarchy is an ironic analogy to the value of the computer in solving problems. (I say ironic because the invention of cache memory followed and depended on the invention of the computer, but it better illustrates the breakthrough concept of the computer.)

Cache memory, a small, fast memory which sits between the CPU and a larger, slower memory, makes the total memory serving the CPU seem faster than it actually is by exploiting temporal and spatial locality properties of problem solving methods. For the computer, mapping an algorithm into memory and having it manipulate expensive computing resources exploits the same properties (temporal and spatial locality) to make the expensive computing resources seem more plentiful than they are. A direct implementation fixes both the computing resources and the algorithm in hardware. A computer implementation fixes the computing resources and allows the algorithm to vary.

III. The Progress of Logic Technology

The first computers were built of relays, vacuum tubes, and other discrete components. Invention of the transistor in 1947 precipitated a change to solid state technology [1]. Miniaturization followed as the transistor displaced relays and vacuum tubes in computers. Invention of the integrated circuit (IC) precipitated another change as integrated logic displaced collections of discrete components in digital systems.

Integrated circuit families such as TTL (transistor-transistor logic) provided design engineers with macro design functions. These logic families, and the TTL 54/74xx series in particular, thrived because they raised the efficiency of the design engineer. The engineer designed with circuit macros (e.g., 4-b adders, registers, and NAND gates) rather than with individual transistors. The resulting designs were smaller, cheaper, and more reliable. As the designer's efficiency improved, the rate of computer implementations accelerated.

From its invention in about 1958, integrated circuit technology has been improving rapidly. This improvement rate is known as Moore's Law [2]. Three major factors and a host of minor factors contribute to the rate. These major factors contribute to improvements in the number of transistors on a chip:

- First, the number of transistors on a chip increases in direct proportion to the area of the chip. Double the size of the chip and you double the number of transistors it can accommodate, since the transistors and the chips are essentially both planar (i.e., two-dimensional structures). Restricting ourselves to commercial microprocessor development for our examples, Intel's 4004, introduced in 1971, was 10.65 square mm [1]. Digital introduced the 21164 (Alpha) in 1994 at 298 square mm [3]. Just these two points show chip area growing at the rate of 16% per year, which means chip transistor capacity would double about every five years.
- Second, we form transistors by manipulating the crystal lattice structure of silicon (selectively creating regions with excess electrons or excess holes in an otherwise intrinsic region). We create these regions by drawing patterns on the silicon and then etching and doping exposed areas. Since the operation of a solid-state transistor is based on the behavior of electrons and holes in the crystal structure of an atomic or molecular lattice, transistors are potentially small. But sizes of practical working transistors have been limited by our ability to draw the finer and finer lines needed to make the transistor-defining patterns. In trying to draw atom-sized structures with a crayon, we are limited by the crayon. The number of transistors on a chip increases as the square of the decrease in line width. Decrease the width of the lines by half and the chip holds four times as many transistors. Line widths to draw transistors on Intel's 8086, which was introduced in 1978, were 4.5 μm. Line widths for Digital's 21164 (introduced in 1994) were 0.5 μm. Again extrapolating from only these two points shows line width shrinking about 13% per year, which would mean they would shrink by half about every five years. Since transistors-per-die increases as the square of this decrease, capacity would double about every 2.5 years.
- Third, circuit design techniques improve. As we build more circuits, we learn more about how to build them efficiently.

Intel's 4004, introduced in 1971, contained 2300 transistors. Intel's P6, introduced in 1995, contains 4.5 M transistors. Extrapolating from just these two points, shows microprocessors improving the number of transistors per die by 37% per year. Transistors per die would be doubling about every 2.2 years. (I put the lines for the last three examples through two points each to illustrate the general rate of progress, not to compute a statistically definitive rate. For my purpose, it isn't important whether the true rate shows transistors per die doubling every 21 months or doubling every 26 months.)

Chips are built in batches by drawing the circuits on a circular wafer. The number of dice (chips are called dice) the wafer can accommodate is determined by the size (and aspect ratio) of the chip and the diameter of the wafer. During the time the process of drawing transistors on a chip has dramatically decreased the area required for a transistor, the diameter of a manufacturable wafer has increased from 50 mm to 200 mm (and there's even talk of moving to 300 mm wafers). The number of chips a wafer can accommodate goes up approximately as the square of the radius. For a small chip, the 200 mm wafer accommodates 16 times as many chips as a 50 mm wafer, but the cost to process a 200 mm wafer is probably only a few times the cost to process a 50 mm wafer. (Think of processing wafers as an elaborate baking process—it doesn't cost much more to bake a larger cookie. In fact, wafers are processed in batches just as cookies are.)

Decreasing line widths and increasing wafer sizes combined to drive the integrated circuit in two directions: 1) cost to implement a fixed function decreased rapidly and 2) function implemented for a fixed cost increased rapidly. Decreasing cost to implement a fixed function fueled growth in the market for macro functions (e.g., TTL). Increasing function for a fixed cost drove a proliferation in circuit variety, so the range and complexity of available macro functions grew.

But there's a problem with growth in the complexity of macro functions: as the complexity of a macro function grows, its range of applications decreases, so production volume decreases. You can build anything from simple NAND gates, but an ALU is used only as an ALU. Further, as the macro functions become more complex, it becomes more difficult to understand them and to use them appropriately. As macro specialization and variety increase, the cost to produce and to stock a wider variety of components increases. The industry was headed for crisis. Should manufacturers continue to produce more complex, lower volume macrofunctions or should the industry convert to application-specific integrated circuits (ASIC's)? The answer turned out to be neither.

IV. Development of the Microprocessor

Introduction of the first commercially available microprocessor in 1971 headed off the battle between general macro functions and custom chips. The microprocessor offered a solution with a few standard parts to cover a broad range of applications. Most applications are cost-sensitive and are not performance sensitive, so the microprocessor offered an excellent alternative to either a macro-based solution, which would use more chips and be less reliable, or a custom chip solution, which would be more expensive and involve substantially more design risk. In these "embedded control" applications, the microprocessor displaces either custom logic or macro functions as the state sequencer. Manufacturers could produce and stock a few standard components, halting the counterproductive proliferation of complex macro functions.

Starting from almost nothing in 1971, the worldwide market for embedded control microprocessors has grown to more than four billion units in 1995 [4]. The success of the microprocessor has been propelled by advances in the underlying silicon process technology—which reduces size and cost and which improves performance and capability. By unit volume measures at least, the microprocessor has always been an embedded control device.

The breakthrough concept in the application of microprocessors for embedded control was the same concept that propelled the computer to success. The problem is mapped across two kinds of resources: scarce, expensive hardware and cheap, plentiful memory (relative to the microprocessor). Implementation cost is closer to that of the cheap resource and performance is closer to that of the expensive resource because the mapping exploits properties of the method for solving the problem (temporal and spatial locality again). The algorithm, sitting in the memory, enables the memory to keep the expensive microprocessor busy.

Following the introduction of the first commercial microprocessor, continuing improvements in the IC process and in the shrinking computer eventually led to the first microprocessor-based personal computer. The first ad for a commercially available microprocessor-based computer appeared in March 1974 [5]. These hobby computers appealed to nerds, but the concept didn't set the world on fire.

Meanwhile, microprocessors were selling well for embedded applications. In embedded applications, the microprocessor is a logic replacement device. To displace TTL in embedded applications, the microprocessor-based solution had to be cheaper and it had to deliver adequate performance. Since the entire market seemed to be embedded control (manufacturers could safely ignore the hobby computer market with its insignificant volume), microprocessor implementations emphasized low cost. Low cost implementation places a premium on chip design optimized for minimum area and on leisurely bus protocols able to tolerate a wide variety of memory and peripheral chips on a single narrow, shared bus. Smaller packages with fewer pins are cheaper, more reliable, and use less board space. In tradeoffs between cost and performance, manufacturers opted for lower cost to serve the cost-sensitive embedded control market.

V. Microprocessor Generations

First generation microprocessors, such as Intel's 4500 transistor 8080, which was introduced in 1974, processed instructions serially [1]. The state sequencer fetched an instruction, then it decoded the instruction, and finally it executed the instruction. Then the process repeated. Each instruction was processed independently. For these first microprocessors, transistors were large and expensive and chips were small.

Second generation microprocessors, such as Motorola's 68 000 transistor MC68000, which was introduced in 1979, pipelined instruction processing [6]. The MC68000 overlapped fetch, decode, and execute. While the first instruction was being executed, the second instruction was being decoded, and a third instruction was being fetched. This kept the fetch, decode, and execution units busy unless the instruction in execution needed access to the only path to memory (to read or write an operand, for example). Pipelined instruction processing is, of course, faster than serial processing, but still retains the advantage that there is only one instruction in execution at a time, so interrupt and exception handling remains relatively simple. These processors had a maximum instruction execution rate equal to instruction access time for simple instructions. Any instruction requiring additional memory accesses or more than simple computation (like multiply and divide) took longer. For second generation microprocessors, transistors were still expensive in area and in design cost.

Third generation microprocessors, such as Motorola's MC68020, with more than 240 000 transistors, extended the pipeline to five stages or more and added on-chip caches [7].

Fourth generation microprocessors, such as Intel's 80960CA and Motorola's MC88110, implemented superscalar instruction processing. These microprocessors issued more than one instruction at each clock cycle. Microprocessor designs passed a million transistors.

Fifth generation microprocessors not only issue more than one instruction per cycle, they also allow instructions

to execute out of order and may have a large number of instructions (e.g., 30) outstanding in various stages of completion. These designs, such as AMD's K5 and Intel's P6, range above three million transistors on a chip.

VI. Where Do We Go From Here?

Where do we go from here in microprocessor design? At least three approaches have been suggested for sixth-generation microprocessor implementations: superscalar, multiprocessor, and very long instruction word (VLIW).

The most direct approach for improving microprocessor performance is to exploit available headroom in superscalar design. One might, for example, increase the sizes of buffers and queues, increase cache size, and improve the sophistication of the branch prediction algorithms [8]. This would still be a fifth-generation design. But the price of moving from an implementation which can issue four instructions per cycle to an implementation which can issue eight instructions per cycle may be high and the return may be relatively small. The design might more than double in complexity and return only a 5–10% performance increase.

Another approach suggests multiprocessor implementations [9]–[11]. Operating systems and applications can be multithreaded. That is, they can be rewritten to enable concurrent processing of independent instruction streams (threads). Multiple processors, which may all be on a single chip, can each be given an independent thread to process, thereby improving performance [12], [13]. This approach, however, doesn't benefit existing object code.

A third approach suggests using a VLIW implementation with a companion compiler scheduling instructions into available slots [14]–[16]. Superscalar implementations ingest groups of sequential instructions and schedule them in real time using on-chip hardware. This dynamic scheduling is limited to the window of potential outstanding instructions (which is perhaps thirty instructions in a fifth-generation design). The compiler supporting a VLIW implementation is able to schedule instructions from a much wider window in the program (potentially the entire program). But the scheduling is static, i.e., based on simulated execution [19], and the compiler and the implementation are tied together, which may be a disadvantage for existing object code and may require a new compiler with each new implementation.

Each of these approaches has major problems. We either maintain object-code compatibility and get a small performance gain or we give up object-code compatibility for a larger performance gain. And, these approaches are all direct attacks on the problem. Instruction execution is the bottleneck. One attribute common to all these approaches is that they all have the objective of executing more instructions in a given period of time. How about a new approach? If instruction execution is the problem, let's stop executing instructions [18]. This approach may not be as daft as it seems [19].

When the personal computer was introduced, applications and the operating system had equal access to the hardware. As the market has matured, the operating system has increasingly interposed itself between the hardware and the applications. Applications no longer access system resources directly, instead, they request services from the operating system. This application programming interface (API) is more efficient for application developers since they no longer have to write low-level service routines themselves. Services provided to application developers through the API are the equivalent of the TTL macrofunctions, so they raise the efficiency of the application developer. Also, applications written to a single API reduce chances for incompatibilities and for interference among applications and make the interfaces to different applications more consistent for the user.

Some industry observers use the rise of standards to predict a change from x86-based systems, which dominate desktop computer use, to systems based on other architectures. After all, if a single operating system, such as Windows NT, for example, is portable and is available on several platforms, and if the applications are written in a high-level language, and are compiled to object code for several different architectures, and are distributed on a single CD-ROM, then the user is free to choose the architecture which offers the best price, the best price/performance, or the best performance. The user is no longer tied to a single instruction set architecture by the limited availability of applications or of operating systems.

The rise of standards has not caused the x86 to be abandoned, but the standards have begun to isolate applications from the underlying system resources and to standardize system services provided by the operating system. This has left an opening to subvert service calls to the operating system. Rather than let the operating system execute the instructions in the low-level service routine, an accelerator board might recognize and intercept the service call and execute the function directly in hardware. As an example, a graphics accelerator board might intercept a request to the operating system for drawing a line on the screen and just draw the line on the screen using hardware resources on the accelerator board [20]. Graphics accelerators are plentiful, capable, and rapidly getting cheaper. In October 1993, *PC World* found more than 100 graphics accelerators available for the PC and reviewed the 17 priced below $600 [21]. Less than a year later, in September 1994, *PC World* found and reviewed 33 PC-based graphics accelerator boards priced below $600 [22]. In Macintosh, there have been several generations of QuickDraw accelerator boards which intercept and execute QuickDraw commands [23], [24]. There are also accelerator boards for applications such as Adobe Photoshop [25]. These accelerators can produce substantial performance increases by avoiding low-level instruction execution.

VII. Programmable Logic Devices

Invention of the programmable logic device (PLD) by Sven Whalstrom was another conceptual breakthrough [26]. Fortunately for the PLD companies, Sven, whose funda-

mental patent (#3 473 160, *Electronically Controlled Microelectronic Cellular Logic Array*) was filed in 1966, was too early. His conceptual breakthrough came before process technology progressed to a point which would make the concept practical. About the time Sven's patent expired, transistors available on a single die rose to the point of practical application and companies such as Altera (1983) and Xilinx (1984) began to develop the market for his conceptual breakthrough.

The PLD is conceptually a two-layer device. The first layer is logic elements and an interconnect structure. The second layer is memory. Values in the memory cells establish connections between the logic elements and the interconnect structure.

If the microprocessor had not been developed, the battle between complex macrofunctions and custom chips may have been headed off instead by programmable logic. Programmable logic has the same advantages the microprocessor had over TTL designs or ASIC's. A single PLD can implement a wide variety of functions, avoiding the parts-proliferation problem, and it doesn't have the high development cost of an ASIC [27]. The microprocessor not only headed off the showdown between custom chips and complex macro functions, *it set back the development of programmable logic by at least 10 years*. If the microprocessor had not been there, programmable logic might have filled the role and the past 24 years of microprocessor development would have been spent instead on the development of programmable logic components and on the development of tools for mapping designs into them.

Arguments over programmable logic devices point to major drawbacks and to compelling advantages. Overhead is a major drawback. Programmable logic devices have more overhead than the US government. A programmable logic device built of 1 M transistors may accommodate only about 50 000 transistors worth of logic. Two hundred physical transistors per logical transistor is substantial overhead. Further, design tools are not mature (at least relative to development tools available for microprocessors), so it can be difficult to map high-level functions into a programmable device. Cost is another barrier to application. High-end programmable logic devices cost as much as a high-end CPU.

On the other side of the argument, progress in IC process technology makes overhead less important. Low-integration TTL functions and glue-logic chips, such as the 22V10, are now pad limited. This means the die size cannot decrease because chip area is determined by perimeter area needed for the connection pads (as opposed to area needed for circuit transistors). Once the die size stops decreasing, cost to manufacture stops decreasing (except when wafer size increases) and performance is difficult to improve. PLD cost and performance are coming down to meet low-integration TTL cost and performance. And a single low-end PLD can displace a variety of low-integration IC's, which leads to high volume for the PLD, which helps drive cost down faster. The PLD industry is driven forward by high-volume applications for low-end devices and by high-margin applications for high-end devices, just as the microprocessor industry has been. Many PLD's are built on processes similar to processes for memory IC's, so they benefit from investment in memory IC process. Programmable logic structures offer the opportunity for compute engines with variable resources and variable algorithms. (The microprocessor is a compute engine with fixed resources and variable algorithms. Combinational logic is a compute engine with fixed resources and a fixed algorithm.)

Future application accelerators for PC's may use SRAM-based PLD's, since the inherently parallel PLD offers significantly better performance on some problems than the microprocessor and offers more flexibility than an ASIC-based implementation [28]–[30]. SRAM-based PLD's can be reconfigured. A variety of functions can be stored on the PC's hard disk to be downloaded as required, so the PLD implementation could amortize its cost across a range of applications. For example, a single PLD-based accelerator board might be configured to accelerate Windows initially, and then configured to accelerate AutoCAD functions whenever that application starts. Eventually, reconfigurable accelerators based on programmable logic may migrate onto the microprocessor. Programmable logic allows hardware configured to suit the problem. Solutions can process data in parallel.

A camera lens processes optical data in parallel and in real time. A computer can perform the same functions as the lens, but processes the data one pixel at a time (though it may look at a large window around the pixel). Programmable logic occupies territory somewhere between the optical lens and the computer.

VIII. Development of the Desktop Computer

The progress I've just outlined in general terms for microprocessors and IC process from 1971 to the present drove corresponding improvements in computers. Computers were shrinking and continue to shrink. Shrinking computers became cheaper to implement, lowering barriers to entry and spawning the minicomputer companies. The number of companies producing computers rose as the computer shrank. Availability of computers rose sharply. Digital Equipment Corporation, which offered substantial discounts to universities, introduced low-cost, hands-on computers to the university environment, deeply influencing generations of students and professors.

In 1981, IBM introduced its personal computer [5]. IBM's personal computer, which provided credibility for the concept of a desktop computer, in combination with compelling applications (word processors and spreadsheets), launched the desktop computer industry. Since the introduction of the IBM Personal Computer (PC), the market has grown to about 60 million units in 1995 [31]. Prior to the introduction of the IBM Personal Computer, the microprocessor was viewed as an embedded control device. After the introduction of the PC, the taxonomy of microprocessors split into two branches: embedded control and CPU. In spite of the introduction and success of the

PC, microprocessor manufacturers continued to produce microprocessors for the overwhelming volume segment of the market: embedded control.

The invention and proliferation of the microprocessor further lowered barriers to participation in the computer revolution. Enthusiasts fostered additional computer conferences and a forest-leveling avalanche of papers and publications. As the computer shrunk, the number of companies offering computer products rose, increasing demand for computer architects. (Computer architects design computer instruction sets.) Enabled by the shrinking CPU, any company with access to process technology could build and market its own microprocessor. Any company worth its salt did so.

Improving availability of computers and the rate of change in the industry fueled increased enthusiasm for computer design topics among students, faculty, and the trade press. This happened at a time when Unix was rapidly rising in popularity [32]. Some operating system had to be found for the proliferating computers. Unix was an excellent candidate since it was portable, minimal, modular (one could add functions), cheap, and available as source code (i.e., ready to customize). Coupled with increased popularity of Unix was the notion of programming in high-level languages.

Prior generations of large computers had to contend with severe limitations in logic and in memory capacity, so early instruction set design focused on efficient instruction encoding. Everyone programmed in assembler, since the computer had neither the performance nor the memory to afford the inefficiencies of the additional level of abstraction introduced by the high-level language. As programmers moved to high-level languages (a move encouraged by improvements in performance and memory capacity and by the proliferation of instruction sets), computer architects attempted to follow the languages by designing instruction sets which would be good compiler targets. The idea was to reduce the "semantic gap" between the programming language and the native instruction set to make the compiler's job simple and direct [33].

Once again, progress in integrated circuits helped drive changes in the industry. Integrated circuit design became a popular topic in universities with the publication of Mead and Conway's book *Introduction to VLSI Systems* in 1980. Universities began offering courses in integrated circuit design and students began designing their own chips, which could be produced for them through the government's MOSIS program. The program allowed several different circuit designs on the same chip, amortizing the cost of wafer processing across several student projects. Students could produce real IC's from their design projects [34]. And no project carried more allure than the design of a central processing unit.

The combination of increasing enthusiasm in computer design and in IC design, the introduction of the personal computer (which popularized the microprocessor as a CPU), and the availability of a shared service (MOSIS) for producing real IC's led some universities to start microprocessor design projects (e.g., RISC at Berkeley and MIPS at Stanford). To accommodate student designs in limited time (quarter or semester increments) with novice engineers (students as designers) and a limited design environment, the processor architecture and implementation were simplified. Instruction and addressing-mode variety were kept to a minimum. Instruction formats were simplified. Simple, synchronous bus protocols were used. The students and professors were surprised and pleased to find their simulations showed better performance for their processors than for commercially available microprocessors.

Enthusiastic students and professors produced a tidal wave of publications extolling their work. Early papers on RISC (reduced instruction set computer) touted no fewer than 18 concepts accounting for reported performance improvements [35]–[37]. Among those were: reduced instruction set, fixed instruction length, shorter cycle time, overlapped register windows, large register set, simplified addressing, high-level language user interface, advanced compiler technology, delayed branch, advanced procedure calls, single-cycle execution, simplified implementation, quicker time to market, better design procedures, better design tools, on-chip cache, wider external buses, and load/store architecture. The tactic of comparing simulated performance of new designs to previous-generation microprocessors helped. But wider, faster, nonmultiplexed synchronous external buses, which increased bandwidth to memory by a factor of six to ten over the narrow, asynchronous multiplexed buses of earlier-generation commercial microprocessors, probably made the biggest contribution to reported performance improvements [38].

RISC's were the first microprocessors designed explicitly for the CPU market and their performance simulations showed designing for performance rather than for low cost had value in CPU applications. The low-cost requirements for embedded control, which had driven the design of all commercial microprocessors, were at odds with the performance requirements for the microprocessor as CPU [39].

These early RISC projects and papers started a shift in microprocessor design and marketing which continues to the present. The advent of RISC also served to split the microprocessor taxonomy once again. The CPU branch split into high-end and low-end applications, with the PC, based on a commercial microprocessor originally designed for embedded control in a terminal application, representing the low end. Now the argument changed from reducing the "semantic gap" to designing instruction sets to match the technology. The instruction set should be independent of the language representing the problem. Match the instruction set architecture to the technology at the time of implementation [40]. Recompile to new instruction sets for more performance as technology improves.

Industry pundits began predicting the ultimate triumph of Unix on the desktop. This fueled enthusiasm and investment—which fostered trade press coverage. The positive feedback in this loop spawned microprocessor design projects at companies such as AMD, Intel, MIPS, Sun, TI,

Fairchild, HP, Motorola, IBM, and AT&T. The success of microprocessors resulting from these projects depended on the rise of Unix and portable applications.

By the early 1980's, memory sizes had grown and microprocessor performance had progressed to a point that made it reasonable to run Unix on a microprocessor-based desktop computer. The workstation industry was born, and with it came new companies such as Sun and MIPS to compete with traditional computer companies such as Digital, IBM, and Hewlett-Packard. For the purposes of this paper, a workstation is any desktop or deskside computer based on a non-x86 microprocessor which runs either Windows NT or some version of Unix [41]. By 1985 the workstation market had grown to about 60 000 units a year and looked as if it was in for a long period of rapid growth. In fact, the workstation industry did grow to somewhat under 800 000 units in 1994 [42], [43].

The original workstations were based on commercially available microprocessors such as the Motorola MC68000. Then two things happened. First, as mentioned above, investigators at universities and corporate research centers noticed the mismatch between attributes of commercially available microprocessors and the requirements for a microprocessor used as the CPU in a computer application. These computer architects began promoting microprocessors specifically designed for computer applications. Second, the workstation companies became frustrated with their suppliers. As the chips became more complex, design cycles increased, which led to delivery delays. The workstation companies were not in control of the key technology (microprocessors) for implementing their products. Further, they depended on a single source for that product. This led companies like Sun and HP to change to processor technology they could control. HP chose to develop their own PA-RISC internally. Sun chose the SPARC architecture (a derivative of the original RISC II design at Berkeley), but chose a different model for producing parts. Rather than design and produce the parts internally as HP was doing, Sun chose to license the technology to separate developers who would then develop and sell parts. Sun's aim was to spawn a competitive market in SPARC designs assuming there would be several suppliers, so they would no longer have to depend on a single source for their key components. Sun signed up companies such as LSI Logic, Cypress, Fujitsu, and Texas Instruments to design and produce SPARC microprocessors.

Another workstation startup company, MIPS, derived from the MIPS project at Stanford, chose yet another strategy. Its strategy was to produce a single design and then license it to companies such as LSI Logic, IDT, Performance Semiconductor, NEC, Siemens, and Toshiba. MIPS would do the design work and licensees would produce and sell components. Sun's licensees would compete on the processor designs as well as in manufacturing, while MIPS' licensees would compete in manufacturing a single design. Advocates for the Sun model argued that Sun would have access to more competitive designs and Sun licensees were free to differentiate themselves by their designs [44]. Advocates for the MIPS strategy argued that MIPS could amortize the cost of its single design across all the producers, which would make their components cheaper, and that MIPS could, therefore, afford to collect one truly "world class" design team. Further, the MIPS licensees would still be able to modify the design to differentiate their offerings.

IX. The Battle for the Desktop

When IBM introduced the IBM Personal Computer, there was no competition, so IBM had the market to itself. But IBM lost control of the market when the IBM-compatible personal computer became an open standard. Apple introduced Macintosh in 1984 and tried to penetrate the personal computer market (Apple had once controlled the hobby market with the original Apple and Apple II). Over the next few years, Apple was able to capture something like 10% of the rapidly growing personal computer market [31]. Apple, unlike IBM, never lost control of Macintosh. Apple was the sole producer of Macintosh-compatible systems. Apple controlled the BIOS, system design, and operating system. This gave Apple much better control over compatibility and application compliance. It also allowed Apple to control margins, which was something IBM could no longer do once IBM-compatible personal computers from other manufacturers were available. Apple's advantage in controlling its margins was offset by its added burden of having to pay for all the system, BIOS, and OS development, which meant higher costs for product development. Then, in 1994, Apple switched the processor in Macintosh to a new architecture and finally began a licensing program, complicating the picture further. As Apple moves away from the MC680x0-based Macintosh, it would like the installed base to upgrade to a PowerPC-based Macintosh, but some users may elect to convert to an x86-based PC instead. Licensing may lower prices for PowerPC-based Macintosh computers, but also creates additional competition for Apple. Apple will probably lose market share [45], [46].

The first microprocessor in a commercial product was Lee Boysel's AL1, which was designed and built at Four-Phase for use in a terminal application in 1969 [47]. Development cost for the AL1 was under $50 K (verified in a phone conversation with Boysel). Since that first commercial application of a microprocessor, development cost for microprocessors has risen rapidly. It probably cost Motorola about $5 M to design the MC68000 during 1977–1979. In 1993, MIPS estimated a design cost of $100 M for their "T5" processor [48]. Intel has estimated the cost of developing the 80486, introduced in 1989, at about $100 M [49] and the cost to develop its follow-up, Pentium, introduced in 1993, at about $250 M. Microprocessor development costs are rising more than 25% per year, which means they double about every three years. And these are just the costs for developing the design. To be competitive, the design must be implemented in a leading-edge process—and it may cost as much to develop a leading-edge process as it costs to develop the design itself.

Development costs rise rapidly for several reasons. The number of transistors in a design doubles almost every two years. Microprocessor performance doubles every 18 months. Design verification difficulty grows rapidly with increased design complexity. As the stakes get higher, the designs become more complex, and design teams grow. As the design teams grow, coordination and communication overhead grow. Designs after the first one face compatibility issues, which add to design and verification complexity.

The workstation market began in the early 1980's when design cost for a microprocessor was passing through $5 M. Companies in the workstation business decided to free themselves from their single-source suppliers as microprocessor design cost was on its way to the range of $30–$100 M. Development cost is relevant because it is fixed for a given design generation. Cost to manufacture a leading-edge microprocessor today is about $65 per unit [50]. The cost to produce and to deliver a microprocessor includes the cost to manufacture the microprocessor plus the amortized cost of development. If development cost is $100 M and you sell 100 M systems containing your microprocessor, the cost of the microprocessor in the system is $66 ($65 for manufacturing and $1 for amortized development cost). If you sell only 100 000 systems, the cost of the microprocessor in the system is $1065 ($65 for manufacturing and $1000 for amortized development cost). These differences are significant, so your chance for long-term success depends on design cost at your starting point, the size of your potential market, and the difference between the rate of growth in microprocessor design cost (about 25% per year) and the rate of growth in the market for your systems. The competitive design cycle for new microprocessor designs is about two years, so workstation companies can expect to amortize their microprocessor development costs over about two years of sales volume. (It may take longer than two years to design a leading-edge microprocessor, so the design team for the next generation microprocessor will have to be working before the design team for the current microprocessor is done.)

Even though there was a microprocessor-based hobby market for personal computers beginning in 1974, the personal computer market didn't take off until IBM introduced the IBM Personal Computer in 1981. When it did take off, it really took off. Even though both the personal computer industry and the workstation industry started about the same time, by 1986 the market for personal computers was more than 15 M units while the workstation market was about 60 000 [51]. In 1995, worldwide volume for personal computers will be about 60 M units, while worldwide workstation sales will be below 1 M units. Just the *growth* in the personal computer market from year to year dwarfs the *entire* workstation market. Not only is the personal computer market significantly larger than the workstation market, but it is also a single standard—the IBM-compatible personal computer. On top of being smaller, the workstation market is also fragmented. HP, Digital, IBM, Silicon Graphics, Sun, and Intergraph are major competitors in the workstation market. Currently, Sun and HP dominate the market, with Sun controlling almost 40% of the market [52]. The leaders in the workstation market can expect to sell a few hundred thousand units a year, while the rest of their competitors may consider 100 000 units a grand target for sales.

Competition for market share, the size of the total market, and escalating microprocessor development costs combine to make it difficult for the workstation companies to fund competitive microprocessor development. In the PC market, unit volumes are so high that the cost of the microprocessor in the system is dominated by manufacturing cost rather than by amortized development cost. In the workstation market, however, just the opposite is true: cost of the microprocessor in the system is dominated by amortized development cost. This situation grows worse if the workstation market grows more slowly than the rate of increase in development cost ($\sim$ 25% per year).

Since the introduction in the early 1980's of the RISC-based workstation, its advocates have been eyeing the personal computer market. The personal computer market has always seemed an easy target for workstation systems. Armed with much higher reported performance and smaller die sizes for their microprocessors, the workstation companies felt the personal computer market was ripe for encroachment [53]. (A smaller die looks cheaper if manufacturing cost is considered and the contribution of amortized development cost is overlooked.) Early arguments for the replacement of x86-based personal computers by RISC-based personal computers rested on the premise that no one programmed in assembly language any longer. Since programming was all in higher-level languages, users would soon notice the higher reported performance for the workstations and convert by simply recompiling their programs for the superior workstation system. Unix provided the operating system bridge which made applications portable, since Unix was available on all the platforms. It didn't happen; Unix and RISC-based workstations didn't supplant the PC. Most of the installed base of PC's ran DOS, not Unix. PC applications ran under DOS and many PC applications had been written in assembler. Users owning the installed base couldn't be convinced to convert to Unix and the applications couldn't be recompiled.

After portable applications and operating systems, the next hope was the Open Software Foundation's 1989 Architecture-Neutral Distribution Format (ANDF) initiative [15]. The instruction set of the underlying microprocessor in the system didn't matter if all applications were distributed in an intermediate form which was independent of any particular instruction set [54]–[56]. It didn't happen, either [57]. Two barriers to the proliferation of ANDF are: 1) inventing and getting consensus on an intermediate form and 2) getting the developer community to commit to intermediate form distribution.

Neutral distribution formats gave way to another wave of standard operating systems, this time Windows NT [58]. As the reasoning went, the nobody-programs-in-assembler effort had failed because applications *had* been programmed

in assembler and also tied to DOS or Windows for the PC and these programs, which hosted all the applications volume, were tied via assembly language programming, to the x86. But this time, applications would be written for the Windows NT application programming interface (API) and Windows NT would be available on many platforms. In addition, the CD-ROM, with its large capacity (660 MB) and low cost, became available about the same time, offering cheap delivery for applications. Developers could write their applications in high-level languages, compile them for several platforms, and distribute several versions on the same CD-ROM. Users would then look at the superior reported performance and price/performance for the various workstations and begin to abandon the personal computer. It didn't happen. While Windows NT did provide the portable operating system unification which the Unix community was never able to achieve, there remain formidable barriers to converting the market from x86-based systems to systems based on any other CPU.

The rise of Windows NT (NT) will not benefit the x86's RISC competition. The rise of NT may force engineering applications, originally written for Unix, onto NT as workstations convert from Unix to NT. The major attraction in NT for RISC vendors is that it provides the OS unification Unix has never been able to achieve and, therefore, frees the RISC vendors of one major system development expense (the OS). Developers porting engineering applications from Unix to NT will look at the 100%–0% relative market shares for x86 and *your_favorite_workstation* and will, therefore, surely provide support for the x86. This will accelerate the migration of engineering applications from RISC to x86. Market share for RISC-based workstations will decline (even faster).

When margins were high, the thresholds for profitable system and application development were low. As margins fall, these thresholds rise. RISC vendors' unit volumes may not be growing fast enough to support profitable development in the face of falling margins. A few years ago, it may have been profitable to develop a system and applications for 50 K units. Today, due to lower margins, the system volume required to net the same cash may be 250 K units, but sales growth in the interim has been too low to reach 250 K units.

But the workstation companies aren't giving up. They must eye volumes in the PC market with considerable envy. Strategic planners for the workstation industry are working from a chart I call *The BB and the Beachball*. The beachball represents the installed base or market share for the PC and the BB represents either the installed base or market share for workstations. Workstation planners look at *The BB and the Beachball* and they brainstorm over ways for the BB to grow at the expense of the beachball. In an ironic twist, strategic planners in the PC industry also use *The BB and the Beachball*, except the beachball represents the world population and the BB represents the PC's installed base or market share. PC strategic planners look at *The BB and the Beachball* and they brainstorm over ways to increase the size of the BB. Workstation strategic planners work to increase their share of a potential market of 60 M units while PC strategic planners work to increase their share of a potential market of 5 B units. Market share for workstations is decreasing as a percent of the installed base.

This must be frustrating for planners at the workstation companies, since they have better reported performance and better reported price/performance than the PC's they would like to displace. Why are they unable to gain market share? First, workstations and PC's have different customers and use different sales models. PC sales is a low-margin, product-oriented business. PC's are sold through mail order and at discount stores. Workstation sales is a high-margin, service-oriented business. Customers for PC's expect low prices for hardware and software and they expect little or no service and support. Customers for workstations expect high prices for hardware and software and they expect plenty of service and support. Second, PC's are designed for best performance in a cost-competitive market. Workstations are designed for best performance in a performance-competitive market. Workstations show superior reported performance because they use expensive, performance-oriented system designs. Top-of-the-line workstations show great performance. But top-of-the-line workstations don't compete with the PC because they are much more expensive than a top-of-the-line PC. Price and price/performance of low-end workstations must be compared with price and price/performance of the PC, since only the lowest priced workstations have any potential overlap with the PC market. The low end of the workstation market is the only segment that matters in the competition for market share, and there the PC has a lower price and better price/performance. Hardware development cost is an issue. It costs more to develop a workstation than it does to develop a PC. Cost of development for a workstation is borne by the workstation manufacturer. Workstation manufacturers compete on raw performance of their high-end systems. Workstations were once the concept cars for the PC market: PC makers copied successful innovations from the workstation market. Workstation margins were high enough to support system innovations for the performance-oriented workstation market. Margins were too low to support system innovations for the cost-conscious PC market. But high volumes and low margins in the PC market have driven innovation from the system makers to the chip makers, where it is faster and more efficient. PC's are now evolving faster than workstations and are still cheaper to produce since investments by chip makers are amortized across a huge market while each workstation manufacturer still pays for development of the operating system, BIOS, and system design. The cost of PC innovations may be amortized across many system manufacturers and tens of millions of units. The cost of workstation innovation is borne by each manufacturer and may be amortized across only tens of thousands of units. The difference between the workstation market and the PC market is similar to the difference between the Macintosh market and the PC market. Apple has controlled the market for its products by retaining control of the operating system, BIOS, and hardware platform. This

control allowed Apple to control margins, but also increased its costs relative to those of PC makers. Price competition and wide availability of systems and components helped the IBM-compatible PC capture about 90% of the personal computer market to Apple's 10%. Software is another issue. PC software is cheap. Workstation software is expensive. PC software is cheap partly because development costs can be amortized across a large installed base and partly because PC software hasn't been enormously complex. Cheap software doesn't get much support. Workstation software is expensive partly because development costs are amortized across a relatively small installed base and partly because software for specialized engineering applications can be enormously complex. Complex software requires substantial support. The workstation market has been protected from encroachment by the PC by performance requirements for its compute-intensive applications and the relatively small size of the application base. As the PC industry matures, PC performance is increasing rapidly and software developers are moving down the food chain. Software developers began by concentrating on applications with potential sales of tens of millions, then moved to applications with potential sales of millions, and now are moving to applications with potential sales of hundreds of thousands. PC software developers are beginning to develop applications which were once the province of workstation developers [51]. The PC will bring adequate performance and low software prices to the workstation market. The engineering design industry may take a step backward in application sophistication and complexity as the industry moves from the support-intensive workstation model to the no-support PC model.

X. RISC CPU Suppliers

Microprocessor suppliers for the workstation companies face a difficult problem. The workstation market is growing slowly and may even begin to decline (assuming the definition of a workstation isn't altered to incorporate high-end PC's). At a time when microprocessor development costs are escalating rapidly, how do five suppliers to a workstation company, with aggregate annual shipments of 100 000 units, recover their microprocessor development costs? Even if you wipe out your competitors and become the sole supplier of 100 K systems a year, you will have to add $1000 to the manufacturing cost for each microprocessor in the system just to amortize the cost of a $200 M microprocessor development over the two year design cycle. So, what do the other four companies do?

Back to *The BB and the Beachball.* Only this time the BB is the market for microprocessors in CPU applications and the beachball is the market for microprocessors in embedded control. The annual market for microprocessors as the CPU in a computer system is about 60 M. The market for microprocessors (of all types) in embedded control is about 4000 M [4]. If strategic planners for high-end microprocessors view volumes in the PC market with envy, imagine what they must feel when they look at volumes in the embedded control market. With a market this size, they reason there *must* be room for high-end RISC microprocessors to capture market share.

But the embedded control market isn't just four billion microprocessors up for grabs. I divided the microprocessor market into CPU and embedded control segments. The embedded control market itself is divided into segments just as the CPU market is divided into x86 and RISC segments. The embedded control market is divided into at least three segments: zero cost, zero delay, and zero power [51].

The zero-cost segment, which to a first approximation, *represents all of the embedded control market,* is the segment for which low cost is the overriding consideration. Most microprocessors go into consumer appliances (microwave ovens, electric razors, blenders, toasters, and washing machines). These are all commodity markets, which means they sell in high volumes (millions of units to tens of millions of units). These markets are characterized by intense price competition, so substantial effort goes into reducing production cost. The ideal would be zero cost to implement.

The zero-power market, which to a first approximation, represents 0% of the embedded control market, is the segment for which zero power dissipation represents the ideal. These applications are consumer items, such as smoke detectors, cellular phones, pagers, pacemakers, hearing aids, and pocket calculators, which should run forever on a single button-size battery or weak ambient light. As with all consumer applications, minimum product cost remains a first-rate concern.

The zero-delay market, which to a first approximation, represents 0% of the embedded control market, is the segment for which zero delay from data in to result out represents the ideal. These applications are also consumer items, such as high-end printers, scanners, copiers, and fax machines, for which processing power and throughput are important—at minimum product cost, of course.

John Wharton suggested I had overlooked one important segment in proposing this dominant-characteristic taxonomy: the zero-volume segment. I thought he was joking until he explained. The zero-volume segment, which to more than a first approximation, represents 0% of the embedded control market, is the segment for which the application potential is nearly zero. If the application volume is going to be very close to zero, there must be some other reason to attempt to capture the application. One such motive is public relations. Intel invested considerable money and effort in the design of the 80960 MX processor, for which, at the time of implementation, the only known application was the YF-12 aircraft. When the only prototype of the YF-12 crashed, the application volume for the 960 MX actually went to zero, but even if the program had been successful, Intel could not have expected to sell more than a few thousand processors for that application. Intel must have made the investment in the 960 MX for reasons other than potential application volume and eventual profit. Perhaps public relations and press attention.

In the early 1990's, chip makers supporting the workstation manufacturers noticed the embedded control market as an attractive target for recovering escalating development costs and began to turn their attention from the workstation market with its limited opportunities to the embedded control market with its obviously attractive unit volumes. With our taxonomy of embedded control applications, we can speculate about their chances for success.

In looking for a home for a microprocessor that began life as a high-end CPU designed for a workstation, we can eliminate applications with a low ASP (average selling price) for the embedded component. There's no way to recover development cost if the microprocessor's manufacturing cost is above the ASP. Eliminating components with an ASP below $9, the zero cost segment, accounts for more than 98% of unit volumes. Microprocessors derived from high-end CPU designs are not good candidates for the zero-cost segment or the zero-power segment. That leaves the zero-delay segment and the zero-volume segment.

The 16-, 32-, and 64-b microprocessors and microcontrollers compete for applications in the zero-delay segment. The entrenched competitors in this market have small custom chip designs and ASP's under $9 (which puts most of the zero-delay segment inside the zero-cost segment, as we would expect). Entrenched competitors have large installed bases of loyal customers with development systems, established working relationships, and large investments in legacy code. So, except for the very high end of the zero-delay segment, the applications are unattractive since ASP's are low and competitors are entrenched.

That leaves the zero-volume segment. This segment is fragmented, open to encroachment, and may allow high ASP's. While design wins are possible and may be desirable from a public-relations point of view, they are economically unattractive due to low volumes.

XI. Conclusion

The x86-based PC dominates the desktop and cannot be displaced. Unit volume disparity between the x86-based PC and any of its potential usurpers is too great to overcome. It gives the x86-based PC both a cost advantage and a rate-of-evolution advantage. The rate-of-evolution advantage derives from the innovation having migrated from the system makers to the chip makers. The cost of innovation is amortized across system manufacturers and tens of millions of unit sales. Cost advantage comes from amortizing BIOS, OS, and application development costs across huge numbers and from amortizing component manufacturing across this same gigantic base.

I predict difficult times for the makers of workstation CPU's. Workstation volumes are too low to amortize the escalating development cost for a competitive high-end microprocessor design. After 14 years of trying, it's about time for them to give up the idea of encroaching on the PC market. It isn't happening and it isn't likely to. PC's are cheaper and will evolve faster. And while the embedded control market seems to offer more opportunities for encroachment than the PC market, and unit volumes are potentially higher, the universal sensitivity of the market to cost and the presence of experienced, entrenched competitors make significant gains unlikely.

Future performance improvements in desktop computers will come from subverting the execution of instructions. Aided by the rise of standardization in both the underlying processor (x86) and the API's for developing applications, application and OS accelerators will proliferate. These accelerators have the advantage of maintaining (and even benefiting from) object code compatibility, so they can invade the installed base incrementally and cheaply. Eventually, application and OS accelerators will be based on programmable logic and in the long term, programmable logic will migrate onto the microprocessor itself where it can be even more efficient at subverting the execution of native x86 instructions. In 1994 Intel, the world's leading manufacturer of x86 microprocessors, sold its PLD business to Altera (a leading PLD maker). It was a mistake: Intel should have bought Altera.

We began with fixed hardware and fixed algorithms in combinational implementations. The computer gave us fixed hardware and variable algorithms. From its commercial introduction in 1971, the microprocessor, representing the computer in the world of integrated circuits, displaced combinational logic in many implementations and grew to a market of over 4000 M units by 1995. The microprocessor is and always has been an embedded control component. But the microprocessor has been an interim solution. It is a weigh station between the fixed hardware and fixed algorithm of combinational logic implementations and the vairable logic and variable algorithm of programmable logic implementations.

References

[1] S. Augarten, *State of the Art.* New Haven, CT: Ticknor & Fields, 1983.

[2] G. Moore, "Cramming more components onto integrated circuits," *Electron.*, pp. 114–117, Apr. 1965.

[3] M. Clarke, "Digital to sample 300 MHz alpha AXP," *EE Times*, p. 14, Sept. 1994.

[4] J. Quinn, "Integrated circuit engineering corporation," *Microprocessor 1995*, Scottsdale, AZ, 1995.

[5] S. Veit, *Stan Veit's History of the Personal Computer.* Asheville, NC: WorldComm, 1993.

[6] N. Tredennick, "Implementation decisions for the MC68000 microprocessor," in *Proc. 3rd Rocky Mt. Symp. on Microcomput.: Syst., Software, Architecture.*

[7] D. McGregor, D. Mothersole, and B. Moyer, "The Motorola MC68020," *IEEE Micro*, pp. 101–118, Aug. 1984.

[8] B. Case, "X86 has plenty of performance headroom," *Microprocessor Rep.*, vol. 8, no. 11, pp. 9–14, Aug. 1994.

[9] M. Jones and P. Penrod, "SMP: Multiprocessing hits the desktop," *Windows Sources*, vol. 3, no. 2, pp. 162–168, Feb. 1995.

[10] H. Norr, "Multiprocessing to define desktop in next PC wave," *MacWeek*, vol. 9, no. 12, p. 1, Mar. 1995.

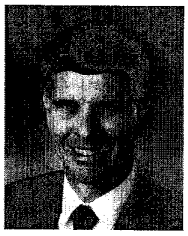
[11] E. Tittel, "Multiprocessing nears prime time," *InfoWorld*, vol. 17, no. 14, p. 51, Apr. 1995.

[12] J. Norwood and S. Vaidyanathan, "Symmetric multiprocessing for PCs," *Dr. Dobb's J.*, vol. 19, no. 1, pp. 80–85, Jan. 1994.

[13] L. Gwennap, "Microprocessors head toward MP on a chip," *Microprocessor Rep.*, vol. 8, no. 6, pp. 18–21, May 1994.

[14] B. Corthers and J. Pontin, "Pentium now, VLIW later: Memphis, VLIW future for Intel, Microsoft," *InfoWorld*, vol. 17, no. 5, p. 1, Jan. 1995.

[15] L. Gwennap, "VLIW: The wave of the future," *Microprocessor Rep.*, vol. 6, no. 2, pp. 18–21, Feb. 1994.
[16] C. Corcoran, "PC makers look beyond RISC to faster VLIW systems," *InfoWorld*, vol. 16, no. 3, p. 29, Jan. 1994.
[17] J. Lenell and N. Bagherzadeh, "A performance comparison of several superscalar processor models with a VLIW processor," *Microprocessors and Microsyst.*, vol. 18, no. 3, pp. 131–139, Apr. 1994.
[18] L. Gwennap, "Architects debate VLIW, single-chip MP; or something completely different—reprogrammable accelerators," *Microprocessor Rep.*, vol. 8, no. 16, pp. 20–21, Dec. 1994.
[19] N. Tredennick, "The future for programmable logic," *Dr. Dobb's J.*, vol. 20, no. 7, pp. 60–68, July 1995.
[20] J. Prosise, "How accelerator boards speed up Windows," *PC Mag.*, vol. 13, no. 4, pp. 257–259, Feb. 1994.
[21] M. Desmond, D. Miller, and U. Diehlmann, "Turbocharging Windows," *PC World*, vol. 11, no. 10, pp. 206–216, Oct. 1993.
[22] M. Desmond and F. Dunn, "Speed demons: Top Windows accelerators," *PC World*, vol. 12, no. 9, pp. 172–181, Sept. 1994.
[23] N. Miner, "PCs edge closer to 3-D environment; Apple's Quickdraw API, graphics boards extend 3-D use," *InfoWorld*, vol. 17, no. 12, p. 1, Mar. 1995.
[24] A. Cataldo, "3D graphics alliances proliferate; Apple adds another API," *Electron. News*, vol. 41, no. 2053, p. 30, Feb. 1995.
[25] C. Seiter, "Photoshop accelerators," *MacWorld*, vol. 11, no. 4, pp. 126–131, Apr. 1994.
[26] S. Wahlstrom, "Programmable logic arrays—cheaper by the millions," *Electronics*, pp. 90–95, Dec. 1967.
[27] S. Trimberger, *Field-Programmable Gate Array Technology*. Boston: Kluwer, 1994, pp. 1–14.
[28] J. Rosenberg, "Hardware acceleration using cache logic FPGAS," in *Proc. Design SuperCon 95*, Feb. 1995.
[29] S. Singh and P. Bellec, "Virtual hardware for graphics applications using FPGAs," in *Proc. FCCM '94*, Apr. 1994.
[30] N. Tredennick, "Reconfigurable hardware in computer systems," in *Proc. Design SuperCon 95: Manage. Design Conf.*, Feb. 1995, chap. 7, pp. 1–18.
[31] ——, "PC market study," *EDGE: Work-Group Computing Rep.*, vol. 6, no. 245, p. 6, Jan. 1995.
[32] S. Garfinkel, D. Weise, and S. Strassman, *Unix-Haters Handbook*. San Mateo, CA: IDG, 1994.
[33] D. Ditzel and D. Patterson, "Retrospective on high-level language computer architecture," in *7th Annu. Symp. on Computer Architecture Conf. Proc.*, pp. 97–104, 1980.
[34] M. Katevenis, "Reduced instruction set computer architectures for VLSI," Ph.D. dissertation, rep. no. UCB/CSD 83/141, Univ. Calif. at Berkeley, Oct. 1983.
[35] D. Patterson and C. Sequin, "RISC I: A reduced instruction set VLSI computer," in *8th Annu. Symp. on Computer Architecture Conf. Proc.*, 1981, pp. 443–457.
[36] G. Radin, "The 801 minicomputer," *IBM J. Res. and Develop.*, vol. 37, no. 3, pp. 237–246, May 1983.
[37] D. Fitzpatrick *et al.*, "A RISCy approach to VLSI," *VLSI Design*, 4th quart., pp. 14–20, 1981.
[38] N. Tredennick, "It's not RISC vs. CISC—It's new vs. old," *Microprocessor Rep.*, vol. 3, no. 2, pp. 12–16, Feb. 1989.
[39] ——, "1991: The year of the RISC," *Microprocessor Rep.*, vol. 5, no. 2, pp. 3, 16–17, Feb. 1991.
[40] D. Patterson, "Reduced instruction set computers," *Commun. ACM*, vol. 28, no. 1, pp. 8–21, Jan. 1985.
[41] N. Tredennick, "MIPS and sunset," *Microprocessor Rep.*, vol. 5, no. 12, pp. 12–17, June 1991.
[42] ——, "Workstation market," *EDGE Work-Group Computing Rep.*, vol. 6, no. 253, p. 2, Mar. 1995.
[43] F. Gardner, "Workstations: Stronger than ever," *Computer Reseller News*, no. 629, p. 89, May 1995.
[44] B. Catanzaro, Ed., "Licensing strategy at sun microsystems," in *The SPARC Technical Papers*. New York: Springer-Verlag, 1991, chap. 31.
[45] N. Tredennick, "The power play," *Microprocessor Rep.*, vol. 7, no. 1, pp. 25–26, Jan. 1993.
[46] ——, "Power play: Year two," *Microprocessor Rep.*, vol. 8, no. 12, pp. 18–21, Sept. 1994.
[47] L. Boysel and J. Murphy, "Four-phase LSI logic offers new approach to computer designer," *Computer Design*, pp. 141–146, Apr. 1970.
[48] ——, "MIPS' T5 moves ahead, but TFP slips out," *Microprocessor Rep.*, vol. 7, no. 7, p. 4, May 1993.
[49] L. Gwennap, "Estimating IC manufacturing costs," *Microprocessor Rep.*, vol. 7, no. 10, pp. 12–16, Aug. 1993.
[50] N. Tredennick, "Die like a man," *Microprocessor Rep.*, vol. 6, no. 6, pp. 18–21, May 1992.
[51] ——, "The business of microprocessors: Beyond the folklore," in *Embedded Syst. Conf. Proc.*, vol. 1, no. 21–24, pp. 5–19, Sept. 1992.
[52] S. Friedman, "RISC workstation wallop," *Inform. Week*, no. 522, p. 76, Apr. 1995.
[53] J. Wharton, "Why RISC is doomed," *Microprocessor Rep.*, vol. 6, no. 11, pp. 14–17, Aug. 1992.
[54] B. O'Connell, "High-tech's twilight zone," *DEC Profess.*, vol. 12, no. 1, pp. 22–27, Jan. 1993.
[55] M. Goulde, "At age 5, OSF looks to a challenging future," *Distrib. Computing Monitor*, vol. 8, no. 8, pp. 24–27, Aug. 1993.
[56] L. Seltzer, "COSE and ANDF: A perfect combination?" *PC Week*, vol. 10, no. 29, p. 8, July 1993.
[57] M. Goulde and J. Rymer, "Moving on: OSF and HyperDesk cut back," *Distrib. Computing Monitor*, vol. 9, no. 4, pp. 29–31, Apr. 1994.
[58] L. Gwennap, "Portable software threatens x86 hegemony; experts debate whether RISC will eventually triumph," *Microprocessor Rep.*, vol. 7, no. 16, pp. 20–21, Dec. 1993.
[59] M. Slater, "Pentium falls short of P5's promises," *Microprocessor Rep.*, vol. 7, no. 15, p. 3, Nov. 1993.



Nick Tredennick received the B.S.E.E. and M.S.E.E. degrees from Texas Tech, Lubbock, TX, and the Ph.D. degree in electrical engineering from the University of Texas.

He has been President, Vice President of R&D, Chief Scientist, and Director of Engineering for various small companies. He worked at IBM Watson Research where he did the logic design for the IBM Micro/370 and at Motorola, where he did the logic design for the MC68000. He taught at the University of California at Berkeley and at the University of Texas in Austin. He has written a textbook and more than 35 technical papers, and has seven patents in microprocessor design and others pending in programmable hardware design.

Dr. Tredennick is a member of the Army Science Board.