1 Fundamentals of Computer Design

And now for something completely different.

Monty Python's Flying Circus

1.1 Introduction

Computer technology has made incredible progress in the roughly 55 years since the first general-purpose electronic computer was created. Today, less than a thousand dollars will purchase a personal computer that has more performance, more main memory, and more disk storage than a computer bought in 1980 for 1 million dollars. This rapid rate of improvement has come both from advances in the technology used to build computers and from innovation in computer design.

Although technological improvements have been fairly steady, progress arising from better computer architectures has been much less consistent. During the first 25 years of electronic computers, both forces made a major contribution; but beginning in about 1970, computer designers became largely dependent upon integrated circuit technology. During the 1970s, performance continued to improve at about 25% to 30% per year for the mainframes and minicomputers that dominated the industry.

The late 1970s saw the emergence of the microprocessor. The ability of the microprocessor to ride the improvements in integrated circuit technology more closely than the less integrated mainframes and minicomputers led to a higher rate of improvement—roughly 35% growth per year in performance.

This growth rate, combined with the cost advantages of a mass-produced microprocessor, led to an increasing fraction of the computer business being based on microprocessors. In addition, two significant changes in the computer marketplace made it easier than ever before to be commercially successful with a new architecture. First, the virtual elimination of assembly language programming reduced the need for object-code compatibility. Second, the creation of standardized, vendor-independent operating systems, such as UNIX and its clone, Linux, lowered the cost and risk of bringing out a new architecture.

These changes made it possible to successfully develop a new set of architectures, called RISC (Reduced Instruction Set Computer) architectures, in the early 1980s. The RISC-based machines focused the attention of designers on two critical performance techniques, the exploitation of instruction-level parallelism (initially through pipelining and later through multiple instruction issue) and the use of caches (initially in simple forms and later using more sophisticated organizations and optimizations). The combination of architectural and organizational enhancements has led to 20 years of sustained growth in performance at an annual rate of over 50%. Figure 1.1 shows the effect of this difference in performance growth rates.

The effect of this dramatic growth rate has been twofold. First, it has significantly enhanced the capability available to computer users. For many applications, the highest-performance microprocessors of today outperform the supercomputer of less than 10 years ago.

Second, this dramatic rate of improvement has led to the dominance of microprocessor-based computers across the entire range of the computer design. Workstations and PCs have emerged as major products in the computer industry. Minicomputers, which were traditionally made from off-the-shelf logic or from

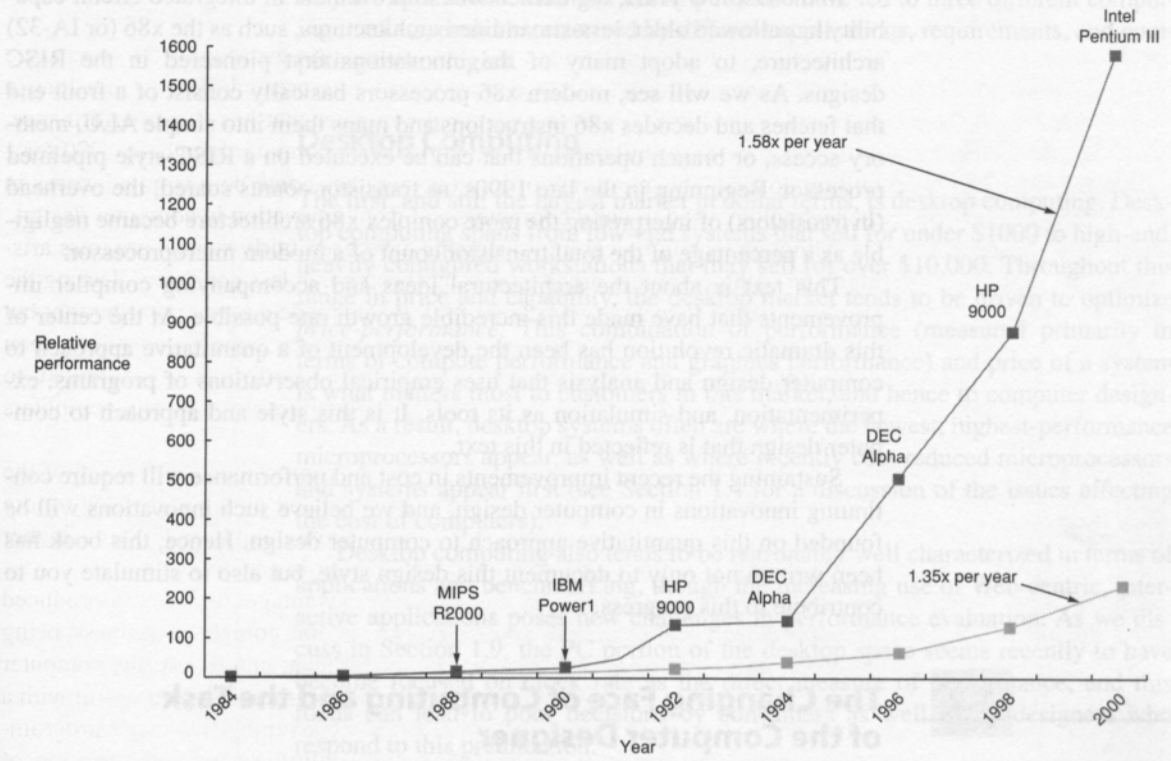

Figure 1.1 Growth in microprocessor performance since the mid-1980s has been substantially higher than in earlier years as shown by plotting SPECint performance. This chart plots relative performance as measured by the SPECint benchmarks with base of one being a VAX 11/780. Since SPEC has changed over the years, performance of newer machines is estimated by a scaling factor that relates the performance for two different versions of SPEC (e.g., SPEC92 and SPEC95). Prior to the mid-1980s, microprocessor performance growth was largely technology driven and averaged about 35% per year. The increase in growth since then is attributable to more advanced architectural and organizational ideas. By 2001 this growth led to a difference in performance of about a factor of 15. Performance for floating-point-oriented calculations has increased even faster.

gate arrays, have been replaced by servers made using microprocessors. Mainframes have been almost completely replaced with multiprocessors consisting of small numbers of off-the-shelf microprocessors. Even high-end supercomputers are being built with collections of microprocessors.

Freedom from compatibility with old designs and the use of microprocessor technology led to a renaissance in computer design, which emphasized both architectural innovation and efficient use of technology improvements. This renaissance is responsible for the higher performance growth shown in Figure 1.1—a rate that is unprecedented in the computer industry. This rate of growth has compounded so that by 2001, the difference between the highest-performance microprocessors and what would have been obtained by relying solely on technology, including improved circuit design, was about a factor of 15.

In the last few years, the tremendous improvement in integrated circuit capability has allowed older, less-streamlined architectures, such as the x86 (or IA-32) architecture, to adopt many of the innovations first pioneered in the RISC designs. As we will see, modern x86 processors basically consist of a front end that fetches and decodes x86 instructions and maps them into simple ALU, memory access, or branch operations that can be executed on a RISC-style pipelined processor. Beginning in the late 1990s, as transistor counts soared, the overhead (in transistors) of interpreting the more complex x86 architecture became negligible as a percentage of the total transistor count of a modern microprocessor.

This text is about the architectural ideas and accompanying compiler improvements that have made this incredible growth rate possible. At the center of this dramatic revolution has been the development of a quantitative approach to computer design and analysis that uses empirical observations of programs, experimentation, and simulation as its tools. It is this style and approach to computer design that is reflected in this text.

Sustaining the recent improvements in cost and performance will require continuing innovations in computer design, and we believe such innovations will be founded on this quantitative approach to computer design. Hence, this book has been written not only to document this design style, but also to stimulate you to contribute to this progress.

1.2 The Changing Face of Computing and the Task of the Computer Designer

In the 1960s, the dominant form of computing was on large mainframes—machines costing millions of dollars and stored in computer rooms with multiple operators overseeing their support. Typical applications included business data processing and large-scale scientific computing. The 1970s saw the birth of the minicomputer, a smaller-sized machine initially focused on applications in scientific laboratories, but rapidly branching out as the technology of time-sharing—multiple users sharing a computer interactively through independent terminals—became widespread. The 1980s saw the rise of the desktop computer based on microprocessors, in the form of both personal computers and workstations. The individually owned desktop computer replaced time-sharing and led to the rise of servers—computers that provided larger-scale services such as reliable, long-term file storage and access, larger memory, and more computing power. The 1990s saw the emergence of the Internet and the World Wide Web, the first successful handheld computing devices (personal digital assistants or PDAs), and the emergence of high-performance digital consumer electronics, from video games to set-top boxes.

These changes have set the stage for a dramatic change in how we view computing, computing applications, and the computer markets at the beginning of the millennium. Not since the creation of the personal computer more than 20 years ago have we seen such dramatic changes in the way computers appear and in how they are used. These changes in computer use have led to three different computing markets, each characterized by different applications, requirements, and computing technologies.

Desktop Computing

The first, and still the largest market in dollar terms, is desktop computing. Desktop computing spans from low-end systems that sell for under $1000 to high-end, heavily configured workstations that may sell for over $10,000. Throughout this range in price and capability, the desktop market tends to be driven to optimize *price-performance*. This combination of performance (measured primarily in terms of compute performance and graphics performance) and price of a system is what matters most to customers in this market, and hence to computer designers. As a result, desktop systems often are where the newest, highest-performance microprocessors appear, as well as where recently cost-reduced microprocessors and systems appear first (see Section 1.4 for a discussion of the issues affecting the cost of computers).

Desktop computing also tends to be reasonably well characterized in terms of applications and benchmarking, though the increasing use of Web-centric, interactive applications poses new challenges in performance evaluation. As we discuss in Section 1.9, the PC portion of the desktop space seems recently to have become focused on clock rate as the direct measure of performance, and this focus can lead to poor decisions by consumers as well as by designers who respond to this predilection.

Servers

As the shift to desktop computing occurred, the role of servers to provide larger-scale and more reliable file and computing services grew. The emergence of the World Wide Web accelerated this trend because of the tremendous growth in demand for Web servers and the growth in sophistication of Web-based services. Such servers have become the backbone of large-scale enterprise computing, replacing the traditional mainframe.

For servers, different characteristics are important. First, availability is critical. We use the term "availability," which means that the system can reliably and effectively provide a service. This term is to be distinguished from "reliability," which says that the system never fails. Parts of large-scale systems unavoidably fail; the challenge in a server is to maintain system availability in the face of component failures, usually through the use of redundancy. This topic is discussed in detail in Chapter 7.

Why is availability crucial? Consider the servers running Yahoo!, taking orders for Cisco, or running auctions on eBay. Obviously such systems must be operating seven days a week, 24 hours a day. Failure of such a server system is far more catastrophic than failure of a single desktop. Although it is hard to estimate the cost of downtime, Figure 1.2 shows one analysis, assuming that downtime is

		Annual losses (millions of $) with downtime of		
Application	**Cost of downtime per hour (thousands of $)**	**1% (87.6 hrs/yr)**	**0.5% (43.8 hrs/yr)**	**0.1% (8.8 hrs/yr)**
Brokerage operations	$6450	$565	$283	$56.5
Credit card authorization	$2600	$228	$114	$22.8
Package shipping services	$150	$13	$6.6	$1.3
Home shopping channel	$113	$9.9	$4.9	$1.0
Catalog sales center	$90	$7.9	$3.9	$0.8
Airline reservation center	$89	$7.9	$3.9	$0.8
Cellular service activation	$41	$3.6	$1.8	$0.4
Online network fees	$25	$2.2	$1.1	$0.2
ATM service fees	$14	$1.2	$0.6	$0.1

Figure 1.2 The cost of an unavailable system is shown by analyzing the cost of downtime (in terms of immediately lost revenue), assuming three different levels of availability and that downtime is distributed uniformly. These data are from Kembel [2000] and were collected and analyzed by Contingency Planning Research.

distributed uniformly and does not occur solely during idle times. As we can see, the estimated costs of an unavailable system are high, and the estimated costs in Figure 1.2 are purely lost revenue and do not account for the cost of unhappy customers!

A second key feature of server systems is an emphasis on scalability. Server systems often grow over their lifetime in response to a growing demand for the services they support or an increase in functional requirements. Thus, the ability to scale up the computing capacity, the memory, the storage, and the I/O bandwidth of a server is crucial.

Lastly, servers are designed for efficient throughput. That is, the overall performance of the server—in terms of transactions per minute or Web pages served per second—is what is crucial. Responsiveness to an individual request remains important, but overall efficiency and cost-effectiveness, as determined by how many requests can be handled in a unit time, are the key metrics for most servers. (We return to the issue of performance and assessing performance for different types of computing environments in Section 1.5).

Embedded Computers

Embedded computers—computers lodged in other devices where the presence of the computers is not immediately obvious—are the fastest growing portion of the computer market. These devices range from everyday machines (most microwaves, most washing machines, most printers, most networking switches, and all cars contain simple embedded microprocessors) to handheld digital devices (such as palmtops, cell phones, and smart cards) to video games and digital set-top boxes. Although in some applications (such as palmtops) the computers are programmable, in many embedded applications the only programming occurs in connection with the initial loading of the application code or a later software upgrade of that application. Thus, the application can usually be carefully tuned for the processor and system. This process sometimes includes limited use of assembly language in key loops, although time-to-market pressures and good software engineering practice usually restrict such assembly language coding to a small fraction of the application. This use of assembly language, together with the presence of standardized operating systems, and a large code base has meant that instruction set compatibility has become an important concern in the embedded market. Simply put, like other computing applications, software costs are often a large part of the total cost of an embedded system.

Embedded computers have the widest range of processing power and cost—from low-end 8-bit and 16-bit processors that may cost less than a dollar, to full 32-bit microprocessors capable of executing 50 million instructions per second that cost under 10 dollars, to high-end embedded processors that cost hundreds of dollars and can execute a billion instructions per second for the newest video game or for a high-end network switch. Although the range of computing power in the embedded computing market is very large, price is a key factor in the design of computers for this space. Performance requirements do exist, of course, but the primary goal is often meeting the performance need at a minimum price, rather than achieving higher performance at a higher price.

Often, the performance requirement in an embedded application is a real-time requirement. A *real-time performance requirement* is one where a segment of the application has an absolute maximum execution time that is allowed. For example, in a digital set-top box the time to process each video frame is limited, since the processor must accept and process the next frame shortly. In some applications, a more sophisticated requirement exists: the average time for a particular task is constrained as well as the number of instances when some maximum time is exceeded. Such approaches (sometimes called *soft real-time*) arise when it is possible to occasionally miss the time constraint on an event, as long as not too many are missed. Real-time performance tends to be highly application dependent. It is usually measured using kernels either from the application or from a standardized benchmark (see the EEMBC benchmarks described in Section 1.5). With the growth in the use of embedded microprocessors, a wide range of benchmark requirements exist, from the ability to run small, limited code segments to the ability to perform well on applications involving tens to hundreds of thousands of lines of code.

Two other key characteristics exist in many embedded applications: the need to minimize memory and the need to minimize power. In many embedded applications, the memory can be a substantial portion of the system cost, and it is important to optimize memory size in such cases. Sometimes the application is expected to fit totally in the memory on the processor chip; other times the application needs to fit totally in a small off-chip memory. In any event, the importance of memory size translates to an emphasis on code size, since data size is dictated by the application. As we will see in the next chapter, some architectures have special instruction set capabilities to reduce code size. Larger memories also mean more power, and optimizing power is often critical in embedded applications. Although the emphasis on low power is frequently driven by the use of batteries, the need to use less expensive packaging (plastic versus ceramic) and the absence of a fan for cooling also limit total power consumption. We examine the issue of power in more detail later in the chapter.

Another important trend in embedded systems is the use of processor cores together with application-specific circuitry. Often an application's functional and performance requirements are met by combining a custom hardware solution together with software running on a standardized embedded processor core, which is designed to interface to such special-purpose hardware. In practice, embedded problems are usually solved by one of three approaches:

1. The designer uses a combined hardware/software solution that includes some custom hardware and an embedded processor core that is integrated with the custom hardware, often on the same chip.

2. The designer uses custom software running on an off-the-shelf embedded processor.

3. The designer uses a digital signal processor and custom software for the processor. *Digital signal processors* (DSPs) are processors specially tailored for signal-processing applications. We discuss some of the important differences between digital signal processors and general-purpose embedded processors in the next chapter.

Most of what we discuss in this book applies to the design, use, and performance of embedded processors, whether they are off-the-shelf microprocessors or microprocessor cores, which will be assembled with other special-purpose hardware. The design of special-purpose, application-specific hardware and architecture and the use of DSPs, however, are outside of the scope of this book. Figure 1.3 summarizes these three classes of computing environments and their important characteristics.

The Task of the Computer Designer

The task the computer designer faces is a complex one: Determine what attributes are important for a new machine, then design a machine to maximize performance while staying within cost and power constraints. This task has many aspects, including instruction set design, functional organization, logic design, and implementation. The implementation may encompass integrated circuit design, packaging, power, and cooling. Optimizing the design requires familiarity with a very wide range of technologies, from compilers and operating systems to logic design and packaging.

In the past, the term *computer architecture* often referred only to instruction set design. Other aspects of computer design were called *implementation*, often

Feature	Desktop	Server	Embedded
Price of system	\$1000–\$10,000	\$10,000–\$10,000,000	\$10–\$100,000 (including network routers at the high end)
Price of microprocessor module	\$100–\$1000	\$200–\$2000 (per processor)	\$0.20–\$200 (per processor)
Microprocessors sold per year (estimates for 2000)	150,000,000	4,000,000	300,000,000 (32-bit and 64-bit processors only)
Critical system design issues	Price-performance, graphics performance	Throughput, availability, scalability	Price, power consumption, application-specific performance

Figure 1.3 A summary of the three computing classes and their system characteristics. Note the wide range in system price for servers and embedded systems. For servers, this range arises from the need for very large-scale multiprocessor systems for high-end transaction processing and Web server applications. For embedded systems, one significant high-end application is a network router, which could include multiple processors as well as lots of memory and other electronics. The total number of embedded processors sold in 2000 is estimated to exceed 1 billion, if you include 8-bit and 16-bit microprocessors. In fact, the largest selling microprocessor of all time is an 8-bit microcontroller sold by Intel! It is difficult to separate the low end of the server market from the desktop market, since low-end servers—especially those costing less than \$5000—are essentially no different from desktop PCs. Hence, up to a few million of the PC units may be effectively servers.

insinuating that implementation is uninteresting or less challenging. We believe this view is not only incorrect, but is even responsible for mistakes in the design of new instruction sets. The architect's or designer's job is much more than instruction set design, and the technical hurdles in the other aspects of the project are certainly as challenging as those encountered in instruction set design. This challenge is particularly acute at the present, when the differences among instruction sets are small and when there are three rather distinct application areas.

In this book the term *instruction set architecture* refers to the actual programmer-visible instruction set. The instruction set architecture serves as the boundary between the software and hardware, and that topic is the focus of Chapter 2. The implementation of a machine has two components: organization and hardware.

The term *organization* includes the high-level aspects of a computer's design, such as the memory system, the bus structure, and the design of the internal CPU (central processing unit—where arithmetic, logic, branching, and data transfer are implemented). For example, two embedded processors with identical instruction set architectures but very different organizations are the NEC VR 5432 and the NEC VR 4122. Both processors implement the MIPS64 instruction set, but they have very different pipeline and cache organizations. In addition, the 4122 implements the floating-point instructions in software rather than hardware!

Hardware is used to refer to the specifics of a machine, including the detailed logic design and the packaging technology of the machine. Often a line of machines contains machines with identical instruction set architectures and nearly identical organizations, but they differ in the detailed hardware implementation. For example, the Pentium II and Celeron are nearly identical, but offer different clock rates and different memory systems, making the Celeron more effective for low-end computers. In this book the word *architecture* is intended to cover all three aspects of computer design—instruction set architecture, organization, and hardware.

Computer architects must design a computer to meet functional requirements as well as price, power, and performance goals. Often, they also have to determine what the functional requirements are, which can be a major task. The requirements may be specific features inspired by the market. Application software often drives the choice of certain functional requirements by determining how the machine will be used. If a large body of software exists for a certain instruction set architecture, the architect may decide that a new machine should implement an existing instruction set. The presence of a large market for a particular class of applications might encourage the designers to incorporate requirements that would make the machine competitive in that market. Figure 1.4

Functional requirements	Typical features required or supported
Application area	*Target of computer*
General-purpose desktop	Balanced performance for a range of tasks, including interactive performance for graphics, video, and audio (Ch. 2, 3, 4, 5)
Scientific desktops and servers	High-performance floating point and graphics (App. G, H)
Commercial servers	Support for databases and transaction processing; enhancements for reliability and availability; support for scalability (Ch. 2, 6, 8)
Embedded computing	Often requires special support for graphics or video (or other application-specific extension); power limitations and power control may be required (Ch. 2, 3, 4, 5)
Level of software compatibility	*Determines amount of existing software for machine*
At programming language	Most flexible for designer; need new compiler (Ch. 2, 6)
Object code or binary compatible	Instruction set architecture is completely defined—little flexibility—but no investment needed in software or porting programs
Operating system requirements	*Necessary features to support chosen OS (Ch. 5, 8)*
Size of address space	Very important feature (Ch. 5); may limit applications
Memory management	Required for modern OS; may be paged or segmented (Ch. 5)
Protection	Different OS and application needs: page vs. segment protection (Ch. 5)
Standards	*Certain standards may be required by marketplace*
Floating point	Format and arithmetic: IEEE 754 standard (App. H), special arithmetic for graphics or signal processing
I/O bus	For I/O devices: Ultra ATA, Ultra SCSI, PCI (Ch. 7, 8)
Operating systems	UNIX, PalmOS, Windows, Windows NT, Windows CE, CISCO IOS
Networks	Support required for different networks: Ethernet, Infiniband (Ch. 8)
Programming languages	Languages (ANSI C, C++, Java, FORTRAN) affect instruction set (Ch. 2)

Figure 1.4 Summary of some of the most important functional requirements an architect faces. The left-hand column describes the class of requirement, while the right-hand column gives examples of specific features that might be needed. The right-hand column also contains references to chapters and appendices that deal with the specific issues.

summarizes some requirements that need to be considered in designing a new machine. Many of these requirements and features will be examined in depth in later chapters.

Once a set of functional requirements has been established, the architect must try to optimize the design. Which design choices are optimal depends, of course, on the choice of metrics. The changes in the computer applications space over the last decade have dramatically changed the metrics. Although desktop computers remain focused on optimizing cost-performance as measured by a single user, servers focus on availability, scalability, and throughput cost-performance, and embedded computers are driven by price and often power issues.

These differences and the diversity and size of these different markets lead to fundamentally different design efforts. For the desktop market, much of the effort goes into designing a leading-edge microprocessor and into the graphics and I/O system that integrate with the microprocessor. In the server area, the focus is on integrating state-of-the-art microprocessors, often in a multiprocessor architecture, and designing scalable and highly available I/O systems to accompany the processors. Finally, in the leading edge of the embedded processor market, the challenge lies in adopting the high-end microprocessor techniques to deliver most of the performance at a lower fraction of the price, while paying attention to demanding limits on power and sometimes a need for high-performance graphics or video processing.

In addition to performance and cost, designers must be aware of important trends in both the implementation technology and the use of computers. Such trends not only impact future cost, but also determine the longevity of an architecture. The next two sections discuss technology and cost trends.

1.3 Technology Trends

If an instruction set architecture is to be successful, it must be designed to survive rapid changes in computer technology. After all, a successful new instruction set architecture may last decades—the core of the IBM mainframe has been in use for more than 35 years. An architect must plan for technology changes that can increase the lifetime of a successful computer.

To plan for the evolution of a machine, the designer must be especially aware of rapidly occurring changes in implementation technology. Four implementation technologies, which change at a dramatic pace, are critical to modern implementations:

- *Integrated circuit logic technology*—Transistor density increases by about 35% per year, quadrupling in somewhat over four years. Increases in die size are less predictable and slower, ranging from 10% to 20% per year. The combined effect is a growth rate in transistor count on a chip of about 55% per year. Device speed scales more slowly, as we discuss below.

- *Semiconductor DRAM* (dynamic random-access memory)—Density increases by between 40% and 60% per year, quadrupling in three to four years. Cycle time has improved very slowly, decreasing by about one-third in 10 years. Bandwidth per chip increases about twice as fast as latency decreases. In addition, changes to the DRAM interface have also improved the bandwidth; these are discussed in Chapter 5.
- *Magnetic disk technology*—Recently, disk density has been improving by more than 100% per year, quadrupling in two years. Prior to 1990, density increased by about 30% per year, doubling in three years. It appears that disk technology will continue the faster density growth rate for some time to come. Access time has improved by one-third in 10 years. This technology is central to Chapter 7, and we discuss the trends in greater detail there.
- *Network technology*—Network performance depends both on the performance of switches and on the performance of the transmission system. Both latency and bandwidth can be improved, though recently bandwidth has been the primary focus. For many years, networking technology appeared to improve slowly: for example, it took about 10 years for Ethernet technology to move from 10 Mb to 100 Mb. The increased importance of networking has led to a faster rate of progress, with 1 Gb Ethernet becoming available about five years after 100 Mb. The Internet infrastructure in the United States has seen even faster growth (roughly doubling in bandwidth every year), both through the use of optical media and through the deployment of much more switching hardware.

These rapidly changing technologies impact the design of a microprocessor that may, with speed and technology enhancements, have a lifetime of five or more years. Even within the span of a single product cycle for a computing system (two years of design and two to three years of production), key technologies, such as DRAM, change sufficiently that the designer must plan for these changes. Indeed, designers often design for the next technology, knowing that when a product begins shipping in volume that next technology may be the most cost-effective or may have performance advantages. Traditionally, cost has decreased at about the rate at which density increases.

Although technology improves fairly continuously, the impact of these improvements is sometimes seen in discrete leaps, as a threshold that allows a new capability is reached. For example, when MOS technology reached the point where it could put between 25,000 and 50,000 transistors on a single chip in the early 1980s, it became possible to build a 32-bit microprocessor on a single chip. By the late 1980s, first-level caches could go on chip. By eliminating chip crossings within the processor and between the processor and the cache, a dramatic increase in cost-performance and performance/power was possible. This design was simply infeasible until the technology reached a certain point. Such technology thresholds are not rare and have a significant impact on a wide variety of design decisions.

Scaling of Transistor Performance, Wires, and Power in Integrated Circuits

Integrated circuit processes are characterized by the *feature size,* which is the minimum size of a transistor or a wire in either the *x* or *y* dimension. Feature sizes have decreased from 10 microns in 1971 to 0.18 microns in 2001. Since the transistor count per square millimeter of silicon is determined by the surface area of a transistor, the density of transistors increases quadratically with a linear decrease in feature size. The increase in transistor performance, however, is more complex. As feature sizes shrink, devices shrink quadratically in the horizontal dimension and also shrink in the vertical dimension. The shrink in the vertical dimension requires a reduction in operating voltage to maintain correct operation and reliability of the transistors. This combination of scaling factors leads to a complex interrelationship between transistor performance and process feature size. To a first approximation, transistor performance improves linearly with decreasing feature size.

The fact that transistor count improves quadratically with a linear improvement in transistor performance is both the challenge and the opportunity that computer architects were created for! In the early days of microprocessors, the higher rate of improvement in density was used to quickly move from 4-bit, to 8-bit, to 16-bit, to 32-bit microprocessors. More recently, density improvements have supported the introduction of 64-bit microprocessors as well as many of the innovations in pipelining and caches, which we discuss in Chapters 3, 4, and 5.

Although transistors generally improve in performance with decreased feature size, wires in an integrated circuit do not. In particular, the signal delay for a wire increases in proportion to the product of its resistance and capacitance. Of course, as feature size shrinks, wires get shorter, but the resistance and capacitance per unit length get worse. This relationship is complex, since both resistance and capacitance depend on detailed aspects of the process, the geometry of a wire, the loading on a wire, and even the adjacency to other structures. There are occasional process enhancements, such as the introduction of copper, which provide one-time improvements in wire delay. In general, however, wire delay scales poorly compared to transistor performance, creating additional challenges for the designer. In the past few years, wire delay has become a major design limitation for large integrated circuits and is often more critical than transistor switching delay. Larger and larger fractions of the clock cycle have been consumed by the propagation delay of signals on wires. In 2001, the Pentium 4 broke new ground by allocating 2 stages of its 20+-stage pipeline just for propagating signals across the chip.

Power also provides challenges as devices are scaled. For modern CMOS microprocessors, the dominant energy consumption is in switching transistors. The energy required per transistor is proportional to the product of the load capacitance of the transistor, the frequency of switching, and the square of the voltage. As we move from one process to the next, the increase in the number of transistors switching, and the frequency with which they switch, dominates the decrease in load capacitance and voltage, leading to an overall growth in power consumption. The first microprocessors consumed tenths of a watt, while a 2 GHz Pentium 4 consumes close to 100 watts. The fastest workstation and server microprocessors in 2001 consumed between 100 and 150 watts. Distributing the power, removing the heat, and preventing hot spots have become increasingly difficult challenges, and it is likely that power rather than raw transistor count will become the major limitation in the near future.

1.4 Cost, Price, and Their Trends

Although there are computer designs where costs tend to be less important—specifically supercomputers—cost-sensitive designs are of growing significance: More than half the PCs sold in 1999 were priced at less than $1000, and the average price of a 32-bit microprocessor for an embedded application is in the tens of dollars. Indeed, in the past 15 years, the use of technology improvements to achieve lower cost, as well as increased performance, has been a major theme in the computer industry.

Textbooks often ignore the cost half of cost-performance because costs change, thereby dating books, and because the issues are subtle and differ across industry segments. Yet an understanding of cost and its factors is essential for designers to be able to make intelligent decisions about whether or not a new feature should be included in designs where cost is an issue. (Imagine architects designing skyscrapers without any information on costs of steel beams and concrete!)

This section focuses on cost and price, specifically on the relationship between price and cost: price is what you sell a finished good for, and cost is the amount spent to produce it, including overhead. We also discuss the major trends and factors that affect cost and how it changes over time. The exercises and examples use specific cost data that will change over time, though the basic determinants of cost are less time sensitive. This section will introduce you to these topics by discussing some of the major factors that influence the cost of a computer design and how these factors are changing over time.

The Impact of Time, Volume, and Commodification

The cost of a manufactured computer component decreases over time even without major improvements in the basic implementation technology. The underlying principle that drives costs down is the *learning curve*—manufacturing costs decrease over time. The learning curve itself is best measured by change in *yield*—the percentage of manufactured devices that survives the testing procedure. Whether it is a chip, a board, or a system, designs that have twice the yield will have basically half the cost.

Understanding how the learning curve will improve yield is key to projecting costs over the life of the product. As an example of the learning curve in action, the price per megabyte of DRAM drops over the long term by 40% per year.