1.11 Concluding remarks

> "Where ... the ENIAC is equipped with 18,000 vacuum tubes and weighs 30 tons, computers in the future may have 1,000 vacuum tubes and perhaps weigh just 1½ tons.
> *Popular Mechanics, March 1949*

Although it is difficult to predict exactly what level of cost/performance computers will have in the future, it's a safe bet that they will be much better than they are today. To participate in these advances, computer designers and programmers must understand a wider variety of issues.

Both hardware and software designers construct computer systems in hierarchical layers, with each lower layer hiding details from the level above. This great idea of *abstraction* is fundamental to understanding today's computer systems, but it does not mean that designers can limit themselves to knowing a single abstraction. Perhaps the most important example of abstraction is the interface between hardware and low-level software, called the *instruction set architecture*. Maintaining the instruction set architecture as a constant enables many implementations of that architecture—presumably varying in cost and performance—to run identical software. On the downside, the architecture may preclude introducing innovations that require the interface to change.

There is a reliable method of determining and reporting performance by using the execution time of real programs as the metric. This execution time is related to other important measurements we can make by the following equation:

$$\frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instructions}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

We will use this equation and its constituent factors many times. Remember, though, that individually the factors do not determine performance: only the product, which equals execution time, is a reliable measure of performance.

The Big Picture

Execution time is the only valid and unimpeachable measure of performance. Many other metrics have been proposed and found wanting. Sometimes these metrics are flawed from the start by not reflecting execution time; other times a metric that is sound in a limited context is extended and used beyond that context or without the additional clarification needed to make it valid.

The key hardware technology for modern processors is silicon. Equal in importance to an understanding of integrated circuit technology is an understanding of the expected rates of technological change, as predicted by *Moore's Law*. While silicon fuels the rapid advance of hardware, new ideas in the organization of computers have improved price/performance. Two of the key ideas are exploiting parallelism in the program, normally today via multiple processors, and exploiting locality of accesses to a *memory hierarchy*, typically via caches.

Energy efficiency has replaced die area as the most critical resource of microprocessor design. Conserving power while trying to increase performance has forced the hardware industry to switch to multicore microprocessors, thereby requiring the software industry to switch to programming parallel hardware. *Parallelism* is now required for performance.

Computer designs have always been measured by cost and performance, as well as other important factors such as energy, dependability, cost of ownership, and scalability. Although this chapter has focused on cost, performance, and energy, the best designs will strike the appropriate balance for a given market among all the factors.

PARTICIPATION ACTIVITY 1.11.1: Computer technologies.

1) The most reliable method to evaluate performance is execution time.
   - ○ True
   - ○ False

2) Software developers do not need an understanding of hardware to write efficient programs.
   - ○ True
   - ○ False

3) Die area is the most critical resource of microprocessor design.
   - ○ True
   - ○ False

4) Power limitations have forced computer designers to exploit parallelism to improve system performance.
   - ○ True
   - ○ False

Road map for this book

At the bottom of these abstractions is the five classic components of a computer: datapath, control, memory, input, and output. These five components also serve as the framework for the rest of the chapters in this book:

- Datapath: COD Chapter 3 (Arithmetic for Computers), COD Chapter 4 (The Processor), COD Chapter 6 (Parallel Processor from Client to Cloud), and COD Appendix B (Graphics and Computing GPUs)
- Control: COD Chapter 4 (The Processor), COD Chapter 6 (Parallel Processor from Client to Cloud), and COD Appendix B (Graphics and Computing GPUs)
- Memory: COD Chapter 5 (Large and Fast: Exploiting Memory Hierarchy)
- Input: COD Chapters 5 (Large and Fast: Exploiting Memory Hierarchy) and 6 (Parallel Processor from Client to Cloud)
- Output: COD Chapters 5 (Large and Fast: Exploiting Memory Hierarchy) and 6 (Parallel Processor from Client to Cloud)

As mentioned above, COD Chapter 4 (The Processor) describes how processors exploit implicit parallelism, COD Chapter 6 (Parallel Processor from Client to Cloud) describes the explicitly parallel multicore microprocessors that are at the heart of the parallel revolution, and COD Appendix B (Graphics and Computing GPUs) describes the highly parallel graphics processor chip. COD Chapter 5 (Large and Fast: Exploiting Memory Hierarchy) describes how a memory hierarchy exploits locality. COD Chapter 2 (Instructions: Language of the Computer) describes instruction sets—the interface between compilers and the computer—and emphasizes the role of compilers and programming languages in using the features of the instruction set. COD Chapter 3 (Arithmetic for Computers) describes how computers handle arithmetic data. COD Appendix A (The Basics of Logic Design) introduces logic design.