Chapman & Hall/CRC
Textbooks in Computing

COMPUTER SYSTEMS ARCHITECTURE

Aharon Yadin

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

Chapman & Hall/CRC
TEXTBOOKS IN COMPUTING

COMPUTER SYSTEMS ARCHITECTURE

Aharon Yadin

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

COMPUTER SYSTEMS ARCHITECTURE

CHAPMAN & HALL/CRC TEXTBOOKS IN COMPUTING

COMPUTER SYSTEMS ARCHITECTURE

Aharon Yadin
Yezreel Valley College (YVC)
Israel

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed on acid-free paper
Version Date: 20160120

International Standard Book Number-13: 978-1-4822-3105-2 (Hardback)

Library of Congress Cataloging-in-Publication Data

Names: Yadin, Aharon, author.
Title: Computer systems architecture / Aharon Yadin.
Description: Boca Raton : Taylor & Francis Group, CRC Press, 2016. | Series: Chapman & Hall/CRC textbooks in computing | Includes bibliographical references and index.
Identifiers: LCCN 2015048879 | ISBN 9781482231052
Subjects: LCSH: Computer architecture.
Classification: LCC QA76.9.A73 Y335 2016 | DDC 004.2/2—dc23
LC record available at http://lccn.loc.gov/2015048879

Contents

Preface

THE PURPOSE OF THIS BOOK is to provide the necessary understanding of and background of hardware for IT (information technology) people. The term IT refers to a variety of disciples related to computer-based systems such as software engineering, information systems, computer science, and so on. Each one of these disciplines has its own focus and emphasis. In most cases, the hardware platform is viewed as an existing infrastructure, and it is sometimes insufficiently addressed or understood. Furthermore, technological developments in hardware in recent decades were aimed mainly at defining the hardware as a separated platform. As such, IT personnel, and especially software designers and developers, may regard it as a required layer of a computing solution, but a layer that they do not have to understand. In a sense, it is like using a car; the driver drives it but does not care about the internal mechanics. On the other hand, recent architectural developments, such as cloud computing,[*] virtualization,[†] and the abstractions required for implementing modern computing systems, emphasize the importance of an understanding of hardware. For example, desktop virtualization provides the capability to access any application, using any device. The user may use a desktop computer, a laptop, a tablet, a smartphone, and even appliances that have not yet been invented, and the application will work properly.

For that reason, this book is not about computer organization, but rather concerns ongoing issues related to computer hardware and the solutions provided by the industry for these issues.

Figure 0.1 defines most of the layers in a computer system, going top-down from the application (or program), which is usually developed using a high-level programming language, such as C#, C++, Java, and so on. In some cases, the compiler[‡] translates the high-level programming languages' instructions into assembly language, which represents the mnemonics of the instructions the machine understands. In other cases, the high-level programming languages are compiled directly into the machine language. The translated program (executable) will be able to run using services provided by the operating system, which is an additional software component, usually considered part of the infrastructure. The next level, which is a mixed software–hardware layer, is virtualization, which provides the possibility of defining virtual machines (this will be elaborated on in Chapter 10, "Additional Architectures"). The next level is the machine instructions. These binary values represent the instructions to be executed and are the only instructions the machine recognizes. In most modern computers, the binary instructions are designed using predefined building blocks, sometimes called microinstructions (this will be elaborated on in Chapter 4, "Central Processing Unit"). These building blocks are defined in the next layer of Figure 0.1. The last level is the digital circuits that understand and execute these building blocks and the instructions they produce. For quite some time, the separation between hardware and software was somewhat clear, as outlined by the dotted line in Figure 0.1.

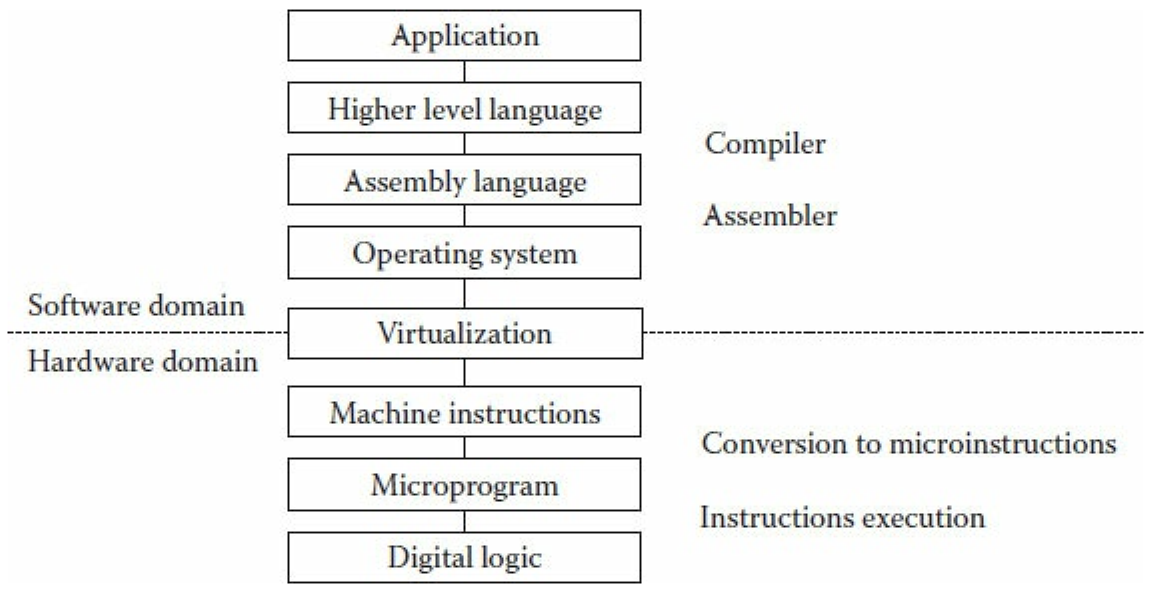

FIGURE 0.1 System layers.

Although the separation between software and hardware was clearly defined in the early days of computing, developers had to understand some hardware principles to produce reliable and fast working software. Even the recent technological developments related to cloud computing, which draw an even clearer line between the hardware and software layers of the solution, did not manage to establish a distinct separation. Furthermore, as more and more hardware components are implemented using programmable chips, software engineering methods are being used as part of the hardware development process. For example, the modular approach of defining new instructions using building blocks is a concept borrowed from systems engineering, wherein a system is usually defined using subsystems. Each subsystem is further divided into smaller and smaller parts, right up to the single function or method. This means that even the second stage of Figure 0.1, which consists of defining the common hardware blocks, is actually software based, so the once-clear separation becomes very blurred. Cloud computing, which emerged in the last decade, is intended to provide a computing infrastructure while reducing costs and allowing customers to concentrate on their core business rather than on computing-related issues. One of the main developments that led to cloud computing is virtualization (explained in Chapter 10, "Additional Architectures"). Virtualization is a software layer that creates scalable "virtual machines." These machines hide the underlying physical hardware. In many cases, the operating system enters the gap between the hardware and software by providing a layer of interface. On one hand, it provides tools and application programming interfaces (APIs) for software development, to ease access to the hardware. On the other hand, it accesses the hardware using specially developed functions that assure a higher degree of flexibility and efficiency. Due to the complex nature of software, in many cases there is a need for an additional layer above the operating system, which provides additional common functionality such as .net by Microsoft or J2EE by Oracle Corporation. These environments ease the process of software development by relieving the developer of system functions such as memory management, data encryptions, and so on. Using these virtual environments' frameworks, the developer can concentrate purely on the application's logic instead of spending time "reinventing the wheel" and developing libraries of common services.

Although all these new developments undermine the importance of hardware for software development, a basic understanding of hardware components and their effect on execution is still very important. Software can be developed with a minimal understanding of hardware; however, for developing good software that is both reliable and efficient, some understanding of hardware is of paramount importance. As part of the first stages of the software engineering process, in which the system's functionality is defined, this hardware understanding becomes vital. After the requirements are defined, the system analysis stage defines the functionality of the new system. For a simple program, the levels of abstraction provided by cloud computing and the operating system as well as additional development frameworks (such as .net a J2EE), may be sufficient. However, for a larger or even an organizational software system, some hardware understanding is needed, for example as part of the feasibility study* and calculations regarding return on investment (ROI†). After the "what" has been defined (what the system will do) as part of the design stage, the "how" is defined (how the system will be implemented). At this stage, an understanding of hardware becomes even more important. Several design decisions may have severe implications, and understanding the hardware components and their interaction with the planned software helps to produce a better design. For example, most modern computers utilize multiple cores (this will be elaborated on in the subsequent chapters). This means that such a system can execute several tasks in parallel. In a sense, it is like a dual-line telephone system that supports two calls in parallel. The telephone provides the functionality; however, a single person cannot use it. The person can switch between the lines but cannot talk simultaneously with two other people (unless it is a conference call). However, when using a multiple-core system, the application can be developed using multithreading* which will better utilize the hardware platform and provide a superior user experience when using the newly developed system. A very simple but clear system engineering example may relate to a web-based application. The old traditional application supported one user, accepted the user's input, performed the required function, produced the result, and then asked for the next input, and so on. A web-based application that has to support a varying number of concurrent users should be designed differently. There will be a small task that collects the users' input requests. This task will not handle the requests, just store them. There will be some (or even many) other tasks that are responsible for fetching the requests that were entered and executing each one of them. This software architecture can be implemented on a single-core processor, but it will be much more efficient using multiple cores, and in extreme cases, it will require virtualization (in which several physical systems will act as a single virtual machine). Another example that demonstrates the importance of hardware understanding is related to large dimensional arrays. If a program performs some calculations on an array with two (or more) dimensions, the order of indexing the array may have significant performance implications, especially when very large arrays are involved. Understanding the way the memory works and building the program accordingly may substantially speed up its execution, even on a cloud platform. It should be noted, however, that some of the compilers are able to modify the code so it will benefit from the specific hardware it is running on.

There are many hardware books that were intended for hardware engineers and, as such, provide a very detailed explanation of the various aspects of hardware. This book, however, was written especially for IT people, for whom various aspects of hardware engineering are less important. The aim of the book is to provide a brief historic description of the trends in computing solutions that led to the current available infrastructures. The historic perspective is important, since some of the stages repeat themselves when new computerized appliances emerge. For example, when the mobile-telephone revolution started, the hardware attributes of mobile devices followed some of the developments in computer systems, for example, regarding memory and its utilizations. Furthermore, many of the less sophisticated appliances use hardware components and technologies that were used in various stages during the history of computers. For example, a simple alarm system uses an embedded computer, but since its functionality is limited, it usually mimics the architecture of computer systems 20 or 30 years old. Nevertheless, the components resemble the components of modern systems (processor, memory, and input and output devices).

The historic perspective and the various development stages are important due to the emerging trend of being connected “always and everywhere.” The Internet of Things, which aims to connect billions of embedded computing devices and create a global interconnected network, provides an additional aspect to the necessity of understanding hardware and its historic evolution.

THE HUMAN COMPUTER

When trying to relate to computer components and architectures, an amazing resemblance to the human body’s functionality is discovered. The human body, as well as that of any other living creature, is composed of different components, each one with a clearly defined purpose. This modular approach (which, at least at present, is for humans very limited, since only a small fraction of organs can be replaced), is well known in the engineered solutions and machines that surround us. Modern computers provide, of course, the same modularity; however, the first computers were different.

For better understanding the various computer hardware components, we will refer to human processing mechanisms. When considering human thinking, there are several organs involved. The brain, which processes information, is similar to the computer’s processor. The brain controls all activities as well as behavior and feelings. This process is based on previous experiences, whether learned or acquired. Similarly, the processor acts on the data it receives based on the program that was loaded. In this sense, the program loaded into the system provides the rules for processing, similar to the way in which experience provides the human brain with a blueprint for behavior. In this sense, the program developed by software engineers may be considered as instructing or teaching the computer how to react to various events. This is similar to teaching students how to act in a given situation, for example, when engineering a computerized solution.

The human memory is even more interesting in its relation to the computer memory. Many researchers regard the human memory as a hierarchy of three levels of memory:

- A temporary (or sensory) memory, which is used as a buffer that stores data received from the various senses. Every sense has its own buffer, which contains all the data received. This data undergoes some initial processing, during which most of it is classified as irrelevant and is discarded. The information that may be relevant is copied to the working memory for additional processing. An example is the vast amounts of data we as human beings are exposed to. For example, while driving a car, we constantly see the surroundings. These images and sounds are transferred to the visual sensory memory, where most of it is classified as irrelevant, and it disappears. This is done in order to reduce the data overload, and, as a matter of fact, we even do not remember many of the scenes we have seen or sounds we heard. The sensory memory is the mechanism that allows humans (like other living creatures) to be part of the environment and constantly react to important situations. For most creatures, these situations may involve threats or opportunities (e.g., acquiring or hunting for food).
- Short-term memory (or the working memory), which is the memory used for processing data. It receives only the information that was classified as important or relevant. Here, it is analyzed to determine the proper required action. The short-term memory has a relatively fast access time (tens of milliseconds); however, it has a limited capacity. The data in the short-term memory is kept for a short time; the length of this time depends on the flow of additional “important” data that is being received, the person’s situation, and their age. Due to its limited capacity, this memory is affected by environmental disturbances, which may cause the data to disappear; then, a special search process will be required (to remember the last thing we were thinking or talking about). Talking on the phone or text messaging while driving is a good example of a disturbance in short-term memory processing, as is a disturbing noise during an exam.

- Long-term memory, which is used for storing information for a long and even unlimited duration. This memory has a very large capacity and can hold large amounts of data items. The access time to that data is long, and it becomes even slower with age. The retrieval mechanisms are unreliable. Sometimes we may try to remember some fact but cannot. This is not because the specific data item was removed, but because the retrieval process is not adequate. Sometime later, we may recall the missing data, which implies that it was stored in memory and was not lost, simply that the pointer that links the data was not clear. For information to be stored in the long-term memory, it has to be very important, or a constant rehearsal is required.

Relating back to computers, all three types of the human memory are present. Unfortunately, as will be elaborated on in this book, although modern computers use a similar memory hierarchy, it took time before it was designed and developed. The long-term memory in computers refers to the various storage devices, such as disks (hard drives). These devices are used for storing the information for a long duration. Among the storage devices we may find magnetic tapes, which were used in the past but were, almost totally, replaced by better, more reliable technologies and disks (magnetic, optical, and electronic, such as solid-state disks). The short-term memory is the memory (or random access memory [RAM]) and it is the working area. For the processor to run a program, it has to be loaded into memory as well as the data it requires. This is usually a limited capacity memory, although in a modern system it may consist of billions of bytes (or characters); it is significantly smaller, nevertheless, than the storage, which may be several orders of magnitude larger. The third level of the sensory memory is implemented in computers by registers as well as the cache memory. Registers are a small working area inside the processor that is intended for the data currently being used and processed. Cache memory is a fast but limited memory used for the recently used instructions and data. Furthermore, due to its contribution to the system's performance, cache memory is implemented using several levels. The closer the cache is to the processor, the faster and smaller it is.

The last and most trivial components are the various senses that act as our input devices (sensors in embedded and real-time systems) and sense the environment; and the muscles that control movement and speech, which represent output devices (actuators in real time systems).

Regarding behavior, in addition to the behavioral blueprints that regulate our behavior in various situations, sometimes the brain tries to correct errors it thinks were introduced by our inherent limitations. One such famous example is the Kanizsa* triangle.

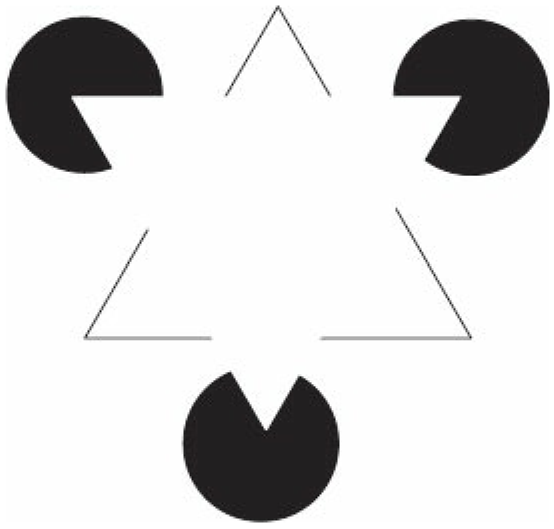

FIGURE 0.2 The Kanizsa triangle.

Most humans will interpret the picture as a white triangle placed on top of three circles. This is an example of a visual illusion, since there is no white triangle but only three circles, each of which has a missing part (similar to the "creature" from the famous Pac-Man game). The erroneous interpretation is due to the brain's attempt to overcome a known limitation. We live in a three-dimensional world; however, drawings, such as Figure 0.2, that are printed on paper cannot convey the three-dimensional experience and are limited in providing just two dimensions. The brain, which is aware of this fact, tries to correct the problem by interpreting the figure as a three-dimensional object. This "recovery" is done automatically without active conscious involvement. Another example of a process that is done automatically among all creatures is the instinctive reflexes, such as moving the hand after we touch a hot surface, or an animal that attacks or runs away when a threat is perceived. These reflexes are managed without the brain's involvement, mainly in order to save time in dealing with a potential dangerous situation. Computers have adopted similar "behavior" using a different approach. One such example is during communications, in which error correction codes (ECC) are used to assess the correctness of the messages sent and received. It is an automatic process in which the system tries to correct mistakes and provide the right interpretation before it gets to the user program (just like the Kanizsa illusions). In cases of error, the controller issues a resend request without even involving the processor. Roughly, this can be viewed as the controller reflex to the erroneous message received. The idea behind this behavior is to relieve the software engineering process of the need to check the integrity of the data received from another component or by communication.

CHAPTER ORGANIZATION

The book was designed to enhance students' understanding regarding the hardware infrastructure used in various software engineering projects. As such, it covers the main hardware components and, in some cases, the logic that the components' development followed. The historic perspective is important because sometimes when new computerized appliances are invented, the old design considerations may surface again. One such example is the memory management used by old mobile telephones before they emerged into fully operational computer systems, as is the case with smart phones. For a better and a gradual understanding, the text in the book is divided into 11 different chapters:

- Chapter 1, "Introduction and Historic Perspective," provides an introduction and a historic perspective, from the initial need for computers, followed by the first developments and additional technological directions, up to modern cloud computing. The historic perspective relates to computers and systems in general. Historic aspects that relate to various components of the system will be elaborated on as part of the appropriate chapter.
- Chapter 2, "Data Representation," describes the computers' data representation. It starts with various numeric systems, followed by the way computers represent and store numbers, and it includes a brief section on computer arithmetic.
- Chapter 3, "Hardware Architecture," provides a brief explanation regarding computer architecture and how this term and its underlying meaning changed over the years.
- Chapter 4, "Central Processing Unit," provides an explanation about the processor (or central processing unit [CPU]). Once again, it includes a brief historic development, since some of the previous implementations are still available and are used by simple appliances, such as calculators. The chapter includes a large section on performance enhancements, which, although

performed by the hardware, resemble the algorithms used by software engineers.

- Chapter 5, "Memory," describes computer memory and discusses its organization, hierarchy, and performance considerations as applied by the operating system, which are important for software engineers.
- Chapter 6, "Cache Memory," describes cache memory, which is an important level in the memory hierarchy responsible for significant performance increase. The chapter concludes with various possible architectures that are the cornerstone of modern and future systems.
- Chapter 7, "Bus," describes the bus system, which is responsible for the movement of data between the various computers' hardware components. Due to the changing definition of buses in modern architectures, various algorithms for ensuring data integrity are discussed as part of the chapter.
- Chapter 8, "Input and Output," provides a brief description of input and output (I/O) components and discusses various methods for performing I/O. Various aspects relevant to software engineering, such as buffers and buffer flushing, are explained.
- Chapter 9, "Storage," describes the various nonvolatile storage devices (the bottom part of the memory hierarchy). For obvious reasons, most of the chapter is dedicated to hard drives and technologies for enhancing performance and reliability. As with previous cases, the methods used are related to hardware (the disk controller), but the algorithms used are implemented using software and are relevant for software engineers.
- Chapter 10, "Additional Architectures," describes additional system architectures, with a special emphasis on virtualization and cloud computing.
- Chapter 11, "Software Architectures," briefly describes the emergence of software-based systems, from the prearchitectural era through some of the architectures that were developed to cope with various market requirements. It concludes with current and future trends.

* Cloud computing is a relatively new computing infrastructure that has the potential to revolutionize the whole industry. Cloud computing became feasible due to several technological developments that have now matured and are capable of providing a coherent computing infrastructure. Cloud computing is based on the success of some of the most well known web-based applications, such Google's search engine and Amazon's virtual store. These and other applications paved the way for Internet-based applications that provide services to a large and extremely dynamic customer population. Based on the experience gathered, cloud computing is based on flexible and on-demand provision of computing servers that reside somewhere in the network. As such, it has the potential to significantly change the whole computing industry and especially information technology (IT) departments within organizations by providing a better service at a fraction of the cost.

† Virtualization is the possibility of executing an application regardless of the physical infrastructure.

‡ A compiler is the software tool that is responsible for translating the high-level programming language instructions into machine instructions, which are the only instructions understood by the hardware. For each programming language, there is a different compiler. Furthermore, usually a compiler will produce the running program that is suitable for a specific hardware. Similarly, the tool that translates assembly-language instructions into machine language is called an assembler.

* A feasibility study is done as part of the system engineering stage in order to assess the feasibility of the required system, which relates to the efforts required, the resources, and the possible implications for the working procedures and infrastructure.

† One of the criteria for deciding about new projects' development is the return on investment (ROI). For each such project, the anticipated benefits are compared to the planned and required resources' costs. The systems with the higher ROI will usually be implemented first.

* Multithreading is a capability that exists in most programming languages and provides the means to define several threads of execution within the same process. These threads share some of the processes' resources but execute independently, thus better utilizing hardware resources (cores).

* Kanizsa Gaetano was an Italian psychologist who published an article about illusory contours, which discussed various visual illusions.