Chapter 1 Introduction
(Revision number 11)

Computers are ubiquitous, from cell phones, to automobiles, to laptops, to desktops, to machines that power search engines such as Google, eBay, and Amazon. Historically, computer architecture is the endeavor that relates to the design of the hardware that is inside each of these instantiations of computing machines. In the early days of computing, there was a clean separation between the hardware design and the software systems. However, a confluence of factors is making this separation impractical and inefficient. From the hardware side, two most significant inter-linked developments are on-chip power dissipation and multi-core processors. For over 40 years, there was unabated increase in processing power as a realization of Intel Co-founder Gordon Moore's prediction back in 1965 that chip density (and indirectly the processing speed) will double roughly every two years. The side effect of this increase in chip density and processing speed is the corresponding increase in power dissipation. This has led in recent times for architects to turn their focus to achieving better utilization of the increased chip density by putting more processors on the same chip, a technique that is referred to as multi-core (each core is an independent processor) in the computing jargon. Simultaneously, software has been growing in complexity as we find computing pervading different aspects of our everyday life. From the software side, application complexity, increased interactivity, real-time response, and the need to tackle parallelism from the get-go rather than as an afterthought are the contributing factors. What these developments from the hardware and software sides mean is that we can no longer afford to treat the other side as a black box. There is a compelling need to train the next generation of *system architects*, who understand the inter-relationship between system software and computer architecture.

The sooner we introduce the inter-relationship to the students the better equipped they will be as computer scientists regardless of their ultimate career pursuits.

1.1 What is Inside a Box?

There is a processor (called the central processing unit or CPU), the memory subsystem, all the various peripherals (such as the keyboard, monitor, mouse, disk, and DVD player), and the network interface that allows you to connect the box to the outside world. Then there is the system software (such as the operating system, compiler, and runtime systems for high-level programming languages) that allows you to do what you want to do at the application level.

Figure 1.1 What's in the box?

1.2 Levels of Abstraction in a Computer System

Consider an application you may be familiar with such as Google Earth (Figure 1.2). You can pretty much navigate over the entire surface of the earth by simply moving your mouse on the earth terrain using the graphical user interface (GUI) provided by Google Earth. You move the mouse over to any specific region of the earth you are interested in, say Mount Everest, and click. Suddenly, you get a 3-D view of the tallest mountain range in the world filling your screen, satellite imagery of the terrain, pictures of the area, etc. What is happening inside the box that gives you this visual experience?

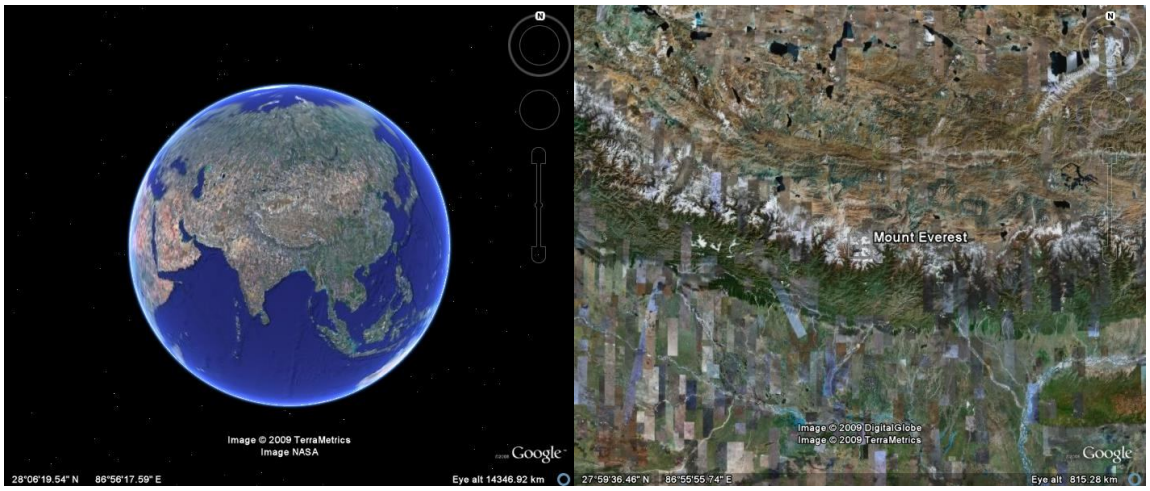

Figure 1.2 Screen shots from Google Earth application

Consider another more complex example, a multi-player video game (Figure 1.3), Halo 3. This is the third edition of the Halo trilogy[1], a futuristic inter-galactic game. In a nutshell, the game revolves around the heroics of a small band of soldiers fighting to save humanity by overcoming the evil powers of a galactic force called the covenant.

Figure 1.3 A Video Game (Halo – 3)[2]

Let us consider the software architecture for developing such an application. One can imagine a (logically) central software component, which we will call a *server* that maintains the state of the game. Similarly, each player is represented by a software component, which we will call a *client*. Since this is a multi-player game, the clients and server are not executing on the same machine but they execute on different machines interconnected by a local area network. It is natural to program such an application in some High Level Language (HLL). We may add some audio/visual content to the video game we are designing. As you can see in Figure 1.4, in addition to our own effort (shown in gray boxes on the right side of the figure), a number of other things have to come together to realize the video game software. The CPU of course does not understand anything other than machine language, so a compiler has to translate the HLL program to the instruction-set understood by the processor so that the programs can be run on the processor hardware.

Now let us look at the processor bottom up (left side of Figure 1.4). At the lowest level of the abstraction hierarchy, there are electrons and holes that make up the semiconductor

[1] Souce: http://www.halo3.com/
[2] Source: http://videogames.techfresh.net/wp-content/uploads/2007/08/h3_1.jpg

substrate. The transistor abstraction brings order to the wild world of electrons and holes. Logic gates are made up of transistors. Combinational and sequential logic elements are realized out of basic logic gates and are then organized into a datapath. A finite state machine controls the datapath to implement the repertoire of instructions in the instruction-set architecture of the processor. Thus, the instruction-set is the meeting point of the software and the hardware. It serves as the abstraction that is needed by the compiler to generate the code that runs on the processor; the software does not care about how the instruction-set is actually implemented by the hardware. Similarly, the hardware implementation does not care what program is going to run on the processor. It simply fulfills the contract of realizing the instruction-set architecture in hardware.

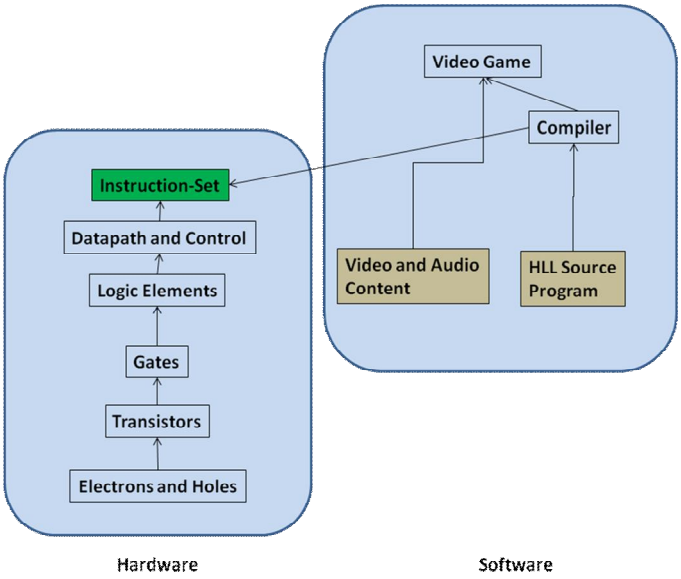

Figure 1.4: Hardware/Software Interface

As can be seen, the successive levels of abstraction have allowed us to control the probabilistic behavior of electrons and holes in the semiconductor substrate from a high-level language program. This is shown pictorially in Figure 1.5.

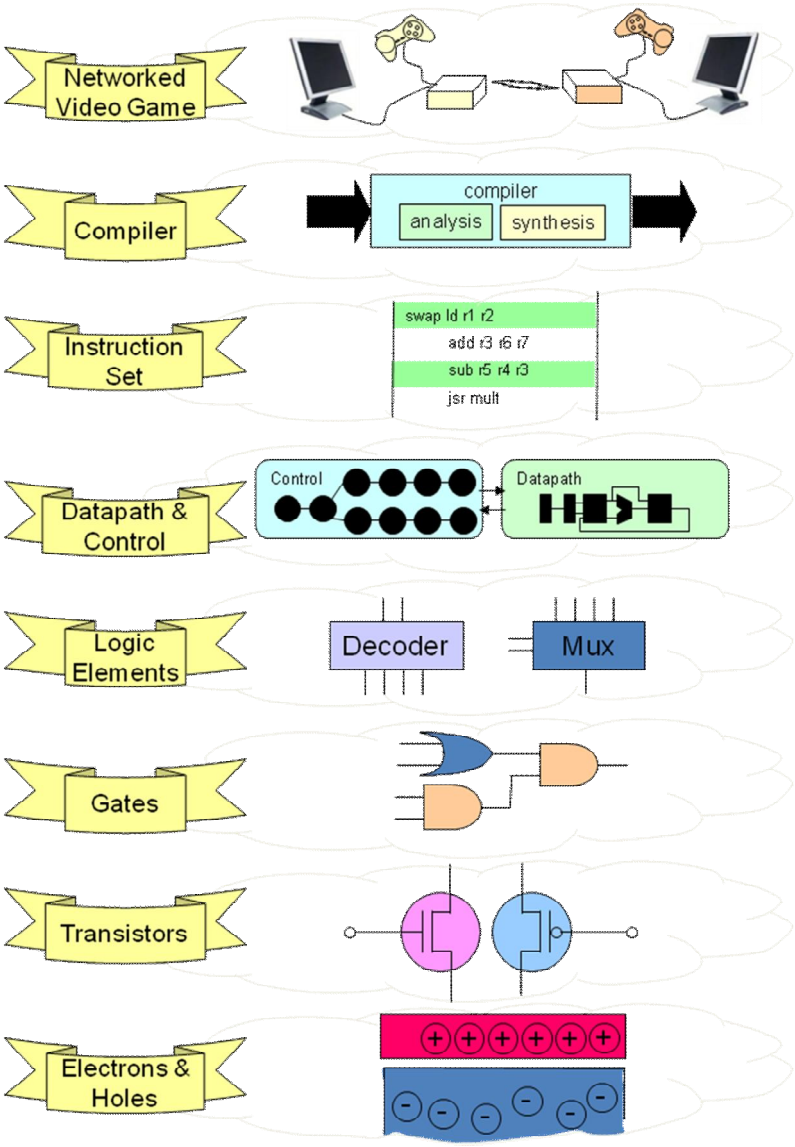

Figure 1.5: From Electrons and Holes to a Multi-player Video Game

1.3 The Role of the Operating System

Where does the operating system figure in the above discussion? It is the resource manager that is responsible for orchestrating the use of the hardware resources for the whole endeavor right from the design of the game to the actual game playing. To see this, let us return to the video game example. We have written the client-server application in high-level language. We may use a simple text editor or a sophisticated program development system such as Visual Studio for the development of our video game. Once developed, we compile the program to the instruction-set of the processor. Each of the text editor, the compiler, etc., is a program that needs to run on the processor. For example, the compiler has to run on the processor taking the HLL program as input and

produce machine code as the output. The operating system makes the processor available for each of these programs to its job. Now let us see what happens when you are actually playing the game.

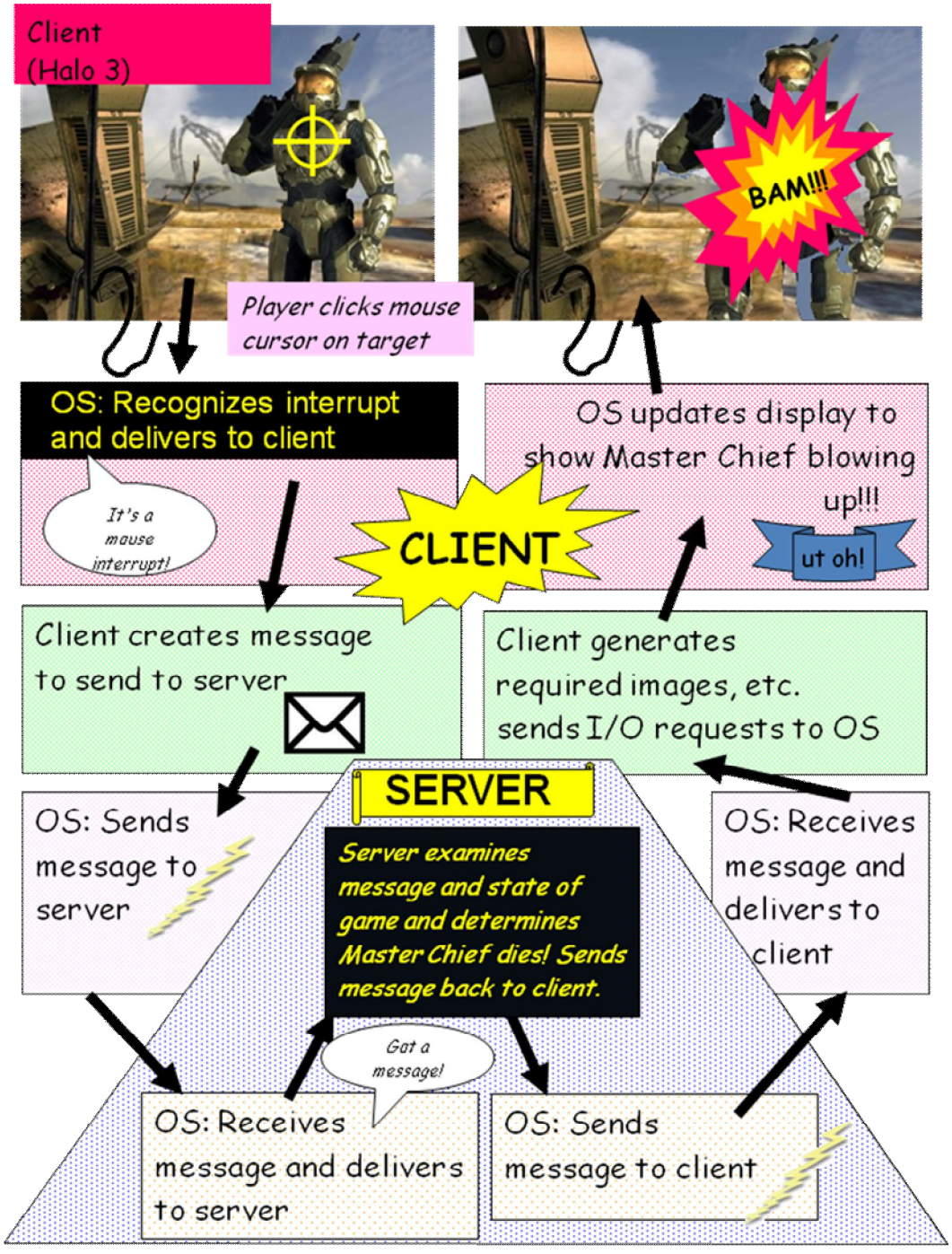

Figure 1.6: Application-Hardware-OS interactions

In the video game example, you click the mouse button that results in the master chief blowing up on your screen and the screens of every one of the players (see Figure 1.6).

What is going on? First, the hardware device controller on your box records your mouse click. The controller then *interrupts* the processor. Remember that the processor is currently running your client program. An interrupt is a hardware mechanism for alerting the processor that something external to the program is happening that requires the processor's attention. It is like a doorbell in a house. Someone has to see who is at the door and what he or she wants. The operating system (which is also a collection of programs) schedules itself to run on the processor so that it can answer this doorbell. The operating system fields this interrupt, recognizes that it is from the mouse intended for the client program, and hence passes it on to the client program. The client program packages this interrupt as a message to the server program through the network. The server program processes this message, updates the state of the game incorporating the new information, and sends messages to all the clients with the updated state. The clients, through their respective operating systems, update their respective displays to show the new world state. As can be seen several hardware resources (processor, memory for the programs and data, mouse, display, network connections) are all being allocated and de-allocated in the course of clicking the mouse button to the point where the display changes. The operating system orchestrates all of these actions.

1.4 What is happening inside the box?

The video game example serves as a teaser for the interactions between applications, operating system, and hardware. To get a good understanding of what is going on inside the box we have to get a good handle on what is happening inside the box with both the system software and the hardware architecture.

Figure 1.7: From PDAs to Supercomputers

First, it is useful to understand that there are several instantiations of computer systems ranging from handheld devices such as cellphones and PDAs, tablet PCs, notebook PCs, desktop computers, parallel machines, cluster computers, to supercomputers, as shown in Figure 1.7.

Interestingly, irrespective of these different manifestations, the organization of the hardware inside a computer system is pretty much the same. There are one or more central processing units (CPUs), memory, and input/output devices. Conduits for connecting these units together are called buses, and the device controllers act as the intermediary between the CPU and the respective devices. The specifics of the computational power, memory capacity, and the number and types of I/O devices may change from one manifestation of the computer system to the next. For example, commensurate with its intended use, a PDA may have very limited I/O capabilities such as a touch-screen display, microphone, and speakers. A high-end supercomputer used for running large-scale scientific applications such as modeling climate changes may employ several hundreds of CPUs, incorporate several Gigabytes of memory, and be connected to an array of disks with storage capacity on the order of several Terabytes. Figure 1.8 shows the organization of the hardware in a typical desktop computer system.

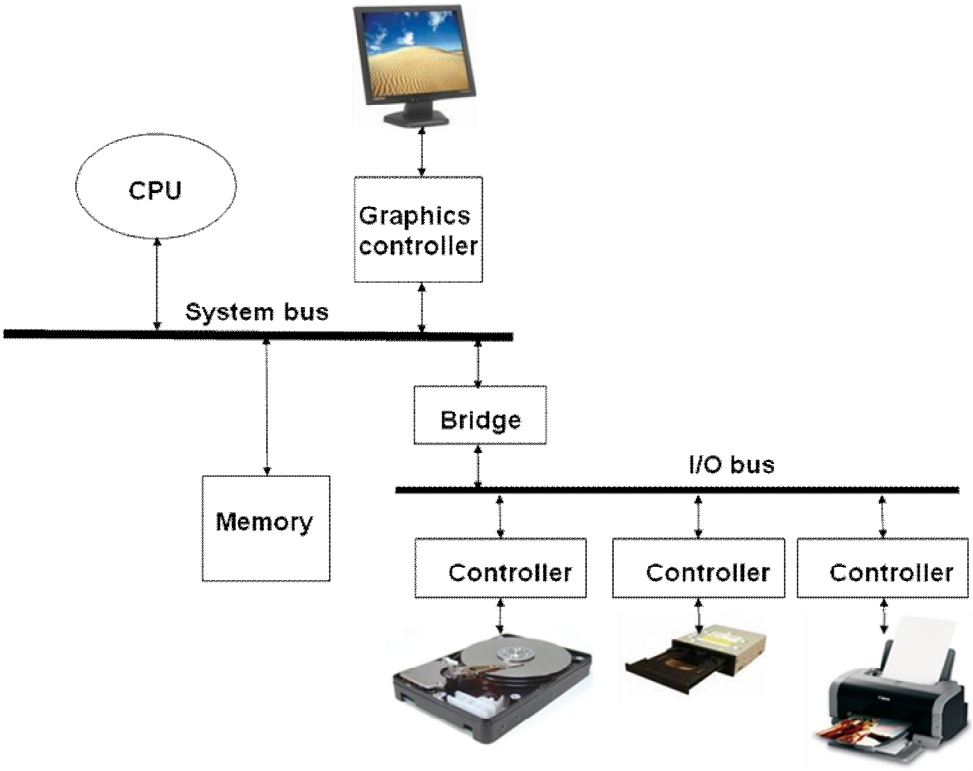

Figure 1.8: Organization of Hardware in a Desktop Computer System

The organization suggests that there is scope for simultaneous activity of the hardware elements (i.e., concurrency). For example, it should be possible for the printer to be printing some document while the hard drive is reading an MP3 file from the disk to play your desired music while you are reading some news story from CNN through your web

browser. The CPU is the brain of the entire system. Everything that happens in the computer system is a result of some program that runs on the CPU. You may observe that simultaneous with your watching CNN on your computer screen the printer may be busy printing some document for you from a document-editing program. Web browser is an application program and so is the document editor. The operating system gives time on the CPU for each of these programs to initiate their activities so that the hardware concurrency that is suggested by the organization is actually realized in practice.

1.4.1 Launching an application on the computer

Let us understand how the various entities shown in Figure 1.8 working with the operating system come together to give you a simple computing experience, say of watching a video on the display device. The box labeled "memory" holds any program that you wish to execute on the CPU. In the absence of any user program, the operating system, which is also a program, is always executing on the CPU, ready for work that the user may want the computer system to carry out. First, with your mouse you may click on an icon on the display device that says "movie player". The movement of the mouse and the mouse click are fielded by the operating system. From the icon clicked, the operating system knows which program the user wishes to execute. All such programs are resident on the hard drive of the computer. The operating system "loads" the executable image of the movie player program into the memory and transfers control of the CPU to start executing this program. The execution of the movie player program results in opening a graphics window on your display and asks the user to specify the specific movie file that the user wishes to watch. You would then use a keyboard perhaps to type the name of the file including the drive on which the file may be found (let us say, the DVD drive). The program then opens the file on the DVD drive and plays it, and you are now ready to watch your favorite movie on the display device. The operating system is involved in every step of the way to deliver you the movie watching experience including: (a) updating the graphics display, (b) capturing the user inputs on the keyboard and delivering them to the movie player program, and (c) moving the data from a storage device such as the DVD drive to memory. The actual mechanics of data movement from/to the I/O devices to/from the memory and/or the CPU may depend on the speed characteristics of the devices. We will discuss these aspects in much more detail in the chapter on I/O subsystem (Chapter 10). The I/O bus and system bus shown in Figure 1.8 serve as the conduits using which data is moved between the various sources and destinations depicted by the hardware elements. Just as highways and surface streets may have different speed limits, the buses may have different speed characteristics for transporting data. The box labeled "bridge" in Figure 1.8 serves to smooth the speed differences between the different conduits in use inside a computer system organization.

1.5 Evolution of Computer Hardware

With the ubiquity of computing in our everyday living, it is difficult to imagine a time when computers were not so commonplace. Yet not so long ago, the computational

power in your current day laptop that you may have purchased for less than $1000 cost over a million dollars and occupied a room the size of a large dance floor with elaborate cooling and raised floors for the cabling.

In the early 40's, ENIAC – Electronic Numerical Integrator and Computer – was built at the University of Pennsylvania and is largely considered as the very first programmable electronic digital computer (Figure 1.9).

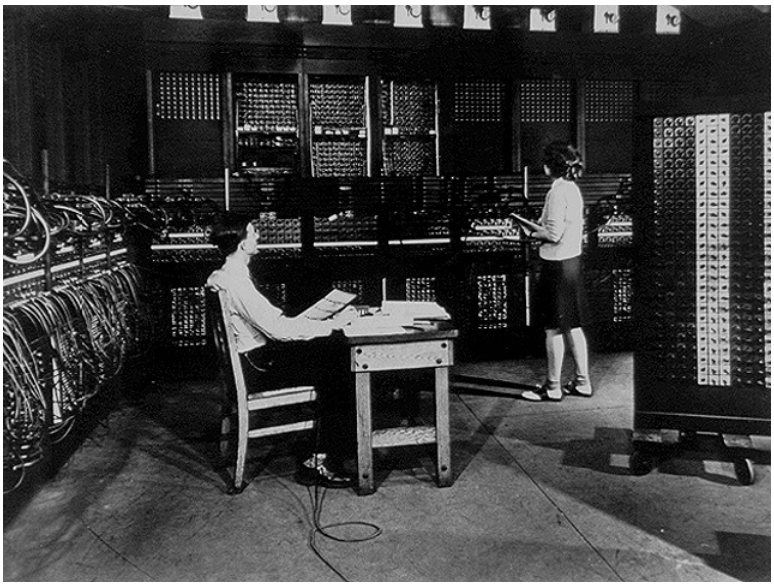

Figure 1.9: ENIAC, first electronic digital computer

Built with over 18,000 vacuum tubes, 1000 bits of random access memory implemented using tiny magnetic ferrite cores, and consuming around 170 Kilowatts of electrical power, the ENIAC had a processing power that is roughly equal to what you may find in a modern day musical greeting card! This is how far computing technology has progressed in a little over 6 decades since the time of ENIAC.

One can attribute the rapid progress of computer hardware due to the ingenuity of scientists and engineers in complementary yet diverse fields including physics, chemistry, electrical engineering, mathematics, and computer science. Of course, the most visible technological revolution that has spurred the rapid advance of the computer industry is in the semiconductor revolution. Digital computers had their modest beginnings using vacuum tubes and magnetic core memory in the 1940's. With the invention of a switching device called the "transistor" at Bell Labs in 1947, the semiconductor revolution began to take shape. The era of digital computers built with discrete transistors gave way to the integration of several such transistors on to a single piece of silicon. Microchip – single chip microprocessors based on CMOS transistors employing Very Large Scale Integration (VLSI) – introduced in the late 80's and early 90's, was perhaps the tipping point in the computer hardware revolution. Today every computing device from cell phones to supercomputers uses the microchip as the basic building

block, and semiconductor memory (typically several hundreds of megabytes or even gigabytes) has completely replaced magnetic core memory.

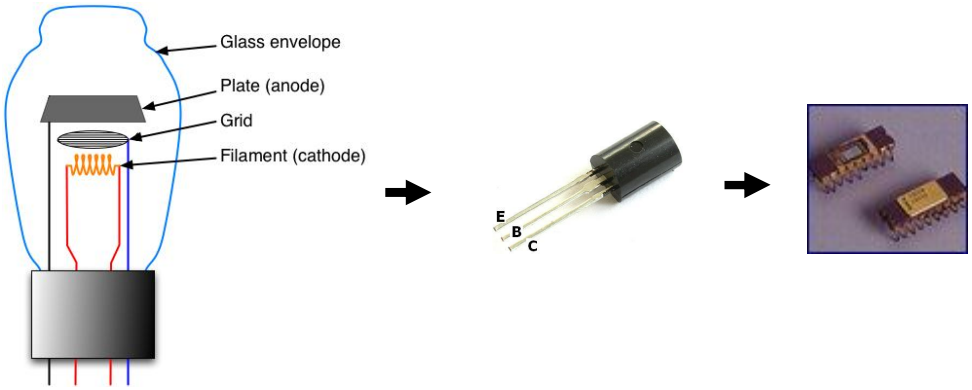

Figure 1.10: From Vacuum Tube to Transistor to Microchip

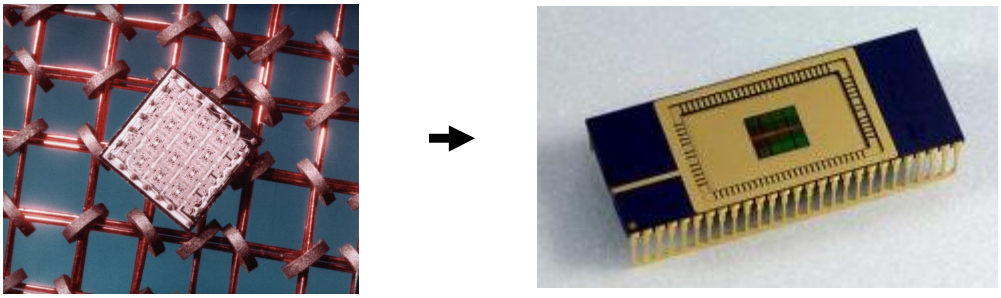

Figure 1.11: From Magnetic Core Memory to Semiconductor Memory

1.6 Evolution of Operating Systems

Operating system evolution dovetails the evolution of the processor and the computer systems built around the processor. Operating systems had their modest beginnings in the 50's with offerings such as FMS (Fortran Monitoring System) and IBSYS (IBM 7094 operating system). Today of course, operating systems span the domain of the computing devices shown in Figure 1.7. Microsoft Windows and Mac OS dominate the PC market. Linux has a strong foothold in the enterprise-computing arena. Embedded devices such as cell phones and PDAs have their own unique requirements and there are specialized operating systems that cater to those needs. Examples of specialized embedded operating systems include Symbian OS and Blackberry OS. Many embedded operating systems are derivatives of desktop operating systems. Examples of such derived operating systems include Mac iPhone OS and Windows CE.

One can trace the evolution of operating systems with the class of computer systems that they were designed to support and the increasing expectations of the user community. *Batch-oriented* operating systems supported the mainframe class of computers. *Multiprogrammed* operating systems emerged to better utilize the available hardware resources in mainframes and mini-computer systems. *Timeshared* operating systems evolved as a response to the user community's desire to get interactive response from the computer systems. With the advent of the personal computers and the graphical user interface (GUI), operating systems for the PCs integrated the GUI into the operating system as exemplified by Microsoft Windows 95 and its successors.

Ultimately, operating systems are concerned with providing computational resources, namely, processing, memory, storage, and other I/O devices for end users. A trend that has emerged in recent times is to make the computing resources available via the Internet. This trend had its beginnings with *grid* computing, which was purely a research endeavor to make high-performance computing resources available across administrative boundaries via the Internet. The term "grid computing" – derived from the way electrical power is distributed and made available ubiquitously using the power grid – is meant to signify that computing power should be made available ubiquitously just as electrical power. Today, several vendors such as Amazon and Microsoft are offering computational resources (processing power and storage) via the web. *Cloud computing* is the industry buzzword associated with this new way of providing computational resources to end users.

1.7 Roadmap of the rest of the book

These are exciting times for the computer system aficionado. This brief introduction unambiguously shows the intimate relationship between computer hardware and system software. Correspondingly, the rest of the book presents the hardware and software issues pertaining to processor, memory, I/O, parallel systems, and networking in an integrated fashion.

Part I: Processor
Chapter 2-5: processor design and hardware issues
Chapter 6: processor-scheduling issues that are addressed by an operating system

Part II: Memory subsystem
Chapter 7 and 8: Memory management issues addressed by an operating system with corresponding architectural assists
Chapter 9: Memory hierarchy, specifically dealing with processor caches

Part III: I/O subsystem
Chapter 10: I/O issues in general as it pertains to interfacing to the processor, with specific emphasis on disk subsystem
Chapter 11: File system design and implementation, a key component of an operating system to manage persistent storage

Part IV: Parallel system
Chapter 12: Programming, system software, and hardware issues as they pertain to parallel processors

Part V: Networking
Chapter 13: Operating system issues encountered in the design of the network protocol stack and the supporting hardware issues

1.8 Review Questions

1. Consider Google Earth application. You launch the application, move the mouse on the earth's surface, click on Mount Everest to see an up-close view of the mountain range. Identify the interactions in layman's terms between the operating system and the hardware during the above sequence of actions.

2. How does a high-level language influence the processor architecture?

3. Answer True or False with justification: "The compiler writer is intimately aware of the details of the processor implementation."

4. Explain the levels of abstractions found inside the computer from the silicon substrate to a complex multi-player video game.

5. Answer True or False with justification: "The internal hardware organization of a computer system varies dramatically depending on the specifics of the system."

6. What is the role of a "bridge" between computer buses as shown in Figure 1.8?

7. What is the role of a "controller" in Figure 1.8?

8. Using the Internet, research and explain 5 major milestones in the evolution of computer hardware.

9. Using the Internet, research and explain 5 major milestones in the evolution of the operating system.

10. Compare and contrast grid computing and the power grid. Explain how the analogy makes sense. Also, explain how the analogy breaks down.

11. Match the left and right hand sides.

Unix operating system	Torvalds
Microchip	Bardeen, Brattain, and Shockley
FORTRAN programming language	Kilby and Noyce
C programming language	De Forest
Transistor	Lovelace
World's first programmer	Thompson and Ritchie
World's first computing machine	Mauchley and Eckert
Vacuum Tube	Backus
ENIAC	Ritchie
Linux operating system	Babbage