ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ, ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Chapter 1: THE HISTORY OF COMPUTING

Contents

Topic Goals

In this section you will see the evolution, over time, of the computer, as we know it today. You will also see how work-related needs have helped to shape the development of computer hardware and software products.

Try to pay attention to the major events, people, technologies, organizations, and time frames while working your way through this topic. It is not necessarily important to memorize the actual date of when an event takes place; but rather focus on the context and the world events surrounding it.

After completing this topic, you should be able to:

- Give examples of the important events in computer history as related to the early years prior to 1930.
- Give examples of the important events in computer history as related to the period of the late 1930s to 1959.
- Give examples of the important events in computer history as related to the period of 1964 to 2000.
- Discuss the importance of the beginnings of the information "Super Highway."
- Analyze the lessons learned from computer history.

The Early Years

As early as 500 BC, mankind was trying to make the work of processing information easier with the advent of the handheld abacus for counting numbers. The history of mankind is replete with attempts to develop machines capable of managing information more efficiently and increasing productivity in the workplace.

Automatic Calculating Machines

In the 1700s, the first automatic calculating machines were invented, initially only able to add and subtract, yet thirty years later capable of multiplication and division. It took another 200 years before the workplace was seriously revolutionized: a Frenchman invented a punch-card loom attachment used in the silk-weaving industry. This automated the process of entering input data for weaving, saving incredible amounts of time. The basic principles used in this loom attachment apply to many later computing devices.

Figure 1.1: *An example of automatic calculating machines*

The Roles of Charles Babbage and Augusta Ada

It was in the late 1800s that a British man named Charles Babbage (1791-1871), fondly termed the "father of computing," invented two computation machines. The first could solve equations, but readily made mistakes. He later conceived of a model that embodied the five key parts of the computer as we know it today: an input device, a storage place, a processor, a control unit, and an output device.

His colleague, Augusta Ada (1815-1852), Countess of Lovelace and daughter of the English poet Lord Byron, was instrumental in the machine's design. She helped Charles make revisions when necessary, obtained money from the British government, and by writing a description of the Analytical Engine, communicated specifics to the public and people of importance. Her understanding of the machine allowed her to create the instruction routines which were fed into the computer using a set of cards capable of carrying out specific instructions. She is often called the first computer programmer.

Notice: In the 1980s, the U.S. Defense Department named a programming language "ADA" in Augusta Ada's honor. However, women have been typically under-represented in the computer and information science fields.

According to a report published by the American Association of University Women Educational Foundation, violent electronic games and dull programming classes turn off more and more girls to the computer culture.

Today only about 20 percent of all IT professionals are women. Since 1984, the number of women receiving Bachelor's degrees in computer science has dropped by about 10 percent. Computer science is the only field in which women's participation has actually decreased over time.

The Electric Computer

Perhaps the most tangible development for us is that of the electric, rather than manual computer. First developed and introduced by Herman Hollerith, a statistician with the U.S. Census, his electromechanical system computed the 1890 census (used to survey the demography of the U.S. population) in a miraculous six weeks compared to the laborious 7 years it took just a decade before!

His excitement for the commercial applications of an electric machine motivated him to form his own company in 1896. In 1924 his company, the Tabulating Machine Company, merged with two others to form International Business Machines, or IBM. The beginning of a commitment to computer technology from a powerful organization with enormous resources was about to begin.

Formation of a technological base: 1930-1959

By 1965, computers and the computer industry were picking up steam. Over the next thirty years, the destiny of many individuals, technologies, and companies would be set. Some would win, some would lose, and still others would muddle along just barely surviving in the high-speed race for dominance in the industry.

The Development of ENIAC

During World War II, U.S. military officials were interested in financing the development of a machine that could quickly calculate trajectories for artillery and missiles. At the University of Pennsylvania, a course in the design of electromechanical instruments was in full swing. Dr. John Mauchly was approached and asked to build a machine that could eliminate the time it took for the ballistics computations to be done manually by the young women with mathematics degrees and the trained recruits from the U.S. Army's Women's Auxiliary Corps. Mauchly referred to the work of Dr. John V. Atanasoff, professor of physics at Iowa State University, and his assistant, Clifford Berry, who had successfully built the first electronic digital computer in the late 1930s. It was named the ABC, for Atanasoff-Berry computer.

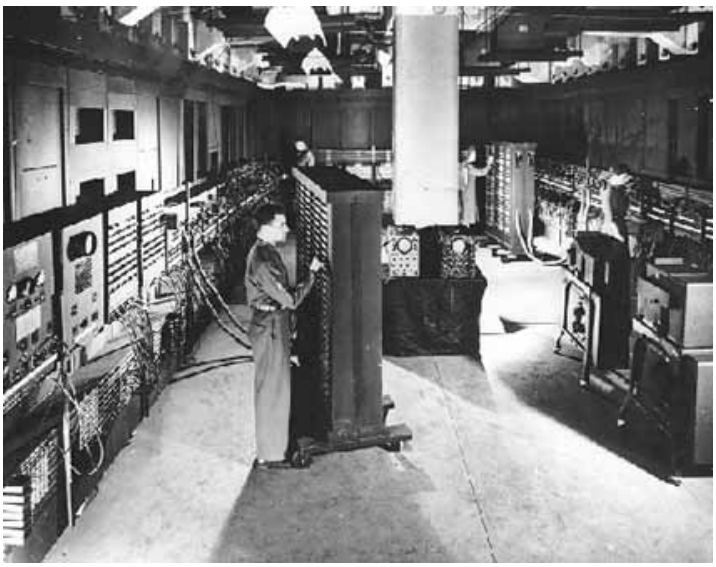


Figure 1.2: *ENIAC*

It was the ABC that Mauchly and his student, J. Presper Eckert, used as the prototype for creating another computer, the ENIAC (short for Electronic Numerical Integrator and Calculator). The ENIAC could add 5,000 numbers or do 14 ten-digit multiplications in a second, fast in those days, a turtle's pace by today's standards. Although finished too late to assist in the calculation of missile trajectories, its first task was to perform top-secret studies to determine if a hydrogen bomb was possible in late 1945. This work led to the development and utilization of the first atomic bomb. However, the ENIAC's primary noteworthiness is that it was the technological predecessor to the Universal Automatic Computer (UNIVAC).

The ENIAC is considered by many to be the first general-purpose digital electronic computer and the primary forerunner of today's modern computers. The ENIAC, as with almost all digital computers, relied on a binary system to represent values such as words and numbers. The binary system is a fundamental building block of information technology. It provided a foundation for the development of ENIAC and subsequent computer systems. In the most basic sense, computers relied on electronic circuits to operate. There are only two possible states in any electronic circuit: On or Off. This is the basis of computerized information processing using the binary system. George Boole, an English mathematician who lived in the nineteenth century, is credited with developing a discipline called Boolean algebra that is the basis for binary systems. According to Boolean algebra, all objects may be divided into categories, each with a unique property. Each category can be described by the presence or absence of that property. For example, Boolean algebra has been applied in the design of computer circuits by making use of the binary principle. The electrical circuits in a computer are either on or off, representing the presence or absence of a property.

UNIVAC and New Languages

Mauchly and Eckert formed their own company and, using technology identical to the ENIAC, developed the UNIVAC computer. No one ever guessed a market existed for computers beyond complicated scientific and mathematical problems. But, UNIVAC gained wide publicity in 1952, when it was used to correctly predict the outcome of the U.S. Presidential election! Still, in 1955, only approximately 150 computers were installed commercially.

Although the UNIVAC sold well commercially, there were huge disadvantages compared to the computers we use today: thousands of vacuum tubes (tubes about the size of light bulbs) were the primary internal components. They generated so much heat that burnout frequently occurred, making the computer unreliable. Also, the use of a numeric language made programming much more cumbersome and time-consuming.

However, it wasn't long before higher level, symbolic programming languages, which use characters and text, were introduced: FORTRAN, in 1953 by IBM, COBOL in 1959 and BASIC, in 1963 at Dartmouth. These languages made programming the computer much easier. It is amazing, but many of these languages are still being used today for millions of applications. Each of these languages has provided the foundation for many of today's most popular software applications.

The IBM 650

IBM entered the commercial computer market in 1953, relatively late, but still took the lead with their Model 650 computers. The 650 only used about 500 vacuum tubes in its central processing unit, and was therefore much smaller and more dependable than the UNIVAC. It also supported a variety of programs and had much better error-checking ability. By 1960, IBM's 650 computer had become the most popular and widely used mid-sized computer in America. And because IBM developed a marketing strategy that discounted rentals of the machine to universities (the regular rental was $3,000 per month), the 650 became the computer around which the new academic discipline of computer science evolved.

The IBM 650 is considered by some to be the earliest ancestor to the personal computer because it was cheaper ($200,000-$400,000 compared to $1 Million for UNIVAC), smaller (it fit into a single room), and more user-friendly than previous systems.

The Invention of Transistors

In 1947, scientists at Bell Laboratories made the next scientific advance that influenced computer technology: they invented the transistor, a small device able to transfer electronic signals across a resistor. The group received the Nobel Prize in 1956 for their work.

The transistor revolutionized the entire computing industry because it was so much smaller, faster, more reliable, and less energy-intensive than vacuum tubes. By the end of the decade, the computer industry utilized more than one-sixth of all transistors manufactured. This helped to lay the foundation for a thriving computer industry.

The Role of the Integrated Circuit

The development of integrated circuits (IC) began in 1959 with Jack St. Clair Kilby's patent. Made of silicon, found in typical beach sand, rocks and clay, the IC acts as a "semiconductor" by conducting electrical current through the silicon when "doped" with chemical impurities. Semiconductor technology quickly replaced transistors as the main internal component in computers due to their exceptional reliability, compact size, and the fact that they could be mass-produced cost-effectively.

By the mid 1960s, the public was getting used to seeing computers outperform human abilities in some areas: predicting a presidential election and playing an almost perfect chess game. But, computers were still far beyond the average person's personal experience and budget, and primarily viewed as very complex and intimidating.

Notice: Silicon Valley surrounds the area of Stanford, California, home to Stanford University. It can be said that the valley contains one of the biggest collections of innovative industries that exist anywhere in the world.

Semiconductor chips, made of silicon, are the principal product of the local high-tech industries. Because of this, in the early days, the term "Silicon Valley" was used occasionally by easterners who would mention making a trip to the area. In 1971, it was popularized in a series of articles, "Silicon Valley USA," written by Don Hoefler for Electronic News. It was probably the first time the term was used in print.

The Personal Computer Story

By 1965, computers and the computer industry were picking up steam. Over the next thirty years, the destiny of many individuals, technologies, and companies would be set. Some would win, some would lose, and still others would muddle along just barely surviving in the high-speed race for dominance in the industry.

IBM's Continued Effort

By 1964, IBM had set the standard for business users with the System/360 family of computers. They cost IBM over $5 billion to bring to market, but customer demand far exceeded expectations: over 1,000 a month were being ordered.

Another notable marketing effort by IBM was to unbundle the software (the programs or instructions that tell the computer how to perform tasks) from their hardware (the machinery and equipment in a computer system), leading to the creation of today's bustling software industry. Unbundling software from hardware allowed new software ideas and companies to form, providing a variety of new applications for computer users as time passed.

The Invention of the Microprocessor

The next major technological breakthrough came in 1971 at Intel Corporation with the invention of the microprocessor (the miniaturized integrated circuit in a computer that manipulates data). When the general purpose, processor-on-a-chip became commercially available, it was eventually used in virtually every home and business machine: cars, copy machines, televisions, digital watches, bread-making machines, and of course, personal computers.

Thanks to the emergence of the microprocessor, computers became hundreds and thousands of times smaller than those initially developed in the 1950s. The invention of the microprocessor continued the miniaturization of computer technology that eventually would lead to the all-pervasive utilization of computers in daily life.

Figure 1.3: *Microprocessor*

Microsoft and Apple are Formed

In 1975 Bill Gates and Paul Allen formed Microsoft Corporation, which would later develop two of the world's leading software operating systems: DOS and Windows. This eventually led to Mr. Gates becoming the wealthiest, and one of the most powerful business leaders in the world.

Just two years later, with only $1300 (from the sales of an old Volkswagen bus and a used Hewlett-Packard programmable calculator) and a garage for an office, two teenagers developed the first personal computer (PC) specifically for home use, called the Apple I. It boasted an easy-to-use keyboard and screen. Although the Apple I wasn't a commercial success, its developers, Steve Jobs and Steve Wozniak, were. After forming Apple Computer and introducing the Apple II PC, they were immediately and wildly successful, forging the way in personal computer innovations. Apple computers were some of the first computers to introduce the easy-to-use typewriter keyboard, thus replacing the complicated front panels consisting of switches and lights.

Xerox's Innovations

An early version of the PC was actually invented by researchers at Xerox's Palo Alto Research Center (PARC) in northern California. At one point, Xerox executives considered the feasibility of marketing the PARC personal computer. Unfortunately, PARC scientists were unable to adequately demonstrate the future profitability of the product and it was eventually relegated to internal use. About this same time, PARC gave a tour to Steve Jobs who saw the PARC personal computer and realized its potential. Jobs was amazed at the graphical user interface (GUI) and its unique input device, commonly called the mouse.

As the founder of Apple Computer, Jobs leveraged this experience at Xerox and eventually used the PARC technology to create the Lisa PC, which was later succeeded by the Macintosh. Both of these machines successfully introduced the GUI and mouse to the general public. The Macintosh computer, unveiled in 1984, was a big hit in the marketplace and is still popular today in education and in desktop publishing.

The Fight for Operating Systems

The 1980s saw rapid developments in both computer hardware and software technology. Seeing the popularity of Apple's personal computer line, IBM, the Goliath of the computer industry, had launched a crash program to produce its own personal computer. IBM was on the verge of a breakthrough, but it needed one more thing, an operating system.

In a move that would ripple down through computer history for decades to come, IBM turned to the not yet rich Harvard dropout Bill Gates and his fledgling Microsoft company for a solution. In a shrewd business move, Gates hurriedly acquired the rights from a third party to an operating system we now know as DOS. He turned around and licensed the rights to IBM. This key event serves as the foundation on which Microsoft was built. DOS was leased, not sold, to IBM and later to many other hardware manufacturers.

Proliferation of DOS

IBM introduced its first PC in 1981 and captured the top market share in just a year and a half. Although they didn't initially offer the user the ease of a graphical interface (a method of presenting information on the computer screen that uses images, icons, menus, and windows) like Apple Computer, the DOS operating system they used was somewhat intuitive. Also, by offering their hardware schematics and software listings to other companies, new products were quickly developed to run on their PC. This accelerated the demand for their machine over Apple's. Also, IBM's PC was built from nonproprietary parts.

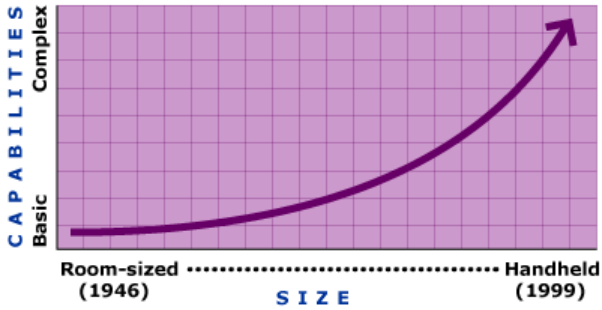
Companies like Dell, Compaq, and Gateway quickly sprang up to build IBM PC clones at a lesser cost, and new software companies formed to offer business and personal programs to support the wide acceptance of IBM's PC. As companies producing IBM clones began to proliferate, they too licensed the rights to Microsoft's DOS operating system. In every case, Microsoft held the rights to DOS, and it became the most widely used operating system in the world.

Software Innovations

In the software industry, while other word processing packages were on the market, such as Word Star and Microsoft Word, WordPerfect's software became the biggest seller in 1982. Word processing software can easily edit and format a document, perform spelling checks, and manage multiple files. Meanwhile, a year later, Lotus Development Corporation developed an electronic spreadsheet program (software that allows the user to enter data and formulas into rows and columns to process numerical and text data efficiently), Lotus 1-2-3, which also became a top seller.

In 1984, Steve Jobs and Steve Wozniak of Apple Computer introduced the Macintosh computer. Perhaps the most notable feature of this computer was the Macintosh operating system. This operating system ran only on Apple Macintosh computers or on Mac clones (formerly produced by Motorola and a few other manufacturers). More importantly, this operating system set the standard for the easy-to-use, icon-based graphical user interfaces. The Macintosh computer quickly earned a reputation as a user-friendly computer system. One explanation for this monumental achievement by Apple Computer was that Apple designed its hardware and software together from the start.

Decreasing Size of Computers



As computer components became smaller so, too, did the size of computers. Decades earlier, computers were huge systems taking up temperature-controlled rooms of their own. Over time it became possible to work with computers that were portable, fit on your desk, used on your lap, and small enough in size to carry in a briefcase. Today, computers that fit in your palm are used for such things as keeping a business calendar and important telephone listings, to name just a few.

Changing Landscapes in the 1980s

In 1985, three significant developments occurred in the rapidly growing computer field: Microsoft introduced Windows, their first software with a graphical user interface for the PC, and Intel introduced the 80386 microprocessor chip, the fastest chip on the market. At this point, the Intel microprocessor was the internal component of choice for most PCs, making the company hugely successful. It was also in this same year that Steve Jobs was ousted and booted out from his own company, Apple Computer. Apple's company entered a long period of turmoil and decline from which it almost didn't recover.

Some companies realized that partnering in such a fast-growing industry could prove beneficial. In 1987 Microsoft and IBM formed an alliance to offer the OS/2 operating system (the underlying system that manages the basic operations of a computer) for the PC. After 10 years, the relationship dissolved because each organization was promoting its own operating system: Microsoft's Windows and IBM's OS/2. Due to a successful marketing effort by Microsoft and the user-friendly attributes of its operating system, Windows became the operating system of choice for most PC users.

Multimedia Begins

Other exciting technological innovations, such as multimedia hardware and software, became available in 1993. These tools began to be used for creating presentations, courses/tutorials, and games that could utilize sound, video, and animation. Having this technology on hand meant one didn't have to go much beyond their own capabilities to learn from an interactive multimedia tutorial, develop a corporate marketing presentation, or play a game of chess online.

The Evolution of Windows

Windows 95, a successor to the Window's 3.x operating system, was released in August 1995. This version of Windows was a true operating system. It did not require a separate DOS program like the older versions of Windows. Its graphical user interface, once just a shell riding on top of the DOS program, was integrated into the operating system. It was also capable of running applications written for DOS, Windows 3.x, as well as those created for Windows 95.

Windows 95 supported e-mail, fax transmission, and multimedia programs. It provided improved menus (so a user could quickly see what files were stored in the computer), the ability to use long file names (up to 256 characters in length), and plug and play capability. Plug and play refers to the ability to add new hardware and/or software to a computer without having to perform complicated technical procedures. Microsoft has continued to release new versions of its popular operating system software including Windows 98, Windows CE (a streamlined version for handheld devices), and Windows NT (a multitasking, multi-user network operating system).

The Continued Story of Microsoft and Apple

During the nineties, the Apple computer story continued to unfold. In 1997, Steve Jobs was brought back as part of a buyout of NeXT Computer to manage the company from which he was exiled 12 years earlier. Soon thereafter, he initiated a relationship with Microsoft. Microsoft eventually became a part owner of Apple, buying $150 million in Apple stock. In return, Microsoft agreed to settle claims that it had leveraged vital Apple patents to develop its own software and develop more applications for the popular Macintosh.

Today, both seem to be thriving. With its introduction of the iMac computer (a user-friendly computer for the Internet generation), Apple began to regain market share and popularity. And Microsoft continues to manufacture the operating system of choice for most users. The coming years may bring major changes to the Microsoft Corporation, however, as the United States Department of Justice has alleged monopoly violations and is threatening a breakup of the company.

The Advent of the Information Super Highway

The ideas of "connectivity" and "interactivity" between computers make up one of the biggest revolutions in the entire history of computers we've covered thus far. Connectivity means that more than one computer is connected by a modem, network, or communications line. Interactivity refers to a back-and-forth dialogue between the users and computers. These two concepts serve as a backdrop to one of the greatest events of modern history, the birth and development of the Internet and World Wide Web.

Networked Computers

In the Cold War atmosphere of the 1950s, the United States Department of Defense worried about the effects of a nuclear attack on its own communication and computing capabilities. To address this concern, the Department of Defense worked with the RAND Corporation to set up a network of several computers that were geographically dispersed so that no single computer would be in charge. That way, if one, or a subset of computers, were destroyed, there would still be several others remaining that were operational.

The messages sent back and forth between the computers in this network were divided up into packets of data. Each packet was labeled with a destination address, and would be sent on its way through the network to eventually be reconstructed into the original message at its destination point.

TCP/IP and the Birth of the Internet

The software that manages the packets of information in a computer network is called Transmission Control Protocol/Internet Protocol (TCP/IP). This software has become the universal standard for information exchange on the Internet. TCP creates the packets and reassembles them into the original message. IP handles packet addressing and ensures that they are transmitted across multiple networks and computers to the correct destination.

One of the major advantages of this mode of communication is that each packet could take its own unique route as long it arrived at the appropriate destination. In time of war, if some routes were cut off or destroyed, the packets could travel along alternative routes. In summary, a packet can travel along a variety of paths to reach its destination. This unique system of connected computers and information pathways was called ARPAnet, an acronym that stands for Advanced Research Projects Agency Network. Today we know this as the Internet.

The Invention of the World Wide Web

The history of the Internet is mercifully short when compared to the evolution of other technologies in history. It slumbered for approximately 25 years before the general public even knew it existed. Dr. Tim Berners-Lee, a physicist working in Geneva, Switzerland, first came up with the idea of the Internet, or a "web" of highways of information between computer systems that were geographically dispersed. He figured out that his work would be easier if he and his colleagues around the world could easily link to one another's computers. The network he envisioned looked like a spider web, and hence the name Web was invented. Dr. Berners-Lee is credited with giving birth to the World Wide Web (WWW).

Marc Andreessen, only a student in 1993, invented the first web browser, able to search and look at the information contained on the computers connected to Internet. The browser utilized a graphical user interface so that users could easily click on and see pictures and text. The first such Web browser was named Mosaic.

Currently, Netscape Navigator and Microsoft Internet Explorer are the two leading Web browsers on the market. They are not the only two, but have become almost standard issue for any Web user.

Why the Internet Grew

The swift, successful growth of the Internet can be attributed to four major factors:

1) adoption of TCP/IP as a universal standard for sharing and distributing information
2) the ability to link from computer to computer or from site to site on the WWW (with just the click of a mouse)
3) the ease of use of the browser and its graphical user interface
4) the increasing popularity and growth of personal computers and local area networks that can be connected to the Internet.

How the Internet Has Changed Us (So Far...)

Linking computers together can provide a variety of benefits, one of which is to send and receive electronic mail or E-mail. E-mail is a software program that allows users to send messages and images to each other over their computers and view them onscreen. Another benefit is that people who own PCs can telecommute. Rather than using traditional transportation to get to work, having a PC makes it possible to work at home and communicate with the office via the computer.

Additionally, the information superhighway allows those using it to share data for the purposes of research, education, and decision making. These days, more and more business transactions are being conducted online using the Internet. Many new businesses have sprung up that sell their products and services online. Needless to say, the Internet has become a boon to many economies around the globe. These incredible developments have opened up the transfer of information and interaction between people and organizations all over the world.

Notice: After reading the facts below, think about whether we are becoming more connected to other people because of the Internet, or whether we are becoming more isolated.

- The Internet is creeping up on the ubiquity of television. 25% of prime TV viewing time (7:30 p.m. to 9:00 p.m.) has been replaced by Internet surfing. (Nielsen)
- 70% of teenagers go online to read and write email, and 50% typically go to a chat room during an Internet session. (Jupiter)
- Teens are increasingly turning to their computers to keep up with their social life as the number of teens meeting friends at parties, nightclubs and sporting events is decreasing. (Nielsen)
- The number of females (age 13-30) online is 9.9 million, and the total number of males (age 13-30) online is 11.8 million. (Media Metrix)

(Source: March 2000)

Lessons learned from Computer's History

Computer science history is both evolutionary and revolutionary. We should pause briefly here to consider a few of the lessons learned on this rapid journey into the information age.

Faster, Smaller, Cheaper

Much of what we've seen in hardware technology has been the evolution of machines to become faster, smaller, and less expensive, such as the progression from vacuum tubes to microprocessors. The transition has affected the size of computers immensely over time. But, some ideas such as certain software programs and the Internet have been truly revolutionary for all mankind.

Software programs have, in many cases, made work easier so that business leaders expect more from their workers. The Internet has made the availability of information almost overwhelming, making it difficult to discern what is important and what can be ignored. Yet, it has also created a thriving global marketplace and the flexibility of a mobile office or being able to do work on the "go."

Downside to Technological Innovations

All of this revolution does not come without a price. The downsides to our ever-increasing use of technology are:

- The dehumanization of transactions which used to be face to face: Whereas the industrial revolution placed manufacturing in the hands of an assembly line rather than an artisan, the information revolution is taking it a step further and eliminating the shopkeeper who sells the finished goods. Information technologies allow us to do a good majority of our business, schooling, and personal tasks without ever leaving home. What will this mean for our society?
- The implications for data security and privacy: All of our personal information is out there in cyberspace. Crime and criminals will obviously change their methodologies and give us new things to be afraid of. How will this be handled by government and policy makers?
- The unpredictability of computer communications: We rely on computers for many more tasks than ever before. What happens when the systems go down (which they always will)? How will we get our jobs done if the systems are not there to support us?

There are a lot of questions to be answered. Nevertheless, if we can learn how to manage technology appropriately, great strides can be made in business and in our personal lives.

Topic Wrap-Up

Much has changed since the first "computing machines" were developed. As fundamentally important as any revolution or invention in modern time, the computer will continue to shape our lives for many years to come. Many parallels are drawn to the industrial revolution; this is not by chance, for the information revolution is the next step in the series of technological revolutions for our modern society. We will never be the same.

Now that you have completed this topic, you should be able to:

- Give examples of the important events in computer history as related to the early years prior to 1930.
- Give examples of the important events in computer history as related to the period of the late 1930s to 1959.
- Give examples of the important events in computer history as related to the period of 1964 to 2000.
- Discuss the importance of the beginnings of the information "Super Highway."
- Analyze the lessons learned from computer history.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ, ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Chapter 5: MAIN MEMORY

(of Introduction to Computer Science course)

Contents

How Computer Memory Works

When you think about it, it's amazing how many different types of electronic memory you encounter in daily life. Many of them have become an integral part of our vocabulary:

- RAM
    - Dynamic RAM
    - Static RAM
    - Virtual memory
    - Video memory
- ROM
    - Flash memory
    - Memory Sticks
    - BIOS
- Cache

You already know that the computer in front of you has memory. What you may not know is that most of the electronic items you use every day have some form of memory also. Here are just a few examples of the many items that use memory:

- Cell phones
- PDAs
- Game consoles
- Car radios
- VCRs
- TVs

Each of these devices uses different types of memory in different ways!

Memory Basics

Although memory is technically any form of electronic storage, it is used most often to identify fast, temporary forms of storage. If your computer's CPU had to constantly access the hard drive to retrieve every piece of data it needs, it would operate very slowly. When the information is kept in memory, the CPU can access it much more quickly. Most forms of memory are intended to store data temporarily.

As you can see in the diagram below, the CPU accesses memory according to a distinct hierarchy. Whether it comes from permanent storage (the hard drive) or input (the keyboard), most data goes in random access memory (RAM) first. The CPU then stores pieces of data it will need to access, often in a cache, and maintains certain special instructions in the register. We'll talk about cache and registers later.

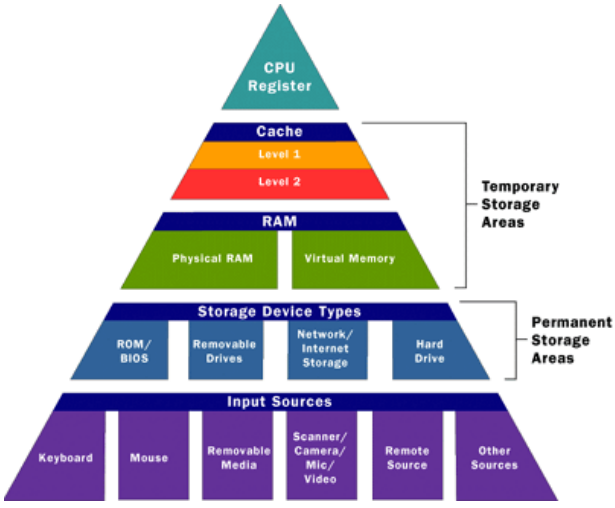


Figure 5.0: *Computer memory pyramid.*

All of the components in your computer, such as the CPU, the hard drive and the operating system, work together as a team, and memory is one of the most essential parts of this team. From the moment you turn your computer on until the time you shut it down, your CPU is constantly using memory. Let's take a look at a typical scenario:

- You turn the computer on.
- The computer loads data from read-only memory (ROM) and performs a power-on self-test (POST) to make sure all the major components are functioning properly. As part of this test, the memory controller checks all of the memory addresses with a quick read/write operation to ensure that there are no errors in the memory chips. Read/write means that data is written to a bit and then read from that bit.
- The computer loads the basic input/output system (BIOS) from ROM. The BIOS provides the most basic information about storage devices, boot sequence, security, Plug and Play (auto device recognition) capability and a few other items.
- The computer loads the operating system (OS) from the hard drive into the system's RAM. Generally, the critical parts of the operating system are maintained in RAM as long as the computer is on. This allows the CPU to have immediate access to the operating system, which enhances the performance and functionality of the overall system.
- When you open an application, it is loaded into RAM. To conserve RAM usage, many applications load only the essential parts of the program initially and then load other pieces as needed.
- After an application is loaded, any files that are opened for use in that application are loaded into RAM.
- When you save a file and close the application, the file is written to the specified storage device, and then it and the application are purged from RAM.

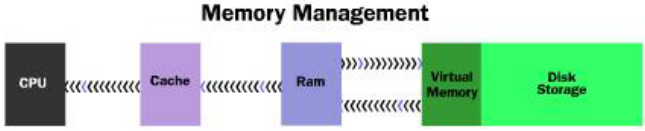
In the list above, every time something is loaded or opened, it is placed into RAM. This simply means that it has been put in the computer's temporary storage area so that the CPU can access that information more easily. The CPU requests the data it needs from RAM, processes it and writes new data back to RAM in a continuous cycle. In most computers, this shuffling of data between the CPU and RAM happens millions of times every second. When an application is closed, it and any accompanying files are usually purged (deleted) from RAM to make room for new data. If the changed files are not saved to a permanent storage device before being purged, they are lost.

The Need for Speed

One common question about desktop computers that comes up all the time is, "Why does a computer need so many memory systems?" A typical computer has:

- Level 1 and level 2 caches
- Normal system RAM
- Virtual memory
- A hard disk

Why so many? The answer to this question can teach you a lot about memory!



Fast, powerful CPUs need quick and easy access to large amounts of data in order to maximize their performance. If the CPU cannot get to the data it needs, it literally stops and waits for it. Modern CPUs running at speeds of about 1 gigahertz can consume massive amounts of data -- potentially billions of bytes per second. The problem that computer designers face is that memory that can keep up with a 1-gigahertz CPU is extremely expensive -- much more expensive than anyone can afford in large quantities.

Computer designers have solved the cost problem by "tiering" memory -- using expensive memory in small quantities and then backing it up with larger quantities of less expensive memory.

The cheapest form of read/write memory in wide use today is the hard disk. Hard disks provide large quantities of inexpensive, permanent storage. You can buy hard disk space for pennies per megabyte, but it

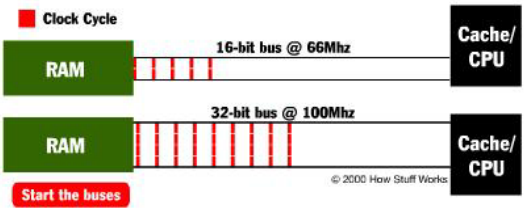
can take a good bit of time (approaching a second) to read a megabyte off a hard disk. Because storage space on a hard disk is so cheap and plentiful, it forms the final stage of a CPUs memory hierarchy, called virtual memory.

The next level of the hierarchy is RAM. The bit size of a CPU tells you how many bytes of information it can access from RAM at the same time. For example, a 16-bit CPU can process 2 bytes at a time (1 byte = 8 bits, so 16 bits = 2 bytes), and a 64-bit CPU can process 8 bytes at a time.

Megahertz (MHz) is a measure of a CPU's processing speed, or clock cycle, in millions per second. So, a 32-bit 800-MHz Pentium III can potentially process 4 bytes simultaneously, 800 million times per second (possibly more based on pipelining)! The goal of the memory system is to meet those requirements.

A computer's system RAM alone is not fast enough to match the speed of the CPU. That is why you need a cache (see the next section). However, the faster RAM is, the better. Most chips today operate with a cycle rate of 50 to 70 nanoseconds. The read / write speed is typically a function of the type of RAM used, such as DRAM, SDRAM, RAMBUS. We will talk about these various types of memory later.

System RAM speed is controlled by bus width and bus speed. Bus width refers to the number of bits that can be sent to the CPU simultaneously, and bus speed refers to the number of times a group of bits can be sent each second. A bus cycle occurs every time data travels from memory to the CPU. For example, a 100-MHz 32-bit bus is theoretically capable of sending 4 bytes (32 bits divided by 8 = 4 bytes) of data to the CPU 100 million times per second, while a 66-MHz 16-bit bus can send 2 bytes of data 66 million times per second. If you do the math, you'll find that simply changing the bus width from 16 bits to 32 bits and the speed from 66 MHz to 100 MHz in our example allows for three times as much data (400 million bytes versus 132 million bytes) to pass through to the CPU every second.



In reality, RAM doesn't usually operate at optimum speed. Latency changes the equation radically. Latency refers to the number of clock cycles needed to read a bit of information. For example, RAM rated at 100 MHz is capable of sending a bit in 0.00000001 seconds, but may take 0.00000005 seconds to start the read process for the first bit. To compensate for latency, CPUs uses a special technique called burst mode.

Burst mode depends on the expectation that data requested by the CPU will be stored in sequential memory cells. The memory controller anticipates that whatever the CPU is working on will continue to come from this same series of memory addresses, so it reads several consecutive bits of data together. This means that only the first bit is subject to the full effect of latency; reading successive bits takes significantly less time. The rated burst mode of memory is normally expressed as four numbers separated by dashes. The first number tells you the number of clock cycles needed to begin a read operation; the second, third and fourth numbers tell you how many cycles are needed to read each consecutive bit in the row, also known as the wordline. For example: 5-1-1-1 tells you that it takes five cycles to read the first bit and one cycle for each bit after that. Obviously, the lower these numbers are, the better the performance of the memory.

Burst mode is often used in conjunction with pipelining, another means of minimizing the effects of latency. Pipelining organizes data retrieval into a sort of assembly-line process. The memory controller simultaneously reads one or more words from memory, sends the current word or words to the CPU and writes one or more words to memory cells. Used together, burst mode and pipelining can dramatically reduce the lag caused by latency.

So why wouldn't you buy the fastest, widest memory you can get? The speed and width of the memory's bus should match the system's bus. You can use memory designed to work at 100 MHz in a 66-MHz system, but it will run at the 66-MHz speed of the bus so there is no advantage, and 32-bit memory won't fit on a 16-bit bus.

Cache and Registers

Even with a wide and fast bus, it still takes longer for data to get from the memory card to the CPU than it takes for the CPU to actually process the data. Caches are designed to alleviate this bottleneck by making the data used most often by the CPU instantly available. This is accomplished by building a small amount

of memory, known as primary or level 1 cache, right into the CPU. Level 1 cache is very small, normally ranging between 2 kilobytes (KB) and 64 KB.

The secondary or level 2 cache typically resides on a memory card located near the CPU. The level 2 cache has a direct connection to the CPU. A dedicated integrated circuit on the motherboard, the L2 controller, regulates the use of the level 2 cache by the CPU. Depending on the CPU, the size of the level 2 cache ranges from 256 KB to 2 megabytes (MB). In most systems, data needed by the CPU is accessed from the cache approximately 95 percent of the time, greatly reducing the overhead needed when the CPU has to wait for data from the main memory.

Some inexpensive systems dispense with the level 2 cache altogether. Many high performance CPUs now have the level 2 cache actually built into the CPU chip itself. Therefore, the size of the level 2 cache and whether it is onboard (on the CPU) is a major determining factor in the performance of a CPU.

A particular type of RAM, static random access memory (SRAM), is used primarily for cache. SRAM uses multiple transistors, typically four to six, for each memory cell. It has an external gate array known as a bistable multivibrator that switches, or flip-flops, between two states. This means that it does not have to be continually refreshed like DRAM. Each cell will maintain its data as long as it has power. Without the need for constant refreshing, SRAM can operate extremely quickly. But the complexity of each cell make it prohibitively expensive for use as standard RAM.

The SRAM in the cache can be asynchronous or synchronous. Synchronous SRAM is designed to exactly match the speed of the CPU, while asynchronous is not. That little bit of timing makes a difference in performance. Matching the CPU's clock speed is a good thing, so always look for synchronized SRAM.

The final step in memory is the registers. These are memory cells built right into the CPU that contain specific data needed by the CPU, particularly the arithmetic and logic unit (ALU). An integral part of the CPU itself, they are controlled directly by the compiler that sends information for the CPU to process.

Types of Memory

Memory can be split into two main categories: *volatile* and *nonvolatile*. Volatile memory loses any data as soon as the system is turned off; it requires constant power to remain viable. Most types of RAM fall into this category.

Nonvolatile memory does not lose its data when the system or device is turned off. A number of types of memory fall into this category. The most familiar is ROM, but Flash memory storage devices such as CompactFlash or SmartMedia cards are also forms of nonvolatile memory. See the links below for information on these types of memory.

RAM

RAM stands for **R**andom **A**ccess **M**emory. This means that Information can be retrieved and stored by the computer at any order. RAM gives your computer a temporary place to process electronic data or think. This means that, RAM chips continue to store information only as long as computer has electrical power. In other words, when you shut off your computer, all the data stored in RAM are lost.

RAM is the best known form of computer memory. The opposite of RAM is **S**erial **A**ccess **M**emory (SAM). SAM stores data as a series of memory cells that can only be accessed sequentially (like a cassette tape). If the data is not in the current location, each memory cell is checked until the needed data is found. SAM works very well for memory buffers, where the data is normally stored in the order in which it will be used (a good example is the texture buffer memory on a video card). RAM data, on the other hand, can be accessed in any order.

RAM Basics

Similar to a microprocessor, a memory chip is an integrated circuit (IC) made of millions of transistors and capacitors. In the most common form of computer memory, dynamic random access memory (DRAM), a transistor and a capacitor are paired to create a memory cell, which represents a single bit of data. The capacitor holds the bit of information -- a 0 or a 1. The transistor acts as a switch that lets the control circuitry on the memory chip read the capacitor or change its state.

A capacitor is like a small bucket that is able to store electrons. To store a 1 in the memory cell, the bucket is filled with electrons. To store a 0, it is emptied. The problem with the capacitor's bucket is that it has a leak.

In a matter of a few milliseconds a full bucket becomes empty. Therefore, for dynamic memory to work, either the CPU or the memory controller has to come along and recharge all of the capacitors holding a 1 before they discharge. To do this, the memory controller reads the memory and then writes it right back. This refresh operation happens automatically thousands of times per second.

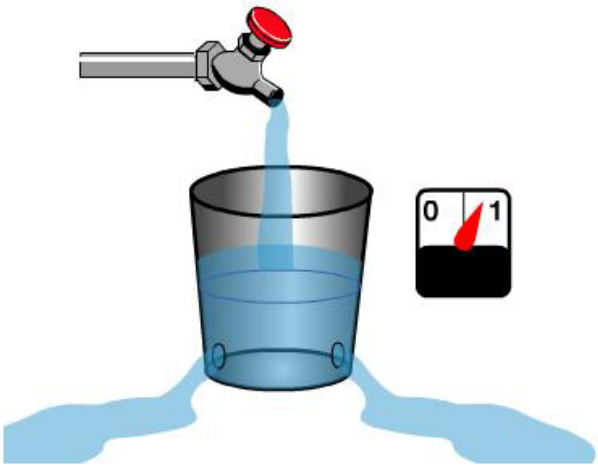


Figure 5.1: *The capacitor in a dynamic RAM memory cell is like a leaky bucket. It needs to be refreshed periodically or it will discharge to 0.*

This refresh operation is where dynamic RAM gets its name. Dynamic RAM has to be dynamically refreshed all of the time or it forgets what it is holding. The downside of all of this refreshing is that it takes time and slows down the memory.

Memory cells are etched onto a silicon wafer in an array of columns (bitlines) and rows (wordlines). The intersection of a bitline and wordline constitutes the address of the memory cell.

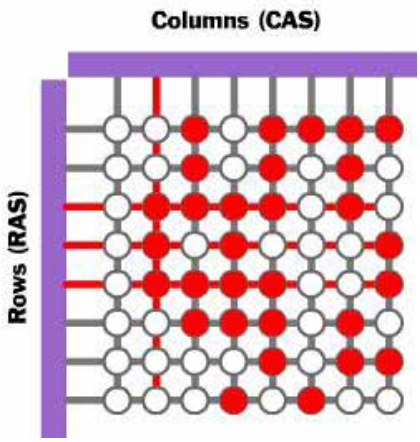


Figure 5.2: *Memory is made up of bits arranged in a two-dimensional grid.In this figure, red cells represent 1s and white cells represent 0s. A column is selected and then rows are charged to write data into the specific column.*

DRAM works by sending a charge through the appropriate column (CAS) to activate the transistor at each bit in the column. When writing, the row lines contain the state the capacitor should take on. When reading, the sense-amplifier determines the level of charge in the capacitor. If it is more than 50 percent, it reads it as a 1; otherwise it reads it as a 0. The counter tracks the refresh sequence based on which rows have been accessed in what order. The length of time necessary to do all this is so short that it is expressed in nanoseconds (billionths of a second). A memory chip rating of 70ns means that it takes 70 nanoseconds to completely read and recharge each cell.

Memory cells alone would be worthless without some way to get information in and out of them. So the memory cells have a whole support infrastructure of other specialized circuits. These circuits perform functions such as:

- Identifying each row and column (row address select and column address select)
- Keeping track of the refresh sequence (counter)
- Reading and restoring the signal from a cell (sense amplifier)
- Telling a cell whether it should take a charge or not (write enable)

Other functions of the memory controller include a series of tasks that include identifying the type, speed and amount of memory and checking for errors.

Static RAM uses a completely different technology. In static RAM, a form of flip-flop holds each bit of memory (see. A flip-flop for a memory cell takes four or six transistors along with some wiring, but never has to be refreshed. This makes static RAM significantly faster than dynamic RAM. However, because it has more parts, a static memory cell takes up a lot more space on a chip than a dynamic memory cell. Therefore, you get less memory per chip, and that makes static RAM a lot more expensive.

So static RAM is fast and expensive, and dynamic RAM is less expensive and slower. So static RAM is used to create the CPU's speed-sensitive **cache**, while dynamic RAM forms the larger system RAM space.

Memory Modules

Memory chips in desktop computers originally used a pin configuration called **d**ual **i**nline **p**ackage (DIP). This pin configuration could be soldered into holes on the computer's motherboard or plugged into a socket that was soldered on the motherboard. This method worked fine when computers typically operated on a couple of megabytes or less of RAM, but as the need for memory grew, the number of chips needing space on the motherboard increased.

The solution was to place the memory chips, along with all of the support components, on a separate printed circuit board (PCB) that could then be plugged into a special connector (memory bank) on the motherboard. Most of these chips use a small outline J-lead (SOJ) pin configuration, but quite a few manufacturers use the thin small outline package (TSOP) configuration as well. The key difference between these newer pin types and the original DIP configuration is that SOJ and TSOP chips are surface-mounted to the PCB. In other words, the pins are soldered directly to the surface of the board, not inserted in holes or sockets.

Memory chips are normally only available as part of a card called a module. You've probably seen memory listed as 8x32 or 4x16. These numbers represent the number of the chips multiplied by the capacity of each individual chip, which is measured in megabits (Mb), or one million bits. Take the result and divide it by eight to get the number of megabytes on that module. For example, 4x32 means that the module has four 32-megabit chips. Multiply 4 by 32 and you get 128 megabits. Since we know that a byte has 8 bits, we need to divide our result of 128 by 8. Our result is 16 megabytes!

The type of board and connector used for RAM in desktop computers has evolved over the past few years. The first types were proprietary, meaning that different computer manufacturers developed memory boards that would only work with their specific systems. Then came **SIMM**, which stands for **s**ingle **i**n-line **m**emory **m**odule. This memory board used a 30-pin connector and was about 3.5 x .75 inches in size (about 9 x 2 cm). In most computers, you had to install SIMMs in pairs of equal capacity and speed. This is because the width of the bus is more than a single SIMM. For example, you would install two 8-megabyte (MB) SIMMs to get 16 megabytes total RAM. Each SIMM could send 8 bits of data at one time, while the system bus could handle 16 bits at a time. Later SIMM boards, slightly larger at 4.25 x 1 inch (about 11 x 2.5 cm), used a 72-pin connector for increased bandwidth and allowed for up to 256 MB of RAM.

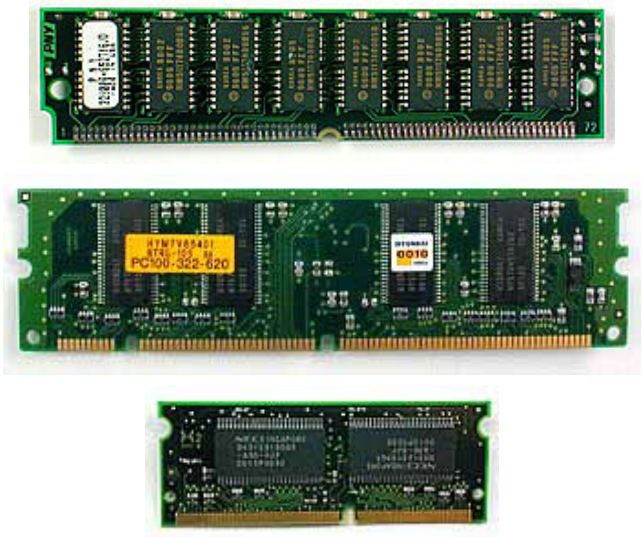


Figure 5.3: *From the top: SIMM, DIMM and SODIMM memory modules*

As processors grew in speed and bandwidth capability, the industry adopted a new standard in **d**ual **i**n-line **m**emory **m**odule (**DIMM**). With a whopping 168-pin connector and a size of 5.4 x 1 inch (about 14 x 2.5

cm), DIMMs range in capacity from 8 MB to 128 MB per module and can be installed singly instead of in pairs. Most PC memory modules operate at 3.3 volts, while Mac systems typically use 5 volts. Another standard, **R**ambus **i**n-line **m**emory **m**odule (**RIMM**), is comparable in size and pin configuration to DIMM but uses a special memory bus to greatly increase speed.

Many brands of notebook computers use proprietary memory modules, but several manufacturers use RAM based on the **s**mall **o**utline dual **i**n-line **m**emory **m**odule (**SODIMM**) configuration. SODIMM cards are small, about 2 x 1 inch (5 x 2.5 cm), and have 144 pins. Capacity ranges from 16 MB to 512 MB per module. An interesting fact about the Apple iMac desktop computer is that it uses SODIMMs instead of the traditional DIMMs.

Common RAM Types

SRAM

Static random access memory uses multiple transistors, typically four to six, for each memory cell but doesn't have a capacitor in each cell. It is used primarily for cache.

DRAM

Dynamic random access memory has memory cells with a paired transistor and capacitor requiring constant refreshing.

FPM DRAM

Fast page mode dynamic random access memory was the original form of DRAM. It waits through the entire process of locating a bit of data by column and row and then reading the bit before it starts on the next bit. Maximum transfer rate to L2 cache is approximately 176 MBps.

EDO DRAM

Extended data-out dynamic random access memory does not wait for all of the processing of the first bit before continuing to the next one. As soon as the address of the first bit is located, EDO DRAM begins looking for the next bit. It is about five percent faster than FPM. Maximum transfer rate to L2 cache is approximately 264 MBps.

SDRAM

Synchronous dynamic random access memory takes advantage of the burst mode concept to greatly improve performance. It does this by staying on the row containing the requested bit and moving rapidly through the columns, reading each bit as it goes. The idea is that most of the time the data needed by the CPU will be in sequence. SDRAM is about five percent faster than EDO RAM and is the most common form in desktops today. Maximum transfer rate to L2 cache is approximately 528 MBps.

DDR SDRAM

Double data rate synchronous dynamic RAM is just like SDRAM except that is has higher bandwidth, meaning greater speed. Maximum transfer rate to L2 cache is approximately 1,064 MBps (for DDR SDRAM 133 MHZ).

RDRAM

Rambus dynamic random access memory is a radical departure from the previous DRAM architecture. Designed by Rambus, RDRAM uses a Rambus in-line memory module (RIMM), which is similar in size and pin configuration to a standard DIMM. What makes RDRAM so different is its use of a special high-speed data bus called the Rambus channel. RDRAM memory chips work in parallel to achieve a data rate of 800 MHz, or 1,600 MBps.

Credit Card Memory

Credit card memory is a proprietary self-contained DRAM memory module that plugs into a special slot for use in notebook computers.

PCMCIA Memory Card

Another self-contained DRAM module for notebooks, cards of this type are not proprietary and should work with any notebook computer whose system bus matches the memory card's configuration.

CMOS RAM

CMOS RAM is a term for the small amount of memory used by your computer and some other devices to remember things like hard disk settings. This memory uses a small battery to provide it with the power it needs to maintain the memory contents.

VRAM

VideoRAM, also known as multiport dynamic random access memory (MPDRAM), is a type of RAM used specifically for video adapters or 3-D accelerators. The "multiport" part comes from the fact that VRAM normally has two independent access ports instead of one, allowing the CPU and graphics processor to access the RAM simultaneously. VRAM is located on the graphics card and comes in a variety of formats, many of which are proprietary. The amount of VRAM is a determining factor in the resolution and color depth of the display. VRAM is also used to hold graphics-specific information such as 3-D geometry data and texture maps. True multiport VRAM tends to be expensive, so today, many graphics cards use SGRAM (synchronous graphics RAM) instead. Performance is nearly the same, but SGRAM is cheaper.

ROM

ROM stands for **R**ead **O**nly **M**emory. ROM is computer memory on which data has been pre-recorded. The programming code and/or data on a ROM chip is written to the chip at the factory. It can be read, but it cannot be erased or removed. It's permanent. ROM retains its data or content even when the computer is turned off, unlike a computer's main memory (RAM), which needs a constant charge of electricity to keep its information. For this reason, ROM is considered to be 'non-volatile' and RAM is 'volatile'.

ROM chips are used not only in computers, but in most other electronic items as well. Let's start by identifying the different types of ROM.

ROM Types

There are five basic ROM types:

- ROM
- PROM
- EPROM
- EEPROM
- Flash memory

Each type has unique characteristics but they are all types of memory with two things in common:

- Data stored in these chips is nonvolatile -- it is not lost when power is removed.
- Data stored in these chips is either unchangeable or requires a special operation to change (unlike RAM, which can be changed as easily as it is read).

This means that removing the power source from the chip will not cause it to lose any data.

ROM at Work

Similar to RAM, ROM chips (Figure 5.3) contain a grid of columns and rows. But where the columns and rows intersect, ROM chips are fundamentally different from RAM chips. While RAM uses transistors to turn on or off access to a capacitor at each intersection, ROM uses a diode to connect the lines if the value is 1. If the value is 0, then the lines are not connected at all.

Figure 5.3: *BIOS uses Flash memory, a type of ROM.*

A diode normally allows current to flow in only one direction and has a certain threshold, known as the forward breakover, that determines how much current is required before the diode will pass it on. In silicon-based items such as processors and memory chips, the forward breakover voltage is approximately 0.6 volts. By taking advantage of the unique properties of a diode, a ROM chip can send a charge that is

above the forward breakover down the appropriate column with the selected row grounded to connect at a specific cell. If a diode is present at that cell, the charge will be conducted through to the ground, and, under the binary system, the cell will be read as being "on" (a value of 1). The neat part of ROM is that if the cell's value is 0, there is no diode at that intersection to connect the column and row. So the charge on the column does not get transferred to the row.

As you can see, the way a ROM chip works necessitates the programming of perfect and complete data when the chip is created. You cannot reprogram or rewrite a standard ROM chip. If it is incorrect, or the data needs to be updated, you have to throw it away and start over. Creating the original template for a ROM chip is often a laborious process full of trial and error. But the benefits of ROM chips outweigh the drawbacks. Once the template is completed, the actual chips can cost as little as a few cents each. They use very little power, are extremely reliable and, in the case of most small electronic devices, contain all the necessary programming to control the device. A great example is the small chip in the singing fish toy. This chip, about the size of your fingernail, contains the 30-second song clips in ROM and the control codes to synchronize the motors to the music.

PROM

Creating ROM chips totally from scratch is time-consuming and very expensive in small quantities. For this reason, mainly, developers created a type of ROM known as programmable read-only memory (PROM). Blank PROM chips can be bought inexpensively and coded by anyone with a special tool called a programmer.

PROM chips (Figure 5.4) have a grid of columns and rows just as ordinary ROMs do. The difference is that every intersection of a column and row in a PROM chip has a fuse connecting them. A charge sent through a column will pass through the fuse in a cell to a grounded row indicating a value of 1. Since all the cells have a fuse, the initial (blank) state of a PROM chip is all 1s. To change the value of a cell to 0, you use a programmer to send a specific amount of current to the cell. The higher voltage breaks the connection between the column and row by burning out the fuse. This process is known as burning the PROM.

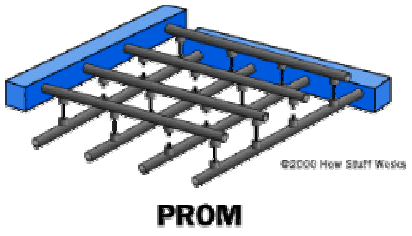


Figure 5.4: *PROM*.

PROMs can only be programmed once. They are more fragile than ROMs. A jolt of static electricity can easily cause fuses in the PROM to burn out, changing essential bits from 1 to 0. But blank PROMs are inexpensive and are great for prototyping the data for a ROM before committing to the costly ROM fabrication process.

EPROM

Working with ROMs and PROMs can be a wasteful business. Even though they are inexpensive per chip, the cost can add up over time. Erasable programmable read-only memory (EPROM) addresses this issue. EPROM chips can be rewritten many times. Erasing an EPROM requires a special tool that emits a certain frequency of ultraviolet (UV) light. EPROMs are configured using an EPROM programmer that provides voltage at specified levels depending on the type of EPROM used.

Once again we have a grid of columns and rows. In an EPROM, the cell at each intersection has two transistors. The two transistors are separated from each other by a thin oxide layer. One of the transistors is known as the floating gate and the other as the control gate. The floating gate's only link to the row (wordline) is through the control gate. As long as this link is in place, the cell has a value of 1. To change the value to 0 requires a curious process called Fowler-Nordheim tunneling. Tunneling is used to alter the placement of electrons in the floating gate. An electrical charge, usually 10 to 13 volts, is applied to the floating gate. The charge comes from the column (bitline), enters the floating gate and drains to a ground.

This charge causes the floating-gate transistor to act like an electron gun. The excited electrons are pushed through and trapped on the other side of the thin oxide layer, giving it a negative charge. These negatively charged electrons act as a barrier between the control gate and the floating gate. A device called a cell sensor monitors the level of the charge passing through the floating gate. If the flow through the gate is

greater than 50 percent of the charge, it has a value of 1. When the charge passing through drops below the 50-percent threshold, the value changes to 0. A blank EPROM has all of the gates fully open, giving each cell a value of 1.

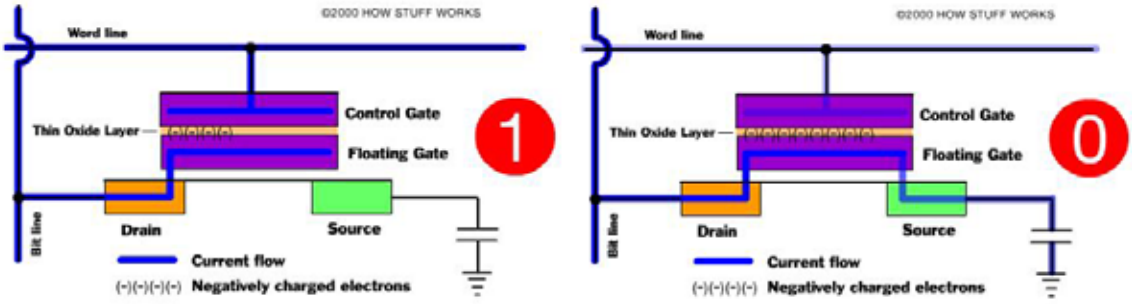


Figure 5.5: *EPROM*.

To rewrite an EPROM, you must erase it first. To erase it, you must supply a level of energy strong enough to break through the negative electrons blocking the floating gate. In a standard EPROM, this is best accomplished with UV light at a frequency of 253.7. Because this particular frequency will not penetrate most plastics or glasses, each EPROM chip has a quartz window on top of it. The EPROM must be very close to the eraser's light source, within an inch or two, to work properly.

An EPROM eraser is not selective, it will erase the entire EPROM. The EPROM must be removed from the device it is in and placed under the UV light of the EPROM eraser for several minutes. An EPROM that is left under too long can become over-erased. In such a case, the EPROM's floating gates are charged to the point that they are unable to hold the electrons at all.

EEPROMs and Flash Memory

Though EPROMs are a big step up from PROMs in terms of reusability, they still require dedicated equipment and a labor-intensive process to remove and reinstall them each time a change is necessary. Also, changes cannot be made incrementally to an EPROM; the whole chip must be erased. Electrically erasable programmable read-only memory (EEPROM) chips remove the biggest drawbacks of EPROMs.

In EEPROMs:

- The chip does not have to be removed to be rewritten.
- The entire chip does not have to be completely erased to change a specific portion of it.
- Changing the contents does not require additional dedicated equipment.

Instead of using UV light, you can return the electrons in the cells of an EEPROM to normal with the localized application of an electric field to each cell. This erases the targeted cells of the EEPROM, which can then be rewritten. EEPROMs are changed 1 byte at a time, which makes them versatile but slow. In fact, EEPROM chips are too slow to use in many products that make quick changes to the data stored on the chip.

Flash Memory

Manufacturers responded to this limitation with Flash memory, a type of EEPROM that uses in-circuit wiring to erase by applying an electrical field to the entire chip or to predetermined sections of the chip called blocks. Flash memory works much faster than traditional EEPROMs because it writes data in chunks, usually 512 bytes in size, instead of 1 byte at a time.

Electronic memory comes in a variety of forms to serve a variety of purposes. Flash memory is used for easy and fast information storage in such devices as digital cameras and home video game consoles. It is used more as a hard drive than as RAM. In fact, Flash memory is considered a solid state storage device. Solid state means that there are no moving parts -- everything is electronic instead of mechanical.

Here are a few examples of Flash memory:

- Your computer's BIOS chip
- CompactFlash (most often found in digital cameras)
- SmartMedia (most often found in digital cameras)
- Memory Stick (most often found in digital cameras)

- PCMCIA Type I and Type II memory cards (used as solid-state disks in laptops)
- Memory cards for video game consoles

The Basics

Flash memory is a type of EEPROM chip. It has a grid of columns and rows with a cell that has two transistors at each intersection (see EEPROM section).

The electrons in the cells of a Flash-memory chip can be returned to normal ("1") by the application of an electric field, a higher-voltage charge. Flash memory uses in-circuit wiring to apply the electric field either to the entire chip or to predetermined sections known as blocks. This erases the targeted area of the chip, which can then be rewritten. Flash memory works much faster than traditional EEPROMs because instead of erasing one byte at a time, it erases a block or the entire chip, and then rewrites it.

You may think that your car radio has Flash memory, since you are able to program the presets and the radio remembers them. But it is actually using Flash RAM. The difference is that Flash RAM has to have some power to maintain its contents, while Flash memory will maintain its data without any external source of power. Even though you have turned the power off, the car radio is pulling a tiny amount of current to preserve the data in the Flash RAM. That is why the radio will lose its presets if your car battery dies or the wires are disconnected.

Removable Flash Memory Cards

While your computer's **BIOS** chip is the most common form of Flash memory, removable solid-state storage devices are becoming increasingly popular. **SmartMedia** and **CompactFlash** cards are both well-known, especially as "electronic film" for digital cameras. Other removable Flash memory products include Sony's **Memory Stick**, PCMCIA memory cards, and memory cards for video game systems such as Nintendo's N64, Sega's Dreamcast and Sony's PlayStation. We will focus on SmartMedia and CompactFlash, but the essential idea is the same for all of these products. Every one of them is simply a form of Flash memory.

There are several reasons to use Flash memory instead of a hard disk:

- Flash memory is noiseless.
- It allows faster access.
- It is smaller in size.
- It is lighter.
- It has no moving parts.

So why don't we just use Flash memory for everything? Because the cost per megabyte for a hard disk is drastically cheaper, and the capacity is substantially more.

SmartMedia

The solid-state floppy-disk card (SSFDC), better known as SmartMedia, was originally developed by Toshiba.

SmartMedia cards are available in capacities ranging from 2 MB to 128 MB. The card itself is quite small, approximately 45 mm long, 37 mm wide and less than 1 mm thick. This is amazing when you consider what is packed into such a tiny package!

As shown below, SmartMedia cards are elegant in their simplicity. A plane electrode is connected to the Flash-memory chip by bonding wires. The Flash-memory chip, plane electrode and bonding wires are embedded in a resin using a technique called over-molded thin package (OMTP). This allows everything to be integrated into a single package without the need for soldering.

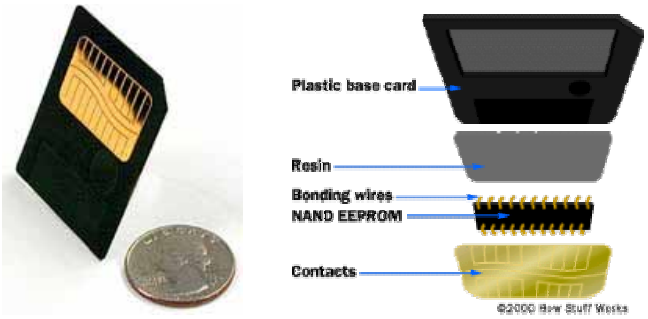


Figure 5.6: *SmartMedia card and diagram*

The OMTP module is glued to a base card to create the actual card. Power and data is carried by the electrode to the Flash-memory chip when the card is inserted into a device. A notched corner indicates the power requirements of the SmartMedia card. Looking at the card with the electrode facing up, if the notch is on the left side, the card needs 5 volts. If the notch is on the right side, it requires 3.3 volts.

SmartMedia cards erase, write and read memory in small blocks (256- or 512-byte increments). This approach means that they are capable of fast, reliable performance while allowing you to specify which data you wish to keep. They are small, lightweight and easy to use. They are less rugged than other forms of removable solid-state storage, so you should be very careful when handling and storing them.

CompactFlash

CompactFlash cards were developed by Sandisk in 1994, and they are different from SmartMedia cards in two important ways:

- They are thicker.
- They utilize a controller chip.

CompactFlash consists of a small circuit board with Flash-memory chips and a dedicated controller chip, all encased in a rugged shell that is several times thicker than a SmartMedia card.

As shown below, CompactFlash cards are 43 mm wide and 36 mm long, and come in two thicknesses: Type I cards are 3.3 mm thick, and Type II cards are 5.5 mm thick.

Figure 5.7: *CompactFlash card*

CompactFlash cards support dual voltage and will operate at either 3.3 volts or 5 volts.

The increased thickness of the card allows for greater storage capacity than SmartMedia cards. CompactFlash sizes range from 8 MB to 192 MB. The onboard controller can increase performance, particularly on devices that have slow processors. The case and controller chip add size, weight and complexity to the CompactFlash card when compared to the SmartMedia card.

Both CompactFlash and SmartMedia, as well as PCMCIA Type I and Type II memory cards, adhere to standards developed by the Personal Computer Memory Card International Association (PCMCIA). Because of these standards, it is easy to use CompactFlash and SmartMedia products in a variety of devices. You can also buy adapters that allow you to access these cards through a standard floppy drive, USB port or PCMCIA card slot (like the one you find on a laptop computer). Sony's Memory Stick is available in a large array of products offered by Sony, and is now showing up in products from other manufacturers as well.

Although standards are flourishing, there are many Flash-memory products that are completely proprietary in nature, such as the memory cards in video game systems. But it is good to know that as electronic components become increasingly interchangeable and learn to communicate with each other (by way of technologies such as Bluetooth), standardized removable memory will allow you to keep your world close at hand.

BIOS

One of the most common uses of Flash memory is for the **b**asic **i**nput/**o**utput **s**ystem of computers, commonly known as the BIOS. On virtually every computer available, the BIOS makes sure all the other chips, hard drives, ports and CPU function together.

Every desktop and laptop computer in common use today contains a microprocessor as its central processing unit. The microprocessor is the hardware component. To get its work done, the microprocessor executes a set of instructions known as **software**. You are probably very familiar with two different types of software:

- The **operating system** - The operating system provides a set of services for the applications running on your computer, and it also provides the fundamental user interface for your computer. Windows 98 and Linux are examples of operating systems.
- The **applications** - Applications are pieces of software that are programmed to perform specific tasks. On your computer right now you probably have a browser application, a word processing application, an e-mail application and so on. You can also buy new applications and install them.

It turns out that the BIOS is the third type of software your computer needs to operate successfully.

What BIOS Does

The BIOS software has a number of different roles, but its most important role is to load the operating system. When you turn on your computer and the microprocessor tries to execute its first instruction, it has to get that instruction from somewhere. It cannot get it from the operating system because the operating system is located on a hard disk, and the microprocessor cannot get to it without some instructions that tell it how. The BIOS provides those instructions. Some of the other common tasks that the BIOS performs include:

- A power-on self-test (POST) for all of the different hardware components in the system to make sure everything is working properly
- Activating other BIOS chips on different cards installed in the computer - For example, SCSI and graphics cards often have their own BIOS chips.
- Providing a set of low-level routines that the operating system uses to interface to different hardware devices - It is these routines that give the BIOS its name. They manage things like the keyboard, the screen, and the serial and parallel ports, especially when the computer is booting.
- Managing a collection of settings for the hard disks, clock, etc.

The BIOS is special software that interfaces the major hardware components of your computer with the operating system. It is usually stored on a Flash memory chip on the motherboard, but sometimes the chip is another type of ROM (see Figure 5.3).

When you turn on your computer, the BIOS does several things. This is its usual sequence:

1. Check the CMOS Setup for custom settings
2. Load the interrupt handlers and device drivers
3. Initialize registers and power management
4. Perform the power-on self-test (POST)
5. Display system settings
6. Determine which devices are bootable
7. Initiate the bootstrap sequence

The first thing the BIOS does is check the information stored in a tiny (64 bytes) amount of RAM located on a **c**omplementary **m**etal **o**xide **s**emiconductor (**CMOS**) chip. The CMOS Setup provides detailed information particular to your system and can be altered as your system changes. The BIOS uses this information to modify or supplement its default programming as needed. We will talk more about these settings later.

Interrupt handlers are small pieces of software that act as translators between the hardware components and the operating system. For example, when you press a key on your keyboard, the signal is sent to the keyboard interrupt handler, which tells the CPU what it is and passes it on to the operating system. The device drivers are other pieces of software that identify the base hardware components such as keyboard,

mouse, hard drive and floppy drive. Since the BIOS is constantly intercepting signals to and from the hardware, it is usually copied, or shadowed, into RAM to run faster.

Booting the Computer

Whenever you turn on your computer, the first thing you see is the BIOS software doing its thing. On many machines, the BIOS displays text describing things like the amount of memory installed in your computer, the type of hard disk and so on. It turns out that, during this boot sequence, the BIOS is doing a remarkable amount of work to get your computer ready to run. This section briefly describes some of those activities for a typical PC.

After checking the CMOS Setup and loading the interrupt handlers, the BIOS determines whether the video card is operational. Most video cards have a miniature BIOS of their own that initializes the memory and graphics processor on the card. If they do not, there is usually video driver information on another ROM on the motherboard that the BIOS can load.

Next, the BIOS checks to see if this is a cold boot or a reboot. It does this by checking the value at memory address 0000:0472. A value of 1234h indicates a reboot, and the BIOS skips the rest of POST. Anything else is considered a cold boot.

If it is a cold boot, the BIOS verifies RAM by performing a read/write test of each memory address. It checks the PS/2 ports or USB ports for a keyboard and a mouse. It looks for a peripheral component interconnect (PCI) bus and, if it finds one, checks all the PCI cards. If the BIOS finds any errors during the POST, it will notify you by a series of beeps or a text message displayed on the screen. An error at this point is almost always a hardware problem.

The BIOS then displays some details about your system. This typically includes information about:

- The processor
- The floppy drive and hard drive
- Memory
- BIOS revision and date
- Display

Any special drivers, such as the ones for small computer system interface (SCSI) adapters, are loaded from the adapter, and the BIOS displays the information. The BIOS then looks at the sequence of storage devices identified as boot devices in the CMOS Setup. "Boot" is short for "bootstrap," as in the old phrase, "Lift yourself up by your bootstraps." Boot refers to the process of launching the operating system. The BIOS will try to initiate the boot sequence from the first device. If the BIOS does not find a device, it will try the next device in the list. If it does not find the proper files on a device, the startup process will halt. If you have ever left a floppy disk in the drive when you restarted your computer, you have probably seen this message.

Configuring BIOS

In the previous list, you saw that the BIOS checks the CMOS Setup for custom settings. Here's what you do to change those settings.

To enter the CMOS Setup, you must press a certain key or combination of keys during the initial startup sequence. Most systems use "Esc," "Del," "F1," "F2," "Ctrl-Esc" or "Ctrl-Alt-Esc" to enter setup. There is usually a line of text at the bottom of the display that tells you "Press ___ to Enter Setup."

Once you have entered setup, you will see a set of text screens with a number of options. Some of these are standard, while others vary according to the BIOS manufacturer. Common options include:

- **System Time/Date** - Set the system time and date
- **Boot Sequence** - The order that BIOS will try to load the operating system
- **Plug and Play** - A standard for auto-detecting connected devices; should be set to "Yes" if your computer and operating system both support it
- **Mouse/Keyboard** - "Enable Num Lock," "Enable the Keyboard," "Auto-Detect Mouse"...
- **Drive Configuration** - Configure hard drives, CD-ROM and floppy drives
- **Memory** - Direct the BIOS to shadow to a specific memory address
- **Security** - Set a password for accessing the computer

- **Power Management** - Select whether to use power management, as well as set the amount of time for standby and suspend
- **Exit** - Save your changes, discard your changes or restore default settings

Be very careful when making changes to setup. Incorrect settings may keep your computer from booting. When you are finished with your changes, you should choose "Save Changes" and exit. The BIOS will then restart your computer so that the new settings take effect.

The BIOS uses CMOS technology to save any changes made to the computer's settings. With this technology, a small lithium or Ni-Cad battery can supply enough power to keep the data for years. In fact, some of the newer chips have a 10-year, tiny lithium battery built right into the CMOS chip!

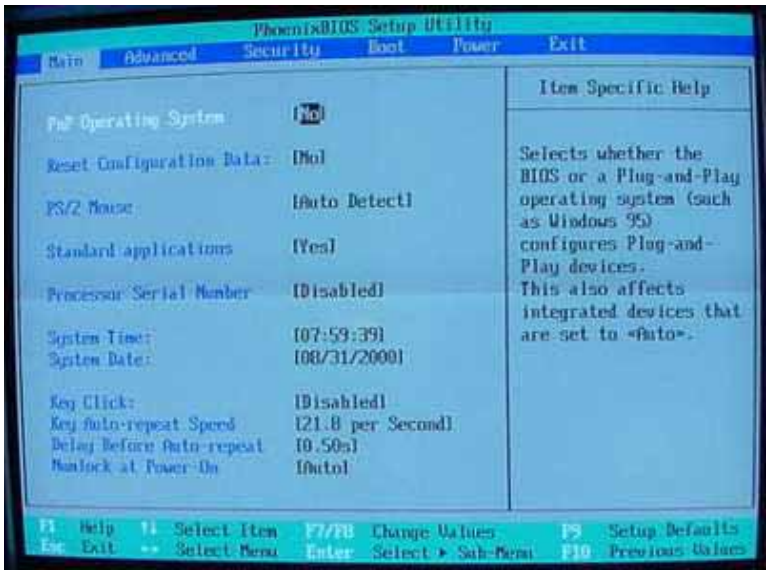


Figure 5.8: *CMOS Setup*

Updating Your BIOS

Occasionally, a computer will need to have its BIOS updated. This is especially true of older machines. As new devices and standards arise, the BIOS needs to change in order to understand the new hardware. Since the BIOS is stored in some form of ROM, changing it is a bit harder than upgrading most other types of software.

To change the BIOS itself, you'll probably need a special program from the computer or BIOS manufacturer. Look at the BIOS revision and date information displayed on system startup or check with your computer manufacturer to find out what type of BIOS you have. Then go to the BIOS manufacturer's Web site to see if an upgrade is available. Download the upgrade and the utility program needed to install it. Sometimes the utility and update are combined in a single file to download. Copy the program, along with the BIOS update, onto a floppy disk. Restart your computer with the floppy disk in the drive, and the program erases the old BIOS and writes the new one. You can find a BIOS Wizard that will check your BIOS at BIOS Upgrades.

Major BIOS manufacturers include:

- American Megatrends Inc. (AMI)
- Phoenix Technologies
- ALi
- Winbond

As with changes to the CMOS Setup, be careful when upgrading your BIOS. Make sure you are upgrading to a version that is compatible with your computer system. Otherwise, you could corrupt the BIOS, which means you won't be able to boot your computer. If in doubt, check with your computer manufacturer to be sure you need to upgrade.

Cache Memory

Cache memory provides the quick access to information. Main memory access by the CPU may take as long as 180ns; however, it make take 45ns to access external cache memory, even less for internal or on-board cache. While the CPU is processing formation retrieved from cache, the cache controller is refreshing cache with data and instructions from main memory or a storage device

If you have been shopping for a computer, then you have heard the word "cache." Modern computers have both L1 and L2 caches. You may also have gotten advice on the topic from well-meaning friends, perhaps something like "Don't buy that Celeron chip, it doesn't have any cache in it!"

It turns out that caching is an important computer-science process that appears on every computer in a variety of forms. There are memory caches, hardware and software disk caches, page caches and more. Virtual memory is even a form of caching.

A Simple Example

Caching is a technology based on the memory subsystem of your computer. The main purpose of a cache is to accelerate your computer while keeping the price of the computer low. Caching allows you to do your computer tasks more rapidly.

To understand the basic idea behind a cache system, let's start with a super-simple example that uses a librarian to demonstrate caching concepts. Let's imagine a librarian behind his desk. He is there to give you the books you ask for. For the sake of simplicity, let's say you can't get the books yourself -- you have to ask the librarian for any book you want to read, and he fetches it for you from a set of stacks in a storeroom (the library of congress in Washington, D.C., is set up this way). First, let's start with a librarian without cache.

The first customer arrives. He asks for the book Moby Dick. The librarian goes into the storeroom, gets the book, returns to the counter and gives the book to the customer. Later, the client comes back to return the book. The librarian takes the book and returns it to the storeroom. He then returns to his counter waiting for another customer. Let's say the next customer asks for Moby Dick (you saw it coming...). The librarian then has to return to the storeroom to get the book he recently handled and give it to the client. Under this model, the librarian has to make a complete round trip to fetch every book -- even very popular ones that are requested frequently. Is there a way to improve the performance of the librarian?

Yes, there's a way -- we can put a cache on the librarian. Let's give the librarian a backpack into which he will be able to store 10 books (in computer terms, the librarian now has a 10-book cache). In this backpack, he will put the books the clients return to him, up to a maximum of 10. Let's use the prior example, but now with our new-and-improved caching librarian.

The day starts. The backpack of the librarian is empty. Our first client arrives and asks for Moby Dick. No magic here -- the librarian has to go to the storeroom to get the book. He gives it to the client. Later, the client returns and gives the book back to the librarian. Instead of returning to the storeroom to return the book, the librarian puts the book in his backpack and stands there (he checks first to see if the bag is full -- more on that later). Another client arrives and asks for Moby Dick. Before going to the storeroom, the librarian checks to see if this title is in his backpack. He finds it! All he has to do is take the book from the backpack and give it to the client. There's no journey into the storeroom, so the client is served more efficiently.

What if the client asked for a title not in the cache (the backpack)? In this case, the librarian is less efficient with a cache than without one, because the librarian takes the time to look for the book in his backpack first. One of the challenges of cache design is to minimize the impact of cache searches, and modern hardware has reduced this time delay to practically zero. Even in our simple librarian example, the *latency* time (the waiting time) of searching the cache is so small compared to the time to walk back to the storeroom that it is irrelevant. The cache is small (10 books), and the time it takes to notice a miss is only a tiny fraction of the time that a journey to the storeroom takes.

From this example you can see several important facts about caching:

- Cache technology is the use of a faster but smaller memory type to accelerate a slower but larger memory type.
- When using a cache, you must check the cache to see if an item is in there. If it is there, it's called a cache hit. If not, it is called a cache miss and the computer must wait for a round trip from the larger, slower memory area.

- A cache has some maximum size that is much smaller than the larger storage area.
- It is possible to have multiple layers of cache. With our librarian example, the smaller but faster memory type is the backpack, and the storeroom represents the larger and slower memory type. This is a one-level cache. There might be another layer of cache consisting of a shelf that can hold 100 books behind the counter. The librarian can check the backpack, then the shelf and then the storeroom. This would be a two-level cache.

Computer Caches

A computer is a machine in which we measure time in very small increments. When the microprocessor accesses the main memory (RAM), it does it in about 60 nanoseconds (60 billionths of a second). That's pretty fast, but it is much slower than the typical microprocessor. Microprocessors can have cycle times as short as 2 nanoseconds, so to a microprocessor 60 nanoseconds seems like an eternity.

What if we build a special memory bank, small but very fast (around 30 nanoseconds)? That's already two times faster than the main memory access. That's called a level 2 cache or an L2 cache. What if we build an even smaller but faster memory system directly into the microprocessor's chip? That way, this memory will be accessed at the speed of the microprocessor and not the speed of the memory bus. That's an L1 cache, which on a 233-megahertz (MHz) Pentium is 3.5 times faster than the L2 cache, which is two times faster than the access to main memory.

There are a lot of subsystems in a computer; you can put cache between many of them to improve performance. Here's an example. We have the microprocessor (the fastest thing in the computer). Then there's the L1 cache that caches the L2 cache that caches the main memory which can be used (and is often used) as a cache for even slower peripherals like hard disks and CD-ROMs. The hard disks are also used to cache an even slower medium -- your Internet connection.

Your Internet connection is the slowest link in your computer. So your browser (Internet Explorer, Netscape, Opera, etc.) uses the hard disk to store HTML pages, putting them into a special folder on your disk. The first time you ask for an HTML page, your browser renders it and a copy of it is also stored on your disk. The next time you request access to this page, your browser checks if the date of the file on the Internet is newer than the one cached. If the date is the same, your browser uses the one on your hard disk instead of downloading it from Internet. In this case, the smaller but faster memory system is your hard disk and the larger and slower one is the Internet.

Cache can also be built directly on peripherals. Modern hard disks come with fast memory, around 512 kilobytes, hardwired to the hard disk. The computer doesn't directly use this memory -- the hard-disk controller does. For the computer, these memory chips are the disk itself. When the computer asks for data from the hard disk, the hard-disk controller checks into this memory before moving the mechanical parts of the hard disk (which is very slow compared to memory). If it finds the data that the computer asked for in the cache, it will return the data stored in the cache without actually accessing data on the disk itself, saving a lot of time.

Here's an experiment you can try. Your computer caches your floppy drive with main memory, and you can actually see it happening. Access a large file from your floppy -- for example, open a 300-kilobyte text file in a text editor. The first time, you will see the light on your floppy turning on, and you will wait. The floppy disk is extremely slow, so it will take 20 seconds to load the file. Now, close the editor and open the same file again. The second time (don't wait 30 minutes or do a lot of disk access between the two tries) you won't see the light turning on, and you won't wait. The operating system checked into its memory cache for the floppy disk and found what it was looking for. So instead of waiting 20 seconds, the data was found in a memory subsystem much faster than when you first tried it (one access to the floppy disk takes 120 milliseconds, while one access to the main memory takes around 60 nanoseconds -- that's a lot faster). You could have run the same test on your hard disk, but it's more evident on the floppy drive because it's so slow.

To give you the big picture of it all, here's a list of a normal caching system:

- L1 cache - Memory accesses at full microprocessor speed (10 nanoseconds, 4 kilobytes to 16 kilobytes in size)
- L2 cache - Memory access of type SRAM (around 20 to 30 nanoseconds, 128 kilobytes to 512 kilobytes in size)
- Main memory - Memory access of type RAM (around 60 nanoseconds, 32 megabytes to 128 megabytes in size)
- Hard disk - Mechanical, slow (around 12 milliseconds, 1 gigabyte to 10 gigabytes in size)

- Internet - Incredibly slow (between 1 second and 3 days, unlimited size)

As you can see, the L1 cache caches the L2 cache, which caches the main memory, which can be used to cache the disk subsystems, and so on.

Cache Technology

One common question asked at this point is, "Why not make all of the computer's memory run at the same speed as the L1 cache, so no caching would be required?" That would work, but it would be incredibly expensive. The idea behind caching is to use a small amount of expensive memory to speed up a large amount of slower, less-expensive memory.

In designing a computer, the goal is to allow the microprocessor to run at its full speed as inexpensively as possible. A 500-MHz chip goes through 500 million cycles in one second (one cycle every two nanoseconds). Without L1 and L2 caches, an access to the main memory takes 60 nanoseconds, or about 30 wasted cycles accessing memory.

When you think about it, it is kind of incredible that such relatively tiny amounts of memory can maximize the use of much larger amounts of memory. Think about a 256-kilobyte L2 cache that caches 64 megabytes of RAM. In this case, 256,000 bytes efficiently caches 64,000,000 bytes. Why does that work?

In computer science, we have a theoretical concept called *locality of reference*. It means that in a fairly large program, only small portions are ever used at any one time. As strange as it may seem, locality of reference works for the huge majority of programs. Even if the executable is 10 megabytes in size, only a handful of bytes from that program are in use at any one time, and their rate of repetition is very high. Let's take a look at the following pseudo-code to see why locality of reference works:

```
Output to screen « Enter a number  between 1 and 100 »
Read input from user
Put value from user in variable X
Put value 100 in variable Y
Put value 1 in variable Z
Loop Y number of time
  Divide Z by X
  If the remainder of the division = 0
    then output « Z is a multiple of X »
  Add 1 to Z
Return to loop
End
```

This small program asks the user to enter a number between 1 and 100. It reads the value entered by the user. Then, the program divides every number between 1 and 100 by the number entered by the user. It checks if the remainder is 0 (modulo division). If so, the program outputs "Z is a multiple of X" (for example, 12 is a multiple of 6), for every number between 1 and 100. Then the program ends.

Even if you don't know much about computer programming, it is easy to understand that in the 11 lines of this program, the *loop* part (lines 7 to 9) are executed 100 times. All of the other lines are executed only once. Lines 7 to 9 will run significantly faster because of caching.

This program is very small and can easily fit entirely in the smallest of L1 caches, but let's say this program is huge. The result remains the same. When you program, a lot of action takes place inside loops. A word processor spends 95 percent of the time waiting for your input and displaying it on the screen. This part of the word-processor program is in the cache.

This 95%-to-5% ratio (approximately) is what we call the locality of reference, and it's why a cache works so efficiently. This is also why such a small cache can efficiently cache such a large memory system. You can see why it's not worth it to construct a computer with the fastest memory everywhere. We can deliver 95 percent of this effectiveness for a fraction of the cost.

Two of the main factors that affect a cache's performance are size (or amount of cache memory) and level.

Cache Level

When we talk about the level of cache we're referring to it's connecting circuits or electronic pathways (bus) and physical proximity to the CPU. What does this have to do with performance? First off, think about the actual physical distance. Does it take you longer to walk 50 feet as opposed to 10 feet? Data that flows as electricity seems almost instantaneous, but it does have to travel along those wires. How fast can it travel along the bus? Remember the clock?

Internal Cache (Level I or L1)

Internal cache is a small amount of SRAM that is placed inside the actual CPU (internally) that is accessed directly. It runs at the same speed as the processor. With the introduction of the 486 CPU, Intel placed 8K of internal cache memory in the processor. Today's computers have 16K - 64K of L1 cache memory. With the newer processors, this means the L1 cache can be accessed at speeds up to 650+ MHz. This can't help but enhance performance.

External Cache (Level II or L2)

External Cache is separate (or external) from the CPU. It's often socketed on the motherboard in DIP chips or in COAST slots ('Cache On A Stick' is SRAM that is socketed on a small circuit board or module and installs in a long thin slot on the system board). If your system bus speed is 66 MHz, then that's how fast the bus speed is to your External Cache. No matter how fast your CPU runs internally, the external cache is limited to the speed of the system bus. External cache is often called Level II, or L2 cache. Newer computers will typically have 256K or 512K of level II cache, but check your documentation because sizes do vary, and so does the possibility of upgrade.

To shorten the bus length and increase the speed to L2 cache, modern computers have the level II cache inside the CPU package, whether its inside the actual CPU chip or in the CPU cartridge (in the case of an Intel PII or PIII). It's still separate from the CPU core and connected by a bridge or what is termed the 'backside bus'. Speeds range from CPU core speed to regular system bus speed. In the case of the PII and PIII, the backside bus allows for transfers at half the processor's core speed.

Intel has always set the standards, so let's use their processors as an example. The Pentium II and the Pentium III typically have 512K of Level II cache separate from the CPU, but in the same slot 1 cartridge. It's connected to the CPU by a backside bus running at half the processor's speed. The Celeron CPU was aimed at the lower end market and had the same core as the PII but no external cache on the backside bus. Not a big seller! Intel revised the Celeron and came out with the Celeron 'A', which had 128K on the backside bus. They've continued that practice with the newer Celerons in the PGA format.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ, ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Chapter 6: Central Processing Unit

(of Introduction to Computer Science course)

Contents

Central Processing Unit

The computer you are using to read this page uses a microprocessor to do its work. The microprocessor is the heart of any normal computer, whether it is a desktop machine, a server or a laptop. The microprocessor you are using might be a Pentium, a K6, a PowerPC, a Sparc or any of the many other brands and types of microprocessors, but they all do approximately the same thing in approximately the same way.

If you have ever wondered what the microprocessor in your computer is doing, or if you have ever wondered about the differences between types of microprocessors, then read on.

Microprocessor History

A microprocessor -- also known as a CPU or central processing unit -- is a complete computation engine that is fabricated on a single chip. The first microprocessor was the Intel 4004, introduced in 1971. The 4004 was not very powerful -- all it could do was add and subtract, and it could only do that 4 bits at a time. But it was amazing that everything was on one chip. Prior to the 4004, engineers built computers either from collections of chips or from discrete components (transistors wired one at a time). The 4004 powered one of the first portable electronic calculators.

The first microprocessor to make it into a home computer was the Intel 8080, a complete 8-bit computer on one chip, introduced in 1974. The first microprocessor to make a real splash in the market was the Intel 8088, introduced in 1979 and incorporated into the IBM PC (which first appeared around 1982). If you are familiar with the PC market and its history, you know that the PC market moved from the 8088 to the 80286 to the 80386 to the 80486 to the Pentium to the Pentium II to the Pentium III to the Pentium 4. All of these microprocessors are made by Intel and all of them are improvements on the basic design of the 8088. The Pentium 4 can execute any piece of code that ran on the original 8088, but it does it about 5,000 times faster!

The following table helps you to understand the differences between the different processors that Intel has introduced over the years.

Name	Date	Transistors	Microns	Clock speed	Data width	MIPS
8080	1974	6,000	6	2 MHz	8 bits	0.64
8088	1979	29,000	3	5 MHz	16 bits 8-bit bus	0.33
80286	1982	134,000	1.5	6 MHz	16 bits	1
80386	1985	275,000	1.5	16 MHz	32 bits	5
80486	1989	1,200,000	1	25 MHz	32 bits	20
Pentium	1993	3,100,000	0.8	60 MHz	32 bits 64-bit bus	100
Pentium II	1997	7,500,000	0.35	233 MHz	32 bits 64-bit bus	~300
Pentium III	1999	9,500,000	0.25	450 MHz	32 bits 64-bit bus	~510
Pentium 4	2000	42,000,000	0.18	1.5 GHz	32 bits 64-bit bus	~1,700

Information about this table:

- The **date** is the year that the processor was first introduced. Many processors are re-introduced at higher clock speeds for many years after the original release date.
- **Transistors** is the number of transistors on the chip. You can see that the number of transistors on a single chip has risen steadily over the years.

- **Microns** is the width, in microns, of the smallest wire on the chip. For comparison, a human hair is 100 microns thick. As the feature size on the chip goes down, the number of transistors rises.
- **Clock speed** is the initial maximum rate that the chip can be clocked at. Clock speed will make more sense in the next section.
- **Data Width** is the width of the ALU. An 8-bit ALU can add/subtract/multiply/etc. two 8-bit numbers, while a 32-bit ALU can manipulate 32-bit numbers. An 8-bit ALU would have to execute four instructions to add two 32-bit numbers, while a 32-bit ALU can do it in one instruction. In many cases, the external data bus is the same width as the ALU, but not always. The 8088 had a 16-bit ALU and an 8-bit bus, while the modern Pentiums fetch data 64 bits at a time for their 32-bit ALUs.
- **MIPS** stands for "millions of instructions per second" and is a rough measure of the performance of a CPU. Modern CPUs can do so many different things that MIPS ratings lose a lot of their meaning, but you can get a general sense of the relative power of the CPUs from this column.

From this table you can see that, in general, there is a relationship between clock speed and MIPS. The maximum clock speed is a function of the manufacturing process and delays within the chip. There is also a relationship between the number of transistors and MIPS. For example, the 8088 clocked at 5 MHz but only executed at 0.33 MIPS (about one instruction per 15 clock cycles). Modern processors can often execute at a rate of two instructions per clock cycle. That improvement is directly related to the number of transistors on the chip and will make more sense in the next section.

Inside a Microprocessor

To understand how a microprocessor works, it is helpful to look inside and learn about the logic used to create one. In the process you can also learn about assembly language -- the native language of a microprocessor -- and many of the things that engineers can do to boost the speed of a processor.

A microprocessor executes a collection of machine instructions that tell the processor what to do. Based on the instructions, a microprocessor does three basic things:

- Using its ALU (Arithmetic/Logic Unit), a microprocessor can perform mathematical operations like addition, subtraction, multiplication and division. Modern microprocessors contain complete floating point processors that can perform extremely sophisticated operations on large floating point numbers.
- A microprocessor can move data from one memory location to another.
- A microprocessor can make decisions and jump to a new set of instructions based on those decisions.

There may be very sophisticated things that a microprocessor does, but those are its three basic activities. The following diagram shows an extremely simple microprocessor capable of doing those three things:

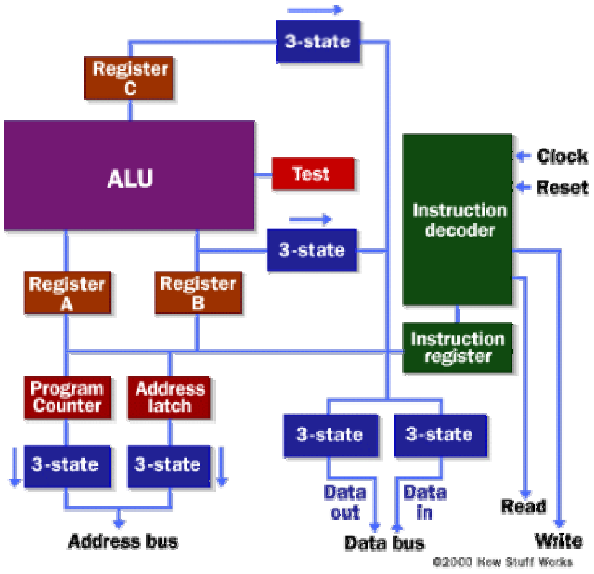


Figure 6.1: *The diagram of a simple micro-processor*

This is about as simple as a microprocessor gets. This microprocessor has:

- An **address bus** (that may be 8, 16 or 32 bits wide) that sends an address to memory

- A **data bus** (that may be 8, 16 , 32 or 64 bits wide) that can send data to memory or receive data from memory
- An **RD** (read) and **WR** (write) line to tell the memory whether it wants to set or get the addressed location
- A **clock line** that lets a clock pulse sequence the processor
- A **reset line** that resets the program counter to zero (or whatever) and restarts execution

Let's assume that both the address and data buses are 8 bits wide in this example.

Here are the components of this simple microprocessor:

- **Registers** A, B and C are simply latches made out of flip-flops. (
- The **address latch** is just like registers A, B and C.
- The **program counter** is a latch with the extra ability to increment by 1 when told to do so, and also to reset to zero when told to do so.
- The **ALU** could be as simple as an 8-bit adder, or it might be able to add, subtract, multiply and divide 8-bit values. Let's assume the latter here.
- The **test register** is a special latch that can hold values from comparisons performed in the ALU. An ALU can normally compare two numbers and determine if they are equal, if one is greater than the other, etc. The test register can also normally hold a carry bit from the last stage of the adder. It stores these values in flip-flops and then the instruction decoder can use the values to make decisions.
- There are six boxes marked "**3-State**" in the diagram. These are tri-state buffers. A tri-state buffer can pass a 1, a 0 or it can essentially disconnect its output (imagine a switch that totally disconnects the output line from the wire that the output is heading toward). A tri-state buffer allows multiple outputs to connect to a wire, but only one of them to actually drive a 1 or a 0 onto the line.
- The **instruction register** and **instruction decoder** are responsible for controlling all of the other components.

Although they are not shown in this diagram, there would be control lines from the instruction decoder that would:

- Tell the A register to latch the value currently on the data bus
- Tell the B register to latch the value currently on the data bus
- Tell the C register to latch the value currently on the data bus
- Tell the program counter register to latch the value currently on the data bus
- Tell the address register to latch the value currently on the data bus
- Tell the instruction register to latch the value currently on the data bus
- Tell the program counter to increment
- Tell the program counter to reset to zero
- Activate any of the six tri-state buffers (six separate lines)
- Tell the ALU what operation to perform
- Tell the test register to latch the ALU's test bits
- Activate the RD line
- Activate the WR line

Coming into the instruction decoder are the bits from the test register and the clock line, as well as the bits from the instruction register.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ, ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Chapter 7: Operating Systems

(of Introduction to Computer Science course)

Contents

Operating Systems

The operating system defines our computing experience. It's the first software we see when we turn on the computer, and the last software we see when the computer is turned off. It's the software that enables all the programs we use. The operating system organizes and controls the hardware on our desks and in our hands, yet most users can't say with any certainty precisely what it is that the operating system does.

The Bare Bones

It's important to realize that not all computers have operating systems. The computer that controls the microwave oven in your kitchen, for example, doesn't need an operating system. It has one set of relatively simple tasks to perform, very simple input and output methods (a keypad and an LCD screen), and simple, never-changing hardware to control. For a computer like this, an operating system would be unnecessary baggage, adding complexity where none is required. Instead, the computer in a microwave oven simply runs a single program all the time.

For computer systems that go beyond the complexity of the microwave, however, an operating system can be the key to greater operating efficiency and easier application development. All desktop computers have operating systems. The most common are the Windows family of operating systems, the UNIX family of operating systems and the Macintosh operating systems. There are hundreds of other operating systems available for special-purpose applications, including specializations for mainframes, robotics, manufacturing, real-time control systems and so on.

At the simplest level, an operating system does two things:

- It manages the hardware and software resources of the computer system. These resources include such things as the processor, memory, disk space, etc.
- It provides a stable, consistent way for applications to deal with the hardware without having to know all the details of the hardware.

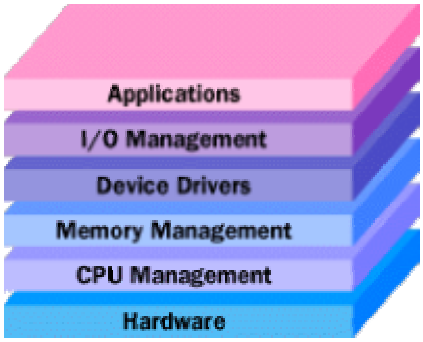


Figure 7.1: *The architecture of an Operating System*

The first task, managing the hardware and software resources, is very important, as various programs and input methods compete for the attention of the central processing unit (CPU) and demand memory, storage and input/output (I/O) bandwidth for their own purposes. In this capacity, the operating system plays the role of the good parent, making sure that each application gets the necessary resources while playing nicely with all the other applications, as well as husbanding the limited capacity of the system to the greatest good of all the users and applications.

The second task, providing a consistent application interface, is especially important if there is to be more than one of a particular type of computer using the operating system, or if the hardware making up the computer is ever open to change. A consistent application program interface (API) allows a software developer to write an application on one computer and have a high level of confidence that it will run on another computer of the same type, even if the amount of memory or the quantity of storage is different on the two machines. Even if a particular computer is unique, an operating system can ensure that applications continue to run when hardware upgrades and updates occur, because the operating system and not the application is charged with managing the hardware and the distribution of its resources. Windows 98 is a great example of the flexibility an operating system provides. Windows 98 runs on hardware from thousands of vendors. It can accommodate thousands of different printers, disk drives and special peripherals in any possible combination.

Within the broad family of operating systems, there are generally four types, categorized based on the types of computers they control and the sort of applications they support. The broad categories are:

- **Real-time operating system (RTOS)** - Real-time operating systems are used to control machinery, scientific instruments and industrial systems. An RTOS typically has very little user-interface capability, and no end-user utilities, since the system will be a "sealed box" when delivered for use. A very important part of an RTOS is managing the resources of the computer so that a particular operation executes in precisely the same amount of time every time it occurs. In a complex machine, having a part move more quickly just because system resources are available may be just as catastrophic as having it not move at all because the system is busy.
- **Single-user, single task** - As the name implies, this operating system is designed to manage the computer so that one user can effectively do one thing at a time. The **Palm OS** for Palm handheld computers is a good example of a modern single-user, single-task operating system.
- **Single-user, multi-tasking** - This is the type of operating system most people use on their desktop and laptop computers today. **Windows 98** and the **MacOS** are both examples of an operating system that will let a single user have several programs in operation at the same time. For example, it's entirely possible for a Windows user to be writing a note in a word processor while downloading a file from the Internet while printing the text of an e-mail message.
- **Multi-user** - A multi-user operating system allows many different users to take advantage of the computer's resources simultaneously. The operating system must make sure that the requirements of the various users are balanced, and that each of the programs they are using has sufficient and separate resources so that a problem with one user doesn't affect the entire community of users. **Unix**, **VMS**, and mainframe operating systems, such as **MVS**, are examples of multi-user operating systems.

It's important to differentiate here between multi-user operating systems and single-user operating systems that support networking. Windows 2000 and Novell Netware can each support hundreds or thousands of networked users, but the operating systems themselves aren't true multi-user operating systems. The system administrator is the only "user" for Windows 2000 or Netware. The network support and all of the remote user logins the network enables are, in the overall plan of the operating system, a program being run by the administrative user.

With the different types of operating systems in mind, it's time to look at the basic functions provided by an operating system.

Wake-Up Call

When the power to a computer is turned on, the first program that runs is usually a set of instructions kept in the computer's read-only memory (ROM) that examines the system hardware to make sure everything is functioning properly. This power-on self test (POST) checks the CPU, memory, and basic input-output systems (BIOS) for errors and stores the result in a special memory location. Once the POST has successfully completed, the software loaded in ROM (sometimes called firmware) will begin to activate the computer's disk drives. In most modern computers, when the computer activates the hard disk drive, it finds the first piece of the operating system: the **bootstrap loader**.

The bootstrap loader is a small program that has a single function: It loads the operating system into memory and allows it to begin operation. In the most basic form, the bootstrap loader sets up the small driver programs that interface with and control the various hardware subsystems of the computer. It sets up the divisions of memory that hold the operating system, user information and applications. It establishes the data structures that will hold the myriad signals, flags and semaphores that are used to communicate within and between the subsystems and applications of the computer. Then it turns control of the computer over to the operating system.

The operating system's tasks, in the most general sense, fall into six categories:

- Processor management
- Memory management
- Device management
- Storage management
- Application interface
- User interface

While there are some who argue that an operating system should do more than these six tasks, and some operating-system vendors do build many more utility programs and auxiliary functions into their

operating systems, these six tasks define the core of nearly all operating systems. Let's look at the tools the operating system uses to perform each of these functions.

Processor Management

The heart of managing the processor comes down to two related issues:

- Ensuring that each process and application receives enough of the processor's time to function properly
- Using as many processor cycles for real work as is possible

The basic unit of software that the operating system deals with in scheduling the work done by the processor is either a **process** or a **thread**, depending on the operating system.

It's tempting to think of a process as an application, but that gives an incomplete picture of how processes relate to the operating system and hardware. The application you see (word processor or spreadsheet or game) is, indeed, a process, but that application may cause several other processes to begin, for tasks like communications with other devices or other computers. There are also numerous processes that run without giving you direct evidence that they ever exist. A process, then, is software that performs some action and can be controlled -- by a user, by other applications or by the operating system.

It is processes, rather than applications, that the operating system controls and schedules for execution by the CPU. In a single-tasking system, the schedule is straightforward. The operating system allows the application to begin running, suspending the execution only long enough to deal with interrupts and user input. **Interrupts** are special signals sent by hardware or software to the CPU. It's as if some part of the computer suddenly raised its hand to ask for the CPU's attention in a lively meeting. Sometimes the operating system will schedule the priority of processes so that interrupts are masked -- that is, the operating system will ignore the interrupts from some sources so that a particular job can be finished as quickly as possible. There are some interrupts (such as those from error conditions or problems with memory) that are so important that they can't be ignored. These **non-maskable interrupts** (NMIs) must be dealt with immediately, regardless of the other tasks at hand.

While interrupts add some complication to the execution of processes in a single-tasking system, the job of the operating system becomes much more complicated in a multi-tasking system. Now, the operating system must arrange the execution of applications so that you believe that there are several things happening at once. This is complicated because the CPU can only do one thing at a time. In order to give the appearance of lots of things happening at the same time, the operating system has to switch between different processes thousands of times a second.

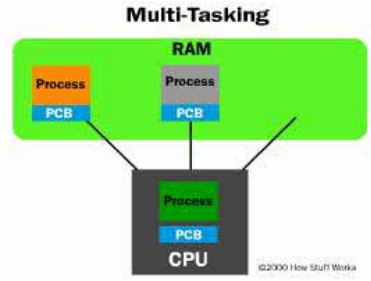


Figure 7.2: *Multitasking process management*

Here's how it happens:

- A process occupies a certain amount of RAM. It also makes use of registers, stacks and queues within the CPU and operating-system memory space.
- When two processes are multi-tasking, the operating system allots a certain number of CPU execution cycles to one program.
- After that number of cycles, the operating system makes copies of all the registers, stacks and queues used by the processes, and notes the point at which the process paused in its execution.
- It then loads all the registers, stacks and queues used by the second process and allows it a certain number of CPU cycles.
- When those are complete, it makes copies of all the registers, stacks and queues used by the second program, and loads the first program.

All of the information needed to keep track of a process when switching is kept in a data package called a process control block. The process control block typically contains:

- An ID number that identifies the process
- Pointers to the locations in the program and its data where processing last occurred
- Register contents
- States of various flags and switches
- Pointers to the upper and lower bounds of the memory required for the process
- A list of files opened by the process
- The priority of the process
- The status of all I/O devices needed by the process

When the status of the process changes, from pending to active, for example, or from suspended to running, the information in the process control block must be used like the data in any other program to direct execution of the task-switching portion of the operating system.

This process swapping happens without direct user interference, and each process gets enough CPU cycles to accomplish its task in a reasonable amount of time. Trouble can come, though, if the user tries to have too many processes functioning at the same time. The operating system itself requires some CPU cycles to perform the saving and swapping of all the registers, queues and stacks of the application processes. If enough processes are started, and if the operating system hasn't been carefully designed, the system can begin to use the vast majority of its available CPU cycles to swap between processes rather than run processes. When this happens, it's called **thrashing**, and it usually requires some sort of direct user intervention to stop processes and bring order back to the system.

One way that operating-system designers reduce the chance of thrashing is by reducing the need for new processes to perform various tasks. Some operating systems allow for a "process-lite," called a thread, that can deal with all the CPU-intensive work of a normal process, but generally does not deal with the various types of I/O and does not establish structures requiring the extensive process control block of a regular process. A process may start many threads or other processes, but a thread cannot start a process.

So far, all the scheduling we've discussed has concerned a single CPU. In a system with two or more CPUs, the operating system must divide the workload among the CPUs, trying to balance the demands of the required processes with the available cycles on the different CPUs. Asymmetric operating systems use one CPU for their own needs and divide application processes among the remaining CPUs. Symmetric operating systems divide themselves among the various CPUs, balancing demand versus CPU availability even when the operating system itself is all that's running.

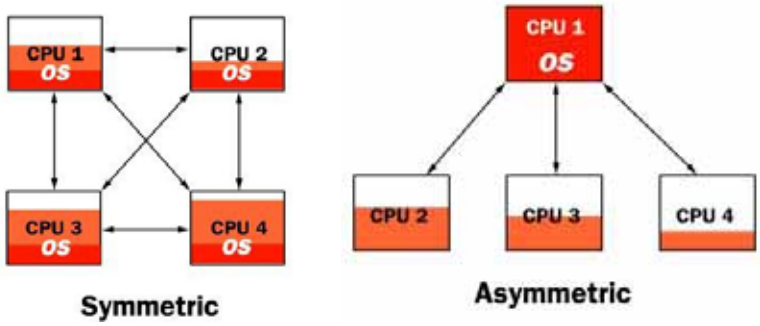


Figure 7.3: *Multiprocessor management*

Even if the operating system is the only software with execution needs, the CPU is not the only resource to be scheduled. Memory management is the next crucial step in making sure that all processes run smoothly.

Memory and Storage Management

When an operating system manages the computer's memory, there are two broad tasks to be accomplished:

- Each process must have enough memory in which to execute, and it can neither run into the memory space of another process nor be run into by another process.
- The different types of memory in the system must be used properly so that each process can run most effectively.

The first task requires the operating system to set up memory boundaries for types of software and for individual applications.

As an example, let's look at an imaginary system with 1 megabyte (1,000 kilobytes) of RAM. During the boot process, the operating system of our imaginary computer is designed to go to the top of available memory and then "back up" far enough to meet the needs of the operating system itself. Let's say that the operating system needs 300 kilobytes to run. Now, the operating system goes to the bottom of the pool of RAM and starts building up with the various driver software required to control the hardware subsystems of the computer. In our imaginary computer, the drivers take up 200 kilobytes. So after getting the operating system completely loaded, there are 500 kilobytes remaining for application processes.

When applications begin to be loaded into memory, they are loaded in block sizes determined by the operating system. If the block size is 2 kilobytes, then every process that is loaded will be given a chunk of memory that is a multiple of 2 kilobytes in size. Applications will be loaded in these fixed block sizes, with the blocks starting and ending on boundaries established by words of 4 or 8 bytes. These blocks and boundaries help to ensure that applications won't be loaded on top of one another's space by a poorly calculated bit or two. With that ensured, the larger question is what to do when the 500-kilobyte application space is filled.

In most computers, it's possible to add memory beyond the original capacity. For example, you might expand RAM from 1 to 2 megabytes. This works fine, but tends to be relatively expensive. It also ignores a fundamental fact of computing -- most of the information that an application stores in memory is not being used at any given moment. A processor can only access memory one location at a time, so the vast majority of RAM is unused at any moment. Since disk space is cheap compared to RAM, then moving information in RAM to hard disk can greatly expand RAM space at no cost. This technique is called *virtual memory management*.

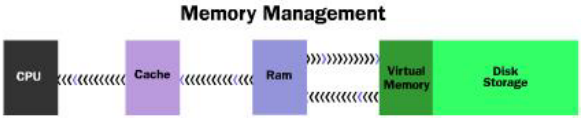


Figure 7.4: *Memory management*

Disk storage is only one of the memory types that must be managed by the operating system, and is the slowest. Ranked in order of speed, the types of memory in a computer system are:

- High-speed cache - This is fast, relatively small amounts of memory that are available to the CPU through the fastest connections. Cache controllers predict which pieces of data the CPU will need next and pull it from main memory into high-speed cache to speed up system performance.
- Main memory - This is the RAM that you see measured in megabytes when you buy a computer.
- Secondary memory - This is most often some sort of rotating magnetic storage that keeps applications and data available to be used, and serves as virtual RAM under the control of the operating system.

The operating system must balance the needs of the various processes with the availability of the different types of memory, moving data in blocks (called pages) between available memory as the schedule of processes dictates.

Device Management

The path between the operating system and virtually all hardware not on the computer's motherboard goes through a special program called a driver. Much of a driver's function is to be the translator between the electrical signals of the hardware subsystems and the high-level programming languages of the operating system and application programs. Drivers take data that the operating system has defined as a file and translate them into streams of bits placed in specific locations on storage devices, or a series of laser pulses in a printer.

Because there are such wide differences in the hardware controlled through drivers, there are differences in the way that the driver programs function, but most are run when the device is required, and function much the same as any other process. The operating system will frequently assign high-priority blocks to drivers so that the hardware resource can be released and readied for further use as quickly as possible.

One reason that drivers are separate from the operating system is so that new functions can be added to the driver -- and thus to the hardware subsystems -- without requiring the operating system itself to be modified, recompiled and redistributed. Through the development of new hardware device drivers,

development often performed or paid for by the manufacturer of the subsystems rather than the publisher of the operating system, input/output capabilities of the overall system can be greatly enhanced.

Managing input and output is largely a matter of managing queues and buffers, special storage facilities that take a stream of bits from a device, perhaps a keyboard or a serial port, hold those bits, and release them to the CPU at a rate slow enough for the CPU to cope with. This function is especially important when a number of processes are running and taking up processor time. The operating system will instruct a buffer to continue taking input from the device, but to stop sending data to the CPU while the process using the input is suspended. Then, when the process needing input is made active once again, the operating system will command the buffer to send data. This process allows a keyboard or a modem to deal with external users or computers at a high speed even though there are times when the CPU can't use input from those sources.

Managing all the resources of the computer system is a large part of the operating system's function and, in the case of real-time operating systems, may be virtually all the functionality required. For other operating systems, though, providing a relatively simple, consistent way for applications and humans to use the power of the hardware is a crucial part of their reason for existing.

Interface to the World

Application Interface

Just as drivers provide a way for applications to make use of hardware subsystems without having to know every detail of the hardware's operation, **application program interfaces** (**API**s) let application programmers use functions of the computer and operating system without having to directly keep track of all the details in the CPU's operation. Let's look at the example of creating a hard disk file for holding data to see why this can be important.

A programmer writing an application to record data from a scientific instrument might want to allow the scientist to specify the name of the file created. The operating system might provide an API function named MakeFile for creating files. When writing the program, the programmer would insert a line that looks like this:

MakeFile [1, %Name, 2]

In this example, the instruction tells the operating system to create a file that will allow random access to its data (1), will have a name typed in by the user (%Name), and will be a size that varies depending on how much data is stored in the file (2). Now, let's look at what the operating system does to turn the instruction into action.

1. The operating system sends a query to the disk drive to get the location of the first available free storage location.
2. With that information, the operating system creates an entry in the file system showing the beginning and ending locations of the file, the name of the file, the file type, whether the file has been archived, which users have permission to look at or modify the file, and the date and time of the file's creation.
3. The operating system writes information at the beginning of the file that identifies the file, sets up the type of access possible and includes other information that ties the file to the application.

In all of this information, the queries to the disk drive and addresses of the beginning and ending point of the file are in formats heavily dependent on the manufacturer and model of the disk drive.

Because the programmer has written her program to use the API for disk storage, she doesn't have to keep up with the instruction codes, data types, and response codes for every possible hard disk and tape drive. The operating system, connected to drivers for the various hardware subsystems, deals with the changing details of the hardware -- the programmer must simply write code for the API and trust the operating system to do the rest.

APIs have become one of the most hotly contested areas of the computer industry in recent years. Companies realize that programmers using their API will ultimately translate into the ability to control and profit from a particular part of the industry. This is one of the reasons that so many companies have been willing to provide applications like readers or viewers to the public at no charge. They know consumers will request that programs take advantage of the free readers, and application companies will be ready to pay royalties to allow their software to provide the functions requested by the consumers.

User Interface

Just as the API provides a consistent way for applications to use the resources of the computer system, a user interface (UI) brings structure to the interaction between a user and the computer. In the last decade, almost all development in user interfaces has been in the area of the graphical user interface (GUI), with two models, **Apple's Macintosh** and **Microsoft's Windows**, receiving most of the attention and gaining most of the market share. There are other user interfaces, some graphical and some not, for other operating systems.

Unix, for example, has user interfaces called *shells* that present a user interface more flexible and powerful than the standard operating system text-based interface. Programs such as the Korn Shell and the C Shell are text-based interfaces that add important utilities, but their main purpose is to make it easier for the user to manipulate the functions of the operating system. There are also graphical user interfaces, such as *X-Windows* and *Gnome*, that make **Unix** and **Linux** more like Windows and Macintosh computers from the user's point of view.

It's important to remember that in all of these examples, the user interface is a program or set of programs that sits as a layer above the operating system itself. The same thing is true, with somewhat different mechanisms, of both Windows and Macintosh operating systems. The core operating-system functions, the management of the computer system, lie in the kernel of the operating system. The display manager is separate, though it may be tied tightly to the kernel beneath. The ties between the operating-system kernel and the user interface, utilities and other software define many of the differences in operating systems today, and will further define them in the future.

The Future

One question concerning the future of operating systems revolves around the ability of a particular philosophy of software distribution to create an operating system useable by corporations and consumers together.

Linux, the operating system created and distributed according to the principles of open source, could have a significant impact on the operating system in general. Most operating systems, drivers and utility programs are written by commercial organizations that distribute executable versions of their software -- versions that can't be studied or altered. Open source requires the distribution of original source materials that can be studied, altered and built upon, with the results once again freely distributed.

The continuing growth of the Internet and the proliferation of computers that aren't standard desktop or laptop machines means that operating systems will change to keep pace, but the core management and interface functions will continue, even as they evolve.
