Lic. Matemática e Ciências da Computação

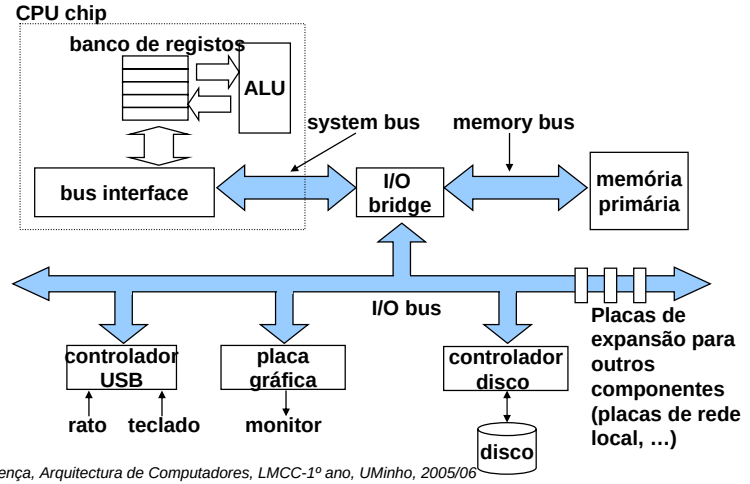
1º ano

2005/06

A.J.Proença

Introdução aos Sistemas de Computação: uma reflexão...

Arquitectura típica de um PC:



howstuffworks

How PCs Work

howstuffworks

How Microprocessors Work

howstuffworks

How Computer Memory Works

Princípios básicos de funcionamento

- **Há 2 partes envolvidas num sistema educativo:**
  - os responsáveis pelo ensino
    ... a equipa docente ...
  - os responsáveis pela aprendizagem
    ... as/os estudantes ...
- **Papel dos responsáveis do ensino**
  - expor, analisar, e debater conceitos e técnicas
  - apoiar, acompanhar e avaliar o processo de aprendizagem
- **Papel das/dos estudantes**
  - participar de modo regular e activo nas actividades lectivas
    ... sessões teóricas/práticas, trabalhos, provas ...

How PCs Work

by Jeff Tyson

When you mention the word "technology," most people think about **computers**. Virtually every facet of our lives has some computerized component. The appliances in our homes have microprocessors built into them, as do our televisions. Even our cars have a computer. But the computer that everyone thinks of first is typically the **personal computer**, or **PC**.

A PC is a general purpose tool built around a microprocessor. It has lots of different parts -- memory, a hard disk, a modem, etc. -- that work together. "General purpose" means that you can do many different things with a PC. You can use it to type documents, send e-mail, browse the Web and play games.

In this edition of **HowStuffWorks**, we will talk about PCs in the general sense and all the different parts that go into them. You will learn about the various components and how they work together in a basic operating session. You'll also find out what the future may hold for these machines.

On the Inside

Let's take a look at the main components of a typical desktop computer.

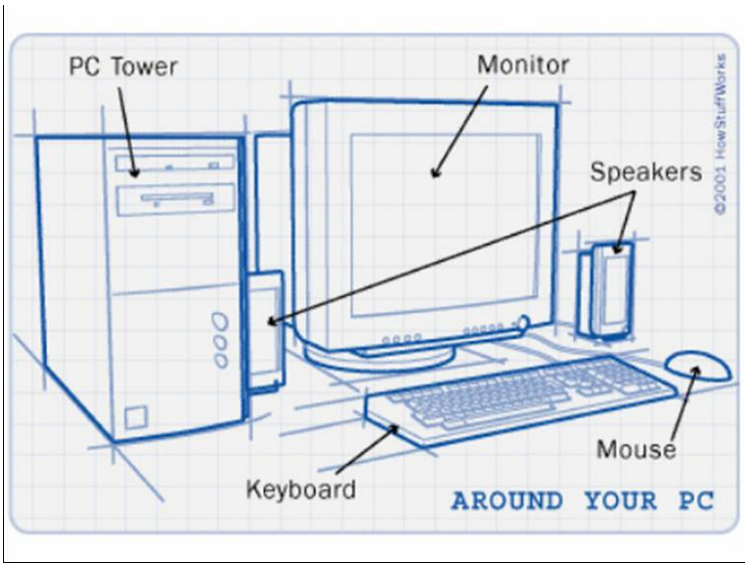
- Central processing unit (CPU) - The microprocessor "brain" of the computer system is called the central processing unit. Everything that a computer does is overseen by the CPU.

- Memory - This is very fast storage used to hold data. It has to be fast because it connects directly to the microprocessor. There are several specific types of memory in a computer:
  - Random-access memory (RAM) - Used to temporarily store information that the computer is currently working with
  - Read-only memory (ROM) - A permanent type of memory storage used by the computer for important data that does not change
  - Basic input/output system (BIOS) - A type of ROM that is used by the computer to establish basic communication when the computer is first turned on
  - Caching - The storing of frequently used data in extremely fast RAM that connects directly to the CPU
  - Virtual memory - Space on a hard disk used to temporarily store data and swap it in and out of RAM as needed

Defining a PC

Here is one way to think about it: A PC is a **general-purpose** information processing device. It can take information from a person (through the keyboard and mouse), from a device (like a floppy disk or CD) or from the network (through a modem or a network card) and process it. Once processed, the information is shown to the user (on the monitor), stored on a device (like a hard disk) or sent somewhere else on the network (back through the modem or network card).

We have lots of special-purpose processors in our lives. An MP3 Player is a specialized computer for processing MP3 files. It can't do anything else. A GPS is a specialized computer for handling GPS signals. It can't do anything else. A Gameboy is a specialized computer for handling games, but it can't do anything else. A PC can do it all because it is general-purpose.



Click on the various PC part labels to learn more about how they work.

- Motherboard - This is the main circuit board that all of the other internal components connect to. The CPU and memory are usually on the motherboard. Other systems may be found directly on the motherboard or connected to it through a secondary connection. For example, a sound card can be built into the motherboard or connected through PCI.
- Power supply - An electrical transformer regulates the electricity used by the computer.
- Hard disk - This is large-capacity permanent storage used to hold information such as programs and documents.
- Operating system - This is the basic software that allows the user to interface with the computer.
- Integrated Drive Electronics (IDE) Controller - This is the primary interface for the hard drive, CD-ROM and floppy disk drive.
- Peripheral Component Interconnect (PCI) Bus - The most common way to connect additional components to the computer, PCI uses a series of **slots** on the motherboard that PCI cards plug into.
- SCSI - Pronounced "scuzzy," the **small computer system interface** is a method of adding additional devices, such as hard drives or scanners, to the computer.
- AGP - **Accelerated Graphics Port** is a very high-speed connection used by the graphics card to interface with the computer.
- Sound card - This is used by the computer to record and play audio by converting analog sound into digital information and back again.
- Graphics card - This translates image data from the computer into a format that can be displayed by the monitor.

Connections

No matter how powerful the components inside your computer are, you need a way to interact with them. This interaction is called **input/output** (I/O). The most common types of I/O in PCs are:

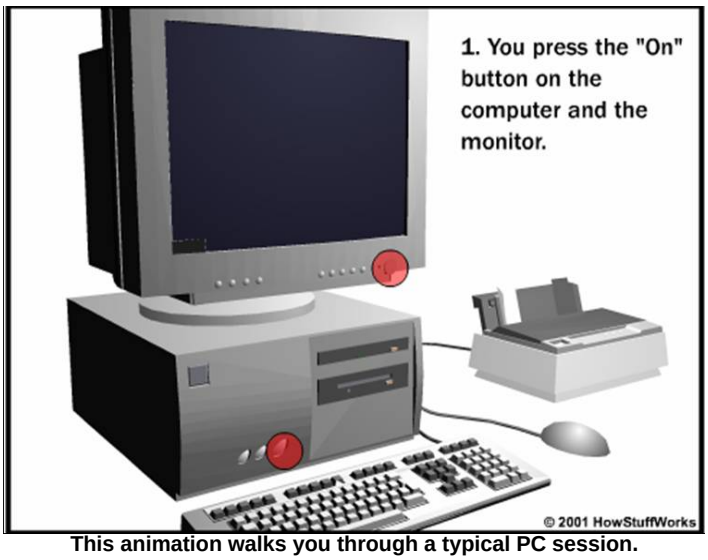
- **Monitor** - The monitor is the primary device for displaying information from the computer.
- **Keyboard** - The keyboard is the primary device for entering information into the computer.
- **Mouse** - The mouse is the primary device for navigating and interacting with the computer
- **Removable storage** - Removable storage devices allow you to add new information to your computer very easily, as well as save information that you want to carry to a different location.
  - Floppy disk - The most common form of removable storage, floppy disks are extremely inexpensive and easy to save information to.
  - CD-ROM - CD-ROM (compact disc, read-only memory) is a popular form of distribution of commercial software. Many systems now offer **CD-R** (recordable) and **CD-RW** (rewritable), which can also record.
  - Flash memory - Based on a type of ROM called **electrically erasable programmable read-only memory** (EEPROM), Flash memory provides fast, permanent storage. CompactFlash, SmartMedia and PCMCIA cards are all types of Flash memory.
  - DVD-ROM - DVD-ROM (digital versatile disc, read-only memory) is similar to CD-ROM but is capable of holding much more information.
- **Ports**
  - Parallel - This port is commonly used to connect a printer.
  - Serial - This port is typically used to connect an external modem.
  - Universal Serial Bus (USB) - Quickly becoming the most popular external connection, USB ports offer power and versatility and are incredibly easy to use.
  - FireWire (IEEE 1394) - FireWire is a very popular method of connecting digital-video devices, such as camcorders or digital cameras, to your computer.
- **Internet/network connection**
  - Modem - This is the standard method of connecting to the Internet.
  - Local area network (LAN) card - This is used by many computers, particularly those in an Ethernet office network, to connected to each other.
  - Cable modem - Some people now use the cable-television system in their home to connect to the Internet.
  - Digital Subscriber Line (DSL) modem - This is a high-speed connection that works over a standard telephone line.

- Very high bit-rate DSL (VDSL) modem - A newer variation of DSL, VDSL requires that your phone line have fiber-optic cables.

From Power-up to Shut-down

Now that you are familiar with the parts of a PC, let's see what happens in a typical computer session, from the moment you turn the computer on until you shut it down:

1. You press the "On" button on the computer and the monitor.
2. You see the **BIOS** software doing its thing, called the **power-on self-test** (POST). On many machines, the BIOS displays text describing such data as the amount of memory installed in your computer and the type of hard disk you have. During this boot sequence, the BIOS does a remarkable amount of work to get your computer ready to run.
    - 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.
    - 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, in which case the BIOS skips the rest of POST. Any other value 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 for a keyboard and a mouse. It looks for a PCI bus and, if it finds one, checks all the PCI cards. If the BIOS finds any errors during the POST, it notifies you with 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 displays some details about your system. This typically includes information about the following:
        - Processor
        - Floppy and hard drive
        - Memory
        - BIOS revision and date
        - Display
    - Any special drivers, such as the ones for SCSI adapters, are loaded from the adapter and the BIOS displays the information.
    - The BIOS 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 tries to initiate the boot sequence from the first device using the **bootstrap loader**.



This animation walks you through a typical PC session.

3. The **bootstrap loader** loads the **operating system** into memory and allows it to begin operation. It does this by setting up the divisions of memory that hold the operating system, user information and applications. The bootstrap loader then establishes the data structures that are used to communicate within and between the sub-systems and applications of the computer. Finally, it turns control of the computer over to the operating system.
4. Once loaded, the operating system's tasks fall into six broad categories:

- Processor management - Breaking the tasks down into manageable chunks and prioritizing them before sending to the CPU
- Memory management - Coordinating the flow of data in and out of RAM and determining when virtual memory is necessary
- Device management - Providing an interface between each device connected to the computer, the CPU and applications
- Storage management - Directing where data will be stored permanently on hard drives and other forms of storage
- Application Interface - Providing a standard communications and data exchange between software programs and the computer
- User Interface - Providing a way for you to communicate and interact with the computer

5. You open up a word processing program and type a letter, save it and then print it out. Several components work together to make this happen:
   - The keyboard and mouse send your input to the operating system.
   - The operating system determines that the word-processing program is the active program and accepts your input as data for that program.
   - The word-processing program determines the format that the data is in and, via the operating system, stores it temporarily in RAM.
   - Each instruction from the word-processing program is sent by the operating system to the CPU. These instructions are intertwined with instructions from other programs that the operating system is overseeing before being sent to the CPU.
   - All this time, the operating system is steadily providing display information to the graphics card, directing what will be displayed on the monitor.
   - When you choose to save the letter, the word-processing program sends a request to the operating system, which then provides a standard window for selecting where you wish to save the information and what you want to call it. Once you have chosen the name and file path, the operating system directs the data from RAM to the appropriate storage device.
   - You click on "Print." The word-processing program sends a request to the operating system, which translates the data into a format the printer understands and directs the data from RAM to the appropriate port for the printer you requested.

6. You open up a Web browser and check out **HowStuffWorks**. Once again, the operating system coordinates all of the action. This time, though, the computer receives input from another source, the Internet, as well as from you. The operating system seamlessly integrates all incoming and outgoing information.

7. You close the Web browser and choose the "Shut Down" option.

8. The operating system closes all programs that are currently active. If a program has unsaved information, you are given an opportunity to save it before closing the program.

9. The operating system writes its current settings to a special configuration file so that it will boot up next time with the same settings.

10. If the computer provides software control of power, then the operating system will completely turn off the computer when it finishes its own shut-down cycle. Otherwise, you will have to manually turn the power off.

The Future of Computing

Silicon microprocessors have been the heart of the computing world for more than 40 years. In that time, microprocessor manufacturers have crammed more and more electronic devices onto microprocessors. In accordance with **Moore's Law**, the number of electronic devices put on a microprocessor has doubled every 18 months. Moore's Law is named after Intel founder Gordon Moore, who predicted in 1965 that microprocessors would double in complexity every two years. Many have predicted that Moore's Law will soon reach its end because of the physical limitations of silicon microprocessors.

The current process used to pack more and more transistors onto a chip is called **deep-ultraviolet lithography** (DUVL), which is a photography-like technique that focuses light through lenses to carve circuit patterns on silicon wafers. DUVL will begin to reach its limit around 2005. At that time, chipmakers will have to look to other technologies to cram more transistors onto silicon to create more powerful chips. Many are already looking at extreme-ultraviolet lithography (EUVL) as a way to extend the life of silicon at least until the end of the decade. EUVL uses mirrors instead of lenses to focus the light, which allows light with shorter wavelengths to accurately focus on the silicon wafer. To learn more about EUVL, see How EUV Chipmaking Works.

Beyond EUVL, researchers have been looking at alternatives to the traditional microprocessor design. Two of the more interesting emerging technologies are **DNA computers** and **quantum computers**.

DNA computers have the potential to take computing to new levels, picking up where Moore's Law leaves off. There are several advantages to using DNA instead of silicon:

- As long as there are cellular organisms, there will be a supply of DNA.
- The large supply of DNA makes it a cheap resource.
- Unlike traditional microprocessors, which are made using toxic materials, DNA biochips can be made cleanly.
- DNA computers are many times smaller than today's computers.

DNA's key advantage is that it will make computers smaller, while at the same time increasing storage capacity, than any computer that has come before. One pound of DNA has the capacity to store more information than all the electronic computers ever built. The computing power of a teardrop-sized DNA computer, using the DNA logic gates, will be more powerful than the world's most powerful supercomputer. More than 10-trillion DNA molecules can fit into an area no larger than 1 cubic centimeter (.06 inch^3). With this small amount of DNA, a computer would be able to hold 10 terabytes (TB) of data and perform 10-trillion calculations at a time. By adding more DNA, more calculations could be performed.

As the computer moves off the desktop and becomes our constant companion, augmented-reality displays will overlay computer-generated graphics to the real world.

Unlike conventional computers, DNA computers could perform calculations simultaneously. Conventional computers operate linearly, taking on tasks one at a time. It is parallel computing that will allow DNA to solve complex mathematical problems in hours -- problems that might take electrical computers hundreds of years to complete. You can learn more about DNA computing in How DNA Computers Will Work.

Today's computers work by manipulating bits that exist in one of two states: 0 or 1. Quantum computers aren't limited to two states; they encode information as quantum bits, or **qubits**. A qubit can be a 1 or a 0, or it can exist in a **superposition** that is simultaneously 1 and 0 or somewhere in between. Qubits represent atoms that are working together to serve as computer memory and a microprocessor. Because a quantum computer can contain these multiple states simultaneously, it has the potential to be millions of times more powerful than today's most powerful supercomputers. A 30-qubit quantum computer would equal the processing power of a conventional computer capable of running at 10 **teraops**, or trillions of operations per second. Today's fastest supercomputers have achieved speeds of about 2 teraops. You can learn more about the potential of quantum computers in How Quantum Computers Will Work.


Photo courtesy IBM
By the end of the decade, we could be wearing our computers instead of sitting in front of them.

Already we are seeing powerful computers in non-desktop roles. Laptop computers and personal digital assistants (PDAs) have taken computing out of the office. Wearable computers built into our clothing and jewelry will be with us everywhere we go. Our files will follow us while our computer provides constant feedback about our environment. Voice- and handwriting-recognition software will allow us to interface with our computers without using a mouse or keyboard. Magnetic RAM and other innovations will soon provide our PC with the same instant-on accessibility that our TV and radio have.

One thing is an absolute certainty: The PC will evolve. It will get faster. It will have more capacity. And it will continue to be an integral part of our lives.

For more information, check out the links on the next page.

Lots More Information!

How Microprocessors Work

by Marshall Brain

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. In this article, you will learn how fairly simple digital logic techniques allow a computer to do its job, whether its playing a game or spell checking a document!

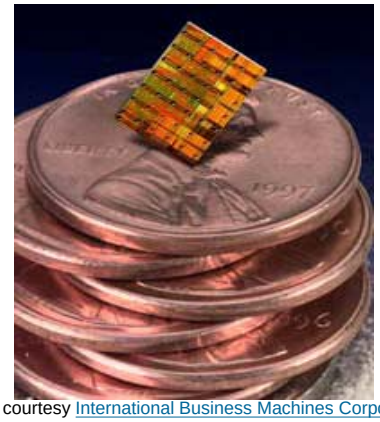


Photo courtesy International Business Machines Corporation.
Unauthorized use not permitted.
CMOS 7S "Copper chip" on a stack of pennies

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!

Intel 8080

Intel 4004 chip

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

Compiled from The Intel Microprocessor Quick Reference Guide and TSCP Benchmark Scores

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 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.

What's a Chip?

A **chip** is also called an **integrated circuit**. Generally it is a small, thin piece of silicon onto which the transistors making up the microprocessor have been etched. A chip might be as large as an inch on a side and can contain tens of millions of transistors. Simpler processors might consist of a few thousand transistors etched onto a chip just a few millimeters square.

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

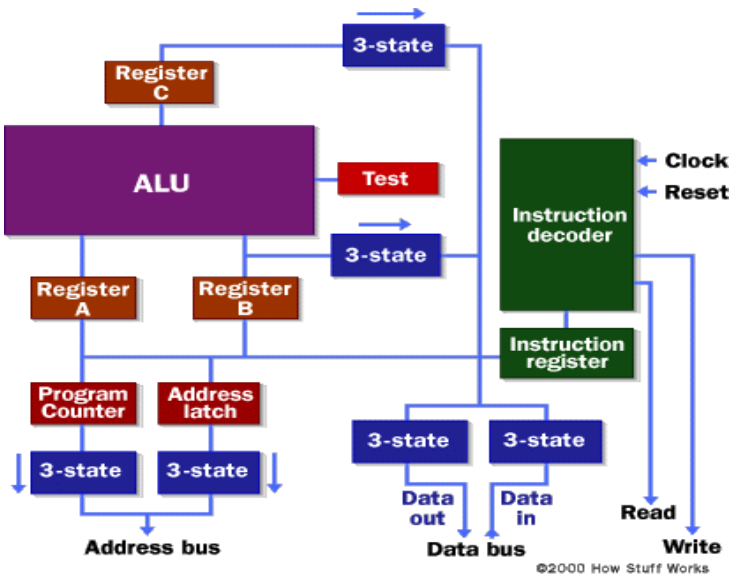
To understand how a microprocessor works, it is helpful to look inside and learn about the logic used to create one. In the process you can also learn about **assembly language** -- the native language of a microprocessor -- and many of the things that engineers can do to boost the speed of a processor.

Photo courtesy Intel Corporation
Intel Pentium 4 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:



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 or 32 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. (See the section on "edge-triggered latches" in How Boolean Logic Works for details.)
- 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 (see the section on adders in How Boolean Logic Works for details), 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

Helpful Articles

If you are new to digital logic, you may find the following articles helpful in understanding this section:

- How Bytes and Bits Work
- How Boolean Logic Works
- How Electronic Gates Work

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.

64-bit Processors

Sixty-four-bit processors have been with us since 1992, and in the 21st century they have started to become mainstream. Both Intel and AMD have introduced 64-bit chips, and the Mac G5 sports a 64-bit processor. Sixty-four-bit processors have 64-bit ALUs, 64-bit registers, 64-bit buses and so on.

One reason why the world needs 64-bit processors is because of their **enlarged address spaces**. Thirty-two-bit chips are often constrained to a maximum of 2 GB or 4 GB of RAM access. That sounds like a lot, given that most home computers currently use only 256 MB to 512 MB of RAM. However, a 4-GB limit can be a severe problem for server machines and machines running large databases. And even home machines will start bumping up against the 2 GB or 4 GB limit pretty soon if current trends continue. A 64-bit chip has none of these constraints because a 64-bit RAM address space is essentially infinite for the foreseeable future -- 2^64 bytes of RAM is something on the order of a quadrillion gigabytes of RAM.

Photo courtesy AMD

With a 64-bit address bus and wide, high-speed data buses on the

motherboard, 64-bit machines also offer faster I/O (input/output) speeds to things like hard disk drives and video cards. These features can greatly increase system performance.

Servers can definitely benefit from 64 bits, but what about normal users? Beyond the RAM solution, it is not clear that a 64-bit chip offers "normal users" any real, tangible benefits at the moment. They can process data (very complex data features lots of real numbers) faster. People doing video editing and people doing photographic editing on very large images benefit from this kind of computing power. High-end games will also benefit, once they are re-coded to take advantage of 64-bit features. But the average user who is reading e-mail, browsing the Web and editing Word documents is not really using the processor in that way. In addition, operating systems like Windows XP have not yet been upgraded to handle 64-bit CPUs. Because of the lack of tangible benefits, it will be 2010 or so before we see 64-bit machines on every desktop.

Check out ExtremeTech - 64-bit CPUs: What You Need to Know and InternetWeek - Athlon 64 Needs A Killer App to learn more.

RAM and ROM

The previous section talked about the address and data buses, as well as the RD and WR lines. These buses and lines connect either to RAM or ROM -- generally both. In our sample microprocessor, we have an address bus 8 bits wide and a data bus 8 bits wide. That means that the microprocessor can address (2^8) 256 bytes of memory, and it can read or write 8 bits of the memory at a time. Let's assume that this simple microprocessor has 128 bytes of ROM starting at address 0 and 128 bytes of RAM starting at address 128.

ROM chip

ROM stands for read-only memory. A ROM chip is programmed with a permanent collection of pre-set bytes. The address bus tells the ROM chip which byte to get and place on the data bus. When the RD line changes state, the ROM chip presents the selected byte onto the data bus.

RAM stands for random-access memory. RAM contains bytes of information, and the microprocessor can read or write to those bytes depending on whether the RD or WR line is signaled. One problem with today's RAM chips is that they forget everything once the power goes off. That is why the computer needs ROM.


RAM chips

By the way, nearly all computers contain some amount of ROM (it is possible to create a simple computer that contains no RAM -- many microcontrollers do this by placing a handful of RAM bytes on the processor chip itself -- but generally impossible to create one that contains no ROM). On a PC, the ROM is called the BIOS (Basic Input/Output System). When the microprocessor starts, it begins executing instructions it finds in the BIOS. The BIOS instructions do things like test the hardware in the machine, and then it goes to the hard disk to fetch the **boot sector** (see How Hard Disks Work for details). This boot sector is another small program, and the BIOS stores it in RAM after reading it off the disk. The microprocessor then begins executing the boot sector's instructions from RAM. The boot sector program will tell the microprocessor to fetch something else from the hard disk into RAM, which the microprocessor then executes, and so on. This is how the microprocessor loads and executes the entire operating system.

Microprocessor Instructions

Even the incredibly simple microprocessor shown in the previous example will have a fairly large set of instructions that it can perform. The collection of instructions is implemented as bit patterns, each one of which has a different meaning when loaded into the instruction register. Humans are not particularly good at remembering bit patterns, so a set of short words are defined to represent the different bit patterns. This collection of words is called the **assembly language** of the processor. An **assembler** can translate the words into their bit patterns very easily, and then the output of the assembler is placed in memory for the microprocessor to execute.

Here's the set of assembly language instructions that the designer might create for the simple microprocessor in our example:

- **LOADA mem** - Load register A from memory address
- **LOADB mem** - Load register B from memory address
- **CONB con** - Load a constant value into register B
- **SAVEB mem** - Save register B to memory address
- **SAVEC mem** - Save register C to memory address
- **ADD** - Add A and B and store the result in C
- **SUB** - Subtract A and B and store the result in C
- **MUL** - Multiply A and B and store the result in C

- **DIV** - Divide A and B and store the result in C
- **COM** - Compare A and B and store the result in test
- **JUMP addr** - Jump to an address
- **JEQ addr** - Jump, if equal, to address
- **JNEQ** addr - Jump, if not equal, to address
- **JG addr** - Jump, if greater than, to address
- **JGE addr** - Jump, if greater than or equal, to address
- **JL addr** - Jump, if less than, to address
- **JLE addr** - Jump, if less than or equal, to address
- **STOP** - Stop execution

If you have read How C Programming Works, then you know that this simple piece of C code will calculate the factorial of 5 (where the factorial of 5 = 5! = 5 * 4 * 3 * 2 * 1 = 120):

```
a=1;
f=1;
while (a <= 5)
{
    f = f * a;
    a = a + 1;
}
```

At the end of the program's execution, the variable **f** contains the factorial of 5.

A **C compiler** translates this C code into assembly language. Assuming that RAM starts at address 128 in this processor, and ROM (which contains the assembly language program) starts at address 0, then for our simple microprocessor the assembly language might look like this:

```
// Assume a is at address 128
// Assume F is at address 129
0   CONB 1      // a=1;
1   SAVEB 128
2   CONB 1      // f=1;
3   SAVEB 129
4   LOADA 128   // if a > 5 the jump to 17
5   CONB 5
6   COM
7   JG 17
8   LOADA 129   // f=f*a;
9   LOADB 128
10  MUL
11  SAVEC 129
12  LOADA 128   // a=a+1;
13  CONB 1
14  ADD
15  SAVEC 128
16  JUMP 4       // loop back to if
17  STOP
```

So now the question is, "How do all of these instructions look in ROM?" Each of these assembly language instructions must be represented by a binary number. For the sake of simplicity, let's assume each assembly language instruction is given a unique number, like this:

- LOADA - 1
- LOADB - 2
- CONB - 3
- SAVEB - 4
- SAVEC mem - 5
- ADD - 6
- SUB - 7
- MUL - 8
- DIV - 9
- COM - 10
- JUMP addr - 11
- JEQ addr - 12
- JNEQ addr - 13
- JG addr - 14

- JGE addr - 15
- JL addr - 16
- JLE addr - 17
- STOP - 18

The numbers are known as **opcodes**. In ROM, our little program would look like this:

```
// Assume a is at address 128
// Assume F is at address 129
Addr opcode/value
0    3             // CONB 1
1    1
2    4             // SAVEB 128
3    128
4    3             // CONB 1
5    1
6    4             // SAVEB 129
7    129
8    1             // LOADA 128
9    128
10   3             // CONB 5
11   5
12   10            // COM
13   14            // JG 17
14   31
15   1             // LOADA 129
16   129
17   2             // LOADB 128
18   128
19   8             // MUL
20   5             // SAVEC 129
21   129
22   1             // LOADA 128
23   128
24   3             // CONB 1
25   1
26   6             // ADD
27   5             // SAVEC 128
28   128
29   11            // JUMP 4
30   8
31   18            // STOP
```

You can see that seven lines of C code became 17 lines of assembly language, and that became 31 bytes in ROM.

The instruction decoder needs to turn each of the opcodes into a set of signals that drive the different components inside the microprocessor. Let's take the ADD instruction as an example and look at what it needs to do:

1. During the first clock cycle, we need to actually load the instruction. Therefore the instruction decoder needs to:
   - activate the tri-state buffer for the program counter
   - activate the RD line
   - activate the data-in tri-state buffer
   - latch the instruction into the instruction register
2. During the second clock cycle, the ADD instruction is decoded. It needs to do very little:
   - set the operation of the ALU to addition
   - latch the output of the ALU into the C register
3. During the third clock cycle, the program counter is incremented (in theory this could be overlapped into the second clock cycle).

Every instruction can be broken down as a set of sequenced operations like these that manipulate the components of the microprocessor in the proper order. Some instructions, like this ADD instruction, might take two or three clock cycles. Others might take five or six clock cycles.

Microprocessor Performance

The number of **transistors** available has a huge effect on the performance of a processor. As seen earlier, a typical instruction in a processor like an 8088 took 15 clock cycles to execute. Because of the design of the multiplier, it took approximately 80

cycles just to do one 16-bit multiplication on the 8088. With more transistors, much more powerful multipliers capable of single-cycle speeds become possible.

More transistors also allow for a technology called **pipelining**. In a pipelined architecture, instruction execution overlaps. So even though it might take five clock cycles to execute each instruction, there can be five instructions in various stages of execution simultaneously. That way it looks like one instruction completes every clock cycle.

Many modern processors have multiple instruction decoders, each with its own pipeline. This allows for multiple instruction streams, which means that more than one instruction can complete during each clock cycle. This technique can be quite complex to implement, so it takes lots of transistors.

The trend in processor design has primarily been toward full 32-bit ALUs with fast floating point processors built in and pipelined execution with multiple instruction streams. The newest thing in processor design is 64-bit ALUs, and people are expected to have these processors in their home PCs in the next decade. There has also been a tendency toward special instructions (like the MMX instructions) that make certain operations particularly efficient. There has also been the addition of hardware virtual memory support and L1 caching on the processor chip. All of these trends push up the transistor count, leading to the multi-million transistor powerhouses available today. These processors can execute about one billion instructions per second!

For more information on microprocessors and related topics, check out the links on the next page.

Lots More Information

Related HowStuffWorks Articles

- How Semiconductors Work
- How PCs Work
- How C Programming Works
- How Java Works
- How Operating Systems Work
- How Computer Memory Works
- How Quantum Computers Will Work
- How DNA Computers Will Work

More Great Links

- Webopedia: microprocessor
- Intel Museum: Processor Hall of Fame
- CPU Central
- Processor Upgrades
- 6th Generation CPU Comparisons
- 7th Generation CPU Comparisons
- TSCP Benchmark Scores

How Computer Memory Works

by Jeff Tyson

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

Computer Memory!

- **How Computer Memory Works**
- How BIOS Works
- How Caching Works
- How Flash Memory Works
- How RAM Works
- How Removable Storage Works
- How ROM Works
- How Virtual Memory Works

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!

In this article, you'll learn why there are so many different types of memory and what all of the terms mean.

Memory Basics

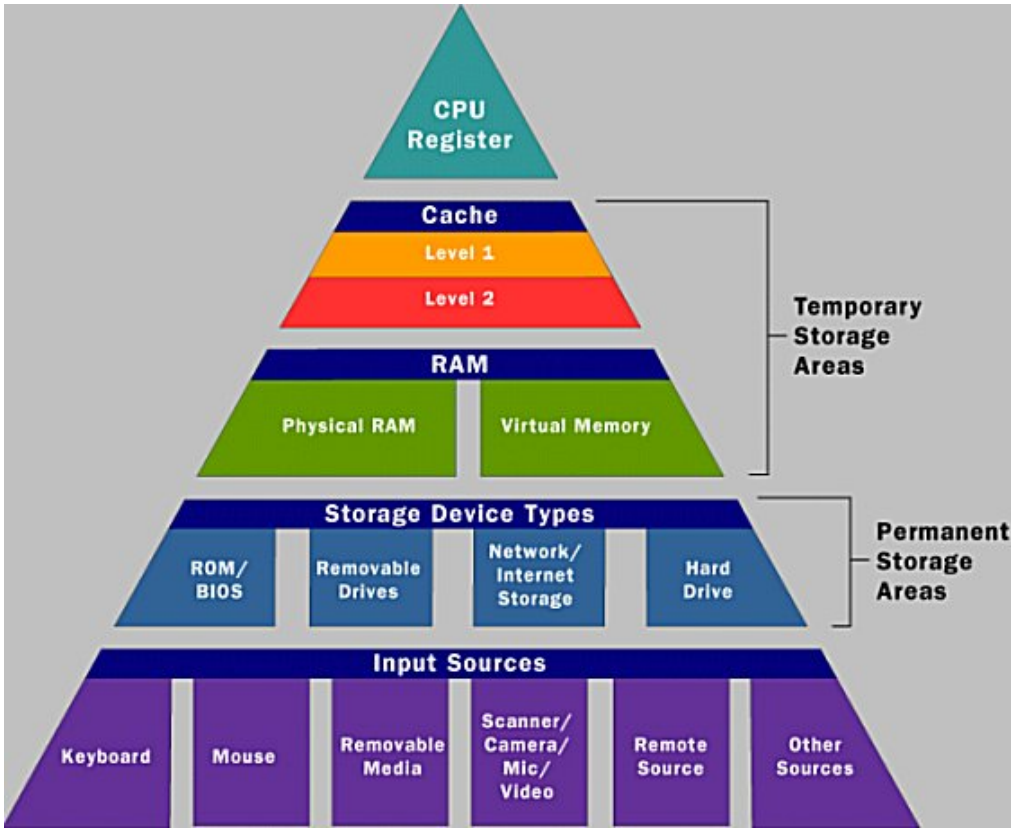
Although memory is technically any form of electronic storage, it is used most often to identify fast, temporary forms of storage. If your computer's CPU had to constantly access the hard drive to retrieve every piece of data it needs, it would operate very slowly. When the information is kept in memory,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

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.



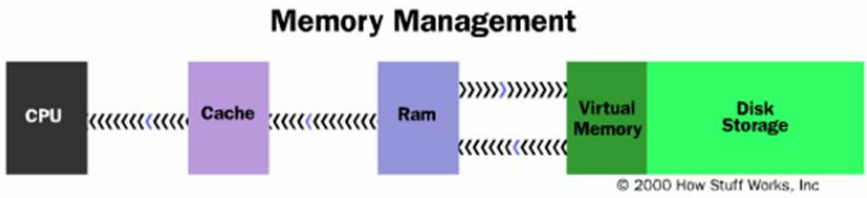
In the list above, every time something is loaded or opened, it is placed into RAM. This simply means that it has been put in the computer's **temporary storage area** so that the CPU can access that information more easily. The CPU requests the data it needs from RAM, processes it and writes new data 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**.

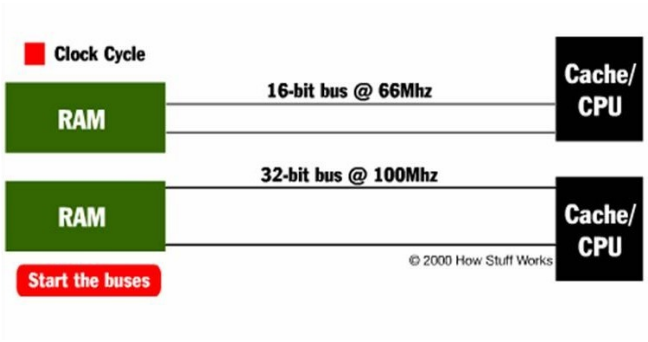
The next level of the hierarchy is **RAM**. We discuss RAM in detail in How RAM Works, but several points about RAM are important here.

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.

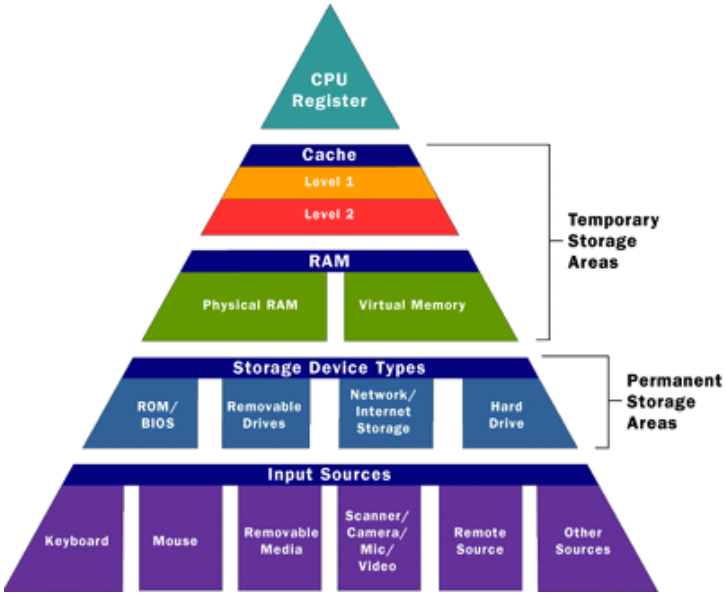
Burst mode is often used in conjunction with **pipelining**, another means of minimizing the effects of latency. Pipelining organizes data retrieval into a sort of assembly-line process. The memory controller simultaneously reads one or more words from memory, sends the current word or words to the CPU 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. For more details on caching, see How Caching Works.

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. (For more information on the various types of RAM, see How RAM Works.)

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. See How Microprocessors Work for details on registers.

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.

For a handy printable guide to computer memory, you can print the HowStuffWorks Big List of Computer Memory Terms.

Objectivos de formação * (1)

- **aquisição de conhecimentos de base relativos ao funcionamento dum computador**
  - apreender a organização e estrutura funcional dum computador, adquirindo ainda capacidades para acompanhar a sua evolução
- **aquisição de capacidades de utilização consciente e eficiente das funcionalidades dum computador**
  - adquirir aptidões intelectuais na análise e modificação da funcionalidade de programas imperativos (tipo C), com recurso ao *assembly* e a linguagem máquina
  - compreender e influenciar os factores com impacto no desempenho dos sistemas de computação

**** Perspectiva do docente***

Objectivos de formação (2)

- **aquisição de aptidões técnicas na utilização de instrumentos/ ferramentas de análise, correcção e/ou melhoria de desempenho na execução de programas**
  - adquirir aptidões técnicas na utilização de ferramentas de compilação, análise e teste de baixo nível de programas (em Linux)
- **aquisição de aptidões transversais no raciocínio crítico e criativo, no trabalho de grupo e na comunicação escrita e oral**

Resultados da aprendizagem* (1)

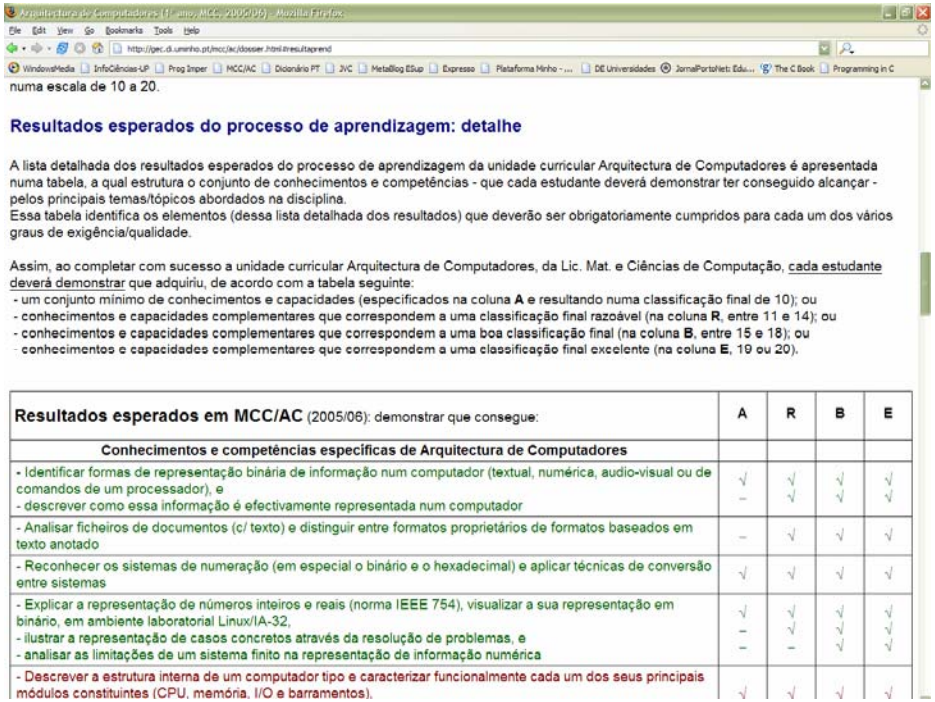
Ao completar com sucesso AC, cada estudante deverá demonstrar que adquiriu um conjunto de conhecimentos, capacidades e aptidões que compreendem:

- identificar, descrever e ilustrar formas de representação binária de informação num computador
- descrever a estrutura interna de um computador tipo e caracterizar funcionalmente cada um dos seus principais módulos constituintes (processador, memória, I/O e barramentos)
- identificar e caracterizar os principais níveis de abstracção presentes num computador e saber utilizar os mecanismos de conversão entre níveis

**** Perspectiva do estudante***

Avaliação

- **Objectivos**
  - manter a/o estudante a par do seu processo de aprendizagem, estimulando-a/o durante o semestre
  - verificar se a/o estudante satisfaz o conjunto mínimo de resultados do processo de aprendizagem
    ... como saber qual o conjunto mínimo? ...
  - classificar numericamente os alunos aprovados
- **Elementos de avaliação**
  - participação nas aulas e realização dos TPC's máx 10%
  - relatório e defesa oral de um projecto máx 33%
  - testes e/ou exame máx 67%



numa escala de 10 a 20.

Resultados esperados do processo de aprendizagem: detalhe

A lista detalhada dos resultados esperados do processo de aprendizagem da unidade curricular Arquitectura de Computadores é apresentada numa tabela, a qual estrutura o conjunto de conhecimentos e competências - que cada estudante deverá demonstrar ter conseguido alcançar - pelos principais temas/tópicos abordados na disciplina.
Essa tabela identifica os elementos (dessa lista detalhada dos resultados) que deverão ser obrigatoriamente cumpridos para cada um dos vários graus de exigência/qualidade.

Assim, ao completar com sucesso a unidade curricular Arquitectura de Computadores, da Lic. Mat. e Ciências de Computação, cada estudante deverá demonstrar que adquiriu, de acordo com a tabela seguinte:
- um conjunto mínimo de conhecimentos e capacidades (especificados na coluna **A** e resultando numa classificação final de 10); ou
- conhecimentos e capacidades complementares que correspondem a uma classificação final razoável (na coluna **R**, entre 11 e 14); ou
- conhecimentos e capacidades complementares que correspondem a uma boa classificação final (na coluna **B**, entre 15 e 18); ou
- conhecimentos e capacidades complementares que correspondem a uma classificação final excelente (na coluna **E**, 19 ou 20).

Resultados esperados em MCC/AC (2005/06): demonstrar que consegue:	A	R	B	E
Conhecimentos e competências específicas de Arquitectura de Computadores				
- Identificar formas de representação binária de informação num computador (textual, numérica, audio-visual ou de comandos de um processador), e - descrever como essa informação é efectivamente representada num computador	√ –	√ √	√ √	√ √
- Analisar ficheiros de documentos (c/ texto) e distinguir entre formatos proprietários de formatos baseados em texto anotado	–	√	√	√
- Reconhecer os sistemas de numeração (em especial o binário e o hexadecimal) e aplicar técnicas de conversão entre sistemas	√	√	√	√
- Explicar a representação de números inteiros e reais (norma IEEE 754), visualizar a sua representação em binário, em ambiente laboratorial Linux/IA-32, - ilustrar a representação de casos concretos através da resolução de problemas, e - analisar as limitações de um sistema finito na representação de informação numérica	√ – –	√ √ –	√ √ √	√ √ √
- Descrever a estrutura interna de um computador tipo e caracterizar funcionalmente cada um dos seus principais módulos constituintes (CPU, memória, I/O e barramentos).	√	√	√	√

Alguns resultados do 1º teste:

- **96 participantes**
- **70 já foram corrigidos**
- **desses,**

  60 já não satisfazem os mínimos globais, i.e., já só iriam a época de recurso!

89	fase1.pdf	21/3/2006 13:24:22
81	Fase1_3.7.zip	20/3/2006 11:07:31
62	Fase1_Grupo_62_PI_AC.zip	20/3/2006 11:05:28
101	Fase1-1.1.zip	20/3/2006 12:01:46
65	fase1-1.10.zip	21/3/2006 10:03:37
82	Fase1-1.2.zip	20/3/2006 11:49:36
123	Fase1-1.3.zip	20/3/2006 18:34:13
91	Fase1-1.zip	21/3/2006 11:19:51
102	Fase1-102.zip	21/3/2006 04:09:29
109	Fase1-109.zip	20/3/2006 14:26:23
115	fase1-115.c	20/3/2006 15:56:21
116	Fase1-116.zip	20/3/2006 15:42:24
118	Fase1-118.zip	20/3/2006 22:28:53
135	Fase1-135.zip	21/3/2006 00:20:58
143	Fase1-2.10 - Trabalho Integrado AC-PI.zip	21/3/2006 10:57:33
142	Fase1-2.11.zip	21/3/2006 10:43:38
59	Fase1-2.11.zip	21/3/2006 10:59:47
93	Fase1-2.3.zip	20/3/2006 11:42:18
70	Fase1-3.3.tar.gz	20/3/2006 09:40:23
84	Fase1-33696.zip	20/3/2006 11:16:24
72	Fase1-4.7.zip	20/3/2006 10:02:32

41	Fase1-41.c	16/3/2006 12:22:43
41	Fase1-41.pdf	16/3/2006 12:23:06
41	Fase1-41.tex	16/3/2006 12:23:17
43	fase1-43.c	20/3/2006 23:33:56
43	fase1-43.pdf	20/3/2006 23:34:32
43	fase1-43.tex	20/3/2006 23:34:51
43	fase1-43.txt	20/3/2006 23:34:41
60	fase1-60.zip	20/3/2006 12:25:52
68	Fase1-68.zip	17/3/2006 21:58:05
98	Fase1-98.zip	20/3/2006 17:06:31
92	Fase1-93.8.rar	20/3/2006 22:26:16
88	ficheiros.zip	20/3/2006 11:40:22
74	fofforos.zip	20/3/2006 10:10:51
8	Fosforos.c	20/3/2006 16:44:30
69	grupo 69 fp1.zip	17/3/2006 23:58:03
140	Grupo3.9.zip	21/3/2006 09:37:09
41	input.txt	16/3/2006 12:23:25
117	joao_carlos_miguel.pdf	21/3/2006 13:32:49
80	jogo21.zip	20/3/2006 11:04:34
90	JogoFosforos.zip	20/3/2006 11:31:04
87	new.c	20/3/2006

Fase1 (projecto): 19 submissões válidas?

papel dos **DOCENTES**

1. Transmitir conceitos+fundamentos, & métodos+técnicas de aquisição de conhecimentos
2. Treinar os estudantes na aquisição de capacidades intelectuais e aptidões práticas e experimentais
3. Estimular o interesse do estudante pelos conteúdos e temas da disciplina e apoiá-los sempre que necessário
4. Ser justo na avaliação, e contribuir para melhorar o desempenho

papel dos **ESTUDANTES**

1. Escutar nas aulas, ler a bibliografia disponibilizada e assimilar os conceitos
2. Exercitar o raciocínio na utilização e integração de conceitos, métodos e práticas
3. Desenvolver a capacidade de auto-aprendizagem pela pesquisa de info e pelo debate de ideias
4. Preparar-se para os elementos de avaliação, com seriedade

DOCENTE-ESTUDANTE

Bom relacionamento, baseado em
1. Honestidade (mútua)
2. Respeito (mútuo)
3. Confiança (mútua)