### 1.3 Two Key Concepts To Keep In Mind

(a) the notion of abstraction and
(b) the importance of not separating in your mind the notions of hardware and software.

Their value to your development as an effective software engineer goes well beyond your understanding of how a computer works and how to program it.
The notion of abstraction is central to all that you will learn and expect to use in practicing your craft, whether it be in mathematics, physics, any aspect of engineering, or business. It is hard to think of any body of knowledge where the notion of abstraction is not critical.
The misguided hardware/software separation is directly related to your continuing study of computers and your work with them.

#### 1.3.1 The Notion of Abstraction

The use of abstraction is all around us. When we get in a taxi and tell the driver, “Take me to the airport,”we are using abstraction. Ifwe had to,we could probably direct the driver each step of the way: “Go down this street ten blocks, and make a left turn.” And, when the driver got there, “Now take this street five blocks and make a right turn.” And on and on. You know the details, but it is a lot quicker to just tell the driver to take you to the airport.
Abstraction is a technique for establishing a simpler way for a person to interact with a system, removing the details that are unnecessary for the person to interact effectively with that system.

Our ability to abstract is very much a productivity enhancer. It allows us to deal with a situation at a higher level, focusing on the essential aspects, while keeping the component ideas in the background.
It allows us to be more efficient in our use of time and brain activity. It allows us to not get bogged down in the detail when everything about the detail is working just fine.

There is an underlying assumption to this, however: when everything about the detail is just fine. What if everything about the detail is not just fine? Then, to be successful, our ability to abstract must be combined with our ability to un-abstract. Some people use the word deconstruct—the ability to go from the abstraction back to its component parts.

As we will see, modern computers are composed of transistors. These transistors are combined to form logic gates”—an abstraction that lets us think in terms of 0s and 1s instead of the varying voltages on the transistors.
A logic circuit is a further abstraction of a combination of gates. When one designs a logic circuit out of gates, it is much more efficient to not have to think about the internals of each gate. To do so would slow down the process of designing the logic circuit. One wants to think of the gate as a component.
But if there is a problem with getting the logic circuit to work, it is often helpful to look at the internal structure of the gate and see if something about its functioning is causing the problem.

When one designs a sophisticated computer application program, whether it be a new spreadsheet program, word processing system, or computer game, one wants to think of each of the components one is using as an abstraction.
If one spent time thinking about the details of each component when it was not necessary, the distraction could easily prevent the total job from ever getting finished.
But when there is a problem putting the components together, it is often useful to examine carefully the details of each component in order to uncover the problem.

The ability to abstract is the most important skill. One should try to keep the level of abstraction as high as possible, consistent with getting everything to work effectively. The approach in this course is to continually raise the level of abstraction. We describe logic gates in terms of transistors. Once we understand the abstraction of gates, we no longer think in terms of transistors.
Then we build larger structures out of gates. Once we understand these larger abstractions, we no longer think in terms of gates.

The Bottom Line Abstractions allow us to be much more efficient in dealing with all kinds of situations. It is also true that one can be effective without understanding what is below the abstraction as long as everything behaves nicely.
So, one should not pooh-pooh the notion of abstraction. On the contrary, one should celebrate it since it allows us to be more efficient.
In fact, if we never have to combine a component with anything else into a larger system, and if nothing can go wrong with the component, then it is perfectly fine to understand this component only at the level of its abstraction.
But if we have to combine multiple components into a larger system, we should be careful not to allow their abstractions to be the deepest level of our understanding. If we don't know the components below the level of their abstractions, then we are at the mercy of them working together without our intervention.
If they don't work together, and we are unable to go below the level of abstraction, we are stuck. And that is the state we should take care not to find ourselves in.

### 1.3.2 Hardware vs. Software

Many computer scientists and engineers refer to themselves as hardware people or software people. By hardware, they generally mean the physical computer and all the specifications associated with it.
By software, they generally mean the programs, whether operating systems like Android, ChromeOS, Linux, orWindows, or database systems like Access, MongoDB, Oracle, or DB-terrific, or application programs like Facebook, Chrome, Excel, or Word.
The implication is that the person knows a whole lot about one of these two things and precious little about the other. Usually, there is the further implication that it is OK to be an expert at one of these (hardware OR software) and clueless about the other.

It is as if there were a big wall between the hardware (the computer and how it actually works) and the software (the programs that direct the computer to do their bidding), and that one should be content to remain on one side of that wall or the other.

The power of abstraction allows us to “usually” operate at a level where we do not have to think about the underlying layers all the time. This is a good thing.
It enables us to be more productive. But if we are clueless about the underlying layers, then we are not able to take advantage of the nuances of those underlying layers when it is very important to be able to.

That is not to say that you must work at the lower level of abstraction and not take advantage of the productivity enhancements that the abstractions provide.
On the contrary, you are encouraged to work at the highest level of abstraction available to you. But in doing so, if you are able to, at the same time, keep in mind the underlying levels, you will find yourself able to do a much better job.

As you approach your study and practice of computing, we urge you to take the approach that hardware and software are names for components of two parts of a computing system that work best when they are designed by people who take into account the capabilities and limitations of both.

Microprocessor designers who understand the needs of the programs that will execute on the microprocessor they are designing can design much more effective microprocessors than those who don’t.
For example, Intel, AMD, ARM, and other major producers of microprocessors recognized a few years ago that a large fraction of future programs would contain video clips as part of email, video games, and full-length movies.
They recognized that it would be important for such programs to execute efficiently. The result: most microprocessors today contain special hardware capability to process those video clips.
Intel defined additional instructions, initially called their MMX instruction set, and developed special hardware for it.
Motorola, IBM, and Apple did essentially the same thing, resulting in the AltiVec instruction set and special hardware to support it.

A similar story can be told about software designers.
The designer of a large computer program who understands the capabilities and limitations of the hardware that will carry out the tasks of that program can design the program so it executes more efficiently than the designer who does not understand the nature of the hardware.

One important task that almost all large software systems need to carry out is called sorting, where a number of items have to be arranged in some order. The words in a dictionary are arranged in alphabetical order.

Students in a class are often graded based on a numerical order, according to their scores on the final exam. There are a large number of fundamentally different programs one can write to arrange a collection of items in order.
Which sorting program works best is often very dependent on how much the software designer is aware of the underlying characteristics of the hardware.

**The Bottom Line** We believe that whether your inclinations are in the direction of a computer hardware career or a computer software career, you will be much more capable if you master both.
This course is about getting you started on the path to mastering both hardware and software. Although we sometimes ignore making the point explicitly when we are in the trenches of working through a concept, it really is the case that each sheds light on the other.

When you study data types, a software concept, in C (Season 12), you will understand how the finite word length of the computer, a hardware concept, affects our notion of data types.

When you study functions in C (Season 14), you will be able to tie the rules of calling a function with the hardware implementation that makes those rules necessary.

When you study recursion, a powerful algorithmic device (initially in Season 8 and more extensively in Season 17), you will be able to tie it to the hardware. If you take the time to do that, you will better understand when the additional time to execute a procedure recursively is worth it.

When you study pointer variables in C (in Season 16), your knowledge of computer memory will provide a deeper understanding of what pointers provide, and very importantly, when they should be used and when they should be avoided.

When you study data structures in C (in Season 19), your knowledge of computer memory will help you better understand what must be done to manipulate the actual structures in memory efficiently.

We realize that most of the terms in the preceding five short paragraphs may not be familiar to you yet. That is OK; you can reread this page at the end of the course.

What is important to know right now is that there are important topics in the software that are very deeply interwoven with topics in the hardware.

Our contention is that mastering either is easier if you pay attention to both. Most importantly, most computing problems yield better solutions when the problem solver has the capability of both at his or her disposal.

# Computer Organization | Von Neumann architecture

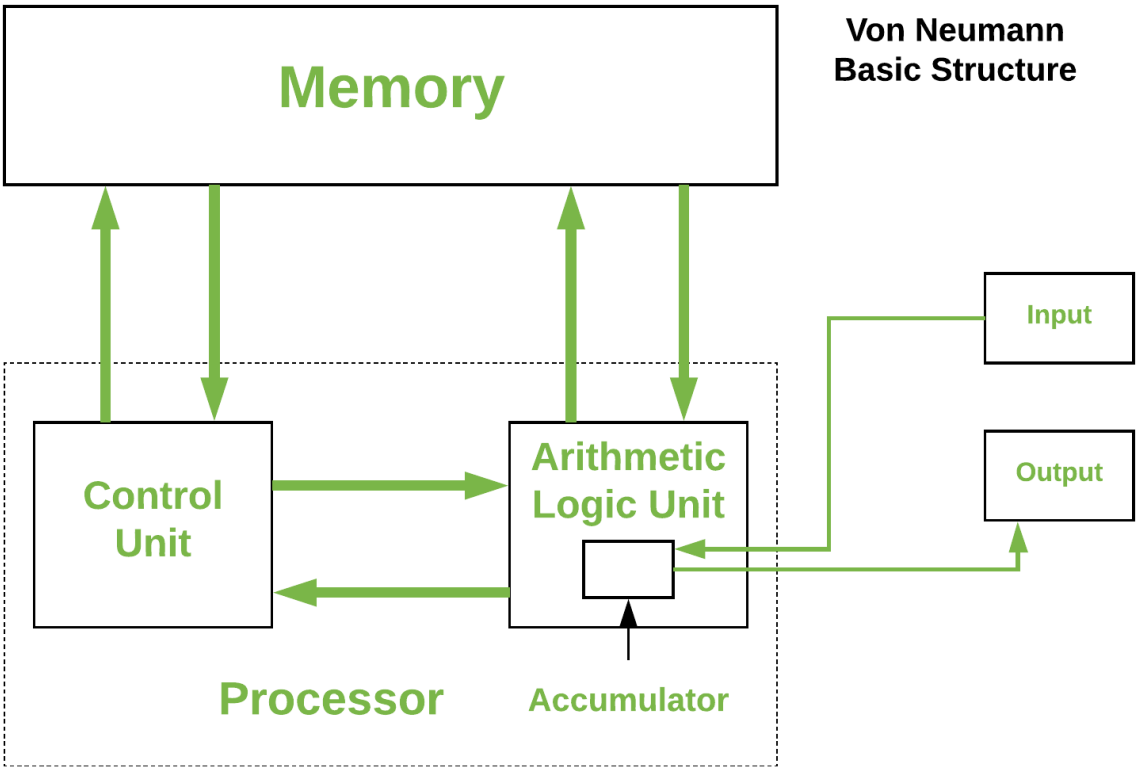
Von-Neumann  computer architecture design was proposed in 1945.It was later known as Von-Neumann architecture.

Historically there have been 2 types of Computers:

1. **Fixed Program Computers** – Their function is very specific and they couldn't be reprogrammed, e.g. Calculators.
2. **Stored Program Computers** – These can be programmed to carry out many different tasks, applications are stored on them, hence the name.

Modern computers are based on a stored-program concept introduced by John Von Neumann. In this stored-program concept, programs and data are stored in a separate storage unit called memories and are treated the same. This novel idea meant that a computer built with this architecture would be much easier to reprogram.

The basic structure is like this,




It is also known as **ISA** (Instruction set architecture) computer and is having three basic units:

1. The Central Processing Unit (CPU)
2. The Main Memory Unit
3. The Input/Output Device Let's consider them in detail.

**1. Central Processing Unit-**

**The central processing unit is defined as the it is an electric circuit used for executing the instruction of a computer program.**

**It has following major components:**

**1.Control Unit(CU)**

**2.Arithmetic and Logic Unit(ALU)**

**3.variety of Registers**

- **Control Unit** –

  A control unit (CU) handles all processor control signals. It directs all input and output flow, fetches code for instructions, and controls how data moves around the system.

- **Arithmetic and Logic Unit (ALU)** –

  The arithmetic logic unit is that part of the CPU that handles all the calculations the CPU may need, e.g. Addition, Subtraction, Comparisons. It performs Logical Operations, Bit Shifting Operations,

and Arithmetic operations.

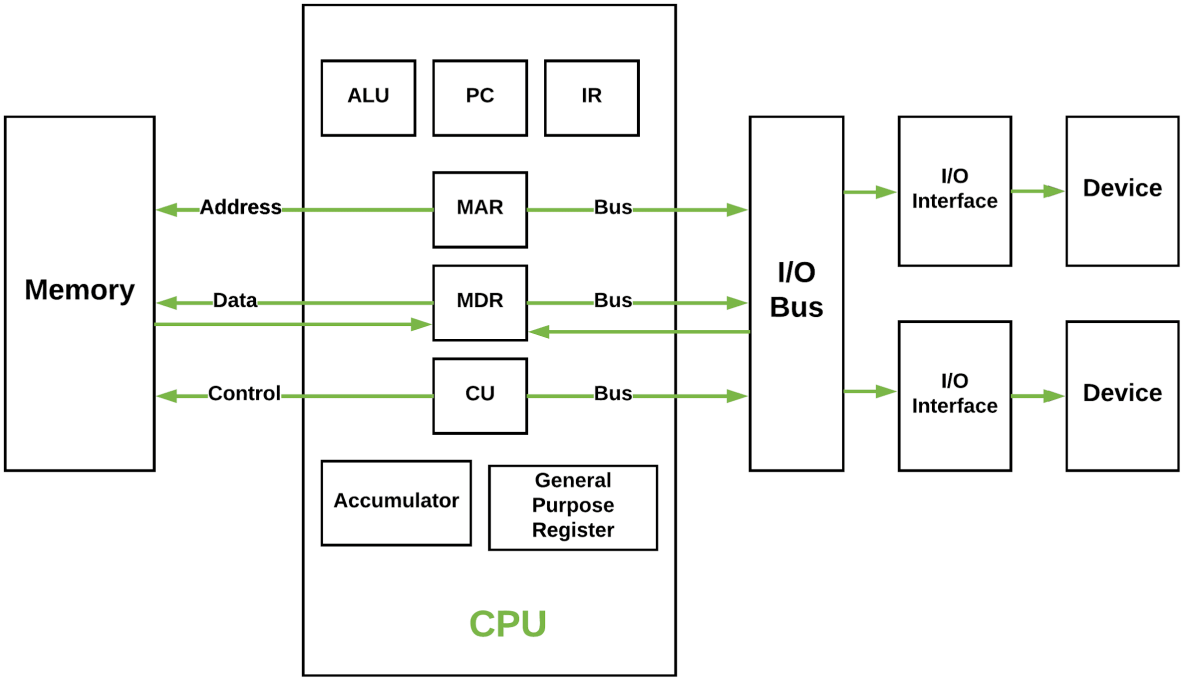



**Figure** – Basic CPU structure, illustrating ALU

- **Registers** – Registers refer to high-speed storage areas in the CPU. The data processed by the CPU are fetched from the registers. There are different types of registers used in architecture :-
    1. **Accumulator:** Stores the results of calculations made by ALU. It holds the intermediate of arithmetic and logical operatoins.it act as  a temporary storage location or device.
    2. **Program Counter (PC):** Keeps track of the memory location of the next instructions to be dealt with. The PC then passes this next address to the Memory Address Register (MAR).

3. **Memory Address Register (MAR):** It stores the memory locations of instructions that need to be fetched from memory or stored in memory.
4. **Memory Data Register (MDR):** It stores instructions fetched from memory or any data that is to be transferred to, and stored in, memory.
5. **Current Instruction Register (CIR):** It stores the most recently fetched instructions while it is waiting to be coded and executed.
6. **Instruction Buffer Register (IBR):** The instruction that is not to be executed immediately is placed in the instruction buffer register IBR.

- **Buses** – Data is transmitted from one part of a computer to another, connecting all major internal components to the CPU and memory, by the means of Buses. Types:
    1. **Data Bus:** It carries data among the memory unit, the I/O devices, and the processor.
    2. **Address Bus:** It carries the address of data (not the actual data) between memory and processor.
    3. **Control Bus:** It carries control commands from the CPU (and status signals from other devices) in order to control and coordinate all the activities within the computer**.**

- **Input/Output Devices** – Program or data is read into main memory from the *input device* or secondary storage under the control of CPU input instruction. *Output devices* are used to output information from a computer. If some results are evaluated by the computer and it is stored in the computer, then with the help of output devices, we can present them to the user.

**Von Neumann bottleneck** –

Whatever we do to enhance performance, we cannot get away from the fact that instructions can only be done one at a time and can only be carried out sequentially. Both of these factors hold back the competence of the CPU. This is commonly referred to as the 'Von Neumann bottleneck'. We can provide a Von Neumann processor with more cache, more RAM, or faster components but if original gains are to be made in CPU performance then an influential inspection needs to take place of CPU configuration.

This architecture is very important and is used in our PCs and even in Supercomputers.