EE 1193

INTRODUCTION TO HDL

MODULE-1: OVERVIEW

Dr. Amit Acharyya, IITH

To begin with…

▸ introduces the course with a brief history of computer
architecture, and progresses through the basic parts of a
computer, leaving you with a high level view of a
computer system.

▸ The conventional von Neumann model of a digital
computer is introduced, followed by the System Bus
Model, followed by a topical exploration of a typical
computer.

‣ Computer **architecture deals with the functional behavior of a computer system** as viewed by a programmer. This view includes aspects such as the sizes of data types (e.g. *using 16 binary digits to represent an integer), and the types of operations* that are supported (like addition, subtraction, and subroutine calls).

‣ Computer **organization deals with structural relationships that are not visible to the** programmer, such as interfaces to peripheral devices, the clock frequency, and the technology used for the memory.

Overview

‣ There is a concept of **levels in computer architecture. The basic idea is that there**

are many levels, or views, at which a computer can be considered, from the highest level, where the user is running programs, or *using the computer, to the lowest*

level, consisting of transistors and wires. Between the high and low levels are a number of intermediate levels.

A Brief History

▸ since at least the 1500's, when rotating pegged cylinders were used in music boxes much as they are today.

▸ Machines that perform calculations, as opposed to   simply repeating a predetermined melody, came in the next century.

▸ B. Pascal (1623 – 1662) developed a mechanical calculator to help in his father's tax work. The Pascal calculator "Pascaline" contains eight dials that connect to a drum, with an innovative linkage that causes a dial to rotate one notch when a carry is produced from a dial in a lower position. A window is placed over the dial to allow its position to be observed, much like the odometer in a car except that the dials are positioned horizontally, like a rotary telephone dial. Some of Pascal's adding machines, which he started to build in 1642, still exist today. It would not be until the 1800's, however, until someone would put the concepts of mechanical control and mechanical calculation together into a machine that we recognize today as having the basic parts of a

▸ digital computer. That person was Charles Babbage.

▸ Charles Babbage (1791 – 1871) is sometimes referred to as the *grandfather of the* computer, rather than the father of the computer, because he never built a practical version of the machines he designed. Babbage lived in England at a time when mathematical tables were used in navigation and scientific work. The tables were computed manually, and as a result, they contained numerous errors. Frustrated by the inaccuracies, Babbage set out to create a machine that would compute tables by simply setting and turning gears. The machine he designed could even produce a plate to be used by a printer, thus eliminating errors that might be introduced by a typesetter.

▸ Babbage's machines had a means for reading input data, storing data, performing calculations, producing output data, and automatically controlling the operation of the machine. These are basic functions that are found in nearly every modern computer. Babbage created a small prototype of his **difference engine, which** evaluates polynomials using the method of finite differences. The success of the difference engine concept gained him government support for the much larger **analytical engine, which was a more sophisticated machine that had a mechanism** for **branching (making decisions) and a means for programming, using** punched cards in the manner of what is known as the **Jacquard pattern-weaving loom.**

The analytical engine was designed, but was never built by Babbage because the mechanical tolerances required by the design could not be met with the technology of the day.

A version of Babbage's difference engine was actually built by the Science Museum in London in 1991, and can still be viewed today.

▸ It took over a century, until the start of World War II, before the next major thrust in computing was initiated. In England, German **U-boat submarines were** inflicting heavy damage on Allied shipping. The U-boats received communications from their bases in Germany using an encryption code, which was implemented by a machine made by Siemens AG known as **ENIGMA.**

▸ The process of encrypting information had been known for a long time, and even the United States president Thomas Jefferson (1743 – 1826) designed a forerunner of ENIGMA, though he did not construct the machine. The process of decoding encrypted data was a much harder task. It was this problem that prompted the efforts of Alan Turing (1912 – 1954), and other scientists in England in creating code breaking machines. During World War II, Turing was the leading cryptographer in England and was among those who changed cryptography from a subject for people who deciphered ancient languages to a subject for mathematicians.

- The **Colossus was a successful code breaking machine that came out of Bletchley** Park, England, where Turing worked.
- Vacuum tubes store the contents of a paper tape that is fed into the machine, and computations take place among the vacuum tubes and a second tape that is fed into the machine. Programming is performed with plugboards.
- Turing's involvement in the various Collosi machine versions remains obscure due to the secrecy that surrounds the project, but some aspects of his work and his life can be seen in the Broadway play *Breaking the Code which was performed in London and New York in the late 1980's.*

▸ Around the same time as Turing's efforts, J. Presper Eckert and John Mauchly set out to create a machine that could be used to compute tables of ballistic trajectories for the U.S. Army.

The result of the Eckert-Mauchly effort was the Electronic Numerical Integrator And Computer (**ENIAC).**

The ENIAC consists of 18,000 vacuum tubes, which make up the computing section of the machine.

▸ Programming and data entry are performed by setting switches and changing cables. There is no concept of a stored program, and there is no central memory unit, but these are not serious limitations because all that the ENIAC needed to do was to compute ballistic trajectories. Even though it did not become operational until 1946, after the War was over, it was considered quite a success, and was used for nine years.

- After the success of ENIAC, Eckert and Mauchly, who were at the Moore School at the University of Pennsylvania, were joined by John von Neumann (1903 –1957), who was at the Institute for Advanced Study at Princeton. Together, they

- worked on the design of a stored program computer called the **EDVAC.**

- **A conflict** developed, however, and the Pennsylvania and Princeton groups split. The concept of a stored program computer thrived, however, and a working model of

- the stored program computer, the **EDSAC, was constructed by Maurice Wilkes,** of Cambridge University, in 1947

The Von Neumann Model

▸ Conventional digital computers have a common form that is attributed to von

Neumann, although historians agree that the entire team was responsible for the design.

The **von Neumann model consists of five major components as illustrated**

in Figure.

The **Input Unit provides instructions and data to the sys**tem, which are subsequently stored in the **Memory Unit. The instructions and** data are processed by the **Arithmetic and Logic Unit (ALU) under the direction** of the **Control Unit. The results are sent to the Output Unit.**

The ALU and control unit are frequently referred to collectively as the **central processing unit (CPU).**

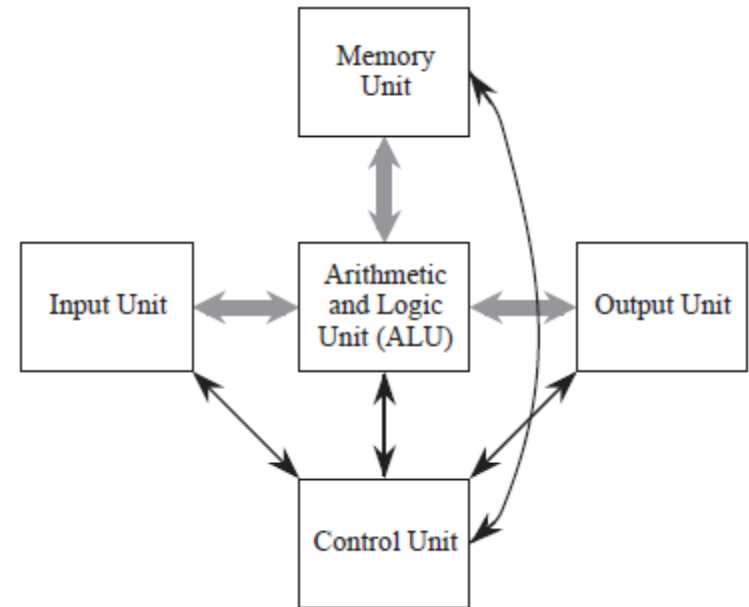**Most commercial computers can be decomposed into these five basic** units.

Figure : The von Neumann model of a digital computer. Thick arrows represent data paths. Thin
arrows represent control paths.

- The **stored program is the most important aspect of the von Neumann model.**

- A program is stored in the computer's memory along with the data to be processed.
- Although we now take this for granted, prior to the development of the stored program computer programs were stored on external media, such as plugboards or punched cards or tape.

- In the stored program computer the program can be manipulated as if it is data. This gave rise to compilers
- and operating systems, and makes possible the great versatility of the modern computer.

The System Bus Model

▸ Although the von Neumann model prevails in modern computers, it has been streamlined. Figure below shows the **system bus model of a computer system. This** model partitions a computer system into three subunits: CPU, Memory, and Input/Output (I/O). This refinement of the von Neumann model combines the ALU and the control unit into one functional unit, the CPU. The input and output units are also combined into a single I/O unit.
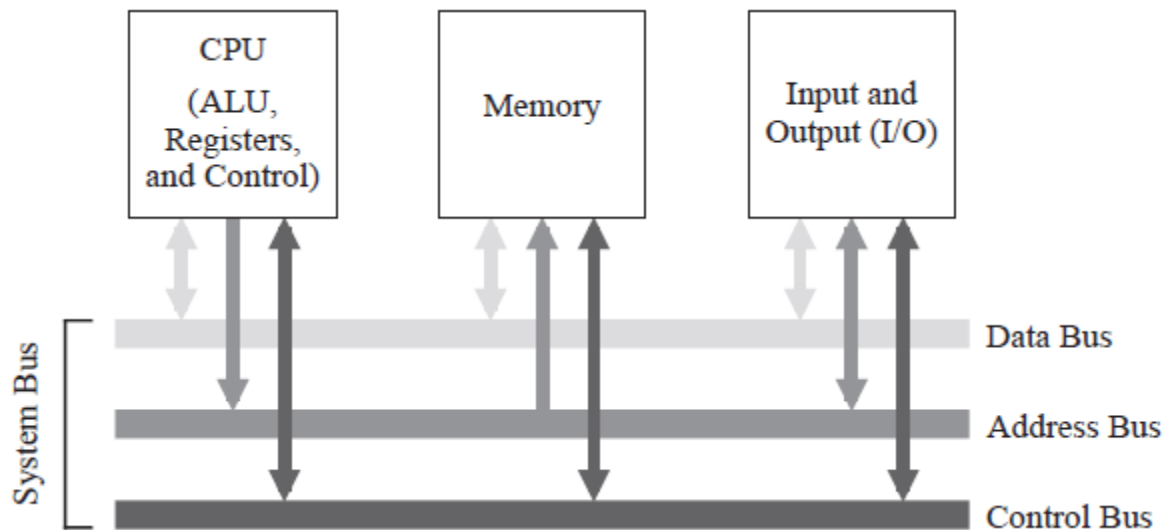

Figure: The system bus model of a computer system.

▶ Most important to the system bus model, the communications among the components are by means of a shared pathway called the **system bus, which is made** up of the **data bus (which carries the information being transmitted), the address bus (which identifies where the information is being sent), and the control bus (which describes aspects of how the information is being sent, and in** what manner).

▶ There is also a **power bus for electrical power to the components,** which is not shown, but its presence is understood.

▶ Some architectures may also have a separate I/O bus.

▶ Physically, busses are made up of collections of wires that are grouped by function.

A 32-bit data bus has 32 individual wires, each of which carries one bit of data (as opposed to address or control information). In this sense, the system bus is actually a group of individual busses classified by their function.

- The data bus moves data among the system components. Some systems have separate data buses for moving information to and from the CPU, in which case there is a **data-in bus and a data-out bus. More often a single data bus moves** data in either direction, although never both directions at the same time.

- If the bus is to be shared among communicating entities, then the entities must have distinguished identities: addresses.

- In some computers all addresses are assumed to be memory addresses whether they are in fact part of the computer's memory, or are actually I/O devices, while in others I/O devices have separate I/O addresses.

▸ A **memory address, or location, identifies a memory location where data is** stored, similar to the way a postal address identifies the location where a recipient receives and sends mail.

▸ During a memory read or write operation the address bus contains the address of the memory location where the data is to be read from or written to.

▸ Note that the terms "read" and "write" are with respect to the CPU: the CPU *reads data from memory and writes data into memory.*

▸ *If data is* to be read from memory then the data bus contains the value read from that address in memory. If the data is to be written into memory then the data bus

▸ contains the data value to be written into memory.

▸ The control bus is somewhat more complex, and we defer discussion of this bus to later modules. For now the control bus can be thought of as coordinating access to the data bus and to the address bus, and directing data to specific components.

Levels of Machines

▸ As with any complex system, the computer can be viewed from a number of perspectives or levels, from the highest "user" level to the lowest, transistor level.

▸ Each of these levels represents an abstraction of the computer. Perhaps one of the reasons for the enormous success of the digital computer is the extent to which these levels of abstraction are separate, or independent from one another.

▸ This is readily seen: a user who runs a word processing program on a computer needs to know nothing about its programming. Likewise a programmer need not be concerned with the logic gate structure inside the computer.

▸ One interesting way that the separation of levels has been exploited is in the development of upwardly-compatible machines.

UPWARD COMPATIBILITY

▸ The invention of the transistor led to a rapid development of computer hardware,
and with this development came a problem of compatibility.

▸ Computer users wanted to take advantage of the newest and fastest machines, but each new computer model had a new architecture, and the old software would not run on the new hardware. The hardware / software compatibility problem became so serious that users often delayed purchasing a new machine because of the cost of rewriting the software to run on the new hardware.

▸ When a new computer was purchased, it would often sit unavailable to the target users for months while the old software and data sets were converted to the new systems.

▸ In a successful gamble that pitted compatibility against performance, IBM pioneered the concept of a "family of machines" with its 360 series. More capable machines in the same family could run programs written for less capable machines without modifications to those programs—upward compatibility.

▸ Upward compatibility allows a user to upgrade to a faster, more capable machine without rewriting the software that runs on the less capable model.

THE LEVELS

▶ Figure shows seven levels in the computer, from the user level down to the transistor level. As we progress from the top level downward, the levels become less "abstract" and more of the internal structure of the computer shows through.
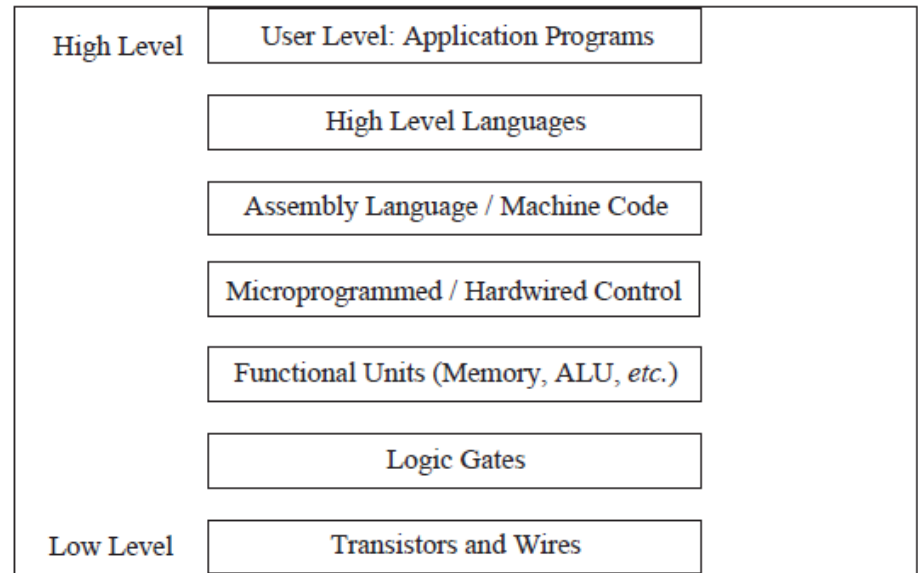
High Level	User Level: Application Programs
	High Level Languages
	Assembly Language / Machine Code
	Microprogrammed / Hardwired Control
	Functional Units (Memory, ALU, *etc.*)
	Logic Gates
Low Level	Transistors and Wires

Figure: Levels of machines in the computer hierarchy.

▶ We discuss these levels next.

User or Application-Program Level

- We are most familiar with the user, or application program level of the computer.

- At this level, the user interacts with the computer by running programs such as word processors, spreadsheet programs, or games.

- Here the user sees the computer through the programs that run on it, and little (if any) of its internal or lower-level structure is visible.

High Level Language Level

▸ Anyone who has programmed a computer in a high level language such as C, Pascal, Fortran, or Java, has interacted with the computer at this level. Here, a programmer sees only the language, and none of the low-level details of the machine.

▸ At this level the programmer sees the data types and instructions of the high-level language, but needs no knowledge of how those data types are actually implemented in the machine. It is the role of the **compiler to map data types and** instructions from the high-level language to the actual computer hardware.

▸ Programs written in a high-level language can be re-compiled for various machines that will (hopefully) run the same and provide the same results regardless of which machine on which they are compiled and run.

▸ We can say that programs are compatible across machine types if written in a high-level language, and this kind of compatibility is referred to as **source code compatibility.**

Assembly Language/Machine Code Level

▸ As pointed out before, the high-level language level really has little to do with the machine on which the high-level language is translated. The compiler translates the source code to the actual machine instructions, sometimes referred to as **machine language or machine code.**

▸ **High-level languages "cater" to the programmer** by providing a certain set of presumably well-thought-out language constructs and data types.

▸ Machine languages look "downward" in the hierarchy, and thus cater to the needs of the lower level aspects of the machine design. As a result, machine languages deal with hardware issues such as registers and the transfer of data between them. In fact, many machine instructions can be described in terms of the register transfers that they effect.

▸ The collection of machine instructions for a given machine is referred to as the **instruction set of** that machine. Of course, the actual machine code is just a collection of 1's and 0's, sometimes referred to as **machine binary code, or just binary code. As we might imagine,** programming with 1's and 0's is tedious and error prone.

▸ As a result, one of the first computer programs written was the **assembler, which translates ordinary** language **mnemonics such as MOVE Data etc. into their corresponding** machine language 1's and 0's. This language, whose constructs bear a one-to-one relationship to machine language, is known as **assembly language.**

- As a result of the separation of levels, it is possible to have many different machines that differ in the lower-level implementation but which have the same instruction set, or sub- or supersets of that instruction set.

- This allowed IBM to design a product line such as the **IBM 360 series with guaranteed upward compatibility** of machine code. Machine code running on the 360 Model 35 would run unchanged on the 360 Model 50, should the customer wish to upgrade to the more powerful machine.

- This kind of compatibility is known as "**binary compatibility**," because the binary code will run unchanged on the various family members. This feature was responsible in large part for the great success of the IBM 360 series of computers.

- **Intel Corporation has stressed binary compatibility in its family members. In** this case, binaries written for the original member of a family, such as the 8086, will run unchanged on all subsequent family members, such as the 80186, 80286, 80386, 80486, and the most current family member, the Pentium processor.

- Of course this does not address the fact that there are other computers that present different instruction sets to the users, which makes it difficult to port an installed base of software from one family of computers to another.

The Control Level

▸ It is the **control unit that effects the register transfers described above.**

▸ **It does so** by means of **control signals that transfer the data from register to register, possibly** through a logic circuit that transforms it in some way.

▸ The control unit interprets the machine instructions one by one, causing the specified register transfer or other action to occur.

▸ How it does this is of no need of concern to the assembly language programmer.

▸ The Intel 80x86 family of processors presents the same behavioral view to an assembly language programmer regardless of which processor in the family is considered. This is because each future member of the family is designed to execute the original 8086 instructions in addition to any new instructions implemented for that particular family member.

▶ As the last Figure indicates, there are several ways of implementing the control unit.

▶ Probably the most popular way at the present time is by "hardwiring" the control unit.

▶ This means that the control signals that effect the register transfers are generated from a block of digital logic components.

▶ Hardwired control units have the advantages of speed and component count, but until recently were exceedingly difficult to design and modify.

▶ A somewhat slower but simpler approach is to implement the instructions as a **micro-program.**

▶ **A micro-program is actually a small program written in an even** lower-level language, and implemented in the hardware, whose job is to interpret the machine-language instructions.

▶ This micro-program is referred to as **firmware** because it spans both hardware and software. Firmware is executed by a **microcontroller,** which executes the actual microinstructions.

Functional Unit Level

▸ The register transfers and other operations implemented by the control unit move data in and out of "functional units," so-called because they perform some function that is important to the operation of the computer.

▸ Functional units include internal CPU registers, the ALU, and the computer's main memory.

Logic Gates, Transistors, and Wires

▸ The lowest levels at which any semblance of the computer's higher-level functioning is visible is at the **logic gate and transistor levels.**

▸ **It is from logic gates** that the functional units are built, and from transistors that logic gates are built.

▸ The logic gates implement the lowest-level logical operations upon which the computer's functioning depends.

▸ At the very lowest level, a computer consists of electrical components such as transistors and wires, which make up the logic gates, but at this level the functioning of the computer is lost in details of voltage, current, signal propagation delays, quantum effects, and other low-level matters. [Digital IC Design and Verification/ Embedded Systems Design]

Interactions Between Levels

▸ The distinctions within levels and between levels are frequently blurred.

▸ For instance, a new computer architecture may contain floating point instructions in a full-blown implementation, but a minimal implementation may have only enough hardware for integer instructions.

▸ The floating point instructions are **trapped prior to execution and replaced with a sequence of machine language** instructions that imitate, or **emulate the floating point instructions using the** existing integer instructions.

▸ This is the case for **microprocessors that use** optional floating point coprocessors.

▸ Those without floating point coprocessors emulate the floating point instructions by a series of floating point routines that are implemented in the machine language of the microprocessor, and frequently stored in a **ROM, which is a read-only memory chip.**

▸ **The assembly language and** high level language view for both implementations is the same except for execution speed.

▸ It is possible to take this emulation to the extreme of emulating the entire instruction set of one computer on another computer.

▸ The software that does this is known as an **emulator, and was used by Apple Computer to maintain** binary code compatibility when they began employing Motorola PowerPC chips in place of Motorola 68000 chips, which had an entirely different instruction set.

▸ The high level language level and the firmware and functional unit levels can be so intermixed that it is hard to identify what operation is happening at which level.

▸ Therefore, the value in stratifying a computer architecture into a hierarchy of levels is not so much for the purpose of classification, which we just saw can be difficult at times, but rather to simply give us some focus when we study these levels in next modules in a methodical manner.

The Programmer's View—The Instruction Set Architecture

▸ As described in the discussion of levels above, the assembly language programmer is concerned with the assembly language and functional units of the machine.

▸ This collection of instruction set and functional units is known as the **instruction set architecture (ISA) of the machine.**
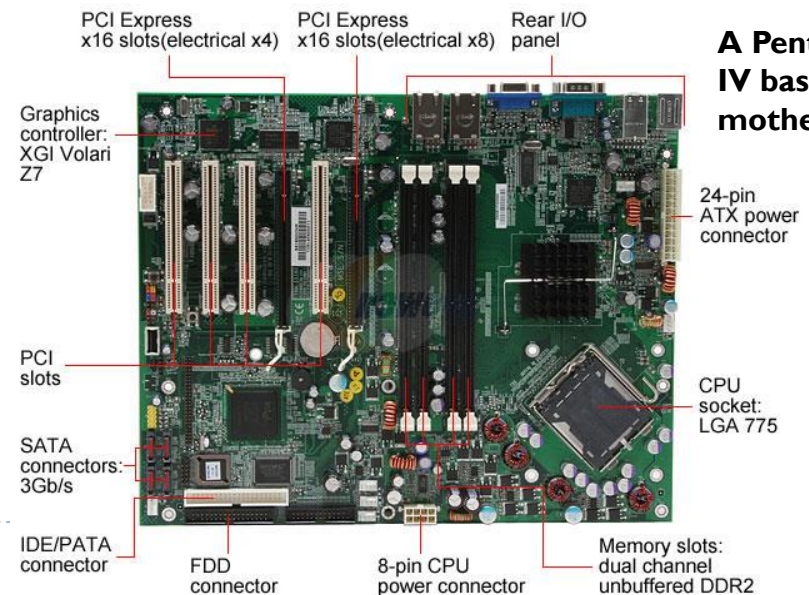
The Computer Architect's View

▸ On the other hand, the computer architect views the system at all levels. The architect that focuses on the design of a computer is invariably driven by performance requirements and cost constraints.

▸ Performance may be specified by the speed of program execution, the storage capacity of the machine, or a number of other parameters.

▸ Cost may be reflected in monetary terms, or in size or weight, or power consumption.

▸ The design proposed by a computer architect must attempt to meet the performance goals while staying within the cost constraints.

▸ This usually requires trade-offs between and among the levels of the machine.

A Typical Computer System

▸ Modern computers have evolved from the great behemoths of the 1950's and 1960's to the much smaller and more powerful computers that surround us today.

▸ Even with all of the great advances in computer technology that have been made in the past few decades, the five basic units of the von Neumann model are still distinguishable in modern computers.

▸ The input unit is composed of the **keyboard, through which a user enters data and commands.**

▸ A **video monitor comprises the output unit, which displays the output in a** visual form.

▸ The ALU and the control unit are bundled into a single microprocessor that serves as the CPU.

▸ The memory unit consists of individual memory circuits, and also a **hard disk unit, a diskette unit, and a CD-ROM (compact** disk - read only memory) device.

▸ As we look deeper inside of the machine, we can see that the heart of the machine is contained on a single **motherboard.** The motherboard contains **integrated circuits (ICs), plug-in expansion** card slots, and the wires that interconnect the ICs and expansion card slots.
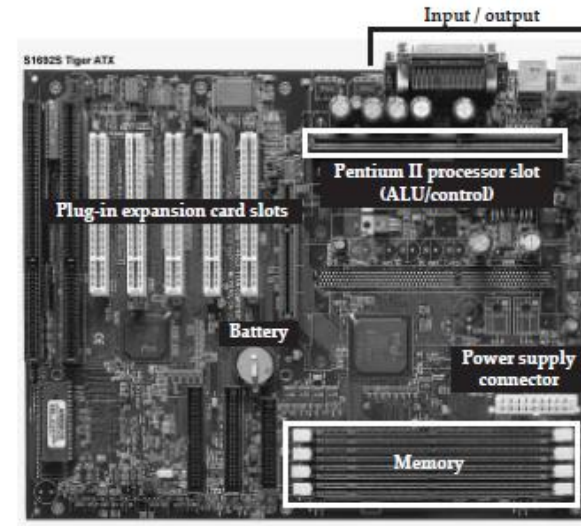
A Pentium II based motherboard

A Pentium IV based motherboard

Summary of Module-1

▸ *Computer architecture deals with those aspects of a computer that are visible to a programmer, while computer organization deals with those aspects that are at a more physical level and are not made visible to a programmer.*

▸ *Historically, programmers had to deal with every aspect of a computer – Babbage with mechanical gears, and ENIAC programmers with plugboard cables.*

▸ *As computers grew in sophistication, the concept of levels of machines became more pronounced, allowing computers to have very different internal and external behaviors while managing complexity in stratified levels.*

▸ *The single most significant development that makes this possible is the stored program computer, which is embodied in the von Neumann model.*

▸ *It is the von Neumann model that we see in most conventional computers today.*