COMPUTER ARCHITECTURE

Introduction to Computer Architecture

- Computer architecture defines how a computer's components communicate through electronic signals to perform input, processing, and output operations.
- It covers the design and organization of the CPU, memory, storage, and input/output devices.
- Describes how these components interact through buses, control signals, and data pathways.
- It directly influences the overall speed, functionality, and reliability of a computer system.

- Key Components
- Instruction Set Architecture (ISA): The programmer's view, defining instructions, registers, data types, and memory addressing.
- CPU (Central Processing Unit): The brain, handling execution; its organization (ALU, registers, control unit) is key.
- Memory: Stores data and instructions (RAM, cache, registers).
- Input/Output (I/O) Subsystems: How the computer communicates with the outside world (keyboard, screen, storage).
- Buses & Interconnects: Pathways for data and control signals between components.

Importance in software development.

- 1. Optimizing Code Performance: Knowing Your Machine Inside Out
- 2. Improved Debugging and Problem Solving
- 3. Mastering Low-Level Programming: More Control, More Possibilities
- 4. Leveraging Multi-Core Processors: Unlocking the Power of Parallelism
- 5. Building Portable and Flexible Code: Ensuring Compatibility Across Platforms

Historical development of computer architectures

Precursors (1800s): Charles Babbage's Analytical Engine conceptualized programmable computation, while Ada Lovelace wrote early algorithms.

- First Generation (1940s-1950s):

Tech: Vacuum tubes (ENIAC, UNIVAC).

Arch: Von Neumann Architecture (stored programs in memory) became foundational.

Programs: Machine code, batch processing.

- Second Generation (1950s-1960s):

Tech: Transistors replaced tubes, making machines smaller, faster, and reliable.

Programs: Assembly language, first high-level languages, early OS.

- Third Generation (1960s-1970s):
- Tech: Integrated Circuits (ICs) combined many transistors on a chip (SSI, MSI).
- Arch: Multiprogramming emerged.
- Fourth Generation (1971-Present):
- Tech: Microprocessors (CPU on one chip), VLSI/ULSI (millions of transistors).
- Arch: Personal computers, RISC/CISC instruction sets, pipelining, caching, parallel processing (multicore, GPUs).

RISC vs. CISC architectures

Reduced Instruction Set Architecture (RISC)

- RISC simplifies processor design by using a small, uniform set of instructions. Each instruction performs a basic operation (e.g., load, compute, store) and is designed to execute in a single clock cycle, enabling efficient pipelining and simpler hardware.

Characteristics of RISC

- Simpler instruction, hence simple instruction decoding.
- Instruction comes in the form of one word.
- An instruction takes a single clock cycle to get executed.
- More general-purpose registers for register-to-register operations.
- Simple Addressing Modes.
- Optimized for pipelining due to uniform instruction size and simplicity.

Complex Instruction Set Architecture (CISC)

- CISC reduces the number of instructions a program needs by using a large set of complex, variable-length instructions. A single instruction can perform multiple operations (e.g., load, compute, and store), which may take multiple clock cycles.

Characteristics of CISC

- Complex instruction, hence complex instruction decoding.
- Instructions are larger than one-word size.
- Instruction may take more than a single clock cycle to get executed.
- Less number of general-purpose registers as operations get performed in memory itself.
- Complex Addressing Modes.

Pipeline processing and superscalar

- **Pipelining** and **superscalar** execution are CPU techniques to boost performance by overlapping instruction processing, but they differ: **Pipelining** splits one instruction into stages (Fetch, Decode, Execute, etc.) to process different parts of *multiple* instructions concurrently using *one* set of hardware, like an assembly line.
- **Superscalar** takes this further, using *multiple* parallel pipelines and *separate* hardware units (ALUs, etc.) to fetch, decode, and execute *several independent instructions simultaneously* in the same clock cycle, exploiting instruction-level parallelism. Think of pipelining as one assembly line for one product, while superscalar is having *multiple identical assembly lines* running at once for different products

Computer arithmetic and ALU design.

- Computer arithmetic & ALU design involves creating digital circuits (ALUs) within CPUs/GPUs to execute binary math (add, subtract) & logic (AND, OR, XOR) using transistor-based combinational logic, scaled up from basic 1-bit blocks with select lines for different operations, driving processor performance by handling complex calculations, floating-point, and bitwise functions, forming the core data processor alongside control units.

Virtual memory and paging systems.

- Virtual memory is a memory management technique used by operating systems to give the appearance of a large, continuous block of memory to applications, even if the physical memory (RAM) is limited and not necessarily allocated in contiguous manner. The main idea is to divide the process in pages, use disk space to move out the pages if space in main memory is required and bring back the pages when needed.

How Virtual Memory Works

- Virtual memory uses both hardware and software to manage memory.
- When a program runs, it uses virtual addresses (not real memory locations).
- The computer system converts these virtual addresses into physical addresses (actual locations in RAM) while the program runs.

Types of Virtual Memory

- In a computer, virtual memory is managed by the Memory Management Unit (MMU), which is often built into the CPU. The CPU generates virtual addresses that the MMU translates into physical addresses. There are two main types of virtual memory:
- Paging
- Segmentation

Paging

- Paging divides memory into small fixed-size blocks called pages. When the computer runs out of RAM, pages that aren't currently in use are moved to the hard drive, into an area called a swap file. Here,
- The swap file acts as an extension of RAM.
- When a page is needed again, it is swapped back into RAM, a process known as page swapping.
- This ensures that the operating system (OS) and applications have enough memory to run.

Page Fault Service Time: The time taken to service the page fault is called page fault service time. The page fault service time includes the time taken to perform all the above six steps.

- Let Main memory access time is: m*m*
- Page fault service time is: ss
- Page fault rate is : p*p*
- Then, Effective memory access time = (p∗s)+(1−p)∗m(*p*∗s)+(1−*p*)∗*m*

Benchmarking and performance

- Benchmark performance refers to the use of a set of integer and floating-point programs (known collectively as a benchmark) that are designed to test different performance aspects of the computing system(s) under test. Benchmark programs should be designed to provide fair and effective comparisons among high-performance computing systems. For a benchmark to be meaningful, it should evaluate faithfully the performance for the intended use of the system.

CPU Components

- **Control Unit:** The control unit manages the CPU by sending signals like clock, hold, and reset to its parts. It ensures all components work together to complete tasks. For example, it synchronizes data movement from cache memory to the ALU.
- **Arithmetic and Logic Unit (ALU)**: The ALU handles arithmetic tasks (like addition, subtraction, multiplication, division) and logical tasks (like AND, OR, comparisons). It uses addition for all calculations, e.g., solving 2×3 as 2+2+2=6.
- **Memory Unit**: The memory unit stores data and instructions. Older CPUs used registers, but modern ones also have fast cache memory. The CPU fetches data from RAM, ROM, or hard disks and stores it in registers or cache during tasks.

CPU Functions

- **Fetch:** the first CPU gets the instruction. That means binary numbers that are passed from RAM to CPU.
- **Decode:** When the instruction is entered into the CPU, it needs to decode the instructions. with the help of ALU(Arithmetic Logic Unit), the process of decoding begins.
- **Execute:** After the decode step the instructions are ready to execute.
- **Store:** After the execute step the instructions are ready to store in the memory.

Cpu Performance factors

- Processor
- Storage
- IO Devices
- Memory

Microarchitecture design and Instruction Set Architecture

- **ISA (Instruction Set Architecture)**

ISA is the language of the CPU that tells it what operations it can perform, such as adding numbers, loading data, or jumping to another instruction.
It defines how software communicates with hardware through specific instruction rules and formats. It includes:

Instruction types (ADD, LOAD, JUMP), registers, data types, and memory access

Interrupt handling and system-level communication

- **Microarchitecture**

Microarchitecture (μarch) is the specific hardware design and internal organization of a processor (CPU), detailing how an Instruction Set Architecture (ISA)—what the CPU can do—is actually built and executed, involving components like pipelines, caches, registers, and ALUs to manage data flow and performance.

- **ISA (The "What"):**

The abstract model, programmer's view, specifying instructions, registers, memory addressing. (e.g., `ADD R1, R2`)

- **Microarchitecture (The "How"):**

The physical realization, how those instructions are pipelined, out-of-order executed, and managed on silicon

- Branch prediction

Branch prediction guesses the outcome of a branch (like an if statement) to keep the processor busy, while speculative execution is the actual execution of instructions along that predicted path before the guess is confirmed, saving time if right, but requiring the CPU to discard results if wrong, all to avoid pipeline stalls and boost performance.

- **Speculative execution**

Speculative execution is a technique that allows the processor to execute instructions before knowing if they are needed.

Processor predicts the outcome of a branch instruction and starts executing instructions along the predicted path

Speculative execution is performed in parallel with the evaluation of the branch condition

If the prediction is correct, the speculatively executed instructions are committed, and their results become architecturally visible.

Parallelism at the instruction level (ILP)

Instruction-Level Parallelism (ILP) refers to the capability of a processor to execute multiple instructions at the same time. Instead of running each instruction strictly one after another, ILP uses hardware and compiler techniques to overlap instruction execution wherever dependencies allow.

- Identifies independent instructions and runs them in parallel.
- Works within a single processor, not across multiple cores.
- Basis of modern CPUs with organised instruction scheduling.

CPU cooling and thermal management

- CPU cooling and thermal management are essential processes for dissipating heat generated by processors to maintain performance, prevent damage, and ensure reliability, primarily using techniques like heatsinks, fans, thermal paste, and airflow optimization, with methods ranging from basic air cooling to advanced liquid cooling to handle increasing heat loads.

The future of CPUs: Quantum computing and beyond.

- Quantum computing is at the center of this revolution. Unlike the classical computers using bits (0s and 1s), quantum computer uses computer Qubits, which can be present in several states simultaneously. This allows them to perform complex calculations at unimaginable speed with today's machines. Problems that will take thousands of years to solve classical computers can be completed in minutes on the quantum system. Industries like medical, climate science and cryptography can completely change with this capacity.

Cache memory and coherence

- Cache memory speeds up processors by storing frequently used data close by, but in multi-core systems, cache coherence ensures all cores see a consistent view of shared data, preventing stale data issues through protocols (like MESI) that manage states (Modified, Shared, Invalid) to synchronize updates, using techniques like snooping or directories to keep multiple cache copies consistent.

Cache Coherence

Cache coherence in computer architecture refers to the consistency of shared resource data that is stored in multiple local caches. When clients in a system maintain caches of a shared memory resource, problems with incoherent data can arise, which is especially true for CPUs in multiprocessing system.

Memory Allocation Algorithms (Manual/OS Level)

- These decide how to fit new requests into available memory blocks (the "heap").
- First-Fit: Finds the first free block large enough for the request.
- Best-Fit: Finds the smallest free block that fits, minimizing wasted space but potentially creating tiny, unusable fragments.
- Worst-Fit: Finds the largest free block, leaving a large chunk for future big requests (less common).
- Buddy System: Divides memory into power-of-2 blocks, splitting and merging them as needed.

Advances in solid-state drives (SSDs) technology

- SSD advances focus on **NVMe/PCIe interfaces** for extreme speed, **3D NAND stacking** for massive capacities (even 100TB+), and emerging tech like **Computational Storage** and **ZNS** for AI/data centers, all while maintaining benefits like lower power, better durability, smaller form factors (M.2), and lower costs, pushing SSDs past HDDs for nearly all uses.
- **Interface & Speed (NVMe/PCIe)**
- **NAND Flash Technology (Capacity & Density)**

Error detection and correction codes in memory

- Error detection codes are used to detect the error(s) present in the received data (bit stream). These codes contain some bit(s), which are included (appended) to the original bit stream. These codes detect the error, if it is occurred during transmission of the original data (bit stream).

- Error correction codes are used to correct the error(s) present in the received data (bit stream) so that, we will get the original data. Error correction codes also use the similar strategy of error detection codes.

Memory hierarchy optimization techniques.

- Memory hierarchy optimization involves software/hardware strategies to reduce data access time by maximizing cache hits, using techniques like **loop tiling/interchange**, **data prefetching**, **cache-conscious data structures**, and hardware tuning (cache size/associativity) to bridge the CPU-memory speed gap, ensuring frequently used data stays in faster levels (Registers, Cache, RAM).

- Memory hierarchy optimization involves software/hardware strategies to reduce data access time by maximizing cache hits, using techniques like **loop tiling/interchange**, **data prefetching**, **cache-conscious data structures**, and hardware tuning (cache size/associativity) to bridge the CPU-memory speed gap, ensuring frequently used data stays in faster levels (Registers, Cache, RAM).

- **Software-Level Optimizations (Compiler/Programmer)**
- **Hardware-Level Optimizations (Architectural Design)**
- **Memory-Management-Techniques(Operating System)**

How Data is Stored

- **Binary Format**
- Physical Media
- Storage Types- primary,secondary,cloud storage
- Datastructures & Databases.

How Data is Accessed

- **Direct Connection**
- **Network connection**
- **Protocols**
- **Management software-dbms,file system**
- **Access types-sequential,random**

Memory management in an OS is the crucial process of efficiently allocating, tracking, protecting, and organizing the computer's main memory (RAM) among various running programs, ensuring optimal utilization, performance, and security by handling allocation/deallocation, preventing conflicts, and using techniques like paging, swapping, and virtual memory to extend capacity

- IO systems, or input/output systems, are the hardware and software components that allow a computer to communicate with the outside world, handling data transfer between the system and external devices like keyboards, displays, and storage drives

- CPU, memory, and I/O communicate via the **system bus** (address, data, control lines) using **I/O interfaces/controllers** that bridge speed/format differences, with the CPU issuing commands (read/write/status) to specific device addresses

- DMA: For big data (like file transfers), a peripheral can directly move data to/from RAM without constant CPU involvement, freeing the CPU.

- An I/O (Input/Output) interface protocol defines the rules for communication between a computer's CPU/memory and peripheral devices (like keyboards, printers, disks), acting as a translator for data, control signals, and status, ensuring devices with different speeds/formats work together.

- **I/O Virtualization Techniques**

Software-based Emulation: The hypervisor emulates generic I/O devices for the guest OS.

Paravirtualization-The guest OS is modified to be "virtualization-aware" and collaborates with the hypervisor through a set of specialized software interfaces

- **Future Trends in IO**

Integration of Containers and VMs

Open Hybrid Cloud Solutions-The market is leaning toward open-source virtualization solutions (like Kubernetes, KubeVirt, and KVM)

AI-Driven Management-predicting and preventing problems, and optimizing performance in dynamic virtualized environments.

- Parallel computing is a method of running many calculations simultaneously by breaking a large problem into smaller tasks, solved concurrently by multiple processors or cores to achieve faster results.
- A multi-core CPU has fewer, powerful cores for sequential, general tasks (like running your OS), prioritizing low latency; a GPU has thousands of simpler cores for massive parallel processing, excelling at repetitive tasks like graphics, AI, and big data, prioritizing high throughput, making them great for different workloads.
- The two basic classifications of parallel processing are

SIMD which stands for Single Instruction Multiple Data and MIMD which stands for Multiple Instruction Multiple Data. The use of SIMD enables the processing of many data with a single instruction and is applicable to most operations that are uniform such as image processing.

On the other hand, **MIMD** permits different instructions on different data items to be performed hence making it more versatile for versatile applications, simulation applications, and multitasking applications. Comparing SIMD and MIMD allows the decision on what architecture to use when concentrating on a certain kind of computational problem.

- **FPGA**

FPGA stands for Field Programmable Gate Array, which is a chip that is reprogrammable with a collection of hundreds of thousands of logic gates that are connecting internally together to build a complex digital circuit. Basically, it is an integrated circuit that can be programmed by the user to capture the logic.

- **ASIC**

ASIC stands for Application Specific Integrated Circuit, built especially for a specific application or any purpose. If one compares this with any other device, it is having improved speed. Basically, it is an integrated circuit that is specified for one specific purpose.

Parallel programming faces challenges like **complexity**, **Amdahl's Law**, **scalability**, **load balancing**, **and communication overhead**, while synchronization is a critical *part* of these challenges, specifically dealing with **correct data access**, **race conditions**, **deadlocks**, **and ensuring proper ordering (causality)** for concurrent tasks

Data Parallelism-The **same** operation is applied to all data subsets.

Task Parallelism-Different operations (tasks) are performed simultaneously.