****

**Table of Contents**

[Week 1: Computer Systems Basics and Arithmetic 1](#_gjdgxs)

[Week 2: Basic Structure of Computers 3](#_30j0zll)

[Week 3: Computer Architectures 5](#_1fob9te)

[Week 4: Basic Processing Unit 7](#_3znysh7)

[Week 5: Instruction Set Architecture 9](#_2et92p0)

[Week 6: Memory System 11](#_tyjcwt)

[Week 7: Virtual Memory and Memory Management 13](#_3dy6vkm)

[Week 8: I/O Organization 14](#_1t3h5sf)

[Week 9: Standard I/O Interfaces 16](#_4d34og8)

[Week 10: Pipelining and Parallel Computing 18](#_2s8eyo1)

[Week 11: Data Path and Control Considerations 20](#_17dp8vu)

[Week 12: Assembly Language Programming 22](#_3rdcrjn)

# Week 1: Computer Systems Basics and Arithmetic

**1. Computer History / Generations**

**Computer Generations:**

1. **First Generation (1940-1956): Vacuum Tubes**
   * **Technology**: Vacuum tubes
   * **Examples**: ENIAC, UNIVAC
   * **Characteristics**:
     + Large size, high power consumption
     + Machine language programming
     + Limited to basic calculations and data processing
2. **Second Generation (1956-1963): Transistors**
   * **Technology**: Transistors
   * **Examples**: IBM 7094, CDC 1604
   * **Characteristics**:
     + Smaller, more reliable, and more energy-efficient than vacuum tubes
     + Assembly language and high-level programming languages like FORTRAN and COBOL
3. **Third Generation (1964-1971): Integrated Circuits**
   * **Technology**: Integrated Circuits (ICs)
   * **Examples**: IBM 360 series, PDP-8
   * **Characteristics**:
     + Increased reliability and performance
     + Introduction of operating systems
     + Multiprogramming capabilities
4. **Fourth Generation (1971-Present): Microprocessors**
   * **Technology**: Microprocessors
   * **Examples**: Intel 4004, IBM PC
   * **Characteristics**:
     + Personal computers and advanced operating systems
     + Large-scale integration (LSI) and very large-scale integration (VLSI)
     + Graphical user interfaces (GUIs), networks, and internet
5. **Fifth Generation (Present and Beyond): Artificial Intelligence**
   * **Technology**: AI, Quantum Computing, Nanotechnology
   * **Examples**: IBM Watson, Google DeepMind
   * **Characteristics**:
     + Development of AI and machine learning
     + Natural language processing
     + Parallel processing and supercomputers

**2. Computer Arithmetic and Number Systems**

**Number Systems:**

1. **Binary System**:
   * Base-2 system (0 and 1)
   * Fundamental to computer systems
2. **Decimal System**:
   * Base-10 system (0-9)
   * Used in everyday arithmetic
3. **Octal System**:
   * Base-8 system (0-7)
   * Used in some computing applications
4. **Hexadecimal System**:
   * Base-16 system (0-9 and A-F)
   * Commonly used in programming and debugging

**Conversions:**

1. **Binary to Decimal**:
   * Sum the binary digits times 2 raised to the power of their position from right to left.
   * Example: 11012=1×23+1×22+0×21+1×20=1310​
2. **Decimal to Binary**:
   * Divide the number by 2 and record the remainder. Repeat with the quotient until it is zero.
   * Example: 1310→13÷2=6 R 1, 6÷2=3 R 0, 3÷2=1 R 1, 1÷2=0 R 1→11012
3. **Binary to Hexadecimal**:
   * Group binary digits in sets of four, starting from the right. Convert each group to its hexadecimal equivalent.
   * Example: 110110102→1101 1010→DA16
4. **Hexadecimal to Binary**:
   * Convert each hexadecimal digit to its 4-bit binary equivalent.
   * Example: DA16→D A→1101 1010→110110102

**3. Fundamentals of Digital Electronics**

**Basic Concepts:**

* **Logic Gates**: AND, OR, NOT, NAND, NOR, XOR, XNOR
* **Boolean Algebra**: Used to simplify logic expressions
* **Flip-Flops**: Basic memory elements in digital electronics
* **Multiplexers and Demultiplexers**: Used for selecting data inputs and outputs
* **Encoders and Decoders**: Convert data between different forms

**Applications:**

* Used in designing arithmetic circuits, memory units, and control units
* Essential for building complex computing systems

**Assignment 1**: Covering topics from Computer History/Generations, Computer Arithmetic and Number Systems, and Fundamentals of Digital Electronics.

# Week 2: Basic Structure of Computers

**1. Functional Units**

**Central Processing Unit (CPU)**:

* **Arithmetic Logic Unit (ALU)**: Performs arithmetic (addition, subtraction, multiplication, division) and logical operations (AND, OR, NOT, XOR).
* **Control Unit (CU)**: Directs operations of the processor by fetching, decoding, and executing instructions. Manages the flow of data within the CPU and coordinates with other parts of the computer.
* **Registers**: Small, fast storage locations within the CPU for temporary data storage and quick access during instruction execution.
  + **Program Counter (PC)**: Holds the address of the next instruction to be fetched.
  + **Instruction Register (IR)**: Holds the current instruction being executed.
  + **Accumulator (ACC)**: Holds intermediate arithmetic and logic results.

**Memory Unit**:

* **Primary Memory (RAM)**: Volatile memory used for temporary storage during processing. Faster than secondary memory, used for currently running applications.
* **Secondary Memory**: Non-volatile storage for long-term data (e.g., hard drives, SSDs). Retains data even when the power is off and provides larger storage capacity compared to primary memory.

**Input/Output (I/O) Units**:

* **Input Devices**: Keyboard, mouse, scanner.
* **Output Devices**: Monitor, printer, speakers.

**System Bus**:

* **Data Bus**: Carries data between the CPU, memory, and I/O devices. Bi-directional.
* **Address Bus**: Carries address information to access memory locations. Uni-directional.
* **Control Bus**: Carries control signals for coordination and control of activities within the computer. Bi-directional.

**2. Basic Operational Concepts**

**Instruction Cycle**:

1. **Fetch**: The Control Unit fetches the next instruction from memory using the Program Counter (PC), which holds the address of the instruction. The instruction is then stored in the Instruction Register (IR).
2. **Decode**: The Control Unit decodes the instruction in the IR to determine what actions are required. It identifies the operation code (opcode) and the operands. The CU translates the instruction into a set of signals to be sent to other parts of the CPU.
3. **Execute**: The decoded instruction is executed by the appropriate functional units. The ALU performs any required arithmetic or logical operations. Data may be moved between registers, or I/O operations may be performed. The result of the operation is stored back in the memory or registers, and the Program Counter is updated to the next instruction.

**3. Performance and Metrics**

**Measuring Computer Performance**:

1. **Clock Speed**: Measured in Hertz (Hz), it indicates the number of cycles the CPU can perform per second. Higher clock speeds generally mean faster processing.
2. **Throughput**: The number of instructions a computer can process in a given time. Higher throughput indicates better performance.
3. **Latency**: The time taken to complete a single instruction from start to finish. Lower latency means faster response times.
4. **MIPS (Million Instructions Per Second)**: Indicates the execution speed of a CPU. Measures the number of instructions the CPU can execute in one second, in millions.
5. **FLOPS (Floating Point Operations Per Second)**: Measures performance in floating-point calculations, crucial for scientific computations.
6. **CPI (Cycles Per Instruction)**: The average number of clock cycles each instruction takes to execute. Lower CPI generally means better performance.
7. **Amdahl's Law**: Calculates the potential speedup of a system by improving a particular part. Highlights the diminishing returns of parallelizing parts of a program.
   * **Formula**: Speedup=1(1−P)+PSSpeedup=(1−P)+SP​1​
     + **P**: Proportion of the program that can be parallelized.
     + **S**: Speedup of the parallelized part.

# Week 3: Computer Architectures

**1. Instructions and Instruction Sequencing**

**Instruction Types:**

* **Arithmetic Instructions:** Perform basic arithmetic operations like addition, subtraction, multiplication, and division.
* **Logic Instructions:** Perform logical operations like AND, OR, NOT, XOR.
* **Data Transfer Instructions:** Move data between registers, between memory and registers, or between I/O devices and memory.
* **Control Flow Instructions:** Direct the sequence of execution of instructions (e.g., jumps, loops, conditional statements).

**Instruction Format:**

* **Opcode:** Specifies the operation to be performed.
* **Operands:** Specify the data to be operated on or the addresses of data.
* **Addressing Mode:** Defines how the operand’s address is determined (immediate, direct, indirect, indexed).

**Instruction Sequencing:**

* **Sequential Execution:** Instructions are executed one after another in the order they appear.
* **Branching:** Alters the normal sequence of instructions (e.g., if-else statements, loops).
* **Subroutine Calls:** Allow the execution of a sequence of instructions from another location in memory.

**2. The von Neumann Architecture**

**Basic Characteristics:**

* **Stored Program Concept:** Instructions and data are stored in the same memory.
* **Sequential Execution:** Instructions are fetched, decoded, and executed in sequence.
* **Single Memory Space:** One memory space is used for both instructions and data.

**Components:**

* **Central Processing Unit (CPU):** Executes instructions.
  + **Arithmetic Logic Unit (ALU):** Performs arithmetic and logical operations.
  + **Control Unit (CU):** Directs operations within the CPU.
* **Memory:** Stores data and instructions.
* **Input/Output (I/O) Devices:** Allow interaction with external environments.

**Fetch-Decode-Execute Cycle:**

1. **Fetch:** The control unit fetches the next instruction from memory.
2. **Decode:** The instruction is decoded to determine the operation and the operands.
3. **Execute:** The operation is executed, and the results are stored.

**3. IAS Architecture**

**Characteristics:**

* **Institute for Advanced Study (IAS) Computer:**
  + Designed by John von Neumann and others in the 1940s.
  + Used binary numbering and stored-program concept.
* **Components:**
  + **Main Memory:** Stores instructions and data.
  + **Control Unit:** Manages the execution of instructions.
  + **Arithmetic Logic Unit:** Executes arithmetic and logical operations.
  + **Input/Output Devices:** Facilitates communication with external devices.

**Instruction Set:**

* **Format:** Consists of an operation code and operand addresses.
* **Operation Codes:** Include arithmetic operations, memory access operations, and control operations.

**4. The Harvard Architecture**

**Characteristics:**

* **Separate Memory Spaces:** Separate memory for instructions and data.
* **Simultaneous Access:** Instructions and data can be fetched simultaneously, improving performance.

**Components:**

* **Instruction Memory:** Stores instructions.
* **Data Memory:** Stores data.
* **CPU:** Executes instructions from instruction memory and processes data from data memory.

**Advantages:**

* **Speed:** Simultaneous access to instructions and data increases processing speed.
* **Simplicity:** Reduced risk of accidental data corruption in instruction memory.

**Disadvantages:**

* **Complexity:** Requires two separate memory systems.
* **Cost:** Increased hardware cost due to separate memory systems.

# Week 4: Basic Processing Unit

**1. Fundamental Concepts**

**Processor Components:**

* **ALU (Arithmetic Logic Unit):** Performs arithmetic and logical operations.
* **CU (Control Unit):** Directs the operations of the processor.
* **Registers:** Small, fast storage locations used for temporary storage during instruction execution.

**Execution Cycle:**

* **Instruction Fetch:** Retrieve an instruction from memory.
* **Instruction Decode:** Decode the retrieved instruction to determine the operation and operands.
* **Operand Fetch:** Retrieve the operands required for the instruction.
* **Execute:** Perform the operation specified by the instruction.
* **Result Store:** Store the result of the operation in the appropriate location.

**2. Execution of a Complete Instruction**

**Steps Involved:**

1. **Fetch:** Retrieve the instruction from memory.
2. **Decode:** Decode the instruction to understand what action is required.
3. **Fetch Operands:** Retrieve the operands from registers or memory.
4. **Execute:** Perform the operation using the ALU.
5. **Store Result:** Store the result back in memory or a register.
6. **Update Program Counter:** Move to the next instruction.

**Example:**

* **Instruction:** ADD R1, R2, R3 (Add contents of R2 and R3, store in R1)
  + **Fetch:** The instruction is fetched from memory.
  + **Decode:** The instruction is decoded as an ADD operation.
  + **Fetch Operands:** The contents of R2 and R3 are fetched.
  + **Execute:** The ALU adds the contents of R2 and R3.
  + **Store Result:** The result is stored in R1.
  + **Update Program Counter:** Move to the next instruction.

**3. Multiple Bus Organization**

**Concept:**

* **Multiple Buses:** Use of multiple buses to connect various components, allowing multiple data transfers simultaneously.

**Types of Buses:**

* **Data Bus:** Transfers data between components.
* **Address Bus:** Transfers addresses of data and instructions.
* **Control Bus:** Transfers control signals.

**Advantages:**

* **Increased Throughput:** Multiple data transfers can occur simultaneously.
* **Reduced Bottlenecks:** Reduces contention for a single bus.

**Disadvantages:**

* **Complexity:** Increased complexity in bus management.
* **Cost:** Higher cost due to additional hardware.

**4. Hardwired Control**

**Concept:**

* **Control Logic:** Implemented using fixed logic circuits (e.g., gates, flip-flops).
* **Control Signals:** Generated by combinational logic circuits based on the current instruction and status of the CPU.

**Advantages:**

* **Speed:** Faster than microprogrammed control due to fixed, low-latency paths.
* **Efficiency:** Efficient for simple, repetitive tasks.

**Disadvantages:**

* **Flexibility:** Less flexible and harder to modify than microprogrammed control.
* **Complexity:** Can become very complex for larger instruction sets.

**Assignment 2:**

* Covering Instruction Sequencing, von Neumann and Harvard Architectures, and Basic Processing Unit concepts.

# Week 5: Instruction Set Architecture

**1. Instruction Set Architecture (ISA)**

**Definition and Importance:**

* **ISA Overview:**
  + The ISA defines how the processor communicates with the programmer.
  + It is an abstract model, not stored in memory, which dictates the set of instructions the processor can execute.
  + The ISA specifies the processor’s register set, memory addressing modes, and instruction formats and encodings.
* **Role in Computer Architecture:**
  + Acts as an interface between hardware and software.
  + Significantly impacts the performance and efficiency of the processor.
  + Defines how the CPU is controlled by software.

**2. Addressing Modes**

**Purpose:**

* Addressing modes determine how the CPU identifies the operand(s) of an instruction.
* Each instruction must include the operation to be performed (opcode) and the operand address information.

**Common Addressing Modes:**

* **Register Indirect Addressing Mode:** Uses a register to hold the address of the operand.
* **Immediate Addressing Mode:** The operand is directly specified within the instruction.
* **Indexed Addressing Modes:** Uses a base address and an index register to calculate the operand’s address.
* **Scaled Indexed Addressing Mode:** Similar to indexed addressing but includes a scaling factor.
* **Direct/Absolute Addressing Mode:** Specifies the exact memory address of the operand.
* **Indirect Addressing Mode:** The instruction specifies a memory location that contains the address of the operand.

**3. Main Types of ISA**

**Complex Instruction Set Computer (CISC):**

* **Characteristics:**
  + Large number of instructions.
  + Instructions can perform multiple operations.
  + Easier for programmers to write code due to high-level commands.
* **Pros:**
  + Facilitates complex instructions through microprogramming.
  + Efficient use of main memory.
  + Easier for compilers to generate efficient code.
* **Cons:**
  + Complex hardware design.
  + Slower clock speeds due to complexity.

**Reduced Instruction Set Computer (RISC):**

* **Characteristics:**
  + Smaller, simpler set of instructions.
  + Each instruction performs a single operation.
* **Pros:**
  + Faster performance due to simple instructions.
  + Efficient use of chip space, allowing for additional features.
  + Lower per-chip cost and faster design time.
* **Cons:**
  + Performance dependent on the code executed.
  + Requires fast memory systems to keep up with instruction processing.

**4. Differences Between RISC and CISC**

* **Instruction Complexity:**
  + RISC: Simple instructions, each performing a single operation.
  + CISC: Complex instructions, performing multiple operations.
* **Execution:**
  + RISC: Executes one instruction per clock cycle.
  + CISC: Instructions take multiple clock cycles.
* **Memory Usage:**
  + RISC: Heavy use of RAM, fixed instruction sizes.
  + CISC: Efficient RAM use, variable instruction sizes.

**5. Instruction Types**

**Categories:**

* **Data Transfer Instructions:** Move data between locations.
  + **Examples:**
    - **MOV:** Moves data from one location to another (e.g., MOV AX, BX).
    - **PUSH:** Pushes data onto the stack (e.g., PUSH CX).
    - **POP:** Pops data from the stack (e.g., POP DX).
* **Data Manipulation Instructions:** Perform arithmetic and logical operations.
  + **Examples:**
    - **ADD:** Adds two operands (e.g., ADD AX, BX).
    - **SUB:** Subtracts one operand from another (e.g., SUB AX, BX).
    - **MUL:** Multiplies operands (e.g., MUL BX).
* **Program Sequencing and Control Instructions:** Direct program execution flow.
  + **Examples:**
    - **JMP:** Jumps to a specified location (e.g., JMP label).
    - **CMP:** Compares two operands (e.g., CMP AX, BX).
    - **JZ:** Jumps if Zero flag is set (e.g., JZ label).
* **Input and Output Instructions:** Handle I/O operations.
  + **Examples:**
    - **IN:** Reads data from an input port (e.g., IN AL, 0x60).
    - **OUT:** Sends data to an output port (e.g., OUT 0x80, AL).
    - **INT:** Triggers a software interrupt (e.g., INT 0x21).

# Week 6: Memory System

**1. Basic Concepts**

**Memory System Overview:**

* Memory is a critical component of the computer system, used to store data and instructions.
* Memory is organized in a hierarchy to balance speed, cost, and size.

**Types of Memory:**

* **Primary Memory:** Directly accessible by the CPU (e.g., RAM, cache).
* **Secondary Memory:** Not directly accessible by the CPU (e.g., hard drives, SSDs).

**2. Memory Hierarchy**

**Memory Hierarchy Structure:**

* Organizes memory in levels based on speed, cost, and size.
* **Registers:** Smallest, fastest, and most expensive.
* **Cache Memory:** Larger and slower than registers but faster than main memory.
* **Main Memory (RAM):** Larger and slower than cache.
* **Secondary Storage:** Largest, slowest, and least expensive (e.g., HDD, SSD).

**Purpose of Memory Hierarchy:**

* To provide a balance between cost and speed.
* Faster, more expensive memory is used closer to the CPU.
* Slower, less expensive memory is used further from the CPU.

**3. Speed, Size, and Cost**

**Speed:**

* Measured in access time (how quickly data can be retrieved).
* Faster memory has shorter access times.

**Size:**

* Measured in bytes (e.g., kilobytes, megabytes, gigabytes).
* Larger memory can store more data but typically has slower access times.

**Cost:**

* Faster and larger memory is more expensive.
* The goal is to optimize the balance between speed, size, and cost.

**4. Cache Memories**

**Cache Memory:**

* Small, fast memory located close to the CPU.
* Stores frequently accessed data and instructions to speed up processing.

**Cache Levels:**

* **L1 Cache:** Smallest and fastest, integrated into the CPU.
* **L2 Cache:** Larger and slower than L1, often located on the CPU chip.
* **L3 Cache:** Larger and slower than L2, shared among multiple CPU cores.

**Cache Operation:**

* **Cache Hit:** When the CPU finds the data it needs in the cache.
* **Cache Miss:** When the CPU does not find the data in the cache, leading to access from slower memory.

**5. Secondary Storage Devices**

**Secondary Storage:**

* Non-volatile memory used for long-term data storage.
* Includes hard disk drives (HDDs) and solid-state drives (SSDs).

**HDD vs. SSD:**

* **HDD:** Uses spinning disks to read/write data, slower access times, higher capacity.
* **SSD:** Uses flash memory, faster access times, lower capacity, more durable.

**6. Memory Hierarchy in Detail**

**Hierarchy Levels:**

1. **Registers:** Fastest, smallest, most expensive.
2. **Cache Memory:** Fast, small, expensive.
3. **Main Memory (RAM):** Moderate speed, size, cost.
4. **Secondary Storage:** Slowest, largest, least expensive.

**Trade-offs:**

* Speed vs. size: Faster memory is smaller.
* Cost vs. performance: Higher performance memory is more expensive.

# Week 7: Virtual Memory and Memory Management

**1. Virtual Memory**

**Virtual Memory Concept:**

* Allows the computer to use more memory than physically available by using disk space.
* Provides the illusion of a large, contiguous memory space.

**Benefits:**

* Enables larger programs to run on systems with limited physical memory.
* Provides isolation between processes, enhancing security and stability.

**Mechanism:**

* **Paging:** Divides memory into fixed-size pages, mapping virtual addresses to physical addresses.
* **Segmentation:** Divides memory into variable-sized segments based on logical divisions.

**2. Memory Management Requirements**

**Memory Management Functions:**

* Allocation and deallocation of memory.
* Keeping track of each memory location's status (free or allocated).
* Managing multiple processes' memory requirements.

**Techniques:**

* **Paging:** Fixed-size pages improve efficiency and manageability.
* **Segmentation:** Logical division of memory improves program organization.

**Swapping:**

* Moving data between physical memory and disk storage to manage memory requirements.
* **Page Fault:** Occurs when a requested page is not in memory, triggering a swap from disk.

**Protection:**

* Ensures processes do not interfere with each other's memory.
* Implemented through hardware mechanisms like memory protection units (MPUs).

# Week 8: I/O Organization

**1. Accessing I/O Devices**

**I/O Devices Overview:**

* Devices that allow interaction between the computer and the external world (e.g., keyboard, mouse, printer).

**Access Methods:**

* **Memory-mapped I/O:** Uses the same address space for memory and I/O devices.
* **Port-mapped I/O:** Uses separate address space for I/O devices.

**2. Programmed Input/Output (PIO)**

**Programmed I/O:**

* CPU directly controls data transfer to/from I/O devices.
* Polling is used to check the status of an I/O device.

**Advantages:**

* Simple and straightforward to implement.

**Disadvantages:**

* Inefficient as CPU spends time polling devices, leading to idle time.

**3. Interrupts**

**Interrupt Mechanism:**

* Allows devices to signal the CPU for attention.
* CPU can be interrupted to handle I/O operations, improving efficiency.

**Types of Interrupts:**

* **Hardware Interrupts:** Triggered by hardware devices.
* **Software Interrupts:** Triggered by software instructions.

**Interrupt Handling:**

* **Interrupt Service Routine (ISR):** Special function executed in response to an interrupt.
* **Interrupt Vector Table:** Stores addresses of ISRs for different interrupts.

**4. Direct Memory Access (DMA)**

**DMA Concept:**

* Allows direct data transfer between I/O devices and memory without CPU intervention.
* Improves system efficiency by offloading data transfer tasks from the CPU.

**DMA Controller:**

* Manages DMA operations, controlling data transfer between devices and memory.

**Advantages:**

* Reduces CPU workload, allowing it to perform other tasks.
* Increases data transfer speed.

**Disadvantages:**

* Adds complexity to the system.

**5. Buses**

**Bus System:**

* Communication pathway connecting various components of the computer (CPU, memory, I/O devices).

**Bus Types:**

* **Data Bus:** Transfers data between components.
* **Address Bus:** Carries memory addresses.
* **Control Bus:** Carries control signals.

**Bus Architecture:**

* **Single Bus:** All components share a single bus, leading to potential bottlenecks.
* **Multiple Buses:** Separate buses for different functions, improving performance and reducing bottlenecks.

# Week 9: Standard I/O Interfaces

**1. Standard I/O Interfaces**

**I/O Interface Overview:**

* Interfaces that allow the connection and communication between the CPU and peripheral devices.
* Standardized to ensure compatibility and interoperability among different hardware and systems.

**2. PCI (Peripheral Component Interconnect)**

**PCI Overview:**

* A local computer bus for attaching hardware devices in a computer.
* Developed to replace older ISA and VL-bus standards.

**Key Features:**

* Supports bus mastering, allowing devices to communicate with each other without CPU intervention.
* Provides plug-and-play functionality, automatically configuring devices.
* Delivers high data transfer rates.

**Types:**

* **PCI:** The original standard.
* **PCI-X:** An enhanced version providing higher speeds.
* **PCI Express (PCIe):** Uses a high-speed serial interface, offering improved performance and scalability.

**3. SCSI (Small Computer System Interface)**

**SCSI Overview:**

* A set of standards for connecting and transferring data between computers and peripheral devices.
* Used primarily for hard drives and other storage devices.

**Key Features:**

* Allows multiple devices (up to 16) to be connected to a single SCSI bus.
* Provides high-speed data transfer rates.
* Supports a wide range of devices, including scanners, CD drives, and printers.

**SCSI Standards:**

* **SCSI-1:** The original standard, offering up to 5 MB/s transfer rate.
* **SCSI-2:** Improved standard with faster speeds and additional features.
* **SCSI-3:** Further enhancements, including Ultra SCSI with up to 320 MB/s transfer rates.

**4. USB (Universal Serial Bus)**

**USB Overview:**

* A standard for connecting peripherals to a computer.
* Designed to replace various serial and parallel ports.

**Key Features:**

* Supports plug-and-play and hot-swapping.
* Provides power to connected devices.
* Offers multiple versions with varying speeds (e.g., USB 1.1, USB 2.0, USB 3.0, USB 3.1, USB 3.2, USB4).

**USB Topology:**

* **Hub-and-Spoke Model:** Multiple devices can be connected using hubs, allowing a star topology.

**5. I/O Devices and Processors**

**I/O Devices:**

* **Input Devices:** Devices used to input data into a computer (e.g., keyboard, mouse, scanner).
* **Output Devices:** Devices used to output data from a computer (e.g., monitor, printer, speakers).
* **Storage Devices:** Devices used to store data (e.g., hard drives, SSDs, optical drives).

**I/O Processors:**

* Dedicated processors designed to handle input/output operations.
* Offload I/O processing tasks from the main CPU, improving system performance.

# Week 10: Pipelining and Parallel Computing

**1. General Concepts**

**Pipelining Overview:**

* A technique used to improve the execution throughput of instructions in a CPU.
* Divides the execution process into several stages, allowing multiple instructions to be processed simultaneously.

**Stages of a Pipeline:**

1. **Fetch:** Retrieve the instruction from memory.
2. **Decode:** Interpret the instruction.
3. **Execute:** Perform the operation specified by the instruction.
4. **Memory Access:** Read or write data from/to memory.
5. **Write-Back:** Store the result in the register.

**Benefits of Pipelining:**

* Increases instruction throughput.
* Efficient use of CPU resources.
* Reduces the average time per instruction.

**Challenges:**

* **Hazards:** Issues that disrupt the smooth execution of the pipeline.
  + **Data Hazards:** Occur when instructions depend on the results of previous instructions.
  + **Control Hazards:** Occur due to branch instructions altering the flow of execution.
  + **Structural Hazards:** Occur when hardware resources are insufficient to support all pipeline stages.

**2. Instruction Pipeline**

**Instruction-Level Parallelism (ILP):**

* The ability to execute multiple instructions simultaneously.
* Pipelining is a form of ILP, where different stages of multiple instructions are processed concurrently.

**Pipeline Optimization:**

* Techniques to minimize hazards and improve pipeline efficiency.
* **Forwarding:** Resolves data hazards by directly passing data between pipeline stages.
* **Branch Prediction:** Predicts the outcome of branch instructions to reduce control hazards.
* **Superscalar Execution:** Uses multiple pipelines to execute more than one instruction per clock cycle.

**3. Instruction-Level Parallelism**

**Superscalar Architecture:**

* CPUs with multiple execution units to handle multiple instructions in parallel.
* Enhances performance by issuing several instructions per cycle.

**Dynamic Scheduling:**

* Out-of-order execution to avoid pipeline stalls and improve efficiency.
* Reorder buffer to maintain the correct program order.

**Speculative Execution:**

* Predicts the direction of branches and executes instructions ahead of time.
* Corrects any mispredictions by discarding speculative results.

**4. Data Path and Control Considerations**

**Data Path Design:**

* The hardware components involved in the execution of instructions (e.g., ALU, registers, buses).
* Efficient design is crucial for maximizing performance in pipelined and parallel systems.

**Control Unit:**

* Manages the execution of instructions by generating control signals.
* Can be implemented using either hardwired control or microprogrammed control.

**Control Signal Generation:**

* Coordinates the actions of different parts of the CPU to ensure correct instruction execution.

**5. Performance Considerations**

**Throughput and Latency:**

* **Throughput:** The number of instructions executed per unit time.
* **Latency:** The time taken to execute a single instruction.

**Factors Affecting Performance:**

* **Clock Speed:** Higher clock speeds generally improve performance but can increase power consumption and heat.
* **Pipeline Depth:** Deeper pipelines can increase throughput but may suffer from higher latency and complexity.
* **Branch Prediction Accuracy:** Better branch prediction reduces pipeline stalls and improves performance.

**Parallel Computing:**

* Using multiple processors or cores to execute instructions simultaneously.
* Enhances performance by dividing tasks among processors, allowing concurrent execution.

**Types of Parallelism:**

* **Data Parallelism:** Distributes data across multiple processors, each performing the same operation on different data.
* **Task Parallelism:** Distributes different tasks or instructions across multiple processors.

# Week 11: Data Path and Control Considerations

**1. Data Path and Control Considerations**

**Data Path Design:**

* The data path is the collection of hardware elements through which data passes in a CPU.
* Includes components like ALUs, registers, and buses that are used for executing instructions.

**Components of the Data Path:**

* **Arithmetic Logic Unit (ALU):** Performs arithmetic and logical operations.
* **Registers:** Small, fast storage locations used to hold data and instructions.
* **Buses:** Pathways that transfer data between components inside the CPU.
* **Multiplexers (MUX):** Selects one of several input signals and forwards the selected input into a single line.
* **Control Unit:** Directs the operation of the processor by generating control signals.

**Types of Data Paths:**

* **Single-Bus Data Path:** Uses one bus for all data transfers, simplifying design but limiting performance.
* **Multiple-Bus Data Path:** Uses multiple buses for concurrent data transfers, increasing complexity but enhancing performance.

**2. Control Unit Design**

**Control Unit:**

* Manages the execution of instructions by generating control signals.
* Two main types: Hardwired Control and Microprogrammed Control.

**Hardwired Control:**

* Uses fixed logic circuits to control signals.
* Faster but less flexible and more complex to design for large instruction sets.

**Microprogrammed Control:**

* Uses a sequence of microinstructions stored in memory to generate control signals.
* More flexible and easier to modify or expand, but slightly slower than hardwired control.

**Control Signal Generation:**

* Ensures the correct sequence of operations in the data path.
* Determines the timing and control of data transfer and processing.

**3. Execution of a Complete Instruction**

**Instruction Execution Cycle:**

1. **Fetch:** Retrieve the instruction from memory.
2. **Decode:** Determine the opcode and operand(s).
3. **Execute:** Perform the operation specified by the instruction.
4. **Memory Access (if needed):** Read or write data from/to memory.
5. **Write-Back:** Store the result in the appropriate register.

**Pipeline Execution:**

* Each stage of instruction execution is handled by different parts of the CPU simultaneously.

**4. Multiple Bus Organization**

**Multiple Bus Organization:**

* Utilizes more than one bus to allow multiple data transfers to occur at the same time.
* Enhances CPU performance by reducing bottlenecks in data transfer.

**Advantages:**

* Improved performance due to parallelism in data transfers.
* Reduced data transfer time compared to single-bus organization.

**Disadvantages:**

* Increased complexity in control and data path design.
* Higher cost due to additional hardware.

**5. Hardwired Control vs. Microprogrammed Control**

**Hardwired Control:**

* Faster operation due to direct implementation.
* Less flexible and more difficult to modify.

**Microprogrammed Control:**

* Easier to design and modify, especially for complex instruction sets.
* Slightly slower due to the need to fetch microinstructions.

**Control Unit Optimization:**

* Techniques such as pipelining in control unit design can enhance performance.

# Week 12: Assembly Language Programming

**1. Instruction Mnemonics and Syntax**

**Assembly Language Basics:**

* Low-level programming language that uses symbolic code and is closely related to machine language.
* Each instruction corresponds directly to a machine code instruction.

**Mnemonics:**

* Short codes representing machine instructions (e.g., MOV, ADD, SUB).
* Easier to remember and use compared to binary machine code.

**Syntax:**

* Typically includes an opcode followed by operand(s).
* Example: MOV AX, BX (Move contents of BX register into AX register).

**Assembly Language Structure:**

* **Label:** An identifier for a memory location or instruction.
* **Opcode:** The operation code specifying the instruction.
* **Operand:** The data to be operated on or the address/location of the data.

**2. Assembler Directives and Commands**

**Assembler Directives:**

* Instructions for the assembler itself, not converted into machine code.
* Examples include defining data segments, reserving storage, and setting the start address of the program.

**Common Directives:**

* **DB (Define Byte):** Reserves byte-sized storage.
* **DW (Define Word):** Reserves word-sized storage.
* **ORG (Origin):** Sets the starting address for the code or data.
* **END:** Marks the end of the source code file.

**Commands:**

* Commands are specific to the assembler being used.
* Examples include commands for assembling, linking, and debugging programs.

**3. Assembler and Execution of Programs**

**Assembly Process:**

1. **Source Code:** Written in assembly language.
2. **Assembler:** Translates assembly code into machine code.
3. **Object Code:** The machine code output by the assembler.
4. **Linker:** Combines object code with other modules or libraries.
5. **Executable:** The final machine code program ready to be executed by the CPU.

**Execution Process:**

* The CPU fetches, decodes, and executes the machine code instructions.
* Involves various stages such as loading the program into memory and initializing registers.

**4. Microprogrammed Control**

**Microprogrammed Control Unit:**

* Uses microinstructions stored in control memory to generate control signals.
* Each microinstruction specifies one or more micro-operations (basic operations performed by the CPU).

**Microinstruction Format:**

* Typically includes fields for the control signals, next microinstruction address, and any constants or immediate values.

**Microprogram Sequencer:**

* Determines the sequence of microinstructions to execute based on the current instruction.

**Advantages:**

* Simplifies the design and implementation of complex instruction sets.
* Easier to modify or update the control logic.

**Disadvantages:**

* Slightly slower execution compared to hardwired control due to the additional memory access.

**Use in Modern CPUs:**

* Many modern CPUs use a combination of hardwired and microprogrammed control to balance performance and flexibility.