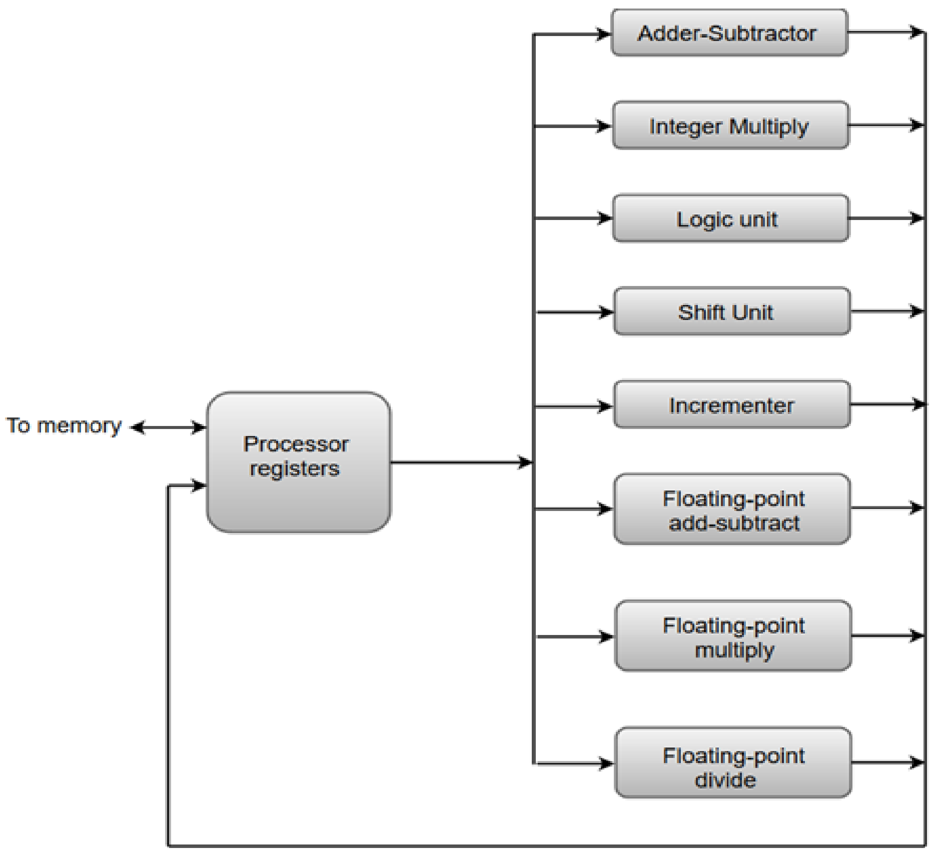
UNIT-4 PARALLELISM

INTRODUCTION TO PARALLEL PROCESSING:

- Parallel processing is a techniques which enables the system to achieve simultaneous data-processing tasks to increase the computational speed of a computer system.
- A parallel processing system can carry out simultaneous data-processing to achieve faster execution time. For example, while an instruction is being processed in the ALU component of the CPU, the next instruction can be read from memory.
- Its primary purpose is to enhance the computer processing capability and increase its throughput, i.e. the amount of processing that can be done during a given interval of time.
- A parallel processing system can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously. The data can be distributed among various multiple functional units.

The following diagram shows one possible way of separating the execution unit into eight functional units operating in parallel.



- The adder and integer multiplier performs the arithmetic operation with integer numbers.
- The floating-point operations are separated into three circuits operating in parallel.
- The logic, shift, and increment operations can be performed concurrently on different data. All units are independent of each other, so one number can be shifted while another number is being incremented.

Advantages of Parallel Processing:

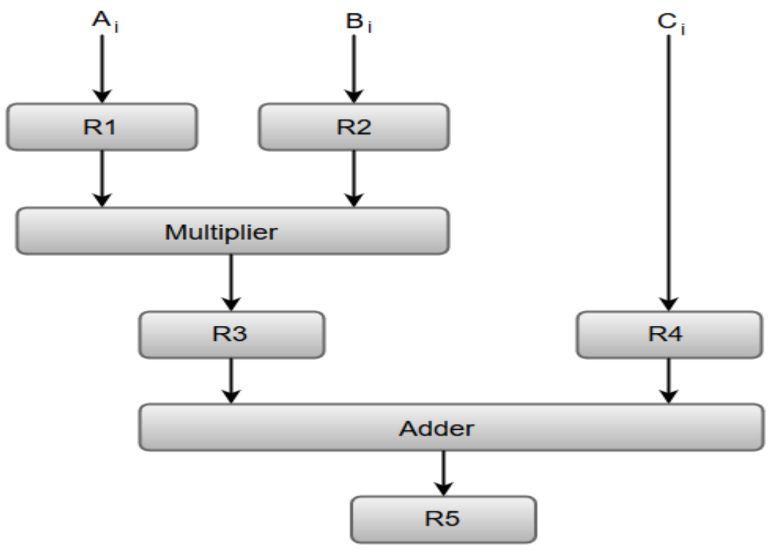
1. It increases the speed and efficiency of computers.
2. Computers can be used to solve more complex and more extensive problems.
3. For a system that has to support billions of operations (for example, bank software), parallel processing makes things easier.

Disadvantages of Parallel Processing:

1. Increases the cost of computers since more hardware is required.
2. Parallel architectures are difficult to achieve.
3. It needs different code modifications depending on the target architecture.

PIPELINING:

- The term Pipelining refers to a technique of decomposing a sequential process into sub-operations, with each sub-operation being executed in a dedicated segment that operates concurrently with all other segments.
- The most important characteristic of a pipeline technique is that several computations can be in progress in distinct segments at the same time.
- The overlapping of computation is made possible by associating a register with each segment in the pipeline. The registers provide isolation between each segment so that each can operate on distinct data simultaneously.
- The structure of pipeline organization can be represented simply by including input register for each segment followed by a combinational circuit. The following diagram represents the combined as well as the sub-operations performed in each segment of the pipeline.



- Registers R1, R2, R3, and R4 hold the data and the combinational circuits operate in a particular segment. The output generated by the combinational circuit in a given segment is applied as an input register of the next segment.

For example, from the block diagram, we can see that the register R3 is used as one of the input registers for the combinational adder circuit.

The combined multiplication and addition operation is done with a stream of numbers such as:

$A_i * B_i + C_i$ for i = 1, 2, 3, ..., 7

The operation to be performed on the numbers is decomposed into sub-operations with each sub-operation to be implemented in a segment within a pipeline. The sub-operations performed in each segment of the pipeline are defined as:

$R1 \leftarrow A_i$,	**$R2 \leftarrow B_i$**	**Input A_i, and B_i**
$R3 \leftarrow R1 * R2$,	**$R4 \leftarrow C_i$**	**Multiply, and input C_i**
$R5 \leftarrow R3 + R4$		**Add C_i to product**

Advantages of Pipelining:

1. **Increased Throughput and performance:** Pipelining allows multiple instructions to be executed simultaneously in different stages of the pipeline. This overlapping of instructions increases the overall throughput and performance of the processor, as more instructions are completed in a given unit of time.
2. **Improved Resource Utilization:** By breaking down the instruction execution into smaller stages, pipelining enables better utilization of the processor's resources. Each stage can be dedicated to a specific operation, allowing different instructions to make use of different functional units simultaneously.
3. **Reduced Latency (Delay):** Pipelining reduces the time taken to execute an instruction by dividing it into smaller stages. As a result, the time between the start and completion of an instruction is reduced. This can lead to faster overall program execution.

Disadvantages of Pipelining:

1. **Pipeline Hazards:** Pipelining introduces several types of hazards that can impact the efficiency of instruction execution. These include structural hazards (resource conflicts), data hazards (dependencies between instructions), and control hazards (branch instructions). Handling these hazards requires additional hardware which can increase the complexity of the processor design.
2. **Pipeline Stalls/Pause:** In certain situations, the pipeline may need to stall or temporarily pause the execution of instructions due to hazards, dependencies, or other events. These stalls reduce the potential performance gain from pipelining and may introduce delays.
3. **Instruction Dependencies:** Dependencies between instructions can limit the level of parallelism achieved in pipeline. If an instruction depends on the result of a previous instruction that has not yet completed, the pipeline may need to introduce no-operation instructions or pause to resolve the dependency, leading to reduced performance.

The Pipeline organization is applicable for two areas of computer design. It includes:

1. Arithmetic Pipeline
2. Instruction Pipeline

Arithmetic Pipeline: Arithmetic Pipelines are mostly used in high-speed computers.

For Example: A and B are two fractions that represent mantissa and a and b are the exponents.

X = A * 10^a =>   0.9504 * 10^3

Y = B * 10^b =>   0.8200 * 10^2

The combined operation of floating-point addition and subtraction is divided into four segments. Each segment contains the corresponding sub-operation to be performed in the given pipeline. The sub-operations that are shown in the four segments are:

1. Compare the exponents by subtraction.
2. Align the mantissas.
3. Add or subtract the mantissas.
4. Normalize the result.

1. **Compare exponents by subtraction:**

   The exponents are compared by subtracting them to determine their difference. The larger exponent is chosen as the exponent of the result.

   The difference of the exponents, i.e., 3 - 2 = 1 determines how many times the mantissa associated with the smaller exponent must be shifted to the right.

2. **Align the mantissas:**

   The mantissa associated with the smaller exponent is shifted according to the difference of exponents determined in segment one.

   X = 0.9504 * 10^3

   Y = 0.08200 * 10^3

3. **Add mantissas:**

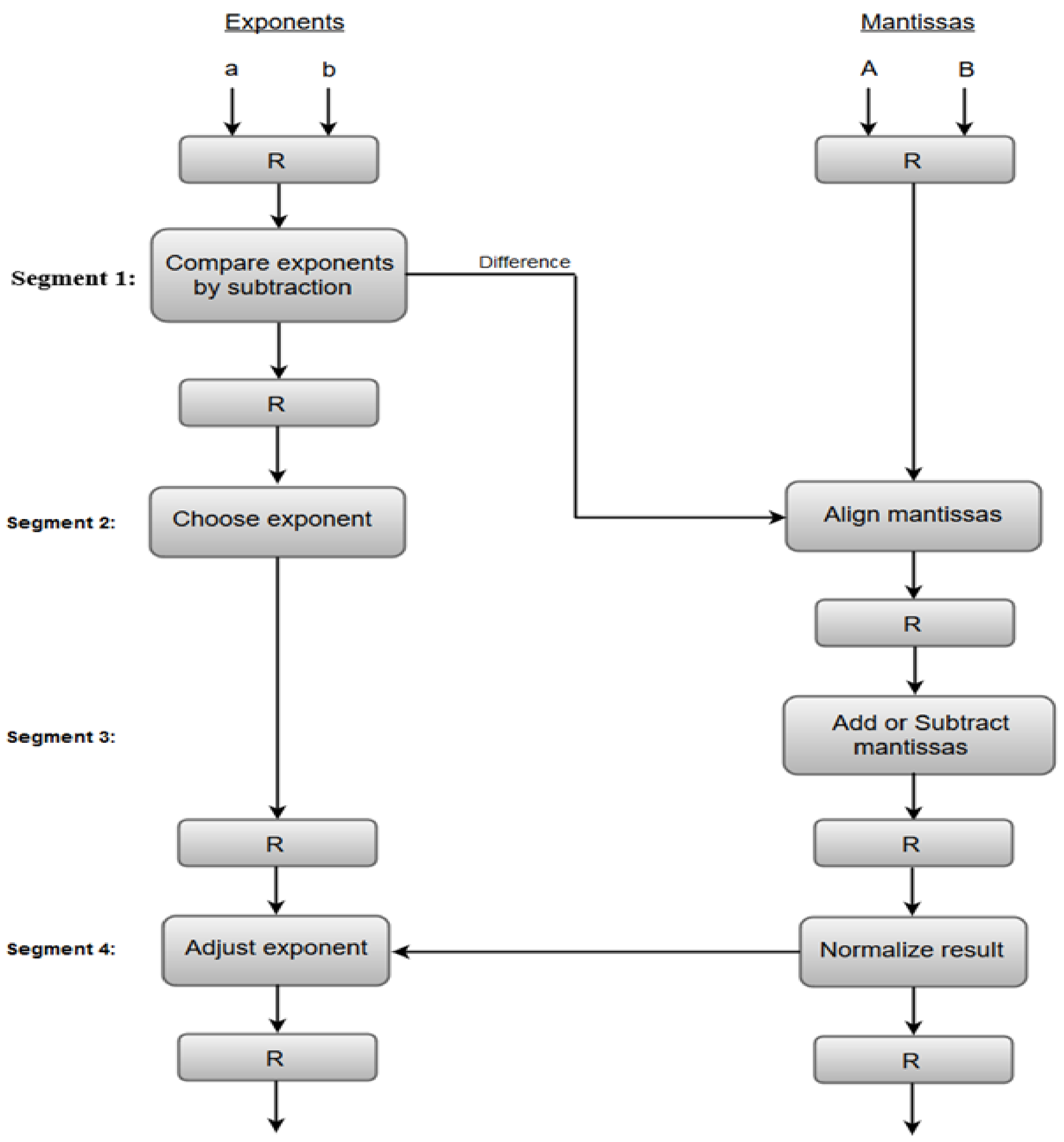
   The two mantissas are added in segment three.   Z = X + Y = 1.0324 * 10^3

4. **Normalize the result:**

   After normalization, the result is written as: Z = 0.1324 * 10^4

This block diagram represents the sub-operations performed in each segment of the pipeline.

Note: **Registers are placed after each sub-operation to store the intermediate results.**



Instruction Pipeline: Instruction Pipeline includes following sequence of steps.

1. Fetch instruction from memory.
2. Decode the instruction.
3. Calculate the effective address.
4. Fetch the operands from memory.
5. Execute the instruction.
6. Store the result in the proper place.

The organization of an instruction pipeline will be more efficient if the instruction cycle is divided into segments of equal duration. One of the most common examples of this type of organization is a **Four-segment instruction pipeline.**

The four segments include:

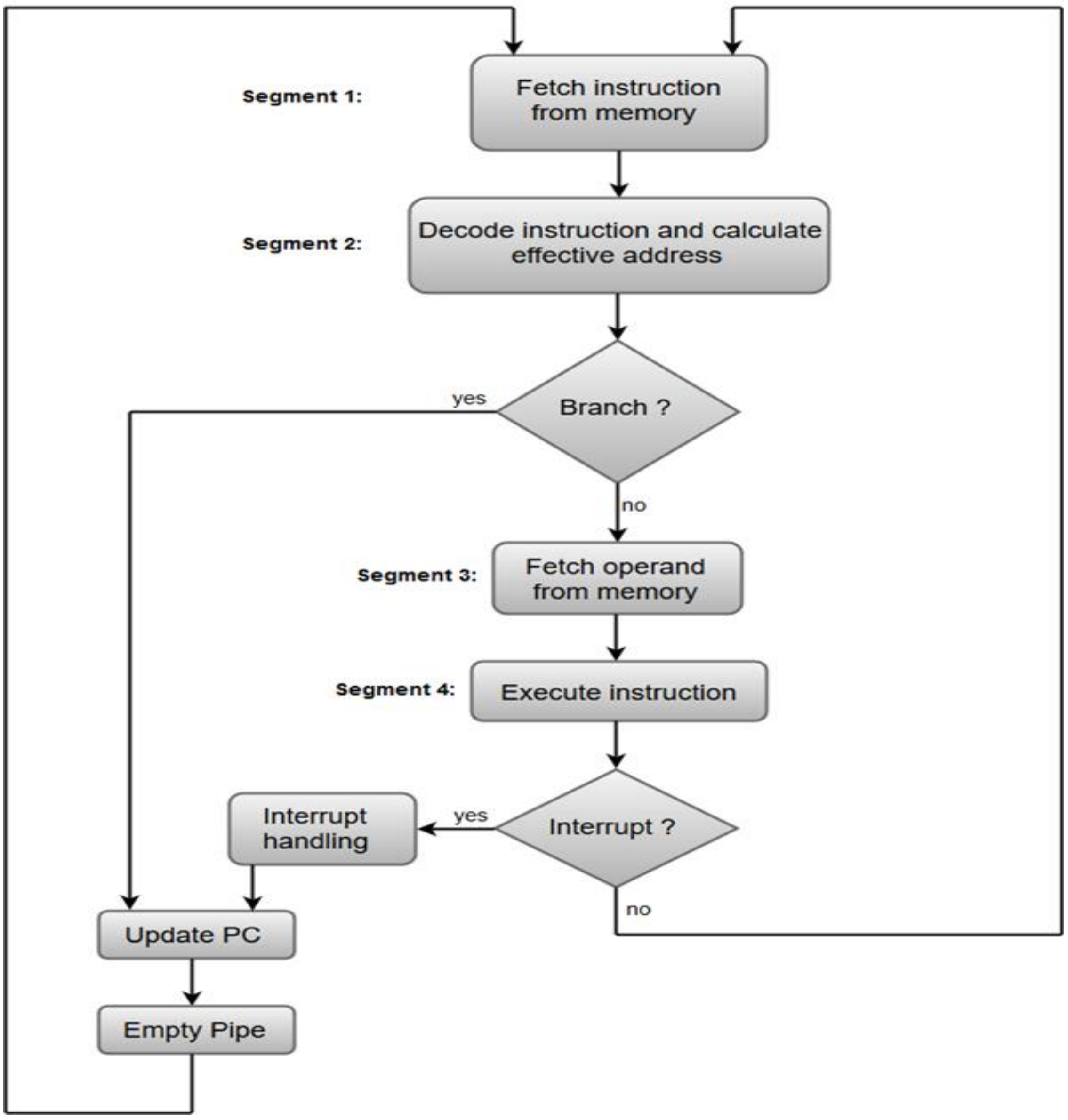
Segment 1: Instruction fetch segment can be implemented using first in first out (FIFO) buffer.

Segment 2: The instruction fetched from memory is decoded in the second segment, and eventually, the effective address is calculated in a separate arithmetic circuit.

Segment 3: An operand from memory is fetched in the third segment.

Segment 4: The instructions are finally executed in the last segment of pipeline organization.

The following block diagram shows a typical example of a four-segment instruction pipeline. The instruction cycle is completed in four segments.



CHARACTERISTICS OF MULTIPROCESSORS:

A **Multiprocessor** is a single computer that has multiple processors. It is possible that the processors in the multiprocessor system can communicate and co-operate at various levels of solving a given problem. The communications between the processors take place by sending messages from one processor to another, or by sharing a common memory.

There are the major characteristics of multiprocessors are as follows −

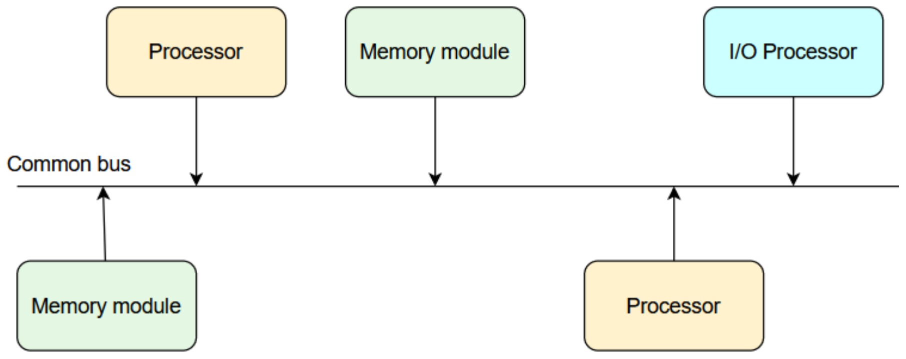
1. **Parallel Execution:** Multiprocessors enable concurrent execution of multiple tasks or processes. Each processor operates independently, executing its own instructions, which allows for parallelism and increased overall system throughput.
2. **Shared Memory:** In a multiprocessor system, processors share a common memory space. This shared memory allows communication and data sharing between processors, simplifying programming and co-ordination between tasks.
3. **Scalability:** Multiprocessor systems can be designed with varying numbers of processors, from a few to hundreds or even thousands of processors. This scalability enables the system to handle a wide range of computational workloads, from small-scale tasks to highly parallelizable applications.
4. **Load Balancing:** Load balancing is crucial in a multiprocessor system to distribute the workload evenly among the processors. Efficient load balancing ensures that each processor is utilized optimally and avoids scenarios where some processors are idle while others are overloaded.
5. **Inter-Processor Communication:** Communication between processors is essential for coordination and synchronization in a multiprocessor system. Inter-processor communication mechanisms, such as shared memory, message passing, or hardware-based communication channels, facilitate data exchange and coordination between processors.
6. **Synchronization and Consistency:** Since multiple processors can access and modify shared data, ensuring synchronization and data consistency is crucial in multiprocessor systems. Techniques such as locks, semaphores, etc are used to coordinate access to shared resources and maintain data integrity.
7. **Fault Tolerance:** Multiprocessor systems can incorporate fault tolerance mechanisms to ensure reliability and availability. Redundancy and error detection/correction techniques can be employed to handle processor failures or temporary errors.

INTERCONNECTION STRUCUTRES:

Interconnection structures in computer architecture refer to the ways in which the various components of a computer system, such as processors, memory modules, and input/output devices, are connected and communicate with each other. The choice of interconnection structure significantly impacts the performance, scalability, and efficiency of the system. **Here are some commonly used interconnection structures:**

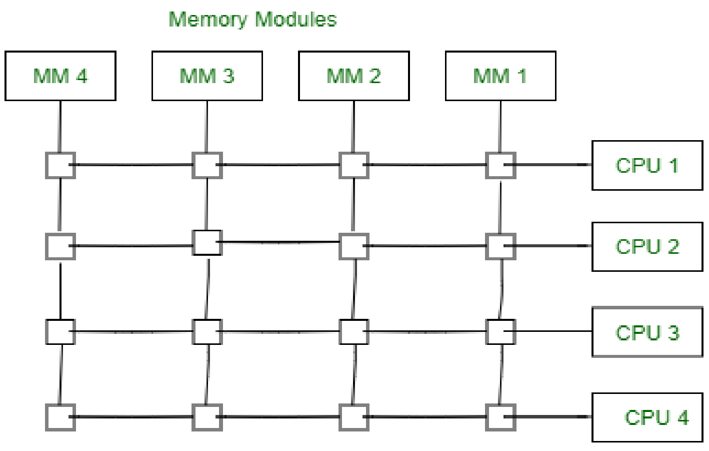
1. Time Shared/ Common Bus:

- A bus is a shared communication pathway that connects multiple components in a computer system. It consists of a set of wires, data, and control signals. Buses are simple and cost-effective but can become a performance blockage when multiple components oppose for access to the bus.



2. Crossbar Switch:

- It contains of a number of cross-points that are kept at intersections among memory module and processor buses paths. In each cross-point, the small square represents a switch which obtains the path from a processor to a memory module.
- Each switch point has control logic to set up the transfer path among a memory and processor. It calculates the address which is placed in the bus to obtain whether its specific module is being addressed.
- In addition, it eliminates multiple requests for access to the same memory module on a predetermined priority basis.

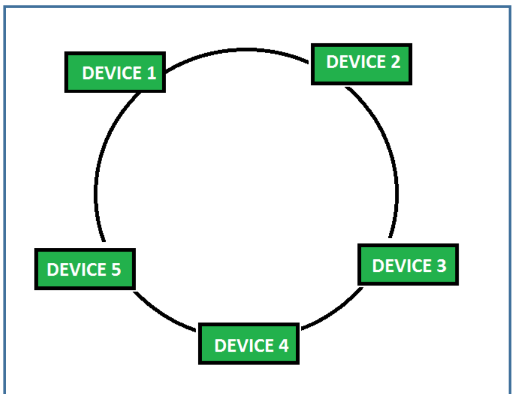


3. **Ring:**

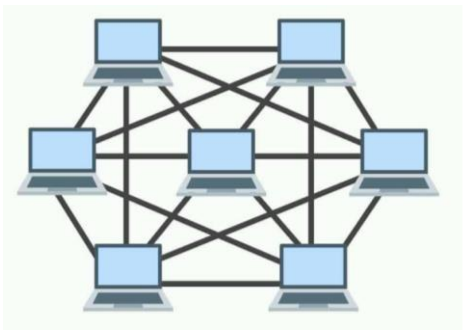
- In a ring interconnection, components are connected in a circular topology, forming a closed loop. Each component receives and forwards data in a sequential manner.
- It provide simplicity and fault tolerance, as messages can be re-routed in case of a failure.
- If any of the connection is broken then the whole network crashes because they are connected sequentially. This is used for connection of LAN or WAN.

4. **Mesh:**

- A mesh interconnection structure arranges components in a grid-like pattern, where each component is connected to its neighbouring components.
- Mesh structures provide scalability and fault tolerance since messages can be re-routed around failed components.
- However, number of connections and wires increase with the size of the mesh, leads to higher complexity and increased delay for long-distance communication.



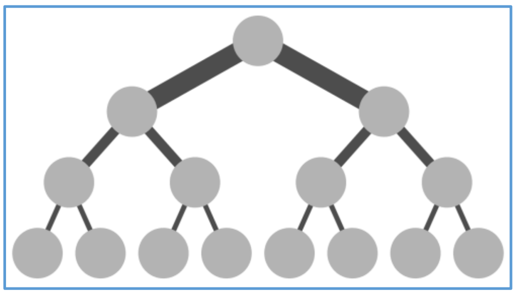
Ring



Mesh

5. **Tree:**

- A tree interconnection structure organizes components in hierarchical manner, similar to a tree structure. A central component, known as the **root**, is connected to multiple **child** components, which in turn connect to other child components, forming hierarchical tree.
- Trees are efficient for broadcast and multicast operations, but communication between non-adjacent components may require multiple steps, leading to increased delay.
- Tree networks are commonly used to arrange data in databases and workstations in corporate networks.



6. **Hypercube:**

- A hypercube interconnection structure is based on concept of n-dimensional cubes, where each dimension represents a connection. It is also known as binary n-cube multiprocessor.
- This system is composed of $N = 2^n$ processors that are linked in an n-dimensional binary cube. Each processor denotes a node of the cube.
- Components are arranged in a geometrically regular hypercube pattern, with each component connected to its adjacent components.

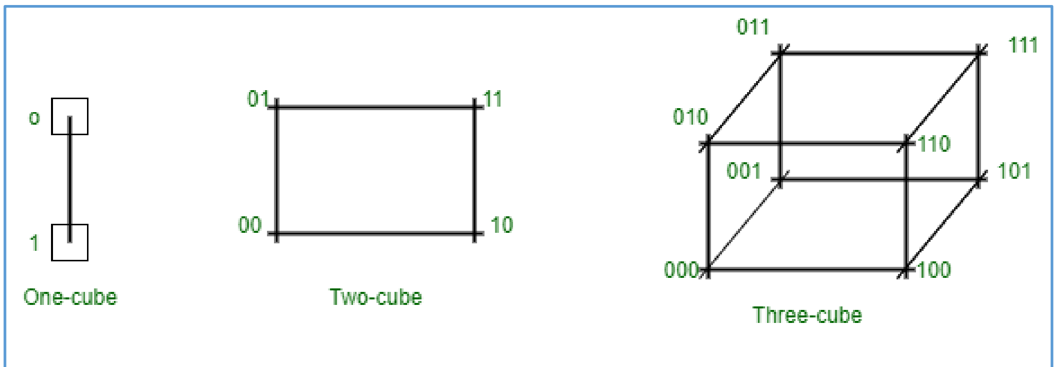


Figure - Hypercube Structures For n = 1,2,3

INTERPROCESSOR ARBITRATION:

- Computer systems need buses to facilitate the transfer of information between their various components.
- There is a dispute in the system bus that connects the CPU, Input-Output processor, and memory (main components of a computer). Only one of them between the CPU, Input-Output processor, and memory gets the grant to use the bus simultaneously.
- Hence, an appropriate priority resolving mechanism is required to decide which processor should get control of the bus. Therefore, a mechanism is needed to handle multiple requests for the bus, known as **Inter-Processor Arbitration**.
- **Inter-Processor Arbitration** is a technique used to resolve conflicts between multiple processors that are struggling for access to a shared bus. Its goal is to ensure that each processor has fair and equal access to bus.

There are several types of shared resources that processors may compete for, such as:

1. **Memory:** When multiple processors need to read from or write to the same memory location simultaneously, conflicts can occur.
2. **I/O Devices:** In systems with shared I/O devices, processors may compete for access to these devices.
3. **Shared Data Structures:** Concurrent access to shared data structures (e.g., queues, buffers, critical sections) may require arbitration.

The **goal** of **inter-processor arbitration** is to manage and prioritize access to these shared resources efficiently, while ensuring fairness and preventing conflicts that could lead to data corruption or system instability.

There are different methods for handling inter-processor arbitration include:

1. **Bus Arbitration:** In systems that use a shared bus to connect processors and other devices, bus arbitration mechanisms are employed to determine which device gets control of the bus at any given time.
2. **Centralized Arbitration:** A central controller is responsible for managing access to shared resources. When a processor needs access, it requests permission from the central controller, which then grants or denies access based on predefined rules or priorities.
3. **Distributed Arbitration:** In this approach, each processor is responsible for determining whether it can access the shared resource independently.
4. **Lock-Based Arbitration:** Processors use locks (e.g., mutexes, semaphores) to synchronize their access to shared resources. When a processor wants to access a resource, it first acquires the corresponding lock and releases it when done.
5. **Contention Resolution:** Some systems use dynamic contention (conflict) resolution techniques to dynamically adjust priorities or allocate resources based on system load, application requirements, or historical behaviour.

Advantages of inter-processor arbitration:

- It ensures that all processors have fair and equal access to the bus.
- It helps to prevent deadlocks and other bus conflict problems.
- It can improve the performance of the system by reducing the amount of time that processors have to wait for access to the bus.

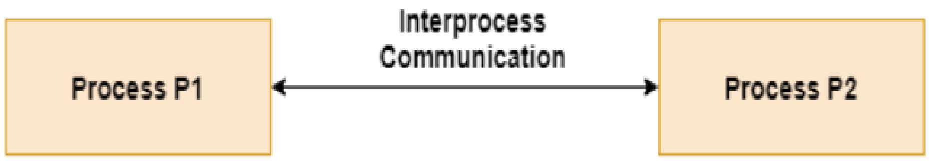
Disadvantages of inter-processor arbitration:

- It can add complexity to the system.
- It can introduce additional latency (delay) into the system.
- It can be difficult to implement in systems with a large number of processors.

INTER-PROCESSOR COMMUNICATION (IPC) & SYNCHRONIZATION:

- **Inter-processor Communication** is a mechanism of operating system that helps in communication between the several processors. It can also be defined as the process of sending information or data from one processor to another.
- It refers to communication between processors within a single computer system or among multiple computer systems.
- It also refers to the mechanisms and protocols used by multiple processors in a computer system to exchange data and coordinate their activities.
- IPC is crucial when tasks need to be distributed across multiple processors for parallel processing or when sharing data between processes executing on different cores.

A diagram that illustrates inter-process communication is as follows −



Approaches to Inter-Processor Communication:

1. **Pipe:** A pipe is a data channel that is unidirectional. Two pipes can be used to create a two-way data channel between two processes. This uses standard input and output methods.
2. **Socket:** It is the endpoint for sending or receiving data in a network. This is true for data sent between processes on same computer or data sent between different computers on the same network.
3. **File:** It is a data record that may be stored on a disk or on a file server. Multiple processes can access a file as required. All operating systems use files for data storage.
4. **Signal:** Signals are useful in inter-process communication in a limited way. They are system messages that are sent from one process to another. Normally, signals are not used to transfer data but are used for remote commands between processes.
5. **Shared Memory:** Shared memory is the memory that can be simultaneously accessed by multiple processes. This is done so that the processes can communicate with each other.
6. **Message Queue:** Multiple processes can read and write data to the message queue without being connected to each other. Messages are stored in the queue until their recipient retrieves them. Message queues are quite useful for inter-processor communication and are used by most operating systems.

Synchronization in Inter-Processor Communication:

- **Synchronization** is a necessary part of inter-process communication. It is either provided by the inter-process control mechanism or handled by the communicating processes.
- **Synchronization** refers to the coordination of activities between processors to ensure the correct order of execution and avoid race conditions, data inconsistencies, and other concurrency-related issues.
- **Race condition:** When two or more concurrent processes attempt to access and modify shared data without proper synchronization, it can lead to unpredictable and inaccurate behaviour, which is known as a **race condition**.

There are several synchronization techniques commonly used in IPC:

1. **Mutexes and Semaphores:** They are used in IPC to provide mutual exclusion and control access to shared resources. Processes use these synchronization primitives to coordinate access to shared data, ensuring that only one process can operate on the shared resource at a time.
2. **Message Passing:** In message passing-based IPC, processes communicate by sending messages to each other through message queues or channels. Synchronization is implicit in this approach, as a process waits for a message to arrive before it can process it. The message queue serves as a synchronization point, and the receiving process blocks until a message becomes available.
3. **Shared Memory with Semaphores:** When processes share a common region of memory, they must use semaphores or other synchronization primitives to protect critical sections of code. These semaphores prevent multiple processes from accessing the shared memory simultaneously and ensure data integrity.
4. **Barrier Synchronization:** It is used to ensure that all participating processes reach a specific point in their execution before any of them can proceed further. This mechanism is particularly useful in parallel processing applications where tasks need to be synchronized at specific stages.
5. **Read-Write Locks:** Read-write locks allow multiple processes to access a shared resource concurrently for reading, but only one process at a time can have exclusive access for writing. This provides higher concurrency in read operations while ensuring mutual exclusion during write operations.

IMPLEMENTATION SCHEME: PIPELINED DATA PATH AND CONTROL:

- **Pipelined data path** refers to the division of instruction execution process into multiple stages, each of which can be executed in parallel. This allows multiple instructions to be executed at the same time, which can significantly improve performance.
- **Pipelined control** refers to the logic that controls the execution of the pipelined data path. This logic must ensure that the instructions are executed in the correct order and that any data hazards are handled correctly.
- The pipelined data path and control are closely related. The **pipelined data path** defines the stages of instruction execution, and the **control** logic determines how the instructions are scheduled through these stages.
- It is a technique used to enhance the performance and throughput of a processor.
- The pipelining concept breaks down the instruction execution process into smaller stages, allowing multiple instructions to be processed simultaneously.
- Each stage of the pipeline performs a specific operation, and instructions move through the pipeline in a sequential manner.

The steps involved in implementing a pipelined data path and control are:

1. **Instruction Fetch (IF):**
    - The first stage of the pipeline is responsible for fetching instructions from memory. The program counter (PC) is used to access the instruction in memory, and the fetched instruction is stored in an instruction register.
2. **Instruction Decode (ID):**
    - In this stage, the fetched instruction is decoded to determine the type of operation to be performed and the operands involved. Control signals for subsequent stages are generated based on the instruction opcode and other relevant information.
3. **Execute (EX):**
    - This stage carries out actual calculation or operation specified by the instruction. This stage can include operations like arithmetic and logical operations, or memory access.
4. **Memory Access (MEM):**
    - In this stage, memory operations such as read/write data from/to memory, are performed. For load instructions, data is read from memory, while for store instructions, data is written into memory.
5. **Write Back (WB):**
    - The final stage of the pipeline is responsible for writing back the results of the instruction execution to the appropriate registers.

HANDLING DATA HAZARDS AND CONTROL HAZARDS:

- Handling data hazards and control hazards are essential aspects of computer architecture design to ensure efficient and correct execution of instructions in pipelined processors.
- **Data hazards** and **control hazards** are two types of hazards that can occur in pipelined processors.

1. Data Hazards:

Data hazards occur when the execution of an instruction depends on the results of a previous instruction that is still being processed in the pipeline.

There are three types of data hazards:

1) **RAW (Read After Write) hazard:**

   This occurs when an instruction tries to read a register that has been written by a previous instruction that is still in the pipeline.

2) **WAR (Write After Read) hazard:**

   This occurs when an instruction tries to write to a register that is being read by a previous instruction that is still in the pipeline.

3) **WAW (Write After Write) hazard:**

   This occurs when an instruction tries to write to a register that is already being written by a previous instruction that is still in pipeline.

Example: Suppose we have two instruction I and instruction J.

- **RAW hazard** occurs when instruction J tries to read data before instruction I writes it.

  I: <u>R2</u> <- R1 + R3

  J: R4 <- <u>R2</u> + R3

- **WAR hazard** occurs when instruction J tries to write data before instruction I reads it.

  I: R2 <- R1 + <u>R3</u>

  J: <u>R3</u> <- R4 + R5

- **WAW hazard** occurs when instruction J tries to write output before instruction I writes it.

  I: <u>R2</u> <- R1 + R3

  J: <u>R2</u> <- R4 + R5

2. Control Hazards:

Control hazards occur when the control flow of the program changes, such as when a branch instruction is executed. This can cause the pipeline to stall, as the processor must wait to determine which path the program will take.

There are several techniques that can be used to handle data hazards and control hazards in pipelined processors.

Data hazards can be handled using the following techniques:

1. **Forwarding:**
    - This technique allows the results of a previous instruction to be forwarded to a subsequent instruction that depends on them.
    - This can be done by adding special circuitry to the processor. It is the most efficient technique, but complex to implement.
2. **Code Re-ordering:**
    - This technique involves reordering the instructions in the program so that data hazards are minimized. This can be done by a compiler or by the programmer.
    - It simpler to implement, but can reduce the performance of the processor.
3. **Stall (Pause) Insertion:**
    - This technique involves stalling the pipeline until the data that is needed by a subsequent instruction is available.
    - It is the simplest technique to implement, but can reduce performance of processor.

Control hazards can be handled using the following techniques:

1. **Branch prediction:**
    - This technique involves predicting which path the program will take after a branch instruction is executed.
    - If prediction is correct, pipeline can continue to execute instructions without stalling.
    - If the prediction is incorrect, the pipeline must be flushed and restarted.
2. **Branch delay slots:**
    - This technique involves inserting an extra instruction into the pipeline after a branch instruction. This instruction is executed if the branch prediction is incorrect.