|  |  |  |
| --- | --- | --- |
| No. | Name | ID |
| 1 | Daniel kumilachew | DBU1401117 |
| 2 | Biruk Ashagre | DBU1401048 |
| 3 | Abdela Nesredin | DBU1400751 |
| 4 | Daniel Hailu | DBU1401119 |
| 5 | Biniyam Nibret | DBU1401987 |
| 6 | Kalkidan Zenebe | DBU1500138 |
| 7 | Tigist Ashenafi | DBU1402532 |
| 8 | Asregide Ferede | DBU1400928 |
| 9 | Daniel Baye | DBU1401105 |
| 10 | Dessalegne Mulat | DBU1401145 |
| 11 | Eliyas Aynekulu | DBU1402639 |
| 12 | Samuel Esubalew | DBU1402286 |
| 13 | Samuel Tale | DBU1402284 |

Debre Birhan University

Debre Birhan University

GROUP MEMBERS

**COMPUTER ORGANIZATION GROUP ASSIGNMENT**

**COLLAGE of COMPUTING**

**DEPARTMENT Of SOFTWARE ENGINEERING**

**COURSE NAME:CO**

**Sumbmitted Date:11/09/2015 sumbmitted to: t.r Yaharewerk**

h

Table of Contents

[1**. Interrupt Structure**: 2](#_Toc137937478)

[1.1 Interrupt sources: 2](#_Toc137937479)

[1.2. Interrupt types: 3](#_Toc137937480)

[1.3 Interrupt priority: 3](#_Toc137937481)

[1.4 Interrupt vectors: 3](#_Toc137937482)

[1.5 Saving and restoring context: 3](#_Toc137937483)

[1.6 Masking and clearing interrupts: 3](#_Toc137937484)

[**2.Pipe line(Arithmetic Pipeline & Instruction Pipeline ) and Vector processing: 3**](#_Toc137937485)

[2.1 Pipe line: 4](#_Toc137937486)

[2.2 Vector Processing: 6](#_Toc137937487)

[**3.Parallel processing :** 6](#_Toc137937488)

[3.1 Multiprocessor Systems: 7](#_Toc137937489)

[3.2 Multicore Systems: 8](#_Toc137937490)

[3.3 Parallelism in Instruction Level: 8](#_Toc137937491)

[3.4 Data Parallelism: 8](#_Toc137937492)

[3.5 Task Parallelism: 8](#_Toc137937493)

[**4.RAID Technology:** 9](#_Toc137937494)

[4.1 Striping: 9](#_Toc137937495)

[4.2 Mirroring: 9](#_Toc137937496)

[4.3 Parity: 9](#_Toc137937497)

[**5.Mapping Functions:** 10](#_Toc137937498)

[5.1 Memory Mapping: 10](#_Toc137937499)

[5.2 Address Space: 10](#_Toc137937500)

[5.3 Cache Mapping Techniques: 11](#_Toc137937501)

[**6.Multiprocessor & it’s characteristic:** 11](#_Toc137937502)

[6.1 Processor Count: 12](#_Toc137937503)

[6.2 Processor Interconnection: 12](#_Toc137937504)

[6. 3 Memory and Cache Organization: 12](#_Toc137937505)

[6. 4 Scalability: 12](#_Toc137937506)

[6. 5 Synchronization: 12](#_Toc137937507)

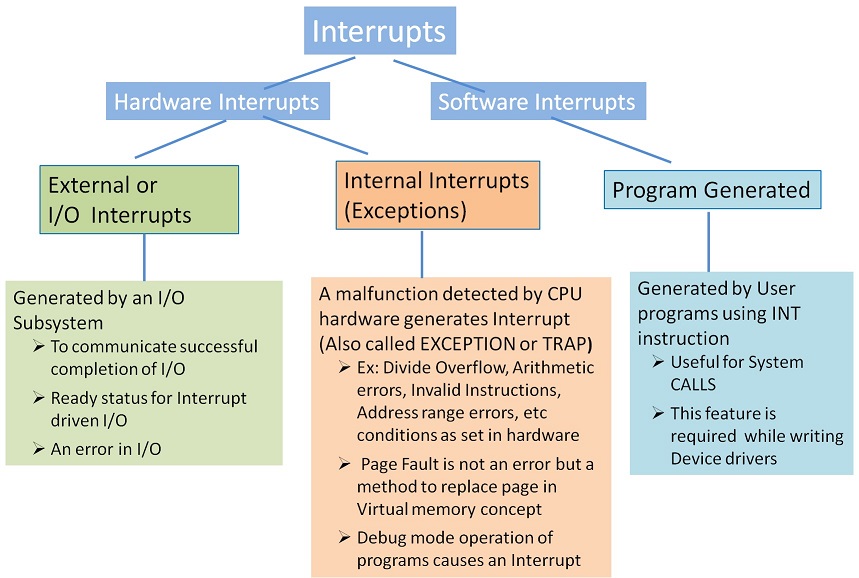
[6.6 Parallelism: 12](#_Toc137937508)

[6. 7.Performance and Efficiency: 12](#_Toc137937509)

# Interrupt Structure:

**Interrupts** are an essential aspect of computer organization and architecture, as they allow the normal flow of a program to be disrupted in response to events that require immediate attention. Interrupts are an efficient way for the computer system to respond to external or internal conditions that warrant urgent handling, such as hardware failures, pressing user requests, or completing time-critical tasks.

**Interrupt structure** refers to the precedence of interrupts. Hardware events that cause interrupts are assigned CPU interrupt levels. The CPU can disable interrupts of a certain level and below, thus allowing an important interrupt to preempt an interrupt of lower priority, but not vice-versa.



The interrupt structure can be broken down into several key components:

## 1.1 Interrupt sources:

These are events or conditions that can trigger an interrupt. Interrupt sources can be either hardware-based (e.g., a key press or a timer reaching a certain value) or software-based (e.g., an instruction explicitly executing an interrupt request).

## 1.2. Interrupt types:

Types of interrupts in computer systems, such as:

**a. Hardware interrupts:** These are triggered by physical devices connected to the computer, like a mouse, keyboard, or printer.

**b. Software interrupts:** Also known as traps or exceptions, these interrupts are caused by events within the software, such as arithmetic overflow or illegal instruction execution.

## 1.3 Interrupt priority:

Since multiple interrupts can occur simultaneously, a priority system is established to determine which interrupt should be handled first. Higher priority interrupts can interrupt the execution of lower priority ones.

## 1.4 Interrupt vectors:

These are memory addresses or pointers to the start of the interrupt service routines (ISRs), which are special functions executed when specific interrupts occur. The ISR is responsible for handling the interrupt, deciding what actions to take, and returning control back to the interrupted program.

## 1.5 Saving and restoring context:

When an interrupt occurs, the processor must save the current state of the system, including the program counter and register values. This saved state is referred to as the "context" of the interrupted program. After executing the ISR, the context must be restored so the program can continue execution where it left off.

## 1.6 Masking and clearing interrupts:

To ensure that the computer executes the ISR without getting interrupted by another request, processors can be configured to "mask" or temporarily disable further interrupts. Once the ISR has concluded, the processor "clears" or acknowledges the interrupt so that it is no longer active.6/17/2023

In summary, interrupt structures are a crucial computer organization and architecture concept for effective management of system events. They allow the computer to promptly respond to critical situations while minimizing disruptions to the ongoing program execution. Proper handling of interrupts ensures that hardware and software can communicate efficiently and maintain system stability.

# 2.Pipe line(Arithmetic Pipeline & Instruction Pipeline ) and Vector processing:

In computer organization and architecture, pipelines and vector processing are important concepts that help in enhancing the performance of a computer system. Let's discuss these concepts in detail.

## 2.1 Pipe line:

A pipeline is a technique that allows multiple instructions to be processed simultaneously by overlapping their execution, thereby improving the overall throughput of a computer system. Pipelining is a technique of decomposing a sequential process into sub operations, with each sub process being executed in a special dedicated segment that operates concurrently with all other segments.

• 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.

• Perhaps the simplest way of viewing the pipeline structure is to imagine that each segment consists of an input register followed by a combinational circuit.

o The register holds the data.

o The combinational circuit performs the sub operation in the particular segment.

• A clock is applied to all registers after enough time has elapsed to perform all segment activity.

• The pipeline organization will be demonstrated by means of a simple example.

o To perform the combined multiply and add operations with a stream of numbers

**Ai \* Bi + Ci for i = 1, 2, 3, …, 7**

• Each sub operation is to be implemented in a segment within a pipeline.

**R1 Ai, R2 Bi Input Ai and Bi**

**R3 R1 \* R2, R4 Ci Multiply and input Ci**

**R5 R3 + R4 Add Ci to product**

• Each segment has one or two registers and a combinational circuit as shown in Fig.

**Ai Bi**

|  |  |  |
| --- | --- | --- |
| **R1** |  | **R2** |

**Ci**

|  |
| --- |
| **multiplier** |

|  |
| --- |
| **R3** |

|  |
| --- |
| **R4** |

|  |
| --- |
| **adder** |

|  |
| --- |
| **R5** |

**It can be classified into two categories:**

**a) Arithmetic Pipeline:** An arithmetic pipeline deals with the parallel execution of arithmetic operations like addition, subtraction, multiplication, and division. The primary objective of an arithmetic pipeline is to minimize the time taken to complete these operations by breaking them down into smaller tasks and executing them concurrently.

For example, if we have a complex arithmetic expression like (A \* (B+C)), the arithmetic pipeline can execute the operations as follows:

**Cycle 1:** Add B and C

**Cycle 2:** Multiply A by the result from Cycle 1

This allows for faster processing of the expression compared to executing each operation sequentially.

**b) Instruction Pipeline:** Instruction pipelines, also known as Instruction Level Parallelism (ILP), involve overlapping the execution of multiple instructions. The primary objective is to improve the performance of a processor by utilizing the idle cycles in its components. An instruction pipeline typically consists of several stages, such as Fetch, Decode, Execute, Memory Access, and Writeback. Each stage is responsible for a specific task, and multiple instructions can be processed in these different stages simultaneously.

For example, consider the following instruction sequence:

**Add R1, R2, R3**

**Sub R4, R5, R6**

**Mul R7, R8, R9**

An instruction pipelined processor could process these instructions as follows:

**Cycle 1:** Fetch Add instruction

**Cycle 2:** Decode Add instruction, Fetch Sub instruction

**Cycle 3:** Execute Add instruction, Decode Sub instruction, Fetch Mul instruction

**Cycle 4:** Write back Add instruction, Execute Sub instruction, Decode Mul instruction

**Cycle 5:** Write back Sub instruction, Execute Mul instruction

**Cycle 6:** Write back instruction

As you can see, the pipeline uses the available resources efficiently by overlapping the execution of multiple instructions, leading to improved overall performance.

## 2.2 Vector Processing:

Vector processing is a technique used in high-performance computing to execute arithmetic operations on large arrays or sets of data. It is particularly suitable for applications involving scientific computing, image processing, and machine learning tasks, and is characterized by performing the same operation on a large number of data elements in parallel.

Vector processing is enabled by specialized hardware called vector processors or array processors, which have multiple processing units capable of performing arithmetic operations on vectors (arrays of data) simultaneously. These processors typically have a large number of registers to store the vectors and dedicated hardware support to perform operations like vector addition, multiplication, and dot product efficiently.

A simple example would be adding two arrays of length 'n':

**A = [a1, a2, a3, ..., an]**

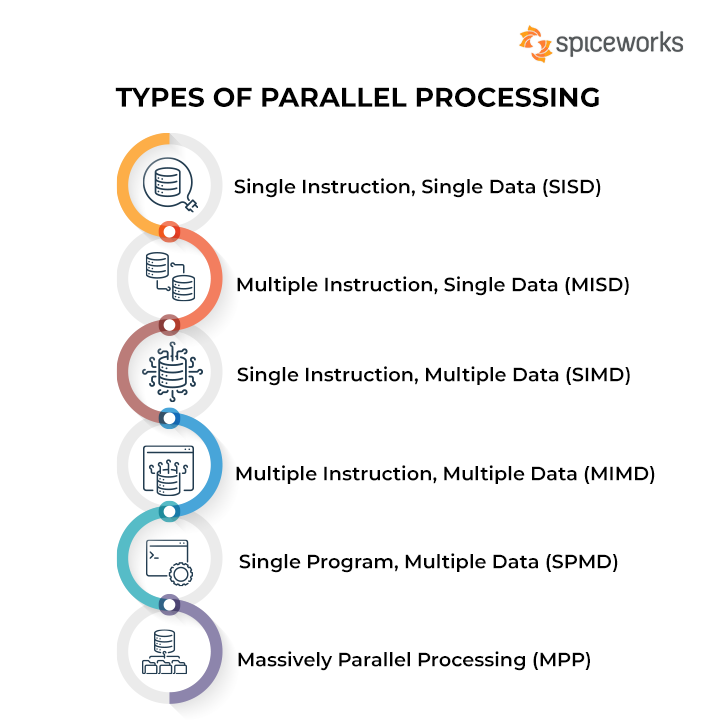
**B = [b1, b2, b3, ..., bn]**

A traditional scalar processor would perform this operation sequentially, adding each pair of elements one at a time. On the other hand, a vector processor would add the corresponding elements of the two arrays simultaneously, resulting in faster processing times and improved performance.

In summary, pipelines and vector processing are crucial concepts in computer architecture that are used to accelerate the processing of arithmetic operations and instructions. While pipelines enable concurrent execution of multiple stages of instruction processing, vector processing focuses on parallel execution of operations on large sets of data. These techniques, together, enable the development of high-performance computing systems capable of tackling complex and computationally intensive tasks.

# 3.Parallel processing :

**Parallel processing** is a technique in computer organization and architecture that involves simultaneously executing multiple tasks or processes using multiple processing units to improve the performance and speed of a computer system. It is based on the principle of dividing large problems into smaller, manageable sub-problems and then solving them concurrently. This method helps to optimize resource utilization, reduce execution time, and achieve optimal performance by leveraging the power of multiple processing units.



There are several forms and approaches to parallel processing based on computer organization and architecture concepts:

## 3.1 Multiprocessor Systems:

In a multiprocessor system, two or more processors share the same memory and other resources. These processors can execute multiple tasks in parallel, making them well-suited for computationally intensive applications. Multiprocessor systems can be further divided into:

**a. Symmetric Multiprocessing (SMP):** In SMP systems, all processors have equal access to memory and other resources. They use a common bus, and the operating system is responsible for allocating tasks to different processors.

**b. Non-uniform Memory Access (NUMA):** In NUMA systems, processors have their local memory, which they can access quickly. Access to remote memory is slower. NUMA systems aim to optimize processing time by reducing the contention for shared resources.

## 3.2 Multicore Systems:

Multicore systems have multiple processing cores integrated into a single chip. These cores can execute multiple tasks in parallel, improving performance without increasing the clock speed. This architecture is more power-efficient and is widely used in modern CPUs and GPUs.

## 3.3 Parallelism in Instruction Level:

This approach aims to execute multiple instructions simultaneously by exploiting parallelism within a single instruction stream. There are several techniques for achieving this:

**a. Pipelining:** Pipelining divides the execution of a single instruction into multiple stages, such as fetching, decoding, executing, and storing results. Each stage is performed by a separate functional unit, allowing multiple instructions to be executed in parallel at different stages.

**b. Superscalar Execution:** In superscalar processors, multiple instruction pipelines or functional units are employed, allowing for the simultaneous execution of multiple instructions per clock cycle.

**c. Vector Processing:** This approach uses specialized hardware to perform operations on large data sets, often in the form of vectors or matrices, in parallel. Vector processors can efficiently handle multimedia, scientific, and engineering tasks.

## 3.4 Data Parallelism:

Data parallelism involves the simultaneous application of the same set of operations or instructions on multiple pieces of data. This approach is well-suited for tasks that require processing large volumes of data, such as in image and signal processing. Many modern GPUs and parallel libraries, like CUDA or OpenCL, are designed with data parallelism in mind.

## 3.5 Task Parallelism:

Task parallelism is the execution of multiple independent tasks concurrently. This approach is useful for complex workflows, where different tasks may require different resources or execution times. Task parallelism can be achieved through multi-threading, where multiple threads run on separate processors or cores, or through parallel programming frameworks, like OpenMP or MPI, which distribute tasks over multiple processors in a cluster.

In conclusion, parallel processing is an essential concept in modern computer organization and architecture that significantly improves performance by leveraging multiple processing units to solve problems concurrently. Various strategies and techniques can be employed to achieve different levels of parallelism, ranging from multiprocessor systems and multicore processors to exploiting parallelism in instruction level, data level, and task level.

# 4.RAID Technology:

**RAID (Redundant Array of Independent Disks)** is a data storage technology that combines multiple disk drives to achieve increased performance, reliability, and storage capacity. RAID technology has its roots in computer organization and architecture concepts, as it involves distributing data across multiple disks in a coordinated way to optimize various aspects of computer performance.

RAID systems are commonly employed in various computer environments, including servers, data centers, and high-performance computing systems, where the protection of data and improved access speed is of utmost importance. The RAID technology is based on the concept of striping, mirroring, and parity, which helps improve performance and fault tolerance for data storage.5

# 4.1 Striping:

Striping is a technique that distributes data across multiple disk drives by breaking it into smaller segments called stripes. Each stripe is then written to a separate disk drive, allowing for parallel access to the different segments of the data. By spreading the data across multiple disks and reading or writing across them simultaneously, striping can lead to a significant increase in the read and write performance.

# 4.2 Mirroring:

Mirroring, as the name suggests, involves creating an exact copy of the data on two or more separate disk drives. This means that each disk in the mirror set contains the same data, providing redundancy and fault tolerance. In the event of a disk failure, the data can still be accessed from the remaining disk(s). Mirroring can be useful in applications where data protection is critical, but it does not provide a significant improvement in data access speed.

# 4.3 Parity:

Parity is a technique used to provide fault tolerance in RAID systems. It involves calculating parity information for the data stored on a group of disks and then storing the calculated parity on a separate disk, often called the parity disk. In case of a single disk failure, the data can be reconstructed using the available data and the parity information. This method provides redundancy without requiring a full mirror of the data, but the performance can be impacted by the overhead of calculating and storing parity.

RAID can have several levels, each combining striping, mirroring, and parity techniques in various configurations to optimize performance, redundancy, and capacity. Some of the common RAID levels include:

**1. RAID 0:** This level employs striping. It offers increased performance at the cost of data redundancy. In case one disk fails, all data on the RAID 0 array is lost.

**2. RAID 1:** This level uses mirroring. It provides high fault tolerance and data redundancy, but the usable storage capacity is halved due to mirroring.

**3. RAID 5:** This level uses striping with distributed parity. RAID 5 provides a balance between performance, data redundancy, and storage capacity. It can tolerate the failure of one disk in the array, allowing the data to be reconstructed using the parity information.

**4. RAID 6:** RAID 6 is similar to RAID 5 but uses two sets of parity information, allowing the array to tolerate the failure of two disks simultaneously.

**5. RAID 10 (1+0):** RAID 10 combines RAID 1 and RAID 0, offering the benefits of both mirroring and striping. It provides excellent performance and fault tolerance but requires a higher number of disks and lower storage efficiency.

In summary, RAID technology is an essential aspect of computer organization and architecture, which aims to improve the performance, reliability, and storage capacity of data storage systems. It uses techniques such as striping, mirroring, and parity to achieve these goals, offering various RAID levels to cater to the specific needs of different applications and systems.

# 5.Mapping Functions:

Mapping functions in computer organization and architecture refer to methods used to manage memory allocation, cache memory mapping, and address transformations. These functions are essential to ensure efficient data flow and fast processing of information in computer systems. In this discussion, we will cover memory mapping, address space, and cache mapping techniques.

## 5.1 Memory Mapping:

Memory mapping is the process of allocating memory addresses to various components of a computer system. There are two primary memory mapping techniques utilized in computer systems:

**a) Physical Address Mapping:** This is the process of assigning physical memory addresses to areas in the main memory, such as RAM. The physical address is a unique location in the memory hardware. When the CPU needs to access a particular memory location, it communicates the physical address to the memory controller.

**b) Virtual Address Mapping:** It is the process by which each process is assigned a range of virtual memory addresses. The virtual memory provides an abstraction layer between the software and hardware, allowing processes to access much larger memory space than the actual physical memory. When a process uses a virtual address, the memory management unit (MMU) translates it into a corresponding physical address.

## 5.2 Address Space:

Address space refers to the range of possible addresses that can be accessed by a computing device or program. There are two types of address space:

**a) Physical Address Space:** It is the set of all physical memory addresses available for a computer system, which includes RAM.

**b) Virtual Address Space:** It is the set of all virtual memory addresses that are available for a program or process to use. The operating system and MMU are responsible for translating between virtual and physical addresses.

## 5.3 Cache Mapping Techniques:

Caching is a technique used to store frequently accessed data in a smaller and faster memory, called cache, reducing the time needed to access the data. Cache memory is organized into cache lines, which are small blocks of continuous memory locations. There are several cache mapping techniques

**a) Direct Mapped Cache:** In this technique, each block of main memory maps to a specific cache line. The cache memory is divided into a fixed number of lines equal to the number of memory blocks. The main memory block is mapped onto a cache line using modulo operation (block number % cache lines). This is a simple and straightforward approach but suffers from cache conflicts when multiple memory blocks map to the same cache line.

**b) Fully Associative Cache:** In this technique, any block of main memory can be stored in any cache line. The cache controller searches for the data using the block number in all cache lines simultaneously (using associative memory). It offers better hit rates than direct mapped cache, but it is more complex and expensive to implement.

**c) Set Associative Cache:** This mapping technique is a hybrid of direct mapped and fully associative cache methods. The cache is divided into a number of sets, and each set contains multiple cache lines. A memory block can be stored in any cache line within its designated set, determined by a modulo operation (block number % sets). This approach offers a balance between the simplicity of direct mapped cache and the higher hit rates of a fully associative cache.

In conclusion, mapping functions in computer organization and architecture play a significant role in optimizing data access and ensuring efficient memory management. Techniques such as memory mapping, address space management, and cache mapping, ensure that modern computer systems achieve high performance and meet the computing demands of the users and applications.

# 6.Multiprocessor & it’s characteristic:

**Multiprocessor systems**, also known as parallel or concurrent processing systems, refer to computer systems with two or more Central Processing Units (CPUs), or processors. These processors are combined within a single system to execute tasks more efficiently and improve overall performance. Multiprocessor systems are a significant component in the study of computer organization and architecture, as they enable effective parallelism and concurrency in processing large volumes of data and complex tasks. Characteristics of Multiprocessor Systems:

## 6.1 Processor Count:

One of the most important characteristics of multiprocessor systems is the number of processors. The more processors there are in a system, the more tasks can be executed concurrently, improving overall performance. Multiprocessor systems may consist of a few processors or hundreds, depending on the requirements and computing workload.

## 6.2 Processor Interconnection:

In a multiprocessor system, an interconnection network is used to facilitate communication and data transfer between the different processors. This may involve shared buses or more sophisticated interconnects like crossbar switches, multistage networks, and hypercubes. Processor interconnection affects the performance, scalability, and reliability of a multiprocessor system.

## 6. 3 Memory and Cache Organization:

Multiprocessor systems can be organized based on their memory architecture. There are two major types of memory organization in multiprocessors: shared memory and distributed memory. In shared memory, all the processors have access to the same global memory, while in distributed memory, each processor has its private local memory. Cache coherence, which ensures that each processor sees the same value for a given memory location, is an essential consideration in shared memory multiprocessors.

## 6. 4 Scalability:

Scalability refers to the ability of a multiprocessor system to maintain performance as processors are added to the system. Scalability depends on factors such as the processor interconnection network and latency, as well as contention for resources like memory and input/output devices.

## 6. 5 Synchronization:

As multiprocessor systems execute tasks simultaneously on different processors, synchronization between tasks becomes critical to ensure that the system's execution is accurate and properly ordered. Synchronization mechanisms, such as barriers, locks, and other communication primitives, are used for coordinating task execution.

## 6.6 Parallelism:

One of the main goals of multiprocessor systems is to increase parallelism, the ability to execute multiple tasks simultaneously. Parallelism consists of data parallelism, in which the same operation is applied to different data elements concurrently, and control parallelism, in which different tasks are executed in parallel on separate processors. Multiprocessor systems can have different levels of parallelism, ranging from single-level parallelism (e.g., vector processors), to multiple instruction and single data (MISD), to multiple instruction and multiple data (MIMD), which is most common in multiprocessor systems.

## 6. 7.Performance and Efficiency:

The performance of a multiprocessor system is affected by various factors, such as workload characteristics, processor count, processor interconnections, memory and cache organization, and synchronization. Performance can be measured in terms of throughput, response time, and speedup over a single processor. The efficiency of a multiprocessor system often depends on how well the system can utilize its parallel processing capabilities. In conclusion, multiprocessor systems are an essential concept in computer organization and architecture. They enable parallelism and concurrency, which are critical for modern computer systems that handle large and complex workloads. Understanding the characteristics of multiprocessor systems, such as processor count, interconnection networks, memory organization, and synchronization, is crucial to designing and building efficient and high-performing systems.

Thank you!!!