NAME	
ROLL NUMBER	
SEMESTER	
COURSE CODE	DCA6105
COURSE NAME	COMPUTER ARCHITECTURE

Question 1.a.) Explain different generations of computer system.

Answer.:- Computer systems have evolved through several generations, each marked by advancements in technology, architecture, and capabilities. These generations represent significant milestones in the development of computers:

1.First Generation (1940s-1950s):

- Vacuum tubes were used for computation.
- These computers were massive, expensive, and consumed a lot of power.
- Examples include ENIAC and UNIVAC.

2.Second Generation (1950s-1960s):

- Transistors replaced vacuum tubes, reducing size, cost, and power consumption.
- Assembly languages and early high-level languages were developed.
- Magnetic core memory was introduced.
- IBM 1401 and CDC 1604 are examples of second-generation computers.

3.Third Generation (1960s-1970s):

- Integrated Circuits (ICs) were developed, allowing more components to be placed on a single chip.
- Operating systems and programming languages like COBOL and FORTRAN were widely used.
- Minicomputers became available, catering to smaller businesses.
- IBM System/360 and DEC PDP-11 are examples of third-generation computers.

4.Fourth Generation (1970s-1980s):

- Microprocessors were invented, leading to the development of personal computers.
- GUIs (Graphical User Interfaces) and networking technologies emerged.
- Laptops and home computers became more accessible to consumers.
- IBM PC, Apple Macintosh, and Commodore 64 represent fourth-generation computers.

5.Fifth Generation (1980s-Present):

- AI (Artificial Intelligence) and parallel processing technologies gained prominence.
- R&D focused on supercomputers, neural networks, and expert systems.
- Advancements in storage, networking, and connectivity continued.
- Modern PCs, smartphones, and supercomputers epitomize fifth-generation advancements.

Each generation of computers brought significant improvements in speed, size, cost-effectiveness, and functionality, transforming computing from large, room-sized machines to the compact and powerful devices we use today.

Question 1.b.) What are the differences between concurrent and parallel execution?

Answer.:- Concurrent and parallel execution are concepts related to the simultaneous execution of multiple tasks in computing, but they differ in how they manage and execute these tasks.

Concurrent Execution:

Concurrent execution refers to the ability of a system to execute multiple tasks or processes seemingly simultaneously. It doesn't necessarily mean tasks are running simultaneously at the exact moment but rather that they're being interleaved or scheduled in such a way that they appear to overlap in time.

Key characteristics of concurrent execution include:

1.Time-sharing: Tasks are executed in overlapping timeframes, allowing multiple tasks to make progress. This can occur within a single-core system through task switching or in a multi-core system where different tasks run on separate cores.

2.Concurrency Control: Mechanisms like threading and multitasking manage and control the execution of multiple tasks. These tasks might share resources and need synchronization to prevent conflicts.

3.Shared Resources: Concurrent tasks often access shared resources, and proper synchronization techniques are necessary to maintain data consistency and integrity.

Parallel Execution:

Parallel execution involves the simultaneous execution of multiple tasks or parts of a task to enhance performance and speed up computation. In parallel execution, tasks are actually being executed simultaneously at the same instant.

Key characteristics of parallel execution include:

1.Simultaneous Processing: Tasks are executed concurrently and truly simultaneously, typically leveraging multiple processing units such as multiple CPU cores, processors, or computing nodes.

2.Resource Isolation: Parallel tasks often work on isolated resources, reducing the need for complex synchronization mechanisms. Each task operates on its dedicated portion of data or resources.

3. Speed and Efficiency: Parallel execution aims to enhance performance by distributing workloads across multiple processors or cores, enabling faster computation and improved throughput.

In summary, concurrent execution focuses on managing multiple tasks to provide an appearance of simultaneity, whereas parallel execution involves the actual simultaneous execution of tasks to achieve speedup and enhanced performance through multiple processing units.

Question 2.a.) Explain zero, one, two and three address instruction with the help of given instruction X=(A/B)+(C*D)

Answer.:- In computer architecture and assembly language programming, the number of addresses an instruction refers to indicates the number of operands the instruction operates on. Zero, one, two, and three-address instructions differ based on the number of operands they handle.

Zero Address Instruction:

Zero-address instructions are stack-based instructions that don't explicitly specify operands. Instead, they work with values implicitly from the top of the stack. These instructions are commonly used in stack-oriented architectures. For instance, in a stack-based system, an instruction like `ADD` might add the top two values on the stack.

One Address Instruction: One-address instructions involve one explicit operand along with the implied accumulator (or register) where the result is stored. For instance, an instruction like `INC A` (increment A) increments the value in register A by one.

Two Address Instruction: Two-address instructions work with two explicit operands, and typically, one operand is the destination where the result is stored. An example would be `MOV A, B` (move the content of register B to register A), where A is the destination and B is the source.

Three Address Instruction: Three-address instructions involve operations with three explicit operands, commonly used in higher-level programming languages and compilers. Each operand serves a specific role, such as source, destination, and intermediary values during an operation.

Let's analyze the expression `X=(A/B)+(C*D)` in the context of three-address instructions:

1. `A/B` can be represented as a three-address instruction:

   `DIV A, B, Temp1` (Divide A by B and store the result in Temp1).

2. `C*D` can be another three-address instruction:

   `MUL C, D, Temp2` (Multiply C by D and store the result in Temp2).

3. Finally, `(A/B)+(C*D)` involves an addition operation:

   `ADD Temp1, Temp2, X` (Add Temp1 and Temp2, store the result in X).

In this example, each arithmetic operation corresponds to a three-address instruction, with each instruction having three explicit operands: source1, source2, and destination, demonstrating the concept of three-address instructions in expressing computations.

Question 2.b.) Discuss different types of addressing modes in detail.

Answer.:- Addressing modes in computer architecture define how instructions specify the location of operands. These modes vary in their ways of accessing data from memory or registers during instruction execution, providing flexibility in programming and efficient memory management. Common addressing modes include:

1.Immediate Addressing:

- In this mode, operands' values are directly specified within the instruction. For instance, an instruction might contain a constant or immediate value like `ADD R1, 5`, where 5 is the immediate operand.

2.Direct Addressing:

- This mode uses a memory address directly to access data. For example, `MOV R2, [1000]` copies the content of memory location 1000 into register R2.

3.Indirect Addressing:

- Indirect addressing uses a memory location that contains the address of the operand. For instance, `MOV R3, [R1]` copies the content of the memory location pointed by the address in register R1 into register R3.

4.Register Addressing:

- In this mode, operands reside in registers. Instructions work directly with register contents. For example, `ADD R1, R2` performs addition on the contents of registers R1 and R2.

5.Register Indirect Addressing:

- Similar to indirect addressing, this mode uses the content of a register as an address to access data. For instance, `MOV R4, [R3]` moves the content of the memory location pointed by the address in register R3 into register R4.

6.Indexed Addressing:

- Indexed addressing involves adding an offset to a base address specified in a register to access memory. For example, `LOAD R5, [R6 + 10]` loads the content at the memory address formed by adding 10 to the content of register R6 into register R5.

These addressing modes enable efficient manipulation and access to data by providing various ways to specify operands, allowing for flexibility in program design and optimization for memory utilization and computation efficiency. Different architectures might support a subset of these modes or introduce additional specialized modes based on specific requirements.

Question 3.a.)Explain different types of techniques to handle hazards in pipelining.

Answer.:- Hazards in pipelining occur due to dependencies among instructions, potentially causing stalls or incorrect execution. Various techniques are employed to mitigate these hazards and maintain the pipeline's efficiency:

1.Data Hazard (or Data Dependency):

Forwarding (or Data Bypassing): Also known as data forwarding, this technique allows data to bypass intermediate pipeline stages and be directly forwarded from one functional unit to another. By forwarding data from the output of one stage to the input of another, stalls caused by data dependencies are reduced or eliminated.

2. Control Hazard (or Branch Hazard):

Branch Prediction: Predictive techniques anticipate the outcome of conditional branches before they are resolved. Speculative execution or prediction algorithms (like static prediction or dynamic prediction using branch history) enable the pipeline to continue fetching and executing instructions based on the predicted branch outcome.

3.Structural Hazard:

Resource Duplication: This involves duplicating hardware resources to eliminate structural hazards. For instance, having multiple functional units or duplicating parts of the pipeline can allow multiple instructions with conflicting resource needs to be executed simultaneously.

4.Pipeline Interlocking (or Stalling):

Stall or Bubble insertion: Inserting bubbles (no-operation instructions) into the pipeline when a hazard is detected. This allows time for resolving the hazard, maintaining the correctness of execution but sacrificing throughput.

5. Compiler Techniques:

Instruction Reordering: Reordering instructions by the compiler to minimize dependencies and hazards. Techniques like instruction scheduling or loop unrolling can reduce hazards by rearranging the order of instructions.

6. Software Pipelining:

Loop Unrolling: Involves duplicating loop bodies to expose more instruction-level parallelism, reducing pipeline stalls by increasing the number of independent instructions in the pipeline.

By employing these techniques individually or in combination, modern processors aim to minimize pipeline stalls, hazards, and dependencies, thereby maximizing throughput and improving overall performance in pipelined architectures.

Question 3.b.) Differentiate between unconditional and conditional branch.

Answer.:- Unconditional and conditional branches are two types of instructions used in computer programming and assembly language, serving different purposes in controlling program flow.

Unconditional Branch:

An unconditional branch, also known as a jump instruction, directs the program to jump to a different location in the code unconditionally, regardless of any conditions or flags. It alters the program counter (PC) or instruction pointer to transfer control to a specific memory address, usually specified explicitly in the instruction.

Key characteristics of unconditional branches include:

1. Direct Jump: It transfers program control directly to a designated memory address without any condition evaluation.

2.Examples: Instructions like `JMP` (Jump), `CALL` (Subroutine Call), or `RET` (Return) in assembly language are typically unconditional branches.

3.Usage: Unconditional branches are often used for implementing loops, function calls, handling exceptions, or transferring program control to different parts of the code unconditionally.

Conditional Branch:

A conditional branch allows the program to make decisions based on certain conditions or flags' evaluation. It changes the program flow depending on the result of a condition, such as comparing values, checking status flags, or evaluating logical expressions. If the condition is met, the branch is taken; otherwise, the program continues to execute the next sequential instruction.

Key characteristics of conditional branches include:

1.Conditional Evaluation: It evaluates a condition or set of conditions and branches based on the result (e.g., equality, inequality, greater than, less than).

2.Examples: Instructions like `JZ` (Jump if Zero), `JNZ` (Jump if Not Zero), `JE` (Jump if Equal), or `JG` (Jump if Greater) in assembly language are conditional branches.

3.Usage: Conditional branches are crucial for implementing if-else statements, loops (like while, for), and decision-making structures in programs based on different scenarios and conditions.

In summary, unconditional branches unconditionally transfer program control to a specified location, while conditional branches alter the program flow based on the evaluation of specific conditions or flags, allowing for decision-making within the program's execution.

Question 4.a.) What is mapping? Explain different types of mapping in detail.

Answer.:- Mapping in computer science refers to the process of associating elements or data from one domain to another, often involving the translation, transformation, or assignment of values between different spaces or structures. There are various types of mapping techniques used in different contexts:

1.Memory Mapping: -

Address Mapping: In computer memory, address mapping refers to the translation of logical addresses used by programs into physical addresses in the memory. Techniques like paging and segmentation manage this mapping, allowing efficient memory allocation and management.

2.Data Mapping:

Entity-Relationship Mapping: In database systems, data mapping involves associating entities, attributes, or relationships in the real world with their representations in the database schema. It ensures the accurate representation of real-world entities within the database structure.

3.Function Mapping:

Mathematical Function Mapping: In mathematics, function mapping refers to the relationship between elements of two sets, where each element in the first set is paired with exactly one element in the second set. Functions define these mappings, associating inputs with corresponding outputs.

4.Hardware Mapping:

Input-Output (I/O) Mapping: In computer systems, hardware mapping involves assigning specific memory addresses or ports to interact with peripheral devices. Input-output mapping connects memory locations to hardware registers, enabling data transfer between the CPU and peripherals.

5.Geographical Mapping:

GIS Mapping: Geographic Information Systems (GIS) use geographical mapping to associate real-world geographical data with digital maps. GIS technologies map real-world features like terrain, cities, or climate data onto digital representations for analysis and visualization.

Each type of mapping involves establishing relationships, associations, or correspondences between different entities, domains, or spaces, enabling efficient data management, addressing, or translation between different systems or structures. The choice of mapping technique depends on the specific context and requirements of the application or system being designed or used.

Question 4.b.) What are vector processors? Explain different types of vector processing.

Answer.:- Vector processors are specialized central processing units (CPUs) designed to perform operations on multiple data elements simultaneously, known as vectors. They excel at handling tasks involving large amounts of data by applying a single operation to multiple elements in parallel, significantly enhancing computational efficiency.

Different types of vector processing include:

1.Single Instruction, Single Data (SISD): SISD architecture represents traditional processors where a single instruction operates on a single piece of data at a time. It doesn't exploit parallelism inherent in vector operations.

2.Single Instruction, Multiple Data (SIMD): SIMD architecture processes multiple data elements using a single instruction simultaneously. It executes the same operation on different data elements in parallel. For instance, adding two arrays of numbers together in one instruction.
3.Multiple Instruction, Single Data (MISD): MISD architecture, although less common, involves multiple instructions acting on a single piece of data concurrently. It's not widely used due to limited practical applications.
4.Multiple Instruction, Multiple Data (MIMD): MIMD architecture involves multiple instructions operating on multiple data elements independently. Each processor executes its own set of instructions on its data. This form is typical in parallel processing systems and multiprocessor systems.

Vector processors primarily employ SIMD architecture to handle vectors efficiently. These processors have specialized hardware, such as vector registers and execution units, optimized for performing operations on vectors. They excel in scientific simulations, signal processing, image and video processing, and other tasks involving extensive data manipulation.

Modern CPUs often integrate SIMD capabilities (such as Intel's SSE, AVX, or ARM's NEON) alongside traditional scalar processors. These SIMD instructions allow regular CPUs to perform certain vector operations efficiently, though dedicated vector processors still offer superior performance for heavy vectorized workloads.

Question 5.a.) Explain fine-grained and coarse-grained architecture. Discuss differences between them.

Answer.:- Fine-grained and coarse-grained architectures refer to different approaches in designing parallel computing systems, indicating the level of granularity at which tasks are divided and managed for execution.

Fine-Grained Architecture:

Fine-grained architectures involve breaking tasks or computations into small, fine-grained units, typically with small execution times. These systems distribute and manage these smaller units across multiple processing elements or cores. Each processing unit handles a tiny portion of the overall task, enabling high concurrency and efficient resource utilization.

Key characteristics of fine-grained architectures include:
1.Small Task Granularity: Tasks are divided into very small units, often requiring minimal computational effort to execute.
2.High Concurrency: Numerous small tasks can be executed concurrently on different processing elements.
3.Low Communication Overhead: Task communication and synchronization overhead are relatively low due to the small size of tasks.

Coarse-Grained Architecture:

Coarse-grained architectures involve dividing tasks or computations into larger, more substantial units before distribution across processing elements. These units are more significant in size and require more computational effort, potentially taking longer to complete. Coarse-grained systems focus on managing larger workloads, aiming to reduce communication overhead and increase efficiency for bigger tasks.

Key characteristics of coarse-grained architectures include:
1.Large Task Granularity: Tasks are divided into more substantial units, requiring more computational effort to execute compared to fine-grained systems.

2.Reduced Concurrency: Fewer, larger tasks are executed concurrently across processing elements, reducing the number of concurrent operations.

3.Higher Communication Overhead: As tasks are larger, coordination, communication, and synchronization between processing elements can result in higher overhead.

Differences:

1. Granularity: Fine-grained architectures use smaller task granularity, while coarse-grained architectures operate with larger task granularity.
2.Concurrency: Fine-grained systems offer higher concurrency due to smaller tasks, while coarse-grained systems have reduced concurrency but deal with larger workloads efficiently.
3.Communication Overhead: Fine-grained architectures generally have lower communication overhead due to smaller tasks, while coarse-grained architectures may experience higher communication overhead due to larger task sizes.

Both architectures have their advantages and trade-offs, and the choice between fine-grained and coarse-grained systems depends on the nature of the workload, the level of parallelism required, and the efficiency in managing communication and synchronization overhead.

Question 5.b.) Explain the different types of Storage devices.

Answer.:- Storage devices encompass a range of hardware used to store and retrieve data. These devices vary in terms of capacity, speed, portability, and access methods. Some common types include:
1.Hard Disk Drives (HDDs): HDDs use rotating magnetic platters to store data. They offer high capacities at relatively lower costs, making them suitable for storing large amounts of data in desktops, laptops, and servers. However, they're slower compared to solid-state drives (SSDs).
2.Solid-State Drives (SSDs): SSDs use flash memory to store data. They provide faster read/write speeds, lower power consumption, and greater durability compared to HDDs. SSDs are commonly used in laptops, desktops, and modern storage arrays where speed is crucial.
3.Flash Drives (USB Drives): These are portable storage devices that use flash memory. They are compact, portable, and offer varying capacities. USB drives are commonly used for data transfer and as portable backups due to their convenience.
4.External Hard Drives: External HDDs or SSDs provide additional storage capacity that can be easily connected to computers via USB, Thunderbolt, or other interfaces. They are used for backups, additional storage, or as portable storage solutions.
5.Optical Drives: Devices like CDs, DVDs, and Blu-ray discs serve as optical storage. While becoming less common due to advancements in other storage technologies, they are still used for media distribution, data backups, and software installations.
6.Tape Drives: Tape drives use magnetic tape to store data sequentially. They offer high capacities and are commonly used for long-term archival storage due to their low cost per gigabyte.
7.Cloud Storage: Cloud storage involves storing data on remote servers accessed over the internet. Services like Google Drive, Dropbox, and Amazon S3 offer scalable storage solutions accessible from various devices.
Storage devices come in various forms, catering to different needs based on speed, capacity, portability, and cost considerations. Advances in technology continually improve storage solutions, offering higher capacities, faster speeds, and more reliable data storage options.

Question 6.a.) What is RAID? Explain RAID0,RAID1, RAID3 and RAID5.

Answer.:- RAID (Redundant Array of Independent Disks) is a data storage technology that combines multiple physical drives into a single logical unit. It offers various levels or configurations (RAID levels) to enhance performance, reliability, or a combination of both.

1.RAID 0 (Striping): RAID 0 distributes data across multiple drives (at least two) to improve performance. It stripes data across the drives, splitting it into small segments written across disks simultaneously. It offers increased read/write speeds since data is accessed in parallel across drives. However, RAID 0 doesn't provide redundancy, so if one drive fails, all data is lost.

2.RAID 1 (Mirroring): RAID 1 creates an exact copy (mirror) of data across two drives. Both drives contain the same information, providing redundancy. If one drive fails, the other continues to function, ensuring data integrity and availability. However, RAID 1 doesn't enhance read or write performance as data is duplicated.

3.RAID 3 (Byte-Level Striping with Dedicated Parity): RAID 3 stripes data at a byte level across multiple drives and dedicates one drive for parity information. Parity is a calculated value used to reconstruct data if one drive fails. It offers improved performance for sequential reads but limited parallelism for writes due to the dedicated parity drive. If any drive fails, the parity information can rebuild the lost data.

4.RAID 5 (Block-Level Striping with Distributed Parity): RAID 5 stripes data at a block level across multiple drives, distributing parity information across all drives instead of a dedicated drive. This provides both improved read and write performance compared to RAID 3. RAID 5 offers fault tolerance; if one drive fails, data can be reconstructed using parity information from other drives.

Each RAID level offers different trade-offs between performance, capacity, and redundancy. Selection depends on specific needs, balancing factors like data protection, performance requirements, and cost-effectiveness in storage solutions.

Question 6.a.) What is multithreading? Explain its advantages.

Answer.:- Multithreading is a programming and execution model that allows multiple threads within a process to execute concurrently. Each thread represents an independent flow of execution, enabling different parts of a program to run simultaneously.

Advantages of Multithreading:

1. mproved Responsiveness: Multithreading enhances system responsiveness by allowing concurrent execution of tasks. For instance, in graphical user interfaces (GUIs), one thread can handle user interactions while another performs background tasks, preventing the UI from becoming unresponsive.

2. Enhanced Performance: Multithreading leverages multiple cores or CPUs effectively, improving overall system performance. It enables efficient utilization of available resources, particularly in multi-core processors, where threads can execute concurrently, speeding up computation.

3.Resource Sharing: Threads within the same process share the same memory space, allowing efficient sharing of resources like data and memory. This eliminates the need for inter-process communication (IPC) and simplifies data exchange between threads.

4. Scalability: Multithreading enhances scalability by allowing programs to efficiently utilize resources in varying conditions. It enables applications to perform well on both single-core and multi-core systems, adapting to different hardware configurations.

5.Simplified Programming: Multithreading simplifies complex tasks by breaking them into smaller, manageable threads. It enables concurrent execution of different aspects of a program, making it easier to design and implement parallelizable tasks.

6.Faster Task Completion: By allowing different threads to execute simultaneously, multithreading can expedite the completion of tasks, particularly in applications with numerous parallelizable operations, such as web servers handling multiple requests concurrently.

However, multithreading also introduces challenges such as thread synchronization, resource sharing conflicts, and potential issues like race conditions or deadlocks. Proper synchronization mechanisms and careful programming practices are essential to harness the benefits of multithreading while ensuring correctness and stability in applications.