1.“The program execution in a computer involves performing instructions cycles, which includes

two types of activities”. What are those?

Ans;

The two types of activities involved in program execution cycles in a computer are:

1. Fetch: In this step, the computer retrieves the next instruction from the program stored in memory.
2. Execute: In this step, the computer carries out the instruction fetched in the previous step. This may involve performing calculations, reading or writing data, or transferring control to another part of the program.

These two steps are repeated for each instruction in the program until the program is completed or terminated. This cycle of fetch and execute is commonly known as the "instruction cycle" or "fetch-execute cycle".

2.What are the two basic strategies used to improve the performance of a computer?

Ans;

The two basic strategies used to improve the performance of a computer are:

**Hardware upgrades:** This strategy involves upgrading the physical components of the computer, such as the processor, RAM, hard drive, and graphics card. By upgrading these components, the computer can perform tasks more quickly and efficiently, resulting in improved performance.

**Software optimization:** This strategy involves optimizing the software and operating system on the computer to improve performance. This can include removing unnecessary programs, updating drivers, defragmenting the hard drive, and tweaking system settings to optimize performance. Software optimization can often result in significant improvements in performance without the need for hardware upgrades.

3.What are the major differences between instruction-level parallelism and processor-level parallelism?

Ans;

Instruction-level parallelism and processor-level parallelism are two different approaches to achieve parallelism in computer systems. The major differences between these two approaches are as follows:

Definition:

Instruction-level parallelism refers to the ability of a processor to execute multiple instructions simultaneously within a single program. It involves identifying and executing multiple independent instructions within a program at the same time.

Processor-level parallelism refers to the use of multiple processors or cores within a single computer system to perform multiple tasks simultaneously. It involves dividing a program or set of instructions into smaller parts and executing them in parallel across multiple processors or cores.

Scope:

Instruction-level parallelism focuses on the internal architecture of a single processor. It involves optimizing the processor design to identify and execute independent instructions simultaneously to increase the overall performance of the processor.

Processor-level parallelism involves using multiple processors or cores to execute multiple tasks simultaneously. It requires a higher level of coordination between processors or cores and may involve communication and synchronization overheads.

Techniques:

Instruction-level parallelism is achieved through techniques such as pipelining, superscalar execution, and out-of-order execution. These techniques allow multiple instructions to be executed simultaneously by breaking them down into smaller stages and executing them in parallel.

Processor-level parallelism is achieved through techniques such as multiprocessing, multithreading, and SIMD (Single Instruction Multiple Data) processing. These techniques involve dividing a program or set of instructions into smaller parts and executing them in parallel across multiple processors or cores.

Performance:

Instruction-level parallelism can improve the performance of a single processor by allowing multiple instructions to be executed simultaneously. However, the performance gains may be limited by factors such as data dependencies, branch mispredictions, and resource contention.

Processor-level parallelism can provide significant performance gains by allowing multiple tasks to be executed simultaneously across multiple processors or cores. However, the performance gains may be limited by factors such as communication and synchronization overheads, load balancing, and scalability.

4. How can you classify parallelism according to Flynn?

Ans;

Parallelism can be classified according to Flynn's taxonomy, which was proposed by Michael J. Flynn in 1966. Flynn's taxonomy classifies parallelism based on the number of instructions and data streams that can be processed simultaneously by a computer system. According to Flynn's taxonomy, there are four main categories of parallelism:

**Single Instruction Single Data (SISD):**

SISD refers to the traditional sequential processing model, where a single processor executes a single instruction at a time on a single stream of data. This is the simplest form of processing and does not involve any parallelism.

**Single Instruction Multiple Data (SIMD):**

SIMD refers to a processing model in which a single instruction is executed simultaneously on multiple streams of data. In SIMD processing, a single processor executes the same instruction on multiple pieces of data simultaneously. This is commonly used in applications such as multimedia processing, image processing, and scientific computing.

**Multiple Instruction Single Data (MISD):**

MISD refers to a processing model in which multiple instructions are executed simultaneously on a single stream of data. This type of processing is less common than the other types of parallelism and is mainly used in specialized applications such as fault-tolerant systems.

**Multiple Instruction Multiple Data (MIMD):**

MIMD refers to a processing model in which multiple instructions are executed simultaneously on multiple streams of data. This is the most general form of parallel processing, and it involves multiple processors or cores working together to execute different instructions on different data simultaneously. MIMD processing is commonly used in parallel computing systems, supercomputers, and distributed computing environments.

In summary, Flynn's taxonomy classifies parallelism based on the number of instructions and data streams that can be processed simultaneously. The four main categories of parallelism are SISD, SIMD, MISD, and MIMD.

5. Is an MISD system practically feasible? If yes, give an example. If no, justify the reason.

Ans;

An MISD (Multiple Instruction Single Data) system is theoretically possible, but it is not practically feasible for several reasons.

Firstly, the concept of executing multiple instructions on a single data stream is not very useful in most practical applications. Most applications require different operations to be performed on different data sets, and executing multiple instructions on the same data set is not efficient.

Secondly, MISD architectures are difficult to implement because they require multiple independent processing units to execute different instructions on the same data stream. This type of architecture would require complex synchronization and coordination mechanisms to ensure that the processing units do not interfere with each other.

Finally, the MISD architecture is not suitable for fault-tolerant systems because the redundancy required to implement the architecture is not cost-effective.

As a result, MISD architectures are not used in any mainstream computing systems. However, some specialized applications, such as fault-tolerant systems, may use MISD architectures for specific purposes. For example, redundant computing systems may use MISD to process multiple copies of the same data and compare the results to ensure that the system is functioning correctly.

6. What is the basic difference between scalar and vector processors? Explain.

Ans;

Scalar processors and vector processors are two types of processors that differ in how they execute instructions.

**Scalar Processor:**

A scalar processor is a traditional processor that operates on scalar values (i.e., single data items) one at a time. It fetches a single instruction at a time, performs the operation specified by the instruction on the operands, and stores the result in memory or a register. Scalar processors are optimized for general-purpose computing, where a single instruction operates on a single data item.

**Vector Processor:**

A vector processor is a processor that operates on vectors (i.e., arrays of data) using a single instruction. The processor fetches a single instruction that specifies the operation to be performed on all the elements in a vector. The processor then performs the same operation on all the elements in the vector simultaneously, using specialized hardware that can perform the operation in parallel.

The basic difference between scalar and vector processors is that scalar processors operate on individual data items, while vector processors operate on arrays of data simultaneously. Scalar processors are optimized for sequential processing, where each instruction operates on a single data item at a time.

Vector processors are optimized for parallel processing, where a single instruction operates on multiple data items in parallel.

Vector processors are particularly well-suited for applications such as scientific computing, where operations can be performed on large arrays of data in parallel.

In contrast, scalar processors are better suited for general-purpose computing, where each instruction operates on a single data item at a time.

7. What is an MFU? How does it help superscalar processing?

Ans;

MFU stands for Most Frequently Used, and it is a mechanism used in superscalar processors to optimize instruction execution. The MFU selects the most frequently used instructions for execution, which can help improve the performance of the processor.

In superscalar processing, the processor fetches and executes multiple instructions simultaneously, allowing for parallel processing of instructions. However, not all instructions can be executed in parallel, and some instructions may have to wait for resources or dependencies to be resolved. This can cause delays and decrease performance.

8. What is the difference between an instruction pipeline and an arithmetic pipeline?

Ans;

An instruction pipeline and an arithmetic pipeline are two types of pipelines used in computer architecture that differ in their purpose and operation.

Instruction Pipeline:

An instruction pipeline is a pipeline used in computer processors to execute instructions in parallel. The pipeline is divided into multiple stages, each of which performs a specific task, such as fetching an instruction, decoding the instruction, executing the instruction, and writing back the results. As each stage completes its task, the next instruction can be fetched and processed in parallel, allowing for faster and more efficient execution of instructions.

Arithmetic Pipeline:

An arithmetic pipeline is a pipeline used to execute arithmetic operations, such as addition or multiplication, in parallel. The pipeline is divided into multiple stages, each of which performs a specific arithmetic operation on a portion of the data. As each stage completes its operation, the results are passed to the next stage, where additional operations are performed, until the final result is produced.

The main difference between an instruction pipeline and an arithmetic pipeline is their purpose and operation.

An instruction pipeline is used to execute instructions in parallel, while an arithmetic pipeline is used to execute arithmetic operations in parallel.

Instruction pipelines are used to improve the overall performance of a processor by executing multiple instructions simultaneously, while arithmetic pipelines are used to improve the performance of arithmetic operations by executing them in parallel.

9. What do you mean by pipeline efficiency? Explain

Ans;

Pipeline efficiency refers to the effectiveness of a pipeline in executing instructions or processing data. It is a measure of how well the pipeline is utilized and how much performance improvement is achieved compared to a non-pipelined implementation.

The efficiency of a pipeline is affected by various factors such as pipeline length, pipeline hazards, pipeline stages, clock speed, and the nature of the workload. A pipeline is said to be efficient if it is able to execute instructions or process data at a faster rate than a non-pipelined implementation.

Pipeline efficiency can be measured using the pipeline throughput, which is the number of instructions or data processed per unit time. The pipeline throughput depends on the clock speed, the number of pipeline stages, the number of instructions or data processed per stage, and the frequency of pipeline hazards.

10. Name different pipeline hazards and their primary sources?

Ans;

There are three main types of pipeline hazards that can occur in a pipelined processor:

**Structural Hazards**: Structural hazards occur when multiple instructions require the same hardware resource at the same time. For example, if two instructions require the use of the same functional unit, such as the ALU or the memory unit, the pipeline may have to stall or delay the execution of one of the instructions. Structural hazards are caused by limited resources or conflicting requirements of instructions.

**Data Hazards:** Data hazards occur when an instruction depends on the results of a previous instruction that has not yet been completed. For example, if an instruction requires the result of a previous instruction that is still being processed in the pipeline, the pipeline may have to stall or delay the execution of the current instruction. Data hazards are caused by dependencies between instructions.

**Control Hazards:** Control hazards occur when an instruction depends on the outcome of a previous instruction that changes the program flow, such as a branch or jump instruction. For example, if a branch instruction is executed in the pipeline, the pipeline may have to flush or discard the instructions that were fetched after the branch instruction. Control hazards are caused by changes in the program flow that affect the pipeline's ability to fetch and execute instructions.

**The primary sources of pipeline hazards are:**

Dependencies between instructions, such as data dependencies or control dependencies, which can cause data hazards or control hazards.

Limited hardware resources, such as functional units or memory units, which can cause structural hazards.

Branch instructions or other control flow instructions that change the program flow, which can cause control hazards.

11. What is chaining used in vector processors? Explain

Ans;

Chaining is a technique used in vector processors to execute a series of vector operations in a single instruction cycle. In chaining, the output of one vector operation is used as the input for the next vector operation, and the results are accumulated in a register. This allows multiple vector operations to be executed in a pipelined fashion, without the need to store intermediate results in memory.

Chaining works by breaking down a long vector operation into smaller sub-operations, each of which is executed in a single clock cycle. The output of each sub-operation is then used as the input for the next sub-operation, forming a chain of operations. The final result is accumulated in a register, which can be used as the input for the next vector operation.

For example, consider the following vector operation in a vector processor:

C = A + B

where A, B, and C are vectors of the same length. This operation can be broken down into smaller sub-operations, such as:

C1 = A1 + B1

C2 = A2 + B2

...

Cn = An + Bn

where Ai, Bi, and Ci are the i-th elements of vectors A, B, and C, respectively.

Each sub-operation can be executed in a single clock cycle, and the output of each sub-operation can be used as the input for the next sub-operation, forming a chain of operations. The final result can be accumulated in a register, which can be used as the input for the next vector operation.

Chaining allows vector processors to achieve high levels of performance by executing multiple vector operations in parallel, without the need to store intermediate results in memory. However, chaining requires careful design and implementation to ensure that the sub-operations can be executed in a pipelined fashion, without introducing pipeline hazards or other performance bottlenecks.

12. What is the effect of the branch instruction in a pipeline? State with a diagram

Ans;

The branch instruction in a pipeline can have a significant effect on pipeline performance. When a branch instruction is encountered in the pipeline, the pipeline must decide whether to continue fetching and executing instructions from the current program path or to switch to a new program path based on the branch condition. This decision may not be known until the branch instruction has completed execution, which can cause a pipeline stall or delay.

The following diagram shows the effect of a branch instruction in a typical 5-stage pipeline:

IF ID EX MEM WB

----------------------------

| 1 | 2 | 3 | 4 | 5 | Initial instruction sequence

----------------------------

| 1 | 2 | 3 | | | Fetch stage of branch instruction

----------------------------

| 2 | 3 | | | | Fetch stage of new program path

----------------------------

| 2 | 3 | 4 | | | Execution stage of branch instruction

----------------------------

| | | | | | Pipeline flush due to branch instruction

----------------------------

| | | | | | Fetch stage after pipeline flush

----------------------------

| 6 | 7 | 8 | 9 | 10 | Fetch stage of new program path

----------------------------

In the initial instruction sequence, the pipeline is executing a series of instructions numbered 1-5. When the branch instruction is encountered in the fetch stage, the pipeline must decide whether to take the branch or continue with the current program path. If the branch is taken, the pipeline must flush the instructions fetched after the branch instruction and begin fetching instructions from the new program path.

In the second row of the diagram, the pipeline is in the fetch stage of the branch instruction, and the instructions fetched after the branch instruction are discarded. In the third row, the pipeline is fetching instructions from the new program path, which begins with instruction 2. In the fourth row, the pipeline is executing the branch instruction, and the decision to take the branch is made. In the fifth row, the pipeline is flushed due to the branch instruction, and the instructions fetched after the branch instruction are discarded. In the sixth row, the pipeline begins fetching instructions from the new program path again, starting with instruction 6.

The branch instruction in this example caused a pipeline stall and a pipeline flush, which can significantly impact pipeline performance. To mitigate the impact of branch instructions, techniques such as branch prediction and speculative execution can be used to improve pipeline efficiency and reduce pipeline stalls.

13. What are the primary aspects a computer architect should consider before finalizing an instruction set?

Ans;

There are several primary aspects that a computer architect should consider before finalizing an instruction set:

**Functionality:** The instruction set should provide a comprehensive set of instructions that can perform all the required operations. The architect should ensure that the instruction set includes instructions for arithmetic and logic operations, data movement, control flow, and other essential operations.

**Performance:** The instruction set should be designed to maximize performance. This can be achieved by reducing the number of instructions required to perform a particular operation, minimizing the number of memory accesses, and ensuring that the instruction set is compatible with the underlying hardware.

**Code density:** The instruction set should be designed to maximize code density, which is the number of instructions that can be stored in a given amount of memory. A dense instruction set can reduce memory requirements and improve performance by reducing the number of memory accesses required to fetch instructions.

**Compatibility:** The instruction set should be compatible with existing software and hardware. The architect should ensure that the instruction set is backward compatible with older versions of the same architecture and compatible with other architectures.

**Ease of use:** The instruction set should be easy to use and easy to learn. The architect should ensure that the instruction set is well documented and that tools and resources are available to support software development.

**Energy efficiency:** The instruction set should be designed to minimize power consumption. This can be achieved by reducing the number of instructions required to perform a particular operation, minimizing the number of memory accesses, and optimizing the instruction set for the underlying hardware.

14. What are the major drawbacks of a CISC computer?

Ans;

There are several major drawbacks of a CISC (Complex Instruction Set Computer) architecture:

**Complexity:** CISC architectures are complex and difficult to design and implement, which makes them more expensive to produce. The complexity also makes it harder to optimize the architecture for specific tasks.

**Slower clock speeds:** CISC processors typically have slower clock speeds compared to RISC (Reduced Instruction Set Computer) processors, which limits their performance.

**Power consumption:** CISC processors consume more power compared to RISC processors due to their complexity, which limits their use in mobile devices.

**Lower code density:** CISC instructions are typically longer than RISC instructions, which reduces code density and increases memory requirements.

**Difficulty in pipelining**: CISC instructions have variable execution times, making it difficult to pipeline the processor and achieve high instruction throughput.

**Limited parallelism:** CISC processors typically have limited support for parallelism, which limits their performance in parallel processing applications.

15. What is the ‘KISS’ principle? Where is it used? How effective is it?

Ans;

The KISS principle, which stands for "Keep It Simple, Stupid," is a design principle that encourages simplicity in systems and products. The principle suggests that simple designs are easier to understand, use, and maintain, and they are less likely to have problems or errors.

In computer architecture, the KISS principle is often applied to the design of instruction sets, processors, and other hardware components. The principle encourages architects to design systems that are easy to use, easy to learn, and efficient.

The KISS principle is effective in computer architecture because it helps to reduce complexity and increase reliability. Simple designs are easier to understand and maintain, which reduces the risk of errors and problems. Simple designs are also easier to optimize for specific tasks and more efficient in terms of power consumption and resource usage.

16. Give two examples of each of CISC and RISC CPUs.

Ans;

Examples of CISC CPUs:

Intel x86 family (including the Pentium, Core, and Xeon processors)

Motorola 68k family (including the 68000 and 68040 processors)

Examples of RISC CPUs:

ARM processors (including the Cortex-A and Cortex-M series)

MIPS processors (including the R3000, R4000, and R5000 processors)

17. What is the difference between multitasking and multiprocessing? Explain

Ans;

Multitasking and multiprocessing are both techniques used in computer systems to improve performance and efficiency, but they differ in the way they achieve this goal.

Multitasking refers to the ability of a computer system to run multiple applications or processes simultaneously. In a multitasking system, the processor switches rapidly between different tasks, giving the appearance that multiple tasks are being executed at the same time. Multitasking is typically achieved using an operating system that manages the scheduling of tasks and allocates processor time to each task in a way that optimizes overall system performance.

Multiprocessing, on the other hand, refers to the use of multiple processors or CPU cores to execute multiple tasks simultaneously. In a multiprocessing system, each processor or core executes a different task simultaneously, allowing for true parallel processing. Multiprocessing can be achieved using hardware or software techniques, such as the use of multiple processors on a single motherboard, or the use of a distributed computing system that uses multiple computers connected over a network.

The main difference between multitasking and multiprocessing is that multitasking achieves concurrency by interleaving tasks on a single processor, while multiprocessing achieves true parallel processing by using multiple processors or cores to execute tasks simultaneously. Multiprocessing generally offers greater performance benefits than multitasking, but requires specialized hardware and software support, and may be more difficult to implement and manage.

18. How does main memory interleaving increase the performance of a computer? Explain

Ans;

Main memory interleaving is a technique used to increase the performance of a computer by improving memory access times. Interleaving involves dividing the main memory into several banks, each with its own address range, and then alternating memory accesses across the banks.

When a memory request is made, the address is divided into two parts: the bank number and the address within the bank. The bank number determines which bank the request will be sent to, and the address within the bank determines the location within the bank that the request will access. By dividing the memory into multiple banks and alternating accesses between them, interleaving increases the effective memory bandwidth, reduces the memory access time, and reduces contention for memory resources.

Interleaving can be implemented in different ways, such as by using an interleaved memory controller, or by using multiple memory modules connected to the CPU via separate channels. The specific implementation depends on the architecture of the computer system and the requirements of the applications being run.

19. What kinds of interrupts are the following?

i. Data transfer

ii. INT instruction

iii. Error in CPU hardware

iv. Overflow

v. Illegal opcode

vi. End of I/O

Ans;

**i. Data transfer** - This is a type of device-initiated interrupt that occurs when a data transfer operation between a peripheral device and the CPU is complete.

**ii. INT instruction** - This is a type of software-initiated interrupt that occurs when an INT (interrupt) instruction is executed by the CPU.

**iii. Error in CPU hardware** - This is a type of hardware-initiated interrupt that occurs when an error is detected in the CPU hardware, such as a parity error or a fault in the memory management unit.

**iv. Overflow** - This is a type of arithmetic interrupt that occurs when the result of an arithmetic operation exceeds the range of values that can be represented by the data type being used.

**v. Illegal opcode** - This is a type of software-initiated interrupt that occurs when the CPU encounters an opcode that is not a valid instruction.

**vi. End of I/O** - This is a type of device-initiated interrupt that occurs when a peripheral device has completed an I/O operation and is ready for the next operation.

20. What is DMA? State with a neat diagram.

Ans;

DMA stands for Direct Memory Access, which is a technique used in computer systems to allow hardware devices to access the system's main memory directly, without the need for intervention from the CPU.

When a device needs to transfer data to or from the main memory, it would typically require the CPU to perform the transfer on its behalf. However, this can be inefficient as the CPU is often busy with other tasks and would not be able to devote its full attention to the data transfer operation.

