

## Searched, Sorted & Merged By @official\_stranger

Q1 Solve any following questions,

- A) What in general terms, is the distinction between computer organization and computer Architecture?

ANSWER:-

| Computer Organization   | Computer Architecture  |
|---|--|
| Concerned with the low-level details of the hardware components and their interconnections  | Concerned with the high-level design of the instruction set, hardware components, and overall system organization  |
| Focuses on how the hardware components work together to execute instructions and manage data  | Focuses on the selection of appropriate hardware components and the design of the system to achieve high performance, reliability, and energy efficiency |
| Deals with the organization of memory system, processor organization, input/output mechanisms, and the overall system bus structure | Deals with the design and implementation of the instruction set, the data path, and the control unit   |
| Focuses on the low-level details of the computer system   | Focuses on the high-level details of the computer system   |
| Examples of topics include instruction set design, memory hierarchy, and pipelining   | Examples of topics include processor microarchitecture, computer networks, and parallel computing  |

In summary, computer organization deals with the low-level details of how the hardware components are organized and interconnected to perform basic functions, while computer architecture deals with the high-level design of the instruction set, hardware components, and overall system organization to meet performance and functionality requirements.

- B) Explain the computer: the top level structure with structural component with neat sketch Diagram

ANSWER:-

A computer is an electronic device that can perform various tasks by processing data and executing instructions. It consists of several hardware components that work together to provide the user with a functional computing system. The top-level structure of a computer can be broken down into the following components:

**Input devices:** Input devices are used to provide data and instructions to the computer. Examples of input devices include keyboard, mouse, scanner, microphone, etc.

**Output devices:** Output devices are used to display the results of the processing done by the computer. Examples of output devices include monitor, printer, speaker, etc.

**Central Processing Unit (CPU):** The CPU is the brain of the computer. It carries out the instructions provided by software programs and controls the flow of data within the computer.

**Memory:** Memory refers to the storage capacity of the computer. It includes both

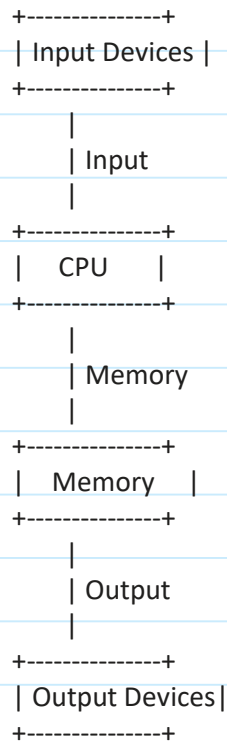
primary memory (RAM) and secondary memory (hard drive, solid-state drive, etc.) where data and programs are stored.

**Motherboard:** The motherboard is the main circuit board of the computer. It connects all the other components and provides power and communication pathways between them.

**Power supply unit (PSU):** The power supply unit provides the necessary power to run the computer.

**Expansion slots:** Expansion slots allow for additional hardware components, such as graphics cards or network cards, to be added to the computer.

Here is a neat sketch diagram that illustrates the top-level structure of a computer:



In summary, the top-level structure of a computer includes input and output devices, CPU, memory, motherboard, power supply unit, and expansion slots. Each of these components plays a vital role in providing a functional computing system.

## Q2 Attempt the following questions.

- A) Enlist and explain any two addressing modes. Given the following memory values and a one Address machine with an accumulator, what values do the following instructions load into the accumulator?
- Word 20 contains 40,
  - Word 30 contains 50.
  - Word 40 contains 60
  - Word 50 contains 70.
- (a) LOAD IMMEDIATE 20
- (b) LOAD DIRECT 20
- (c) LOAD INDIRECT 20
- (d) LOAD IMMEDIATE 30

ANSWER:-

Sure, I can explain two addressing modes and also provide the values loaded into the accumulator for the given instructions.

Addressing modes are methods used by a processor to access data from memory. There are several addressing modes, but I'll explain two common ones:

**Immediate Addressing Mode:** In this addressing mode, the operand is given as a part of the instruction itself. The instruction uses the operand directly, without accessing memory. For example, the instruction "LOAD IMMEDIATE 20" would load the value 20 into the accumulator.

**Direct Addressing Mode:** In this addressing mode, the operand is a memory address that directly identifies the location of the data in memory. The instruction uses the operand to access the data at the memory location. For example, the instruction "LOAD DIRECT 20" would load the value stored in memory location 20 (which is 40) into the accumulator.

Now, let's move on to the given instructions and their corresponding accumulator values:

(a) **LOAD IMMEDIATE 20:** This instruction would load the immediate value 20 into the accumulator. Therefore, the accumulator value would be 20.

(b) **LOAD DIRECT 20:** This instruction would load the value stored in memory location 20 into the accumulator. As mentioned earlier, the value stored in memory location 20 is 40. Therefore, the accumulator value would be 40.

(c) **LOAD INDIRECT 20:** This instruction would first access memory location 20 to get the address of the actual data (which is 40). It would then load the value stored in memory location 40 (indirectly specified by the value in memory location 20) into the accumulator. Therefore, the accumulator value would be 60.

(d) **LOAD IMMEDIATE 30:** This instruction would load the immediate value 30 into the accumulator. Therefore, the accumulator value would be 30

B)

I) Convert the following instruction into Accumulator based CPU, Register based CPU.

- Instruction:  $(A*B)-(R+Z)/T$

ANSWER:-

Assuming that A, B, R, Z, and T are all operands in memory, the instruction can be converted as follows:

**Accumulator based CPU:**

Load A into the accumulator

Multiply the accumulator with B

Store the result in memory

Load R from memory

Add Z to R

Divide the result of step 3 in the accumulator by T

Subtract the result of step 6 from the accumulator

Store the final result in memory

**Register based CPU:**

Load A into register 1

Load B into register 2

Multiply registers 1 and 2

Store the result in memory

Load R into register 3

Load Z into register 4

Add registers 3 and 4

Divide the result of step 4 in register 2 by T

Subtract the result of step 8 from the result of step 3 in register 3

Store the final result in memory.

II) Is RISC better than CISC? Illustrate your answer with example Of processor.

### ANSWER:-

The question of whether RISC (Reduced Instruction Set Computing) is better than CISC (Complex Instruction Set Computing) is a complex one, and it largely depends on the specific application and context in which the processor is being used.

Historically, CISC processors were popular in the early days of computing because they were able to perform complex operations with a single instruction, which made programming easier and more efficient. However, as technology has advanced, RISC processors have become more popular, particularly in high-performance computing environments.

RISC processors have a simpler instruction set, which allows them to execute instructions faster and more efficiently than CISC processors. This simplicity also makes it easier for compiler designers to optimize code for RISC processors, which can result in faster and more efficient programs. RISC processors are also better suited to pipelining, a technique that allows multiple instructions to be executed simultaneously, which can further improve performance.

An example of a RISC processor is the ARM processor, which is widely used in mobile devices and embedded systems. ARM processors have a simple instruction set and are designed to consume minimal power, making them well-suited for battery-powered devices.

An example of a CISC processor is the x86 processor, which is used in many desktop and laptop computers. x86 processors have a more complex instruction set, which can make programming easier but can also make the processor slower and less efficient. However, x86 processors have been optimized over the years and can still be very fast and efficient in many applications.

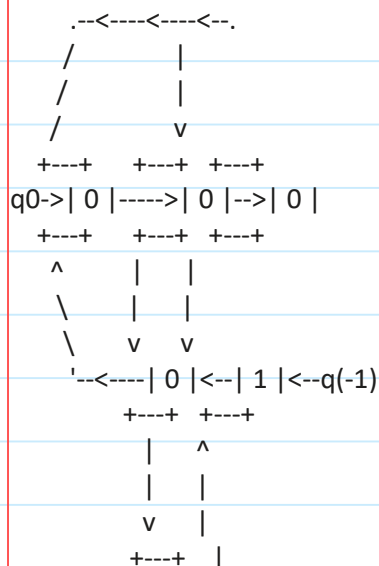
In summary, the choice between RISC and CISC processors depends on the specific application and context in which the processor will be used. Both types of processors have their advantages and disadvantages, and the optimal choice will depend on factors such as performance requirements, power consumption, and ease of programming.

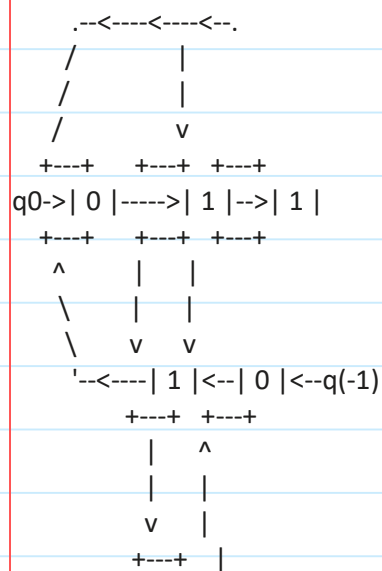
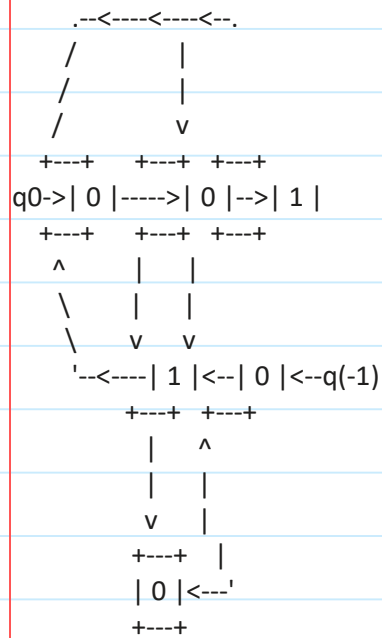
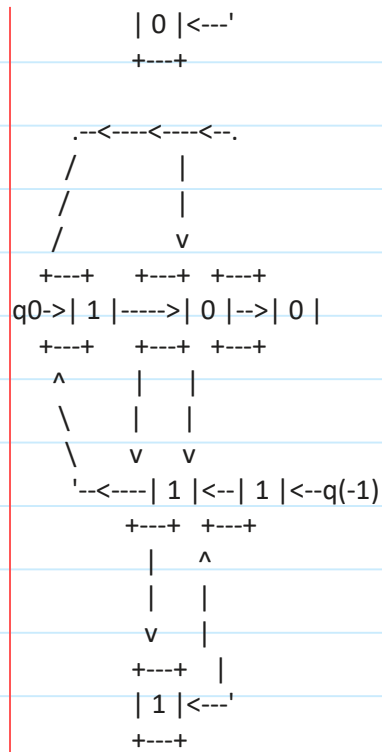
### Q3 Attempt the following questions.

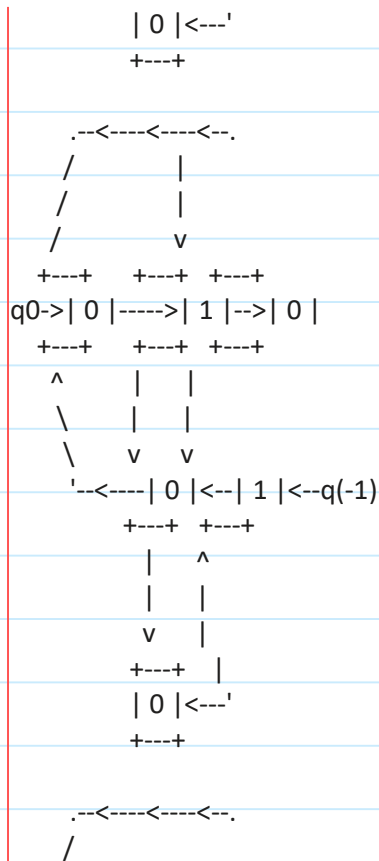
A) Given  $x = 1011$  and  $y = 1001$  in two's complement notation (i.e.,  $x = -5$ ,  $y = -7$ ), draw and compute the product  $p = x * y$  with Booth's algorithm flowchart.

### ANSWER:-

the Booth's algorithm flowchart for computing the product  $p = x * y = (-5) * (-7) = 35$  using 8-bit two's complement notation:







- B) Show how the following floating-point additions are performed (where significands are truncated to 4 decimal digits). Show the results in normalized form.
- $5.566 \times 10^2 + 7.777 \times 10^3$
  - $3.344 \times 10^1 + 8.877 \times 10^{-2}$
  - $6.21 \times 10^5 / 8.877 \times 10^1$

ANSWER:-

To add  $5.566 \times 10^2$  and  $7.777 \times 10^3$ , we need to adjust the exponents so that they are the same. We can do this by moving the decimal point in  $5.566 \times 10^2$  to the right by three places:

$$5.566 \times 10^2 + 7.777 \times 10^3 = 0.5566 \times 10^4 + 7.777 \times 10^3$$

Now, we can add the two numbers:

$$0.5566 \times 10^4 + 7.777 \times 10^3 = 6.3346 \times 10^3$$

To normalize the result, we need to adjust the exponent and move the decimal point accordingly. Since  $6.3346 \times 10^3$  is already in scientific notation, we just need to adjust the exponent to get the normalized form:

$$6.3346 \times 10^3 = 6.3346 \times 10^3$$

b) To add  $3.344 \times 10^1$  and  $8.877 \times 10^{-2}$ , we need to adjust the exponents so that they are the same. We can do this by moving the decimal point in  $8.877 \times 10^{-2}$  to the right by one place:

$$3.344 \times 10^1 + 8.877 \times 10^{-2} = 3.344 \times 10^1 + 0.8877 \times 10^0$$

Now, we can add the two numbers:

$$3.344 \times 10^1 + 0.8877 \times 10^0 = 3.4327 \times 10^1$$

To normalize the result, we need to adjust the exponent and move the decimal point accordingly:

$$3.4327 \times 10^1 = 34.327$$

c) To perform the division  $6.21 \times 10^5 / 8.877 \times 10^1$ , we can divide the significands and subtract the exponents:

$$6.21 \times 10^5 / 8.877 \times 10^1 = (6.21 / 8.877) \times 10^{(5-1)} = 0.6985 \times 10^4$$

To normalize the result, we need to adjust the exponent and move the decimal point accordingly:

$$0.6985 \times 10^4 = 6.985 \times 10^3$$

Q4 Attempt the following questions.

- A) What are the differences among direct mapping, associative mapping, and set-associative mapping? A set-associative cache consists of 64 lines, or slots, divided into four-line sets. Main Memory contains 4K blocks of 128 words each. Show the format of main memory addresses.

ANSWER:-

**Direct Mapping:** In direct mapping, each block of data in memory can be mapped to only one specific location in the cache. This is done by dividing the cache into equal-sized blocks and then associating each block of memory with a specific cache block.

**Associative Mapping:** In associative mapping, each block of data in memory can be mapped to any location in the cache. In other words, a block of memory can be stored in any available cache block. Associative mapping eliminates the problem of conflicts, but it is more complex and requires more hardware than direct mapping.

**Set-Associative Mapping:** Set-associative mapping is a compromise between direct and associative mapping. In set-associative mapping, the cache is divided into a number of sets, each of which contains a small number of cache blocks. Each block of memory is mapped to one set.

In summary, direct mapping is simple and easy to implement but may result in conflicts, associative mapping eliminates conflicts but is more complex, and set-associative mapping is a compromise between the two.

Given that the cache has 64 lines divided into 4-line sets, there are 16 sets in the cache. Therefore, the number of bits required to represent the set index is  $\log_2(16) = 4$  bits.

The block size is 128 words, which requires 7 bits to represent (since  $2^7 = 128$ ). Thus, the remaining bits in the main memory address (after the tag and set index bits have been determined) are used to represent the word offset.

Putting it all together, the format of a main memory address would be:

| Tag (T)           | Set Index (S) | Word Offset (W) |
|-------------------|---------------|-----------------|
| (32 - 4 - 7) bits | 4 bits        | 7 bits          |

where the total number of bits in a main memory address is 32 bits

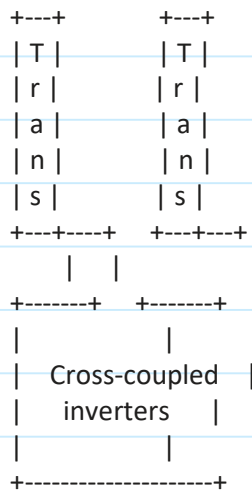
- B) Elaborate the concept of SRAM and DRAM memory with typical memory cell structure.

ANSWER:-

SRAM (Static Random Access Memory) and DRAM (Dynamic Random Access Memory) are two types of volatile computer memory that are commonly used in modern computers and other electronic devices.

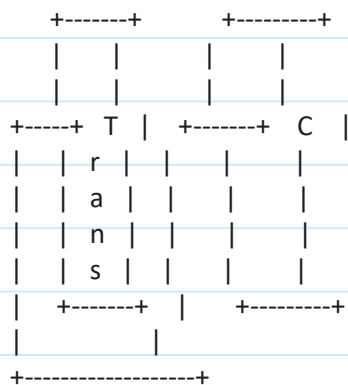
SRAM is a type of memory that stores data using a flip-flop circuit. This circuit consists of two cross-coupled inverters that maintain a stable output state, which can represent either a "0" or a "1". The memory cell structure of SRAM typically consists of six transistors - four transistors form the two inverters, while two additional transistors are used as access transistors to control the flow of data in and out of the cell.

Here is a simplified diagram of an SRAM memory cell:



DRAM, on the other hand, stores data using a capacitor to store charge. The capacitor is charged or discharged to represent a "0" or a "1", respectively. The memory cell structure of DRAM typically consists of a single transistor and a capacitor. The transistor is used as an access device to control the flow of data to and from the capacitor.

Here is a simplified diagram of a DRAM memory cell:



In DRAM, the charge stored in the capacitor leaks over time and needs to be periodically refreshed. This makes DRAM slower and less power-efficient compared to SRAM, but it is also cheaper and allows for higher densities.

Both SRAM and DRAM have their own advantages and disadvantages, and they are typically used in different parts of a computer system depending on the specific requirements of the application.

Q5. Attempt the following questions.

- A) What is the overall function of a processor's control unit? A stack is implemented show the sequence of micro-operations for
- popping
  - pushing the stack
- PUSH 10  
PUSH 70  
PUSH 8  
ADD  
PUSH 20  
SUB  
MUL

ANSWER:-



The control unit is a critical component of a processor (also called CPU, central processing unit) that manages and coordinates the execution of instructions in a computer program. Its overall function is to fetch instructions from memory, decode them, and execute them in a sequential manner.

The control unit performs the following specific functions:

**Instruction Fetch:** The control unit fetches instructions from main memory (or cache) and stores them in its instruction register (IR).

**Instruction Decode:** Once an instruction has been fetched, the control unit decodes the instruction to determine what operation needs to be performed.

**Operand Fetch:** After the instruction has been decoded, the control unit fetches any operands required for the instruction from memory or registers.

**Instruction Execution:** Once the operands have been fetched, the control unit executes the instruction by performing the appropriate operation on the data.

**Result Write-Back:** Finally, the control unit writes the result of the operation back to memory or a register.

Assuming a simple stack implementation where the stack pointer (SP) is pointing to the top of the stack, the following are the sequence of micro-operations for popping and pushing the stack, as well as the sequence of micro-operations for executing the given set of instructions:

**Popping the stack:**

- a.  $SP \leftarrow SP - 1$  // decrement the stack pointer
- b.  $Data \leftarrow Stack[SP]$  // load the topmost data from the stack
- c. // Use the loaded data as needed

**Pushing the stack:**

- a.  $Stack[SP] \leftarrow Data$  // store the data at the top of the stack
- b.  $SP \leftarrow SP + 1$  // increment the stack pointer

**Sequence of micro-operations for executing the instructions:**

**PUSH 10:**

- a.  $Data \leftarrow 10$  // load the data to be pushed onto the stack
- b.  $Stack[SP] \leftarrow Data$  // store the data at the top of the stack
- c.  $SP \leftarrow SP + 1$  // increment the stack pointer

**PUSH 70:**

- a.  $Data \leftarrow 70$
- b.  $Stack[SP] \leftarrow Data$
- c.  $SP \leftarrow SP + 1$

**PUSH 8:**

- a.  $Data \leftarrow 8$
- b.  $Stack[SP] \leftarrow Data$
- c.  $SP \leftarrow SP + 1$

**ADD:**

- a.  $SP \leftarrow SP - 1$  // decrement the stack pointer
- b.  $Data1 \leftarrow Stack[SP]$  // load the first operand from the stack
- c.  $SP \leftarrow SP - 1$  // decrement the stack pointer again
- d.  $Data2 \leftarrow Stack[SP]$  // load the second operand from the stack
- e.  $Result \leftarrow Data1 + Data2$  // perform the addition operation
- f.  $Stack[SP] \leftarrow Result$  // store the result at the top of the stack
- g.  $SP \leftarrow SP + 1$  // increment the stack pointer

PUSH 20:

- a. Data  $\leftarrow$  20
- b. Stack[SP]  $\leftarrow$  Data
- c. SP  $\leftarrow$  SP + 1

SUB:

- a. SP  $\leftarrow$  SP - 1
- b. Data1  $\leftarrow$  Stack[SP]
- c. SP  $\leftarrow$  SP - 1
- d. Data2  $\leftarrow$  Stack[SP]
- e. Result  $\leftarrow$  Data2 - Data1 // note the order of operands as stack is LIFO
- f. Stack[SP]  $\leftarrow$  Result
- g. SP  $\leftarrow$  SP + 1

MUL:

- a. SP  $\leftarrow$  SP - 1
- b. Data1  $\leftarrow$  Stack[SP]
- c. SP  $\leftarrow$  SP - 1
- d. Data2  $\leftarrow$  Stack[SP]
- e. Result  $\leftarrow$  Data1 \* Data2
- f. Stack[SP]  $\leftarrow$  Result
- g. SP  $\leftarrow$  SP + 1

- B) What is the difference between a hardwired Implementation and a microprogrammed implementation Of a control unit?

ANSWER:-

| Hardwired Implementation  | Microprogrammed Implementation  |
|---|---|
| Uses physical circuits to control the operation of a processor  | Uses software (microcode) to control the operation of a processor   |
| Control signals are generated by the physical circuits  | Control signals are generated by executing instructions in the microcode  |
| Faster and more efficient than microprogrammed implementation   | Slower and less efficient than hardwired implementation   |
| Difficult to modify or update once the circuits are designed and implemented  | Easier to modify or update because changes can be made to the microcode   |
| Requires a highly skilled designer to create the circuits   | Requires a highly skilled programmer to create the microcode  |
| The design process is complex and time-consuming, but once the circuits are designed, they can be produced at scale | The design process is less complex and time-consuming, but updating the microcode requires reprogramming                          |
| Hardwired implementations are commonly used in simple processors or embedded systems                                | Microprogrammed implementations are commonly used in complex processors or systems that require frequent updates or modifications |
| Examples include the first computers, calculators, and early microprocessors  | Examples include modern CPUs, GPUs, and digital signal processors   |

Q6 Attempt any two questions.

- A) In virtually all systems that include DMA modules, DMA access to main memory is give Priority than CPU access to main memory. Why?

**ANSWER:-**

In systems that include DMA (Direct Memory Access) modules, DMA access to main memory is given priority over CPU access to main memory for several reasons:

1. **Efficiency:** DMA transfers can be much faster than CPU transfers because they bypass the CPU and directly access memory. This is particularly true when transferring large amounts of data. Giving priority to DMA transfers can help increase overall system performance and efficiency.
2. **Reliability:** DMA transfers are often used for time-critical operations, such as real-time data acquisition or transmission. Giving priority to DMA transfers can help ensure that these operations are performed reliably and in a timely manner.
3. **Reduced CPU overhead:** By giving priority to DMA transfers, the CPU can be freed up to perform other tasks while the DMA transfer is taking place. This can help reduce CPU overhead and improve overall system performance.
4. **DMA Controller:** DMA transfers are typically controlled by a separate DMA controller, which is designed specifically for this purpose. This controller can manage data transfers more efficiently than the CPU, which has other tasks to perform as well.

Overall, giving priority to DMA transfers can help improve system performance, reliability, and efficiency.

- B) What is the meaning Of each Of the four states in the MESI protocol? Can you foresee any Problem with the write-once cache approach on bus-based multiprocessors? If so, suggest a Solution.

**ANSWER:-**

The MESI protocol is a widely used cache coherence protocol in bus-based multiprocessor systems. The protocol defines four states for each cache line in a cache:

1. **Modified (M):** The cache line has been modified and is not coherent with other caches or main memory.
2. **Exclusive (E):** The cache line is valid and is not shared with any other cache. It can be modified without notifying other caches or main memory.
3. **Shared (S):** The cache line is valid and is shared with one or more other caches. It cannot be modified without notifying other caches.
4. **Invalid (I):** The cache line is not valid and does not contain any useful data.

The MESI protocol is designed to maintain cache coherence, ensuring that all caches have a consistent view of memory. When a processor wants to modify a cache line, it must first request ownership of the line by sending a request to other caches on the bus. If the line is in the shared state, the other caches must invalidate their copies before the requesting processor can modify the line.

Regarding the second question, a write-once cache approach in bus-based multiprocessor systems can lead to cache coherence issues. If a cache line is modified by one processor, other processors may still have invalid copies of the line, leading to inconsistent views of memory. One solution to this problem is to use a cache coherence protocol, such as the MESI protocol, to ensure that all caches have a consistent view of memory. Another solution is to use a cache line invalidation scheme, where all caches are notified when a cache line is modified, and their copies of the line are invalidated. This can help ensure that all caches have a consistent view of memory, even with a write-once cache approach.

- C) How does instruction pipelining enhance system performance? Elaborate your answer using RISC instruction stages.

ANSWER:-

Instruction pipelining is a technique used in computer architecture to improve system performance by overlapping the execution of multiple instructions. It breaks down the execution of an instruction into a sequence of stages and processes multiple instructions at the same time, with each instruction at a different stage in the pipeline. By doing so, the processor can begin executing the next instruction before completing the previous one, reducing the time it takes to execute a single instruction.

The RISC (Reduced Instruction Set Computing) architecture is particularly well-suited for instruction pipelining because it uses a simpler instruction set and a fixed instruction format. RISC processors typically have five stages in their pipeline:

Instruction Fetch (IF): The processor fetches the instruction from memory and places it in the instruction register.

Instruction Decode (ID): The processor decodes the instruction, determining which operation to perform and which operands to use.

Execute (EX): The processor executes the operation specified by the instruction, performing arithmetic, logic, or memory operations.

Memory Access (MEM): If the instruction involves memory access, the processor accesses memory to retrieve or store data.

Write Back (WB): The processor writes the result of the operation back to a register.

Each instruction passes through these stages, with each stage processing a different instruction at any given time. As soon as one instruction finishes one stage, the next instruction can enter that stage. This overlap of execution allows multiple instructions to be processed simultaneously and results in a higher instruction throughput and improved system performance.

However, there are several challenges in implementing pipelining, such as branch instructions and data hazards that can cause pipeline stalls and reduce performance. To address these challenges, modern processors use techniques like branch prediction, forwarding, and out-of-order execution.

In summary, instruction pipelining enhances system performance by overlapping the execution of multiple instructions, reducing the time it takes to execute a single instruction. The RISC architecture is particularly well-suited for pipelining due to its simple instruction set and fixed instruction format. By dividing the execution of an instruction into five stages, RISC processors can process multiple instructions simultaneously, resulting in a higher instruction throughput and improved system performance.