COA

Unit - 4

Important 12-marks Questions :-

1. Explain Flynn's classification and four architecture in detail.
2. Elaborate on software parallelism & its types.
3. Write a note on Hardware multithreading
4. Explain about various type of processors.
5. Explain about memory management of multiprocessor.
6. Explain ~~Coherence~~ Cache Coherence in detail.
7. Explain about various protocols of Cache coherence in multiprocessor system.

↑

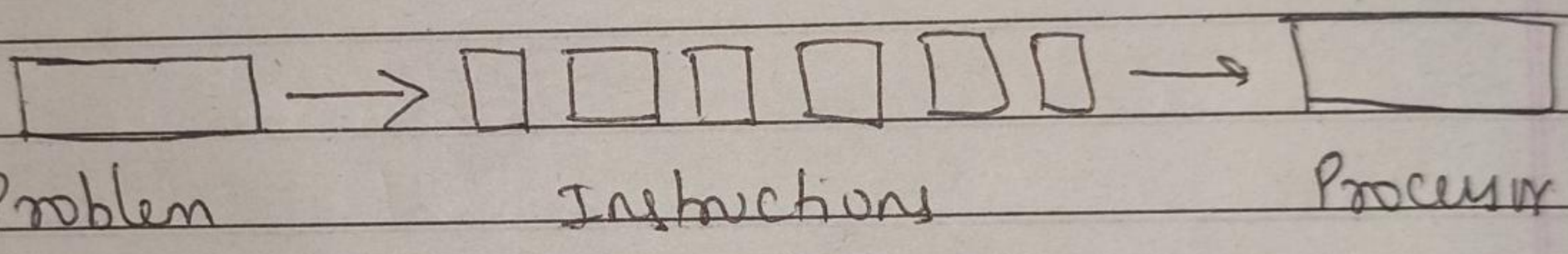
All possible Question in Exam

COA

Unit - 4

Parallelism

Executing two or more operations at the same time is known as parallelism.



Parallel processing :-

Parallel processing improves the performance by executing two or more instructions simultaneously.

A parallel computer is a set of processor that are able to work cooperatively to solve a computational problem. On a single processor there may be two or more ALU which work concurrently to increase the performance. Also system may have two or more processor operating concurrently.

Goals of parallelism :-

(a) To increase the computational speed (i.e reduce the amount of time that we need to wait for problem to be solved.)

(b) To increase the throughput (i.e the number of processes completed on unit time).

(c) To improve the performance of computer for a given clock speed.

(d) To solve bigger problem that might not fit in limited memory of a single CPU.

(e) To take advantage of non-local resource when local resource are finite.

Types of parallelism :-

(a) Hardware Parallelism
(b) Software Parallelism.

(a) Hardware Parallelism :- It is based on the processor multiplicity and machine architecture. One way to characterize the parallelism in a processor is by the number of instruction issued per machine cycle. If a machine sends k instruction per machine cycle, it is called k-issue processor.

(b) Software Parallelism :- It is determined by dependency among the sub tasks. The flow control graph is used to represent the degree of parallelism in program. It shows the possible ways in which the sub-program can be executed in parallel. Parallelism in a program varies during execution period.

Example :-

$A = L1 * L2 + L3 * L4$

$B = L1 * L2 - L3 * L4$

↳ This contain 8 instructions.

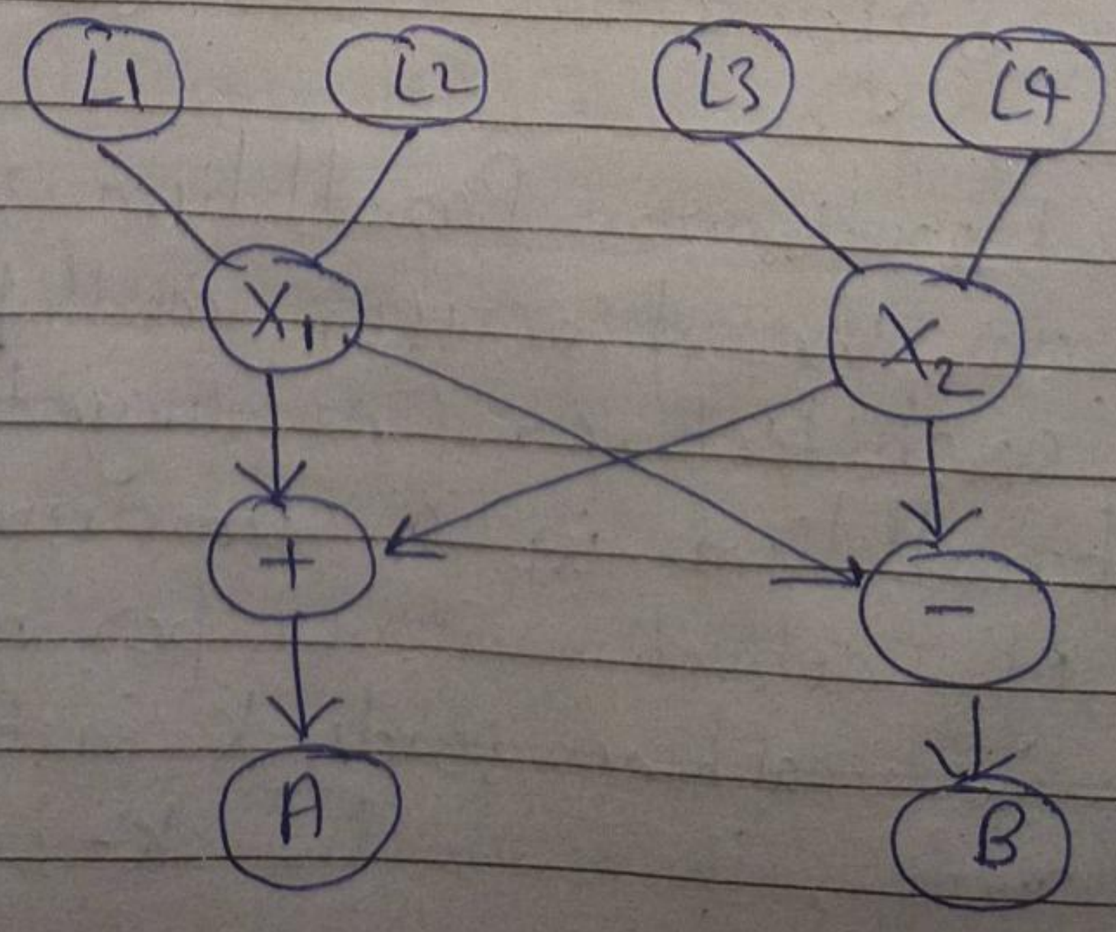
→ 4 Load instruction

→ 2 multiply instruction

→ One Add

→ One Subtract

This parallelism varies from 4 to 2 in three cycles.



Average s/w $= \frac{8}{3}$

Types of Software Parallelism :-

(a) Instruction Level Parallelism
(b) Task-level Parallelism
(c) Data Parallelism
(d) Transaction Level parallelism.

(a) Instruction Level Parallelism :-

It is a measure of how many operation (instructions) can be performed in parallel at the same time in a computer program.

Parallel instructions are set of instructions that are not ~~parallel~~ dependent to each other.

ILP allows compiler to overlap the execution of multiple instructions or change the order of execution of instructions.

Example :-

1. $x = a + b$
2. $y = c - d$
3. $z = x * y$

Here, 1 and 2 are not dependent on each other. So they can be computed simultaneously. If we assume each operation can be completed in unit time, these ~~th~~ operation can be done in 2 unit.

ILP factor is $3/2 = 1.5$ which is greater than without ILP.

(b) Data-level Parallelism :-

Data parallelism is parallelization across multiple processor in parallel computing environments. It focus on distributing the data across different nodes which operate on the data in parallel. Instructions from single steam operate concurrently on several data.

Example :- Let us assume, we want to sum all element of given array of size n and time for a single addition is Ta time unit. Then for sequential execution total time will be n*Ta unit. If we execute this job as data parallel job on 4 processor, then total time would reduce to (n/4)* Ta + merging time.

(c) Instruction level parallelism :-

It executes the multiple instructions on same instruction stream simultaneously. These are generated and manged by either hardware or by compiler.

(d) Thread-level or Task-level Parallelism :-
Multiple program/Task threads are created for same application and executed simultaneously.
These task threads are created and managed by compiler and hardware, however the program threade are created by programmer and managed by compiler and hardware during run-time.

Flynn's Classification :-

It was proposed by Micheal J. Flynn in 1966.
In this classification computer are classified by whether it processes a single instruction at a time or multiple instruction simultaneously. and whether it operate on one or multiple data sets.

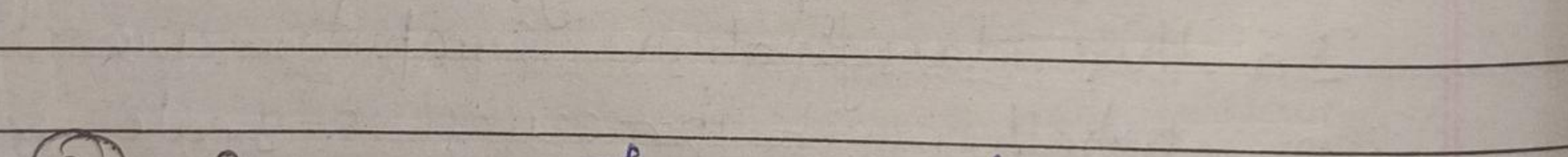
This taxonomy distinguishes multiprocessor computer architectures according two independent dimension.

(a) Instruction steam → It is sequence of instruction executed by machine
(b) Data Steam → It is sequence of data including input, partial or temporary results used by instruction steam.

Each of these dimension can have only one of two possible states :- Single or Multiple.

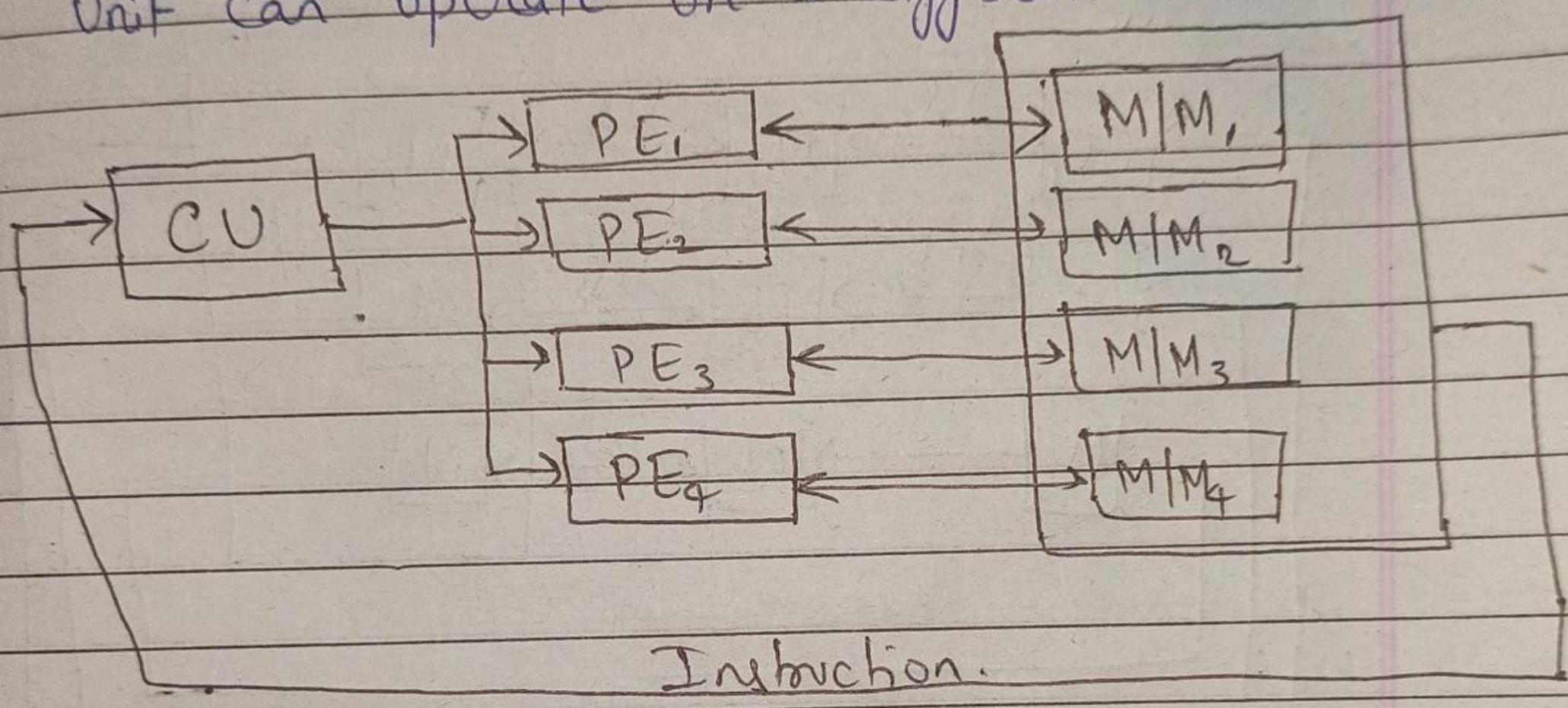
Instruction Stream \ Data Stream	Single	Multiple
Single	Single Instruction Single Data (SISD)	Single Instruction Multiple Data (SIMD)
Multiple	Multiple Instruction Single Data (MISD)	Multiple Instruction Multiple Data (MIMD)

(a) SISD :- They are also called scalar processor i.e. one instruction at a time and each instruction have only one set of operands.



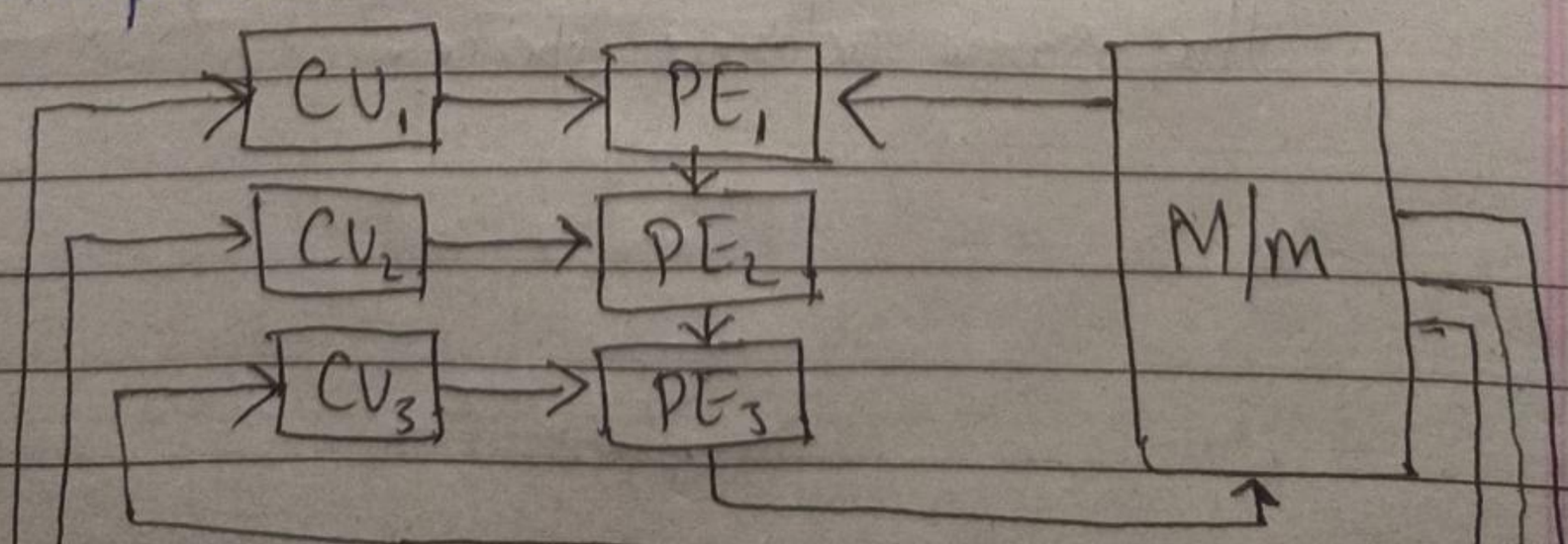
SISD have one control unit, one processor unit and single memory unit. In this instructions are excated sequentially.

(b) SIMD :- A type of parallel computer. Single Instruction is executed by different processing unit on different set of data. In this, all processing units execute the same instruction issued by control unit at any given clock cycle, also each processing unit can operate on a different data element.

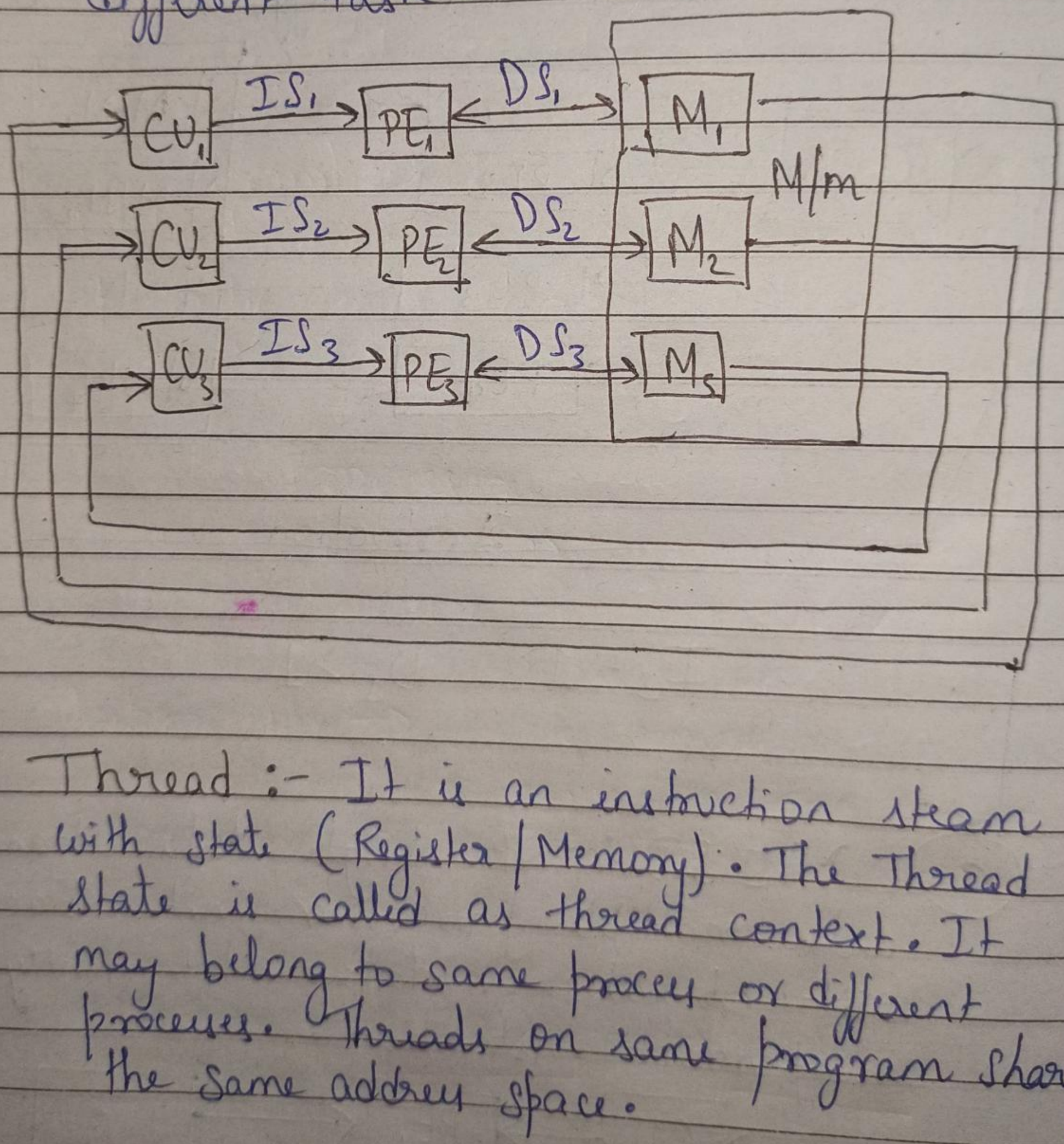


$I_s = 1 \quad D_s > 1$

(c) MISD :- A single data steam is fed into multiple processor unit. Each processing unit operates on data independently via independent instruction.



(d) MIMD :- In this, every processor may be executing a different instruction steam and different data steam too. Execution can be synchronous or asynchronous. Different processor are there with each of them processing different task.



Thread :- It is an instruction steam with state (Register / Memory). The Thread state is called as thread context. It may belong to same process or different processes. Threads on same program share the same address space.

Hardware Threading :- It allows multiple threads to share the functional units of a single processor in an overlapping fashion to try to ~~all~~ utilize the hardware resources efficiently.

To permit this sharing the processor must duplicate the independent state of each thread.

Types of Multithreading :-

(a) Fine Grained
(b) Course Grained

(a) Fine Grained :- Fine-grained multithreading switches between threads on each instruction, resulting in interleaved execution of multiple threads. This interleaving is often done in around robin-fashion. To make fine-grained multithreading practical, processor must be able to switch threads on every clock cycle.

One ~~adv~~ advantage of this is that it can hide throughput losses that arise from both short and long stalls.

One disadvantage is that it slows down execution of individual threads, since a thread ready to execute without stalls will be delayed by instructions from or threads.

(b) Coarse grain Multithreading :-

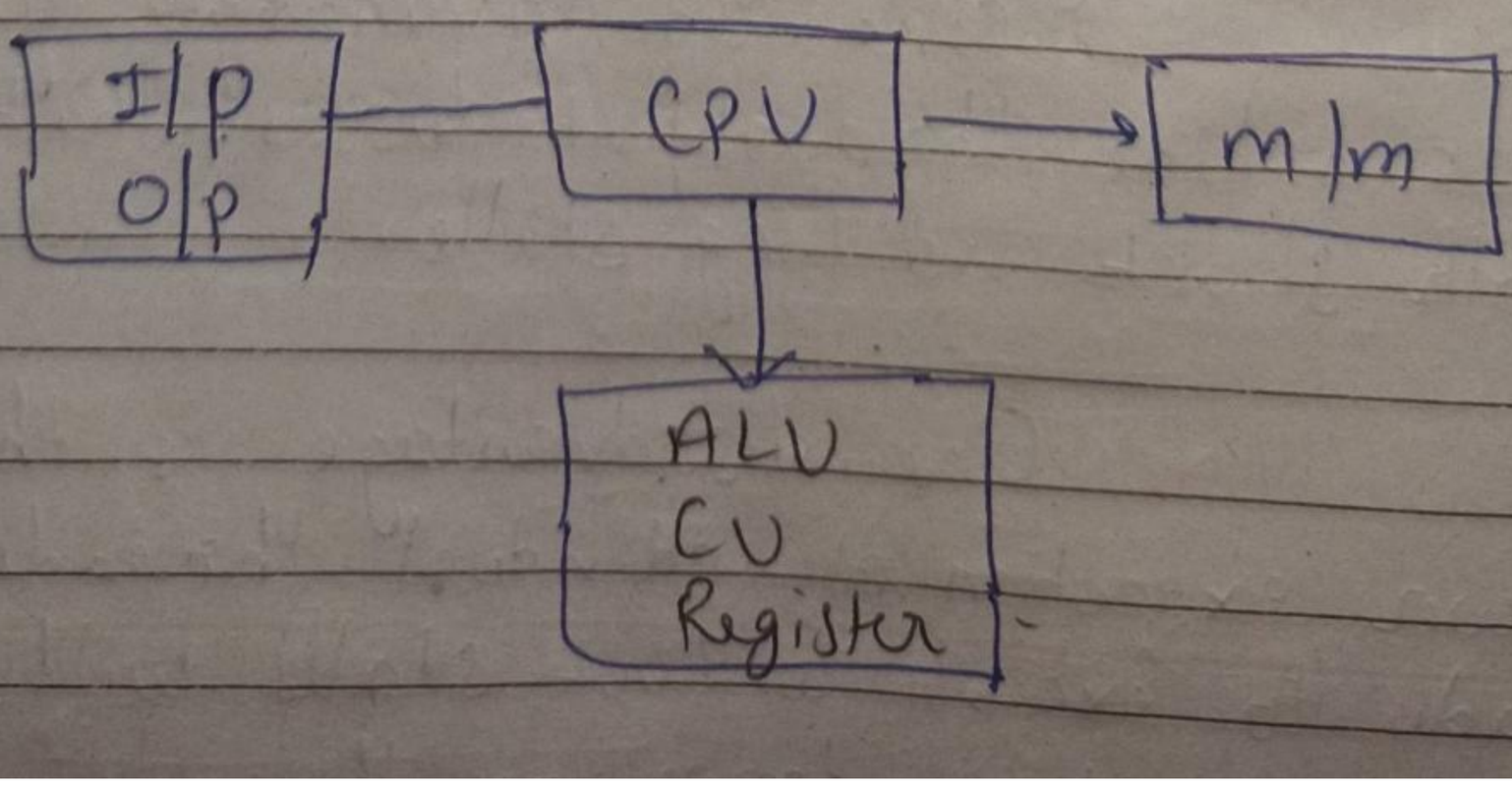
Coarse-grained multithreading switches threads only on costly stalls such as last-level cache-misses. This change relieves need to have thread switching to extremely fast and is much likely to slow down execution of individual thread.

Drawback :- It is limited in its ability to overcome throughput losses, especially from shorter stalls.

Processor

Uniprocessor Multiprocessor

Uniprocessor → Single processor
⇒ Only depends on one single processor to perform operations.



Multiprocessor :- A system with two or more CPU's that allows simultaneous processing of programs.

Difference b/w MultiCore & Multiprocessor :-

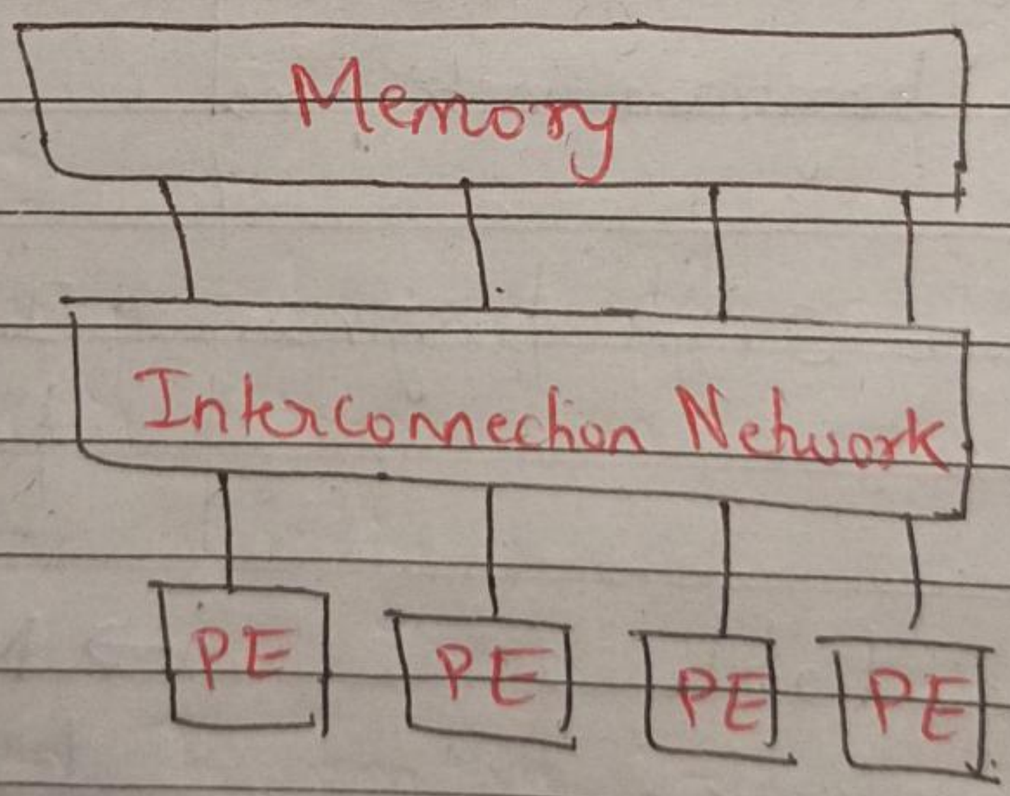
MultiCore	Multiprocessor
→ A single CPU or processor with two or more independent processing units called cores that are capable of reading and executing program instructions.	→ A system ~~that~~ a with two or more CPU's that allows simultaneous processing of programs.
→ It execute single program faster.	→ It executes multiple program faster.
→ Not as reliable as multiprocessor.	→ More reliable since failure in one CPU will not affect other.
→ It has less traffic	→ It has more traffic
→ It does not need to be configured	→ It need little complex configuration
→ It is cheaper	→ It is Expensive

Memory in Multiprocessor

Shared Distributed

(a) Shared Memory Multiprocessor :-

In this, multiple processor can operate independently but share the same common memory. Change in memory location effected by one processor are visible to all other processor.



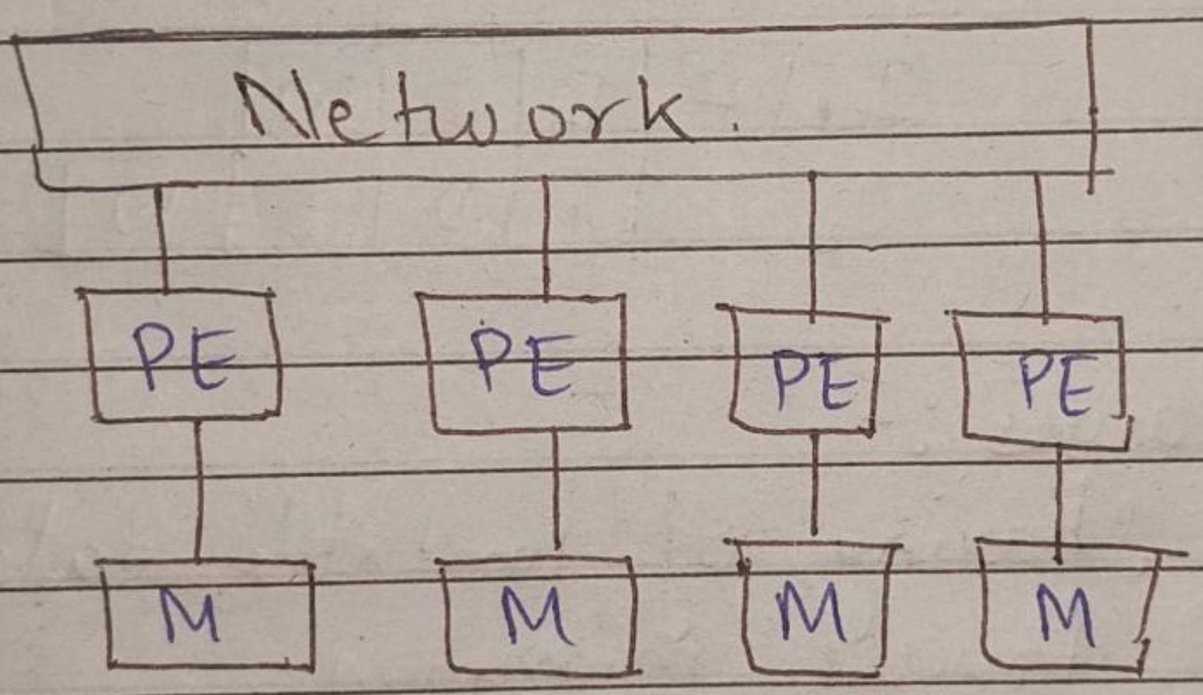
It is also known as UMA. (Uniform Memory Accen).

(b) Distributed Memory Multiprocessor :-

In this, each processor has its own memory and it require a communication network to connect inter-processor memory.

Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processor.

Because each processor has its own local memory, its operate independently.



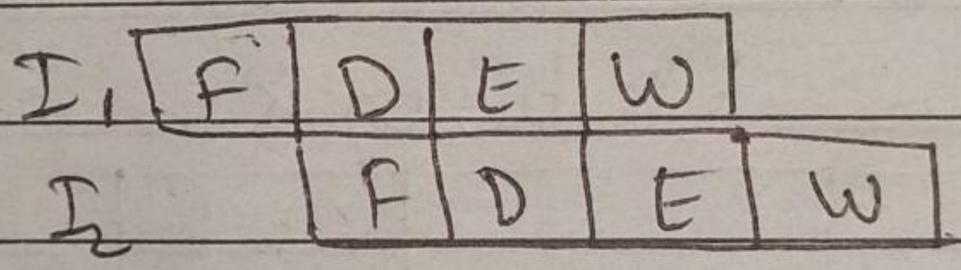
(i) NUMA :- It is often made by physically linking two or more SMPs. Memory acces across link is slower.

(ii) COMA :- It is special case of NUMA in which distributed main memories are converted to caches. All caches form a global address space and there is no memory hierachy at each processor node.

Cache Coherence :-

Cache Coherence is the discipline that ensure that ensures that changes in the values of shared operands are propagated throughout the system in a timely fashion.

* When different multiprocessor work on same data element, Cache Coherence happens.



Causes :-

(a) Sharing of writable data
(b) Process migration
(c) I/O activity

Solution Technique :-

- Write update
- Write through
- Write back
- Write invalidate.

Solution :-

(a) writ updat - writ through
(b) writ updat - writ back
(c) Writ invalidat - writ through
(d) Writ invalidat - writ back.

Writ updat → Propagation of data element to all other ~~of~~ caches.
Writ through → Simultaneously updat
Writ back → End result / final result / value at end is transfered to cache.
Writ invalidat → invalid value is propagated to other ~~problem~~ processor.

Cache coherence

- Cache coherence is the discipline that ensures that changes in the values of shared operands are propagated throughout the system in a timely fashion.

There are three distinct level of cache coherence :-

- Every write operation appears to occur instantaneously.
- All processors see exactly the same sequence of changes of values for each separate operand.
- Different processors may see an operation and assume different sequences of values; this is known as non-coherent behavior.

There are various Cache Coherence Protocols in multiprocessor system. These are :-

- MSI protocol (Modified, Shared, Invalid)
- MOSI protocol (Modified, Owned, Shared, Invalid)
- MESI protocol (Modified, Exclusive, Shared, Invalid)
- MOESI protocol (Modified, Owned, Exclusive, Shared, Invalid)

Modified –

It means that the the value in the cache is dirty, that is the value in current cache is different from the main memory.

Exclusive –

It means that the value present in the cache is same as that present in the main memory, that is the value is clean.

Shared –

It means that the cache value holds the most recent data copy and that is what shared among all the cache and main memory as well.

Owned –

It means that the current cache holds the block and is now the owner of that block, that is having all rights on that particular blocks.

Invalid –

This states that the current cache block itself is invalid and is required to be fetched from other cache or main memory.

Cache Coherence Problem

- In multiprocessor system where many processes needs a copy of same memory block, the maintenance of consistency among these copies raises a raises a problem referred to as Cache Coherence Problem.

This occurs mainly due to these causes:-

- Sharing of writable data.
- Process migration.
- Inconsistency due to I/O.

Cache Coherence Protocols:

1. MSI Protocol:

- This is a basic cache coherence protocol used in multiprocessor system. The letters of protocol name identify possible states in which a cache can be. So, for MSI each block can have one of the following possible states:
  Modified –
- The block has been modified n cache, i.e., the data in the cache is inconsistent with the backing store (memory). So, a cache with a block in "M" state has responsibility to write the block to backing store when it is evicted.
  Shared –
- This block is not modified and is present in atleast one cache. The cache can evict the data without writing it to backing store.
  Invalid –
- This block is invalid and must be fetched from memory or from another cache if it is to be stored in this cache.

2. MOSI Protocol:

- This protocol is an extension of MSI protocol. It adds the following state in MSI protocol:

  Owned –
- It indicates that the present processor owns this block and will service requests from other processors for the block.

3. MESI Protocol –

- It is the most widely used cache coherence protocol. Every cache line is marked with one the following states:

  Modified –
- This indicates that the cache line is present in current cache only and is dirty i.e its value is different from the main memory. The cache is required to write the data back to main memory in future, before permitting any other read of invalid main memory state.

  Exclusive –
- This indicates that the cache line is present in current cache only and is clean i.e its value matches the main memory value.

  Shared –
- It indicates that this cache line may be stored in other caches of the machine.

  Invalid –
- It indicates that this cache line is invalid.

4. MOESI Protocol:

- This is a full cache coherence protocol that encompasses all of the possible states commonly used in other protocols. Each cache line is in one of the following states:

  Modified –
- A cache line in this state holds the most recent, correct copy of the data while the copy in the main memory is incorrect and no other processor holds a copy.

  Owned –
- A cache line in this state holds the most recent, correct copy of the data. It is similar to shared state in that other processors can hold a copy of most recent, correct data and unlike shared state however, copy in main memory can be incorrect. Only one processor can hold the data in owned state while all other processors must hold the data in shared state.

  Exclusive –
- A cache line in this state holds the most recent, correct copy of the data. The main memory copy is also most recent, correct copy of data while no other holds a copy of data.

  Shared –

- A cache line in this state holds the most recent, correct copy of the data. Other processors in system may hold copies of data in shared state as well. The main memory copy is also the most recent, correct copy of the data, if no other processor holds it in owned state.

  Invalid –
- A cache line in this state does not hold a valid copy of data. Valid copies of data can be either in main memory or another processor cache.