

UNIT 5

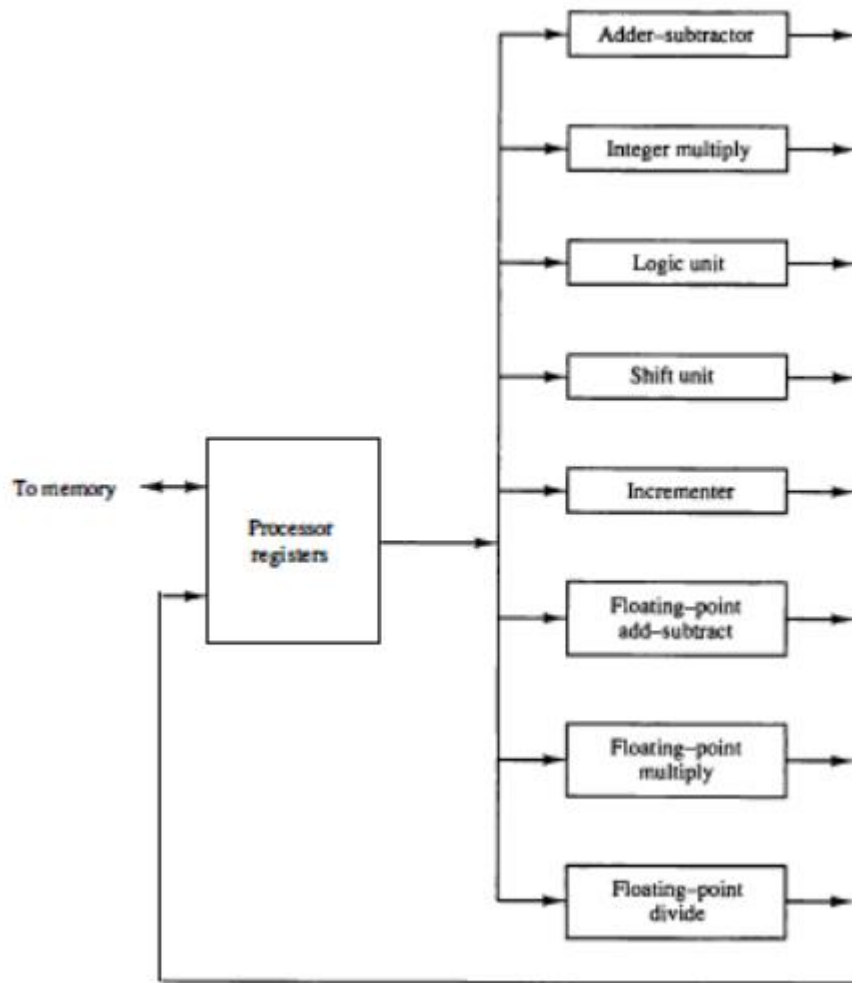
Parallel Processors

Parallel processing is a term used to denote a large class of techniques that are used to provide simultaneous data-processing tasks for the purpose of increasing the computational speed of a computer system. Instead of processing each instruction sequentially as in a conventional computer, a parallel processing system is able to perform concurrent data processing to achieve faster execution time.

The purpose of parallel processing is to speed up the computer processing capability and increase its throughput, that is, the amount of processing that can be accomplished during a given interval of time. The amount of hardware increases with parallel processing and with it, the cost of the system increases. However, technological developments have reduced hardware costs to the point where parallel processing techniques are economically feasible.

Parallel processing can be viewed from various levels of complexity. At the lowest level, we distinguish between parallel and serial operations by the type of registers used. Shift registers operate in serial fashion one bit at a time, while registers with parallel load operate with all the bits of the word simultaneously. Parallel processing at a higher level of complexity can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously. Parallel processing is established by distributing the data among the multiple functional units. For example, the arithmetic, logic, and shift operations can be separated into three units and the operands diverted to each unit under the supervision of a control unit.

Figure shows one possible way of separating the execution unit into eight functional units operating in parallel. The operands in the registers are applied to one of the units depending on the operation specified by the instruction associated with the operands. The operation performed in each functional unit is indicated in each block of the diagram. The adder and integer multiplier perform the arithmetic operations with integer numbers.



There are a variety of ways that parallel processing can be classified. It can be considered from the internal organization of the processors, from the interconnection structure between processors, or from the flow of information through the system.

One classification introduced by M. J. Flynn considers the organization of a computer system by the number of instructions and data items that are manipulated simultaneously. The normal operation of a computer is to fetch instructions from memory and execute them in the processor.

The sequence of instructions read from memory constitutes an instruction stream. The operations performed on the data in the processor constitutes a data stream. Parallel processing may occur in the instruction stream, in the data stream, or in both.

Flynn's classification divides computers into four major groups as follows:

- It is based on the multiplicity of instruction Streams and Data Streams.
- Instruction Stream: Sequence of Instructions ready from memory.
- Data Stream: Operations performed on the data in processor.

		Number of Data Streams	
		Single	Multiple
Number of Instruction Streams	Single	SISD	SIMD
	Multiple	MISD	MIMD

Single instruction stream, single data stream (SISD)

Single instruction stream, multiple data stream (SIMD)

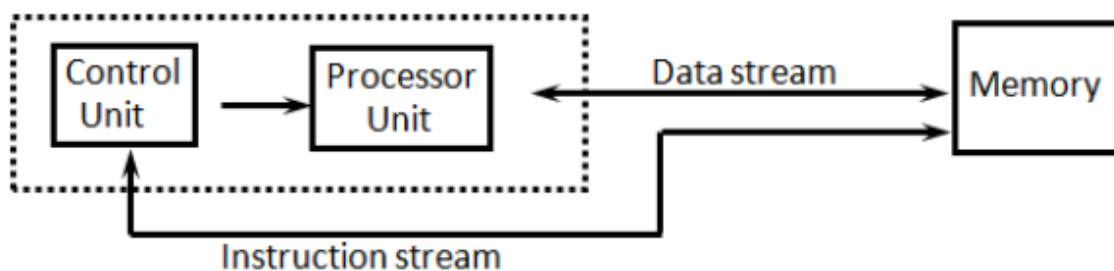
Multiple instruction stream, single data stream (MISD)

Multiple instruction stream, multiple data stream (MIMD)

SISD

- Single instruction Stream, Single data stream
- SISD represents the organization of a single computer containing a control unit a processor unit and a memory unit.

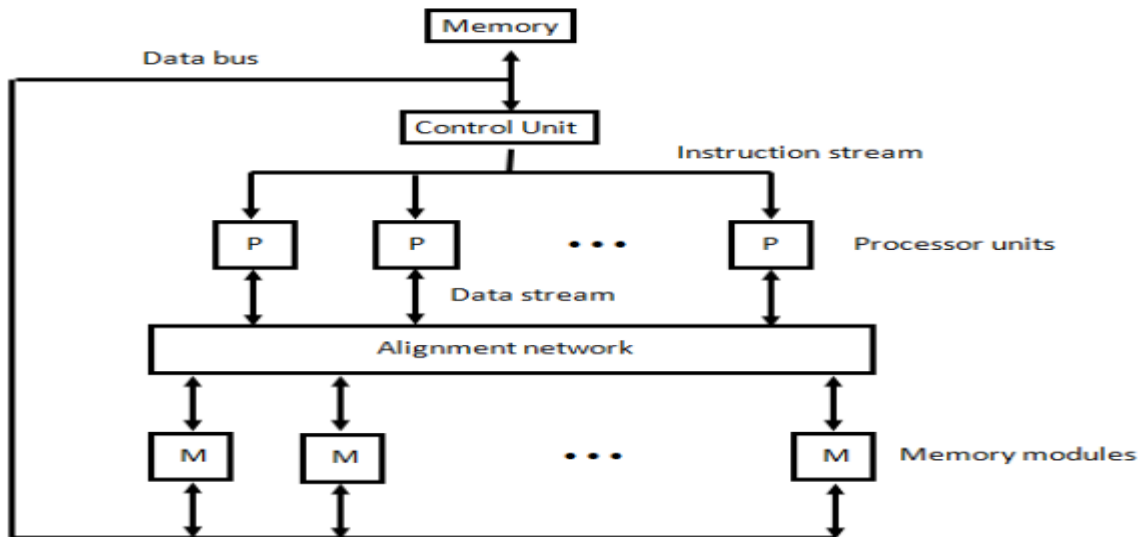
Instructions are executed sequentially and the system may or may not have internal parallel processing capabilities



SIMD

- SIMD represents an organization that includes many processing units under the supervision of a common control unit.

All processors receive the same instruction from the control unit but operate on different items of data.



MISD

- There is no computer at present that can be classified as MISD.

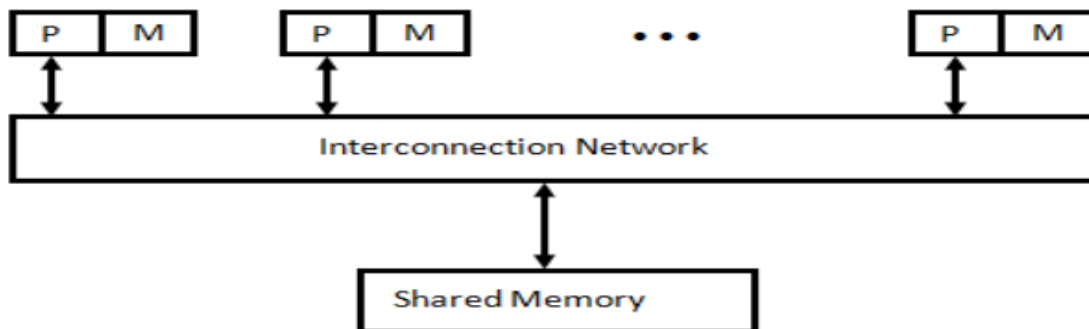
MISD structure is only of theoretical interest since no practical system has been constructed using this organization.

MIMD

- MIMD organization refers to a computer system capable of processing several programs at the same time.

Most multiprocessor and multicomputer systems can be classified in this category.

- Contains multiple processing units.
- Execution of multiple instructions on multiple data.



Concurrent access to memory and cache coherency:

The primary advantage of cache is its ability to reduce the average access time in uniprocessors. When the processor finds a word in cache during a read operation, the main memory is not involved in the transfer. If the operation is to write, there are two commonly used procedures to update memory.

Write-through policy: In the write-through policy, both cache and main memory are updated with every write operation.

Write-back policy: In the write-back policy, only the cache is updated and the location is marked so that it can be copied later into main memory.

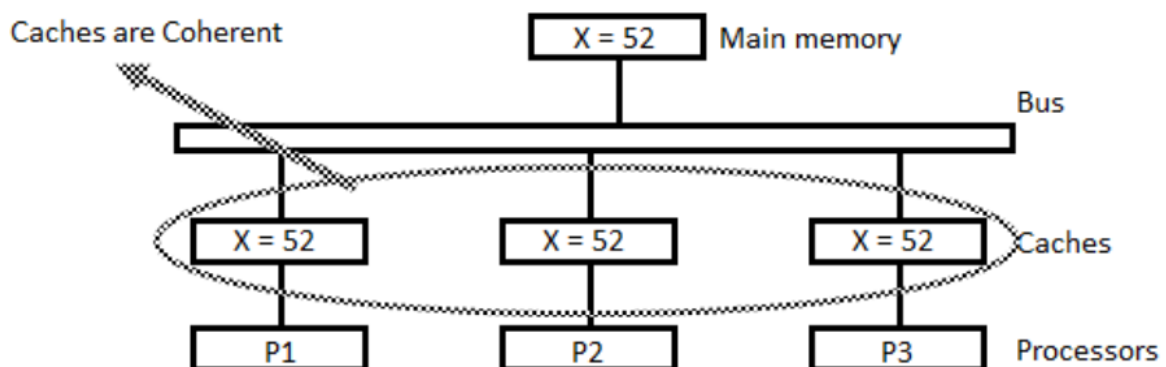
In a shared memory multiprocessor system, all the processors share a common memory. In addition, each processor may have a local memory, part or all of which may be a cache. The compelling reason for having separate caches for each processor is to reduce the average access time in each processor. The same information may reside in a number of copies in some caches and main memory.

To ensure the ability of the system to execute memory operations correctly, the multiple copies must be kept identical. This requirement imposes a cache coherence problem. A memory scheme is coherent if the value returned on a load instruction is always the value given by the latest store instruction with the same address. Without a proper solution to the cache coherence problem, caching cannot be used in bus oriented multiprocessors with two or more processors.

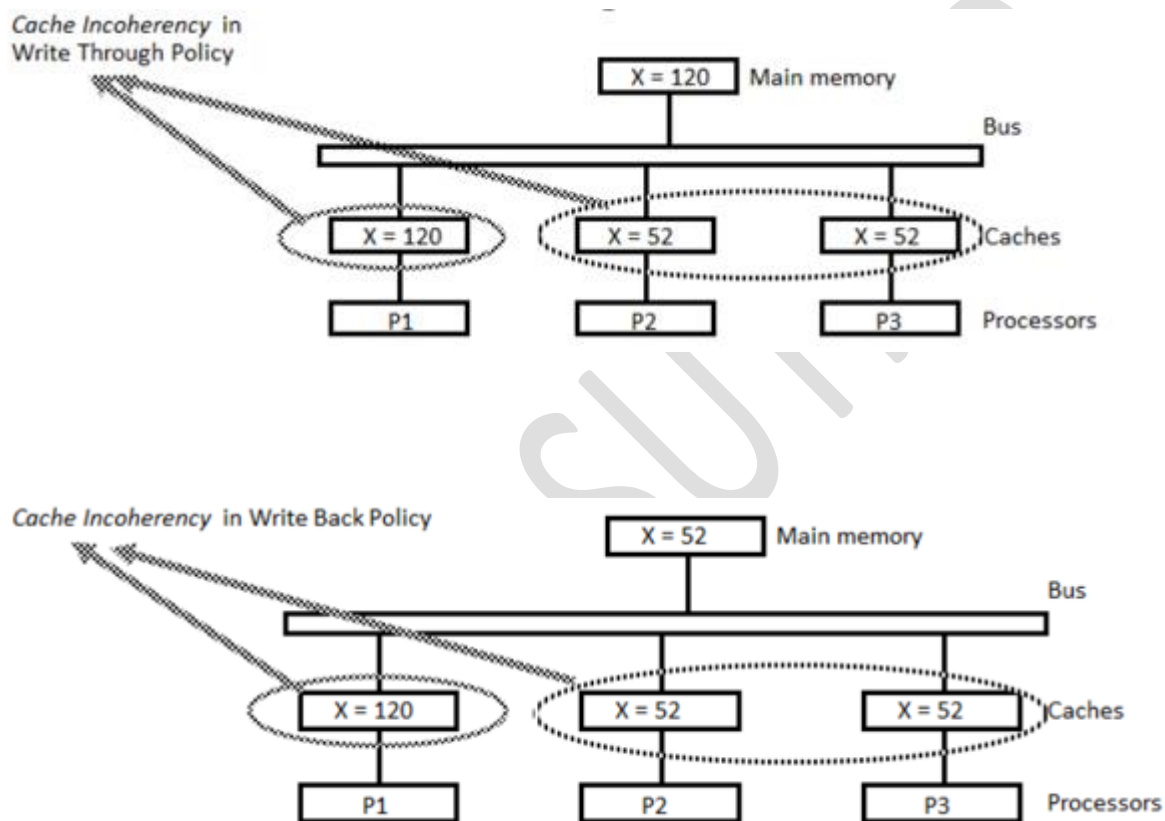
Conditions for Incoherence

Cache coherence problems exist in multiprocessors with private caches because of the need to share writable data. Read-only data can safely be replicated without cache coherence enforcement mechanisms.

To illustrate the problem, consider the three-processor configuration with private caches shown in Fig.. Sometime during the operation an element X from main memory is loaded into the three processors, P1, P2, and P3. As a consequence, it is also copied into the private caches of the three processors. For simplicity, we assume that X contains the value of 52. The load on X to the three processors results in consistent copies in the caches and main memory. If one of the processors performs a store to X, the copies of X in the caches become inconsistent. A load by the other processors will not return the latest value. Depending on the memory update policy used in the cache, the main memory may also be inconsistent with respect to the cache.



This is shown in below Fig. A store to X (of the value of 120) into the cache of processor P1 updates memory to the new value in a write-through policy. A write-through policy maintains consistency between memory and the originating cache, but the other two caches are inconsistent since they still hold the old value. In a write-back policy, main memory is not updated at the time of the store. The copies in the other two caches and main memory are inconsistent. Memory is updated eventually when the modified data in the cache are copied back into memory.



Another configuration that may cause consistency problems is a direct memory access (DMA) activity in conjunction with an IOP connected to the system bus. In the case of input, the DMA may modify locations in main memory that also reside in cache without updating the cache. During a DMA output, memory locations may be read before they are updated from the cache when using a write-back policy. VO-based memory incoherence can be overcome by making the IOP a participant in the cache coherent solution that is adopted in the system.