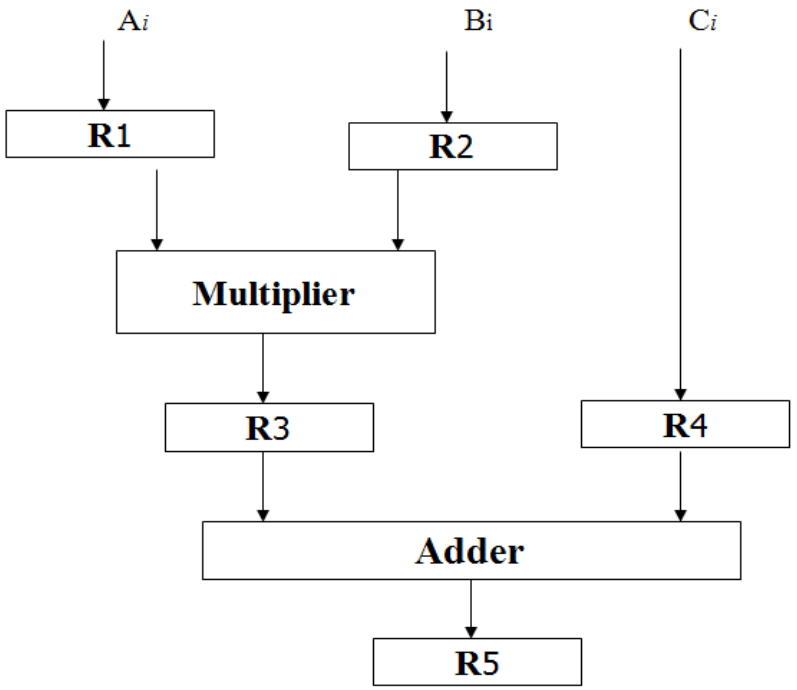
PIPELINING

- Pipelining is a technique of *decomposing a sequential process into sub-operations,* with each sub-process being executed in a special dedicated segment that operates *concurrently* with all other segments.
- The overlapping of computation is made possible by associating a *register* with each segment in the pipeline.
- The registers provide isolation between each segment so that each can operate on distinct data *simultaneously*.
- Perhaps the simplest way of viewing the pipeline structure is to imagine that each segment consists of an *input register* followed by a *combinational circuit*.
- The register holds the data.
- The combinational circuit performs the sub-operation in the particular segment.

PIPELINING

- A clock is applied to all registers after *enough time* has elapsed to perform all segment activity.
- The pipeline organization will be demonstrated by means of a simple example.

- A*i* * B*i* + C*i* for *i* = 1, 2, 3, …, 7
- Each suboperation is to be implemented in a segment within a pipeline.
- R1 ← A*i*, R2 ← B*i* Input A*i* and B*i*
- R3 ← R1 * R2, R4 ← C*i* Multiply and input C*i*
- R5 ← R3 + R4 Add C*i* to product



PIPELINING

- Each segment has one or two registers and a combinational circuit as shown in above figure. The five registers are loaded with new data every clock pulse. The effect of each clock is shown in following table.
- If the above concept is executed without the pipelining, then each data operation will be taking 5 cycles, totally they are 35 cycles of CPU are needed to perform the operation.
- But if are using the concept of pipeline, we will be cutting off many cycles.

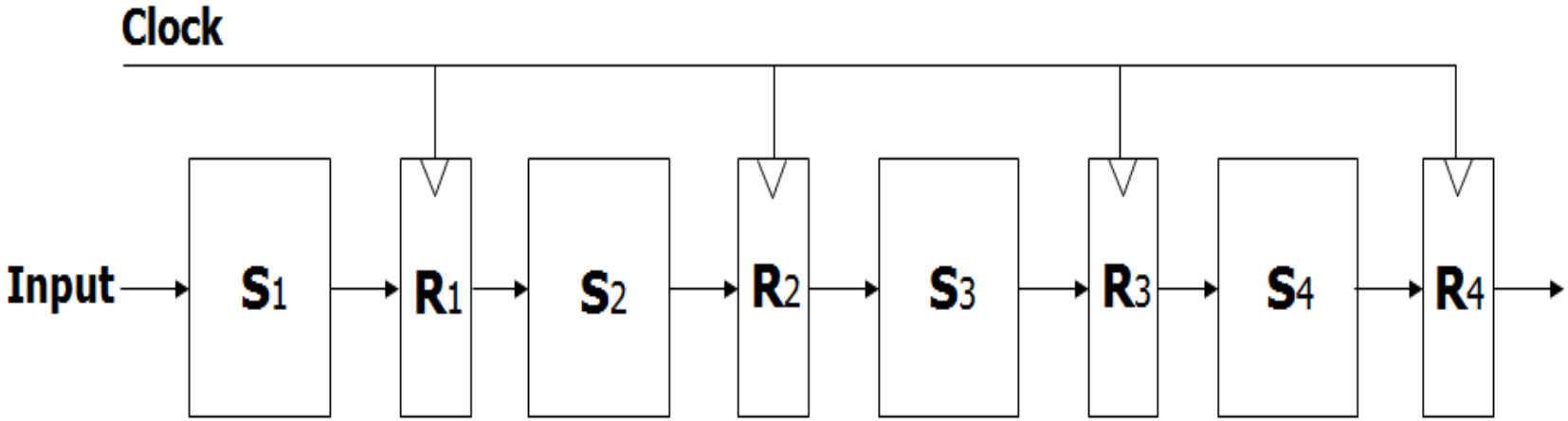
Clock Pulse number	Segment1		Segment2		Segment3
	R1	R2	R3	R4	R5
1	A_1	B_1	----	----	----
2	A_2	B_2	A_1*B_1	C_1	----
3	A_3	B_3	A_2*B_2	C_2	$A_1*B_1+C_1$
4	A_4	B_4	A_3*B_3	C_3	$A_2*B_2+C_2$
5	A_5	B_5	A_4*B_4	C_4	$A_3*B_3+C_3$
6	A_6	B_6	A_5*B_5	C_5	$A_4*B_4+C_4$
7	A_7	B_7	A_6*B_6	C_6	$A_5*B_5+C_5$
8	----	----	A_7*B_7	C_7	$A_6*B_6+C_6$
9	----	----	----	----	$A_7*B_7+C_7$

PIPELINING

General Considerations:

- Any operation that can be decomposed into a sequence of suboperations of about the *same complexity* can be implemented by a pipeline processor.
- The technique is efficient for those applications that need to repeat the same task many time with different sets of data.
- The general structure of a four-segment pipeline is illustrated in following figure. We define a *task* as the total operation performed going through all the segments in the pipeline.
- The behavior of a pipeline can be illustrated with a *space-time* diagram. It shows the segment utilization as a function of time.

PIPELINING



	1	2	3	4	5	6	7	8	9
Segment: 1	T1	T2	T3	T4	T5	T6			
2		T1	T2	T3	T4	T5	T6		
3			T1	T2	T3	T4	T5	T6	
4				T1	T2	T3	T4	T5	T6

Clock cycle →

PIPELINING

- The space-time diagram of a four-segment pipeline is demonstrated in following figure.Where a *k*-segment pipeline with a clock cycle time t_p is used to execute *n* tasks.
- The first task T1 requires a time equal to kt_p to complete its operation.
- The remaining *n*-1 tasks will be completed after a time equal to (*n*-1)tp
- Therefore, to complete *n* tasks using a *k*-segment pipeline requires *k*+(*n*-1) clock cycles.
- Consider a nonpipeline unit that performs the same operation and takes a time equal to t_n to complete each task. The total time required for *n* tasks is nt_n.

Speedup:

- The *speedup of a pipeline processing* over an equivalent non-pipeline processing is defined by the ratio
- $$S = \frac{nt_n}{(k + n - 1)t_p}.$$
- As the number of tasks increase, the speedup becomes
- $$S = \frac{t_n}{t_p}$$
- If we assume that the time it takes to process a task is the same in the pipeline and non-pipeline circuits.
- i.e., $t_n = kt_p$, the speedup reduces to $S = kt_p / t_p = k$.
- This shows that the theoretical maximum speed up that a pipeline can provide is *k*, where *k* is the number of segments in the pipeline.

Speedup:

- Example:
- Cycle time = t_p = 20 ns
- Number of segments = k = 4
- Number of tasks = n = 100
- The pipeline system will take $(k + n – 1)t_p$ = (4 + 100 –1)20ns = 2060 ns
- Assuming that $t_n = kt_p$ = 4 * 20 = 80 ns,
- A nonpipeline system requires nkt_p = 100 * 80 = 8000 ns
- The speedup ratio = 8000/2060 = 3.88

Pipeline Hazards

- There are situations, called hazards, that prevent the next instruction in the instruction stream from executing during its designated cycle
- There are three classes of hazards
  - Structural hazard
  - Data hazard
  - Branch hazard

Structural Hazards. They arise from resource conflicts when the **hardware cannot support** al possible combinations of instructions in simultaneous overlapped execution.

Data Hazards. They arise when an instruction depends on the **result of a previous instruction** in a way that is exposed by the overlapping of instructions in the pipeline.

Control Hazards.They arise from the **pipelining of branches and other instructions** that change the PC.

Pipeline Hazards

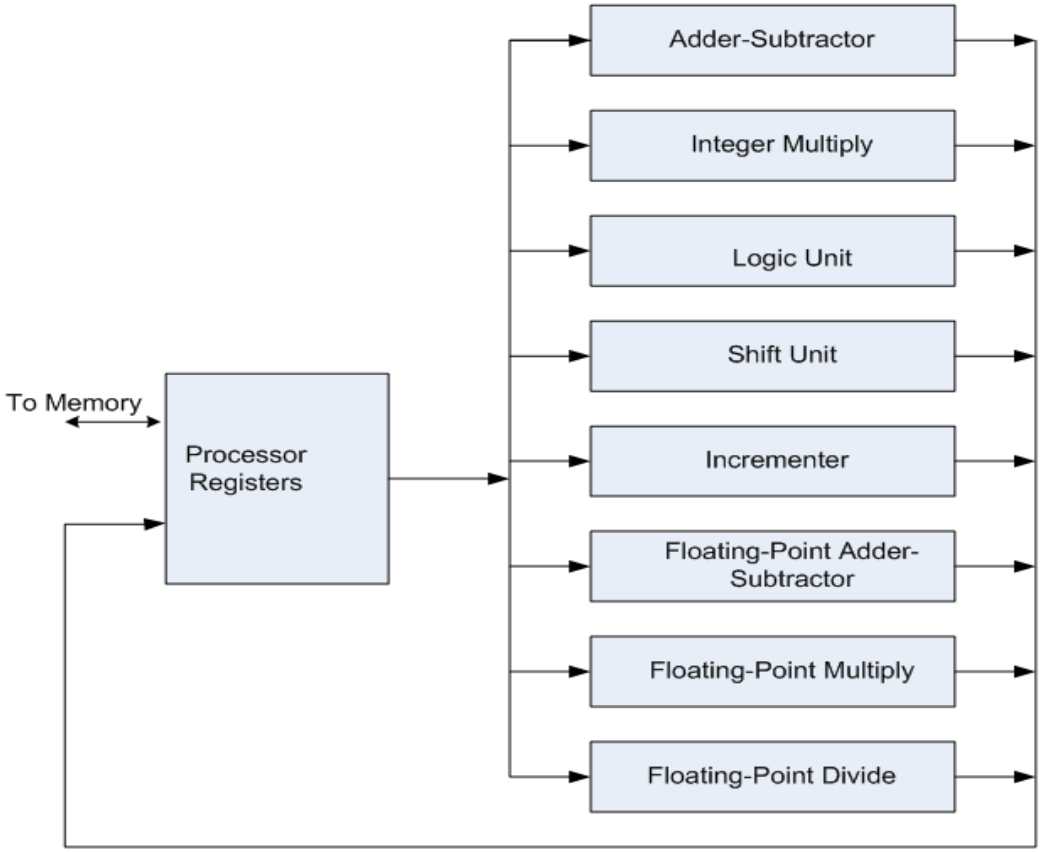
- Where one instruction cannot immediately follow another
- Types of hazards
  - Structural hazards - attempt to use the same resource by two or more instructions
  - Control hazards - attempt to make branching decisions before branch condition is evaluated
  - Data hazards - attempt to use data before it is ready
- Can always resolve hazards by waiting

PARALLEL PROCESSING

- *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.
- 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.
- 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. e.g. shift registers and registers with parallel load
- At a **higher level**, it can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously.

PARALLEL PROCESSING

- Following Fig. shows one possible way of separating the execution unit into eight functional units operating in parallel.
- A multifunctional organization is usually associated with a complex control unit to coordinate all the activities among the various components.



PARALLEL PROCESSING

- In the above figure we can see that the data stored in the processor registers is being sent to separate devices basing on the operation needed on the data.
- If the data inside the processor registers is requesting for an arithmetic operation, then the data will be sent to the arithmetic unit and if in the same time another data is requested in the logic unit, then the data will be sent to logic unit for logical operations.
- Now in the same time both arithmetic operations and logical operations are executing in parallel. This is called as parallel processing.

PARALLEL PROCESSING

- The computers are classified into 4 types based on the Instruction Stream and Data Stream. They are called as the Flynn's Classification of computers.
- ***Instruction Stream:*** The sequence of instructions read from the memory is called as an Instruction Stream
- ***Data Stream:*** The operations performed on the data in the processor are called as a Data Stream.
- There are a variety of ways that parallel processing can be classified.
    1) .Internal organization of the processors.
    2) .Interconnection structure between processors.
    3) .The flow of information through the system.

M. J. Flynn Classifications

- **M. J. Flynn considers the organization of a computer system by the number of instructions and data items that are manipulated simultaneously.**
    1. Single instruction stream, single data stream (SISD)
    2. Single instruction stream, multiple data stream (SIMD)
    3. Multiple instruction stream, single data stream (MISD)
    4. Multiple instruction stream, multiple data stream (MIMD)
- **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.
- Parallel processing may be achieved by means of multiple functional units or by pipeline processing.
- **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.
- The shared memory unit must contain multiple modules so that it can communicate with all the processors simultaneously.

M. J. Flynn Classifications

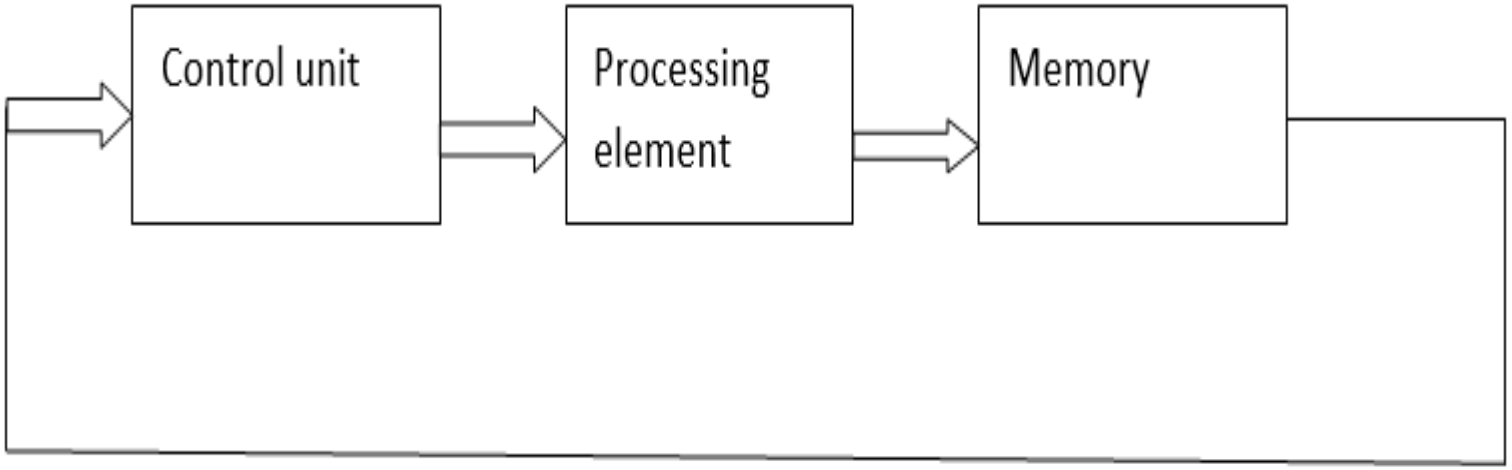
- **MISD & MIMD**
- MISD structure is only of theoretical interest since no practical system has been constructed using this organization. MIMD organization refers to a computer system capable of processing several programs at the same time. e.g. multiprocessor and multicomputer system
- One type of parallel processing that does not fit Flynn's classification is pipelining. We consider parallel processing under the following main topics:
- **Pipeline processing**:

  Is an implementation technique where arithmetic sub operations or the phases of a computer instruction cycle overlap in execution.
- **Vector processing:**

  Deals with computations involving large vectors and matrices.
- **Array processing:**

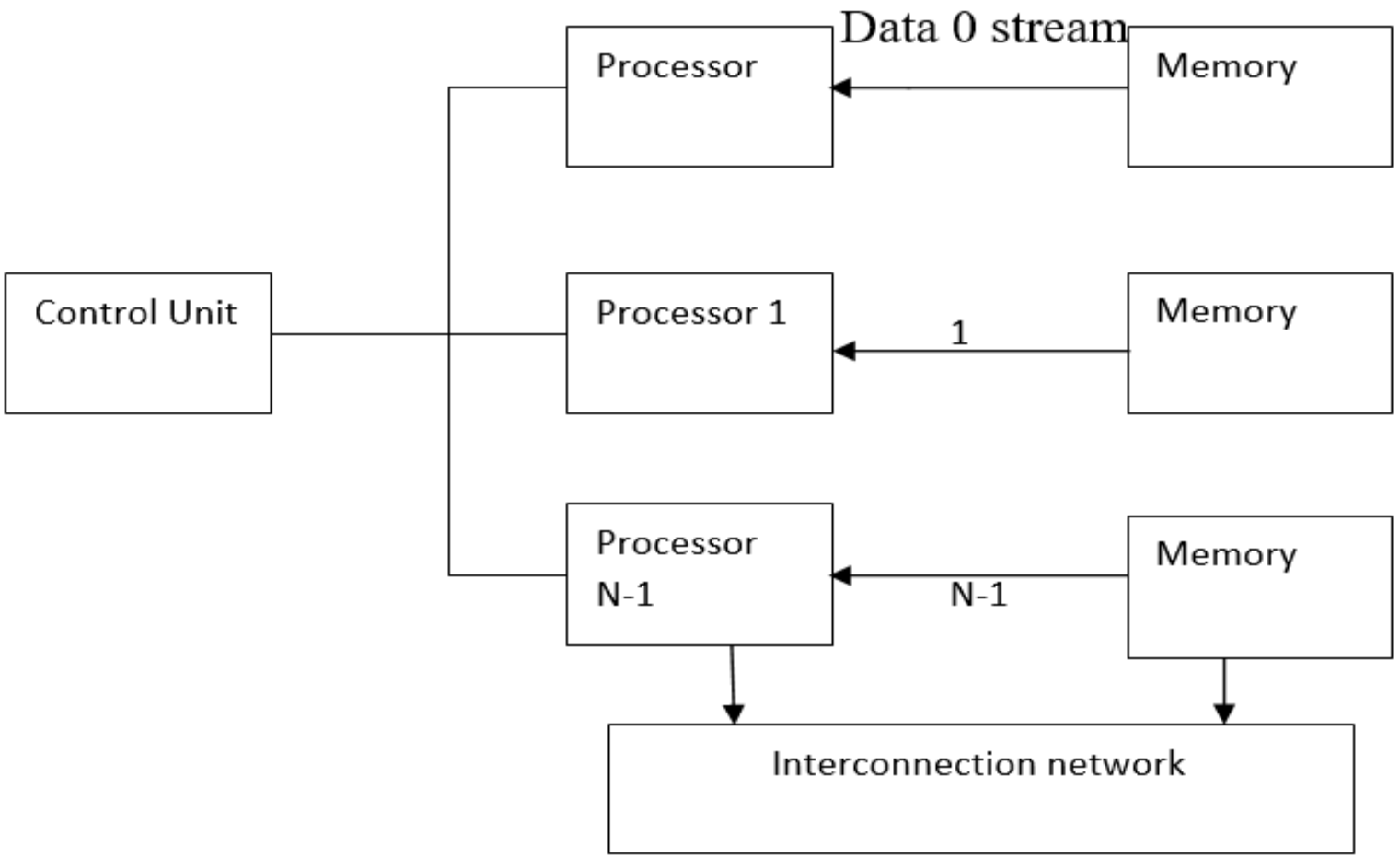
  Perform computations on large arrays of data.

SISD

- **Single instruction:** Only one instruction stream is being acted or executed by CPU during one clock cycle.
- **Single data stream:** Only one data stream is used as input during one clock cycle.
- A SISD computing system is a uniprocessor machine that is capable of executing a single instruction operating on a single data stream.
- Most conventional computers have SISD architecture where all the instruction and data to be processed have to be stored in primary memory.
- 

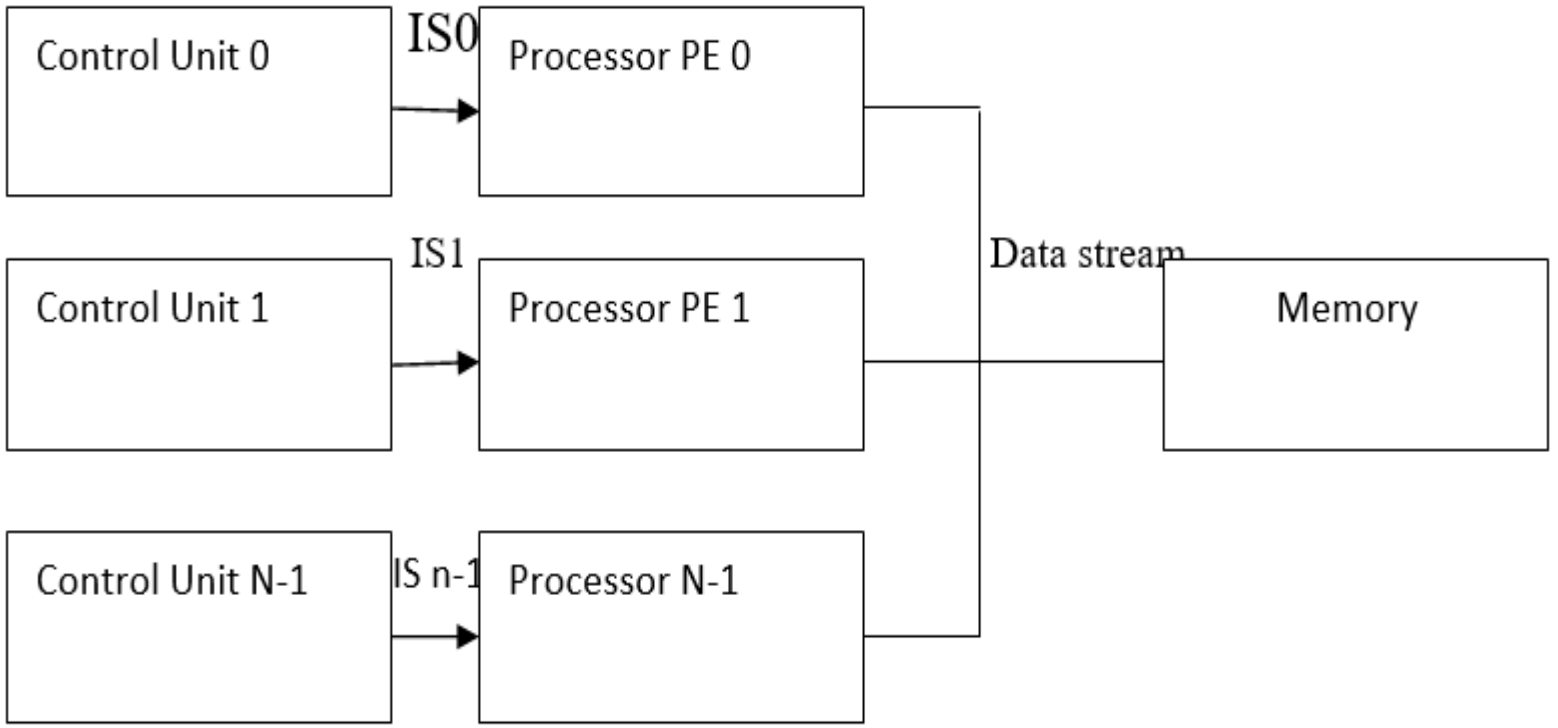
SIMD (Single Instruction Multiple Data Stream)

- A SIMD system is a multiprocessor machine, capable of executing the same instruction on all the CPUs but operating on the different data stream.
-



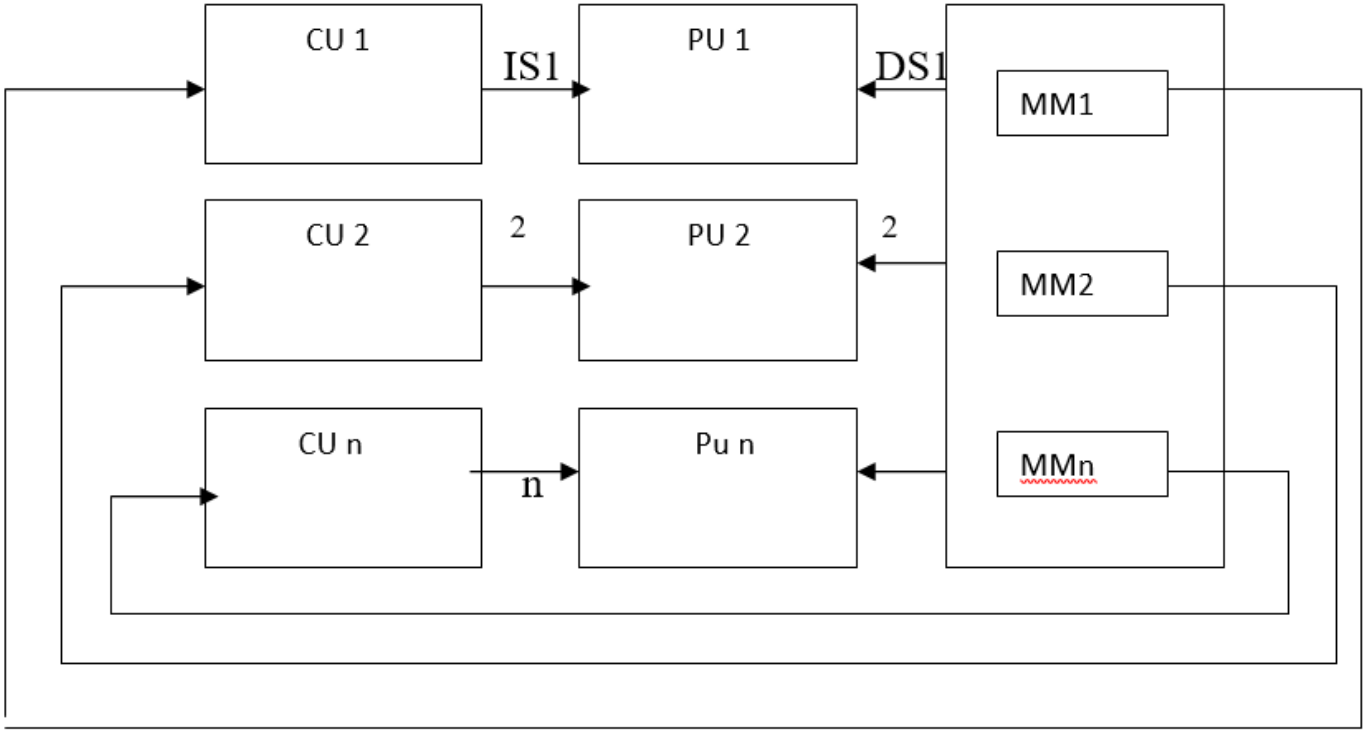
MISD (Multiple Instruction Single Data stream)

- An MISD computing is a multiprocessor machine capable of executing different instructions on processing elements but all of them operating on the same data set.



MIMD (Multiple Instruction Multiple Data Stream)

- A MIMD system is a multiprocessor machine that is capable of executing multiple instructions over multiple data streams. Each processing element has a separate instruction stream and data stream.

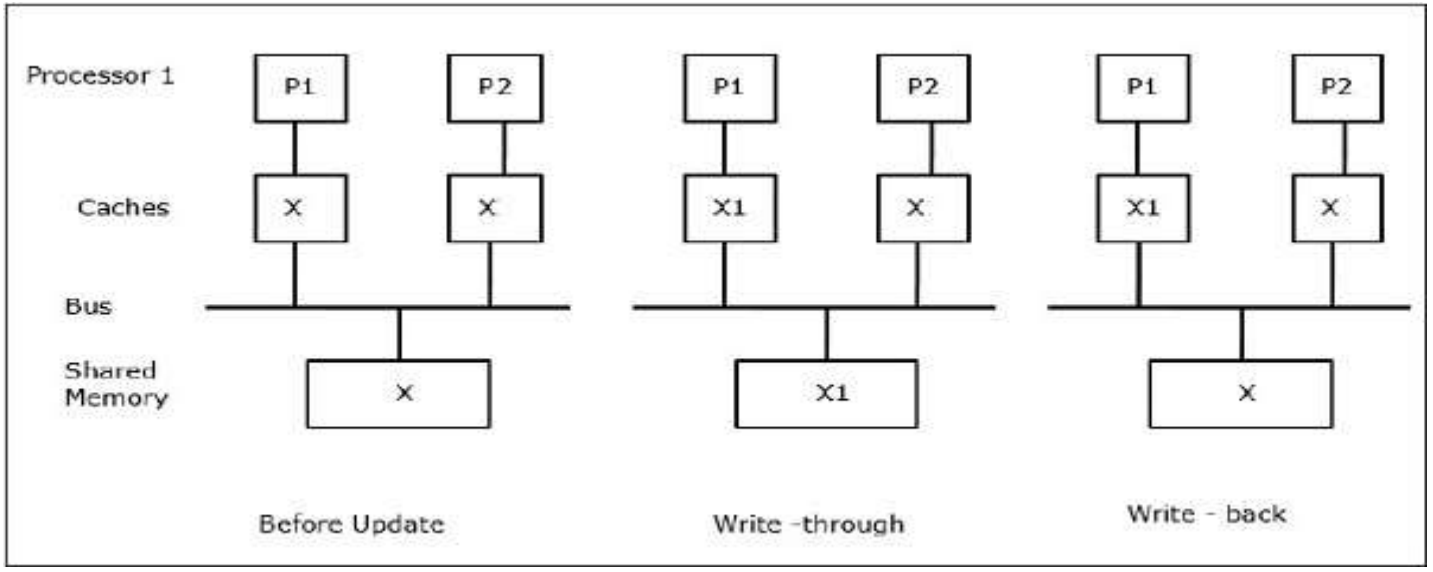


Cache Coherence

- In a multiprocessor system, data inconsistency may occur among adjacent levels or within the same level of the memory hierarchy.
- For example, the cache and the main memory may have inconsistent copies of the same object.
- As multiple processors operate in parallel, and independently multiple caches may possess different copies of the same memory block, this creates **cache coherence problem**.
- **Cache coherence schemes** help to avoid this problem by maintaining a uniform state for each cached block of data.

Cache Coherence

- Let X be an element of shared data which has been referenced by two processors, P1 and P2.
- In the beginning, three copies of X are consistent.
- If the processor P1 writes a new data X1 into the cache, by using **write-through policy**, the same copy will be written immediately into the shared memory.
- In this case, inconsistency occurs between cache memory and the main memory. When a **write-back policy** is used, the main memory will be updated when the modified data in the cache is replaced or invalidated.



Cache Write Policies

- here two main cache write policies.
- ***Write back*** : Write operations are usually made only to the cache. Main memory is only updated when the corresponding cache line is flushed from the cache.
- ***Write through*** : All write operations are made to main memory as well as to the cache, ensuring that main memory is always valid.