SUBJECT NAME: Computer Organization & Architecture

SUBJECT CODE: 203105253

UNIT 5

PIPELINING:

Basic concepts of pipelining, throughput and speedup, pipeline hazards.

PARALLEL PROCESSORS:

Introduction to parallel processors, Concurrent access to memory and cache coherency.

PIPELINING:

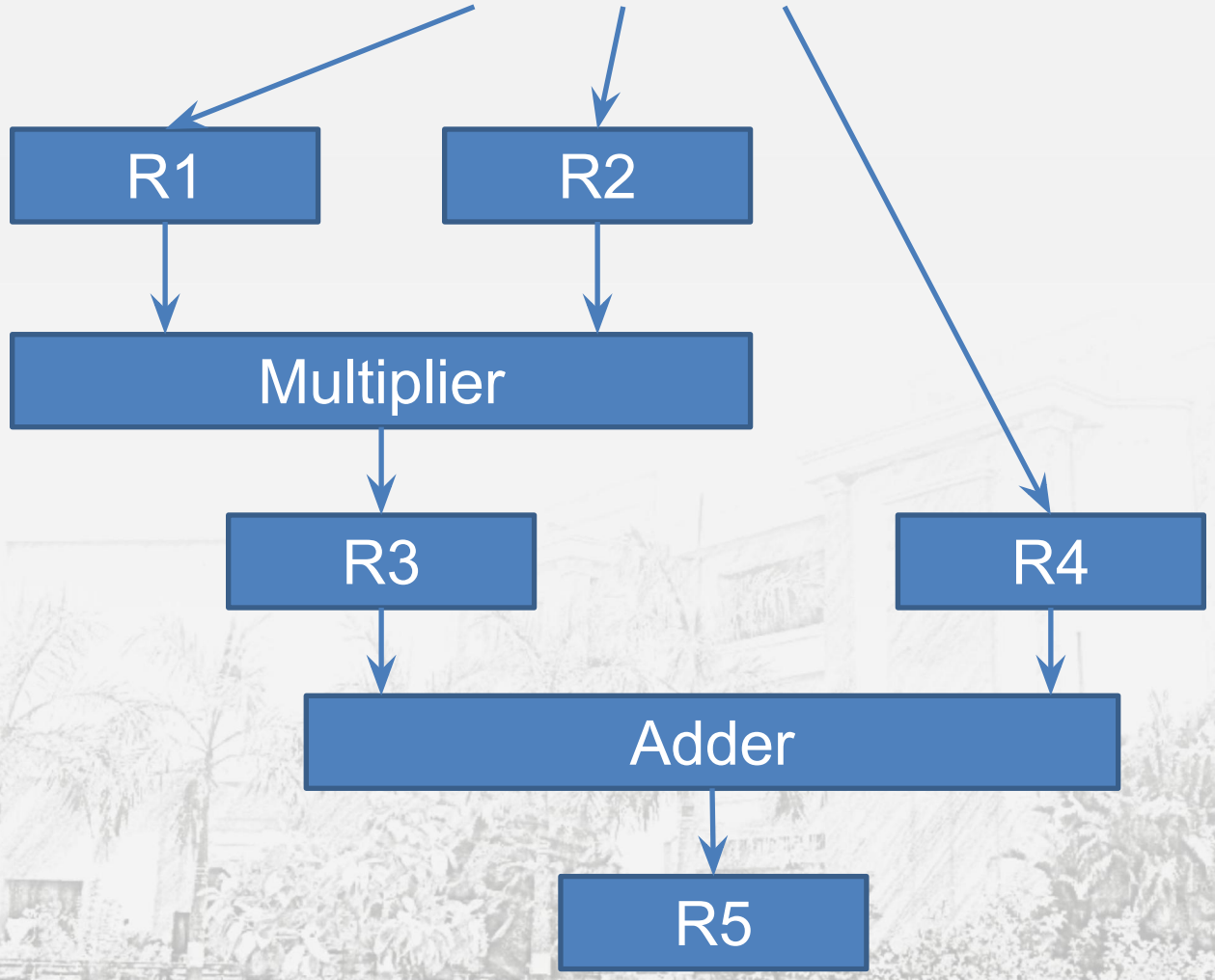
- **Basic concepts of pipelining:**
  Performance of a computer can be increased by increasing the performance of the CPU.

- This can be done by executing more than one task at a time. This procedure is referred to as pipelining.

- The concept of pipelining is to allow the processing of a new task even though the processing of previous task has not ended

Pipelining example

$A_i * B_i + C_i$ for $i = 1,2,3,\ldots,7$



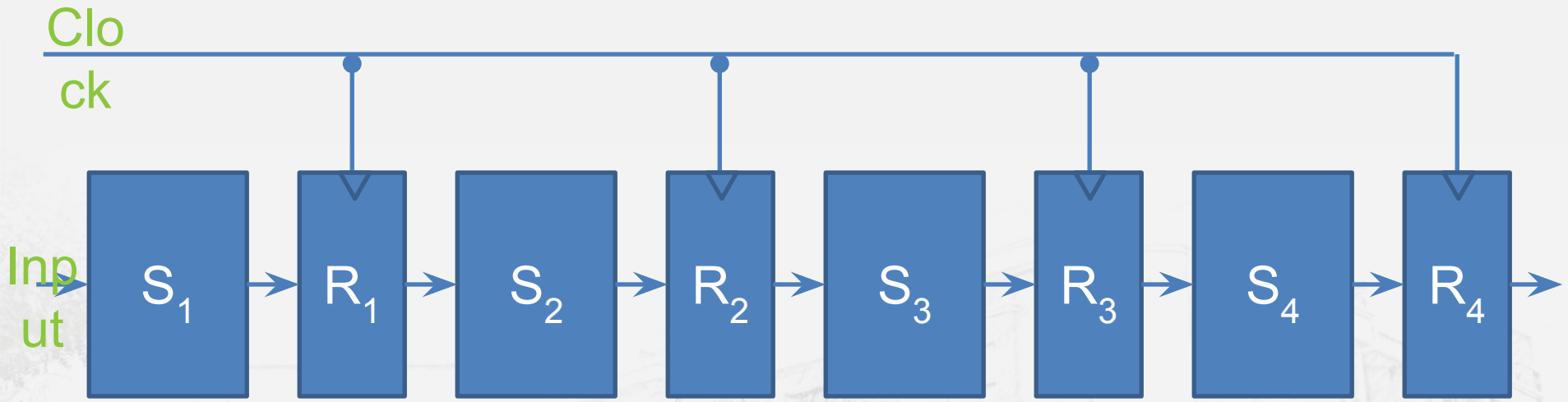
- **Pipelining** is a technique of decomposing a sequential process into sub operations, with each subprocess being executed in a special dedicated segment that operates concurrently with all other segments.
- A pipeline can be visualized as a collection of processing segments through which binary information flows.
- Each segment performs partial processing dictated by the way the task is partitioned.
- The result obtained from the computation in each segment is transferred to the next segment in the pipeline.
- The registers provide isolation between each segment.
- The final result is obtained after the data have passed through all segments.
- The technique is efficient for those applications that need to repeat the same task many times with different sets of data.

- Consider the following operation: **Result=(A+B)*C**
  First the A and B values are Fetched which is nothing but a “Fetch Operation”.
  The result of the Fetch operations is given as input to the Addition operation, which is an Arithmetic operation.
  The result of the Arithmetic operation is again given to the Data operand C which is fetched from the memory and using another arithmetic operation which is Multiplication in this scenario is executed.
  Finally the Result is again stored in the “Result” variable.

- In this process we are using up-to 5 pipelines which are
  Fetch Operation (A), Fetch Operation(B)
  Addition of (A & B), Fetch Operation(C)
  Multiplication of ((A+B), C)
  Load ((A+B)*C)

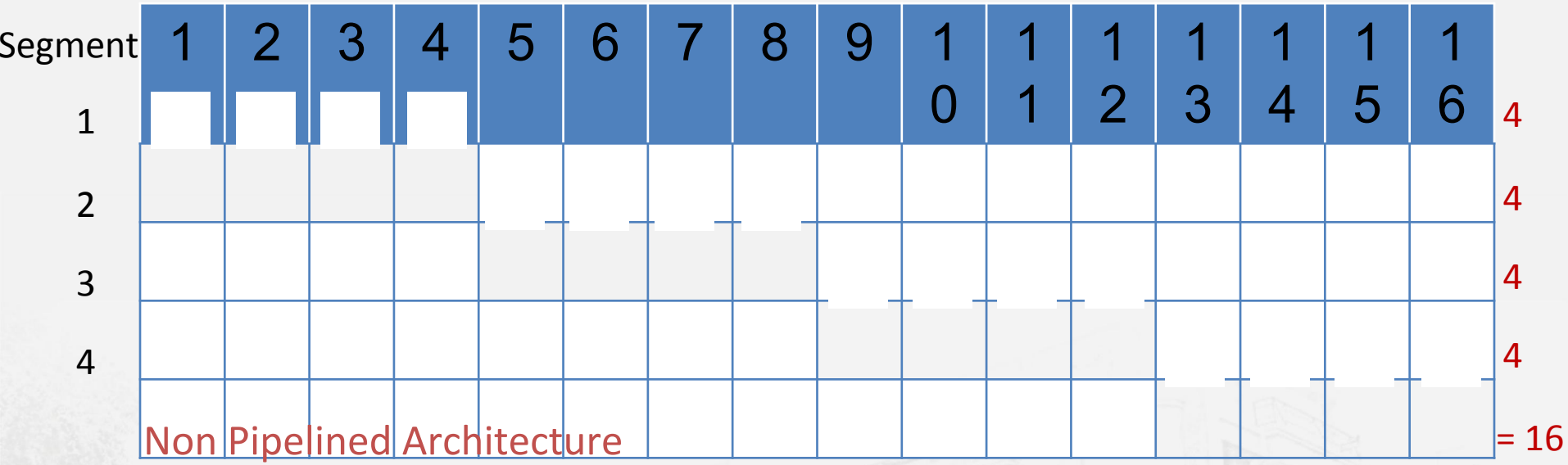
Pipelining

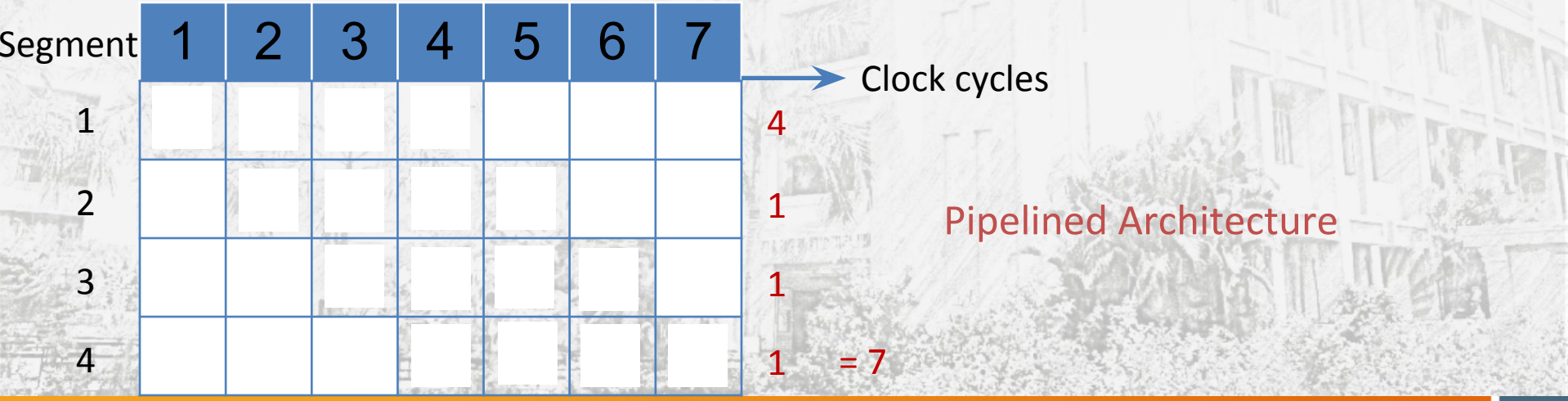
- General structure of four segment pipeline



- The operands pass through all 4 segments in a fixed sequence.
- Each segment consists of a combinational circuit S (performs sub operation)
- The segments are separated by registers R, which hold the intermediate results. b/w stages.

Space-time Diagram





Throughput and Speedup

- 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.
- **Throughput:** 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

Speedup

- Speedup of a pipeline processing over an equivalent non-pipeline processing is defined by the ratio (no of instruction/ total time to complete instructions)

$$S = \frac{nt_n}{(k+n-1)t_p}$$

- If number of tasks in pipeline increases w.r.t. number of segments then n becomes larger than $k-1$, under this condition speedup becomes
- Assuming time to process a task in pipeline and non-pipeline circuit is same then

$$S = \frac{nt_n}{nt_p} = \frac{t_n}{t_p}$$

- Theoretically maximum speedup achieved is the number of segments in the pipeline.

$$S = \frac{kt_p}{t_p} = k$$

Arithmetic Pipeline

Arithmetic Pipeline

- Usually found in high speed computers.
- Used to implement floating point operations, multiplication of fixed point numbers and similar operations.

Example of Arithmetic Pipeline

- Consider an example of floating point addition and subtraction.

$$X = A \times 10^{a}$$
$$Y = B \times 10^{b}$$

- A and B are two fractions that represent the mantissas and a and b are the exponents.

Example of Arithmetic Pipeline

- Consider the two normalized floating-point numbers:

$$X = 0.9504 \times 10^3 \qquad Y = 0.8200 \times 10^2$$

- Segment-1: The larger exponent is chosen as the exponent of result.
- Segment-2: Aligning the mantissa numbers

$$X = 0.9504 \times 10^3 \qquad Y = 0.0820 \times 10^3$$

- Segment-3: Addition of the two mantissas produces the sum

$$Z = 1.0324 \times 10^3$$

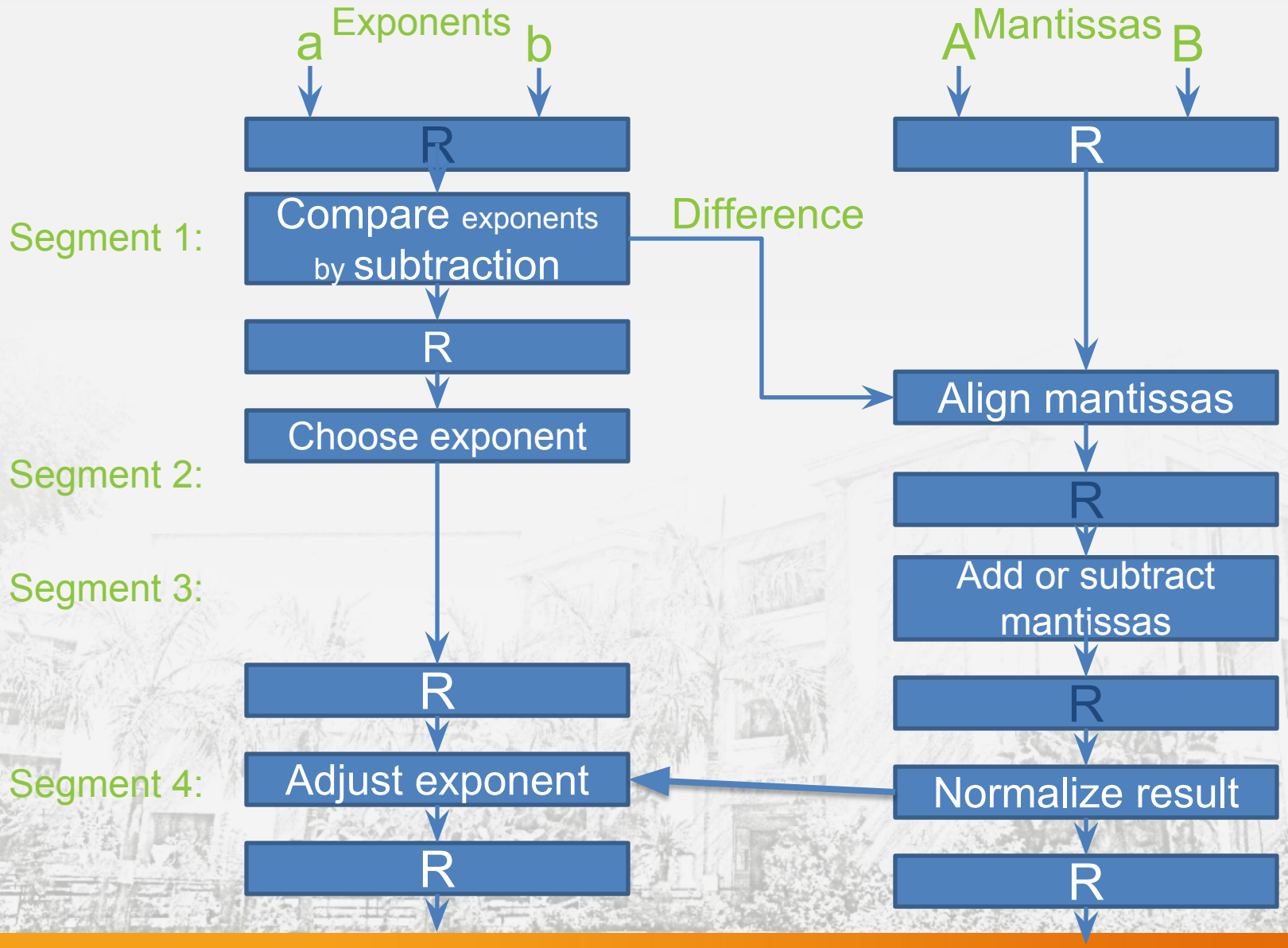
- Segment-4: Normalize the result

$$Z = 0.10324 \times 10^4$$

Example of Arithmetic Pipeline

- The sub-operations that are performed in the four segments are:
  1. Compare the exponents
  2. Align the mantissas
  3. Add or subtract the mantissas
  4. Normalize the result

Arithmetic pipeline



Instruction Pipeline

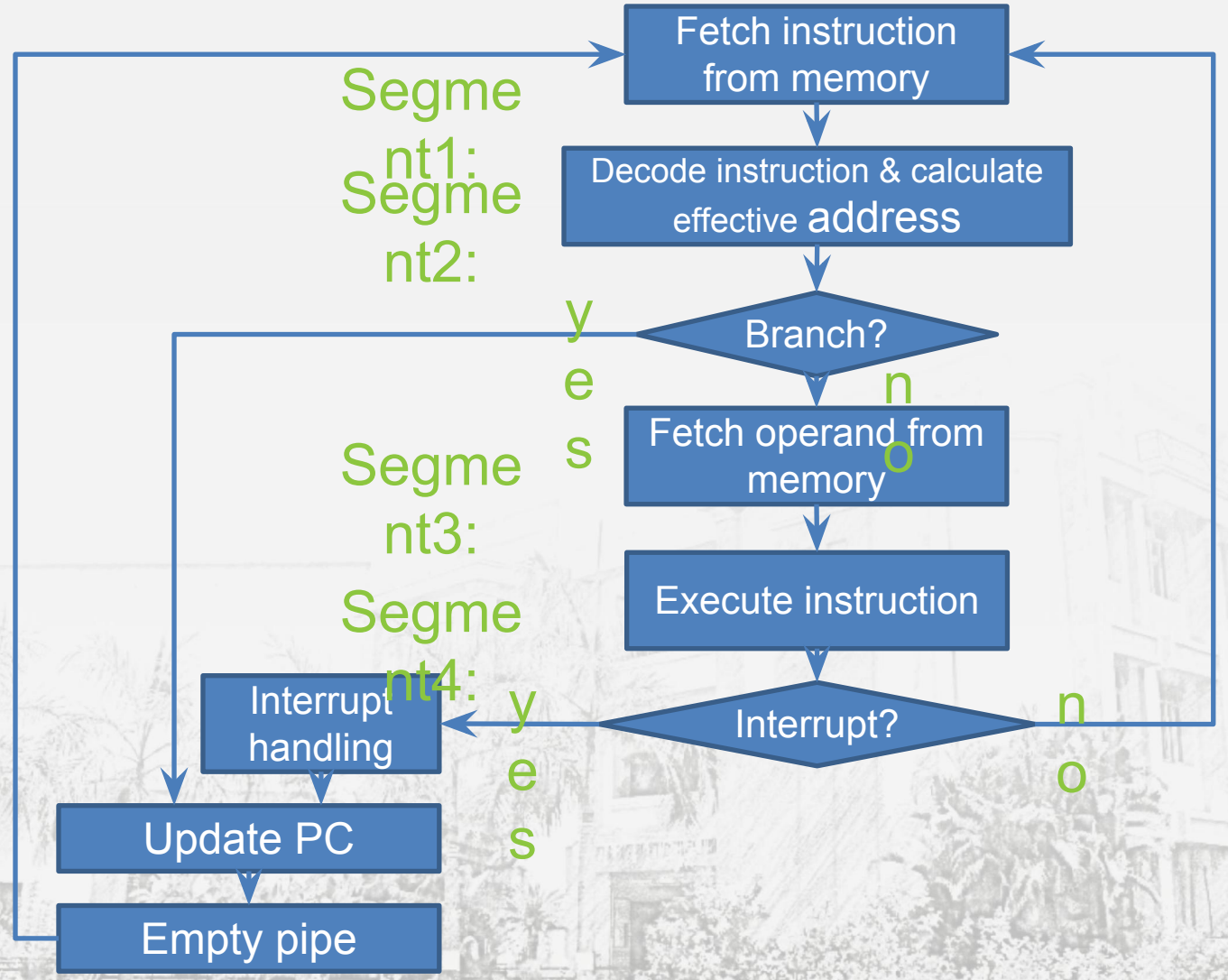
Instruction Pipeline

- In the most general case, the computer needs to process each instruction with the following sequence of steps
  1. Fetch the instruction from memory.
  2. Decode the instruction.
  3. Calculate the effective address.
  4. Fetch the operands from memory.
  5. Execute the instruction.
  6. Store the result in the proper place.
- Different segments may take different times to operate on the incoming information.
- Some segments are skipped for certain operations.
- The design of an instruction pipeline will be most efficient if the instruction cycle is divided into segments of equal duration.

Instruction Pipeline

- Assume that the decoding of the instruction can be combined with the calculation of the effective address into one segment.
- Assume further that most of the instructions place the result into a processor registers so that the instruction execution and storing of the result can be combined into one segment.
- This reduces the instruction pipeline into four segments.
  1. FI: Fetch an instruction from memory
  2. DA: Decode the instruction and calculate the effective address of the operand
  3. FO: Fetch the operand
  4. EX: Execute the operation

Four segment CPU pipeline



Space-time Diagram

Segment (Branch)	1	2	3	4	5	6	7	8	9	10	11	12	13
1	FI	DA	FO	EX									
2		FI	DA	FO	EX								
3			FI	DA	FO	EX							
4				FI	DA	FO	EX						
5					FI	DA	FO	EX					
6						FI	DA	FO	EX				
7							FI	DA	FO	EX			

Pipeline Hazards

- There are situations in pipelining when the next instruction cannot execute in the following clock cycle. These events are called *hazards*,
- there are three different types of hazards.
- 1) **structural hazard**
- **2) Data Hazards**
- **3) Control Hazards**

structural hazard

- The first hazard is called a **structural hazard**. It means that the hardware cannot support the combination of instructions that we want to execute in the same clock cycle.
- A structural hazard in the laundry room would occur if we used a washer dryer combination instead of a separate washer and dryer, or if our roommate was busy doing something else and wouldn't put clothes away. Our carefully scheduled pipeline plans would then be foiled.

Data Hazards

- **Data hazards** occur when the pipeline must be stalled because one step must wait for another to complete.
- Suppose you found a sock at the folding station for which no match existed. One possible strategy is to run down to your room and search through your clothes bureau to see if you can find the match. Obviously, while you are doing the search, loads must wait that have completed drying and are ready to fold as well as those that have finished washing and are ready to dry.
- In a pipeline, data hazards arise from the dependence of one instruction on an earlier one that is still in the pipeline (a relationship that does not really exist when doing laundry).

Control Hazards

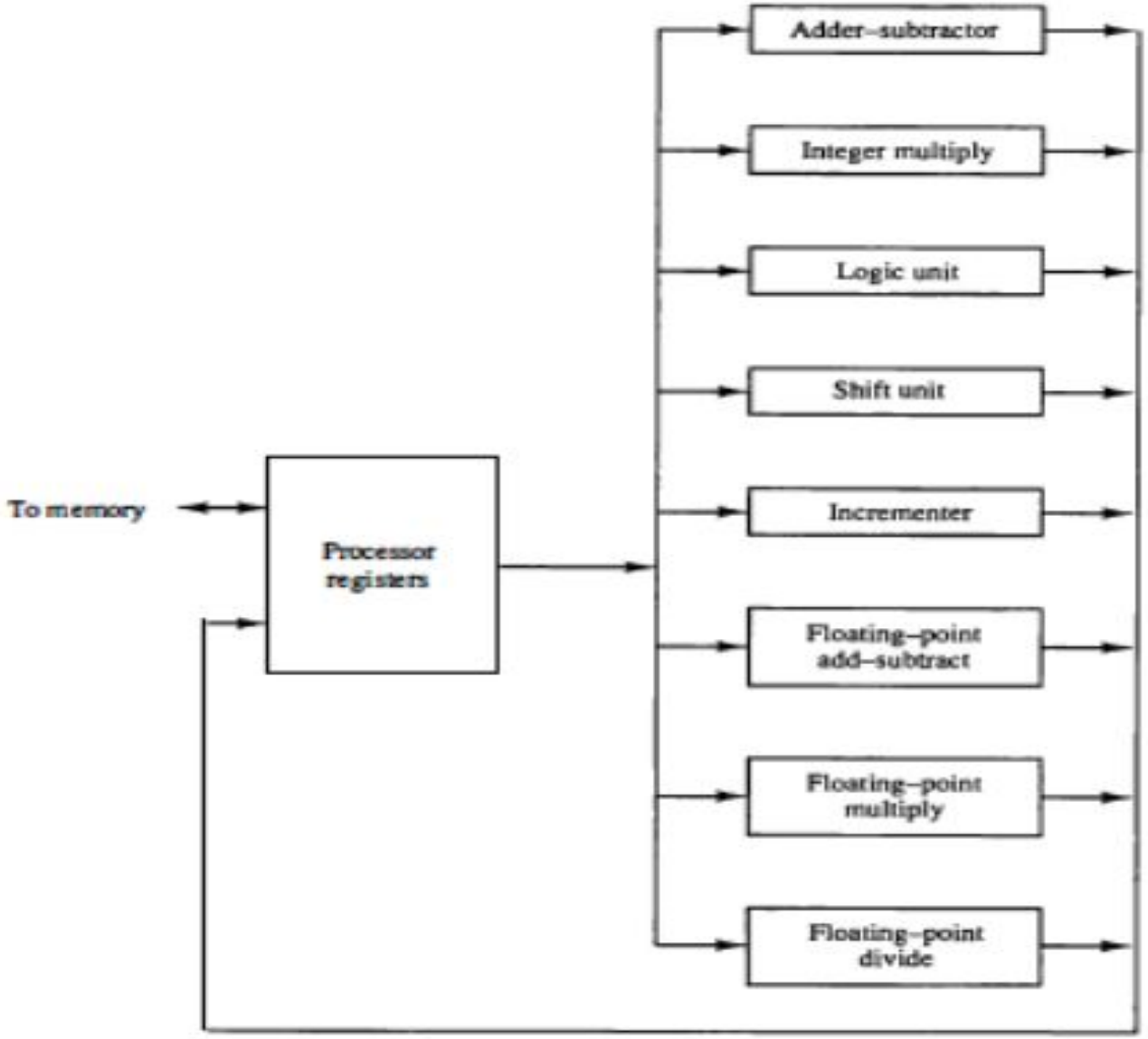
- The third type of hazard is called a **control hazard**, arising from the need to make a decision based on the results of one instruction while others are executing.
- Suppose our laundry crew was given the happy task of cleaning the uniforms of a football team. Given how filthy the laundry is, we need to determine whether the detergent and water temperature setting we select is strong enough to get the uniforms clean but not so strong that the uniforms wear out sooner. In our laundry pipeline, we have to wait until after the second stage to examine the dry uniform to see if we need to change the washer setup or not. What to do? Here is the first of two solutions to control hazards in the laundry room and its computer equivalent.
  Stall: Just operate sequentially until the first batch is dry and then repeat until you have the right formula.
- This conservative option certainly works, but it is slow.

Parallel Processors

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 in a easing 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 a.re 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.


Adder–subtractor
Integer multiply
Logic unit
Shift unit
Incrementer
Floating–point add–subtract
Floating–point multiply
Floating–point divide
To memory
Processor registers

- 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.

Flynn's Classification

- 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.

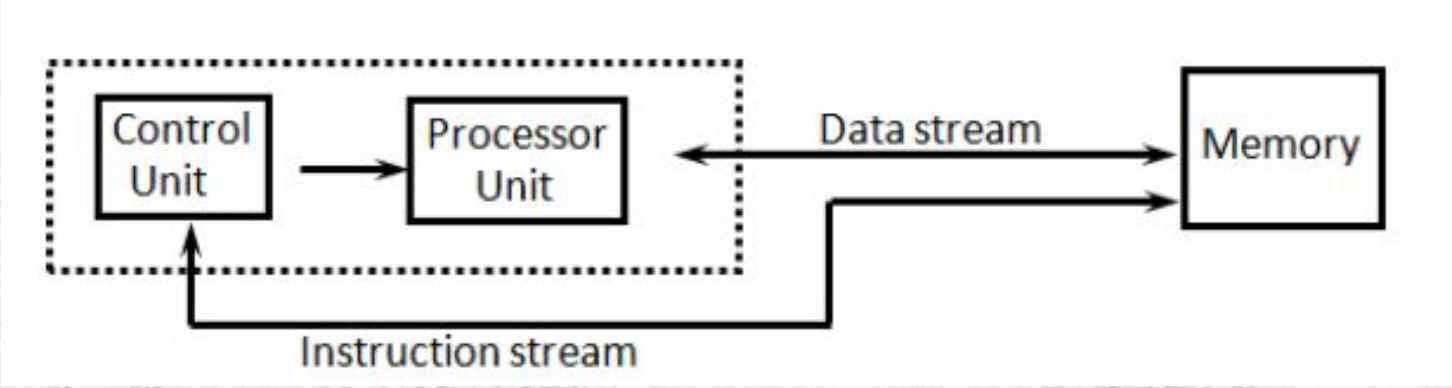
		Number of Data Streams	
		Single	**Multiple**
Number of Instruction Streams	**Single**	SISD	SIMD
	Multiple	MISD	MIMD

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

- 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)

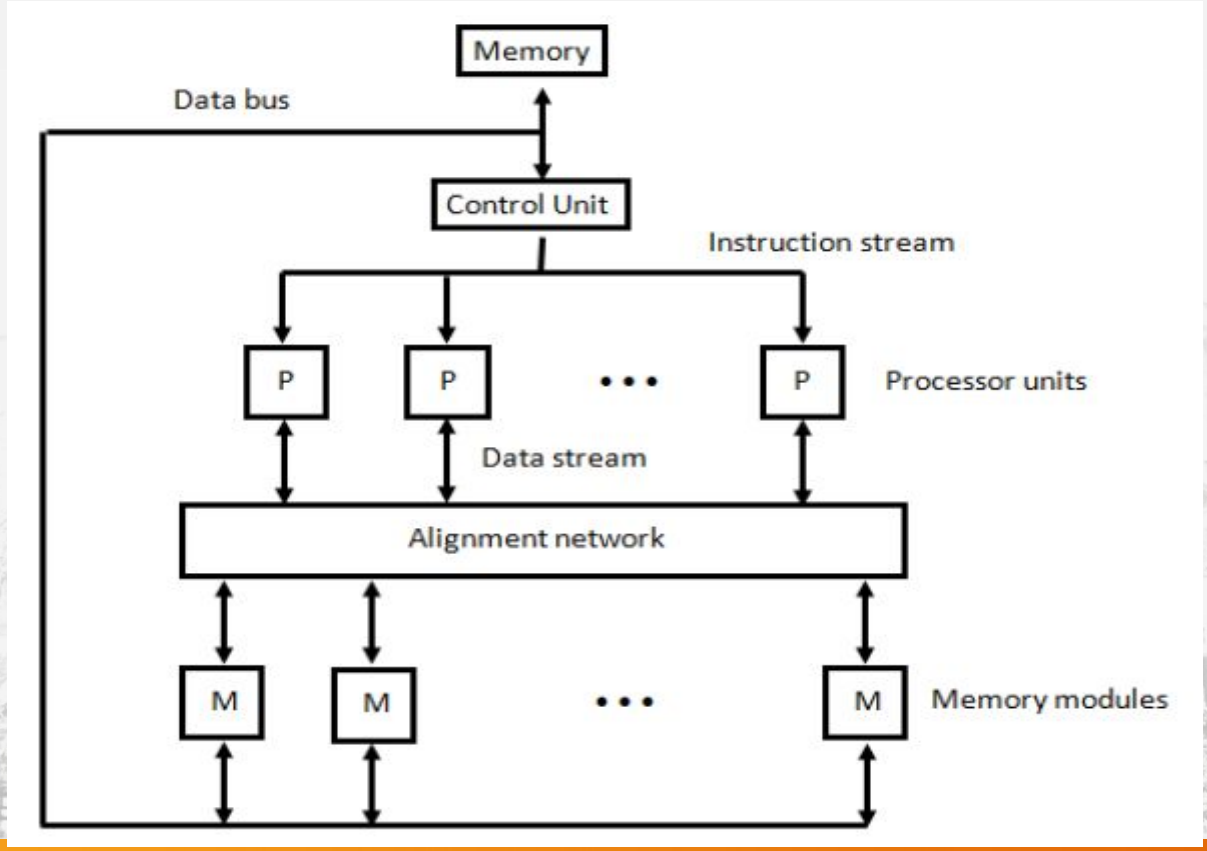
SISD

- 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: Single instruction stream, multiple data stream (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.



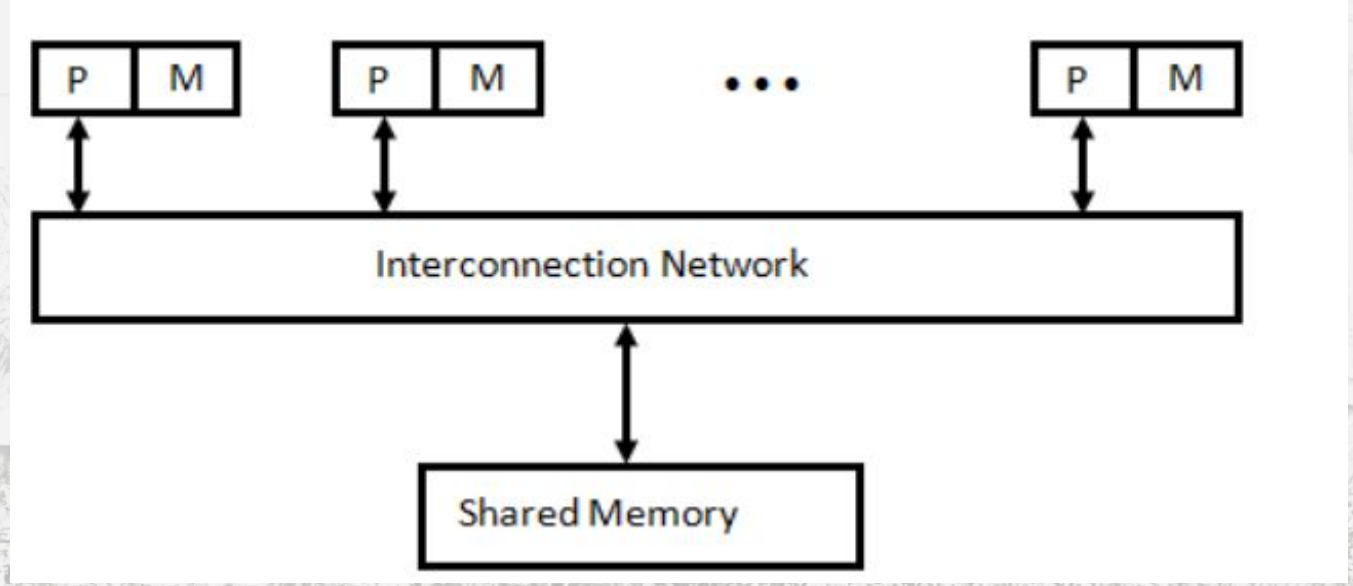
MISD: Multiple instruction stream, single data stream (MISD)

- 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: Multiple instruction stream, multiple data stream (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.

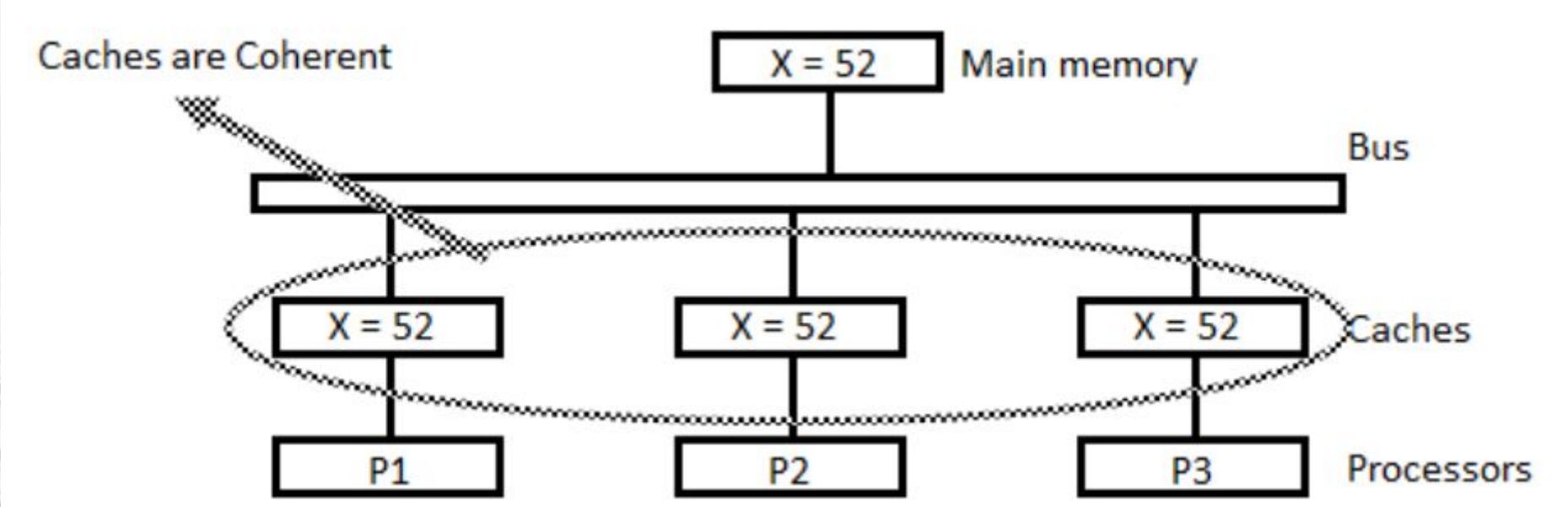
- This requirement imposes a cache coherence problem. A memory scheme is coherent if the value returned on a load instruction is always the value given by the latest store instruction with the same address.
- Without a proper solution to the cache coherence problem, caching cannot be used in bus oriented multiprocessors with two or more processors

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.

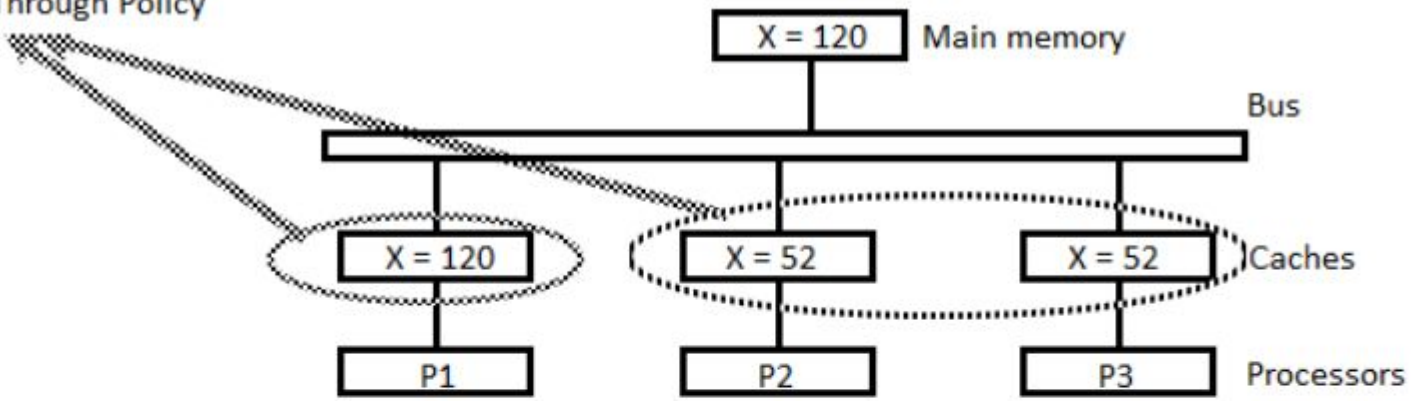
- Cache coherence problems exist in multiprocessors with private caches because of the need to share writable data.

- 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 figure

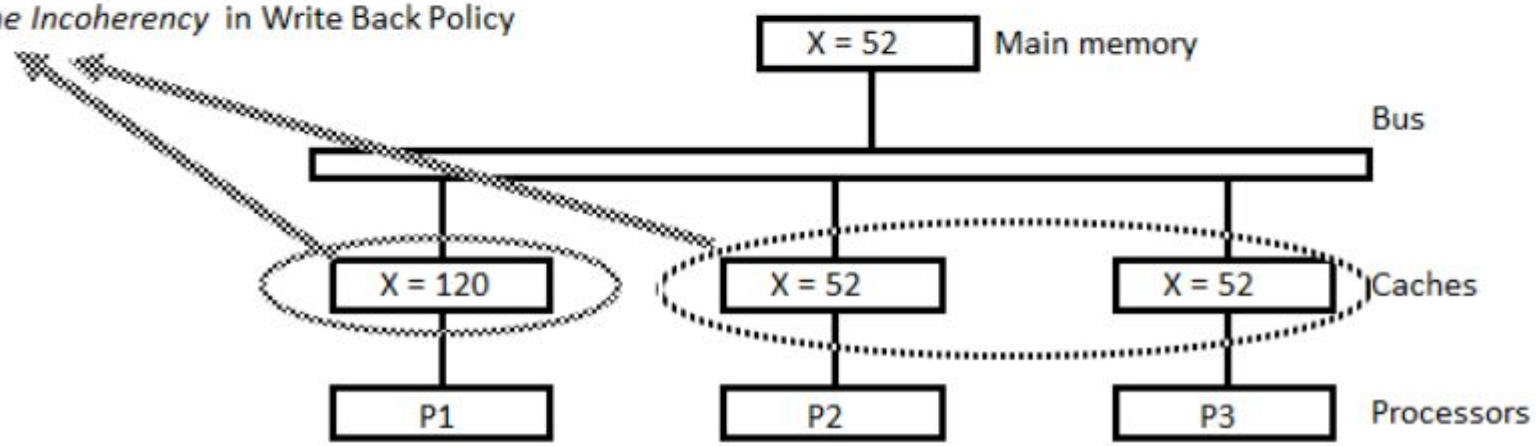


- During the operation an element X from main memory is loaded into the three processors, P1, P2, and P3.
- It is also copied into the private caches of the three processors.
- For simplicity, we assume that X= 52.
- The load on X to the three processors results in consistent copies in the caches and 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.

Cache Incoherency in Write Through Policy



Cache Incoherency in Write Back Policy

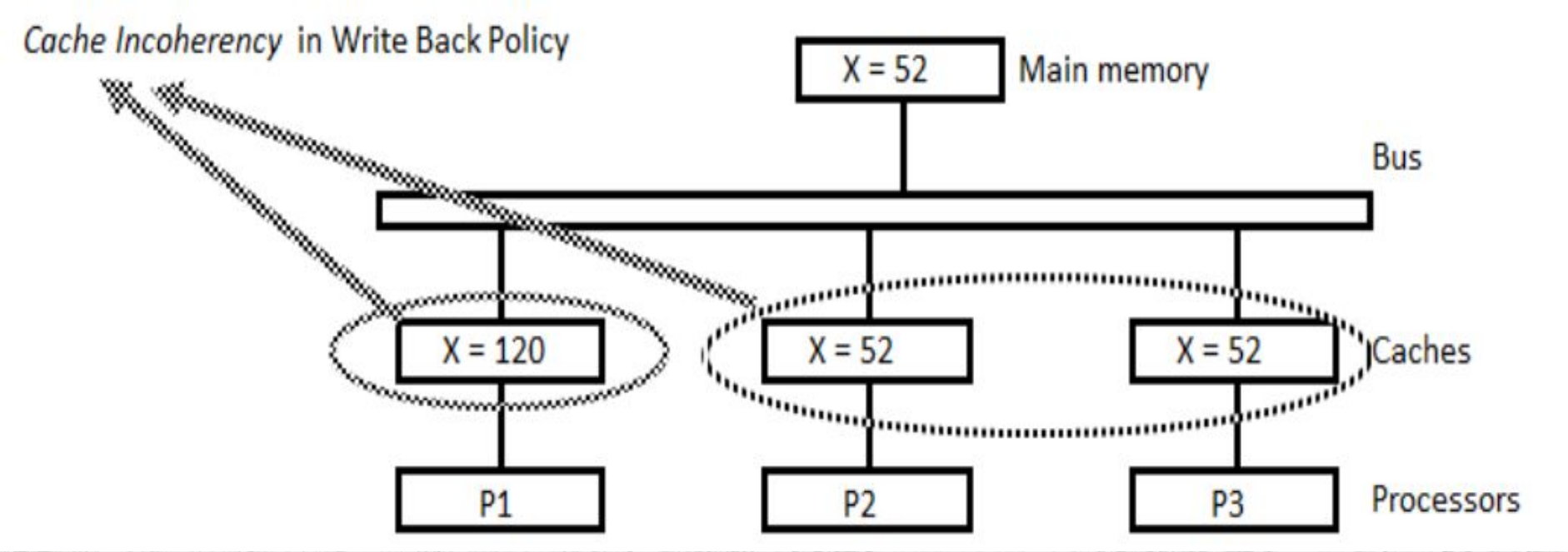


Write-through policy

- As shown in figure, 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.

Write-back policy

- 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


Cache Incoherency in Write Back Policy
X = 52
Main memory
Bus
X = 120
X = 52
X = 52
Caches
P1
P2
P3
Processors

Solution of cache coherence problem

- Various schemes have been proposed to solve the cache coherence problem in shared memory multiprocessors.
- Disallow private caches
  · A simple scheme is to disallow private caches for each processor and have a shared cache memory associated with main memory.
- Every data access is made to the shared cache.
- This method violates the principle of closeness of CPU to cache and increases the average memory access time.
- In effect, this scheme solves the problem by avoiding it

Software Approaches

- **Read-Only Data are Cacheable**
- The scheme that allows only nonshared and read-only data to be stored in caches. Such items are called cachable.
- Shared writable data are non cachable.
- The compiler must tag data as either cachable or noncachable, and the system hardware makes sure that only cachable data are stored in caches.
- The non cachable data remain in main memory.
- This method restricts the type of data stored in caches and introduces an extra software overhead that may degrades performance.

- **Centralized Global Table**
- A scheme that allows writable data to exist in at least one cache is a method that employs a centralized global table in it compiler.
- The status of memory blocks is stored in the central global table.
- Each block is identified as read-only (RO) or read and write (RW).
- All caches can have copies of blocks identified as RO.
- Only one cache can have a copy of an RW block.
- Thus if the data are updated in the cache with an RW block, the other caches are not affected because they do not have a copy of this block