UNIT - VI

PARALLEL PROCESSING

Objective:

- To familiarize the concept of pipelining and its efficiency.
- To gain knowledge of multiprocessors and their working.

Syllabus:

Parallel Processing, Pipelining, Arithmetic Pipeline, Instruction Pipeline.

Multi Processors: Characteristics of multiprocessors, interconnection structures, inter processor arbitration, cache coherence

Learning Outcomes:

At the end of the unit student will be able to:

1. Demonstrate the use of pipelining and multiprocessor.

Learning Material

6.1 PARALLEL PROCESSING

- Parallel Processing provide simultaneous data-processing tasks for the purpose of increasing the computational speed of a computer system.
- For example, while an instruction is being executed in the ALU, the next instruction can be read from memory. The system may have two or more ALUs and be able to execute two or more instructions at the same time.
- The purpose of parallel processing is to speed up the computer process-ing capability and increase its throughput.
- 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.

- The following figure 1 shows one possible way of separating the execution unit into eight functional units operating in parallel.

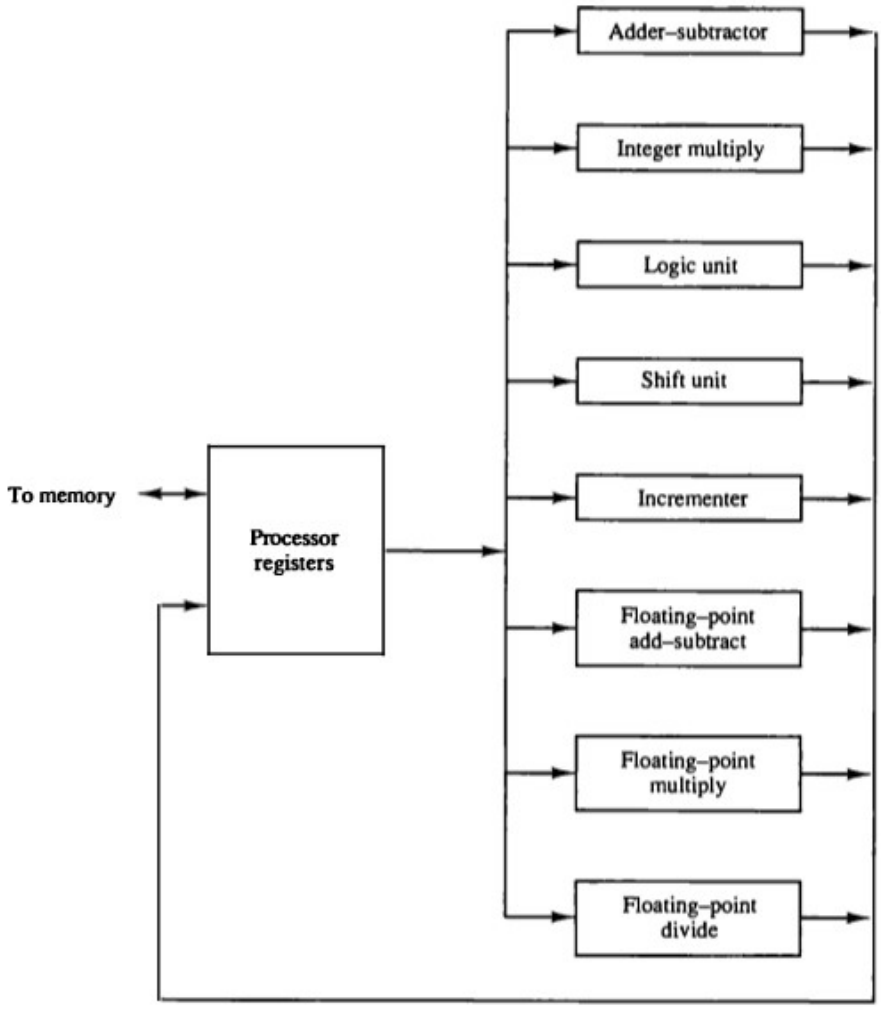

Figure 1: Processor with multiple functional units

- There are a variety of ways that parallel processing can be classified.
- Flynn's classification divides computers into four major groups as follows:
  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)

6.1.1 Single instruction stream, single data stream (SISD)

- 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 in this case may be achieved by means of multiple functional units or by pipeline processing.

Eg: C = A + B

6.1.2 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.
- The shared memory unit must contain multiple modules so that it can communicate with all the processors simultaneously.

Eg: C[i] = A[i] + B[i]

6.1.3 Multiple instruction stream, single data stream (MISD)

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

6.1.4 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.

Parallel processing can be achieved by the following different ways

1. Pipeline processing
2. Vector processing
3. Array processors

6.2 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.
- 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 final result is obtained after the data have passed through all segments.
- The name pipeline implies a flow of information analogous to an industrial assembly line.

- For example, suppose that we want to perform the combined multiply and add operations with a stream of numbers.

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

- Each sub-operation is to be implemented in a segment within a pipeline.
- Each segment has one or two registers and a combinational circuit as shown in Fig. 2.
- The sub-operations performed in each segment of the pipeline are as follows:

  $R1 \leftarrow A_i$, $R2 \leftarrow B_i$ Input A_i and B_i

  $R3 \leftarrow R1 * R2$, $R4 \leftarrow C_i$ Multiply and input C_i

  $R5 \leftarrow R3 + R4$ Add C_i to product

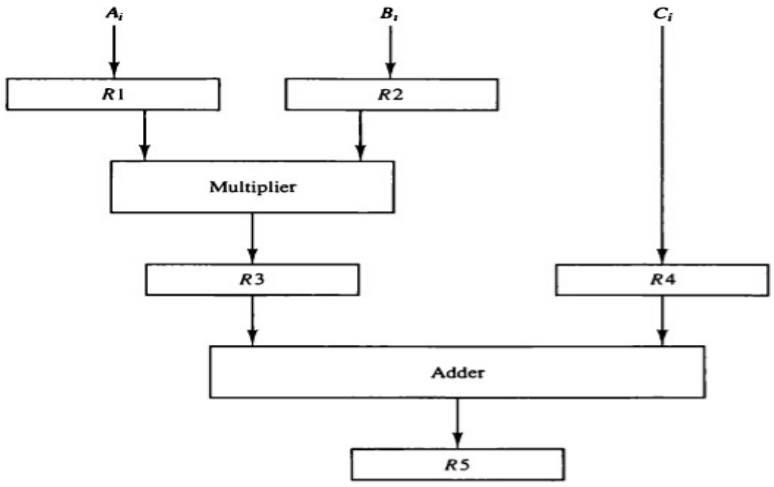

Figure 2: Example of pipeline processing

- The five registers are loaded with new data every clock pulse. The effect of each clock is shown in Table given below.

Table 1: Content of Registers in Pipeline Example

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	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	—	—	—	—	$A_7 * B_7 + C_7$

- The first clock pulse transfers A1 and B1 into R1 and R2.
- The second clock pulse transfers the product of R1 and R2 into R3 and C1 into R4. The same clock pulse transfers A2 and B2 into R 1 and R2.
- The third clock pulse operates on all three segments simultaneously. It places A, and B, into R1 and R2, transfers the product of R1 and R2 into R3, transfers C, into R4, and places the sum of R3 and R4 into R5. It takes three clock pulses to fill up the pipe and retrieve the first output from R5.
- From there on, each clock produces a new output and moves the data one step down the pipeline. This happens as long as new input data flow into the system. When no more input data are available, the clock must continue until the last output emerges out of the pipeline.
- The general structure of a four-segment pipeline is illustrated in Fig.3.
- The operands pass through all four segments in a fixed sequence. Each segment consists of a combinational circuit S_i that performs a sub-operation over the data stream flowing through the pipe.
- The segments are separated by registers R_i that hold the intermediate results between the stages.

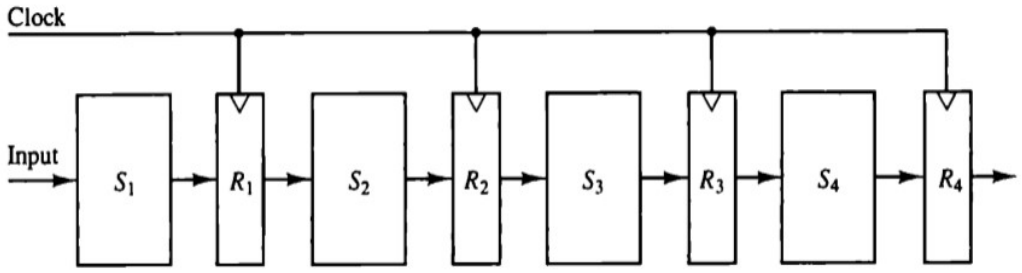

Figure 3: Four segment pipeline

- The behavior of a pipeline can be illustrated with a space-time diagram. This is a diagram that shows the segment utilization as a function of time.
- The space-time diagram of a four-segment pipeline is demonstrated in Fig. 4.

Segment:	1	2	3	4	5	6	7	8	9	→ Clock cycles
1	T_1	T_2	T_3	T_4	T_5	T_6				
2		T_1	T_2	T_3	T_4	T_5	T_6			
3			T_1	T_2	T_3	T_4	T_5	T_6		
4				T_1	T_2	T_3	T_4	T_5	T_6	

Figure 4: Space time diagram for pipeline

- The above figure 4 shows six tasks T1 through T6 executed in four segments.
- Initially, task T1 is handled by segment 1. After the first clock, segment 2 is busy with T1, while segment1 is busy with task T2. Continuing in this manner, the first task T1 is completed after the fourth clock cycle. From then on, the pipe completes a task every clock cycle.

- Consider the case 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 tokt_p to complete its operation since there are *k* segments in the pipe.
- The remaining *n-1* tasks emerge from the pipe at the rate of one task per clock cycle and they will be completed after a time equal to $(n-1)t_p$.
- Therefore, to complete *n* tasks using a k-segment pipeline requires $k + (n-1)$ clock cycles.
- For example, the diagram shows four segments and six tasks. The time required to complete all the operations is 4 + (6 - 1) = 9 clock cycles, as indicated in the diagram.
- Consider a non pipeline unit that performs the same operation and takes a time equal to tn. to complete each task. The total time required for n tasks is nt_n. 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 increases, *n* becomes much larger than $k - 1$, and $k + n - 1$ approaches the value of *n*. Under this condition, the speedup becomes

$$S = \frac{n\,t_n}{n\,t_p} \quad \Rightarrow \quad S = \frac{t_n}{t_p}$$

- If we assume that the time it takes to process a task is the same in the pipeline and non pipeline circuits, we will have $t_n = kt_p$. Including this assumption, the speedup reduces to:

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

- 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.
- To duplicate the theoretical speed advantage of a pipeline process by means of multiple functional units, it is necessary to construct k identical units that will be operating in parallel.
- In the following Fig. 5, where four identical circuits are connected in parallel.

- Each P circuit performs the same task of an equivalent pipeline circuit.
- Instead of operating with the input data in sequence as in a pipeline, the parallel circuits accept four input data items simultaneously and perform four tasks at the same time

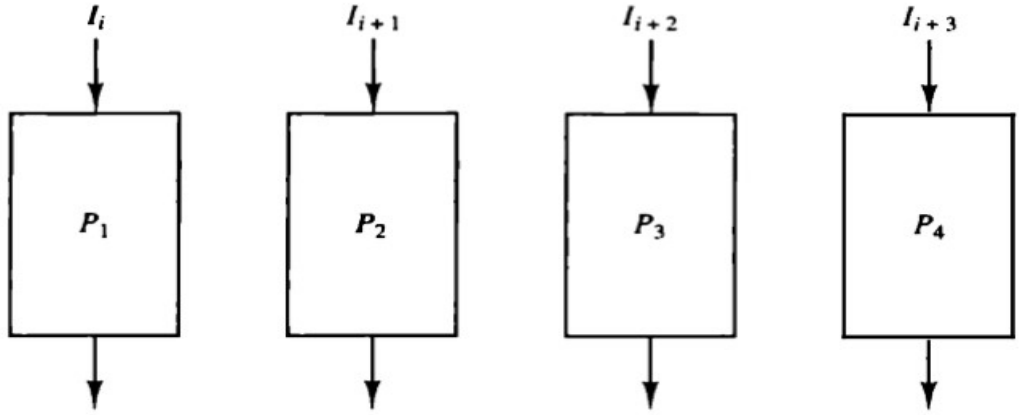

Figure 5: Multiple functional units in parallel

- In the above figure 5, four identical circuits are connected in parallel.
- The implication is that a k-segment pipeline processor can be expected to equal the performance of k copies of an equivalent non pipeline circuit under equal operating conditions.
- Each P circuit performs the same task of an equivalent pipeline circuit.
- Instead of operating with the input data in sequence as in a pipeline, the parallel circuits accept four input data items simultaneously and perform four tasks at the same time.

There are two areas of computer design where the pipeline organization is applicable.

1. Arithmetic pipeline
2. Instruction pipeline

6.2.1. Arithmetic pipeline

- Pipeline arithmetic units are usually found in very high speed computers.
- They are used to implement floating-point operations, multiplication of fixed-point numbers, and similar computations encountered in scientific problems.
- The inputs to the floating-point adder pipeline are two normalized floating-point binary numbers.

$$X = A \times 2^a$$
$$Y = B \times 2^b$$

- The floating-point addition and subtraction can be performed in four segments, as shown in Figure 6.

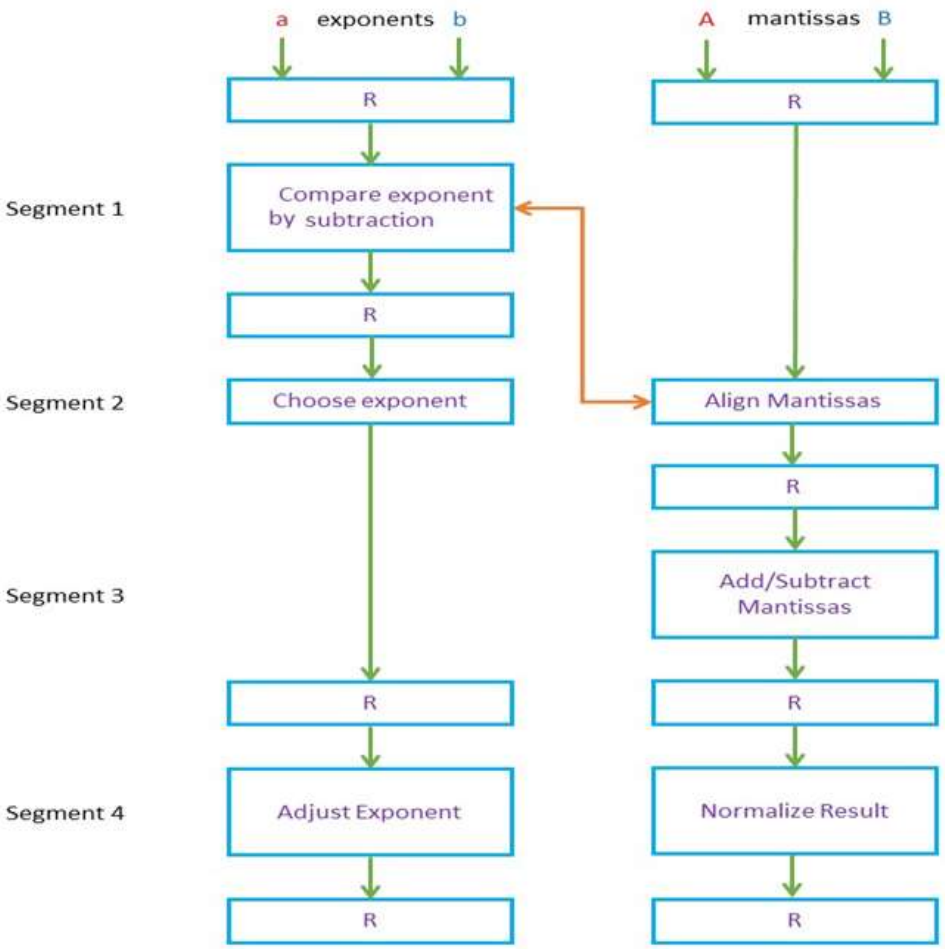

Figure 6: Pipeline for floating point addition and subtraction operation

- The registers labeled R are placed between the segments to store intermediate results.
- 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.
- The exponents are compared by subtracting them to determine their difference. The larger exponent is chosen as the exponent of the result.
- The exponent difference determines how many times the mantissa associated with the smaller exponent must be shifted to the right. This produces an alignment of the two mantissas.
- The two mantissas are added or subtracted in segment 3.

- The result is normalized in segment 4. When an overflow occurs, the mantissa of the sum or difference is shifted right and the exponent incremented by one. If an underflow occurs, the number of leading zeros in the mantissa determines the number of left shifts in the mantissa and the number that must be subtracted from the exponent.

6.2.2. Instruction pipeline

- An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments.
- 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.
- The design of an instruction pipeline will be most efficient if the instruction cycle is divided into segments of equal duration.
- The time that each step takes to fulfill its function depends on the instruction and the way it is executed.
- Example: Four-Segment Instruction Pipeline.
- Figure 7 shows how the instruction cycle in the CPU can be processed with a four-segment pipeline.

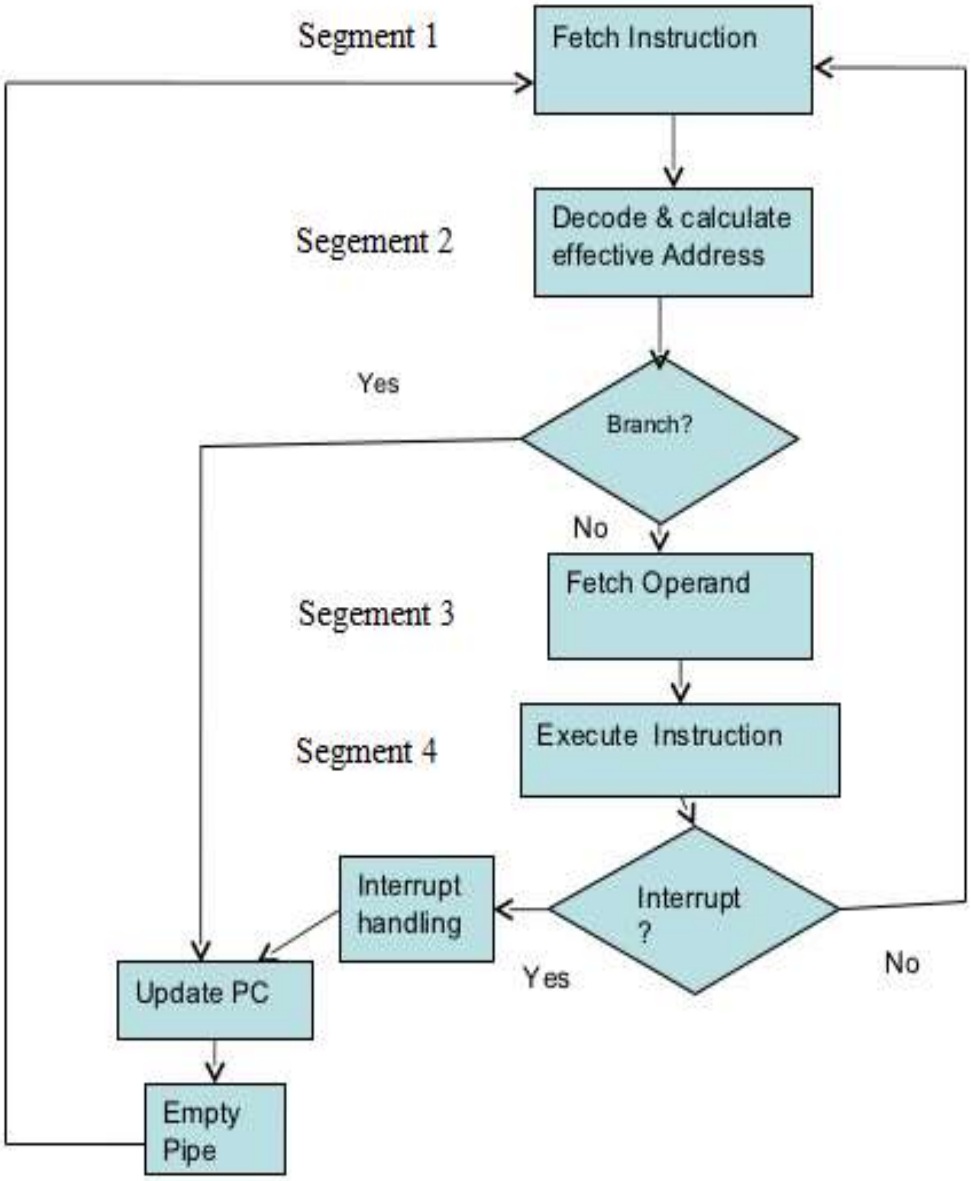

Figure 7: Four segment CPU Pipeline

- The following figure 8 shows the operation of the instruction pipeline.

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

Figure 8: Timing of instruction pipeline

- The time in the horizontal axis is divided into steps of equal duration.
- The four segments are represented in the diagram with an abbreviated symbol.
  1. FI is the segment that fetches an instruction.
  2. DA is the segment that decodes the instruction and calculates the effective address.
  3. FO is the segment that fetches the operand.
  4. EX is the segment that executes the instruction.
- It is assumed that the processor has separate instruction and data memories so that the operation in FI and FO can proceed at the same time.
- In the absence of a branch instruction, each segment operates on different instructions. Thus, in step 4, instruction 1 is being executed in segment EX; the operand for instruction 2 is being fetched in segment FO; instruction 3 is being decoded in segment DA; and instruction 4 is being fetched from memory in segment FL.
- Assume now that instruction 3 is a branch instruction. As soon as this instruction is decoded in segment DA in step 4, the transfer from FI to DA of the other instructions is halted until the branch instruction is executed in step 6. If the branch is taken, a new instruction is fetched in step 7. If the branch is not taken, the instruction fetched previously in step 4 can be used. The pipeline then continues until a new branch instruction is encountered.
- A delay may occur in the pipeline if the EX segment needs to store the result of the operation in the data memory while the FO segment needs to fetch an operand.
- In that case, segment FO must wait until segment EX has finished its operation.

6.3 CHARACTERISTICS OF MULTIPROCESSORS

- A multiprocessor system is an interconnection of two or more CPUs with memory and input-output equipment.
- The term "processor" In multiprocessor can mean either a central processing unit (CPU) or an input-output processor (lOP).
- However, a system with a single CPU and one or more lOPs is usually not included in the definition of a multiprocessor system unless the lOP has computational facilities comparable to a CPU.
- As it is most commonly defined, a multiprocessor system implies the existence of multiple CPUs, although usually there will be one or more lOPs.
- Multiprocessors are classified as Multiple Instruction stream, Multiple Data stream (MIMD) systems.
- There are some similarities between multiprocessor and multicomputer systems since both support concurrent operations.
- However, there exists an important distinction between a system with multiple computers and a system with multiple processors.
- Computers are interconnected with each other by means of communication lines to form a computer network.
- A multiprocessor system is controlled by one operating system that provides interaction between processors and all the components of the system cooperate in the solution of a problem.
- Multiprocessing improves the reliability of the system so that a failure or error in one part has a limited effect on the rest of the system.
- If a fault causes one processor to fail, a second processor can be assigned to perform the functions of the disabled processor.
- Multiprocessors are classified by the way their memory is organized as follows.
  1. Tightly coupled
  2. loosely coupled

6.3.1 Tightly coupled

- A multiprocessor system with common shared memory is classified as a shared memory or tightly coupled multiprocessor.

6.3.2 Loosely coupled

- An alternative model of microprocessor is the distributed-memory or loosely coupled system.
- Here each processor element in a loosely coupled system has its own private local memory.

6.4 INTERCONNECTION STRUCTURES

- There are several physical forms available for establishing an interconnection network. Some of these schemes follows:
  1. Time-shared common bus
  2. Multiport memory
  3. Crossbar switch

6.4.1. Time-shared common bus

- A common-bus multiprocessor system consists of a number of processors connected through a common path to a memory unit.
- Only one processor can communicate with the memory or another processor at any given time.

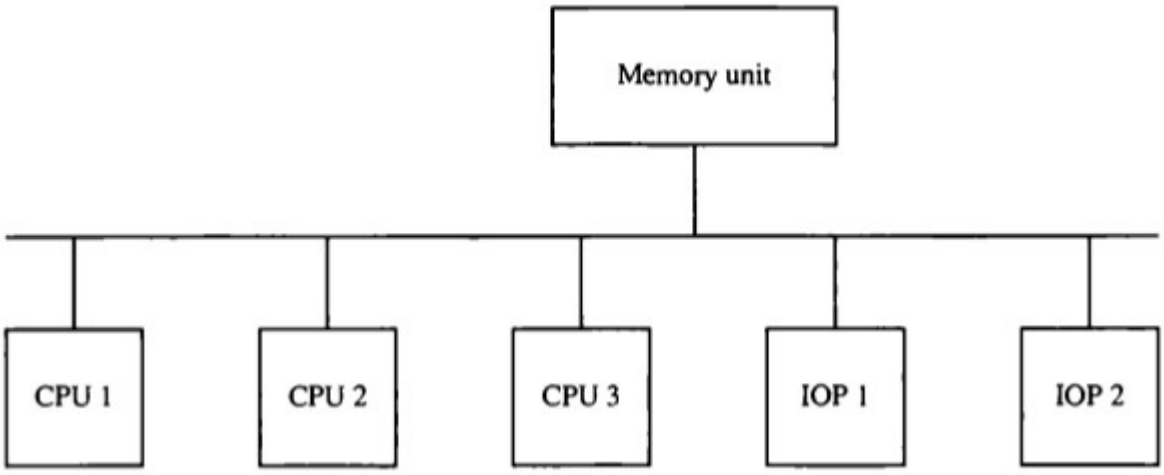

Figure 9: Time shared common bus organization

- Any processor wishing to initiate a transfer must first determine the availability status of the bus, and only after the bus becomes available can the processor address the destination unit to initiate the transfer.
- A command is issued to inform the destination unit what operation is to be performed. The receiving unit recognizes its address in the bus and responds to the control signals from the sender, after which the transfer is initiated.
- The system may exhibit transfer conflicts since one common bus is shared by all processors. These conflicts must be resolved by incorporating a bus controller that establishes priorities among the requesting units.
- A single common-bus system is restricted to one transfer at a time.
- The total overall transfer rate within the system is limited by the speed of the single path.

- Dual bus structure is depicted in Fig.10. Here we have a number of local buses each connected to its own local memory and to one or more processors.
- Each local bus may be connected to a CPU, anlOP, or any combination of processors.
- A system bus controller links each local bus to a common system bus.
- The I/O devices connected to the local lOP, as well as the local memory, are available to the local processor.

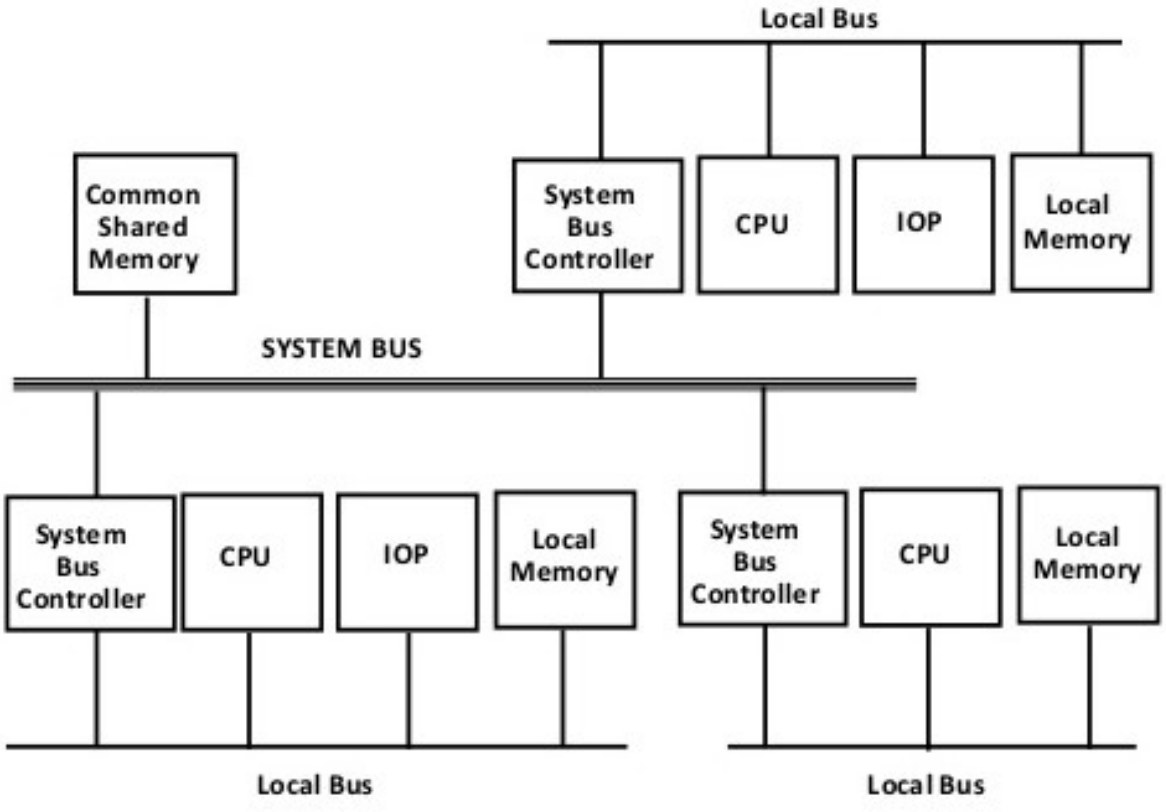

Figure 10: System bus structure for multiprocessors

- The memory connected to the common system bus is shared by all processors.
- If an lOP is connected directly to the system bus, the I/O devices attached to it may be made available to all processors.
- Only one processor can communicate with the shared memory and other common resources through the system bus at any given time.
- In this way, the average access time of the local memory can be made to approach the cycle time of the CPU to which it is attached.

6.4.2. Multiport Memory

- A multiport memory system employs separate buses between each memory module and each CPU.
- This is shown in Fig 11, for four CPUs and four memory modules (MMs).
- Each processor bus is connected to each memory module.
- A processor bus consists of the address, data, and control lines required to communicate with memory.
- The memory module is said to have four ports and each port accommodates one of the buses.

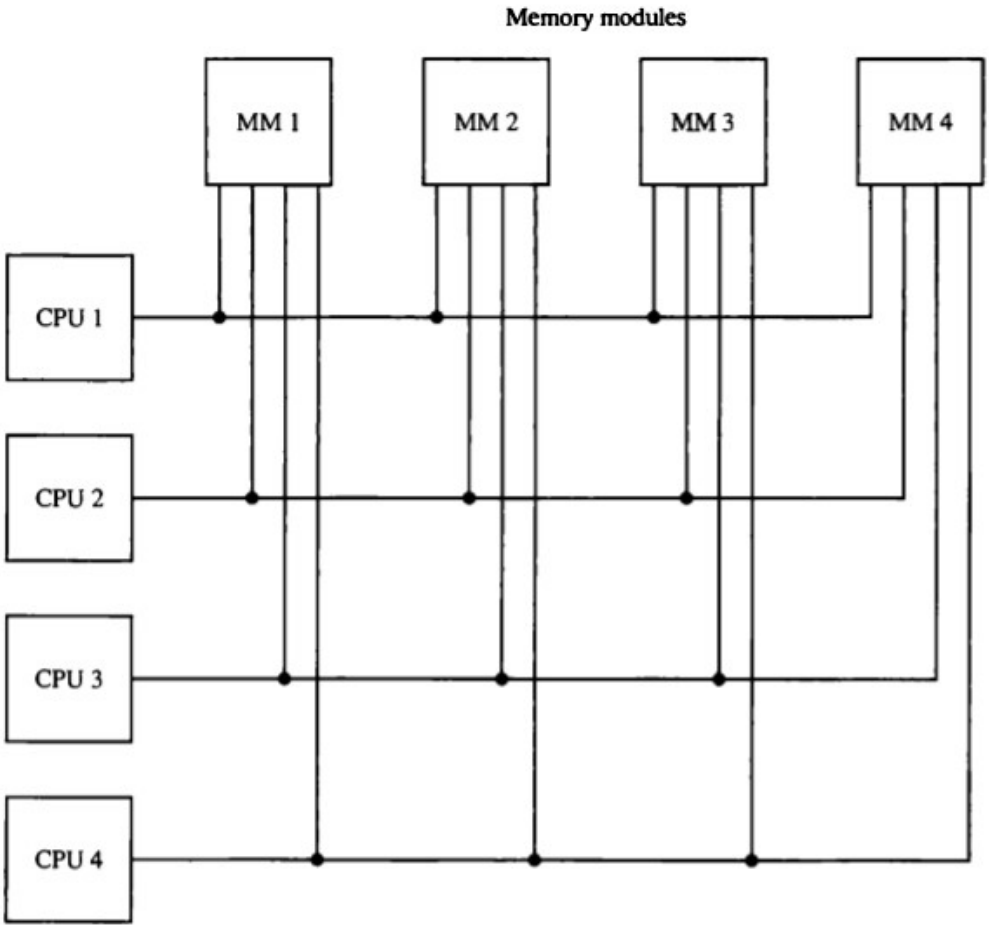

Figure 11: Multiport memory organization

- Memory access conflicts are resolved by assigning fixed priorities to each memory port.
- The priority for memory access associated with each processor may be established by the physical port position that its bus occupies in each module.
- Thus CPU 1 will have priority over CPU 2, CPU 2 will have priority over CPU 3, and CPU 4 will have the lowest priority.

6.4.3. Crossbar Switch

- The crossbar switch organization consists of a number of cross-points that are placed at intersections between processor buses and memory module paths.
- Figure 12 shows a crossbar switch interconnection between four CPUs and four memory modules.
- The small square in each cross-point is a switch that determines the path from a processor to a memory module.

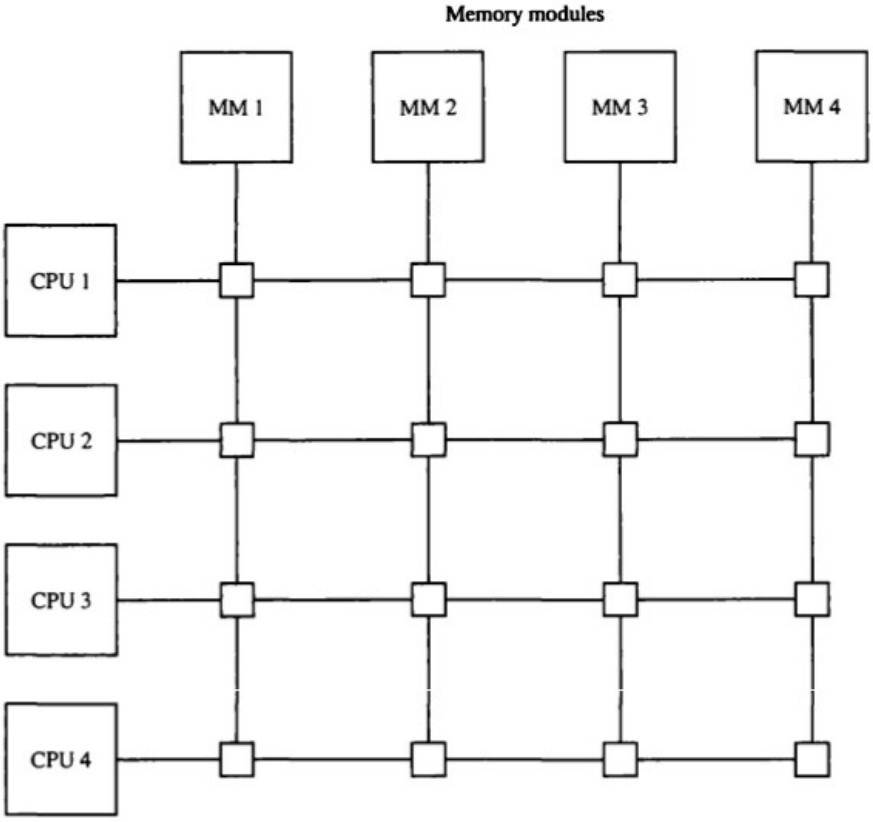

Figure 12: Crossbar switch

- Each switch point has control logic to set up the transfer path between a processor and memory.
- Switch point examines the address that is placed in the bus to determine whether its particular module is being addressed. It also resolves multiple requests for access to the same memory module on a predetermined priority basis.
- The following figure 13 shows the functional design of a crossbar switch connected to one memory module. The circuit consists of multiplexers that select the data, address, and control from one CPU for communication with the memory module.

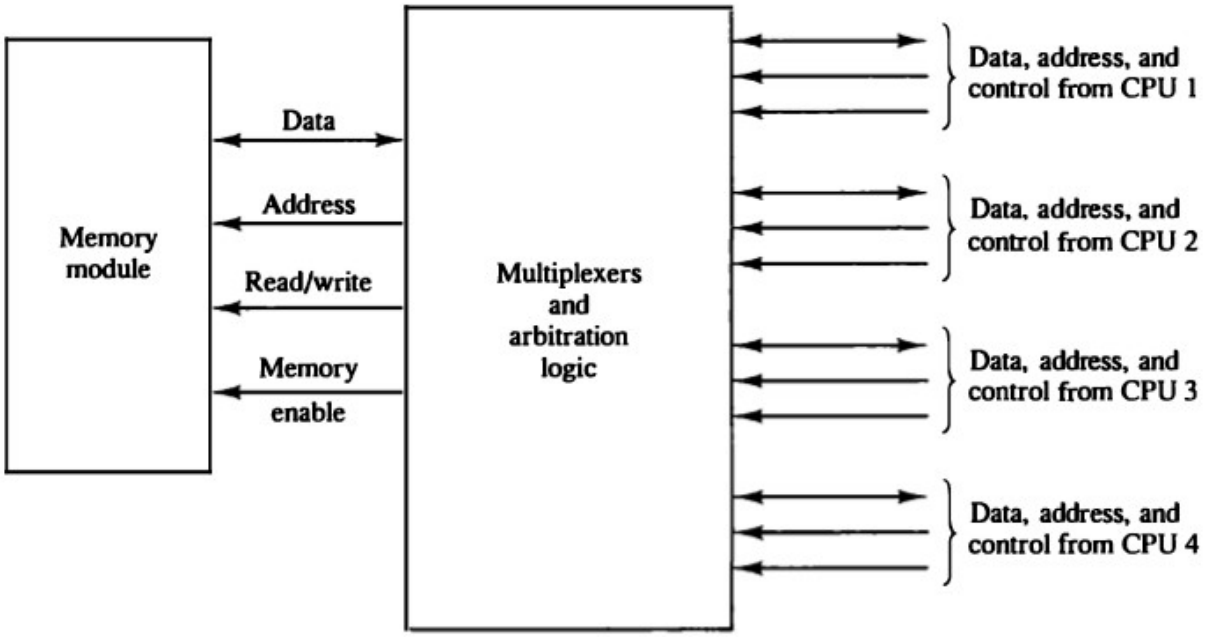

Figure 13: Block diagram of crossbar switch

- Priority levels are established by the arbitration logic to select one CPU when two or more CPUs attempt to access the same memory.
- A crossbar switch organization supports simultaneous transfers from memory modules because there is a separate path associated with each module.

6.5 INTERPROCESSOR ARBITRATION

- A bus that connects major components in a multiprocessor system, such as CPUs, lOPs, and memory, is called a system bus.
- The processors in a shared memory multiprocessor system request access to common memory or other common resources through the system bus.
- If no other processor is currently utilizing the bus, the requesting processor may be granted access immediately.
- However, the requesting processor must wait if another processor is currently utilizing the system bus. Furthermore, other processors may request the system bus at the same time.
- Arbitration must then be performed to resolve this multiple contention for the shared resources.
- Arbitration must then be performed to resolve this multiple contention for the shared resources.
- Arbitration procedures service all processor requests on the basis of established priorities.
- A hardware bus priority resolving technique can be established by means of following ways.

1. Static Arbitration
    i. Serial Arbitration Procedure
    ii. Parallel Arbitration Logic
2. Dynamic Arbitration

6.5.1.1 Serial Arbitration Procedure

- The serial priority resolving technique is obtained from a daisy-chain connection of bus arbitration circuits similar to the priority interrupt logic.
- The processors connected to the system bus are assigned priority according to their position along the priority control line.
- When multiple devices concurrently request the use of the bus, the device with the highest priority is granted access to it.

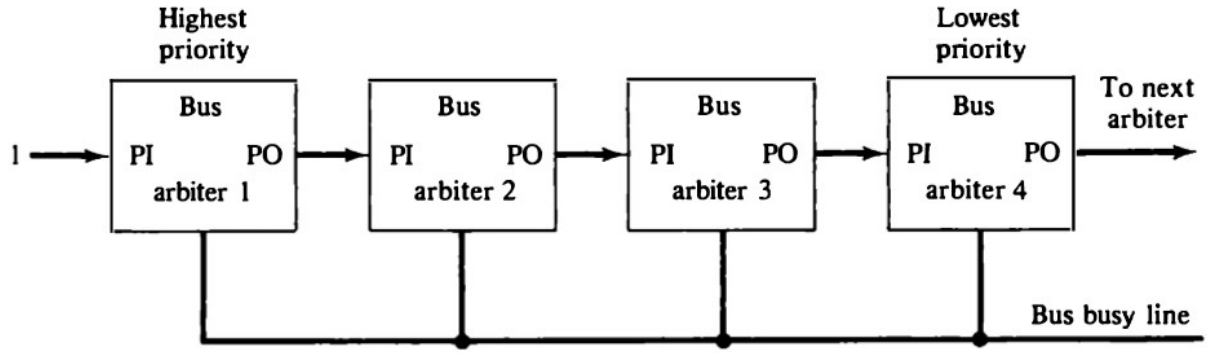

Figure 14: Serial (Daisy-chain) arbitration

- A processor may be in the middle of a bus operation when a higher priority processor requests the bus. The lower-priority processor must complete its bus operation before it hand over control of the bus.
- The bus busy line shown in above figure 14 provides a mechanism for an orderly transfer of control.

6.5.1.2 Parallel Arbitration Logic

- The parallel bus arbitration technique uses an external priority encoder and a decoder as shown in following figure 15.
- Each bus arbiter in the parallel scheme has a bus request output line and a bus acknowledge input line.
- Each arbiter enables the request line when its processor is requesting access to the system bus.
- The processor takes control of the bus if its acknowledge input line is enabled. The bus busy line provides an orderly transfer of control, as in the daisy-chaining case.

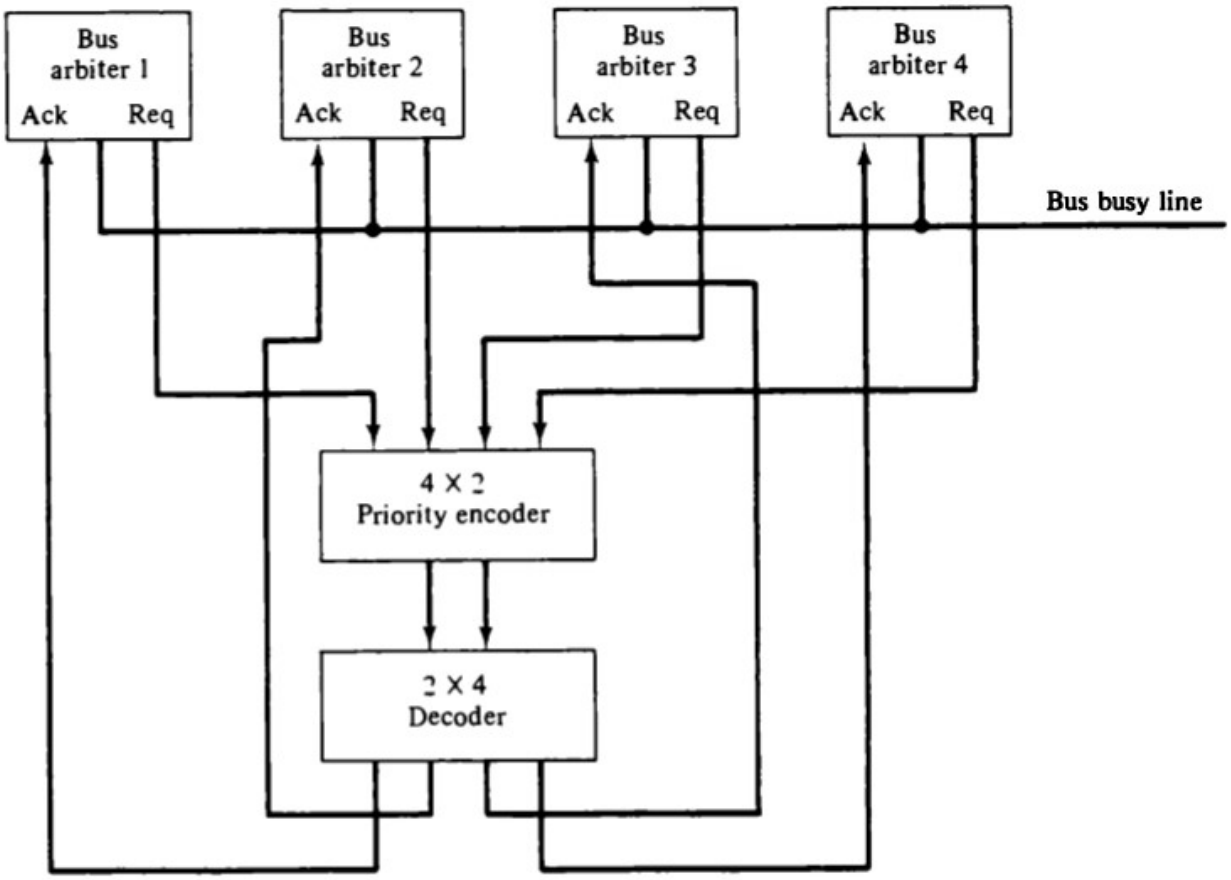

Figure 15: Parallel arbitration

- The request lines from four arbiters going into a 4x2 priority encoder.
- The output of the encoder generates a 2-bit code which represents the highest-priority unit among those requesting the bus.
- The 2-bit code from the encoder output drives a 2x4 decoder which enables the proper acknowledge line to grant bus access to the highest-priority unit.

6.5.2 Dynamic Arbitration

- The two bus arbitration procedures just described use a static priority algorithm since the priority of each device is fixed by the way it is connected to the bus.
- Dynamic priority algorithm gives the system the capability for changing the priority of the devices while the system is in operation.
- A few arbitration procedures that use dynamic priority algorithms:

  1. Time slice
  2. Polling
  3. LRU
  4. FIFO

5. Rotating daisy-chain

6.5.2.1 Time slice

- The time slice algorithm allocates a fixed-length time slice of bus time that is offered sequentially to each processor, in round-robin fashion.

6.5.2.2 Polling

- In polling, the bus grant signal is replaced by a set of lines called poll lines which are connected to all units.
- These lines are used by the bus controller to define an address for each device connected to the bus.
- When a processor that requires access recognizes its address, it activates the bus busy line and then accesses the bus

6.5.2.3 LRU

- The least recently used (LRU) algorithm gives the highest priority to the requesting device that has not used the bus for the longest interval.

6.5.2.4 FIFO

- In the first-come, first-serve (FIFO) scheme, requests are served in the order received

6.5.2.5 Rotating daisy-chain

- The rotating daisy-chain procedure is a dynamic extension of the daisy-chain algorithm.
- In this scheme there is no central bus controller, and the priority line is connected from the priority-out of the last device back to the priority-in of the first device in a closed loop.

6.6 CACHE COHERENCE

- If the operation is to write, there are two commonly used procedures to update memory.
- In the *write-through* policy, both cache and main memory are updated with every write operation.
- 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 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.

6.6.1 Conditions for Incoherence

- Cache coherence problems exist in multiprocessors with private caches because of the need to share writable data.
- Consider the three-processor configuration with private caches shown in figure 16.
- 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.

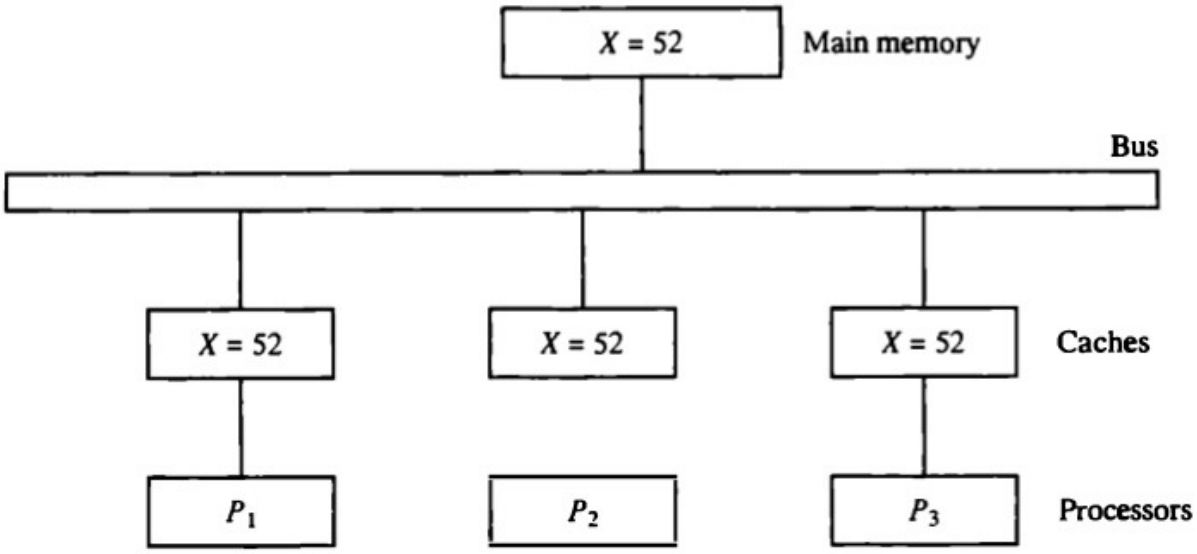

Figure 16: Cache configuration after a load on X

- 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 as shown in the following figure 17.

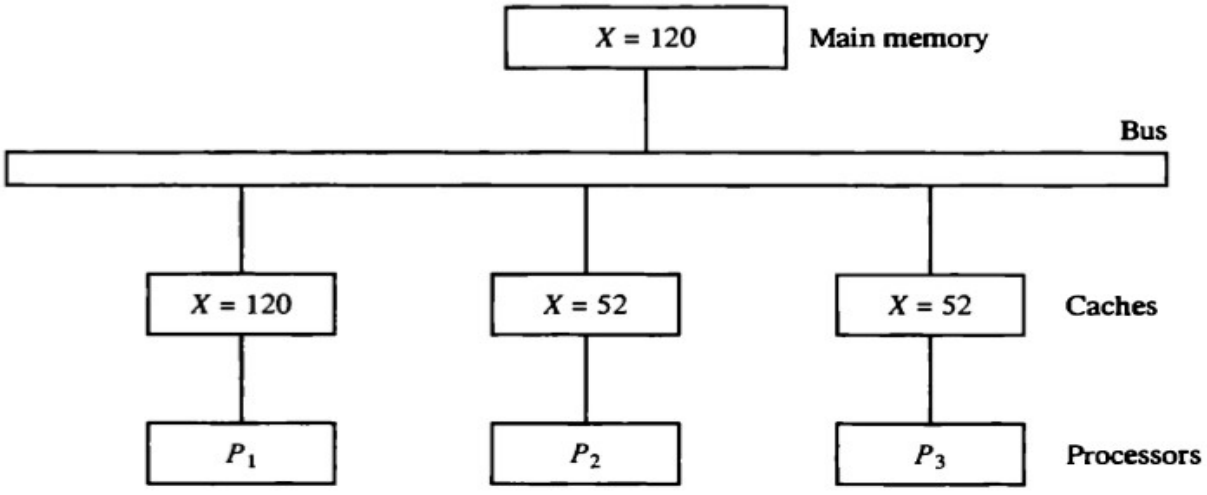

Figure 17: Cache configuration after a store to X by processor P1 With write-through cache policy

- 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 main 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.

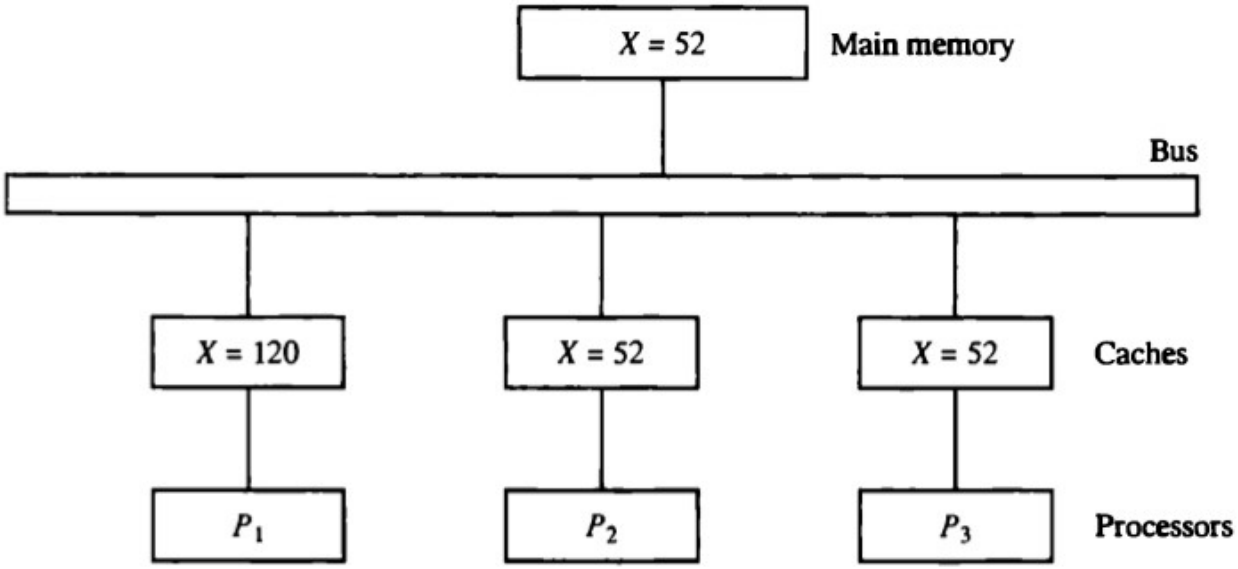

Figure 18: Cache configuration after a store to X by processor P1 With write-back cache policy

- Another configuration that may cause consistency problems is a direct memory access (DMA) activity in conjunction with an lOP 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.

6.6.2 Solutions to the Cache Coherence Problem

1. A simple scheme is to disallow private caches for each processor and have a shared cache memory associated with main memory. For performance considerations it is required to attach a private cache to each processor.
2. One scheme that has been used allows only non-shared 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.

3. A scheme that allows writable data to exist in at least one cache is a method that employs a centralized global table in its 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 affect because they do not have a copy of this block.

   The cache coherence problem can be solved by means of a combination of software and hardware or by means of hardware-only schemes. The two methods mentioned previously use software based procedures.
4. In the hardware solution, the cache controller is specially designed to allow it to monitor all bus requests from CPUs and lOPs. All caches attached to the bus constantly monitor the network for possible write operations. The bus controller that monitors this action is referred to as a snoopy cache controller. This is basically a hardware unit designed to maintain a bus-watching mechanism over all the caches attached to the bus. Various schemes have been proposed to solve the cache coherence prob-lem by means of snoopy cache protocol.
   - When a word in a cache is updated by writing into it, the corresponding location in main memory is also updated. The local snoopy controllers in all other caches check their memory to determine if they have a copy of the word that has been overwritten.
   - If a copy exists in a remote cache, that location is marked invalid. Because all caches snoop on all bus writes, whenever a word is written, the net effect is to update it in the original cache and main memory and remove it from all other caches.
   - If at some future time a processor accesses the invalid Item from its cache, the response is equivalent to a cache miss, and the updated Item is transferred from main memory. In this way, inconsistent versions are prevented.