Chapter 11
Multiprocessor

11.1 Introduction and Characteristics of Multiprocessor

→ The system in which two or more processing units (CPU or IOP) are connected to the memory and I/O devices is known as multiprocessor system.

→ Multiprocessors are classified as multiple instruction stream, multiple data stream (MIMD) systems.

→ A multiprocessor system is controlled by one operating system that provides interaction between processors and all the components of the system.

Characteristics of multiprocessors

- Multiple processing elements
- System is controlled by single operating system
- System is more reliable

The system derives its high performance from the fact that computations can proceed in parallel in one of two ways:

- Multiple independent jobs can be made to operate in parallel.
- A single job can be partitioned into multiple parallel tasks.

Types of multiprocessor

→ Multiprocessors are classified by the way their memory is organized. They are:

→ **Shared memory or tightly-coupled multiprocessor:** the multiprocessor system in which all processing elements share a common memory. In this type, there is no local memory with processor but they have their own cache memory.

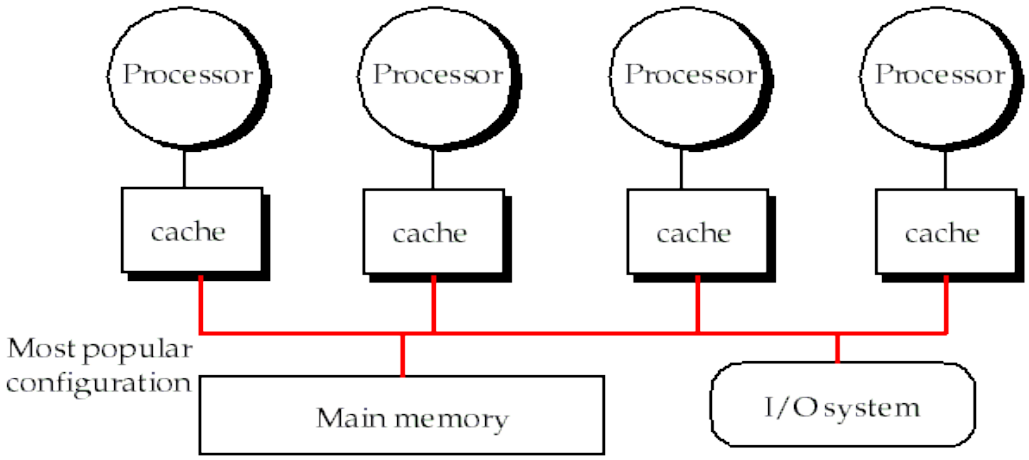

Fig: Shared Memory Architecture

→ **Distributed memory or loosely-coupled multiprocessor:** the multiprocessor system in which each processing element has its own private local memory and all the processors are tied together by a switching mechanism to route information from one processor to another through a message passing scheme.

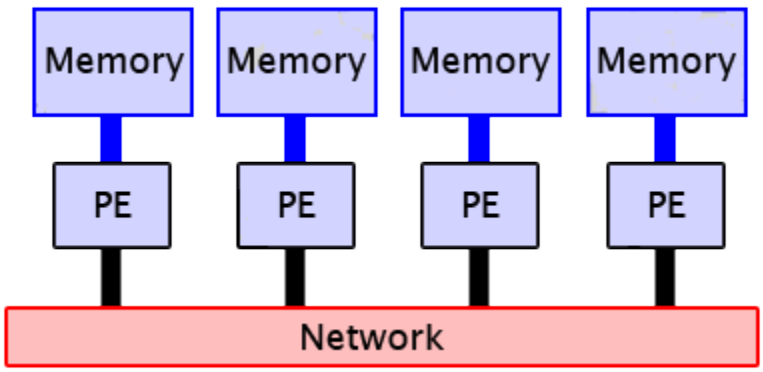

Fig: Distributed Memory Architecture

11.2 Interconnection Structures

→ The components that form a multiprocessor system are CPUs, IOPs connected to I/O devices, and a memory unit that may be partitioned into a number of separate modules.

→ The interconnection between the components can have different physical configurations, depending on the number of transfer paths that are available between the processors and memory in a shared memory system or among the processing elements in a loosely coupled system.

→ There are several interconnection networks. Some of these schemes are given as:

1. Time shared common bus
2. Multiport memory
3. Crossbar switch
4. Multistage switching network
5. Hypercube system

1. Time-shared common bus

→ A common bus is used for all CPU to communicate with shared memory. At any given time, only one processor can communicate with the memory or another processor but all the processors are either busy with their internal operation or idle waiting for the bus.

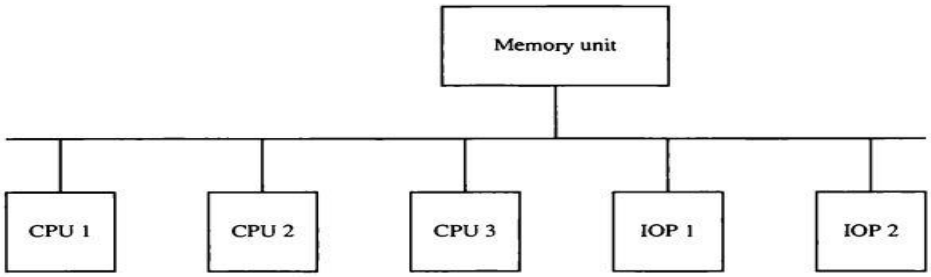

Fig: Time-shared Common Bus Organization.

→ Advantages:

- Simplicity
- Flexibility
- Reliability

→ Disadvantages

- Performance limited by bus cycle time
- Each processor should have local cache
- Reduce number of bus accesses
- Leads to problems with cache coherence

2. Multiport Memory

→ It uses separate buses between each memory module and each CPU.

→ Each processor has direct independent access of memory modules by their own bus connected to each module.

→ The modules must have internal control logic to determine which port will have access to memory at any given time.

→ Memory access conflicts are resolved by assigning fixed priorities to each memory port. CPU1 will have priority over CPU2, CPU2 will have priority over CPU3 and CPU4 will have the lowest priority.

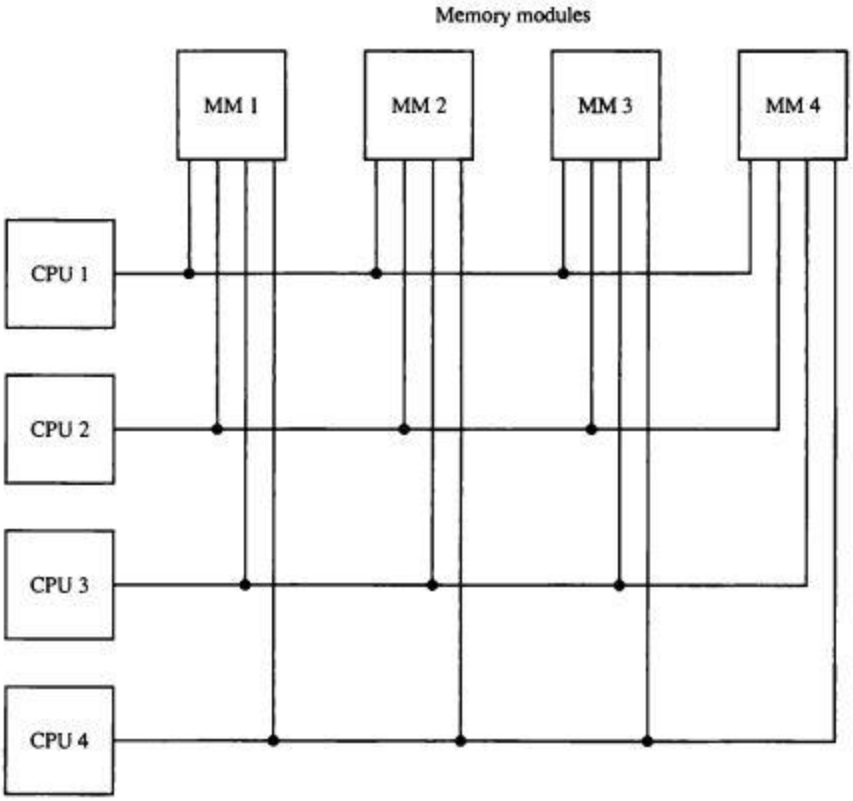

Fig: Multiport memory organization.

→ Advantages:
- Better performance because each processor has dedicated path to each module.
- Can configure portions of memory as private to one or more processors so increased security.

→ Disadvantages:
- Requires extra control logic so more complex and increase the cost.
- Requires large connection wires.

3. Crossbar Switch

→ A crossbar switch (also known as cross-point switch or matrix switch) is a switch connecting multiple inputs to multiple outputs in a matrix manner.

→ The crossbar switch organization consists of a number of cross points that are placed at interconnection between processor bus and memory module path.

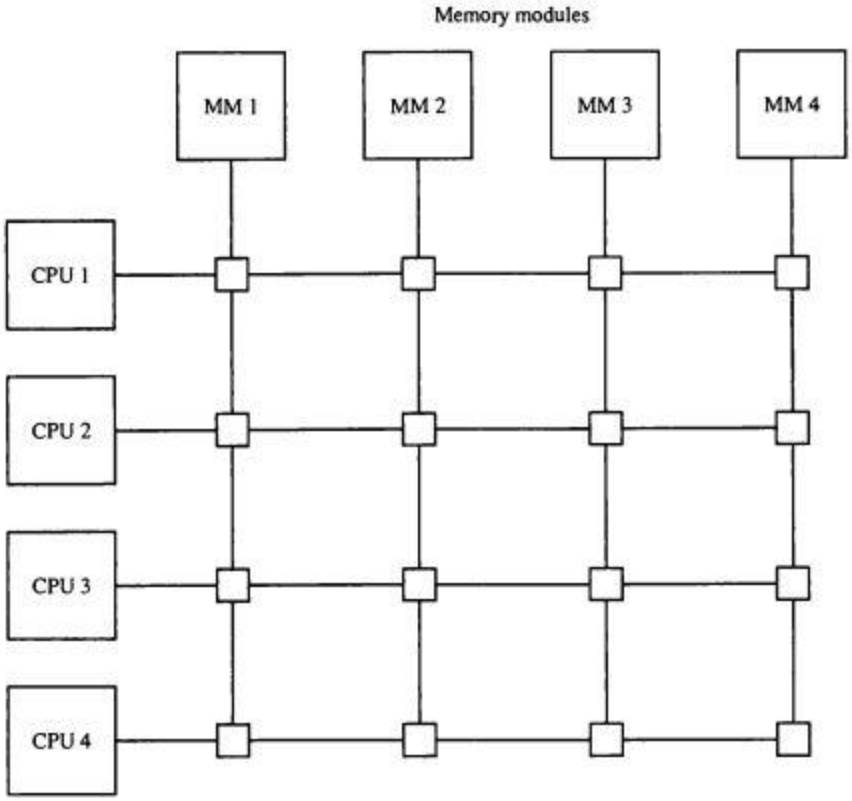

Fig: Crossbar switch.

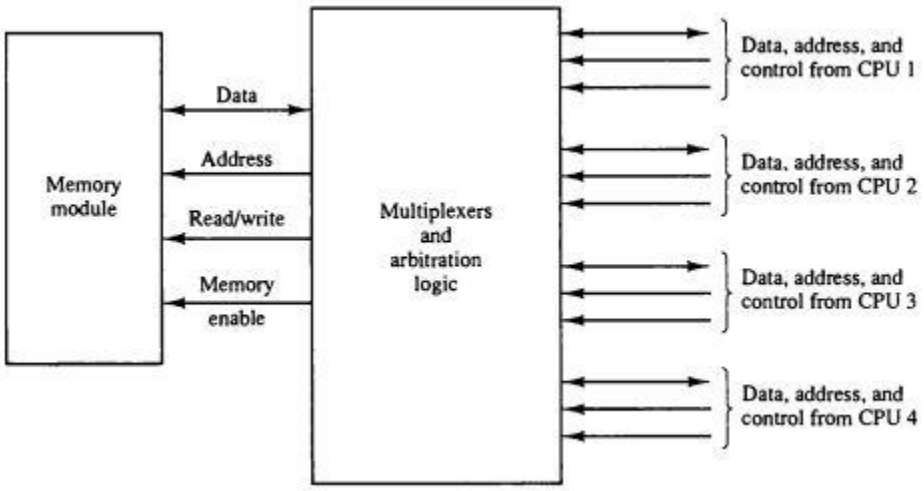

Fig: Block diagram of crossbar switch.

→ Above figure shows a crossbar switch interconnection of 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.

→ Each switch point has control logic to set up the transfer path between a processor and memory. It 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.

4. Multistage Switching Network

→ Controls the communication between a number of resources and destinations.

→ Basic components of a multistage switching network are two-input, two-output interchange switch.

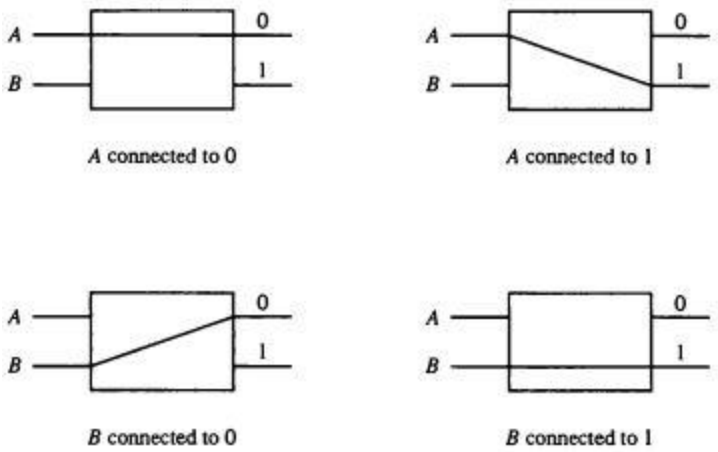

Fig: Operation of a 2×2 interchange switch.

→ As shown in above figure, the 2*2 switch has two inputs labeled A and B, and two outputs 0 and 1. There are control signals associated with the switch that establish the interconnection between the input and output terminals.

→ The switch has capacity of connecting input A to either of the outputs. Terminal B of the switch behaves in a similar fashion. The switch also has the capability to arbitrate between conflicting requests. If inputs A and B both request the same output terminal, only one of them will be connected, the other will be blocked.

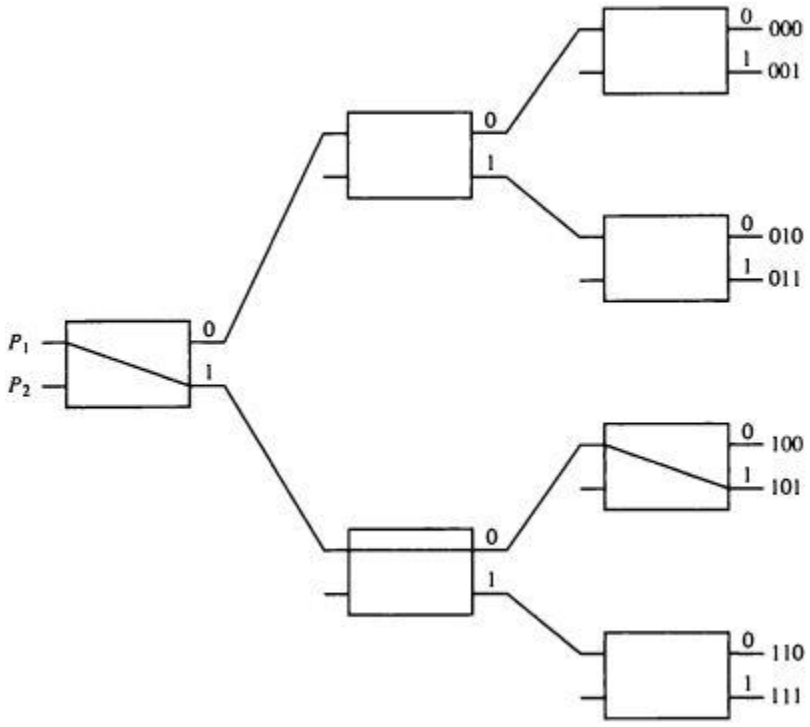

Fig: Binary tree with 2×2 switches.

→ Using the 2*2 switch as a building block, it is possible to build a multistage network as in figure.

5. Hypercube Interconnection

→ The hypercube or binary n-cube multiprocessor structure is a loosely coupled system composed of $N=2^n$ processors interconnected in an n-dimensional binary cube.

→ Each processor forms a node of the cube. Each processor has direct communication paths to n other neighbor processors. These paths correspond to the edges of the cube.

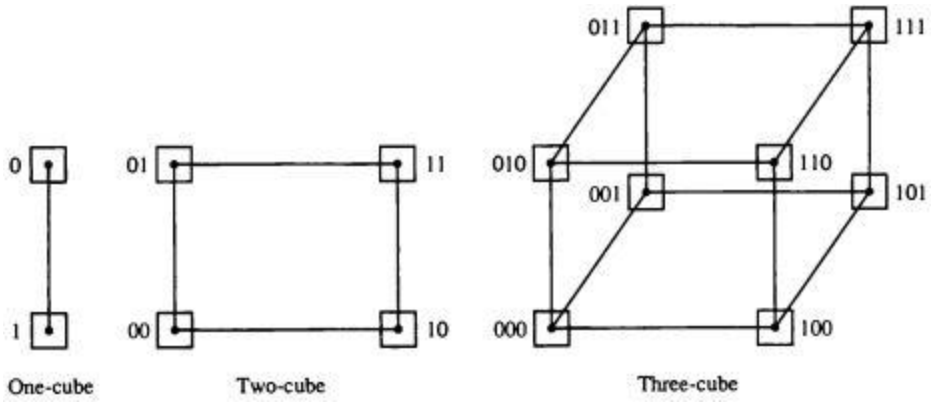

Fig: Hypercube structures for n = 1, 2, 3.

→ Above figure shows the hypercube structure for n = 1, 2, and 3.

→ A one-cube structure has n = 1 and $2^1 = 2$. It contains two processors interconnected by a single path.

→ A two-cube structure has n = 2 and $2^2 = 4$. It contains four nodes interconnected as a square.

→ A three-cube structure has eight nodes interconnected as a cube. An n-cube structure has 2^n nodes with a processor residing in each node. Each node is assigned a binary address in such a way that the addresses of two neighbors differ in exactly one bit position. For example, the three neighbors of the node with address 100 in a three cube structure are 000, 110, and 101. Each of these binary numbers differs from address 100 by one bit value.

→ Routing messages through an n cube structure may take from one to n links from a source node to a destination node. For example, in a three cube structure, node 000 can communicate directly with node 001. It must cross at least two links to communicate with 011 (from 000 to O01 to 011 or from 000 to 010 to 011). It is necessary to go through at least three 1ines to communicate from node 000 to node 111.

→ A routing procedure can be developed by computing the exclusive OR of the source node address with the destination node address. The resulting binary value will have 1 bits corresponding to the axes on which the two nodes differ. The message is then sent along any one of the axes. For example, in a three cube structure, a message at 010 going to 001 produces an exclusive OR of the two addresses equal to O11. The message can be sent along the second axis to 000 and then through the third axis to 001.

11.3 Interprocessor Arbitration

→ 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. The arbitration logic would be part of the system bus controller placed between the local bus and the system bus.

→ Some arbitration processes are:

1. Serial Arbitration Procedure (Daisy-Chain Arbitration)

→ Each processor has its bus arbiter logic with priority-in (PI) priority-out (PO) lines.

→ The priority out (PO) of each arbiter is connected to the priority in (Pl) of the next lower priority arbiter. The PI of the highest priority unit is maintained at logic 1 value.

→ The highest priority unit in the system will always receive access to the system bus when it requests it.

→ The PO output for a particular arbiter is equal to 1 if its PI input is equal to 1 and the processor associated with the arbiter logic is not requesting control of the bus. This is the way that priority is passed to the next unit in the chain. If the processor requests control of the bus and the corresponding arbiter finds its PI input equal to 1, it sets its PO output to 0. Lower priority arbiters receive a 0 in PI and generate a O in PO. Thus the processor whose arbiter has a PI = 1 and PO = 0 is the one that is given control of the system bus.

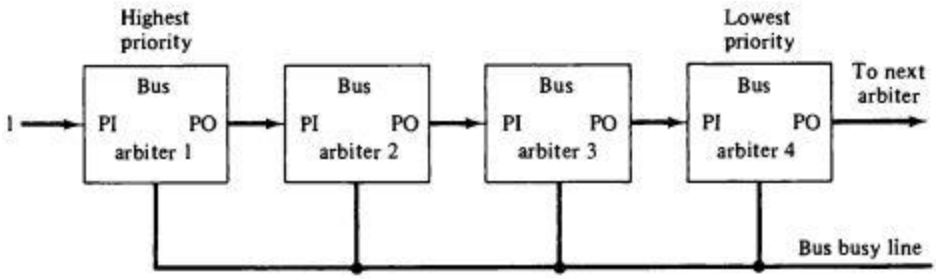

Fig: Serial (daisy chain) arbitration.

2. Parallel Arbitration Process

→ The parallel bus arbitration technique uses an external priority encoder and a decoder. Each bus arbiter in the parallel scheme has a bus request output line and a bus acknowledge input line.

→ The processor takes control of the bus if its acknowledged input line is enabled. The bus busy line provides an orderly transfer of control, as in the daisy chaining case.

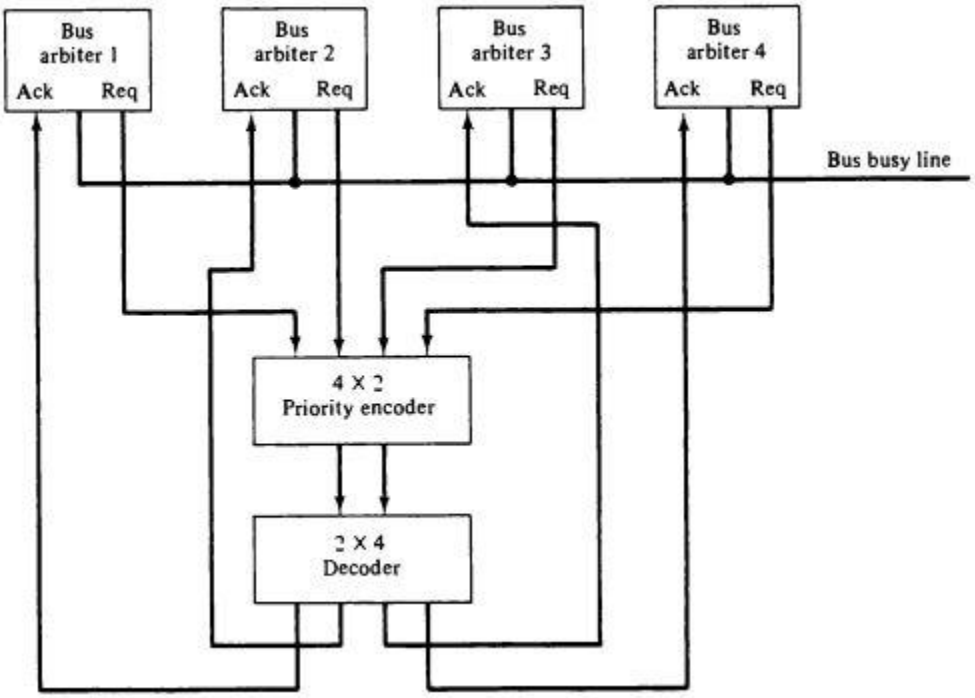

Fig: Parallel arbitration.

→ Above figure shows the 2-bit code from the encoder output drives a 2 X 4 decoder which enables the proper acknowledge line to grant bus access to the highest priority unit.

3. Dynamic Arbitration Algorithm

→ Serial and Parallel arbitration procedures use a static priority algorithm since the priority of each device is fixed by the way it is connected to the bus.

→ In contrast, a **DAA** gives the system the capability for changing the priority of the devices while the system is in operation.

→ There are 5 types of DAA:

i) **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. The service given to each system component with this scheme is independent of its location along the bus. No preference is given to any particular device since each is allotted the same amount of time to communicate with the bus.

ii) **Polling**: In a bus system that uses 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. After a number of bus cycles, the polling process continues by

choosing a different processor. The polling sequence is normally programmable, and as a result, the selection priority can be altered under program control.

iii) **Least Recently Used (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. With this, no processor is favored over any other since the priorities are dynamically changed to give every device an opportunity to access the bus.

iv) **First-Come, First-Serve (FIFO) scheme**: In the first come, first serve scheme, requests are served in the order received. To implement this algorithm, the bus controller establishes a queue arranged according to the time that the bus requests arrive.

v) **Rotating Daisy-Chain**: The rotating daisy chain procedure is a dynamic extension of the daisy chain algorithm. Each arbiter priority for a given bus cycle is determined by its position along the bus priority line from the arbiter whose processor is currently controlling the bus. Once an arbiter releases the bus, it has the lowest priority.

11.4 Interprocessor Communication and Synchronization

Interprocessor Communication

→ Communication refers to the exchange of data between different processes. For example, parameters passed to a procedure in a different processor constitute inter processor communication.

→ The various processors in a multiprocessor system must be provided with a facility for communicating with each other. A communication path can be established through common input output channels. In a shared memory multiprocessor system, the most common procedure is to set aside a portion of memory that is accessible to all processors.

→ In addition to shared memory, a multiprocessor system may have other shared resources. For example, a magnetic disk storage unit connected to an IOP may be available to all CPUs.

→ To prevent conflicting use of shared resources by several processors there must be a provision for assigning resources to processors. This task is given to the operating system. There are three organizations that have been used in the design of operating system for multiprocessors: **1) master slave configuration, 2) separate operating system, and 3) distributed operating system**.

→ In a **master slave mode**, one processor, designated the master, always executes the operating system functions. The remaining processors, denoted as slaves, do not perform operating system functions. If a slave processor needs an operating system service, it must request it by interrupting the master and waiting until the current program can be interrupted.

→ In the **separate operating system organization**, each processor can execute the operating system routines it needs. This organization is more suitable for loosely coupled systems where every processor may have its own copy of the entire operating system.

→ In the **distributed operating system organization**, the operating system routines are distributed among the available processors. However, each particular operating system function is assigned to only one processor at a time. This type of organization is also referred to as a floating operating system since the routines float from one processor to another and the execution of the routines may be assigned to different processors at different times.

→ In a loosely coupled multiprocessor system the memory is distributed among the processors and there is no shared memory for passing information. The communication between processors is by means of message passing through l/O channels.

Interprocessor Synchronization

→ The instruction set of a multiprocessor contains basic instructions that are used to implement communication and synchronization between cooperating processes.

→ Synchronization refers to the special case where the data used to communicate between processors is control information. Synchronization is needed to enforce the correct sequence of processes and to ensure mutually exclusive access to shared writable data.

→ A number of hardware mechanisms for mutual exclusion have been developed. One of the most popular methods is through the use of a binary **semaphore**.
→ A properly functioning multiprocessor system must provide a mechanism that will guarantee orderly access to shared memory and other shared resources. This is necessary to protect data from being changed simultaneously by two or more processors. This mechanism has been termed **mutual exclusion**.
→ Mutual exclusion must be provided in a multiprocessor system to enable one processor to exclude or lock out access to a shared resource by other processors when it is in a critical section. A **critical section** is a program sequence that, once begun, must complete execution before another processor accesses the same shared resource.
→ A binary variable called a semaphore is often used to indicate whether or not a processor is executing a critical section. A semaphore is a software controlled flag that is stored in a memory location that all processors can access. When the semaphore is equal to 1, it means that a processor is executing a critical program, so that the shared memory is not available to other processors.
→ When the semaphore is equal to 0, the shared memory is available to any requesting processor. Processors that share the same memory segment agree by convention not to use the memory segment unless the semaphore is equal to 0, indicating that memory is available.
→ They also agree to set the semaphore to 1 when they are executing a critical section and to clear it to 0 when they are finished.
→ A semaphore can be initialized by means of a test and set instruction in conjunction with a hardware **lock** mechanism.
→ Assume that the semaphore is a bit in the least significant position of a memory word whose address is symbolized by SEM. Let the mnemonic TSL designate the "test and set while locked" operation.
→ The instruction TSL SEM will be executed in two memory cycles (the first to read and the second to write) without interference as follows:

R←M [SEM]	Test semaphore
M [SEM]←1	Set semaphore

11.5 Cache Coherence

→ In multiprocessor system, if the update made on some data item on local cache of any processor is reflected on other processor having the same data item, then it is said to be cache coherence. Or, if same data item resides in cache of multiple processors and its value is same to all cache, then it is known as cache coherence.
→ Cache coherence arises when shared data is to be written as well as read. If one processor modifies a cached value shared in cache by other processors, then all processors must eventually agree on the updated value.
→ For example, the system shown on fig. 1 is coherent system because in all local cache, the value of X=52.

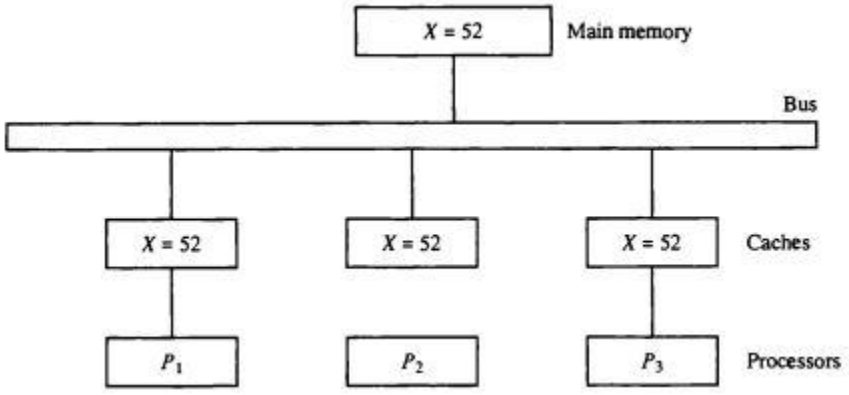

Fig: Cache configuration after a load on X.

→ But in fig. 2, when value at X is updated on local cache of processor P1 using write through cache, then it is not reflected to local caches of P2 and P3. Hence, it is not coherent system.

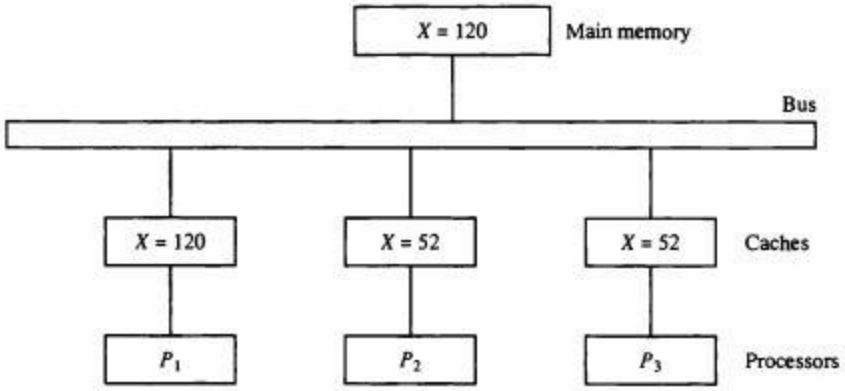

Fig: With write-through cache policy

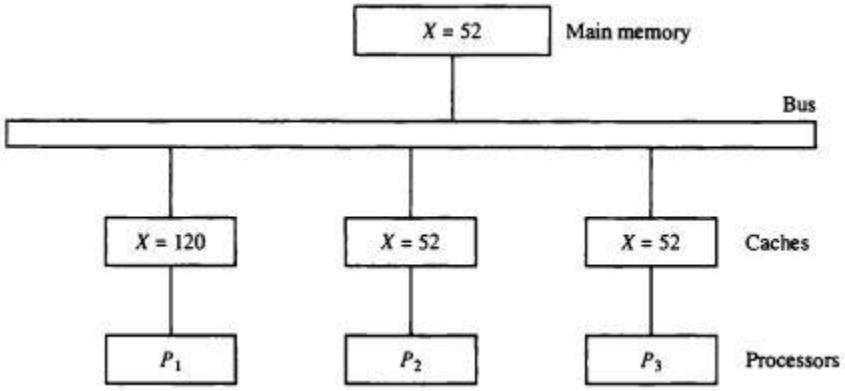

Fig: With write-back cache policy

→ Solution to cache coherence

i) Using shared cache
- All the processors use the same global cache.

ii) Non-cacheable data
- Shared writable data are made non-cacheable.
- Non-shared and readable data are made cacheable.

iii) Two separate caches
- One global cache for writable block of data.
- Local cache for readable data.

iv) Snoopy cache controller
- It is a special hardware that monitors the write operation in any local cache.
- If the cache write is observed, then main memory is updated, and every cache controller sees. If they have the same data, then they mark the data invalid.