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 (IOP).
- As it is most commonly defined, a multiprocessor system implies the existence of multiple CPUs, although usually there will be one or more IOPs as well.
- 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 important distinction between a system with multiple computers and a system with multiple processors.
- Computers are interconnected with each other means of communication lines to form a computer network.
- The network consists of several autonomous computers that may or may not communicate with each other.
- 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.**
- The system as a whole can continue to function correctly with perhaps some loss in efficiency.
- The benefit derived from a multiprocessor organization is an improved system performance.
- The system derives its high performance from the fact that computations can proceed in parallel in one of two ways.
- 1. Multiple independent jobs can be made to operate in parallel.
- 2. A single job can be partitioned into multiple parallel tasks.

Tightly coupled

- Multiprocessors are classified by the way their memory is organized.
- A multiprocessor system with common shared memory is classified as a **shared-memory** or **tightly coupled multiprocessor.**
- In fact, most commercial tightly coupled multiprocessors provide a **cache memory** with each CPU.
- in addition, there is a global common memory that all CPUs can access.
- Information can therefore be shared among the CPUs by placing it in the **common global memory**.

Loosely coupled

- An alternative model of microprocessor is the **distributed-memory or loosely coupled system**. Each processor element in a loosely coupled system has its own private local memory.
- The processors are tied together by a switching scheme designed to route information from one processor to another through a **message-passing scheme.**
- The processors relay program and data to other processors in packets.
- A packet consists of an address, the data content, and some error detection code.
- The packets are addressed to a specific processor or taken by the first available processor, depending on the communication system used.
- Loosely coupled systems are most efficient when the interaction between tasks is minimal, whereas tightly coupled systems can tolerate a higher degree of interaction between tasks.

Interconnection Structures

- The components that form a multiprocessor system are CPUs, lOPs connected to input—output 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 physical forms available for establishing an interconnection network, Some of these schemes are presented in this section:
- **1. Time-shared common bus**
- **2, Multiport memory**
- **3. Crossbar switch**
- **4. Multistage switching network**
- **5. Hypercube system**

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.
- Transfer operations are conducted by the processor that is in control of the bus at the time.
- Any other 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.

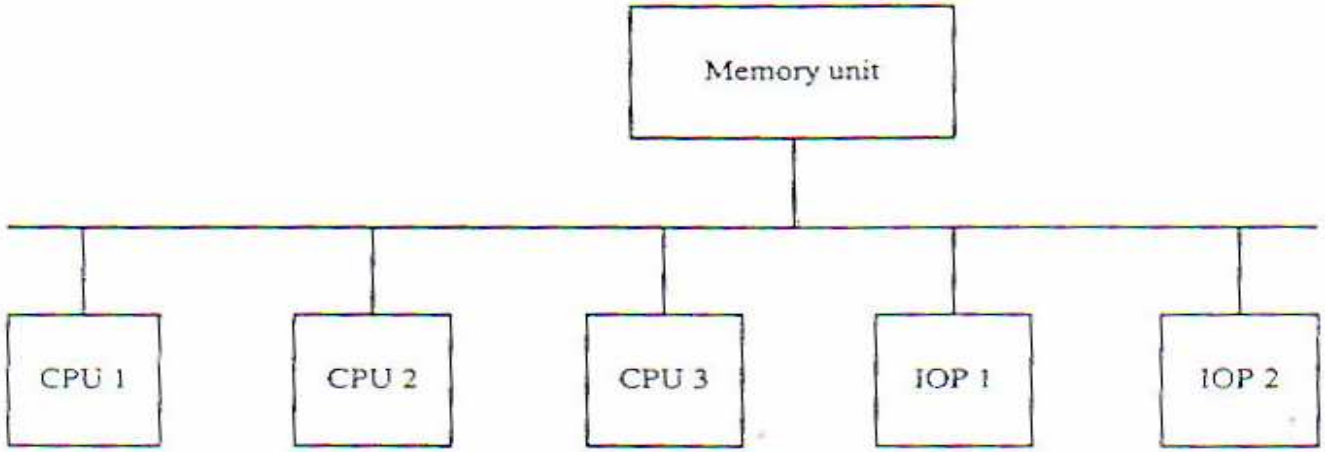


Figure 13-1 Time-shared common bus organization.

- A more economical implementation of a dual bus structure is depicted in Fig. 13-2.
- 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, an lOP, 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 IOP, as well as the local memory, are available to the local processor.
- The memory connected to the common system bus is shared by all processors.
- Only one processor can communicate with the shared memory and other common resources through the system bus at any given time.
-

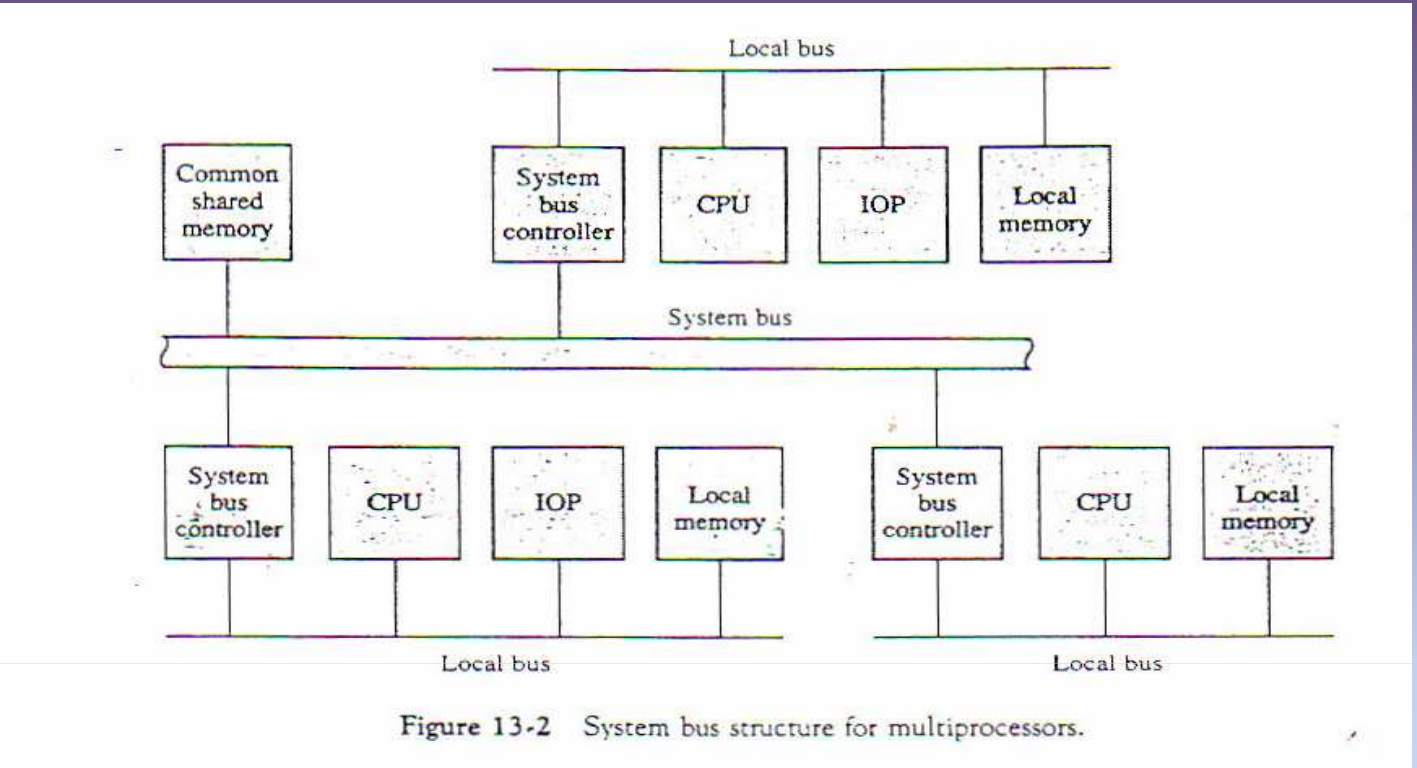


Figure 13-2 System bus structure for multiprocessors.

- The other processors are kept busy communicating with their local memory and I/O devices.

Multiport Memory

- A multiport memory system employs separate buses between each memory module and each CPU. This is shown in Fig. 13-3 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.

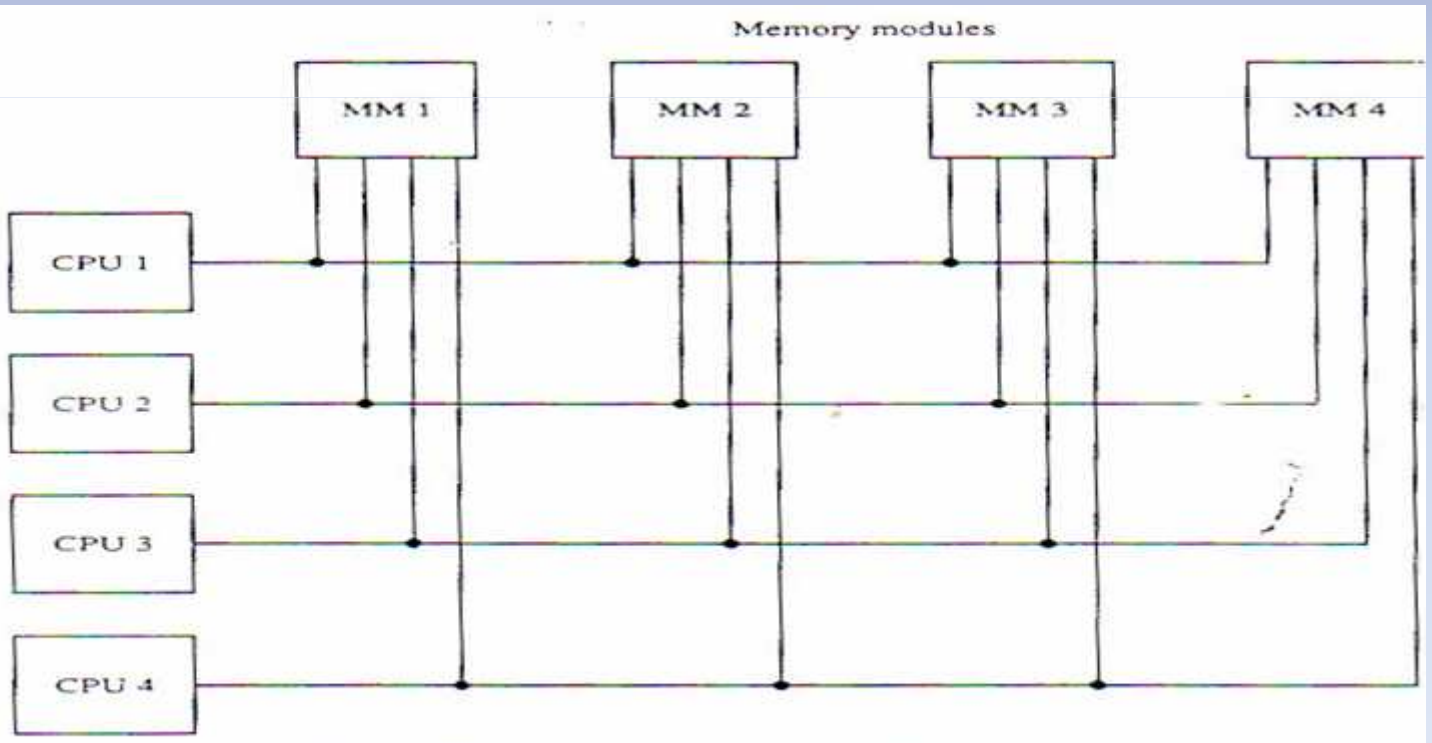


Figure 13-3 Multiport memory organization.

- The memory module is said to have four ports and each port accommodates one of the buses. The module 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.
- 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.

- **The advantage** of the multiport memory organization is the high transfer rate that can be achieved because of the multiple paths between processors and memory.
- **The disadvantage** is that it requires expensive memory control logic and a large number of cables and connectors.
- As a consequences this interconnection structure is usually appropriate for systems with a small number of processors.

Crossbar Switch

- The crossbar switch organization consists of a number of crosspoints that are placed at intersections between processor buses and memory module paths.
- Figure 13-4 shows a crossbar switch interconnection between four CPUs and four memory modules.
- The small square in each crosspoint 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.

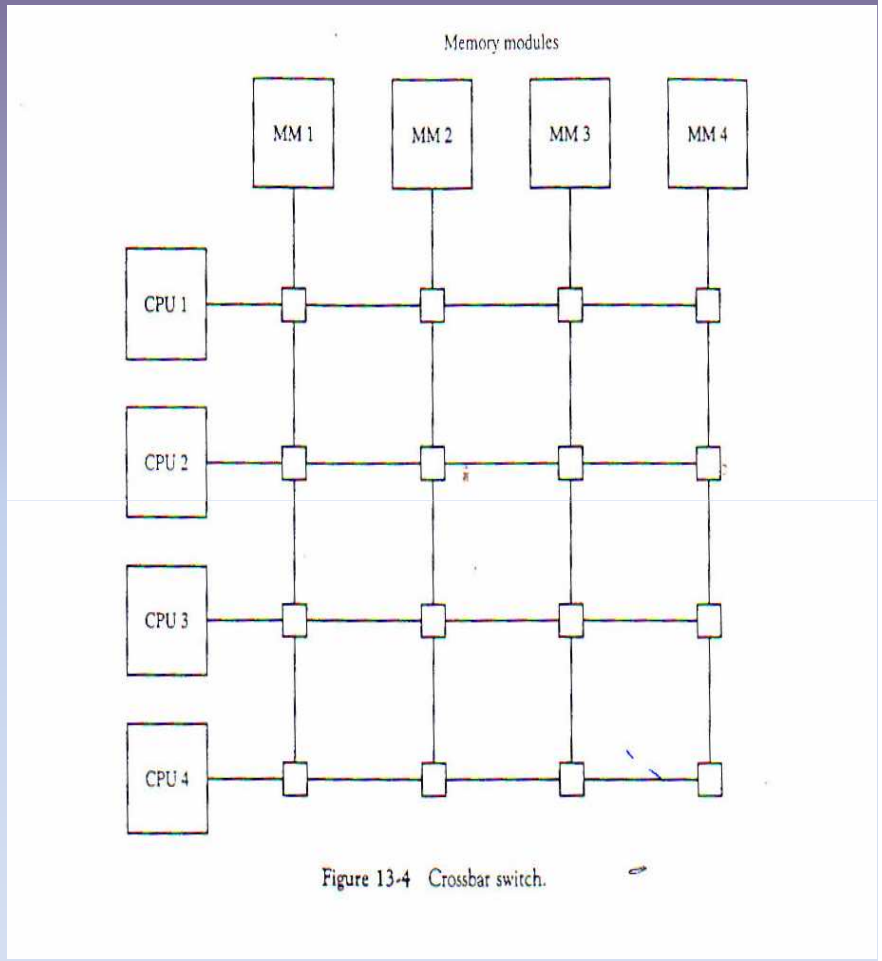
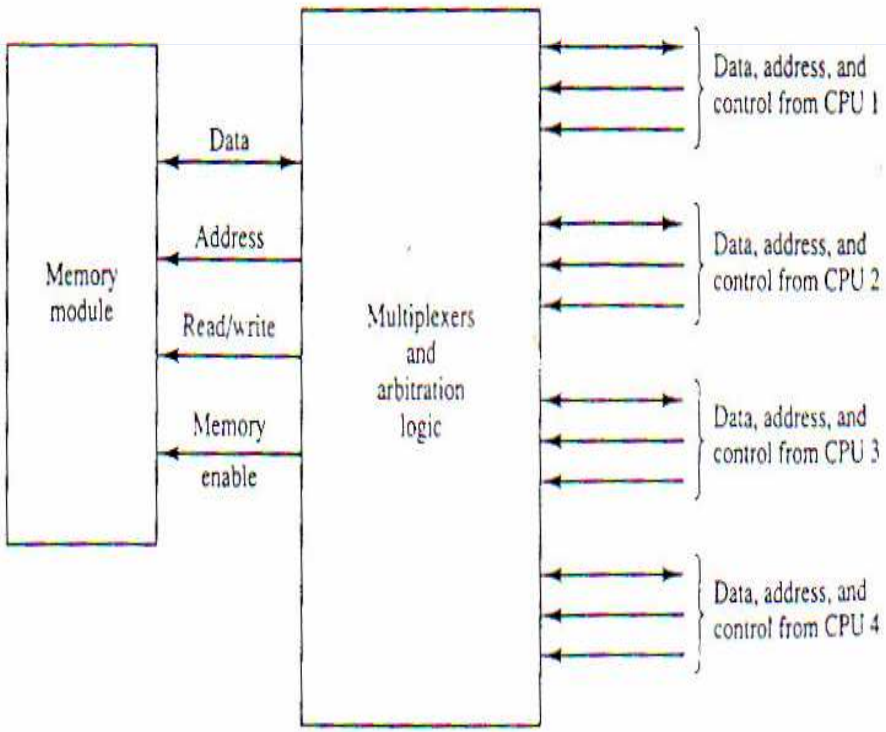


Figure 13-4 Crossbar switch.

- It also resolves multiple requests for access to the same memory module on a predetermined priority basis.
- Figure 13-5 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.
- Priority levels are established by the arbitration logic to select one CPU when two or more CPUs attempt to access the same memory.

Figure 13-5 Block diagram of crossbar switch.



A crossbar switch organization supports simultaneous transfers from all memory modules because there is a separate path associated with each module. However, the hardware required to implement the switch can become quite large and complex.

Multistage switching Network

- The basic component of a multistage network is a two-input, two-output interchange switch. As shown in Fig. 13-6, the 2 x 2 switch has two inputs, labeled A and B, and two outputs, labeled 0 and 1.
- There are control signals (not shown) associated with the switch that establish the interconnection between the input and output terminals.
- .

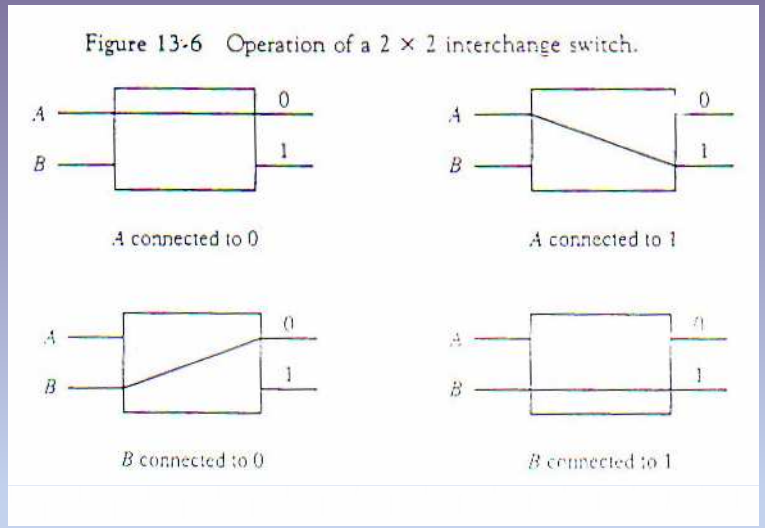


Figure 13-6 Operation of a 2 × 2 interchange switch.

•The switch has the capability 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.

•Using the 2 x 2 switch as a building block, it is possible to build a multistage network to control the communication between a number of sources and destinations

- To see how this is done, consider the binary tree shown in Fig. 13-7.
- The two processors P1 and P2 are connected through switches to eight memory modules marked in binary from 0 0 0 through 111.
- The path from a source to a destination is determined from the binary bits of the destination number .

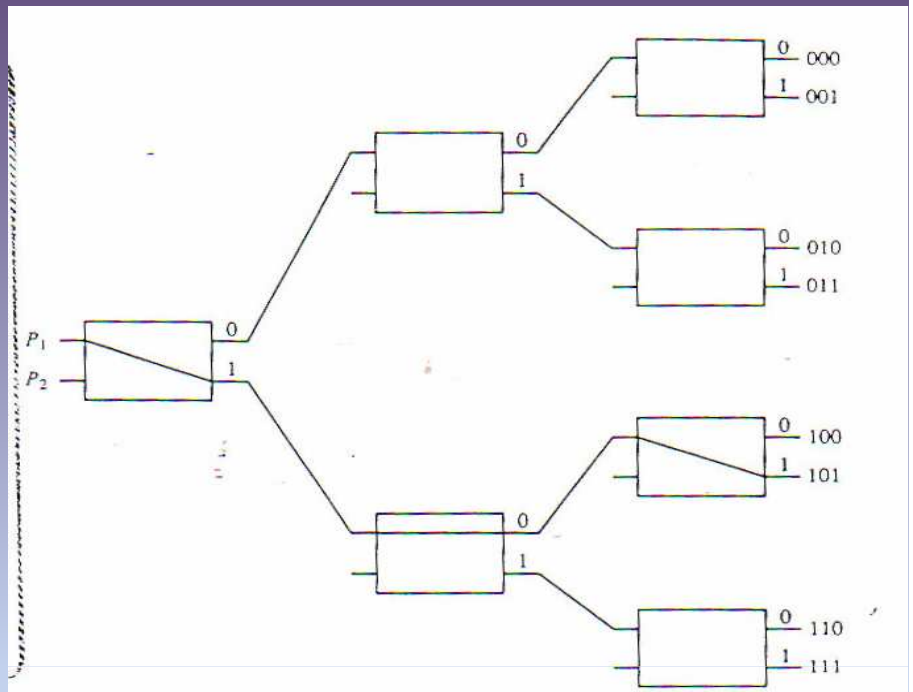


Figure 13-7 Binary tree with 2 × 2 switches.

The first bit of the destination number determines the switch output in the first level. The second bit specifies the output of the switch in the second level, and the third bit specifies the output of the switch in the third level.

For example, to connect P1 to memory 101, it is necessary to form a path from P to output 1 in the first-level switch, output 0 in the second-level switch, and output 1 in the third-level switch. It is clear that either P1 or P2 can be connected to any one of the eight memories, Certain request patterns, however, cannot be satisfied simultaneously. For example, if P1 is connected to one of the destinations 0 0 0 through 0 11 ,P2 can be connected to only one of the destinations 1 0 0 through 111.

Many different topologies have been proposed for multistage switching networks to control processor—memory communication in a tightly coupled multiprocessor system or to control the communication between the processing elements in a loosely coupled system.

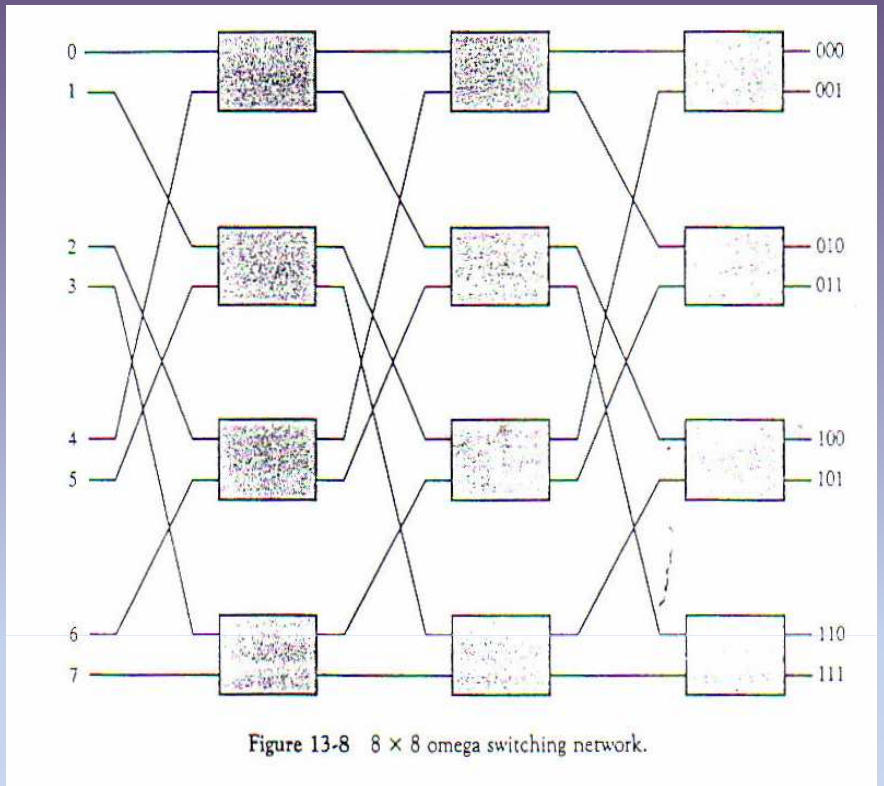


Figure 13-8 8 × 8 omega switching network.

One such topology is the omega switching network shown in Fig. 13-8. . In this configuration, there is exactly one path from each source to any particular destination. Some request patterns, however, cannot be connected simultaneously. For example, any two sources cannot he connected simultaneously to destinations 0 0 0 through 111.

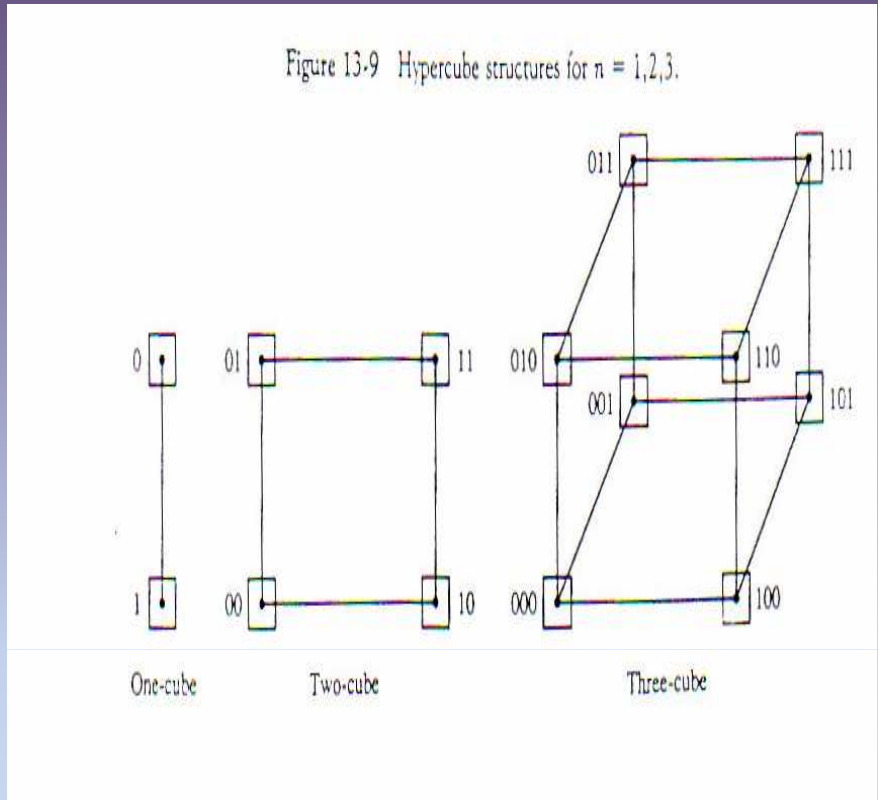
- A particular request is initiated in the switching network by the source, which sends a 3-bit pattern representing the destination number. As the binary pattern moves through the network, each level examines a different bit to determine the 2 x 2 switch setting. Level 1 inspects the most significant bit, level 2 inspects the middle bit, and level 3 inspects the least significant bit.
- W hen the request arrives on either input of the 2 x 2 switch, it is routed a the upper output if the specified bit is 0 or to the lower output if the bit is 1.

- In tightly coupled multiprocessor system, the source is a processor and the destination is a memory module.
- The first pass through the network sets up the path.
- Succeeding passes are used to transfer the address into memory and then transfer the data in either direction, depending on whether the request is a read or a write.
- In a loosely coupled multiprocessor system, both the source and destination are processing elements. After the path is established, the source processor transfers a message to the destination processor.

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. Although it is customary to refer to each node as having a processor, in effect it contains not only a CPU but also local memory and I/O interface.
- Each processor has direct communication paths-to n other neighbor processors. These paths correspond to the edges of the cube. There are 2^ n distinct n-bit binary addresses that can be assigned to the processors.
- Each processor address differs from that of each of its n neighbors by exactly one bit position.

- Figure 13- 9 shows the hypercube structure for n = 1, 2, and 3.
- A one-cube structure has n = 1 and 2^ n = 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.



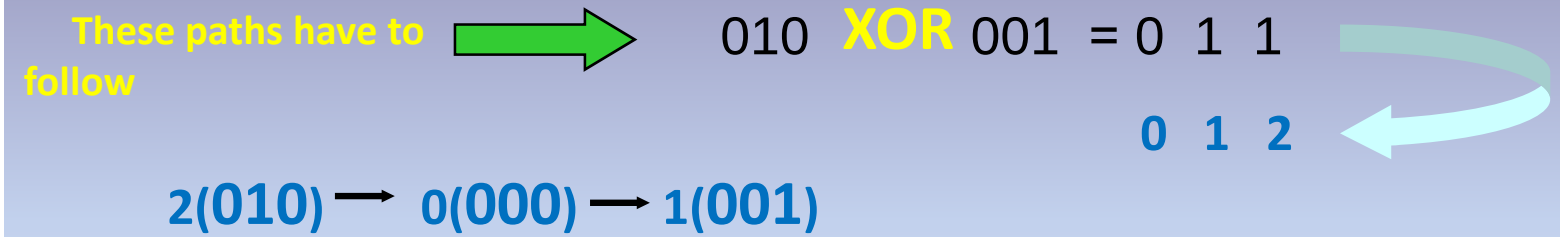
•A three-cube structure has eight nodes interconnected as a cube. An n-cube structure has 2^ n nodes with a processor residing in each node.

• 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 0 0 0 ,11 0 ,and 1 0 1. 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 001 to 011 or from 000 to 010 to 011).
- It is necessary to go through at least three links 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.

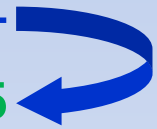
- The resulting binary value will have 1 bits corresponding to the axes on which the two nodes differ.
- EX. Assume S=010 ,and D= 001



- The following example illustrates the use of a deterministic routing technique in a hypercube network.

- **Example** : Assume that S =S5S4 . . . S1S0 to be the source node address, and that D = D5D4 . . . D1D0 is the destination node address in a six-dimensional hypercube message passing system. Let R = S XOR D be the exclusive OR function executed bitwise for each node in the path.
- Consider the case whereby S = 10(001010) and D = 39(100111).

These pathes have to follow  **001010 XOR 100111 =1 0 1 1 0 1**



0 2 3 5

The order in which these dimensions are traversed is not important. Let us assume that the message will follow the route by traversing the following dimensions 5, 3, 2, and 0. Then the route is totally determined as:

10 (001010) → 42(101010) → 34(100010) → 38(100110) → 39 (100111).

source

destination

Cache Coherence

- 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.
- 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.
- Without a proper solution to the cache coherence problem, caching cannot be used in bus-oriented multiprocessors with two or more processors.

Conditions for Incoherence

- Cache coherence problems exist in multiprocessors with private caches because of the need to share writable data.
- Read-only data can safely be replicated without cache coherence enforcement mechanisms.
- To illustrate the problem, consider the three-processor configration with private caches shown in Fig. 13-12.

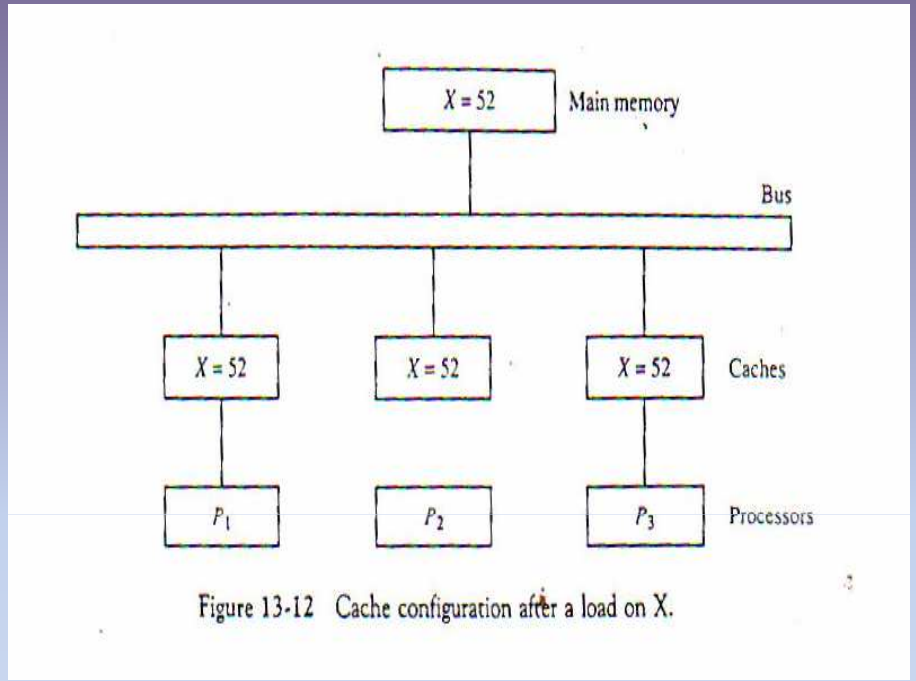
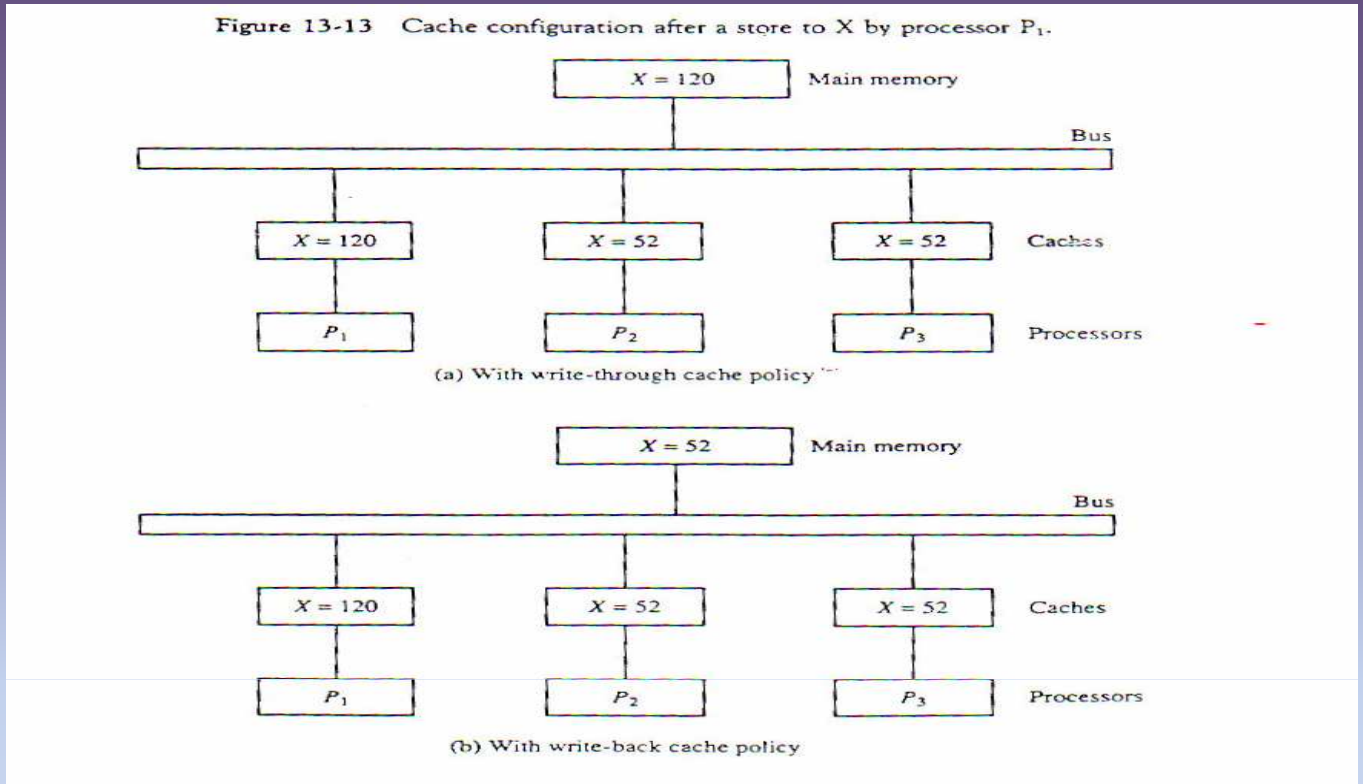


Figure 13-12 Cache configuration after a load on X.

Sometime during the operation an element X from main memory is loaded into the three processors, P1, P2, and P3.

As a consequence, it is also copied into the private caches of the three processors. For simplicity, we assume that X contains the value of 52. The load on X to the three processors results in consistent copies in the caches and main memory.

- If one of the processors performs a store to X, the copies of X in the caches become inconsistent. A load by the other processors will not return the latest value.
- Depending on the memory update policy used in the cache, the main memory may also be inconsistent with respect to' the cache.
- This is shown in Fig. 13-13. A store to X (of the value of 120) into the cache of processor P1 updates memory to the new value in a write-through policy.
- **A write-through policy** maintains consistency between memory and the originating cache, but the other two caches are inconsistent since they still hold the old value.
- In **a write-back policy**, main memory is not updated at the time of the store. The copies in the other two caches and main memory are inconsistent. Memory is updated eventually when the modified data in the cache are copied back into memory.



• Another configuration that may cause consistency problems is a **direct memory access** (**DMA**) activity in conjunction with an TOP 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. 1/0-based memory incoherence can he overcome by making the IOP a participant in the cache coherent solution that is adopted in the system.

Solutions to the Cache Coherence Problem

- 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.
- For performance considerations it is desirable to attach a private cache to each processor.
- One scheme that has been used allows only nonshared and read-only data to be stored in caches. Such items are called **Cachable.**
- Shared writable data are **noncachable.**
- 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 noncachable data remain in main memory.

- This method restricts the type of data stored in caches and introduces an extra software overhead that may degradate performance.
- 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 RW block, the other caches are not affected 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 require the ability to tag information in order to disable caching of shared writable data.
- Hardware-only solutions are handled by the hardware automatically and have the advantage of higher speed and program transparency.
- In the hardware solution, the cache controller is specially designed to allow it to monitor all bus requests from CPUs and IOPs.
- All caches attached to the bus constantly monitor the network for possible write operations.
- Depending on the method used, they must then either update or invalidate their own cache copies when a match is detected.
- 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 problem by means of snoopy cache protocol.
- The simplest method is to adopt a write-through policy and use the following procedure.
- All the snoopy controllers watch the bus for memory store operations.
- 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 case inconsistent versions are prevented.

Hardware Solution: Snooping Cache

- Widely used in bus-based multiprocessors.
- The cache controller constantly watches the bus.
- **Write Invalidate**
- When a processor writes into C*, all copies of it in*
- *other processors are invalidated. These processors*
- have to read a valid copy either from M, or from the
- processor that modified the variable.
- **Write Broadcast**
- Instead of invalidating, why not broadcast the updated
- value to the other processors sharing that copy?
- This will act as write through for shared data, and
- write back for private data.
- ***Write broadcast consumes more bus bandwidth* compared *to write invalidate. Why?***

MESI Protocol

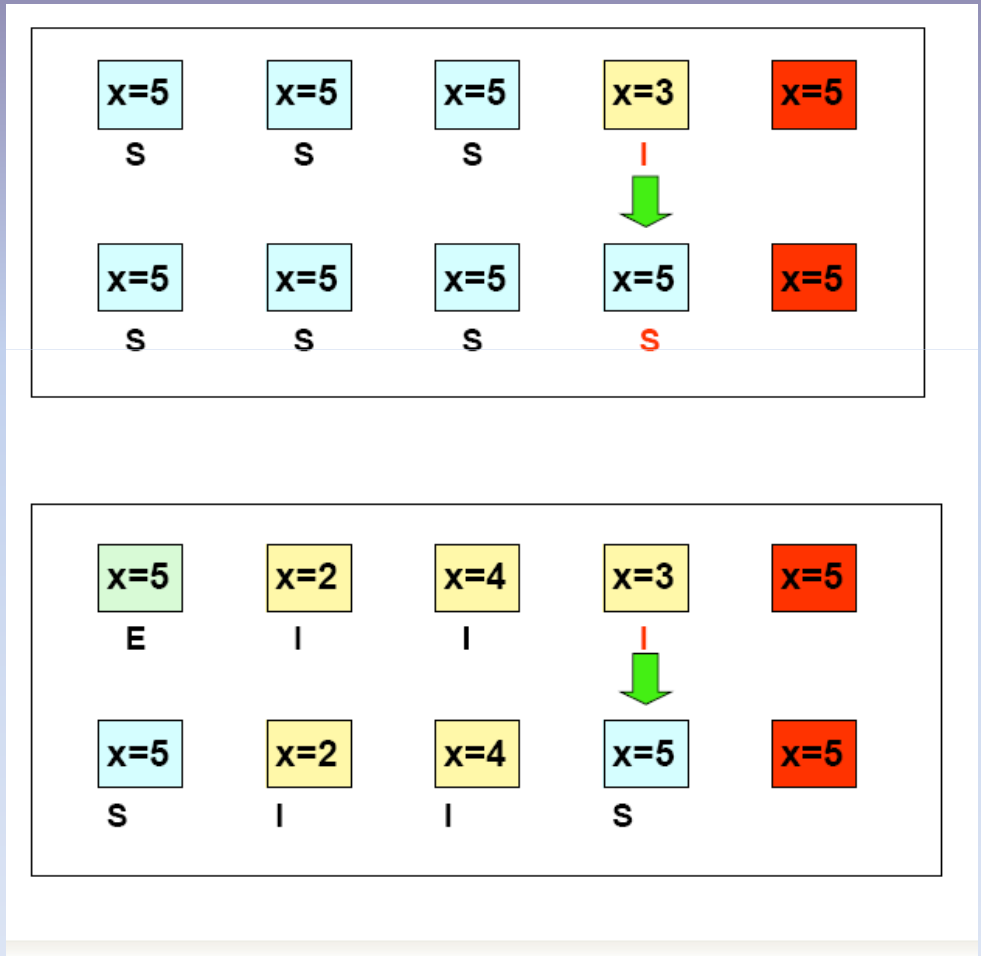
- It is a version of the snooping cache protocol.
- Each cache block can be in one of four states:
- **INVALID Not valid**
- **SHARED Multiple caches may hold valid copies.**
- **EXCLUSIVE No other cache has this block, M-block is valid**
- **MODIFIED Valid block, but copy in M-block is not valid.**

Event	*Local*	*Remote*
Read hit	Use local copy	No action
Read miss	I to S, or I to E	(S,E,M) to S
Write hit	(S,E) to M	(S,E,M) to I
Write miss	I to M	(S,E,M) to I

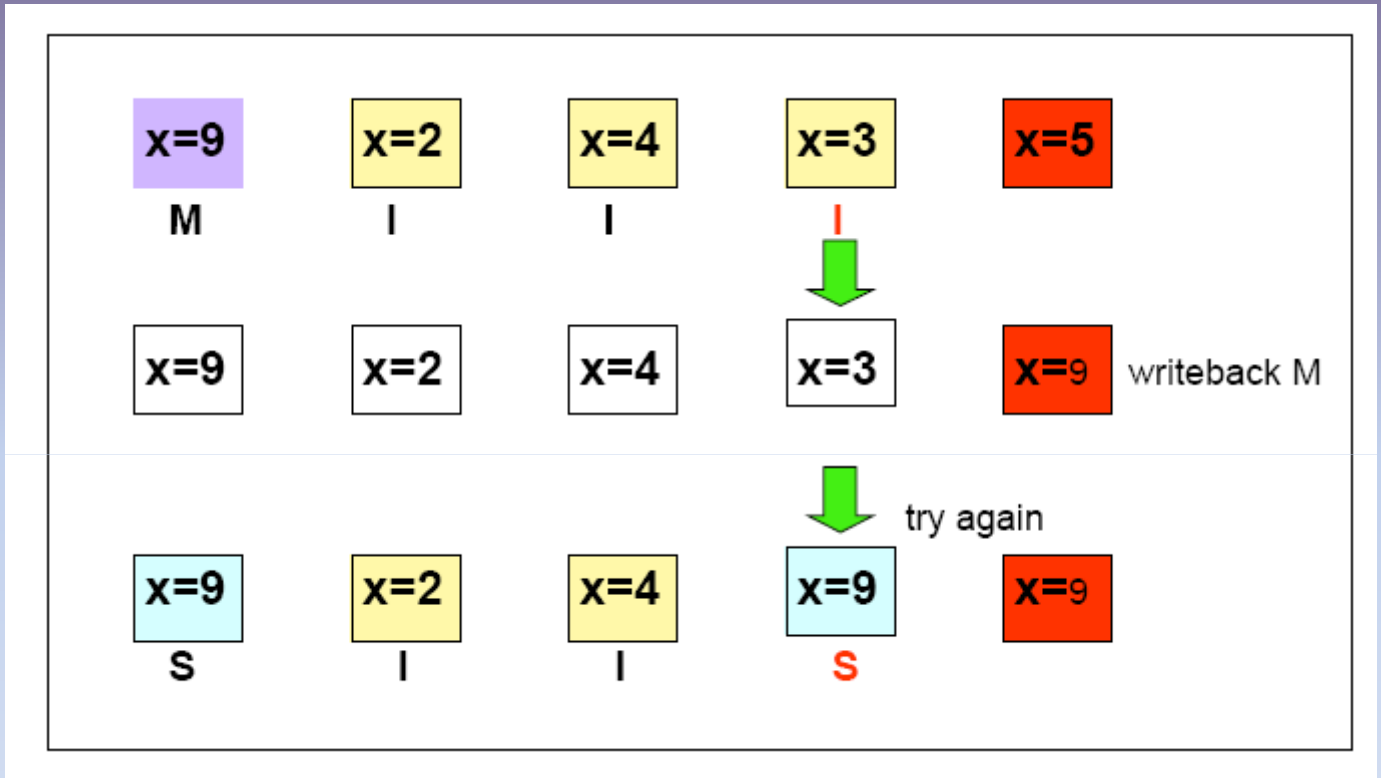
When a cache block changes its status from M, it first updates the main memory.

Examples of state transitions under MESI protocol

A. Read Miss

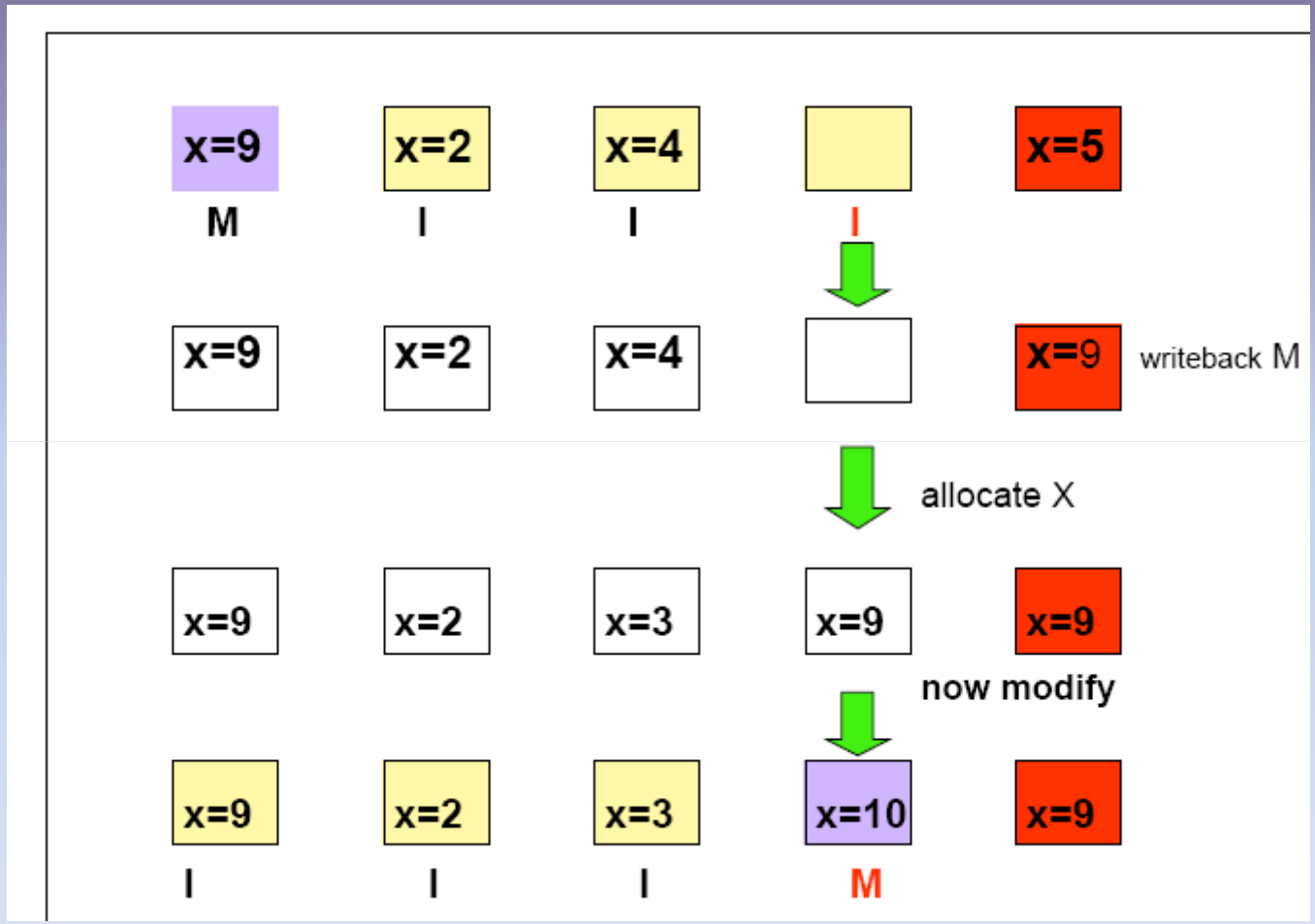


B. More Read Miss



Following the read miss, the holder of the modified copy signals the initiator to try again. Meanwhile, it seizes the bus, and write the updated copy into the main memory.

C. Write Miss


x=9
M
x=2
I
x=4
I
I
x=5
x=9
x=2
x=4
x=9
writeback M
allocate X
x=9
x=2
x=3
x=9
x=9
now modify
x=9
I
x=2
I
x=3
I
x=10
M
x=9