UNIT-VIII

MULTI PROCESSORS

Characteristics of Multiprocessors :-

A Multiprocessor system is an interconnection of two or more CPU's with memory & i/o equipment. The term "processor" in Multiprocessor can have either CPU or i/o process

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

There are some similarities b/w multiprocessor and multicomputer systems since both support concurrent operations. Computers are interconnected with each other by means of communication lines to form a computer n/w. The n/w consists of several autonomous computers that may or may not communicate with each other.

A multiprocessor system is controlled by one O.S. that provides interaction b/w processors & all the components of the system cooperate in the solution of a problem.

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

Multiprocessors are classified by the way their memory is organized.

1) Tightly Coupled Multiprocessor.
2) Loosely coupled Multiprocessor.

A Multiprocessor system with common shared memory is classified as a Shared Memory or tightly coupled multi processor.

In this, each provide a Cache memory with each CPU. There is a global common memory that all CPU's can access. Information can be shared among the CPU's by placing it in the common global memory.

Another Alternative model of microprocessor is distributed Memory or loosely coupled system. Each PE 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 & data to other processor in packets. A packet consists of an address, the data content & some error detection code. The packets are addressed to a specific processor or taken by the first available processor.

Loosely coupled systems are most efficient when the interaction b/w tasks is minimal, where as tightly coupled systems can tolerate a higher degree of interaction b/w task's

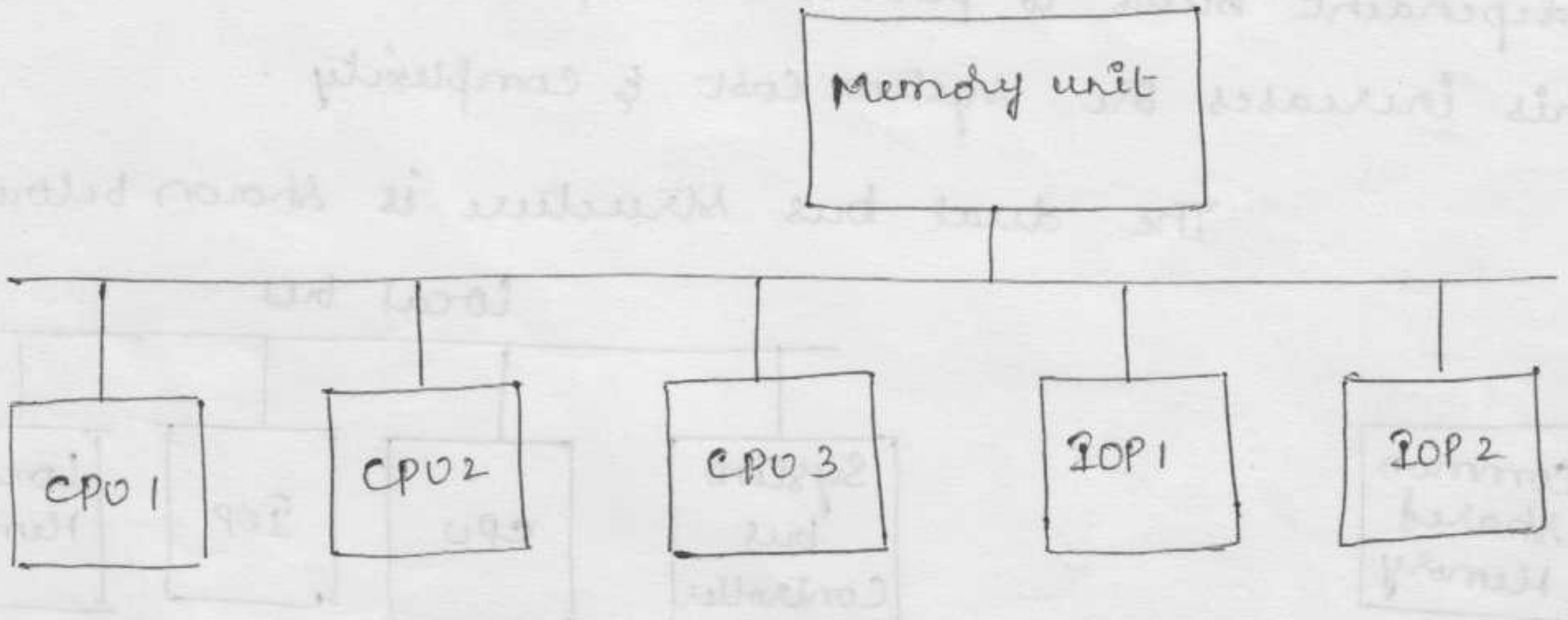
Interconnection Structures :-

There are several physical forms available for establishing an interconnection n/w.

1) Time - Shared common bus
2) Multiport Memory
3) cross bar switch
4) Multistage switching n/w.
5) Hypercube system.

Time Shared Common bus :- 2

A Common-bus multiprocessor system consists of a no. of processors connected through a common path to a memory unit. A time-shared Common bus is shown below.

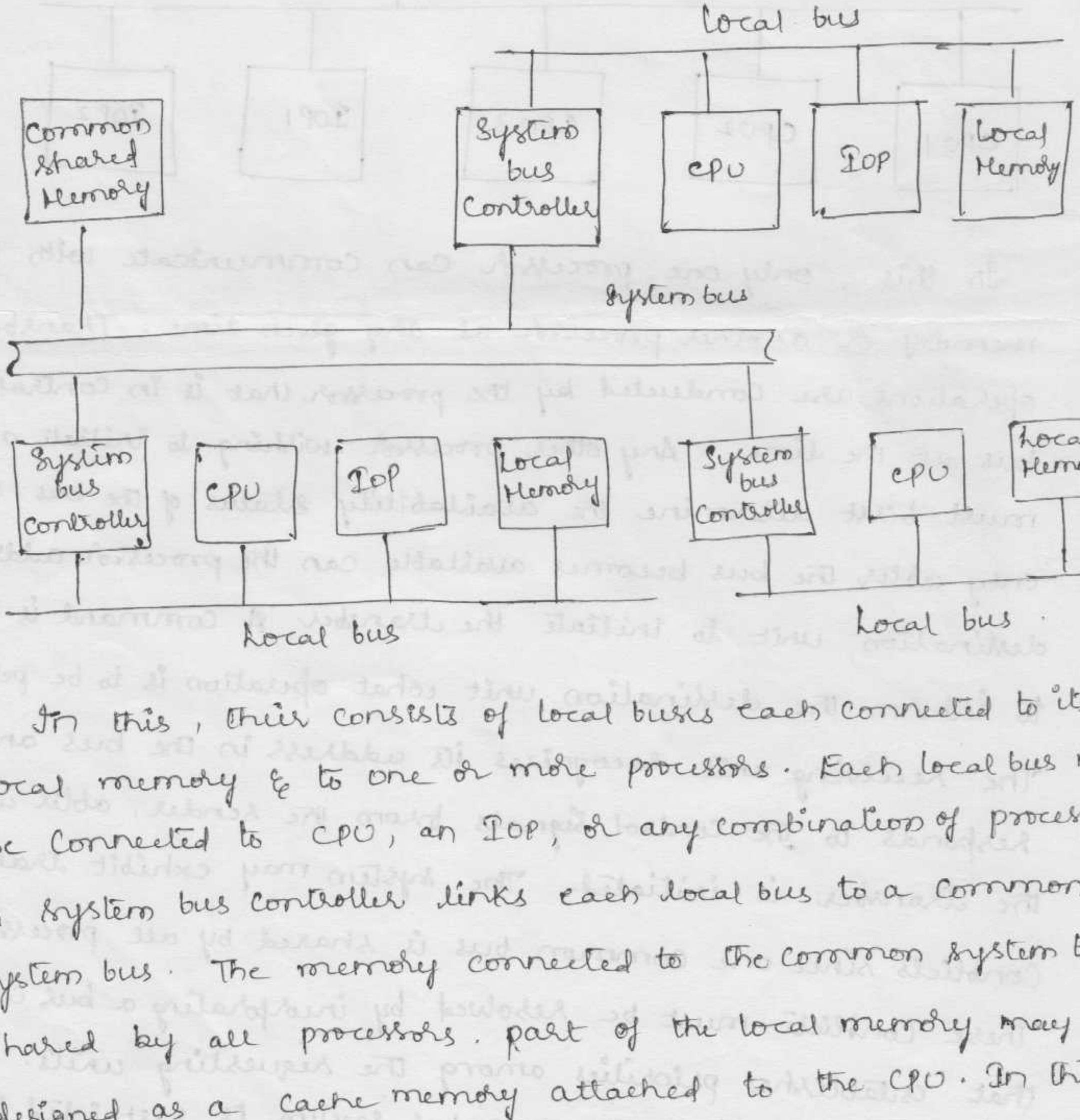


In this, only one processor can communicate with the memory or another processor at any given time. Transfer of operations are conducted by the processor that is in control of the bus at the time. Any other processor wishing to initiate a transfer must first determine the availability status of the bus 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.

A single common-bus system is restricted to one transfer at a time. i.e., when one processor is communicating

with the memory, all other processors are either busy with internal operations or must be idle waiting for bus. The total overall transfer rate with in the system is limited by the speed of the single path. The processors in the system can be kept busy, that the implementation of two or more independent buses to permit multiple simultaneous bus trans-s. This increases the system cost & complexity.

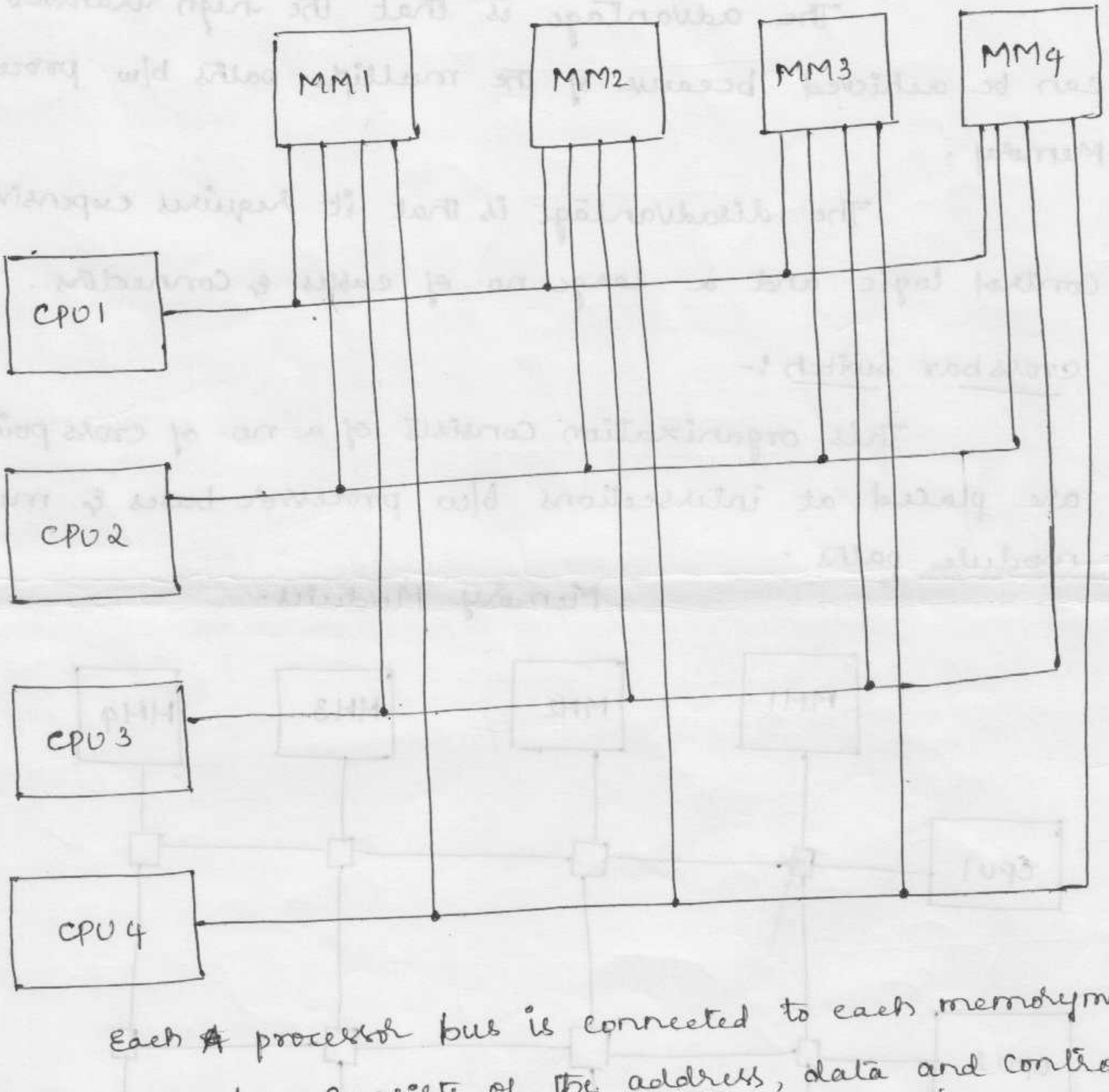
The dual bus structure is shown below



In this, their consists of local buses each connected to its own local memory & to one or more processors. Each local bus may be connected to CPU, an IOP, or any combination of processors. A system bus controller links each local bus to a common system bus. The memory connected to the common system bus is shared by all processors. part of the local memory may be designed as a cache memory attached to the CPU. 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.

Multiport Memory :-

A multiport memory system consists of separate buses b/w each memory module & each CPU. The below fig. consists for four CPU's & four memory modules (MM).



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 MM's is said to have four ports & each port accommodates one of the buses.

The Module must have 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. The priority for memory acces associated with each processor may be established by the physica port position that its bus occupies in each Module. The CPU1 has the highest-priority than CPU2, CPU3, CPU4.

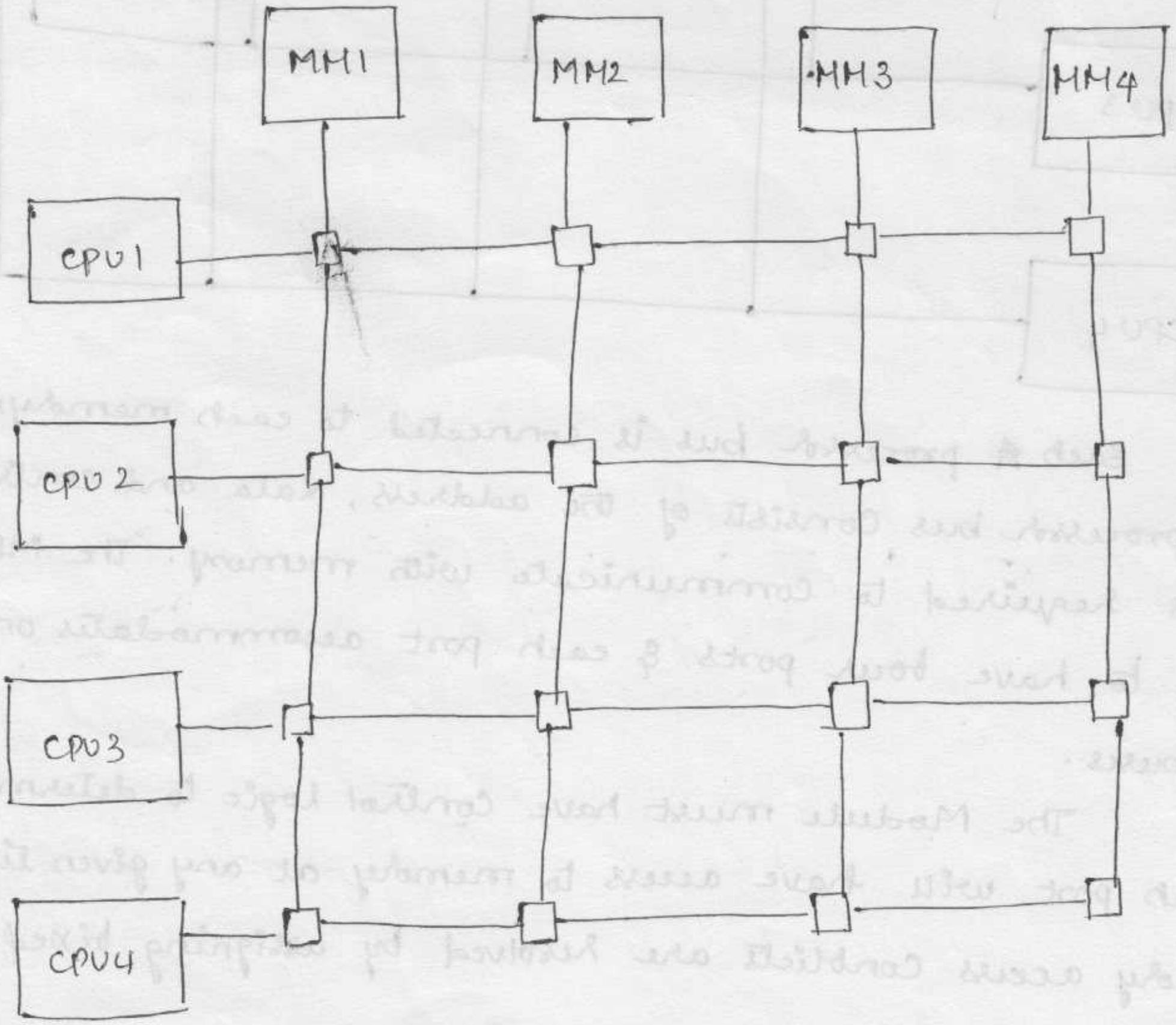
The advantage is that the high transfer rate that can be achieved because of the multiple paths b/w processors and Memory.

The disadvantage is that it requires expensive memory control logic and a large no. of cables & connectors.

crossbar Switch :-

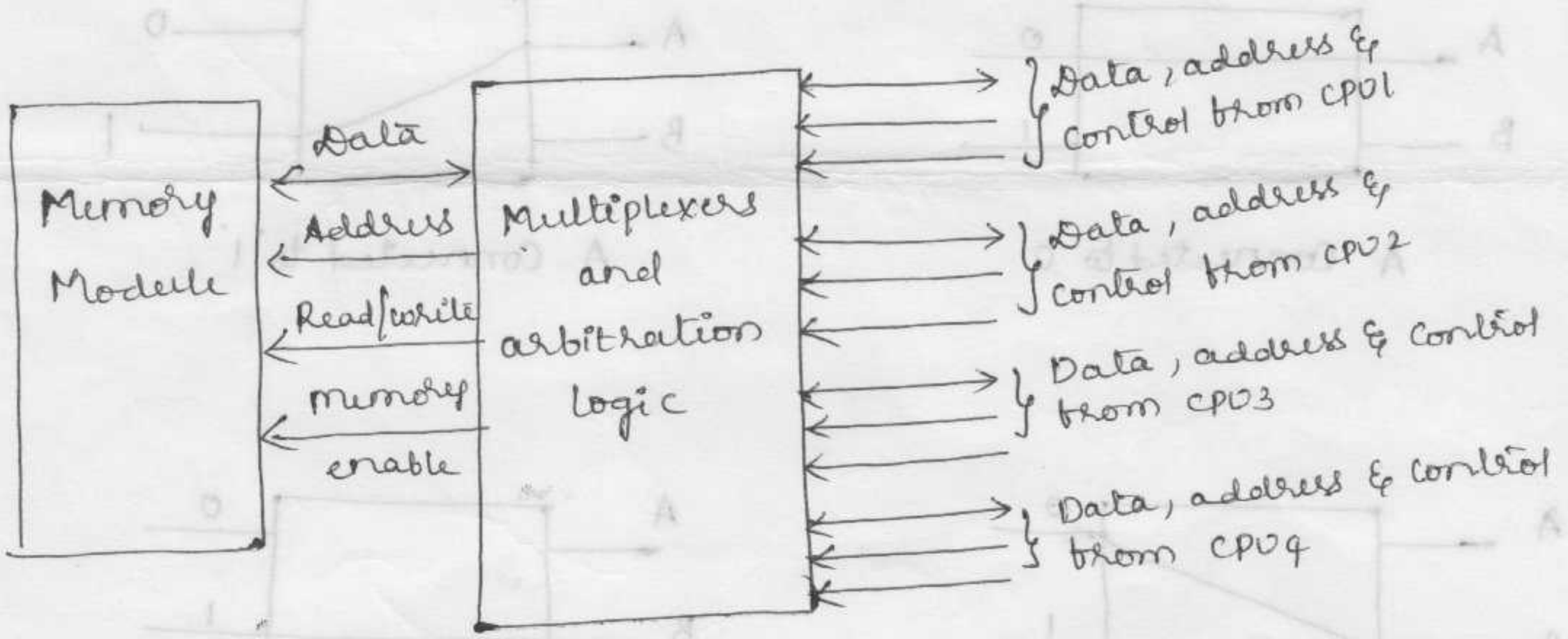
This organization consists of a no. of cross points that are placed at intersections b/w processor buses & memory module paths.

Memory Modules.



The fig. shows a cross bar switch interconnection b/w four CPU's & four memory Modules. The small square in each crosspoint is a switch that determines the path from a processor to a memory module. Each point has control logic to set up the transfer path b/w a processor and memory. The address is placed in the bus to determine whether its particular module is being accessed. It also resolves multiple requests for access the same memory module on a priority basis.

The below fig. shows the functional design of a cross bar switch connected to one memory module.



The circuits consists of multiplexers that select the data, address & control from one CPU for communication with the memory module.

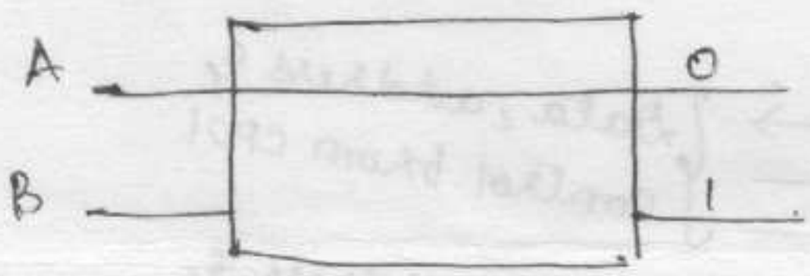
priority levels are established by the arbitration logic to select one CPU when two or more CPU's attempts to access the same memory. The multiplexers are controlled with the the binary code that is generated by a priority encoder with in the arbitration logic.

A cross bar switch organization supports simultaneous transfers from all memory modules because there is a separate path associated with each module.

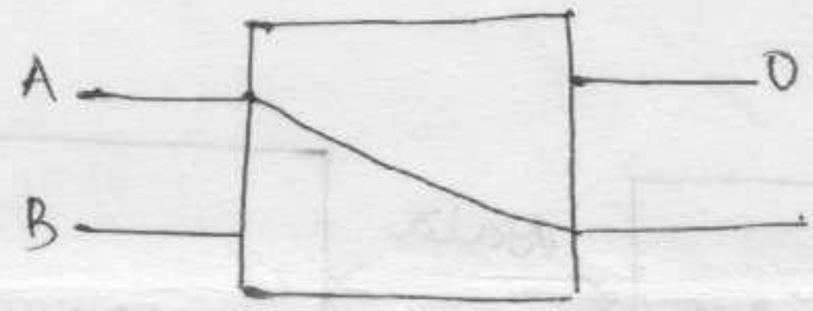
The h/w required to implement the switch can become quite large & complex.

Multistage Switching Network :-

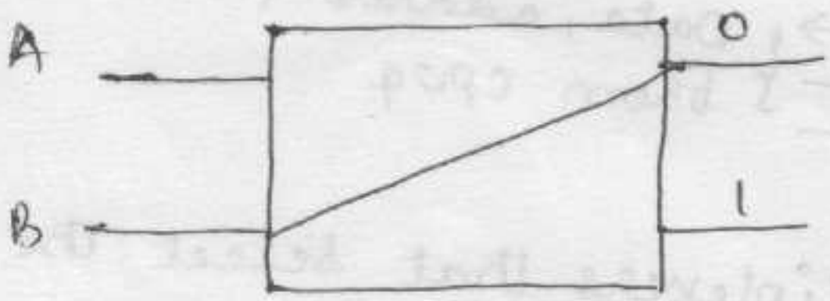
The basic component of a multistage n/w is a two - i/p, two - o/p interchange switch. The below fig. are the 2x2 switch has two - i/p's, labeled as A, B & two o/p labeled as 0 & 1. There are control signals associated with the switch that establish the interconnection b/w the i/p & o/p terminals.



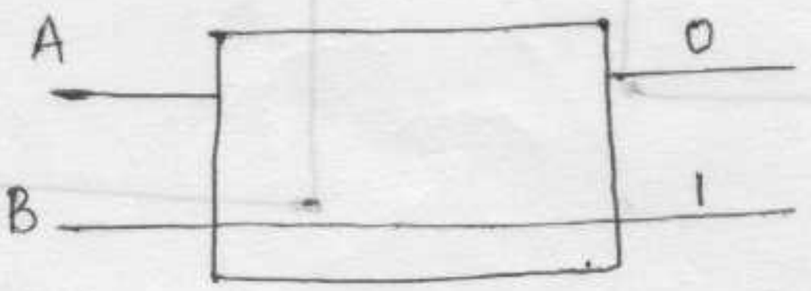
A Connected to 0



A Connected to '1'



B Connected to 0

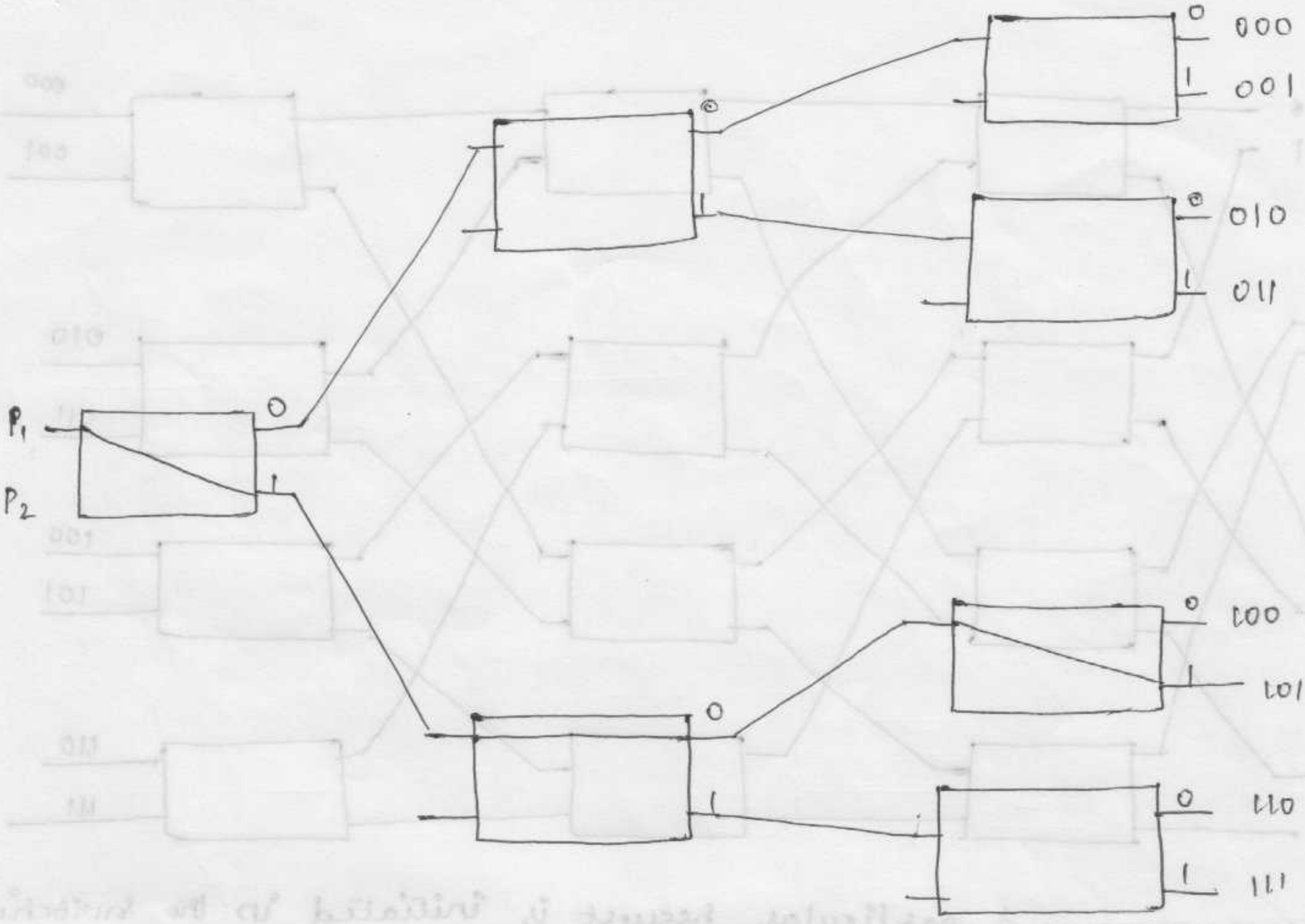


B Connected to '1'

The switch has the capability of connecting i/p's A & B to either of the o/p's. If i/p's A & B both request the same o/p terminal, only one of them will be connected, the other is blocked.

Using the 2x2 switch, we are going to build a multistage n/w to control the communication b/w a no. of sources & destinations.

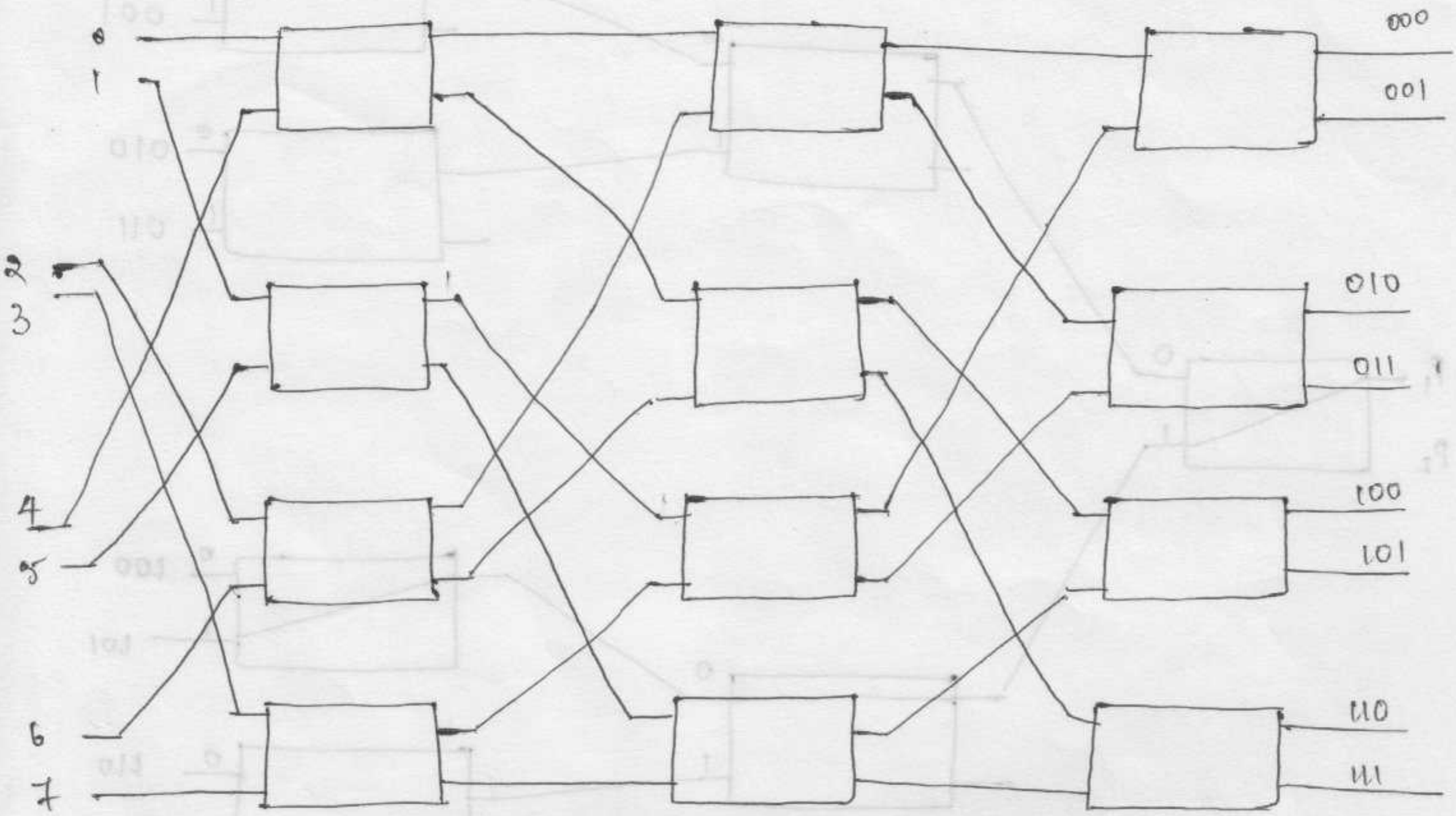
The construction of binary tree is shown below.



The two processors P_1 & P_2 are connected through switches to eight memory modules marked in binary from 000 through 111. The path from a source to a destination is determined from the binary bits of the destination number.

The first bit of the destination number determines the switch o/p in the first level. The second bit specifies the o/p in the second level. The third bit specifies the o/p in the third level.

Many different topologies have been proposed in the multistage switching n/w's to control processor - memory communication in a tightly - coupled multiprocessor system or to control the comm^n b/w the PE's in a loosely - coupled system. One such topology is the omega switching n/w which is shown below. In this configuration there is exactly one path from each source to any particular destination. Request patterns can't be connected simultaneously. For eg, any two sources can't be connected simultaneously to destinations 000 & 001.



A particular request is initiated in the switching n/w by the source, which sends a 3-bit pattern representing the destination number. As the binary pattern moves through the n/w, each level examines a different bit to determine the 2x2 switch setting. Level 1 inspects the MSB, Level 2 inspects the middle bit, level 3 inspects the LSB. When the request arrives on either i/p of the 2x2 switch, it is routed to the upper o/p if the specified bit is '0' or to the lower o/p if the bit is 1.

In a tightly-coupled system, the source is a processor & the destination is a memory module. The first pass through the n/w sets up the path. Succeeding passes are used to transfer the address into memory & then transfer the data in either direction, depending on whether the request is a read / write.

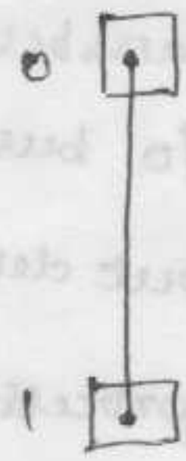
In a loosely coupled system, both the source & destination are PE's. After the path is established, the source processor transfer a message to the destination processor.

Hyper cube Inter connection :-

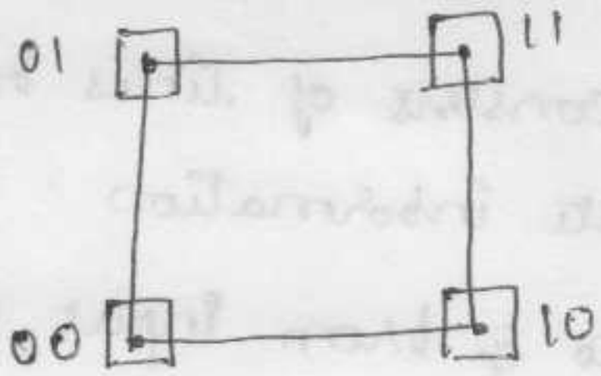
The hypercube or binary n-cube multiprocessor structure is a loosely coupled system composed of $N = 2^n$ processors inter-connected in an n-dimensional binary cube. Each processor forms a 'node' of the cube. Each ~~processor~~ node as processor but also local memory & I/o interface. Each processor has direct communication paths to 'n' other neighbour 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' neighbours by exactly one bit position.

The below fig. shows the hypercube structure.

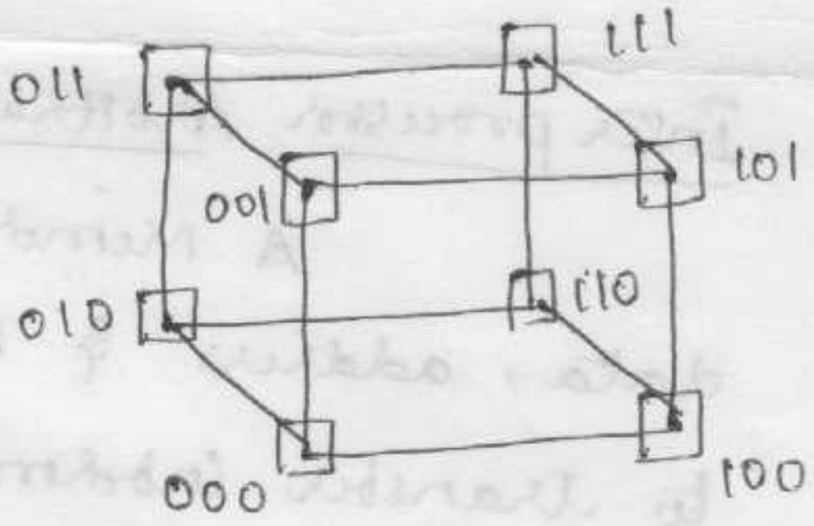
for $n = 1, 2, 3$.



One-cube



Two-cube



Three-cube.

A one-cube structure has $n = 1$ & $2^n = 2$. It contains two processors interconnected by a single path. A two-cube structure has $n = 2$ & $2^2 = 4$. It contains four nodes interconnected as a square. A three-cube structure has eight nodes interconnected as a cube. A 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 neighbours differ in exactly one bit position.

Routing messages through an n-cube structure may take from one to 'n' links from a source node to a destination node. For eg, in a three-cube structure, node 000 can communicate directly with node 001. But it communicates with 011, it takes two links.

A routing procedure can be developed by computing the EX-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 eg, a message at 010 going to 001 produces an ex-or of two addresses equal to 011. The message can be sent along the second axis to 000 & then through the third axis to 001.

Inter processor Arbitration :-

A Memory bus consists of lines for transferring the data, address & read/write information. An I/O bus is used to transfer information to & from input & output devices. A bus that connects major components in a multiprocessor system, such as CPU's, IOP's & memory is called 'System bus'.

The 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 b/w local bus & the system bus.

System Bus :-

A typical system bus consists of approximately 100 Sig lines. These lines are divided into three functional units i.e., data, address & control

In addition to these there are power distribution lines that supply power to the components.

The datalines provide a path for the transfer of data b/w processors & common memory. The number of datalines is usually a multiple of 8 i.e., 16 & 32

The address lines are used to identify a memory address or any other source or destination such as i/p or o/p ports.

Data transfers over the system bus are divided into two types 1) Synchronous bus

2) Asynchronous bus.

In synchronous bus, each data item is transferred during a time slice known to both source & distination units. Synchronization is achieved by driving both units from a common clock source. An alternative procedure is to have separate clocks of approximately the same frequency in each unit. The synchronization signals are transmitted periodically.

In Asynchronous bus, each data item being transferred is accompanied by handshaking control signals to indicate when the data are transferred from the source & received by the distination.

The control lines provide signals for controlling the information b/w units. The timing signals indicate the validity of data & address information. The command signals specify operations to be performed.

Some of the control signals are memory read & write, acknowledge of a transfer, interrupt requests, bus control signals such as bus request & bus grant & signals for arbitration procedures.

If the processor requests control of the bus & the 8 corresponding arbiter finds its PI i/p equal to 1, it sets its PO o/p to '0'. Lower-priority arbiters receive a 0 in PI & generate a '0' in PO. Thus the processor whose arbiters has a PI=1 & PO=0 is the one that is given control of the system bus.

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 relinquishes control of the bus.

The bus busy line provides a mechanism for an orderly transfer of control. The busy line comes from open-collector circuits in each unit & provides a wired-OR logic connection. When an arbiter receives control of the bus it examines the busy line. If the line is 'inactive', it means that no other processor is using the bus. The arbiter activates the busy line & its processor takes control of the bus.

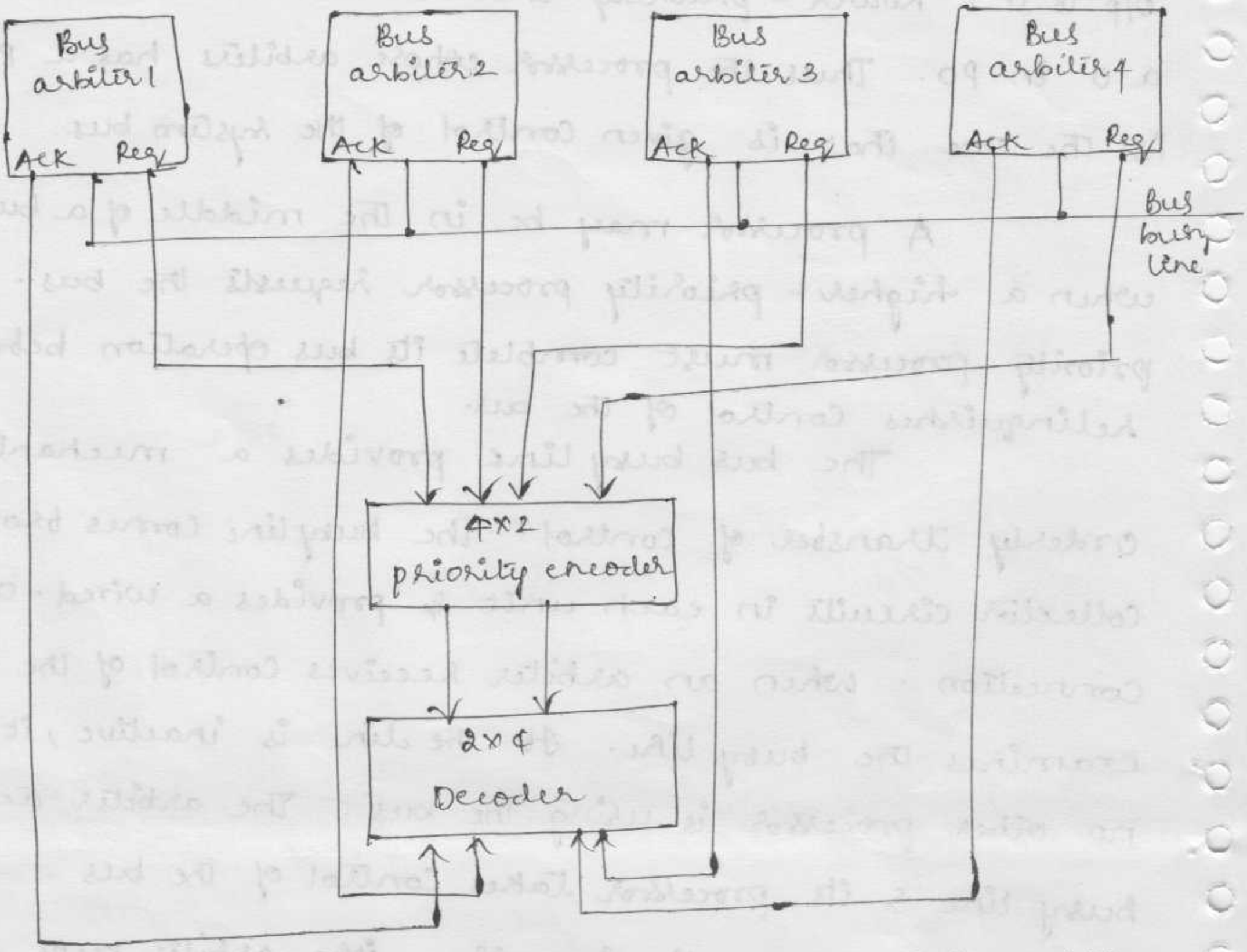
If the line is active, the arbiter keeps examining the busy line while the lower-priority processor that lost control of the bus completes its operation. When the bus busy line returns to its inactive state, the higher-priority arbiter enables the busy line & then required data transfers

parallel Arbitration logic :-

This technique uses an external priority encoder & a decoder can be shown in below fig.

Each bus arbiter in the parallel scheme has a bus request o/p line & a bus acknowledge i/p 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 i/p line is enabled. The bus busy line provides an

orderly transfer of control.



The request lines from four arbiters going into a 4x2 priority encoder. The o/p of the encoder generates a 2-bit code which represents the highest-priority unit among those requesting bus.

The 2-bit code from the encoder o/p drives a 2x4 decoder which enables the proper acknowledge line to grant bus access to the highest-priority unit.

The bus priority-in (BPRN) & bus priority-out (BPRO) are used for daisy-chain connection.

The signals used to construct a parallel arbitration procedure are bus request (BREQ) & priority-in (BPRN). The bus clock (BCLK) is used to synchronize all bus transactions.

The above algorithms are static priority algorithm (9) because the priority of each device is fixed.

Dynamic Arbitration Algorithms :-

This algorithm gives the system the capability for changing the priority of the devices while the system is in operation

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

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 debine an address bor each device connected to the bus. The polling sequence is programmable & the selection priority can be altired.

The LRU algorithm gives the highest priority to the requesting device that has n't used the bus for the longest interval. The priorities are dynamically changed to give every device an opportunity to access the bus.

In 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. Each processor must wait bor its turn to use the bus on a first-in, first-out (FIFO) basis.

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

Interprocessor Communication & Synchronization :

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 I/O channels. In shared memory, is to set aside a portion of memory that is accessible to all processors. The primary use of the common memory is to act as a message center similar to a mail box, where each processor can leave messages for other processors & pick up messages intended for it.

The sending processor structures a request, a message or a procedure & places it in the memory mailbox. Status bits residing in common memory are generally used to indicate the condition of the mail box, whether there is ~~vacant~~ meaningful messages or not.

The receiving processor can check the mail box, to determine if there are valid messages for it. The response time of this procedure can be time consuming.

An efficient procedure is for the sending processor to alert the receiving processor directly by means of an interrupt signal. This can be ~~done~~ used through a S/W-initiated interprocessor interrupt by means of an instruction in the program of one processor which when executed produces an external interrupt condition in a second processor.

In addition to shared Memory, a multiprocessor system may have other shared resources. For eg, a magnetic disk storage unit connected to an IOP may be available to all CPU's. This provides a facility for sharing of systems. A communication path is also established.

To prevent conflicting use of shared resources by several processors there must be providing resources to each processor

This task is given to the O.S.

There are three different organizations for design of O.S for multiprocessors:

1) Master - slave configuration
2) Separate Operating systems
3) Distributed operating system

In a Master - slave mode, one processor is treated as Master which performs all O.S. functions. The remaining processors denoted as slaves which not perform O.S. functions. If slave processor needs an O.S service, it must request it by interrupting the master & waiting until the current program can be interrupted.

In a Separate O.S., each processor can execute the O.S. routines it needs. This organization ~~needs~~ is suitable for loosely coupled systems where every processor may have its own copy of the entire O.S.

In a distributed O.S, the O.S. routines are distributed among the available processors. Each particular O.S. function is assigned to only one processor at a time & the execution may be assigned to different processors at different times. This most suitable for floating operating system.

In a loosely coupled multiprocessor system the memory is distributed among the processors & there is no shared memory for passing information. The communication b/w processors is by means of message passing through I/o channels.

A message is then sent with a header & various data objects used to communicate b/w nodes. There may be a no. of possible paths available to send the message b/w any two nodes. The O.S. in each node contains routing information indicating the alternative paths that can be used to send a message to other nodes

The Communication efficiency of the inter processor n/w depends on the Communication routing protocol, processor speed, data link speed & the topology of the n/w.

Inter processor Synchronization :-

The instruction set of a multiprocessor contains basic instructions that are used to implement Communication & Synchronization b/w co-operating processors. Communication refers to exchange of data b/w different processes. Synchronization refers to the where data used to communicate b/w processors & control information. Synchronization is needed to enforce the correct sequence of processes & to ensure mutually exclusive access to shared writable data.

Multiprocessor systems usually include various mechanisms to deal with the synchronization of resources. Low-level primitives are implemented directly by the h/w. This basic mechanisms for primitives is that enforce mutual exclusion for more complex mechanisms implemented in s/w.

The h/w mechanisms for mutual exclusion is developed i.e., binary semaphore.

Mutual Exclusion with a Semaphore :

The multiprocessor system provide a mechanism that the orderly access to shared memory & other shared resources. It is necessary to protect the changing data. This mechanism has been called 'mutual Exclusion'.

Mutual exclusion in a multiprocessor system to enable one processor to exclude or lock out access to a shared resource by other processes when it is in critical section. A critical section is a program sequence that must complete execution

before another processes access the shared resource.

A binary variable called a semaphore is often used to indicate whether or not a processor is executing in critical section. A semaphore is a s/w controlled flag that is stored in memory.

When semaphore is equal to 1, it means that a processor is executing a critical program, so that shared memory is not available to other processors.

When semaphore is equal to 0, the shared memory is available to any processors.

Testing & setting the semaphore is itself is a critical operation & must be performed as a single indivisible operation. If it is not, two or more processes may test the semaphore simultaneously and then each set it, and enter them into CS. This results an erroneous.

A semaphore can be initialized by means of test & set instruction in conjunction with a h/w lock mechanism. A h/w lock is a processor generated signal that serves to prevent other processors from using the system bus as long as the signal is active.

The test and set instruction tests & sets a semaphore and activates the lock mechanism during the time that the instruction is being executed.

Assume that the semaphore is a bit in the LSB of a memory word whose address is symbolized by SEM. Let the mnemonic TSL designate the "test & set while locked" operation. The instruction

TSL SEM

will be executed in two memory cycles (i.e., first to read & second to write)

This can be represented as follows:

R ← M [SEM] Test Semaphore

M[SEM] ← 1 Set Semaphore.

The Semaphore is tested by transferring its value to a processor register R and then it is set to 1. The value in R determines what to do next. If the processor finds that R=1, it knows that the Semaphore was originally set.

If R=0, it means that the common memory is available. The Semaphore is set to '1' to prevent other processors from accessing memory. The processor can now execute the critical section. The last instruction in the program must clear location SEM to zero to release the shared resource to other processors.

Cache coherence :-

We are using separate caches for each processor is to reduce the average access time in each processor. The same information may reside in a no. of copies in some caches & main memory. The ability of a system to execute memory operation correctly, the multiple copies must be kept identical. Here, there arises a 'Cache coherence' problem.

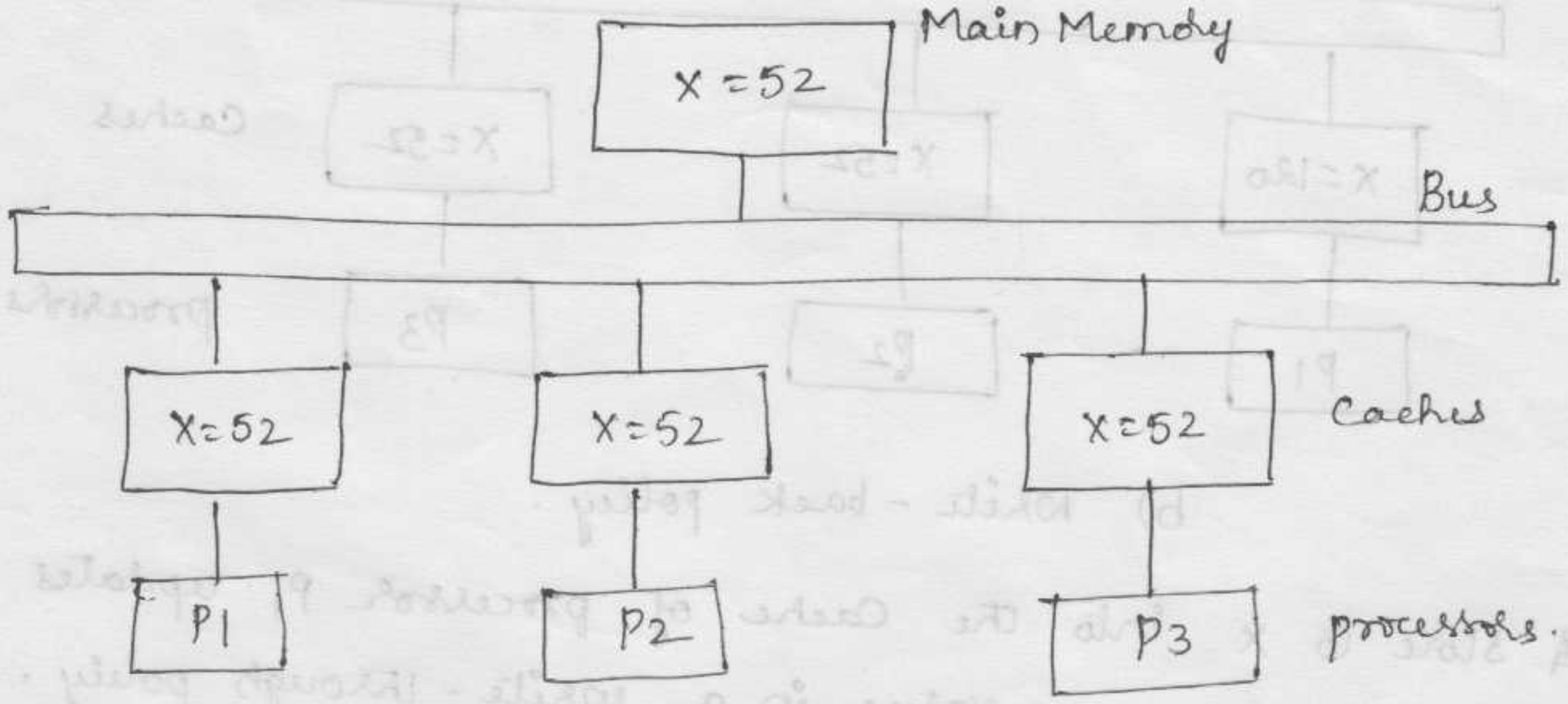
A memory scheme is 'coherent' if the value returned on a load instruction is always the value given by the latest store instruction with the same address. Without a proper solution to the Cache coherence problem, caching can't be used in bus-oriented multiprocessors with two or more processors.

Conditions for Incoherence :-

Cache coherence exists in multiprocessors with private caches because of the need to share writeable data. Read-only data can be safely placed with out Cache coherence mechanisms.

To Illustrate the problem,

Consider the three-processor configuration with private caches as shown below.

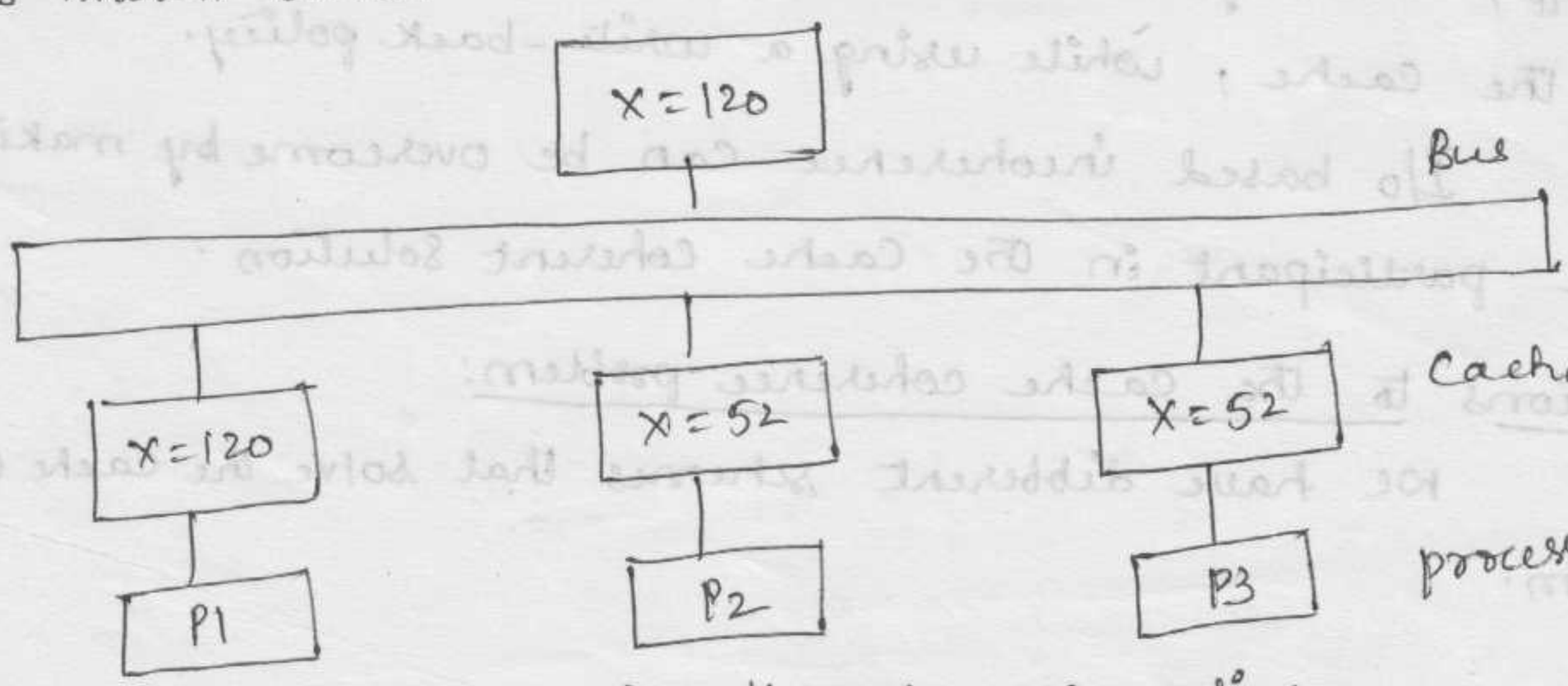


During the operation on element X from main memory is loaded into three processors P1, P2, P3. It is also copied into the private caches of the three processors.

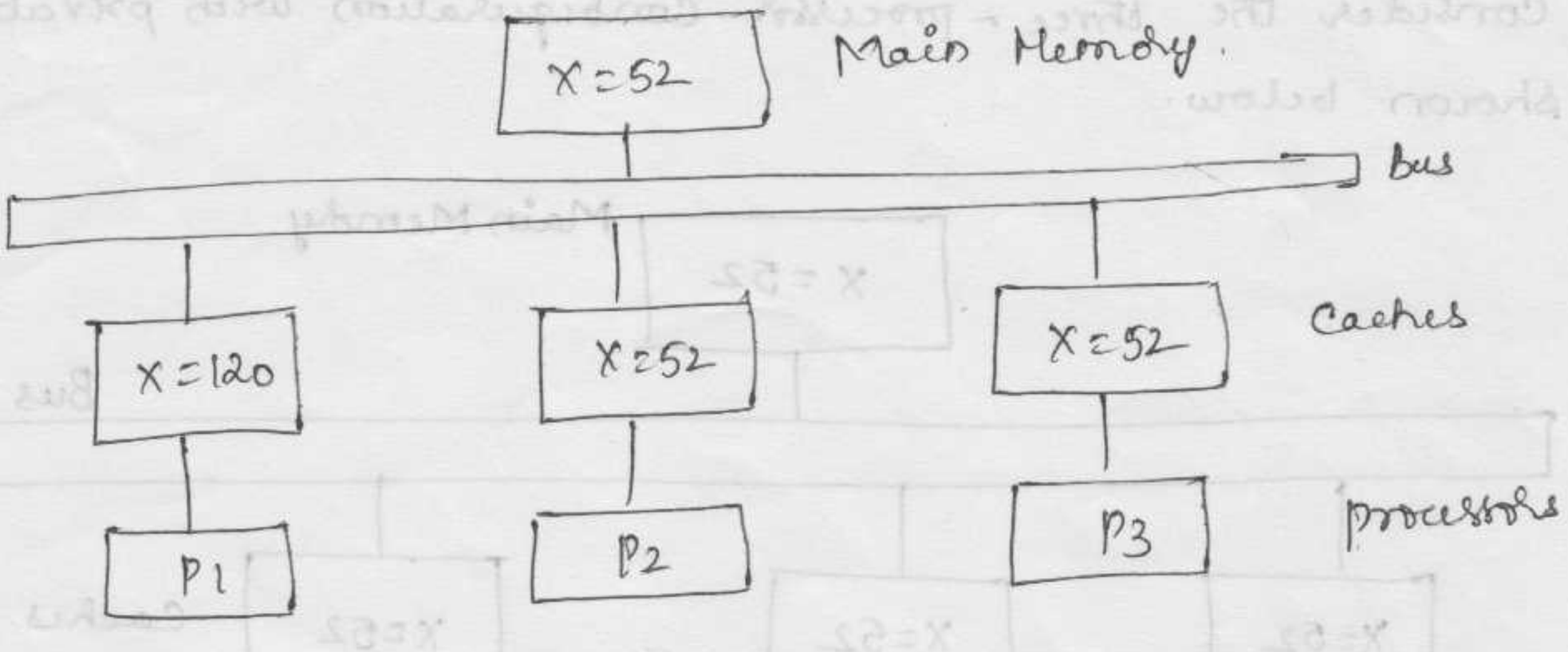
We assume that X=52, is load on X to three processors results in consistent copies in the caches & main memory.

If one of the processors performs a store to X, the copies of X in the caches become inconsistent. A load by other processors will not return the latest value.

Depending on the memory update policy in the cache, the main memory may also be inconsistent with respect to the cache. This shown below:



a) Write-through cache policy.



b) write - back policy.

In fig@ A store to x into the cache of processor P, updates the memory to the new value in a write - through policy. A write - through policy maintains consistency b/w memory & the originating cache, but the other two caches are inconsistent since they hold the old value.

In fig (b) Main Memory is not updated at the time of the store. The copies in other two caches & main memory are inconsistent.

Another configuration that may cause Consistency problems is a DMA. During DMA i/p, the locations in main memory that also reside in cache with out updating the cache. During DMA o/p, memory locations may be read before they are updated from the cache, while using a write - back policy.

I/o based incoherence can be overcome by making the IOP a participant in the Cache Coherent Solution.

Solutions to the cache coherence problem:

We have different schemes that solve the Cache Coherence problem.

→ A scheme that each processor & have a shared cache memory associated with main memory. Every data access made shared to cache. This increases the memory access time.

→ For performance considerations, it is desirable to attach a private cache to each processor. One scheme that has been used allows only non shared & read-only data to be stored in caches. Such items are called cacheable. Shared writeable data are non cachable. The system h/w makes that only cachable data are stored in caches & the non cacheable data remains in main memory. This methods restricts the type of data stored in caches & introduces an extra s/w.

→ A scheme that allows writable data to exist in at least one cache is a called centralized global table. The status of memory blocks is stored in the central global table. Each block is identified as Read-only (RO) or read & write (RW). All caches can have copies of blocks identified as RO. only one cache can have a copy of an RW block.

. The cache coherence problem can be solved by means of a combination of s/w & h/w or by means of h/w-only schemes.

H/w-only solutions are handled by the h/w automatically & have the advantage of higher speed.

In h/w solution, the cache controller is specially designed to monitor all bus requests from CPU's & IOP's.

Depending on the Method used, they must either update or invalidate their own cache copies when a match is detected. The bus controller that monitors this action is referred as 'Snoopy cache controller'. This is a h/w unit designed to

maintain a bus-watching mechanisms over all the caches attached to the bus.

All the snoopy controllers watch the bus for memory store operations. The local snoopy controllers in all other caches check their memory to determine if they have a copy that has been overwritten. If a copy exists in a remote cache, the 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 & main memory & remove it from all other caches.