Bit Vector Directory Organization Operation

Consider a ~~invalidate~~ protocol with three stable cache states. On a read miss or write miss at node-i, the local communication assist or controller looks up the address of the memory block to determine, if the ~~—~~ is local or remote. If it is remote, a network transaction is sent to the home node for the block. There the directory entry for the block is looked up, and then assist at the home may treat the miss as follows:

- If the dirty bit is OFF, then the assist obtains the block from main memory, supplies it to the ~~requester~~ requestor in a reply network transaction, and turns the i-th presence bit, presence[i], ON.
- If the dirty bit is ON, then the assist responds to the requestor with the identity of the node whose presence bit is ON. (i.e the owner of the dirty node).

Then requestor then sends a request network transaction to that owner node. At the owner, the cache changes its state to shared and supplies the block to ~~both~~ the requesting node, which stores the block in its cache in shared state, as well as to main memory of the home node.
At memory, the dirty bit is turned OFF and presence[i] is turned ON.

A write-miss by processor i goes to memory and is handled as follows:

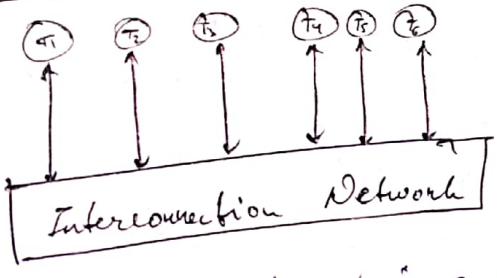
- If the dirty bit is OFF, then main memory has a clean copy of the data. Invalidation request transactions must be sent to all the nodes j for which presence[j] is ON. Assuming a strict request-response scenario, the home node supplies the block to the requesting node i together with the presence bit vector. The directory entry is cleared,

leaving only presence[i] and the dirty bit ON (if the request is an upgrade (from shared) instead of a read exclusive, an acknowledgement containing the bit vector is returned to the requestor instead of the data itself). The assist at the requestor sends invalidation requests to the required nodes and waits for the invalidation acknowledgement transactions from the nodes, indicating that the write has completed with respect to them. Finally, the requestor places the block in the cache in dirty state.

- If the dirty bit is ON, then the block is first recalled from the dirty node (whose presence bit is ON), using network transactions, with the home and the dirty node that cache changes its state to invalid, and then the block is supplied to the requesting processor, which places the block in its cache in dirty state. The directory entry is cleared, leaving only presence[i] and the dirty bit ON.

Interconnection Network

Functional view of an interconnection network,



Terminals ($T_1, \ldots T_6$) are connected to the network using channels.

→ An interconnection network is a programmable system, that transports data between terminals.

When terminals T_3 wishes to communicate some data with terminal T_5, it sends a message containing the data, into the network and the network delivers the message to T_5.

→ The network is programmable in sence that it makes different connections at different points in time.

The network in the figure may deliver a message from T_3 to T_5 in one cycle and then use the same resources to deliver a message from T_3 to T_1 in the next cycle.

The network is a system because it is composed of many components: buffers, channels, switches and controls that work together to deliver data.

Topology:

The interconnection network is implemented with a collection of shared router nodes connected by shared channels. The connection pattern of these nodes define the networks topology. A message is then delivered between terminals by making several hops across the shared channels and nodes from its source terminals to its destination terminal.

Routing:

Once a topology has been chosen, there can be many possible paths (sequences of nodes and channels) that a message could take through the network, to reach its destination. Routing determines which of these possible paths a message actually takes.

Flow Control: Flow control dictates which messages get access to particular network resources over time.

Circuit Switching: Circuit switching is a form of bufferless flow control that operates by first allocating channels to form a circuit from source to destination, and then sending one or more packets along the circuit. When no further packets need to be sent, the circuit is deallocated.

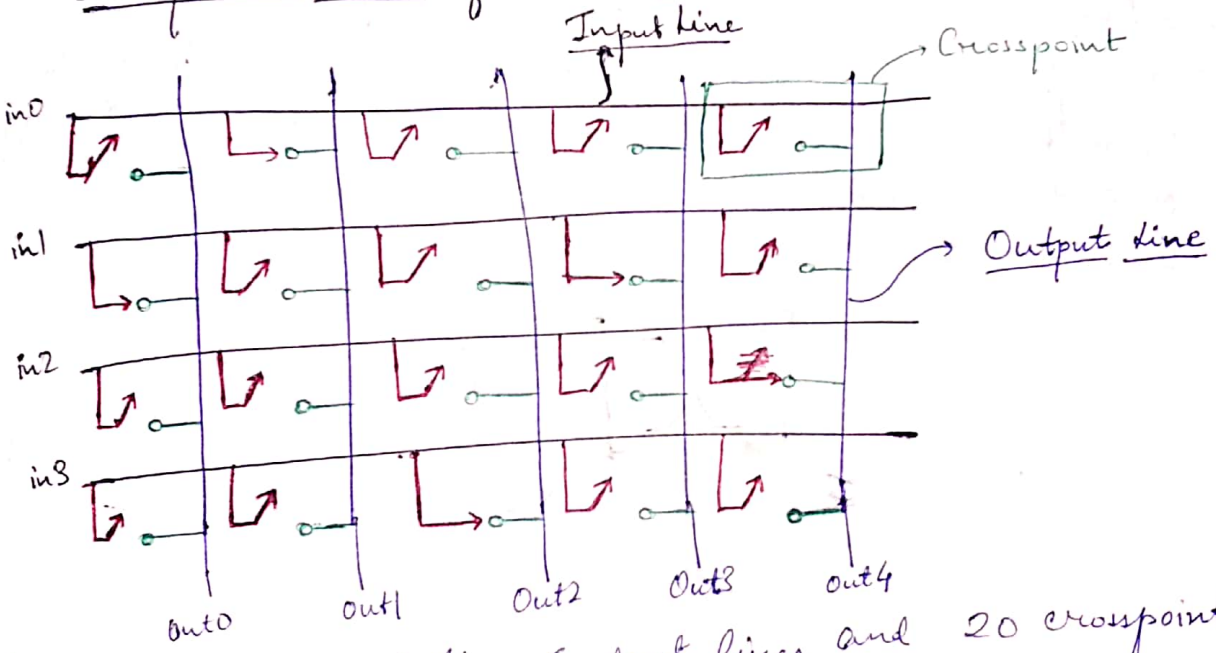
Non-blocking Networks: A network is said to be non-blocking if it can handle all circuit requests that are a permutation of inputs and outputs. That is, a dedicated path can be formed from each input to its selected output without any conflicts (shared channels). Conversely, a network is *blocking* if it CANNOT handle all such circuit requests without conflicts.

STRICTLY NON-BLOCKING NETWORKS: A network is strictly non-blocking if any permutation can be set up incrementally, one circuit at a time without the need to reroute (or rearrange) any of the channels that are already set up.

REARRANGEABLY NON-BLOCKING (OR REARRANGEABLE) NETWORKS: Such a network can route circuits for arbitrary permutations, but incremental construction of a permutation may require rearranging some of the early circuits to permit later circuits to be set up.

CROSSBAR NETWORKS

Conceptual Model of a 4×5 Crossbar



It consists of 4 input lines, 5 output lines and 20 crosspoints. Each output may be connected to at most one input, while each input may be connected to any number of outputs. This switch has inputs 1, 0, 3, 1, 2 connected to outputs 0, 1, 2, 3, 4, respectively.

An n×m crossbar or crosspoint switch directly connects n-inputs to m outputs with no intermediate stages. In effect, such a switch consists of m n:1 multiplexers one for each output.