RV32IM Extended features Network on Chip (NoC)

Damsy De Silva (E/16/069)
Shirly Ekanayaka (E/16/094)
Buddhi Perera (E/16/276)

Table of Contents

1. Introduction

Under extended features we designed a Network of Chip to interconnect 16 RV32IM CPU instances using a mesh network. Mesh network is used by the CPU instances to communicate with other CPU instances and to access the main memory through the memory controllers. Figure 1 shows the basic design of the NoC.

As shown in Figure 1, the main hardware components of the NoC,

- Nodes
- Routers
- Main memory
- Memory Controller

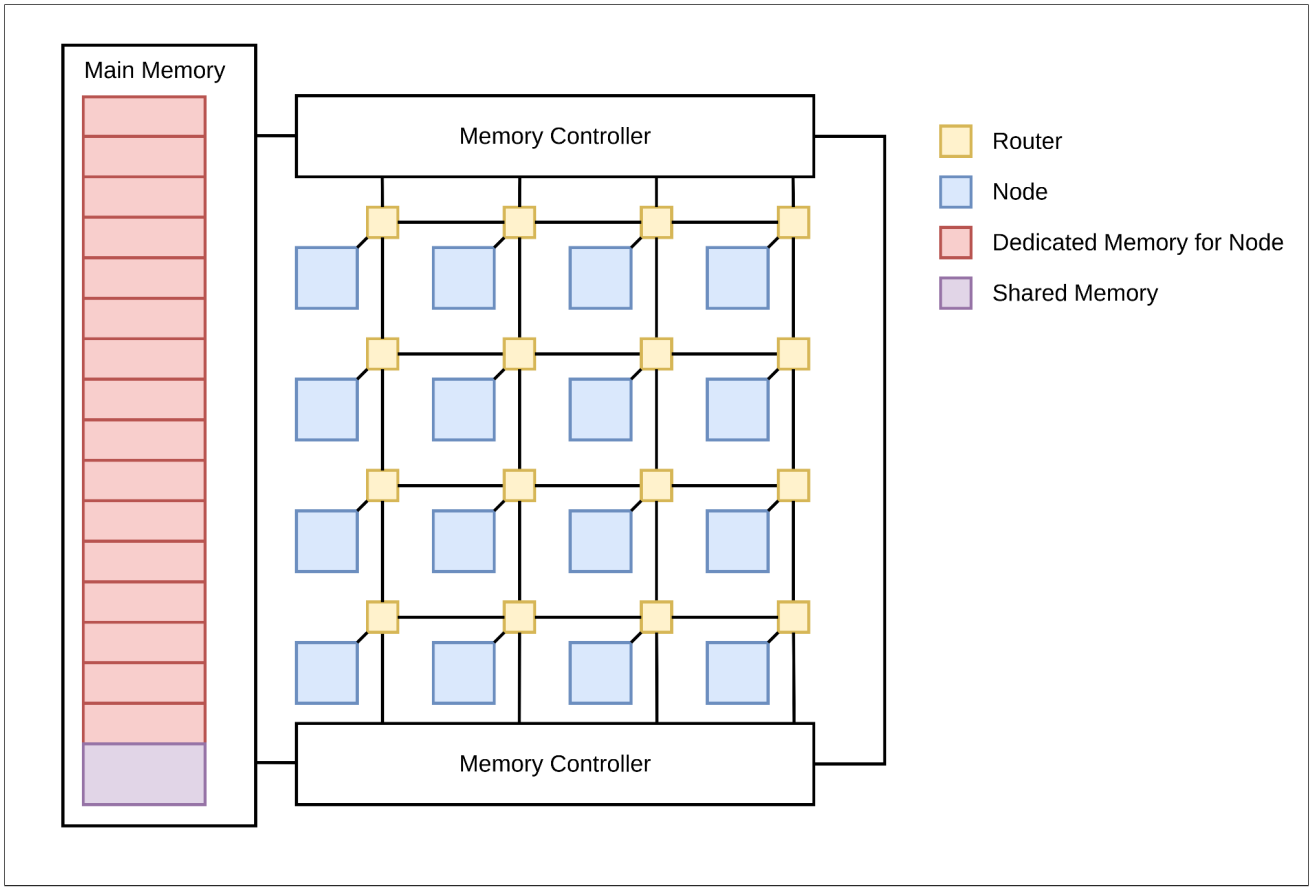


Figure 1 : Overview of the NoC

2. Nodes

Nodes are RV32IM pipeline CPU instances. There are 16 nodes in this design and it can be scaled up. Node contains the following components,

- The core CPU components
  - ALU
  - Register File
  - Control Unit
  - Program Counter
  - Pipeline Registers
- Local Instruction cache
- Local Data Cache
- Network Interface

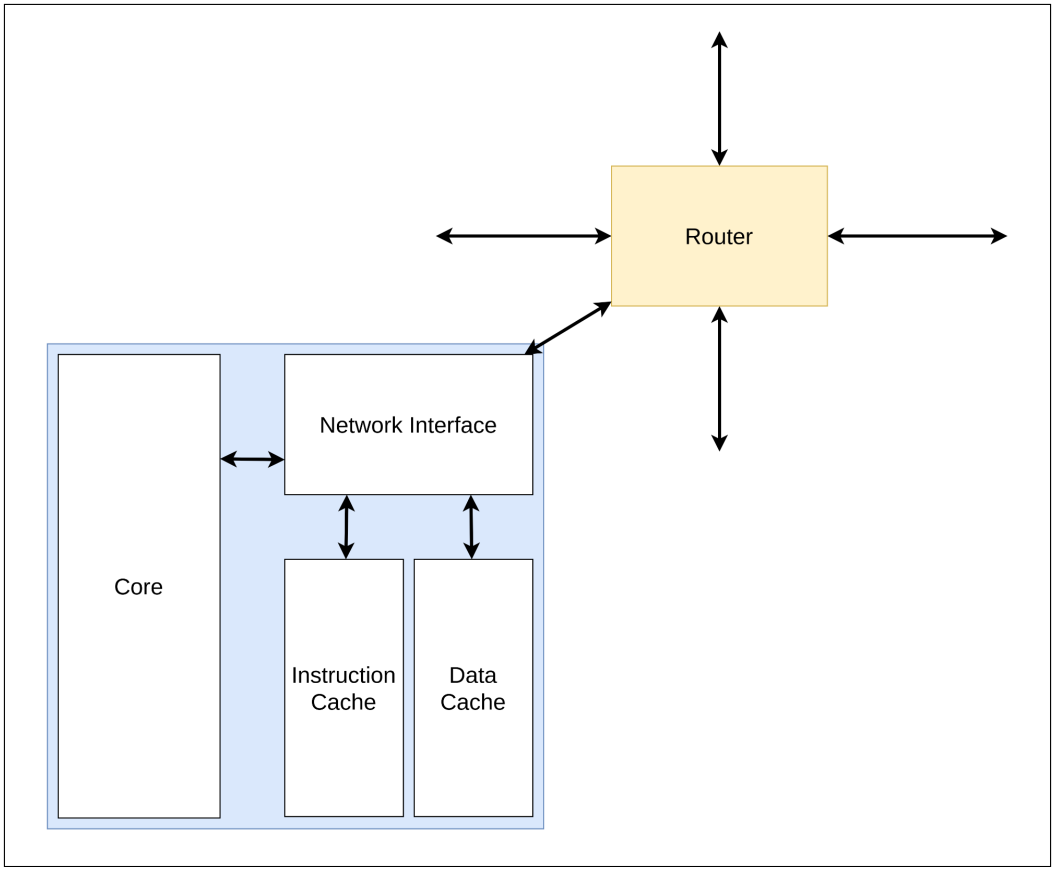


Figure 2 : Node Structure

2.1. Network Interface

Network interface is responsible for the interconnection of the node containing the CPU core and the local caches with the router. This interface provides necessary data to the router's local node processing unit .

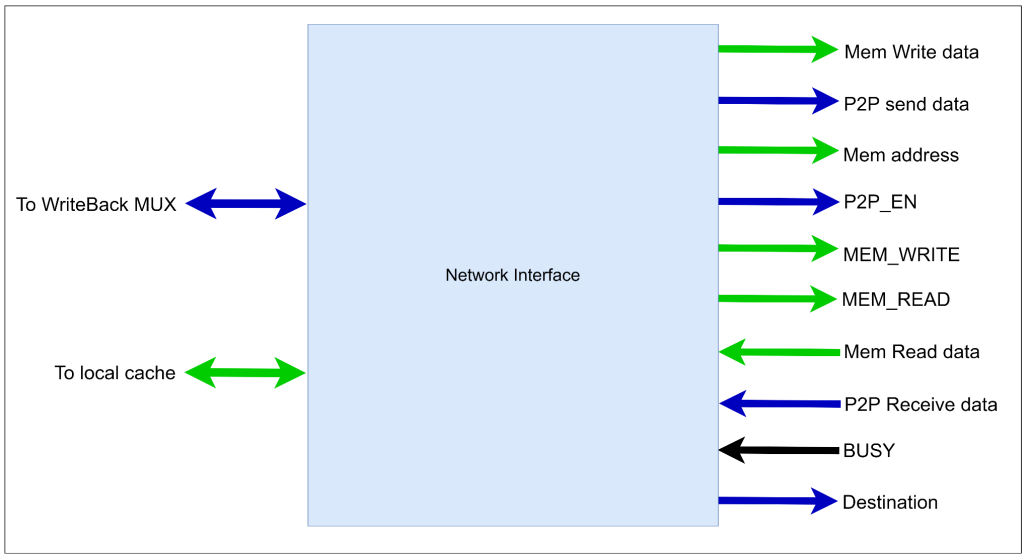


Figure 3 : Network Interface

Inputs from the router's local node processing unit,

- BUSY

  This signal is asserted when the CPU needs to stall. As an example when P2P data receiving CPU should be stalled. Also When the router buffer is filled and the CPU needs to send data to the router, this signal will be asserted.
- Mem Read Data

  This is a 4 byte long bus. This bus sends the read data which is coming from the Memory Controller.
- P2P Receive data

  This also a 4 byte data bus which provides the receive data which is coming from another node in the network via P2P communication.

Outputs to the router's local node processing unit,

- Mem Write Data

  This is a 16 byte data bus which will provide the memory write data to the router.
- P2P send data

  This is a 4 byte long data bus. This will provide the data that should be sent to another node in the network via P2P communication.
- Mem address

  This bus indicates the address that particular memory access should happen. Bus length is 4 byte.
- P2P_EN

  When P2P instruction is executing, the CPU control unit will assert this signal.
- Mem Write
- Mem Read
- Destination

  This bus will indicate the destination address that P2P communication should happen. Bus length is 4 bits long.

3. Communication

3.1. Packet Structure

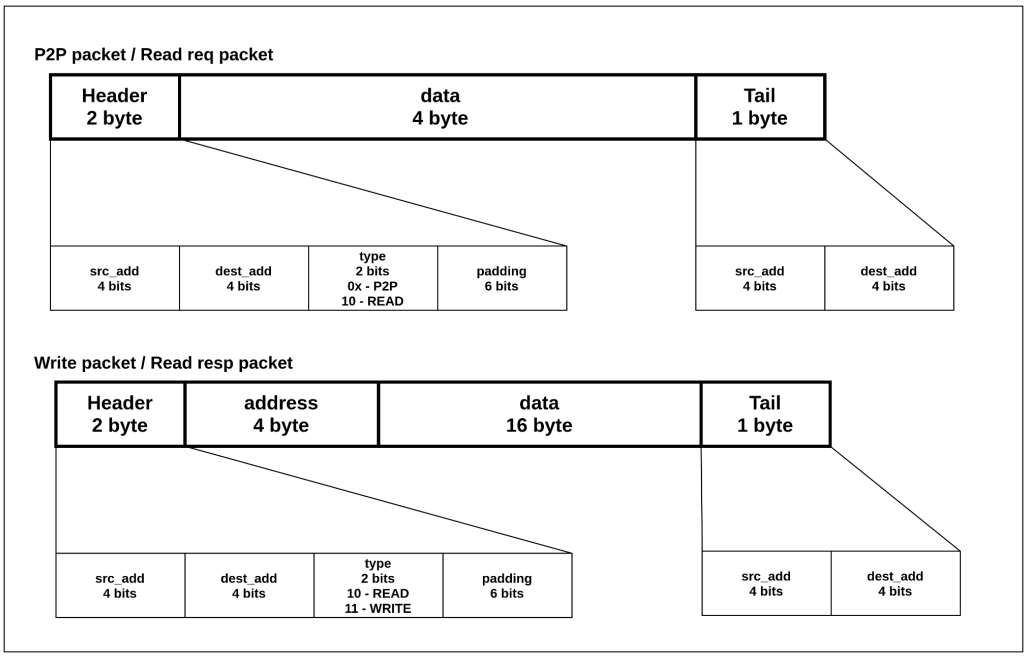


Figure 4 : Packet Structure

As shown in the figure above there are 2 types of packets transmitted in the network.

- Point to point communication packet and memory read request packet uses the same type of packet which is 7 bytes long.When two processing elements (nodes) need to share some information it uses point to point packet structure. So the data section in the packet structure refers to the data that is shared between nodes. When node wants to read from the main memory, it sends a read request packet. So the data section in the packet structure refers to the address that node wants to fetch the data from. When the memory controller receives the read request packet it fetches the data from the main memory and replies back with a read response packet.

- Read response packet and memory write packet have similar structure with a length of 23 bytes.

Both packet structures have header and tail sections. In the header packet it contains the destination address,the source address and 2 bits to indicate the type of the packet. Encodings of the header packet for identification of the type of packet is shown below.

0x - p2p packet
10 - read packet
11 - write packet

The tail section has the destination address and the source address. This information is used to free the allocated resources at the router when the entire packet is sent to the receiving router.

These packets are separated into 8 bit segments and each of these segments is called a flit. The communications, routing, buffering in the mesh network happens at the level of flits.

3.2. Point to Point Communication

For Point to Point communication the RV32IM ISA needs an extension with 2 new instructions for sending and receiving between P2P.

- **SND <reg address> <dest node>**

Above instruction enables us to send some data stored in a register file to another node. When executing this instruction, it first reads data from the register file and supplies the data to the network interface in the memory access stage in the pipeline. So the network interface provides the data to the local processing unit of the router to create the packet. This instruction is not a blocking instruction. The data path of the instruction is shown in the below figure.

- **RCV <reg address> <dest node>**

When some node in the NoC network sends data to another node, the receiving node should wait until the data is received and store the data to a register file. Since the receiving node has to wait until the data arrives, RCV instruction is a blocking instruction.

Destination node address is encoded as an immediate value in the instruction. The data path of the RCV instruction is shown in the below figure.

3.3. Communication with the Memory Controller

When a particular node wants to access a memory for reading and writing that will happen though the network via the memory controller. When a node needs to perform a read operation it should send a read request packet as mentioned earlier. Then the memory controller will reply back with a read response packet. Similarly when a node needs to perform a write operation then it should send a write packet. Then Memory controller will write the required data to the corresponding memory region depending on the request made by the node.

3.4. Datapath for SND <reg address> <dest node>

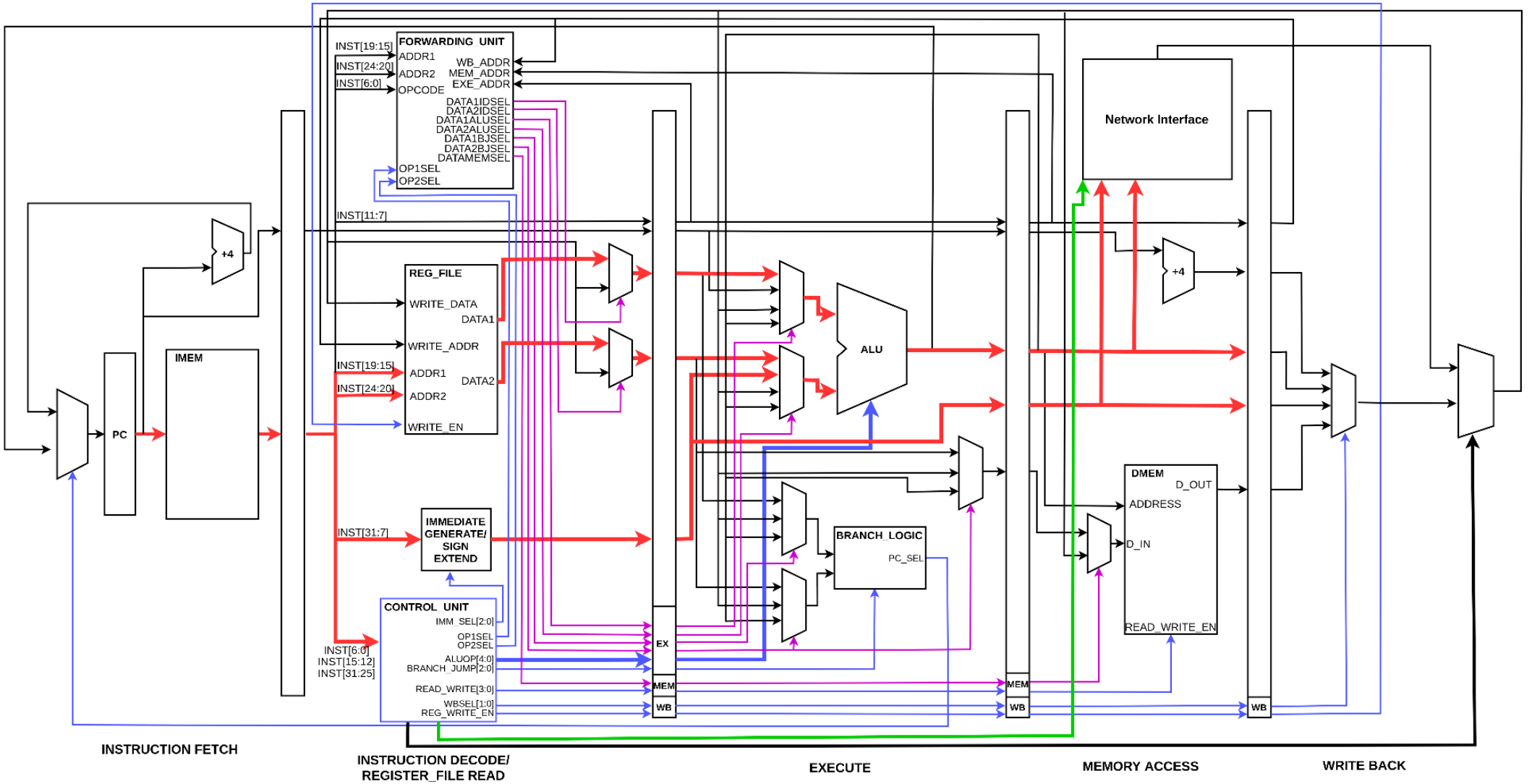


Figure 5 : Datapath for SND instruction

3.5. Datapath for RCV <reg address> <dest node>

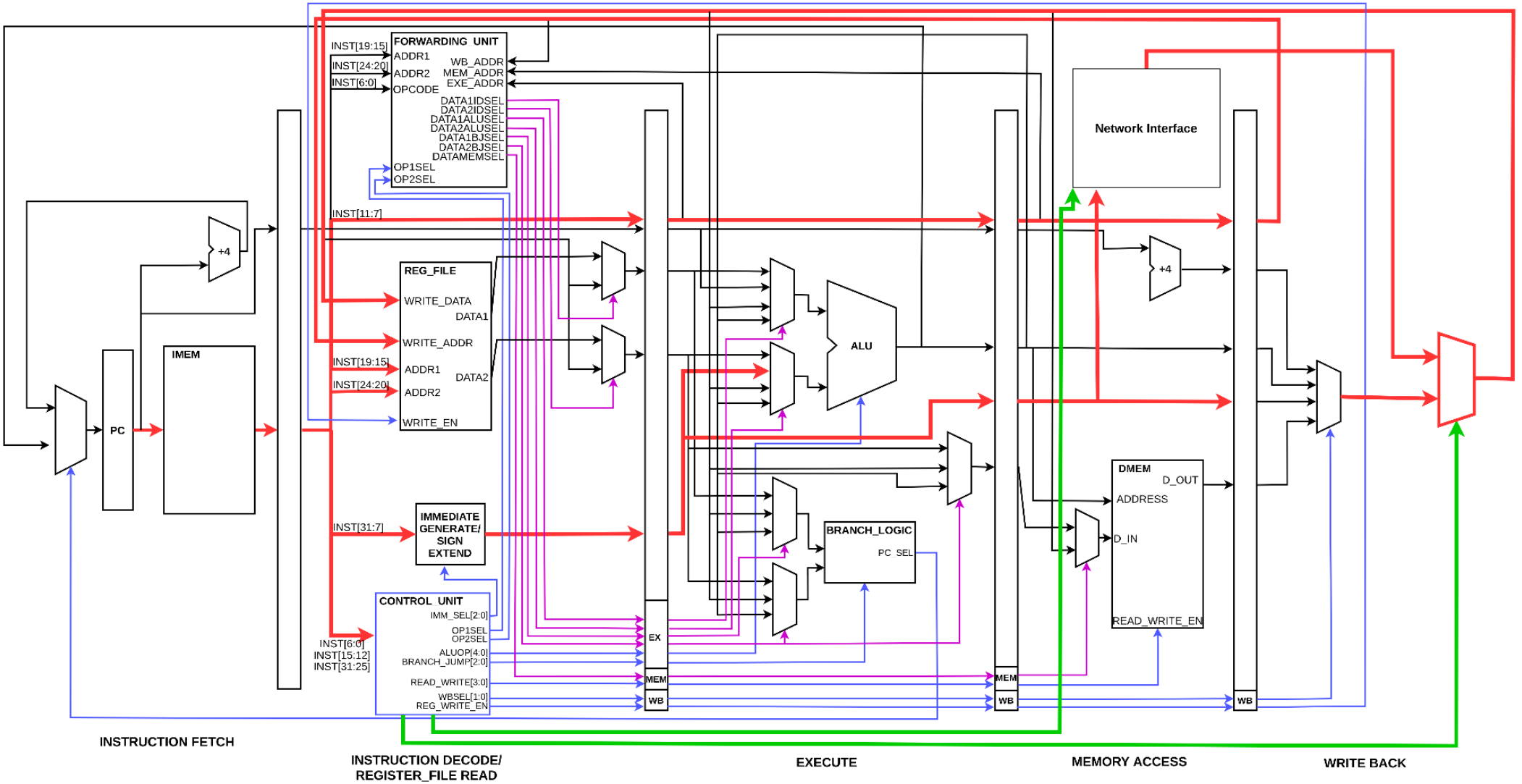


Figure 6 : Datapath for RCV instruction

4. Routers

Routers are the main components in the mesh network. There are 16 routers in the network and they are positioned at the junctions in the network. The main functions performed by the router are,

- Creating the packets
- Disassembling the packets
- Determining the route in which the packets has to be forwarded
- Flow controlling

Each router has 4 ports to transfer the packets to neighboring routers in North, South, West, East directions and 1 port to transfer data with the network interface in the local node. The 4 ports connected to other routers have 4 channels for,

- Input data - 8 bit channel
- Output data - 8 bit channel
- Input control - 1 bit
- Output control - bit

Data channels are for transferring flits which are of 8 bits and control channels are for flow controlling. Each data input is connected to a buffer which can store 4 bytes and the buffer is connected to a 5x5 crossbar switch to direct the packets to the corresponding output. The reason for buffering the input is because the data flits have to be kept on hold until the route is computed using the header flits and buffers are used in flow controlling.

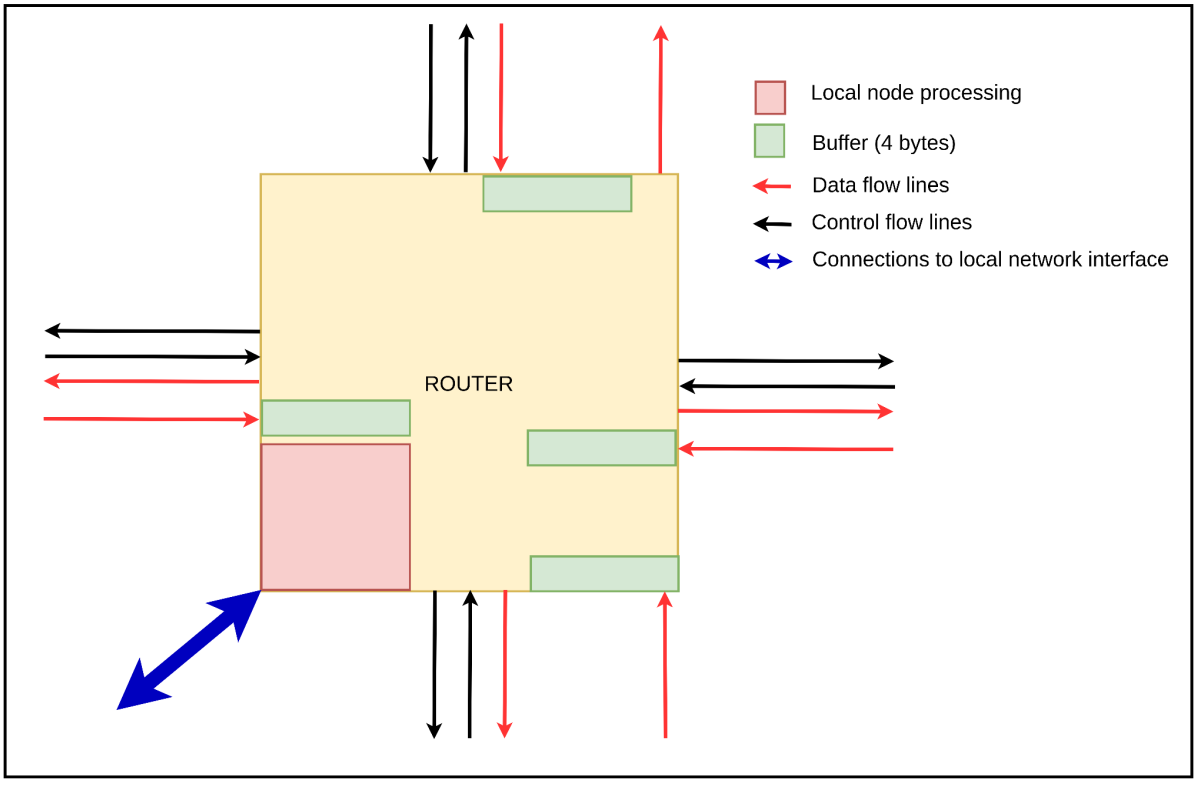


Figure 7 : Router Structure

4.1. Flow Control Mechanism

Flow control is used to prevent the loss of flits when transmitting. This mechanism ensures that the receiver buffer has enough empty bits in the buffer to store the incoming flits. Wormhole flow control is used as the flow control mechanism where the flow control is done at the flit level.

If the receiving router has empty 8 bits in the buffer it will assert its ON-OFF Signal Out so that the sending router will send a flit. Since the flits are of size 8 bits, a minimum of 8 bits is required in the buffer. When the buffer is full, the receiver will deassert its ON-OFF Signal Out so that the sender will stop sending. This mechanism will not overwhelm the receiver and flits will not be lost during the transmission.

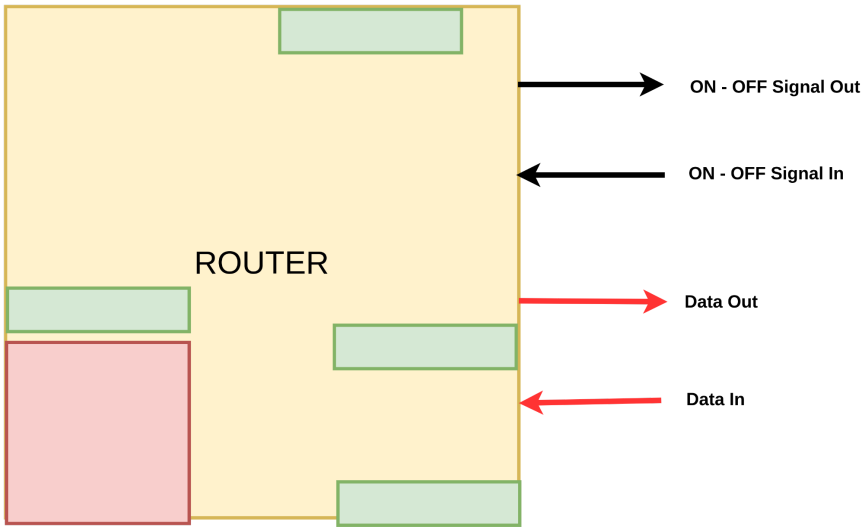


Figure 8

However this flow control mechanism suffers from head of the line blocking. In order to overcome the head of the line blocking, virtual channel concept can be used. Each physical channel will be separated into several virtual channels and the flits of a particular packet will be sent with an allocated virtual channel. This mechanism will make the router complex therefore we thought of using the wormhole flow control mechanism.

When a flit arrives, the routing computation unit will send a channel request signal to switch allocation. If the downstream buffer at the neighboring router has vacant space, it will allocate the channel in the crossbar and flit will route through the crossbar switch towards the next router at the switch traversal stage.

4.2. Router Design

Inside the router the flits will follow 4 stages.

1. Buffer - Flits are buffered in this stage.
2. Route Determination - At this stage the forwarding path will be determined by using the header flits with the help of the route determination logic.
3. Switch Allocate - Allocation the switch path in the 5x5 crossbar switch by looking at the forwarding path.
4. Switch Traversal - Once the switch path is allocated, the flits will flow from the buffer to the egress port. After all the flits of a packet are being sent, the allocated path will be freed by looking at the tail flits.

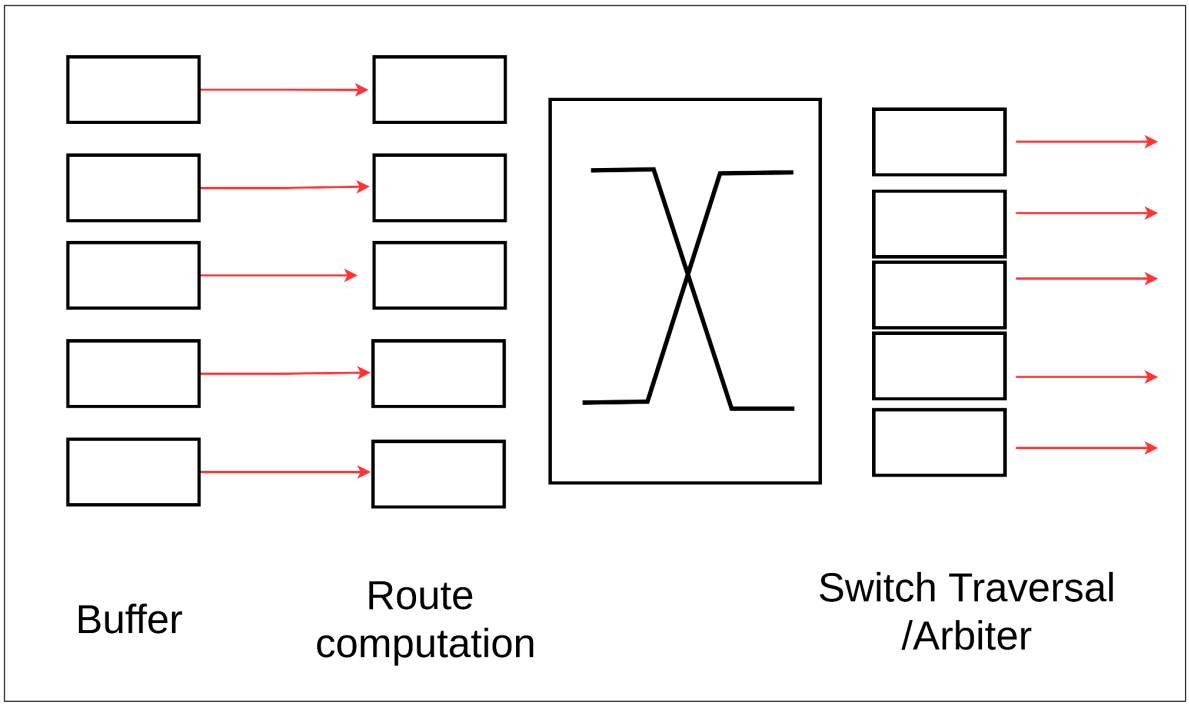


Figure 9 : Router Design

4.2.1. Route Determination Logic

Router contains a route determination logic to decide the routing direction. Each router has a static routing table with entries to all other routers, the local node and the relevant memory controller. Altogether the routing table has 17 entries. Since these are static routes the entries have to be predetermined. These entries are predetermined with the North Last algorithm.

In the north last algorithm the north direction is determined at last in order to prevent flits moving in a loop. If the flits encounter a loop the flit will not be delivered to the destination. The north last algorithm will prevent this from happening.

Once the entire header (first 2 flits) of the packet is received the router determination logic will determine the forwarding path and allocate the path in the 5x5 crossbar switch. Then the flits can be sent in that path.

If two packets are to be sent in the same path, a round robin arbiter is used to allocate the switch path. By using an arbiter it is possible to allocate the switch path only for one packet.

4.2.2. Local Node Processing Unit

This unit is responsible for creation and disassembling the packet. Packets will be created based on the signals coming from the network interface and when a packet is received to the local node, this unit will disassemble the packet and send the data to the respective unit in the local node.

Header and tail sections are generated as shown in the figure below. Type resolver outputs the corresponding type of the packet depending on the P2P_EN, MEM_READ and MEM_WRITE signals. After creating the separate sections of the packet each section is enqueued to a Packet Processing Queue, then forwarded to the output buffer.

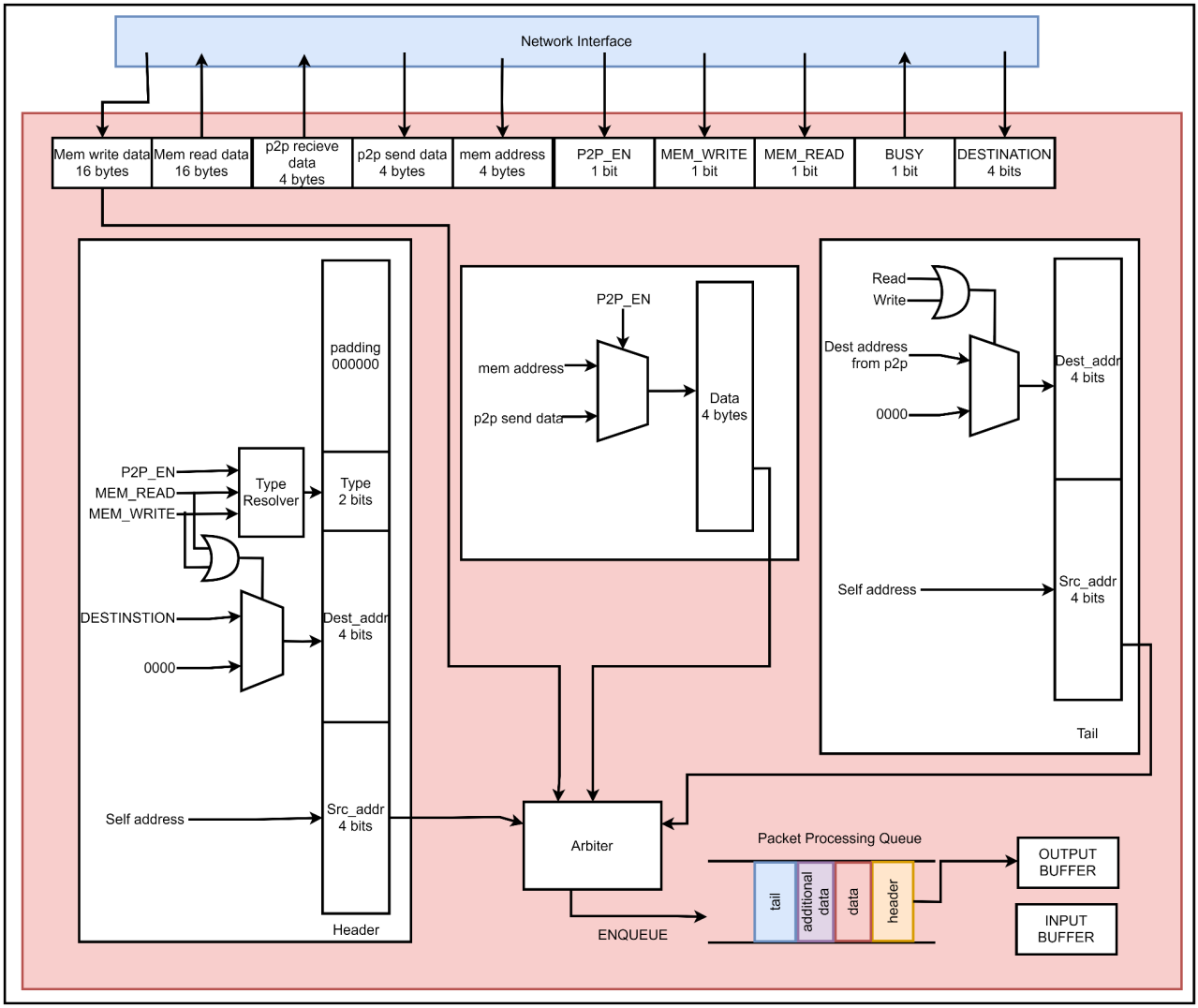


Figure 10 : Node Processing Unit

5. Main Memory

Each node has been allocated a dedicated memory of 1kB. And there is a 1kB memory space shared between all the nodes in the system. Since there are 15 nodes and 1 shared space, the total main memory in our design is 17kB.

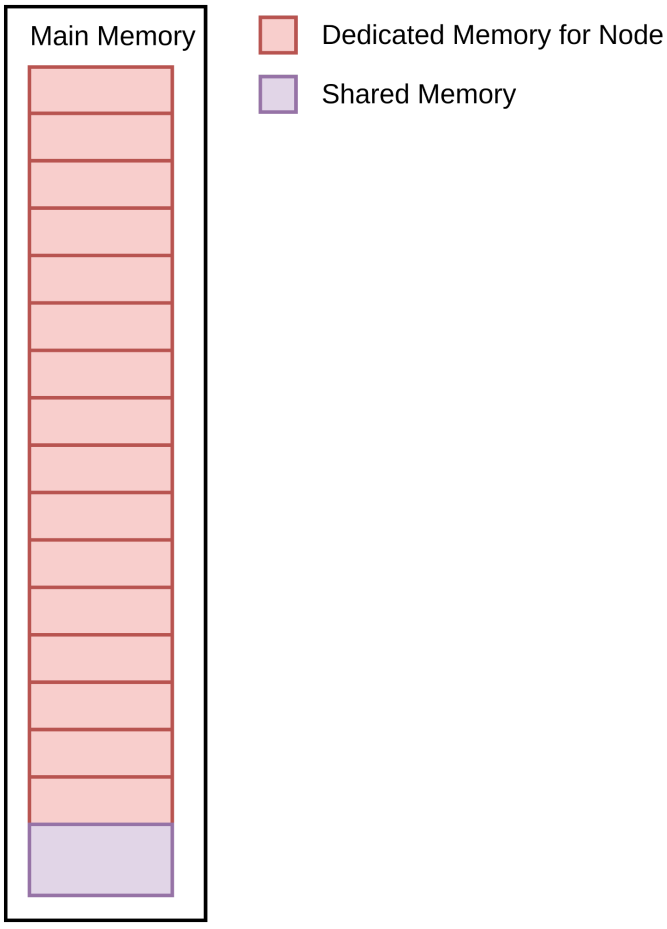


Figure 11 : Main Memory Design

6. Memory Controller

Memory controller is responsible for serving the memory requests from the nodes and maintaining the cache coherency. This unit will control which node is writing or reading the memory. Each node has 2kB of address space (1kB of dedicated space and 1kB of shared space).

Memory controller receives the node address and the memory address that node needs to access. Node address is 4 bits long and it generates from X and Y coordinates of the node in the network. We use these X, Y coordinates to map the node address space to the actual address space in the main memory.

First Main memory is divided into 4kB of blocks. X coordinate of the node is used to access the 4kB block. After that, Y coordinate is used to pick the correct region of the memory that node belongs to (1kB slot).

Comparator is used to determine if the node accesses the memory that lies in the shared space. As you can see in the figure below it parallely calculates the dedicated memory location and the shared memory location, then picks the correct mapped address by using the result from the comparator. Then the memory controller accesses the relevant data from the memory and serves the data to the requested node.

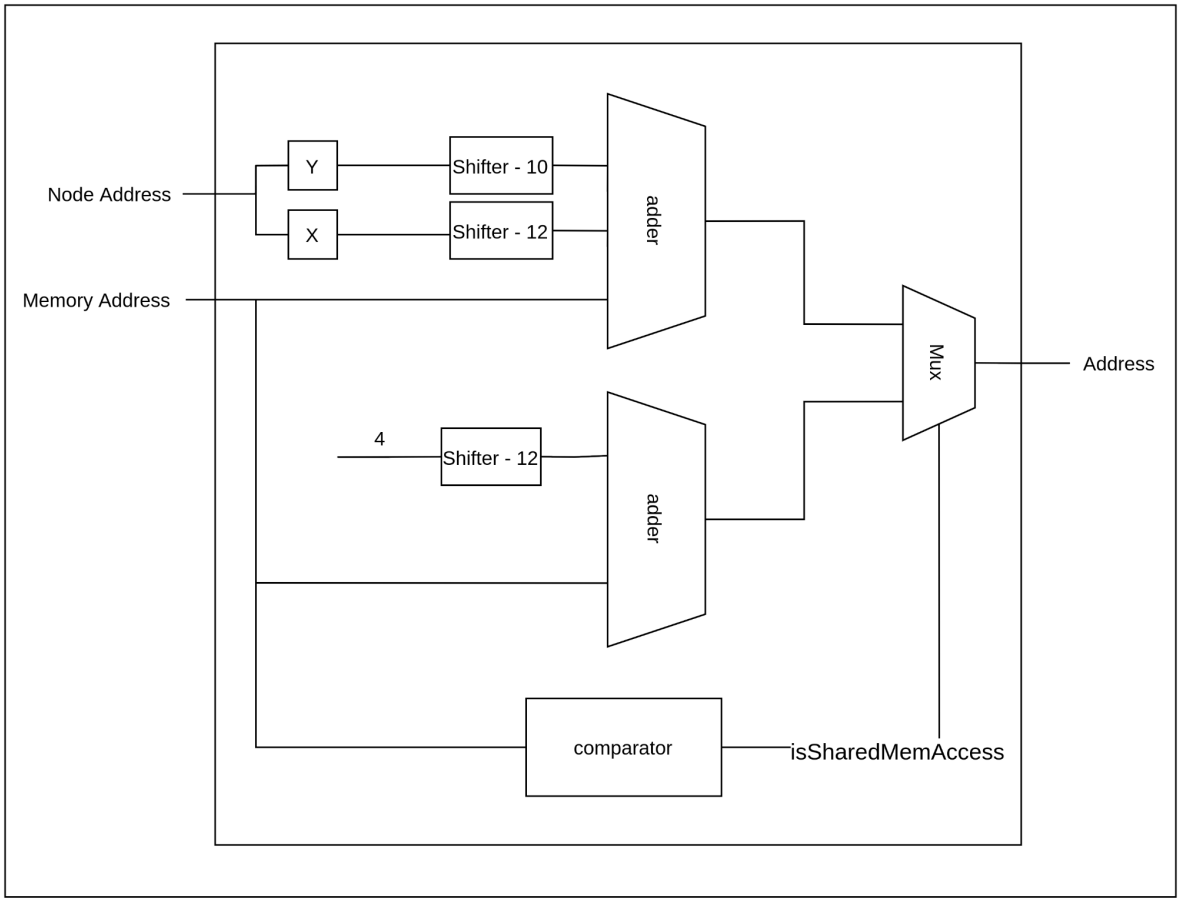


Figure 12 : Memory Controller Design

6.1. Cache Coherency

Cache coherency is maintaining the consistency of data in all the caches. Since there is a shared memory which can be accessed by all the nodes, it is required to maintain cache coherency in the local caches of each node. If there is no cache coherency the incorrect values will be used in computations. The cache coherency has to be maintained only for the shared address space.

The mechanism used to maintain the cache coherency in our design is the directory based method. The snoop bus method cannot be used in this design because the number of nodes are high and it does not suit the mesh networks.

In the directory based cache coherence mechanism there is a directory and the directory will contain entries for each shared block in the shared memory space. Since the RV32IM pipeline CPU's cache block size is 16 bytes, the directory will contain an entry for each 16 byte block in shared memory(1kB) resulting in 64 entries in the directory. This directory is placed at the memory controller. Each entry will keep a record of the node address that contains the valid cache block.

In order to maintain the sync between the two directories in two memory controllers, a snoop bus is connected in between the memory controllers to exchange the messages related to updates in the directory controller.

Following figure shows the NoC with the directory at the memory controller.

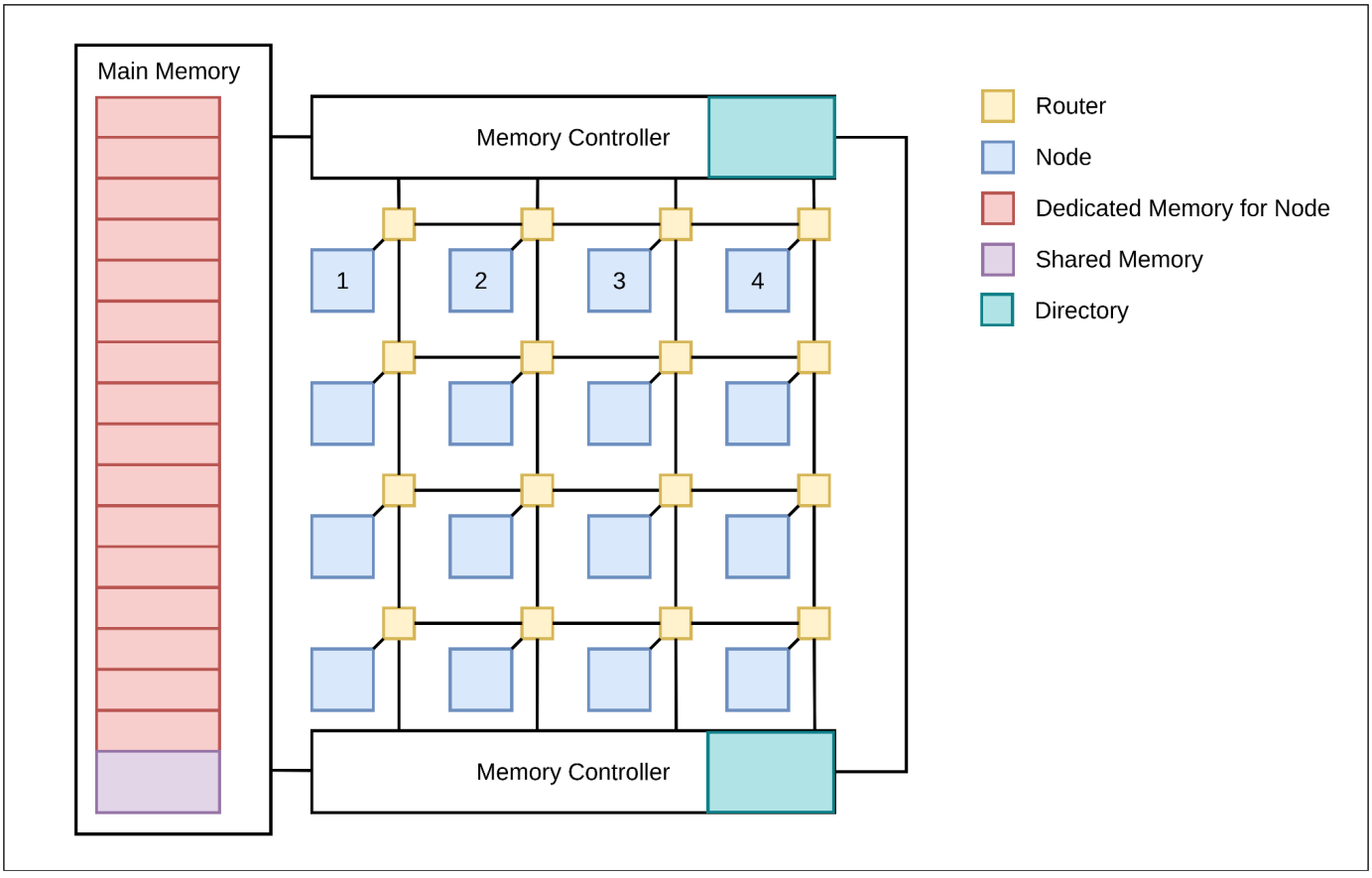


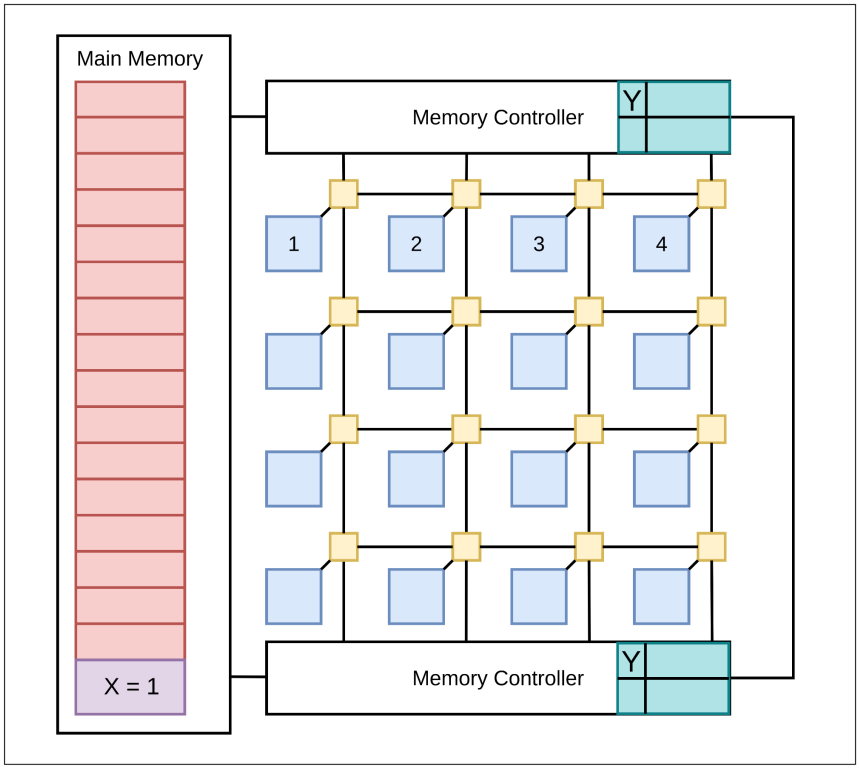
Figure 13: NoC architecture with directory

Consider the following scenario,

1. X memory location Y block contains value .
2. Node 1 requests X.
3. Node 2 requests X.
4. Node 1 updates the value in X.
5. Node 2 updates the value in X.

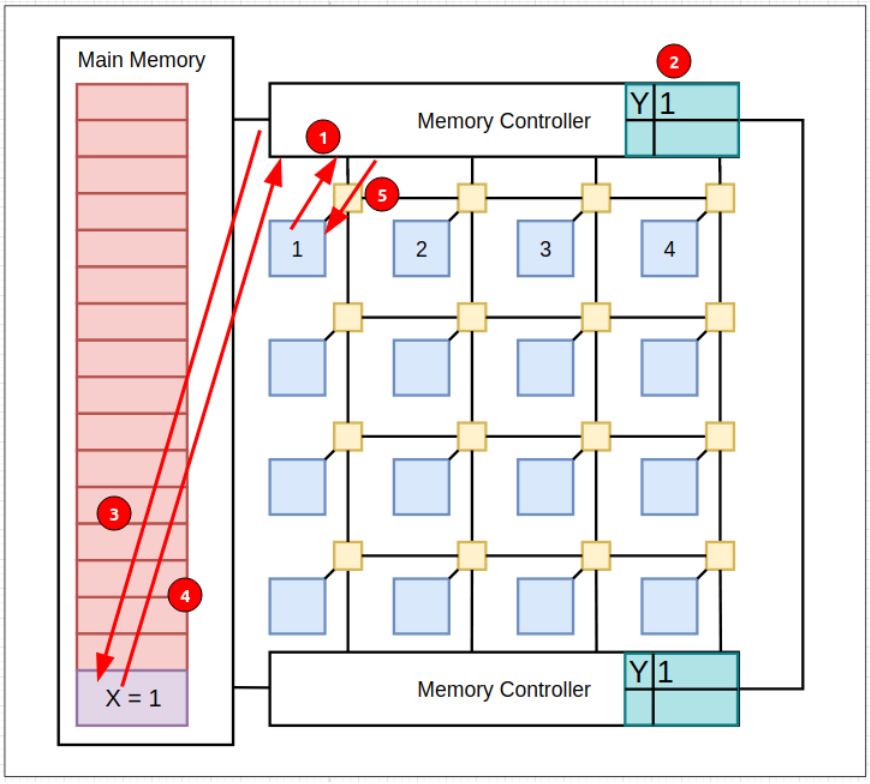
X memory location Y block contains value 0

At the beginning of this scenario, X memory location contains 1. X is a byte in Y block. Since the local cache deals with blocks the directory entry corresponding to Y is marked in the figure.



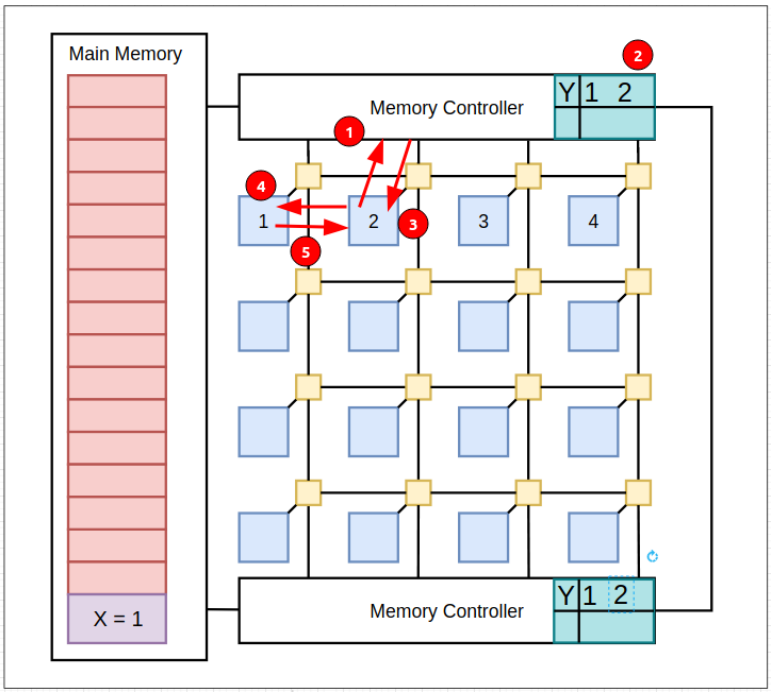
Node 1 requests X

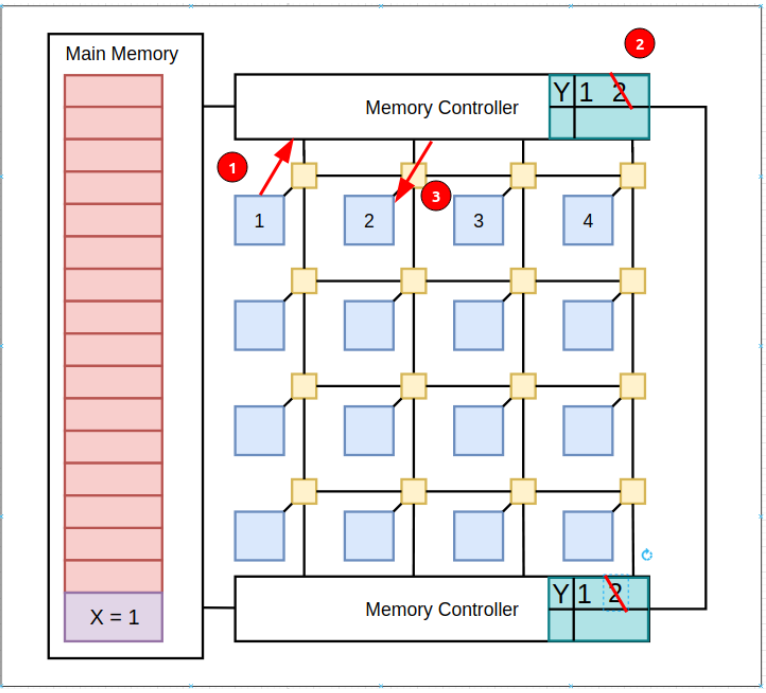
Now, the CPU in node 1 tries to load the value in X memory address from the local cache but the local cache does not contain the valid block(Y). Therefore the cache needs to fetch block Y from the memory and send a request to the controller. The controller will look at the directory and the directory does not contain any node address that has Y, the directory will add the node 1 to the entry and fetch the block Y from the memory and send the block Y to the node 1's cache.



Node 2 requests Y

Now, the CPU in node 2 tries to load the value in X and it is not available in the local cache so the cache needs to fetch the block Y and send a request to the memory. Memory controller will look at the directory if Y and find out the node 1's cache contains Y. Memory controller adds the node address 2 to the directory and informs node 2 that node 1 has the value in its cache. Node 2's cache will fetch it from node 1's cache.





Node 1 updates the value in X

Now, the node 1 updates the value X to 0 and before writing it sends a message to the memory controller. The memory controller will inform the node 2's cache to invalidate block Y and remove node 2 from the entry in the directory.

Node 2 updates the value in X

Now node 2's CPU needs the value in X and since the block is invalid the cache will fetch the block from memory. Memory controller will add node 2 to the entry and inform node 1 has the block. Node 2 will fetch the block from node 1.