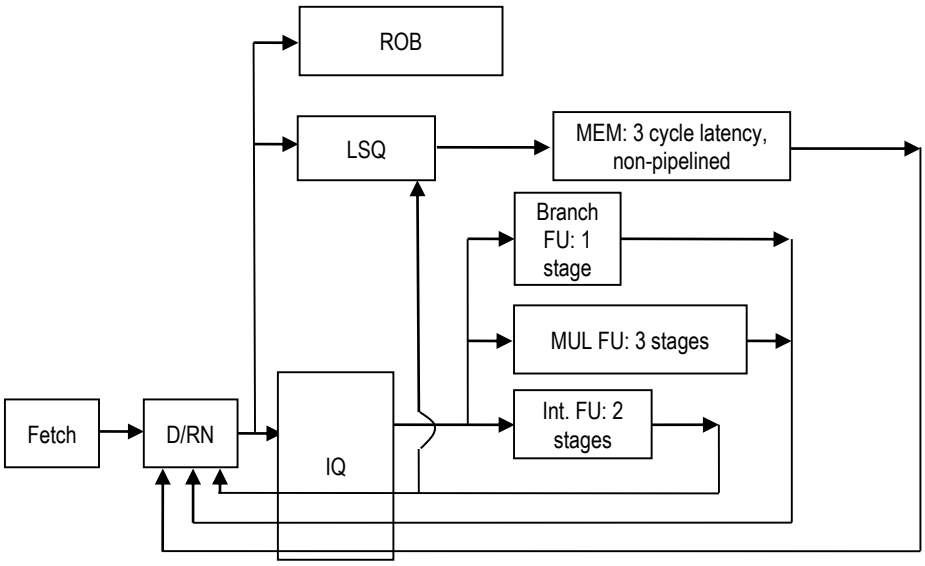
# CS 520 – Fall 2019 – Ghose

**Project 2 – Team Project – At most 3 Students per Team**
**Due: Thursday December 12, 2019, by 11:59 pm. No submissions will be entertained after Friday, Dec. 13, 11:59 pm. Friday submissions will carry a 20% penalty**

**All demos to be completed by Dec. 15, 2019. You can finish earlier and arrange for early demos subject to availability of the TAs. ALL team members must be present for the demonstration.**

This project requires you to implement the simulator for an out-of-order processor using register renaming, similar to that of **Variation 2** in the lecture notes. The ISA of this processor is identical to the processor for Project 1.

The datapath of the simulated processor, excluding the physical and architectural register files, is as follows:




Note again that the above diagram does **not** show other components like the architectural and physical register files, branch instruction stack, rename table etc. and their relevant connections, as well as forwarding paths to the inputs of the function units (for supporting back-to-back execution).

The specific details of the datapath are as follows:

- The instruction fetch stage and the decoding/renaming stage (D/RN) each have a delay of one cycle.
- The issue queue (IQ) has a maximum of 8 entries. Register operands are read at the time of issue. It takes one cycle for an IQ entry to wake up, be granted a FU and move to the required function unit.
- The LSQ has a maximum capacity of 6 entries.
- The physical register file has 24 registers. Each physical register has an extension that holds any flag values that were generated along with the contents of that register. When the simulation starts, no

physical registers are allocated. Physical registers are always allocated in increasing order of their addresses. Assume that a physical register that is freed up in a cycle can be reallocated in the **same cycle**.

- The ROB has a capacity of 12 entries.
- The function unit for the multiply operations is pipelined into **three** stages (each with a delay of a single cycle).
- The integer function unit is pipelined into **two** stages (each with a delay of single cycle). All integer operations (excluding multiplication), logical operations and *address calculations for loads and stores* are performed in the integer function unit.
- The branch function unit *has its own adder to calculate a target address* and has a single stage (one cycle delay). The issue of a branch instruction checkpoints the allocation list of physical registers and the rename table to speed up recovery on a branch misprediction.
- Memory operations, once initiated, take 3 cycles to complete and memory and is implemented by a memory function unit as shown. Addresses for the loads and stores are calculated in the integer function unit and written directly to the associated LSQ entry. The write of a calculated memory address to the LSQ takes one cycle, after which the LSQ can begin memory operation *as long other conditions for starting memory operations are valid*. Memory operations cannot be overlapped.
- **Instructions can be issued from the issue queue to multiple function units simultaneously** and the physical register file has sufficient number of ports to allow all register operands for simultaneously-issued instructions to be read out in parallel.
- The processor supports **back-to-back execution** by implementing wakeup tag broadcasts (from all FUs, including the memory FU) one cycle before the availability of the corresponding result, which can be forwarded to an instruction as it issues.
- The processor supports **speculative execution**. A speculation depth of 2 is supported, allowing for at most two unresolved branch instructions at any time. As soon as the branch instruction discovers a mispredicted branch, it takes one cycle to flush all instructions and resume execution along the correct path. A JUMP instruction requires all following instructions that entered the pipeline to be flushed.

Some other assumptions to be made for designing the simulator are as follows:

1. A sufficient number of forwarding buses exist to permit simultaneous forwarding from all function units that can forward a result.
2. If two or more instructions become eligible for issue to the same FU in the same cycle, the instruction with the lowest associated PC value (that is, address) is chosen for issue. (This tie breaking solution is complex to implement in real designs, but we need this to facilitate grading!)
3. A physical register can be freed up in the same cycle it is written to (and reallocated in the same cycle), as long as all conditions for freeing it up are valid.
4. An IQ entry is freed up at the end of the cycle in which the instruction it held was issued.
5. Loads cannot bypass earlier stores.
6. At the time of dispatching, IQ, ROB and LSQ entries can be set up simultaneously.
7. Instruction commitment takes one cycle.
8. Updates to the rename table take place at the beginning of the cycle in which a physical register is being updated, so the dispatching logic is aware of the validity of the physical register.

**A few things to note:**

You need to think of an appropriate way of determining when a physical register can be freed up. For the purpose of simulation, even complex things that are difficult to implement in reality can be implemented in your simulator!

The simulation of a HALT is easy with a ROB in place: when D/RF encounters a HALT, it stalls the D/RF stage and adds ONLY a ROB entry for the HALT at the end of the ROB. An IQ entry is not needed. When the ROB entry for the HALT reaches the head of the ROB, simply return control to the user prompt!

On stalls in the decode/rename stage due to the lack of any of the resources needed for dispatch, a similar approach is needed.

**ADDITIONAL REQUIREMENTS**

Since this is a team project, team members MUST document their contributions – what portions they have contributed to actively. **Team members must contribute equally on all aspects of the project**. This document is part of the submission requirement. **ALL team members must be present during the demo.**

The "simulate" command have to be modified if needed to support simulation for N cycles ("simulate N") and then enable the display of the simulated processor/memory status using "Display". A subsequent "simulate M", say, should be able to **continue** simulation for the **next** M cycles from the state where it left off at the end of the N-th cycle and then have the ability to display the state etc.

Other submission requirements will be announced later.