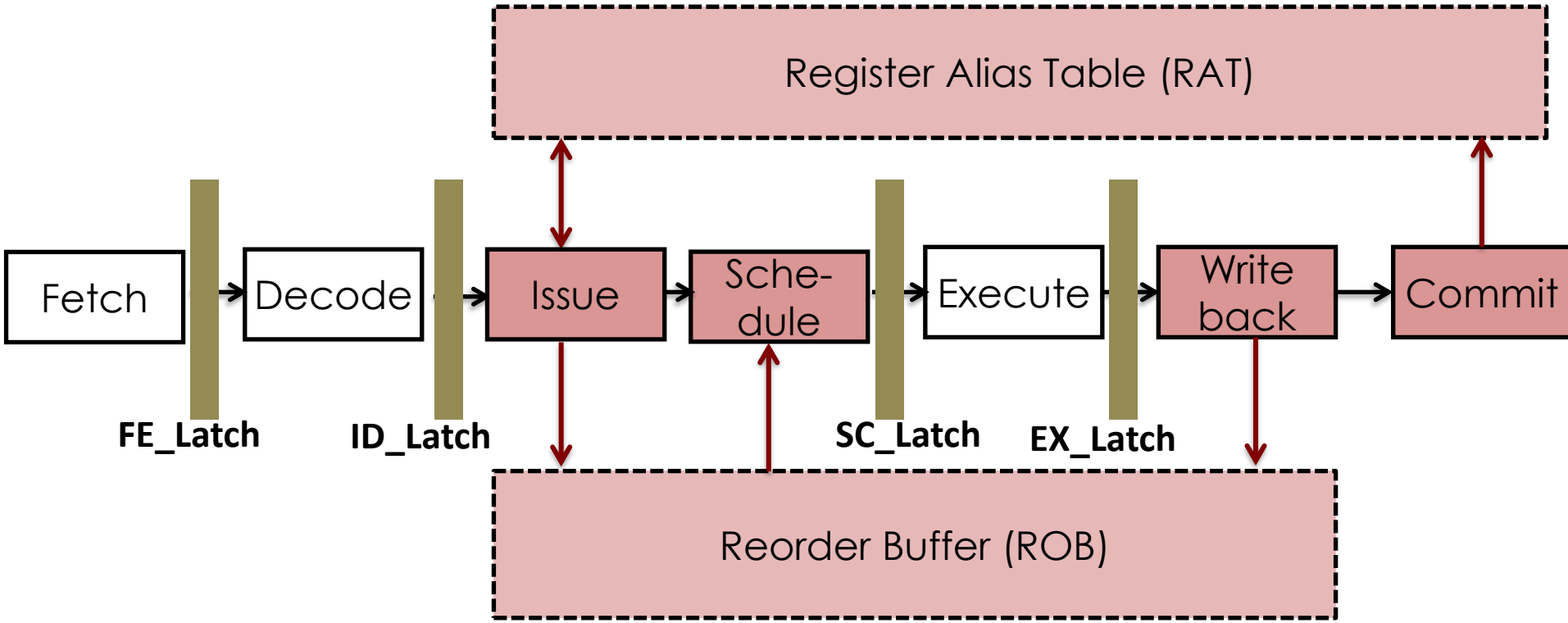
Out of Order Pipeline



All boxes in Red have to be implemented by students for the Lab.

Register Alias Table (RAT)

ARF_ID	valid	PRF_ID
1		
2		
3		

The number of entries in the RAT is equal to the number of ARF registers

Each entry has a valid bit and the id of the physical register

- *If the RAT entry is invalid, it means that the register is not renamed and the value is present in the Architecture Register File (ARF)*
- If the entry is valid, the PRF id is the index into the ROB ("data in ROB" design)

The RAT needs to provide the following functions:

During Issue:

- **Get the PRF corresponding to the ARF sources**
- **Set the PRF for the ARF destination**

During Commit:

- **Reset (i.e., valid = 0) for ARF destination**

Reorder Buffer (ROB)

PRF_ID	valid	exec	op	p1	src1 (PRF id)	p2	src2 (PRF id)	dest (ARF id)	ready
1									
2									
3									

Each entry has a valid bit, exec bit, tags and presence (i.e., **ready**) bits for two sources (p1, src1 and p2, src2), the destination ARF and a ready bit for the destination.

We use a simplified ROB from the one in class. The index of the ROB is the renamed PRF_ID (or tag) for the destination ARF id. Thus a separate "PRd" field is not needed.

Also, for simplicity, we do not worry about any rollbacks due to exceptions/mis-speculations for this lab. Thus the LPRd (last name of the physical reg Rd) need not be stored here like we did in class.

The ROB contains a head pointer (pointing to oldest uncommitted instruction) & a tail pointer (pointing to where a new instruction can be written to)

ROB (contd)

During Issue :
A new entry is allocated to the ROB, and the tail moves. The entry is marked as valid (use=1).

During Schedule :
When a valid entry has both sources ready (i.e., p1 and p2 are both true), it is scheduled and exec becomes 1.

During Writeback :
When an instruction completes execution, the ready bit for the destination in the ROB is marked 1.

During Commit :
If the oldest entry is valid and ready, it is removed from the ROB and committed.

ROB (contd)

The ROB needs to provide the following functions:

- **Check if there is space in the ROB**
- **Ability to insert an entry in the ROB**
- **Ability to execute (changing from exec=0 to exec=1)**
- **Ability to wakeup waiting instructions on a matching writeback (and mark p*=1)**
- **Ability to mark an ROB entry as ready**
- **Ability to check if the head entry is valid and ready**
- **Ability to read the valid and ready head entry**
- **Ability to remove the head entry if it is valid and ready**

Given Pipeline Stages: IF, ID, EX

Instruction FETCH (IF): Responsible for fetching the instruction and passing it to the decode stage. Note that we are assuming perfect branch prediction. So, we do not need to account for control flow stalls.

Instruction DECODE (ID): Responsible for decoding the instruction and passing it to the issue stage. Note that we are not doing any dependency check in this stage. Dependencies are taken care of later in the pipeline.

Execution Stage (EXE): The EXE stage takes the instruction from the SC_latch and transfers it to the EX_latch. However, we also need to support multi-cycle instructions. The way we handle this is by having a waiting queue (EXEQ) that buffers waiting instruction and tracks their wait times. Once the wait time of the instructions get reduced to zero these Instructions are transferred to the EX_latch.

NOTE: Our code base will already have the above three stages implemented. So, you will need to focus on implementing only the stages new to the out-of-order pipeline.

New Pipeline Stage: Issue

Insert new instruction into the ROB

- Check if the ROB is not full
- Insert the instruction
- Get remapped sources using RAT
  - If the RAT entry is invalid, mark the source as ready
  - If the RAT entry is valid, get the source ready bits from the ROB
- Get the index (i.e., PRF_ID) of ROB entry. This is the new map for the dest reg
- Update RAT with destination register map

New Pipeline Stage: Schedule

Find up to N instructions ready to execute (N: pipewidth)

You will implement two scheduling policies:

1. In order

- Check if the oldest instruction in the ROB is ready (and unscheduled).
  - If yes, schedule it
  - If not, stall

2. Out of Order

- Obtain a list of all instructions for which both sources are ready, and the instruction is unscheduled.
- Find the oldest instruction in this list and schedule it

New Pipeline Stage: Writeback

For all instructions out of the EX stage (note these can be more than pipewidth)

- Broadcast the results to all the entries in the ROB
  - This should wake up any sources waiting for this particular tag
- Mark the ROB entry as ready

New Pipeline Stage: Commit

- Check the head of the ROB to see if the head of the ROB is ready
- If yes, commit that instruction using pipe_commit_inst() (which will update the stats, halt the pipeline if need be)
- Remove the instruction from the ROB, which will increment the ROB head
- Update the RAT