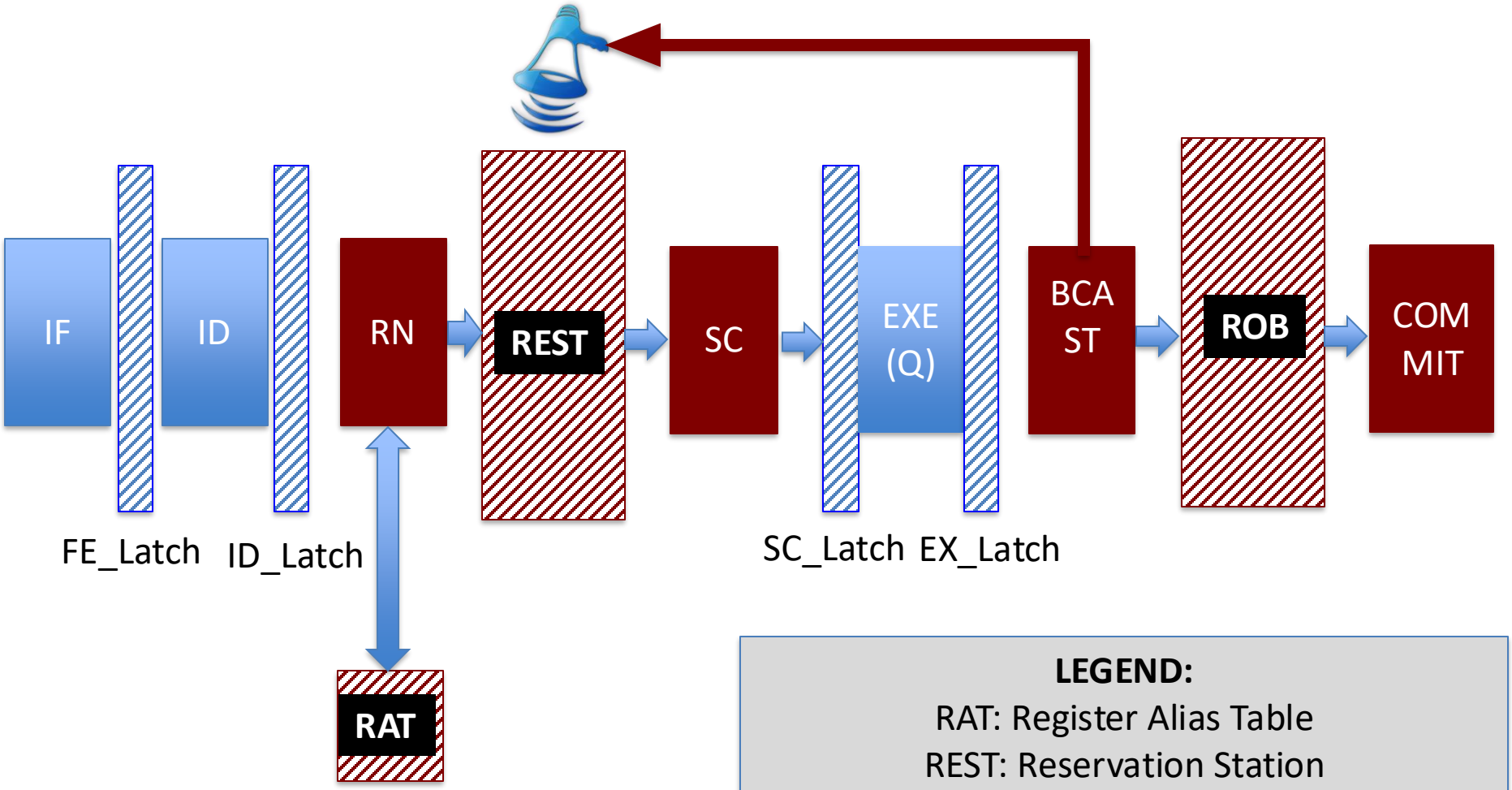
Lab 3: Out of Order Pipeline



LEGEND:

RAT: Register Alias Table

REST: Reservation Station

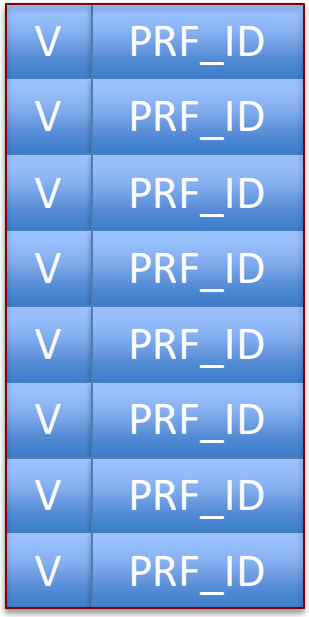
ROB: ReOrder Bufer

Solid boxes are pipeline stage

Dashed boxes are latches or structures

Boxes in RED to be coded by the students

New UNIT: Register Alias Table (RAT)



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

In our case, the physical register is embedded in the ROB, so The RAT maps the destination register to the ROB entry.

If the RAT entry is invalid, then the register is not renamed And the value can be read from the Architecture Register File (ARF) right away.

The number of entries in RAT is equal the number of ARF regs.

The RAT needs to provide three functions:

1. Getting the PRF corresponding to ARF
2. Setting the PRF for a given ARF entry
3. Resetting the mapping for a given ARF entry (on retirement)

New UNIT: Reservation Station (REST)

V	S	SRC1tag	R1	SRC2tag	R2

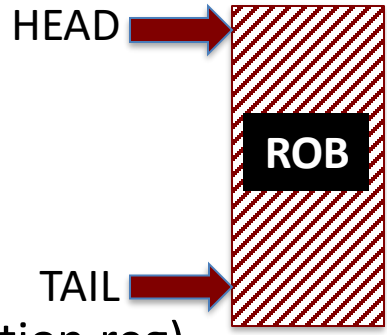
Each entry has a valid bit, scheduled bit, tags for two sources (SRC1tag And SRC2tag) and if the two inputs are ready.

When a valid entry has both R1 and R2 as true, it can be scheduled. Once an entry gets scheduled S=1. Note that a scheduled entry does not leave REST. It leaves the REST only when it finishes execution and is in BCAST stage

The REST needs to provide the following functions:

1. Check if there is space in the REST
2. Ability to insert an entry in the REST
3. Ability to wakeup waiting instructions on a matching broadcast (and mark R=1)
4. Ability to schedule (changing from S=0 to S=1)
5. Ability to remove instructions from the REST

New UNIT: ReOrder Buffer (ROB)



Each entry has a valid bit, ready bit, and instruction info (for destination reg)

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

When a entry is allocated to the ROB, the tail moves. The entry is marked as valid (V=1) but not ready (R=0). When an instruction completes execution and Reaches BCAST stage, the ROB entry is marked as ready (R=1). In commit stage, if the oldest entry is valid and ready, it is removed from the ROB and committed.

The ROB needs to provide the following functions:

1. Check if there is space in the ROB
2. Ability to insert an entry in the ROB
3. Ability to mark the ROB entry as ready (from BCAST stage)
4. Ability to read a valid and ready ROB entry (from RENAME stage)
5. Ability to check if the head entry is valid and ready
6. Remove the head entry if it is valid and ready

Known 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 rename stage. Note that we are not doing 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 SC_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 Stage: Rename (RN) and Reg Read

RN

Need to first check:

1. Access the ROB to obtain an entry where to place this inst
2. Access the REST to obtain an entry where to place this inst

If (1) and (2) completes:

1. Get remapped value of sources using RAT
2. Update the RAT with destination register remap
3. If the value can be obtained from the ARF/ROB , mark the sources as ready
4. Insert instruction in REST

New Stage: Schedule (SC)

You will implement two scheduling policies:

1. **In order**

Check if the oldest instruction in REST 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 Stage: Broadcast(BCAST)

The Broadcast stage is responsible for the following:

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

1. **Broadcast the results to all the entries in the REST**
This should wake up any sources waiting for this particular tag

2. **Mark the ROB entry as ready**

3. **Remove the entry from the reservation station (REST)**

New Stage: COMMIT

The commit stage is responsible for the following:

Check the head of the ROB to see if the head of the ROB is ready

If yes, commit that instruction, update the stats, halt the pipeline if need be

Also, remove the instruction from the ROB, which will increment the ROB head