## Reorder Buffer

<table>
<thead>
<tr>
<th>Method</th>
<th>Issue</th>
<th>Execute</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classic 5-stage pipeline</td>
<td>In-order</td>
<td>In-order</td>
<td>In-order</td>
</tr>
<tr>
<td>Multi-cycle FP pipeline</td>
<td>In-order</td>
<td>In-order</td>
<td>Out-of-order</td>
</tr>
<tr>
<td>Scoreboard</td>
<td>In-order</td>
<td>Out-of-order</td>
<td>Out-of-order</td>
</tr>
<tr>
<td>Tomasulo</td>
<td>In-order</td>
<td>Out-of-order</td>
<td>Out-of-order</td>
</tr>
</tbody>
</table>

### Maintaining precise interrupts:
- Complicated when instructions can complete (write) out of order.
  - Earlier instruction may *raise interrupt* long *after later instructions* have *completed write*
  - Later instructions may have *overwritten registers*

### Reorder Buffer:
- Mechanism for emulating *In-order writes* without sacrificing concurrency
  - **Buffer** the results of completing *instructions reorder* them and *write* them *in order*
  - Idea of reorder buffer can be used to implement aggressive *branch speculation*
Speculative Execution

Block 1

Decision Delayed

A

B

Block 2
Speculative Execution

Speculative task:
Task that is performed without assurance that it is required

Speculatively Executed Block
Overlap Execution with Stall Delay

Decision Delayed
Guess future execution path
Speculative Execution

Block 1

Decision Delayed
Guess future execution path

A

Speculatively Executed Block

Also Speculatively Executed

B

Block 2
Speculative Execution

Speculatively Executed Block

Speculatively Executed Block

Decision Delayed
Guess future execution path

Decision Resolved
Rollback to decision point
Speculative Execution

Block 1

Decision Delayed
Guess future execution path

Speculatively Executed Block

A

Also Speculatively Executed

Block 2

B

Speculative task:
Incorporate rollback mechanism to undo consequences of the task

Rollback to decision point
Checkpoint Based Speculation

- Create **checkpoint** immediately prior to speculation
- At checkpoint: Take a **snapshot** of relevant **system state**
- On rollback: **Restore** saved **system state**

**Checkpoint:**

- Make copies of register contents
- Copy-On-Write mechanism to make copies of memory locations written by speculative instructions

- Primarily used in software implementations of fault tolerance or for maintaining transactional semantics
Reorder Buffer for Speculation

- Do not allow speculative instructions to update system state
  - Writes by speculative instruction are stalled till speculative status is resolved

- Use renaming mechanisms to transfer information between speculative instructions
  - Rename source registers (a la Tomasulo)

- On resolution
  - Mis-speculation: Squash the speculative instructions
  - Correct execution: Commit (the writes) of the speculative instructions

- Reorder Buffer:
  - Mechanism to force In-Order Writes of Instructions
    - Buffer the results of completing instructions reorder them and write them in order
Extending Tomasulo Pipeline with Reorder Buffer

 ISSUE → DISPATCH → EX → WRITE → COMMIT

RoB:
- Storage to buffer writes until ready to commit
- Circular queue written and released in FIFO (instruction) order
- Each entry allocates space for 1 instruction to store its results + identifying information
Tomasulo’s Pipeline with RoB based Commit

Common Data Bus (CDB)

Accesses RoB and REG FILE
Extending Tomasulo Pipeline with Reorder Buffer

Destination registers need to distinguish between 3 possible states:

1. **Available (A):** No pending write to register. Register has its final value.

2. **In Flight (I):** Writer instruction is in flight:
   
   The last instruction with that destination register has not yet completed its write.

3. **Ready (R):** Writer has completed write but not yet committed. The value from the reorder buffer will be written to the register when it commits.

**Note:** A and I are the same two register states of regular Tomasulo. The state of a register is used by an issuing instruction to find out where to get its source operand.
Key Features (Tomasulo with Reorder)

Issue instruction \( X \) (ALU instruction):

1. Stall issue until
   - Free Reservation Station and Reorder Buffer slots are available
   - Let \( RS_X \) and \( RB_X \) denote the RS and RoB slots allocated to \( X \).
   - Henceforth \( X \) is identified using its Reorder Buffer tag \( RB_X \)
   - Reservation Station \( RS_1 \) fields exactly the same as regular Tomasulo
   - \( RB_X \) made up of the following fields:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Destination</th>
<th>State</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Tag:** The identifier for instruction \( X \) (usually implicit index into the buffer)
- **Destination:** The destination register of instruction \( X \)
- **State:** Yes/No---- RoB entry is valid result
  - Yes: \( X \) has completed write
  - No: \( X \) is In flight
- **Value:** Result of \( X \) (broadcast during write by \( X \))
Example

A: MUL F4, F0, F2
B: ADD F8, F4, F6

Issue A

<table>
<thead>
<tr>
<th>RS_A</th>
<th>v0</th>
<th>v2</th>
<th>MUL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>F4</td>
</tr>
</tbody>
</table>

Tomasulo’s without RoB

<table>
<thead>
<tr>
<th>RB_A</th>
<th>v0</th>
<th>v2</th>
<th>MUL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>F4</td>
</tr>
</tbody>
</table>

Tomasulo’s with RoB

<table>
<thead>
<tr>
<th>RB_A</th>
<th>F4</th>
<th>No</th>
<th>-----</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

STATE: 1
Example

A: MUL F4, F0, F2
B: ADD F8, F4, F6

Issue B

Tomasulo’s without RoB

Tomasulo’s with RoB
Key Features: Instruction Issue (contd ….)

3. For each source operand register \( S \):
   - Action depends on state of source register \( (A, I, R) \)
     - \( A \): copy value from \( S \) immediately to \( RS_X \)
     - \( I \) (pending write by instruction \( J \)): tag the source field of \( RS_X \) with \( RB_J \)
     - \( R \) (pending update from \( RB_J \)): read value from \( RB_J \) and copy to \( RS_I \)

\[
\text{ADDD F2, F0, F4}
\]
4. For destination register D
   • Make \( X \) the writer of \( D \)
   • Set the state of \( D \) to I (Implicitly cancels the previous write if any).