Course on: "Advanced Computer Architectures"

REORDER BUFFER

Prof. Cristina Silvano
Politecnico di Milano
cristina.silvano@polimi.it

Hardware-based Speculation

- We introduce the concept of HW-based speculation, which extends the ideas of dynamic scheduling beyond branches.

- HW-based speculation combines **3 key concepts:**
  1. **Dynamic branch prediction;**
  2. **Speculation** to enable the execution of instructions before the branches are solved by undoing the effects of mispredictions;
  3. **Dynamic scheduling** beyond branches.

Hardware-based Speculation

- To support **HW-based speculation,** we need to enable the execution of instructions **before** the control dependences are solved.
- In case of **misprediction,** we need to undo the effects of an incorrectly speculated sequence of instructions.
- Therefore, we need to separate the process of execution completion from the commit of the instruction result in the RF or in memory.
- **Solution:** We need to introduce an HW buffer, called **ReOrder Buffer,** to hold the result of an instruction that has finished execution, but not yet committed in RF/memory.

ReOrder Buffer (ROB)

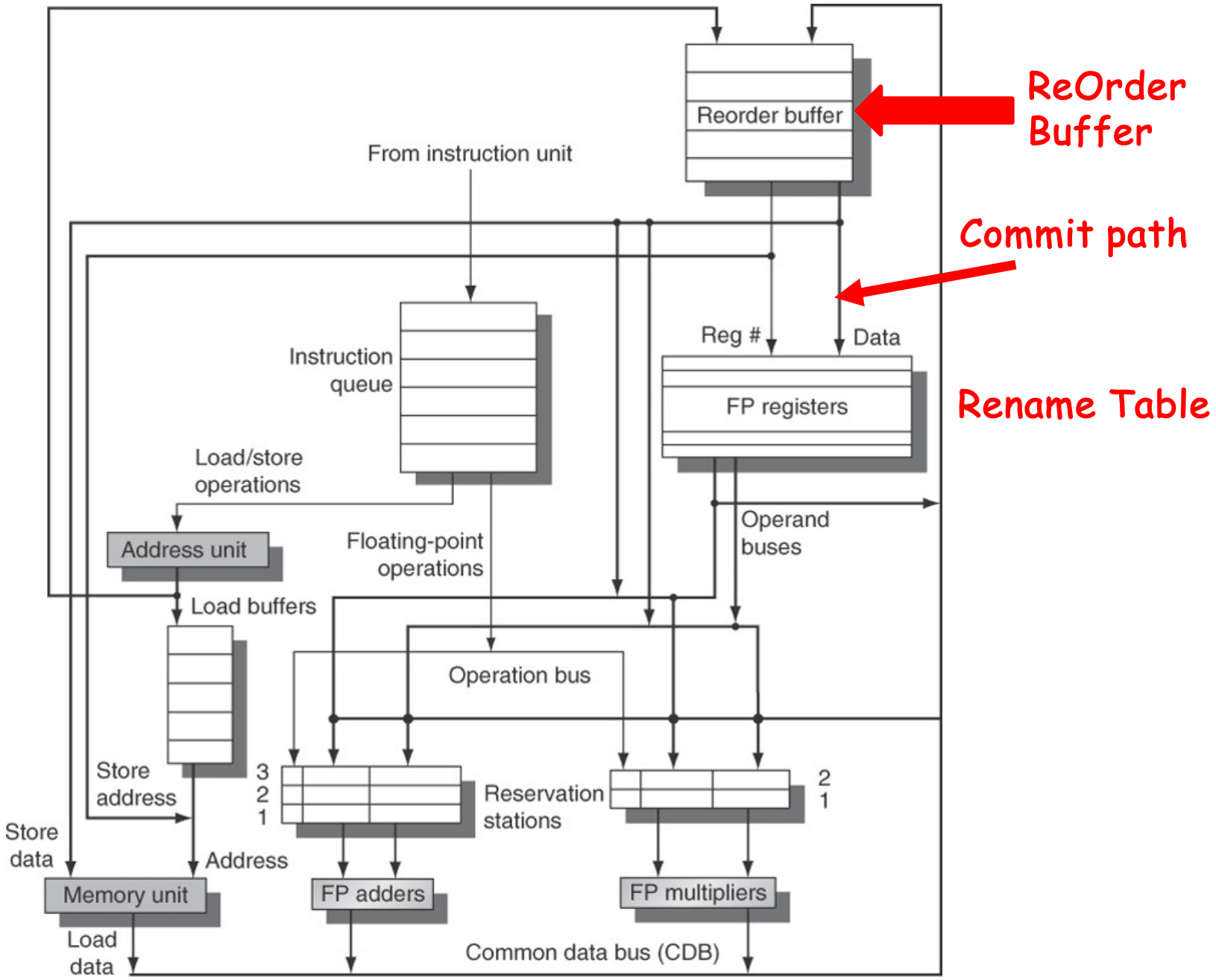
ReOrder Buffer (ROB)

- ReOrder Buffer has been introduced to support ***out-of-order execution but in-order commit.***
- Buffer to hold the results of instructions that have finished execution ***but not yet committed.***
- Buffer to pass results among instructions that have started ***speculatively*** after a branch.
- Originally (1988) introduced to maintain the ***precise interrupt model*** → Processors with ROB can execute dynamically while maintaining a precise interrupt model because instruction commit happens in order.

ReOrder Buffer (ROB) and RSs

- The **ROB** is also used to pass results among dependent instructions to start execution as soon as possible
  → **The renaming function of Reservation Stations is replaced by ROB entries.**
- Use **ReOrder Buffer numbers** instead of reservation station numbers as pointers to pass data between dependent instructions.
- Reservation Stations now are used only to buffer instructions and operands to FUs (to reduce structural hazards).

Speculative Tomasulo Architecture with ROB

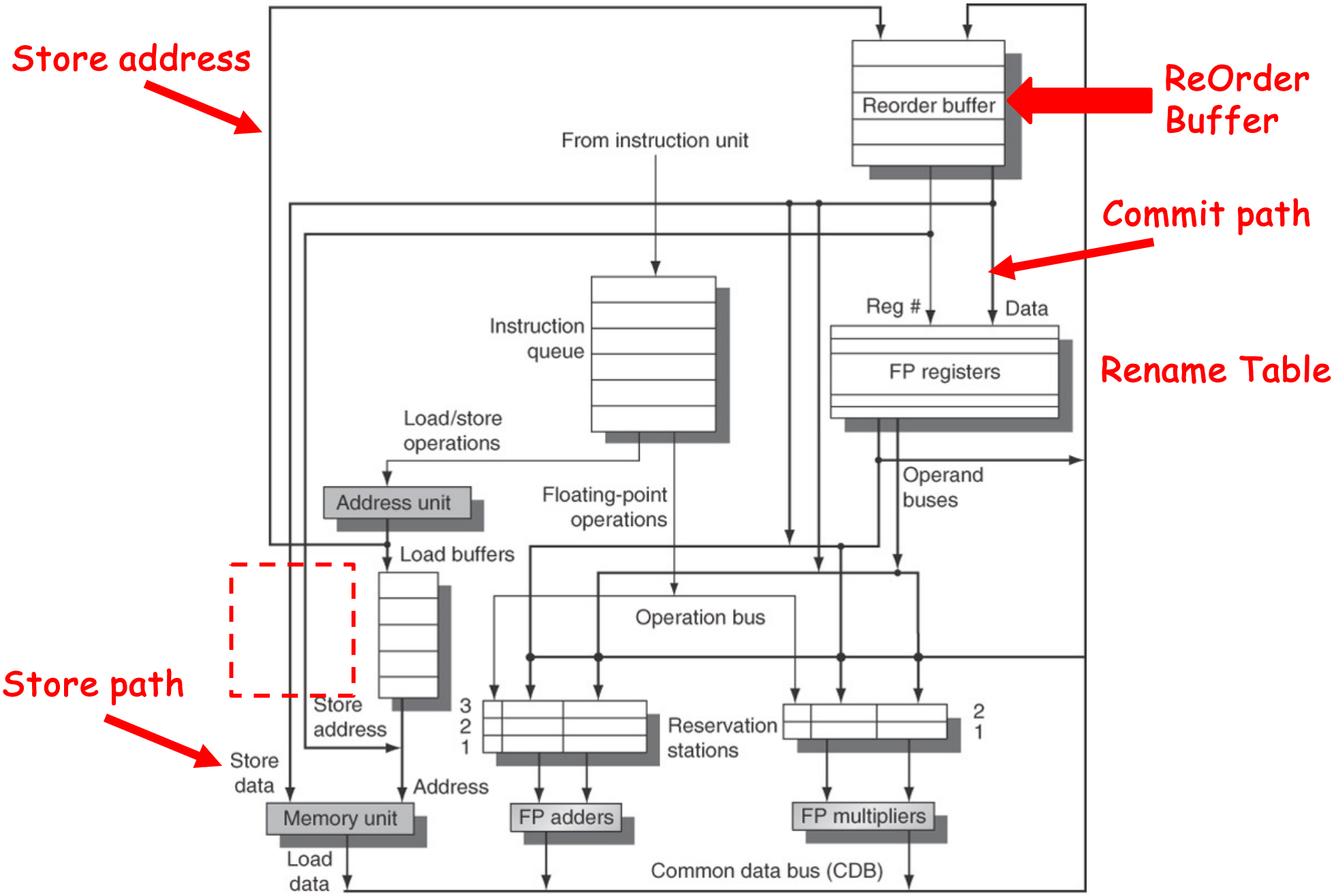


ReOrder Buffer

Commit path

Rename Table

ROB also replaces the Store Buffers



ReOrder Buffer (ROB)

- **The ReOrder Buffer entries can be operand sources**
  ⇒ Use **ReOrder Buffer numbers** instead of reservation station numbers as pointers to pass data between dependent instructions.
- ROB supplies operands to other RSs between execution complete & commit (as a sort of forwarding).
- Once the instruction commits, the result will be written from the RoB to the register file.

ReOrder Buffer (ROB)

- Instructions are written in ROB in strict program order – when an instruction is issued, an entry is allocated to the ROB **in sequence *(in-order)*.**
- Each entry must keep the **status of an instruction**: issued, execution started, execution completed, ready to commit.
- Each entry includes a ***speculative status flag***, indicating whether the instruction is being speculatively executed or not.
- An instruction is **ready to *commit (retire)*** if and only if:
  1. Its execution has completed, and
  2. It is not marked as speculative, and
  3. All previous instructions have already retired.

RoB Content

Each entry in ROB contains the following fields:

- ***Busy field:*** indicates whether ROB entry is busy or not;
- ***Instruction type field*** – indicates whether instruction is a branch (no destination result), a store (memory address destination), or a load/ALU (register destination);
- ***Destination field***: supplies register number (for loads and ALU instructions) or memory address (for stores) where results should be written;
- ***Value field***: used to hold the value of the result until instruction commits;
- ***Ready field:*** indicates that instruction has completed execution, value is ready
- ***Speculative field:*** indicates whether instruction is executed speculatively or not;

ReOrder Buffer (ROB)

The ROB is a **circular buffer** with two pointers:

- **Head pointer** indicating the instruction that will **commit** (leaving ROB) **first** and
- **Tail pointer** (indicating next free entry)

ROB#	BUSY	INSTR. TYPE	READY	DEST.	VALUE	SPEC.
ROB0	Yes	I_n	No	F0	-	No
ROB1	No	-	-	-	-	-
ROB2	No	-	-	-	-	-
ROB3	No	-	-	-	-	-

ReOrder Buffer (ROB) & Rename Table

Register #	Pointer	Value
F0	ROB0	
F2	ROB1	
F4		
F6		
F8		

ROB#	BUSY	INSTR. TYPE	READY	DEST.	VALUE	SPEC.
ROB0	Yes	I_n	No	F0	-	No
ROB1	Yes	I_{n+1}	No	F2	-	No
ROB2	No	-	-	-	-	-
ROB3	No	-	-	-	-	-

ReOrder Buffer (ROB) & Rename Table

Register #	Pointer	Value
F0	ROB0	
F2	ROB1	
F4	ROB2	
F6		
F8		

ROB#	BUSY	INSTR. TYPE	READY	DEST.	VALUE	SPEC.	
ROB0	Yes	I_n	No	F0	-	No	Head
ROB1	Yes	I_{n+1}	No	F2	-	No	
ROB2	Yes	I_{n+2}	No	F4	-	No	
ROB3	No	-	-	-	-	-	Tail

ReOrder Buffer (ROB) & Rename Table

Register #	Pointer	Value
F0	ROB0	
F2	ROB1	
F4	ROB2	
F6	ROB3	
F8		

ROB#	BUSY	INSTR. TYPE	READY	DEST.	VALUE	SPEC.
ROB0	Yes	I_n	No	F0	-	No
ROB1	Yes	I_{n+1}	No	F2	-	No
ROB2	Yes	I_{n+2}	No	F4	-	No
ROB3	Yes	I_{n+3}	No	F6	-	Yes

Head

ReOrder Buffer (ROB) & Rename Table

Register #	Pointer	Value
F0	ROB0	
F2	ROB1	
F4	ROB2	
F6	ROB3	
F8		

ROB#	BUSY	INSTR. TYPE	READY	DEST.	VALUE	SPEC.
ROB0	Yes	I_n	Yes	F0	10	No
ROB1	Yes	I_{n+1}	No	F2	-	No
ROB2	Yes	I_{n+2}	No	F4	-	No
ROB3	Yes	I_{n+3}	No	F6	-	Yes

ReOrder Buffer (ROB) & Rename Table

Register #	Pointer	Value
F0		10
F2	ROB1	
F4	ROB2	
F6	ROB3	
F8		

ROB#	BUSY	INSTR. TYPE	READY	DEST.	VALUE	SPEC.
ROB0	No	-	-	-	-	-
ROB1	Yes	I_{n+1}	No	F2	-	No
ROB2	Yes	I_{n+2}	No	F4	-	No
ROB3	Yes	I_{n+3}	No	F6	-	Yes

Tail

Head

ReOrder Buffer (ROB) & Rename Table

Register #	Pointer	Value
F0		10
F2	ROB1	
F4	ROB2	
F6	ROB3	
F8	ROB0	

ROB#	BUSY	INSTR. TYPE	READY	DEST.	VALUE	SPEC.
ROB0	Yes	I_{n+4}	No	F8	-	Yes
ROB1	Yes	I_{n+1}	Yes	F2	20	No
ROB2	Yes	I_{n+2}	No	F4	-	No
ROB3	Yes	I_{n+3}	No	F6	-	Yes

Head

ReOrder Buffer (ROB) & Rename Table

Register #	Pointer	Value
F0		10
F2		20
F4	ROB2	
F6	ROB3	
F8	ROB0	

ROB#	BUSY	INSTR. TYPE	READY	DEST.	VALUE	SPEC.
ROB0	Yes	I_{n+4}	No	F8	-	Yes
ROB1	No	-	-	-	-	-
ROB2	Yes	I_{n+2}	No	F4	-	No
ROB3	Yes	I_{n+3}	No	F6	-	Yes

Tail

Head

Speculative Tomasulo Architecture with ReOrder Buffer

Speculative Tomasulo Architecture with ROB

- **Pointers are directed towards the ROB entries** instead of reservation stations;
- **Implicit register renaming** by using the ROB entries to eliminate WAR and WAW hazards and enable dynamic loop unrolling;
- A register or memory location is updated only when the instruction reaches the **head of ROB**.
- By using the RoB to hold the result, it is easy to:

1. Undo speculated instructions on mispredicted branches;
2. Manage exceptions precisely.

Four Phases of Speculative Tomasulo Algorithm

1. **Issue** – get instruction from instr. queue
2. **Execution started**
3. **Execution completed & Write result in ROB**
4. **Commit – Update RF or memory with ROB result**

Four Phases of Speculative Tomasulo Algorithm

1. **Issue** – get instruction from instr. queue:

   - **If there are one RS entry free and one ROB entry free ⇨ instr. Issued;**
     If its operands available in RF or in ROB, they are sent to RS;
     Number of ROB entry allocated for result is sent to RSs (to tag the result when it will be placed on the CDB).

   - **If RS full or/and ROB full: instruction stalls.**

Four Phases of Speculative Tomasulo Algorithm

2. **Execution started** – start and execute on operands (EX)
   - When both operands are ready in the RS (RAW solved) => start to execute;
   - If one or more operands are not yet ready => check for RAW solved by monitoring CDB for result;
   - For a **store:** only base register needs to be available (at this point, only effective address is computed).

Four Phases of Speculative Tomasulo Algorithm

3. **Execution completed & Write result in ROB**
   - Write result on Common Data Bus to all awaiting FUs **& to ROB value field;**
   - Mark RS as available.
   - For a store: the value to be stored is written in the ROB value field; otherwise monitor CDB until value is broadcast.

Four Phases of Speculative Tomasulo Algorithm

4. **Commit – update RF or memory with ROB result**
   - **When instr. at the head of ROB & result ready, update RF with result (or store to memory) and remove instruction from ROB entry.**
   - **Mispredicted branch flushes ROB entries (sometimes called "graduation")**

About Commit Phase

Commit: there are 3 different possible sequences:

1. **Normal commit:** instruction reaches the head of the ROB, result is present in the buffer. Result is stored in the Register File, instruction is removed from ROB;
2. **Store commit:** as above, but result is stored in memory rather than in the RF;
3. **Instruction is a mispredicted branch =>** speculation was wrong => ROB is flushed ("graduation") and execution restarts at correct successor of the branch.
   If the branch was correctly predicted => branch is completed.

RoB Example:

```
Loop:   LD      F0, 0(R1)
        MULTD   F4, F0, F2   # RAW F0
        SD      F4, 0(R1)    # RAW F4
        SUBI    R1, R1, #8
        BNEZ    R1, Loop     # RAW R1
                             # pred. taken
```

WAW F0 and WAW F4 with next iteration

RoB Example: First iteration issued

ROB#	BUSY	INSTR. TYPE	READY	DEST.	VALUE	SPEC.
ROB0	Yes	`LD F0, 0(R1)`	No (Exec. Started)	F0	--	No
ROB1	Yes	`MULTD F4,F0,F2`	No (Issued)	F4	--	No
ROB2	Yes	`SD F4, 0(R1)`	No (Issued)	M[0+[R1]]	--	No
ROB3	Yes	`SUBI R1,R1,#8`	No (Issued)	R1	--	No
ROB4	Yes	`BNEZ R1, Loop`	No (Issued)	-	--	No
ROB5	No					
ROB6	No					
ROB7	No					

Head

Tail

Example 2: Rename Table

Register #	Pointer
F0	ROB0
F2	
F4	ROB1
F6	
F8	
F10	

RoB Example: after some cycles: ROB *full!*

ROB#	BUSY	INSTR. TYPE	READY	DEST.	VALUE	SPEC.
ROB0	Yes	`LD F0, 0(R1)`	**Yes:Ready to Commit**	F0	M[0+[R1]]	No
ROB1	Yes	`MULTD F4,F0,F2`	No (Issued)	F4	--	No
ROB2	Yes	`SD F4, 0(R1)`	No (Issued)	M[0+[R1]]	--	No
ROB3	Yes	`SUBI R1,R1,#8`	Execution Started	R1	--	No
ROB4	Yes	`BNEZ R1, Loop`	No (Issued)	--	--	No
ROB5	Yes	`LD F0, 0(R1)`	No (Issued)	F0	--	**Yes**
ROB6	Yes	`MULTD F4,F0,F2`	No (Issued)	F4	--	**Yes**
ROB7	Yes	`SD F4, 0(R1)`	No (Issued)	M[0+[R1]]	--	**Yes**

Example 1: Rename Table

Register #	Pointer
F0	~~ROB0~~ ROB5
F2	
F4	~~ROB1~~ ROB6
F6	
F8	
F10	

RoB Example: next cycle

ROB#	BUSY	INSTR. TYPE	READY	DEST.	VALUE	SPEC.
ROB0	No					
ROB1	Yes	`MULTD F4,F0,F2`	Execution Started	F4	(*)	No
ROB2	Yes	`SD F4, 0(R1)`	No (Issued)	M[0+[R1]]	--	No
ROB3	Yes	`SUBI R1,R1,#8`	Execution Started	R1	--	No
ROB4	Yes	`BNEZ R1, Loop`	No (Issued)	--	--	No
ROB5	Yes	`LD F0, 0(R1)`	No (Issued)	F0	--	Yes
ROB6	Yes	`MULTD F4,F0,F2`	No (Issued)	F4	--	Yes
ROB7	Yes	`SD F4, 0(R1)`	No (Issued)	M[0+[R1]]	--	Yes

Tail

Head

(*) The value M[0+[R1]][F2] is under computation*

RoB Example: next cycle with ROB full!

ROB#	BUSY	INSTR. TYPE	READY	DEST.	VALUE	SPEC.
ROB0	Yes	`SUBI R1,R1,#8`	No (Issued)	R1	--	Yes
ROB1	Yes	`MULTD F4,F0,F2`	Execution Started	F4	(*)	No
ROB2	Yes	`SD F4, 0(R1)`	No (Issued)	M[0+[R1]]	--	No
ROB3	Yes	`SUBI R1,R1,#8`	Execution Started	R1	--	No
ROB4	Yes	`BNEZ R1, Loop`	No (Issued)	--	--	No
ROB5	Yes	`LD F0, 0(R1)`	No (Issued)	F0	--	Yes
ROB6	Yes	`MULTD F4,F0,F2`	No (Issued)	F4	--	Yes
ROB7	Yes	`SD F4, 0(R1)`	No (Issued)	M[0+[R1]]	--	Yes

Head

(*) The value M[0+[R1]][F2] is under computation*

ReOrder Buffer: Summary

- Only retiring instructions can ***complete***, i.e., **update architectural registers and memory;**
- ROB can support ***both speculative execution and precise exception handling***
- **Speculative execution**: each ROB entry includes a ***speculative status flag***, indicating whether the instruction has been executed speculatively;
- Finished instructions ***cannot retire as long as they are in speculative status****!*
- **Interrupt handling**: Exceptions generated in connection with instruction execution are made ***precise*** by accepting exception request only when instruction becomes ***"ready to retire"*** (exceptions are processed ***in order)***