Reorder Buffer

ROB

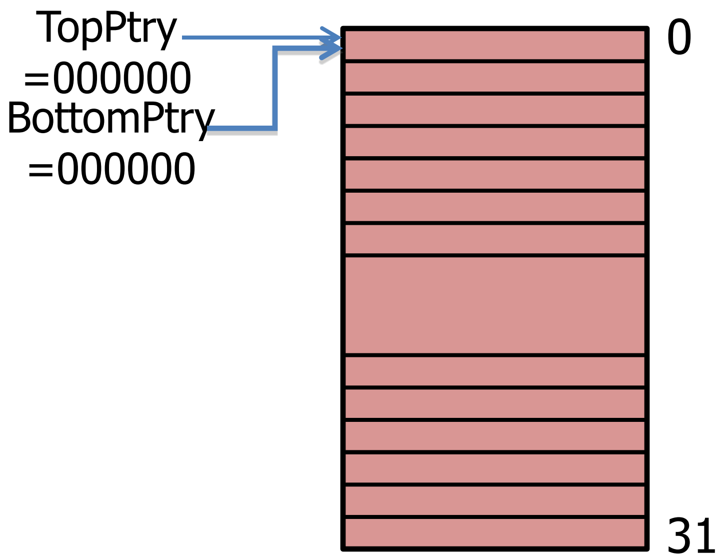
- The Re-order Buffer has 32 entries, with each entry 41-bit wide, and it is a FIFO structure.
- The Head as well as Bottom pointers are 6-bit wide. Lower 5 bits are used for indexing and the most significant bit is used to determine full and empty. Outside ROB, the 5 bit tag is enough for identification purpose
- Initially, both of two pointers are set to 000000.

  As in FIFO, we can use HP **=** BP to infer Empty and HP **xor** BP = 100000 to infer FULL.

🟩 Entries to be committed

🟥 Entries to be filled up

Instruction dispatch and commit

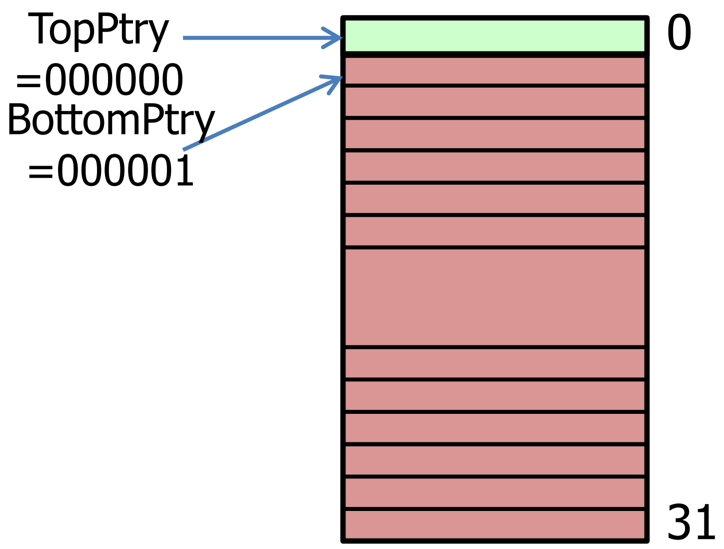


Left diagram shows the initial state of ROB: both pointers located at the TOP (location 0), and ROB is empty.

Empty condition can also happen whenever the values of the two pointers are same during the normal operation.

🟩 Entries to be committed
🟥 Entries to be filled up

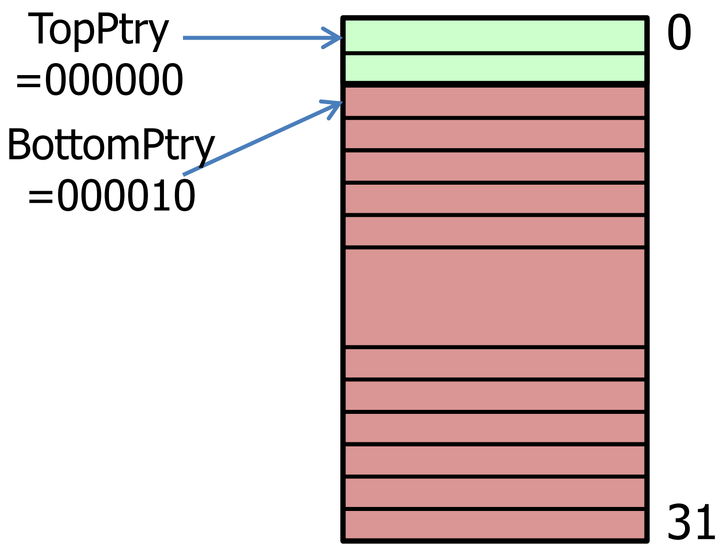
Instruction dispatch and commit



Gradually, entries in ROB are filled up one by one via bottom pointer as instructions are dispatched. In the meanwhile, instructions commit from the top pointer.

🟩 Entries to be committed
🟥 Entries to be filled up

Instruction dispatch and commit

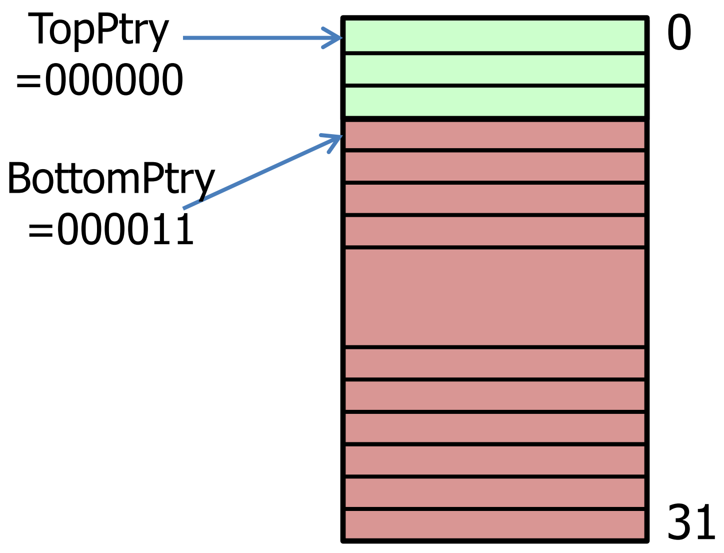


Gradually, entries in ROB are filled up one by one via bottom pointer as instructions are dispatched. In the meanwhile, instructions commit from the top pointer.

🟩 Entries to be committed

🟥 Entries to be filled up

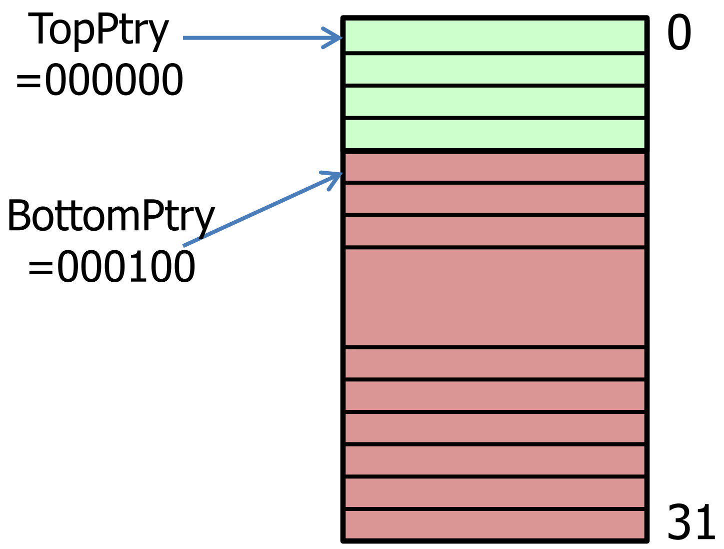
Instruction dispatch and commit



Gradually, entries in ROB are filled up one by one via bottom pointer as instructions are dispatched. In the meanwhile, instructions commit from the top pointer.

🟩 Entries to be committed
🟥 Entries to be filled up

Instruction dispatch and commit

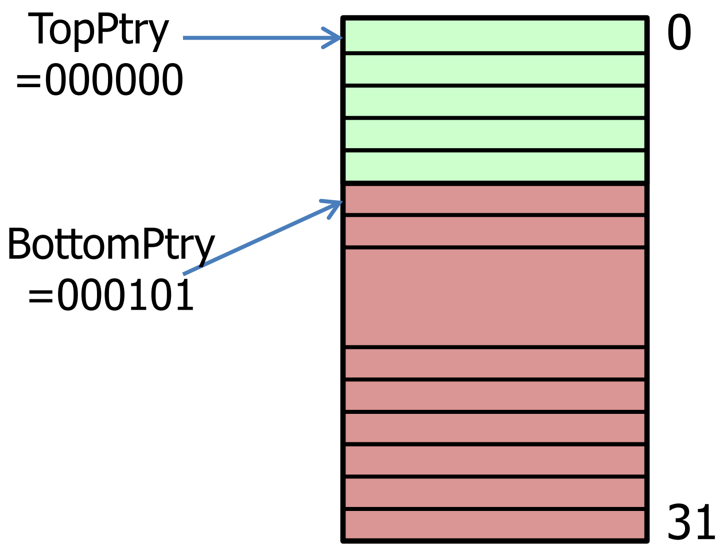


Gradually, entries in ROB are filled up one by one via bottom pointer as instructions are dispatched. In the meanwhile, instructions commit from the top pointer.

🟩 Entries to be committed

🟥 Entries to be filled up

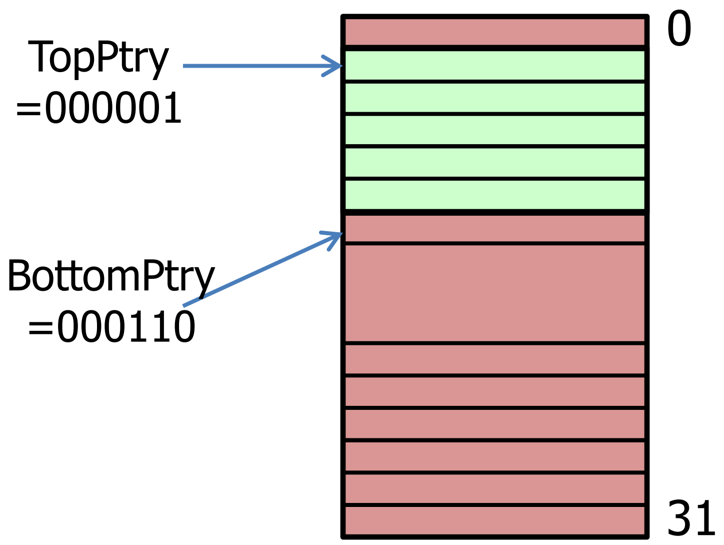
Instruction dispatch and commit



Gradually, entries in ROB are filled up one by one via bottom pointer as instructions are dispatched. In the meanwhile, instructions commit from the top pointer.

🟩 Entries to be committed
🟥 Entries to be filled up

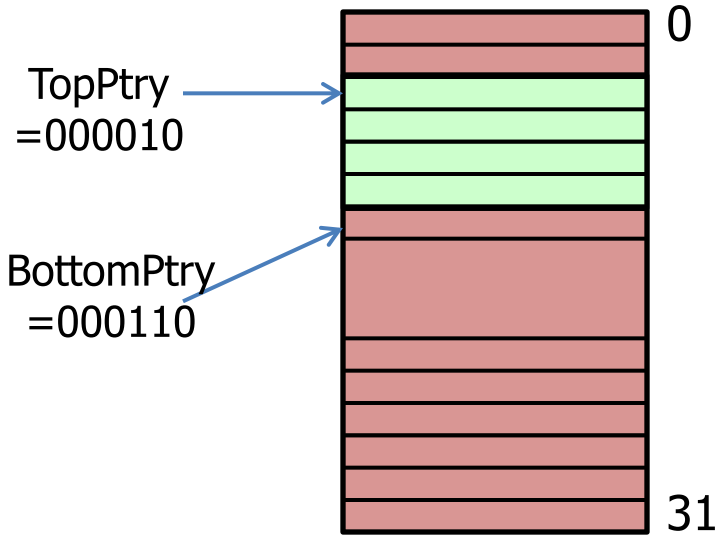
Instruction dispatch and commit



Gradually, entries in ROB are filled up one by one via bottom pointer as instructions are dispatched. In the meanwhile, instructions commit from the top pointer.

🟩 Entries to be committed
🟥 Entries to be filled up

Instruction dispatch and commit

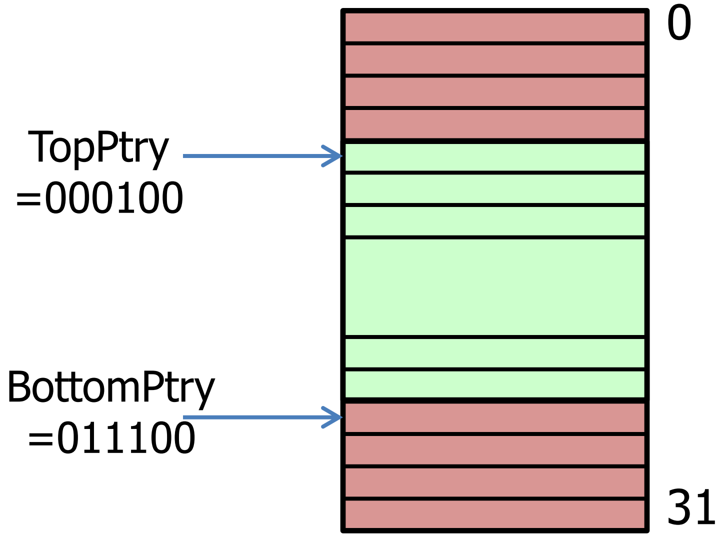


Gradually, entries in ROB are filled up one by one via bottom pointer as instructions are dispatched. In the meanwhile, instructions commit from the top pointer.

Think about conditions when Bottom Pointer is not incremented

🟩 Entries to be committed
🟥 Entries to be filled up

Instruction dispatch and commit

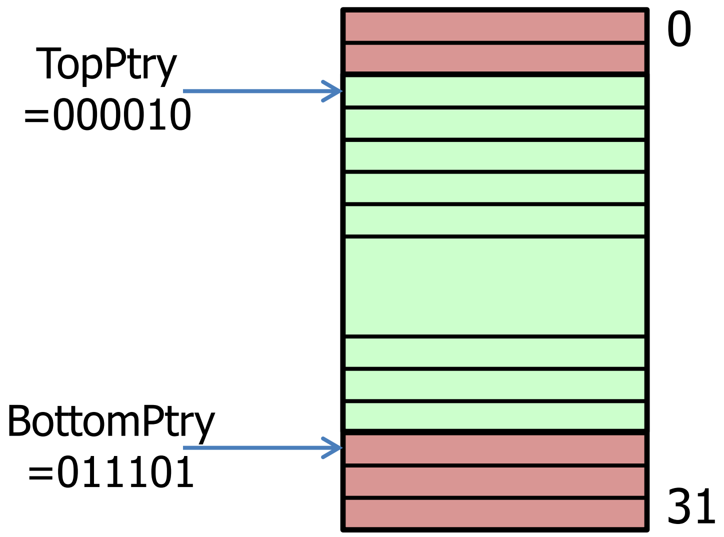


As shown in the left, this is a random intermediate state during ROB's normal operation, in which there are 24 instructions to be committed. Some of them may have been completed, and the rest of them may not.

Detailed description in the following slides.

🟩 Entries to be committed
🟥 Entries to be filled up

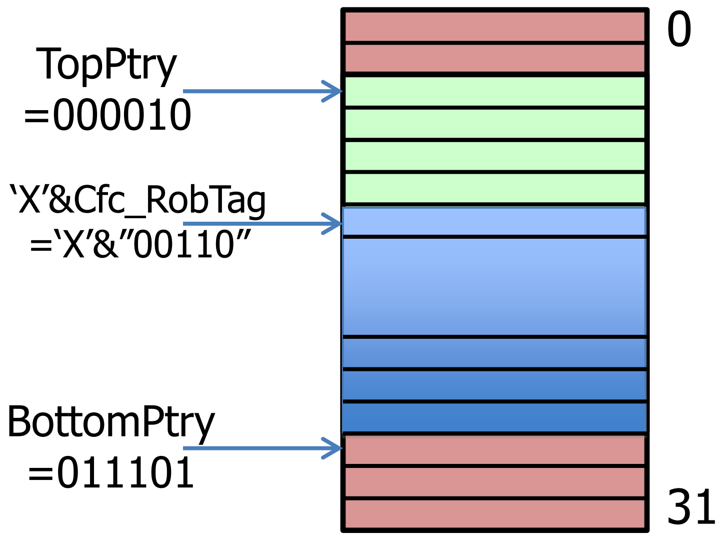
Flush in ROB



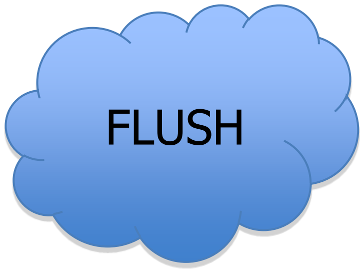
On the left shows the ROB state when Flush is announced. Bottom pointer is pointed to 29th location at that time.

🟩 Entries to be committed
🟥 Entries to be filled up

Flush in ROB

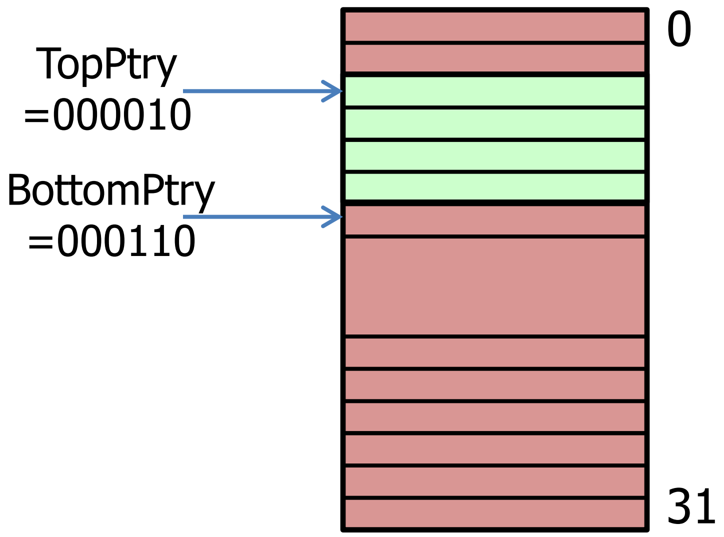


For phase 1, CFC provides the ROB tag (6) of the mis-predicted branch instruction or jr $31 instruction at the CDB.
For phase 2, CFC provides the ROB tag (6) of the nearest checkpointed instruction, or from CDB as in phase 1.



🟩 Entries to be committed
🟥 Entries to be filled up

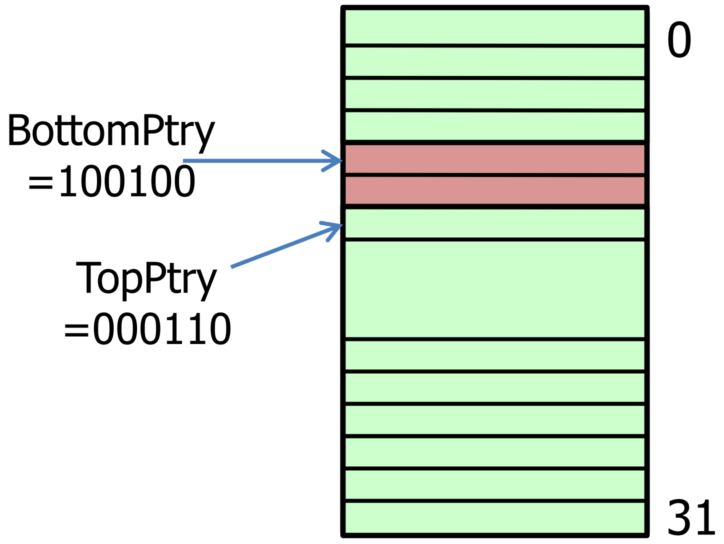
Flush in ROB



Then ROB moves bottom pointer to location 6 and flushes all entries in between.

🟩 Entries to be committed
🟥 Entries to be filled up

Full and almost Full

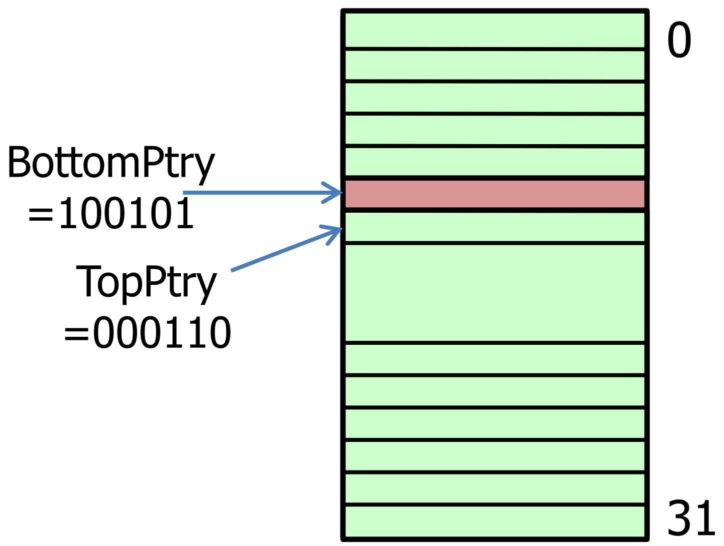


Left diagram shows how ROB becomes full, as we don’t have many instructions committed and in the meanwhile, we are dispatching a lot of instructions.

Left shows the marginal condition for signal: more than one vacant location.

🟩 Entries to be committed
🟥 Entries to be filled up

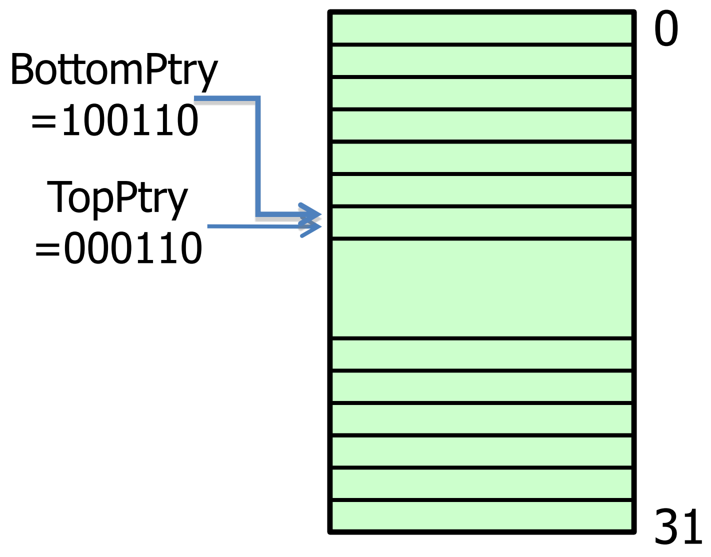
Full and almost Full



Left diagram shows how ROB becomes full, as we don’t have many instructions committed and in the meanwhile, we are dispatching a lot of instructions.

🟩 Entries to be committed
🟥 Entries to be filled up

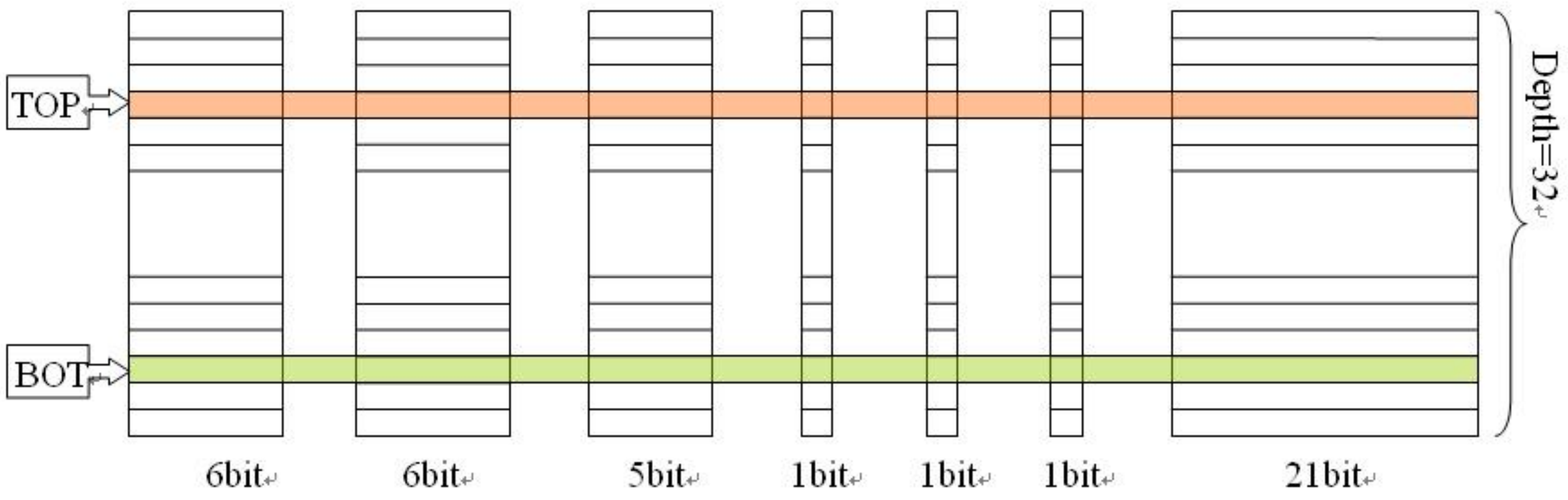
Full and almost Full

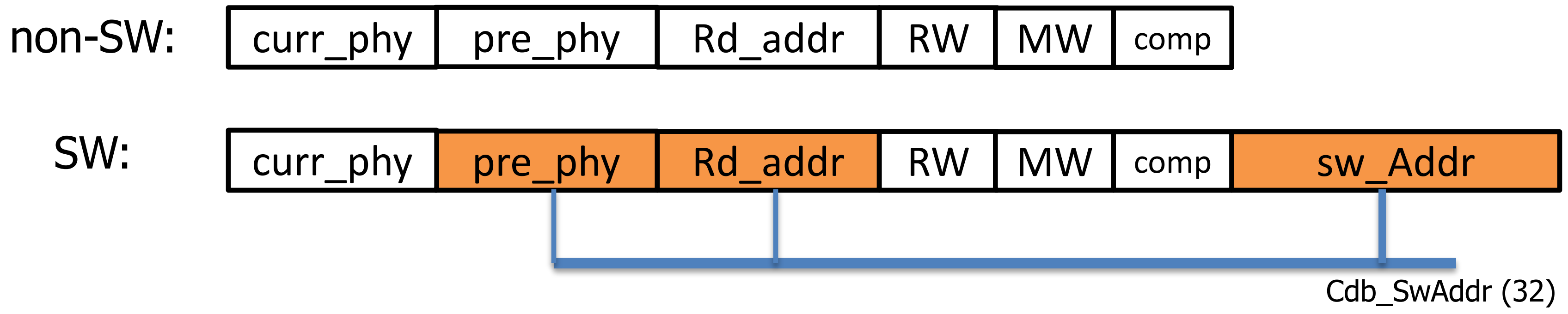


Left diagram shows how ROB becomes full, as we don't have many instructions committed and in the meanwhile, we are dispatching a lot of instructions.

As we keep dispatching, ROB reaches the following condition. **Bottom pointer for the first time points to an unavailable location, which indicates ROB full**.
In this condition, we can only commit and can not dispatch instructions any more.

ROB entry





curr_phy:	CurrPhyArray(i)	RW:	RegWriteArray(i)
pre_phy:	PrePhyArray(i)	MW:	MemWriteArray(i)
rd_addr:	RdAddrArray(i)	comp:	CompleteArray(i)
i:	Rob tag for indexing	sw_addr:	SwAddrArray(i)
			Used to store additional bits of store word address

ROB in Tomasulo

There are 4 critical steps in an instruction's life cycle:

- *Dispatch:*

  The instruction is dispatched from IFQ by the dispatch unit to the corresponding instruction issue queue according to the instruction's type.
- *Issue:*

  The issue unit issues instructions from different instruction issue queues to their corresponding execution units provided that all operands are ready and no conflict on the execution unit or the Cdb.
- *Execution:*

  After the instruction finishes its execution, it reports its result on the Cdb.
- *Commit:*

  When an instruction reaches the top of the ROB, it becomes the most senior instruction and hence can commit once it is ready.
- **ROB is associated with following steps: dispatch/execution/commit.**

How to update ROB

Each time an instruction is dispatched:

- The entry currently pointed by the bottom pointer of ROB is partially updated with the following contents
- Complete bit of this entry is set to 0,which means the instruction in this slot is waiting to be completed.
- Whether the dispatched instruction is memory write or register write will be informed to ROB and stored in this entry. Here we are using two 32-slot 1-bit arrays to store this information.
- If the instruction is not memory write instruction, three register IDs will be sent to ROB: current physical register ID, previous physical register ID and architectural register ID. If it is a memory write instruction, only the current physical register is sent to ROB at this moment.
- Bottom pointer is incremented by one, prepared for the next dispatched instruction.

How to update ROB

Each time an instruction appeared on CDB

- The entry will be indexed by the ROB tag of the instruction at CDB.
- The complete bit of the entry is set to 1 in ROB.
- If it is a memory writing instruction, ROB will also receive the memory address from CDB, which was calculated previously in LSQ and stored in ROB in 3 arrays separately.

How to update ROB

Each time an instruction commits from the top of ROB

- This instruction has to be (stored in ROB as an entry) pointed by the top pointer right now.
- The complete bit of this entry will be set to 0.
- Top pointer is incremented by one, prepared for the next instruction to commit.

How to update ROB

<u>Requirements for an instruction to be committed:</u>

- The complete bit of this entry has to be 1, which means that it has finished execution and come out from CDB.
- Either it is not a memory-write instruction, or it is a memory write instruction with the store buffer not full in the meanwhile.
- ROB can not be empty: top pointer not equal to bottom pointer. **Important: we don’t need to clear complete bit when flush as we have this condition.**

How to update ROB

When flushing happens

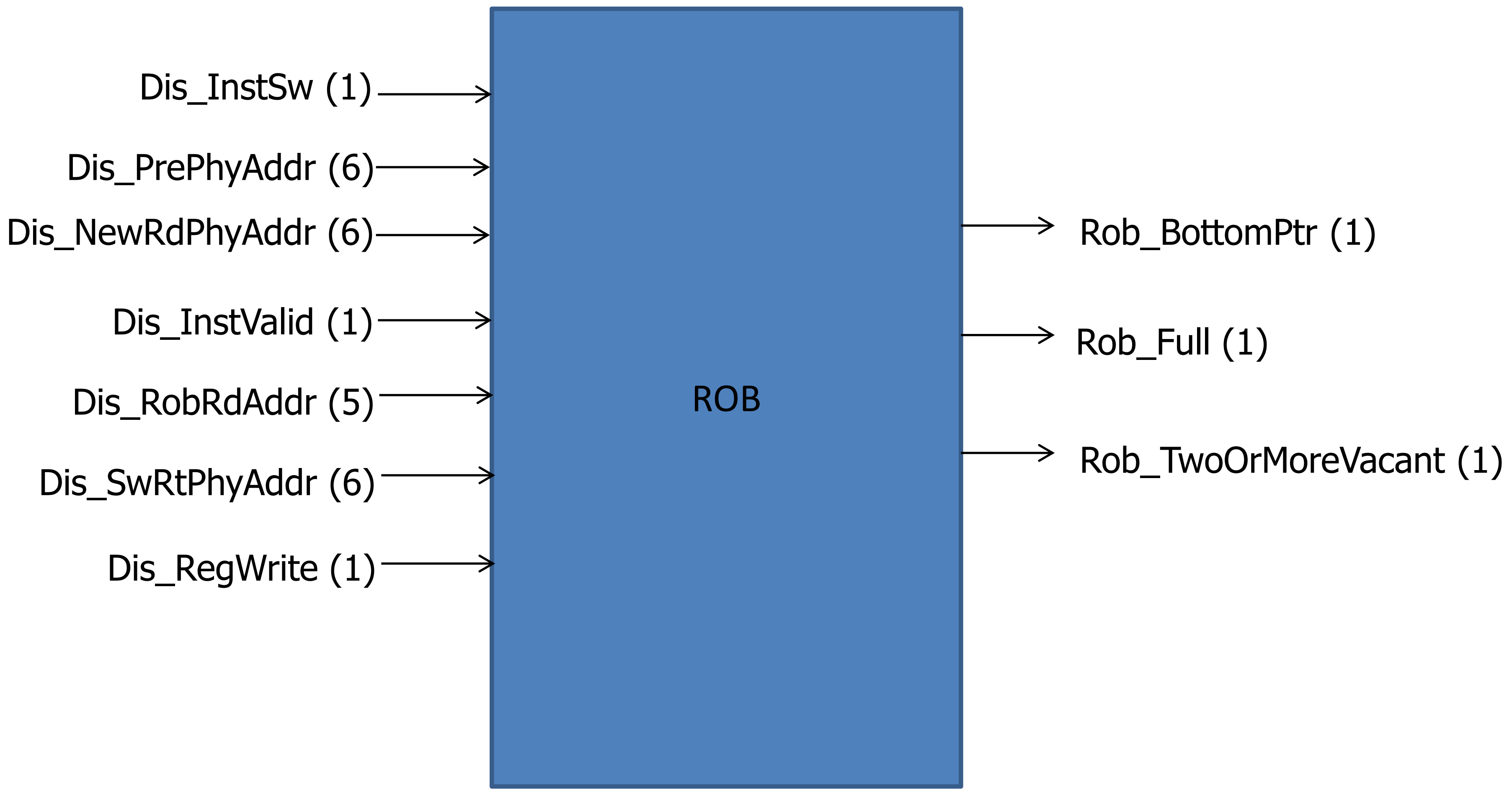
- Flushing signal can be generated only when a BEQ/JR 31 instruction arrives at CDB
- ROB tag of the instruction at CDB is used to generate a flush depth.
- For all instructions in ROB, we move the bottom pointer to invalidate/clear instructions that are not supposed to be processed.

To detect ROB full

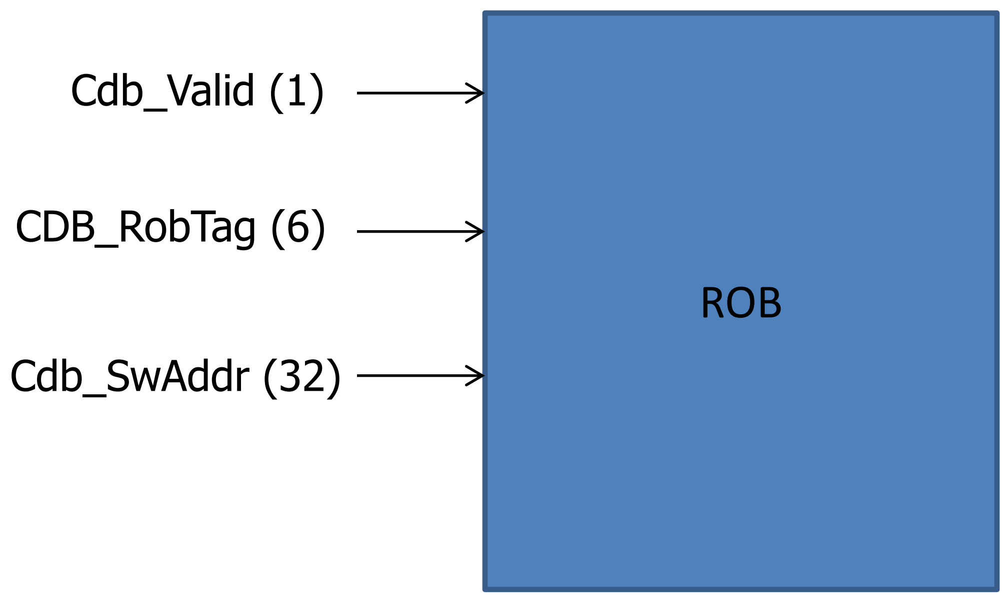
- ROB contains 32 entries, thus 5-bit pointers are enough to manage all entries. But as we need to check its full condition, we use 6-bit pointer in ROB.
- Consequently, the full condition is:

  top pointer ***XOR*** *bottom pointer = “100000”*

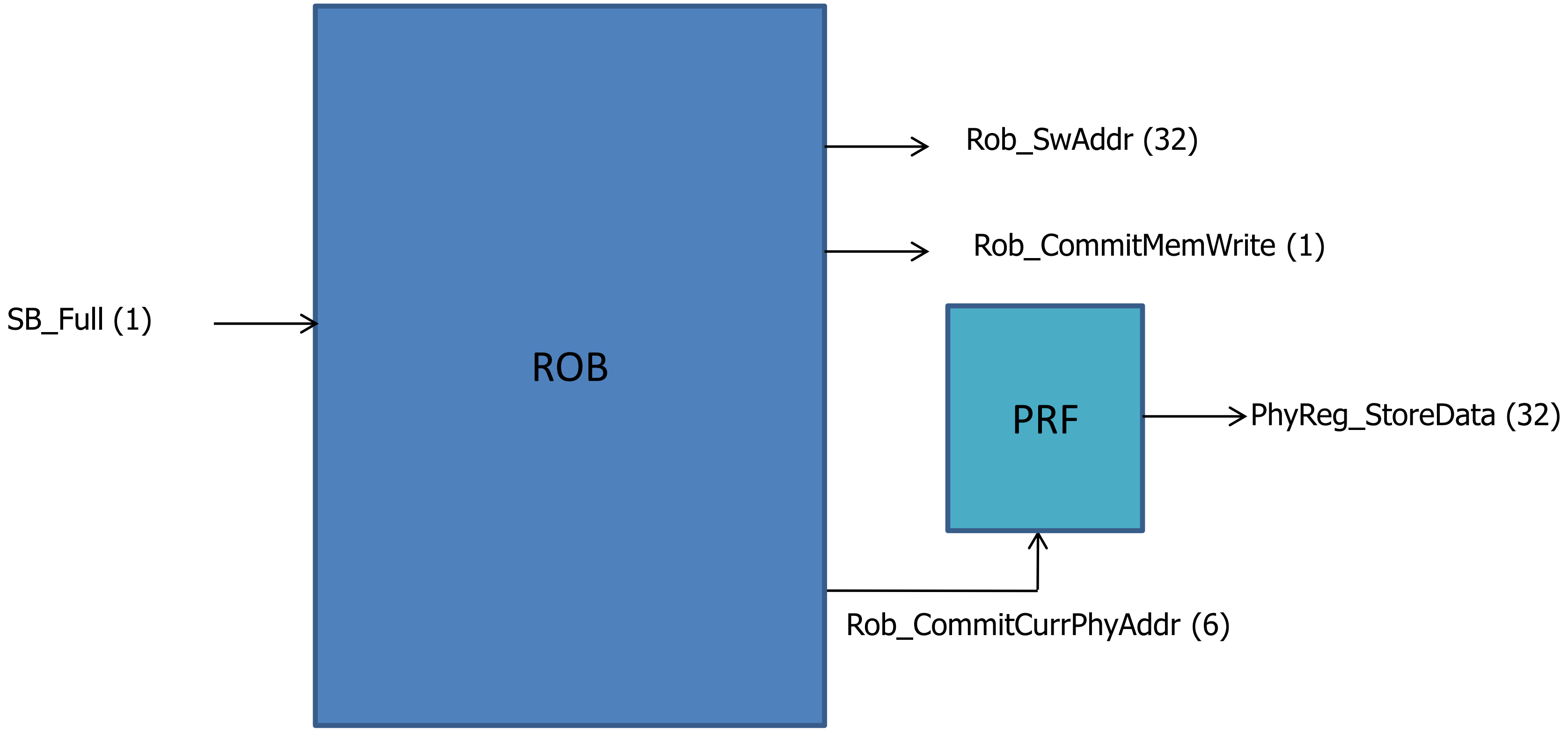
Interaction with Dispatch unit



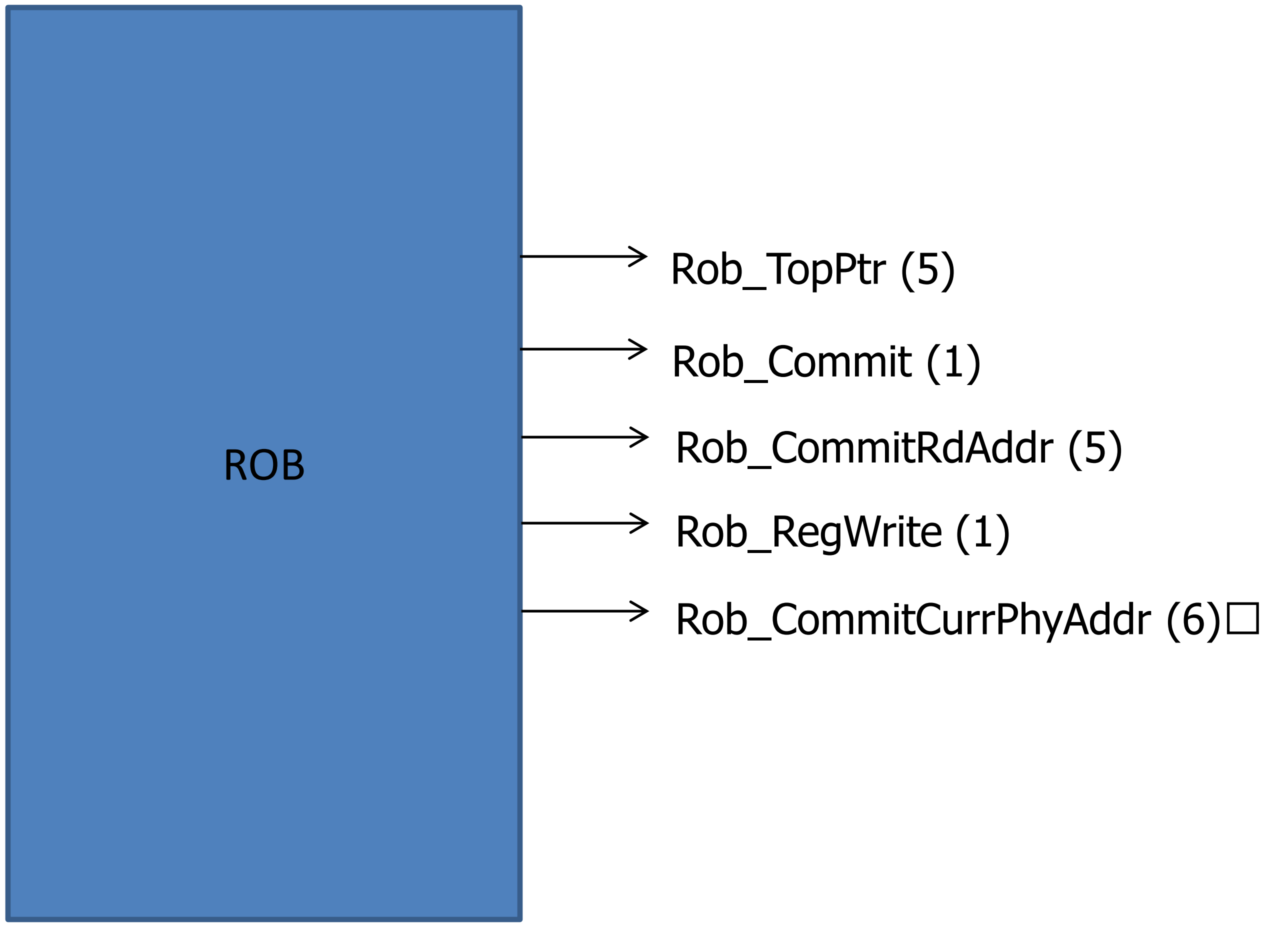
Interaction with CDB



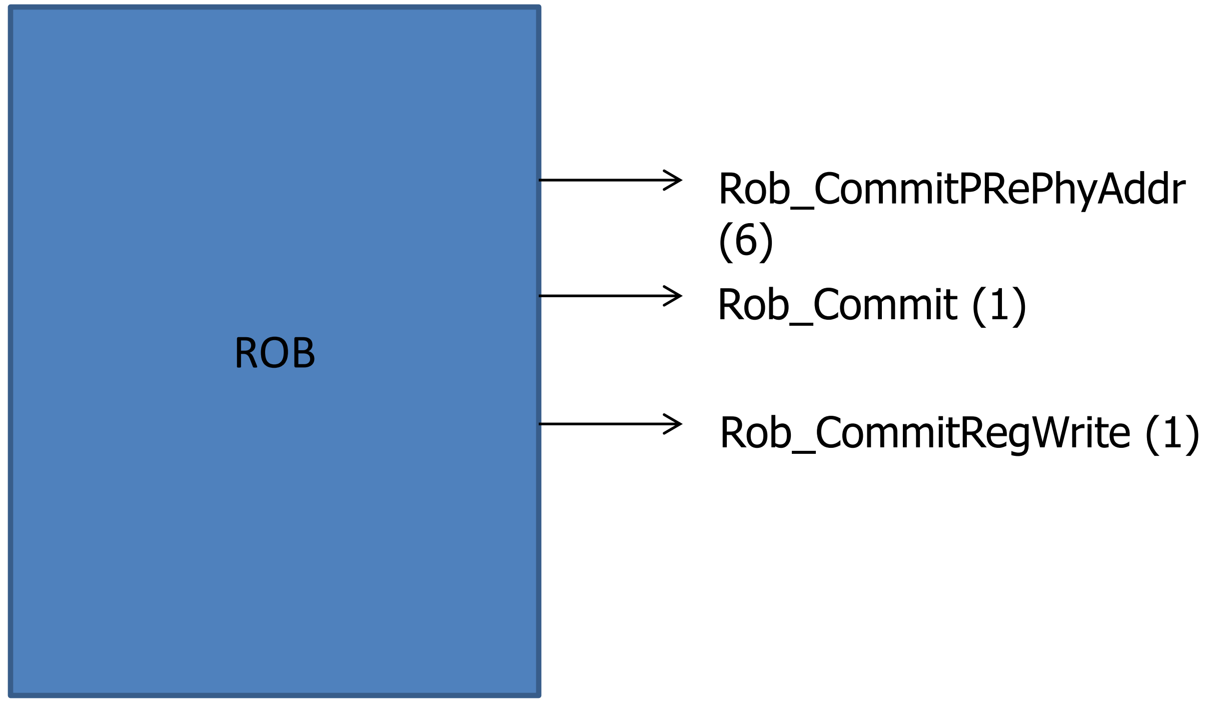
Interface with store buffer & Physical register file



Interface with CFC



Interaction with FRL


ROB
Rob_CommitPRePhyAddr (6)
Rob_Commit (1)
Rob_CommitRegWrite (1)