Hardware-Based Speculation

- Or "how it is in real life"
- *Dynamic branch prediction* to choose which instructions
- *Speculation* to allow instructions to execute before resolving control dependences
- *Dynamic scheduling* to schedule instructions

Why Hardware?

- Disambiguate memory references at run-time
- Handles statically difficult to predict branches
- Maintains a precise exception model
- No compensation or bookkeeping code
- Schedule code at run-time (avoids problem of different architectures)

Why Not Hardware?

- Complexity
- More complexity
- And still more complexity!
- Chip area, power, scalability, design, verification, and more....

Hardware-Based Speculation

- Extension of Tomasulo's approach
  - With speculative execution
  - With dynamic branch prediction
- Extension for speculative execution
  - Separate result bypassing from instruction completion
  - Let instruction execute and bypass results to other instructions without updating any state that can't be undone
  - Commit updates when instruction is no longer speculative
  - Speculative register read: We are "speculatively" reading the result of an instruction we aren't sure will be committed.
  - Instruction commit: When instruction is no longer speculative and architected state can be updated

OOO Execution, In-order Commit

- Let instructions execute out of order, but commit their results in order
- Thus, we know whether the instruction was issued from a speculative path

- Separate *completing execution* from *commit*
  - Instructions may execute well before they should be committed
  - Need buffers to hold values that haven't been committed yet
  - Reorder buffer: Holds speculative values, commits instructions from the buffer in order
  - Easy to undo execution on mispredicted branches or exceptions

Reorder Buffer

- Holds results of an instruction - *between when it finishes execution and commits*
- Extended register set - dynamic register renaming

- Reorder buffer is a *source of register values*, just like reservation stations in Tomasulo's approach

- Unlike Tomasulo's approach - an instruction that finishes execution *doesn't write results* to register file where they are read by subsequent instructions in Tomasulo's technique

Reorder Buffer

- Register file updated from reorder buffer
- Only when an instruction commits and is deemed to have come from the correct path (hence, it's no longer speculative)
- Also, replaces store and load buffers in Tomasulo's approach
- Flush buffer on branch misprediction - undo speculative actions

Reorder Buffer Structure

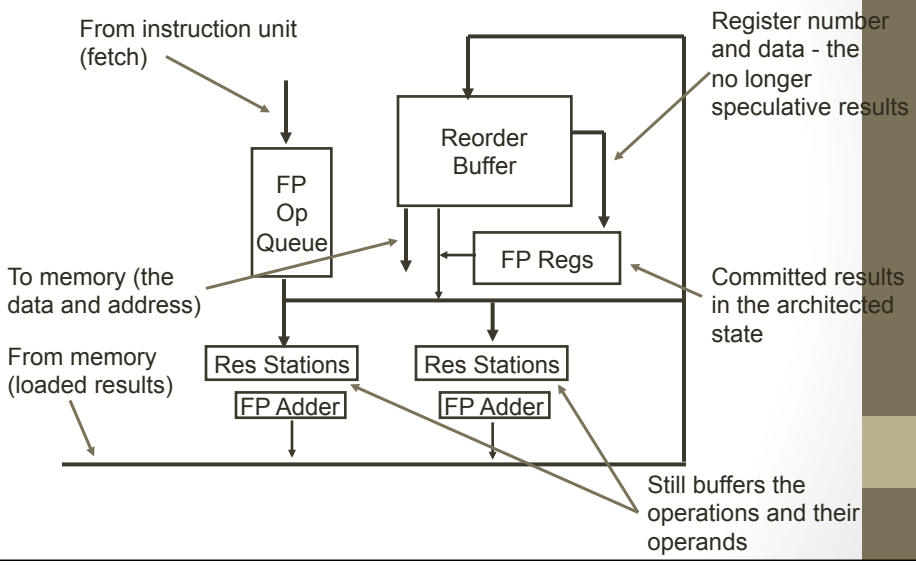
- Fields

  Instruction type
  - Branch - no destination
  - Store - memory address destination
  - Register operation - has destination register

  Destination
  - Where the value should be written (reg. num. or address)
  - Register number - for loads and register operations
  - Memory address - for stores

  Value
  - Value of instruction result
  - Held until instruction commits

Reorder Buffers



Instruction Execution Steps

- Issue (I)
  - get instruction from queue
  - issue when empty reservation station and reorder buffer
  - send operands (or their tag) from reorder buffer or regfile
  - send destination reorder buffer tag to reservation station
  - if reservation station full or reorder buffer full - stall issue
- Execute (EX)
  - monitor CDB for needed operands (checks for RAW hazards)
  - when operands available, execute

Instruction Execution Steps

- Write result (WR)
  - when result ready, write to CDB
  - send reorder buffer tag with result so reorder buffer can be updated with the value of the result
  - tag also used by any instructions waiting on this result (they were given the tag when issued)
  - ***mark reservation station as available***
- Commit (C)
  - when instruction reaches head of reorder buffer and result in the buffer, write it to register file and remove instruction
  - *how to handle mispredicted branches????*
    - For mispredicted branch - reorder buffer flushed and execution started at correct address

Reorder Buffer Implementation

- Circular queue
- Easy to update for insertion and removal of instruction entries
- Reclaim buffer entry of instruction result just written to the register file - pointer update for the queue
- When reorder buffer fills, stop issuing instructions until a new slot becomes available

Recovery

- Recall: with dynamic branch prediction, we want to keep misprediction recovery penalty small
- When mispredicted branch encountered
  - Flush buffer
  - In practice, try to detect early and begin to fetch from correct path as soon as possible
  - Flush buffer from the branch point downward

Exceptions

- ***How can we deal with exceptions???***
- Record exception status in reorder buffer
- Handle exceptions as instructions graduate from the reorder buffer
- Hence, exceptions are in-order and precise
- Very importantly, exceptions from misspeculated paths will be flushed when the reorder buffer is flushed on a branch misprediction