Quiz 2 Review

Ryan Lee

(Adapted from prior course offerings)

Quiz 2 logistics

- Time: 1pm EDT on Friday April 9
  - Prepare to receive an emailed PDF about 10 minutes before the quiz
- Zoom link: same as recitations
- Handout: Reservation stations & Store sets

Hazards

- Structural hazards
- Data hazards
- Control hazards
  - Not just branches and jumps!
  - Typically resolved by speculation (eager vs. lazy)

Complex pipelining

- Scoreboard
  - A data structure that detects hazards dynamically
  - Needed because
    - Many execution units
    - Variable execution latency
    - Dynamic instruction scheduling
  - Orthogonal to in-order vs. out-of-order issue

Out-of-order issue

- Strategy: find something else to do
- Difference from in-order issue
  - More hazards to consider (e.g., WAR and control)
- Techniques typically combined with OOO issue
  - Register renaming
    - Critical since it reduces/eliminates WAR and WAW hazards
  - In-order commit
    - Critical since it simplifies speculative execution
    - Speculation requires per-instruction buffering/logging
      - Partial flush is critical
      - Circular buffer management is preferred

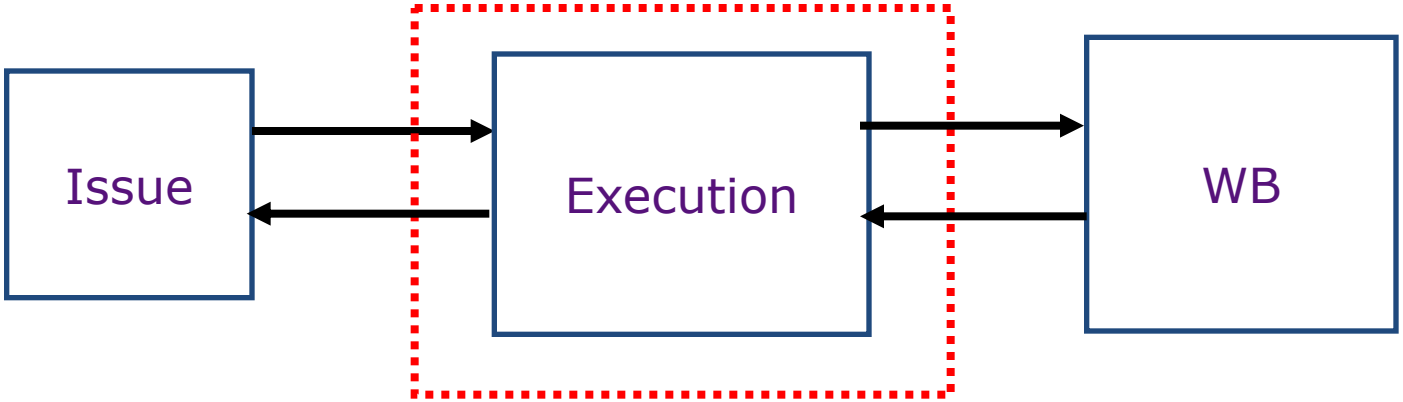
OOO design tradeoffs

- Implementations
  - Data-in-ROB
  - Unified-register-file
  - More!

- Tradeoffs
  - Are there pointers or values in ROB? Are register reads delayed or immediate?
  - Can speculative values share resources with non-speculative values?
  - Centralized ROB vs. reservation stations
  - ROB vs. issue queue + commit queue

Little's Law

Throughput ($\overline{T}$) = Number in Flight ($\overline{N}$) / Latency ($\overline{L}$)



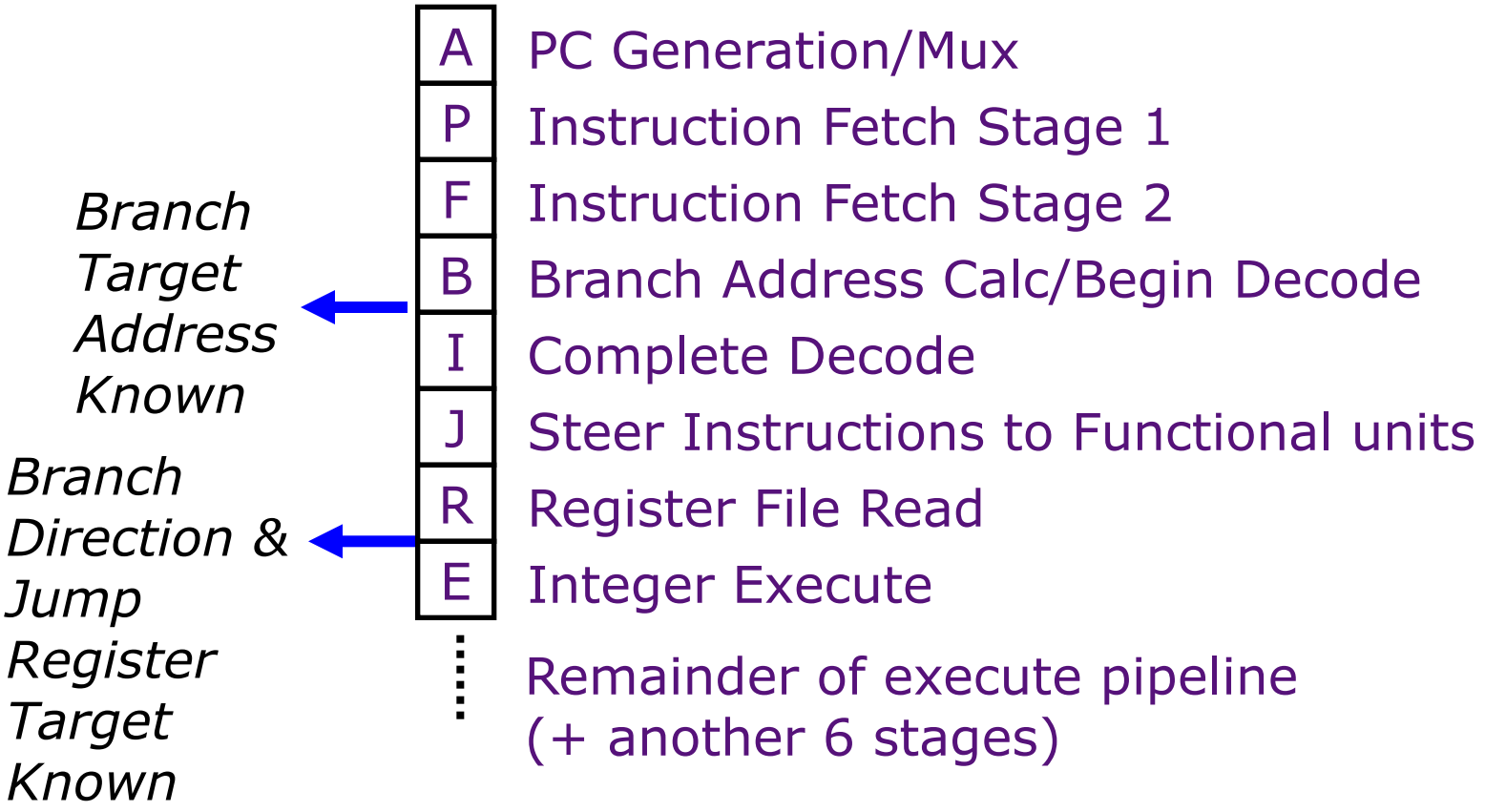
Example:

4 floating point registers
8 cycles per floating point operation

⇒ *½ issues per cycle!*

Branch prediction

- To reduce the control flow penalty



Branch prediction implementation

- Static vs. dynamic predictor
- Example: two-level branch predictor
  - Access a local/global history in the first level
  - Access a counter in the second level (with or without bits from PC)
- Branch target buffer
- Subroutine return stack
- …

Speculative Value Management

- When do we do speculation?
  - Branch prediction
  - Assume no exceptions/interrupts
  - Assume no memory dependency
  - …
- How do we manage speculative values?
  - Greedy (or eager) update
    - Update value in place
    - Maintain log of old values to use for recovery
  - Lazy update
    - Buffer the new value and leave old value in place
    - Replace the old value on commit

Advanced memory operations

- Write policy
  - Hits: write through vs. write back
  - Misses: write allocate vs. write no allocate

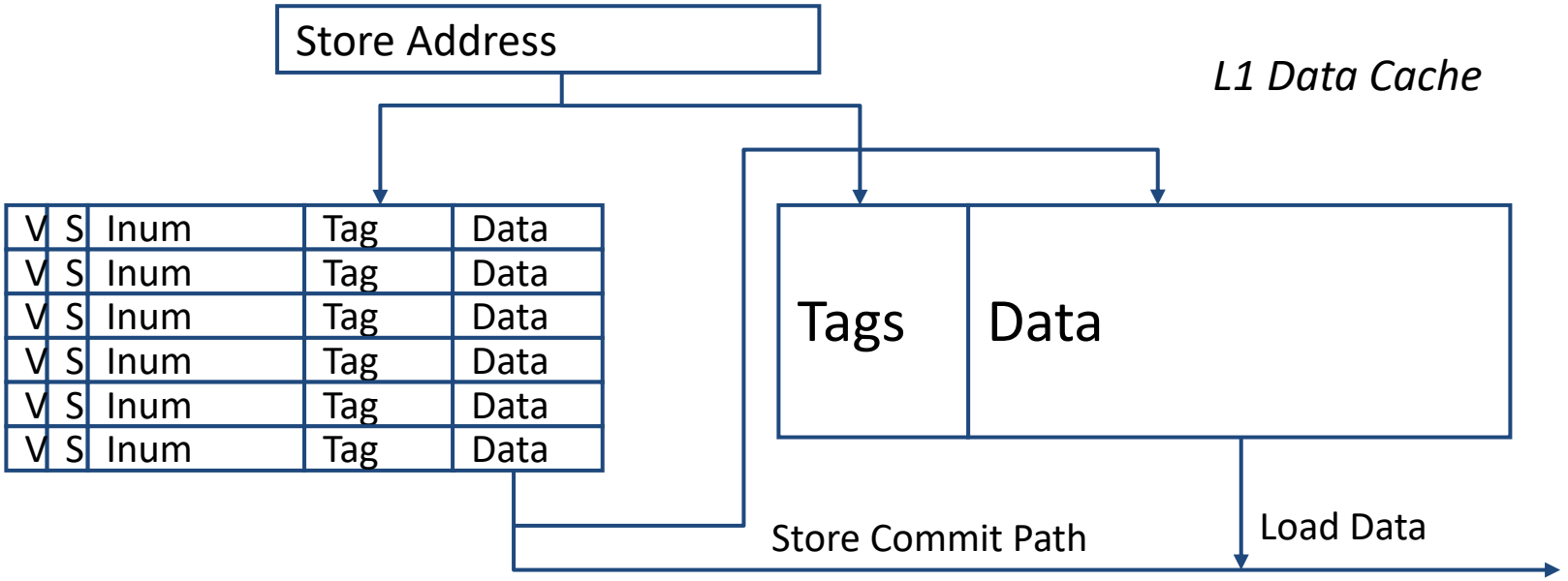
- Speculative loads/stores
  - Cause 1: control dependency
    - Just like other instructions
    - Solution: buffer the stores and commit them in order
  - Cause 2: (memory-location-based) data dependency
    - Simple solution: buffer stores; loads search addresses of all previous stores
    - Problem: addresses of previous stores may be unknown
    - Solution: speculate no data dependency
      - Use a data structure to keep track of this speculation: speculative load buffer

Store Buffer

» Enables data forwarding

» Handles OoO stores

» Handles speculative stores



» On store execute:

- mark valid and speculative; save tag, data and instruction number.

» On store commit:

- clear speculative bit and eventually move data to cache

» On store abort:

- clear valid bit

» One entry per store

» Written by stores

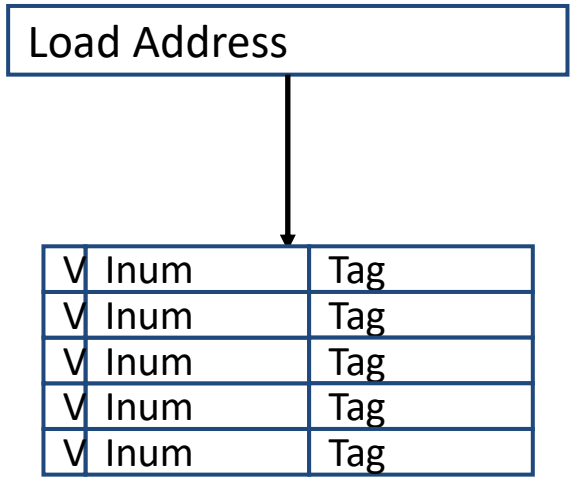
» Searched by loads

» Writes to data cache

Load Buffer

- On load execute:
  - mark entry valid, and instruction number and tag of data.
- On load commit:
  - clear valid bit
- On load abort:
  - clear valid bit

Speculative Load Buffer



- One entry per load
- Written by loads
- Searched by stores

- Enables aggressive load scheduling
- Detects ordering violations

Advanced memory operations

- Prefetching vs. on-demand data movement

Store sets

- Store set: A set of stores that a load is found to have depended upon
  - Naively execute loads and stores out of order.
  - After the load and store are found to conflict, we should predict that they will conflict next time.
- Enforce dependence between the store and the latest store in the store set
  - Stores in store sets execute in order

Wish you all the best!