

Quiz 1 Review

Ryan Lee

(Adapted from prior course offerings)

Quiz 1 Logistics

- Time: 1pm-2:30pm EDT on Friday, October 13th
- Location: 32-141
- Covered Materials: L01-09
- Additional handouts will be provided
 - Reservation stations + commit queue & physical register file
 - Predication for RISC-V

Please ask any questions you have!
(I hope to keep this interactive)

Agenda

- Caches
- Virtual Memory
- Pipelining
- Complex Pipelines
 - Scoreboarding
 - Register Renaming
 - Branch Prediction
 - OoO Issue & Reorder Buffer

Self-Modifying Code

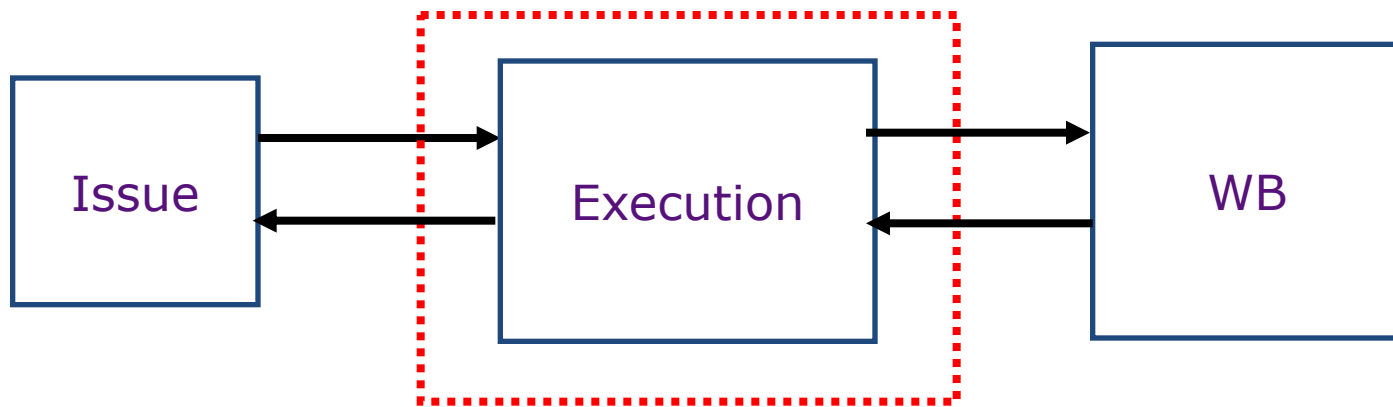
- Necessity in early days of computing due to lack of sufficient general-purpose registers
 - Accumulator-based
 - No concept of index registers, PC
 - Use self-modifying code for indirect accesses, subroutine calls, etc.
- Try out Problem set 1 & EDSACjr-based problems.

Caches

- Motivation: A small but fast storage that exploits locality
 - Decreases latency of access by exploiting *spatial* and *temporal* locality
- Allows you to achieve high throughput without large buffers: *Little's Law*
 - $Average\ Throughput(T) = \frac{Number\ of\ Requests\ in\ Flight\ (N)}{Average\ Latency\ (L)}$

Little's Law

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



Example:

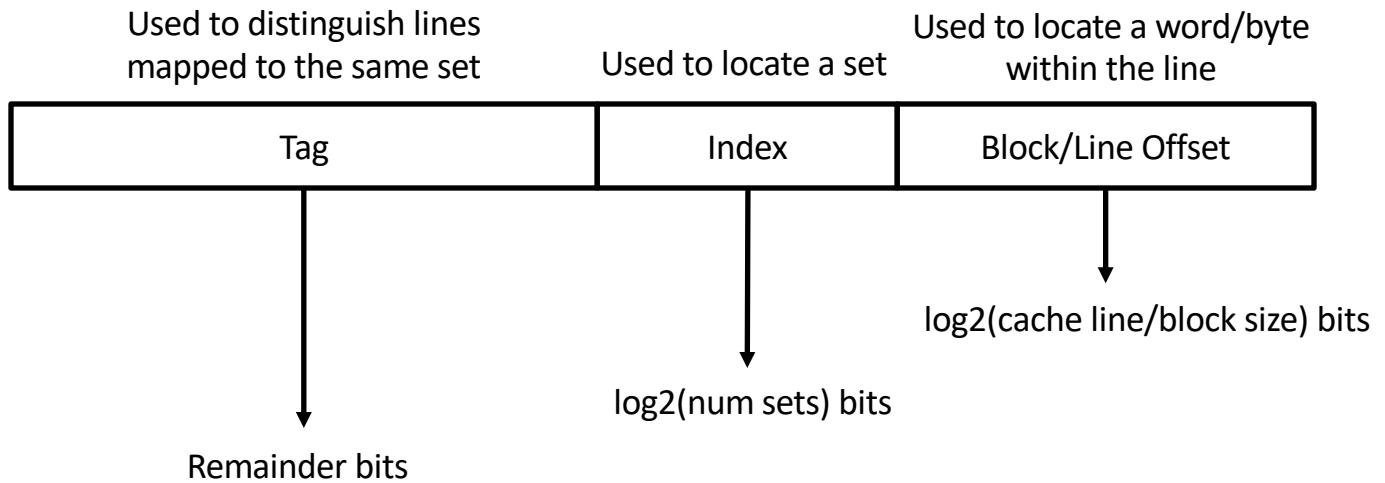
4 floating point registers
8 cycles per floating point operation

\Rightarrow *1/2 issues per cycle!*

Caches

- Performance metrics
 - $AMAT = \text{hit time} + \text{miss rate} * \text{miss penalty}$
- Design options
 - # of sets, # of ways
 - Block size
 - Replacement policy
 - Inclusivity and exclusivity
 - More to cover in the future lectures

Caches



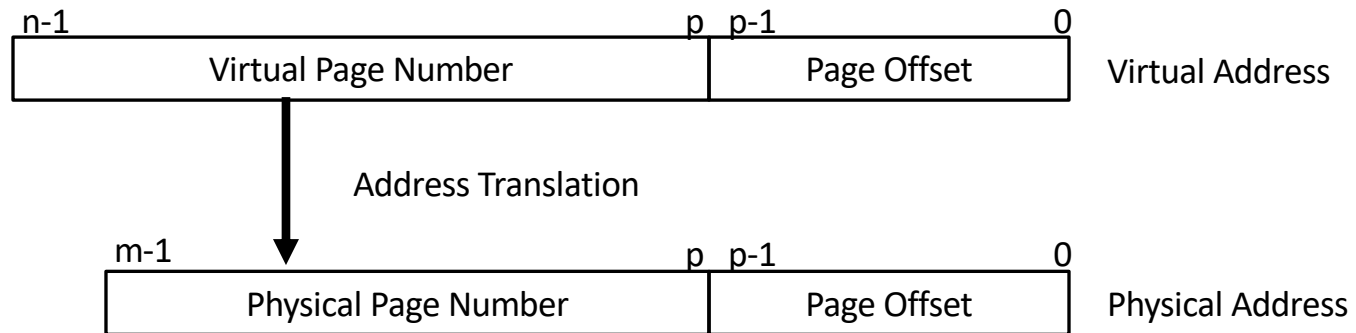
Virtual Memory

- A way to provide isolation and protection between programs
 - Segmentation with base & bound registers
 - Paged memory systems

Address Translation

Parameters

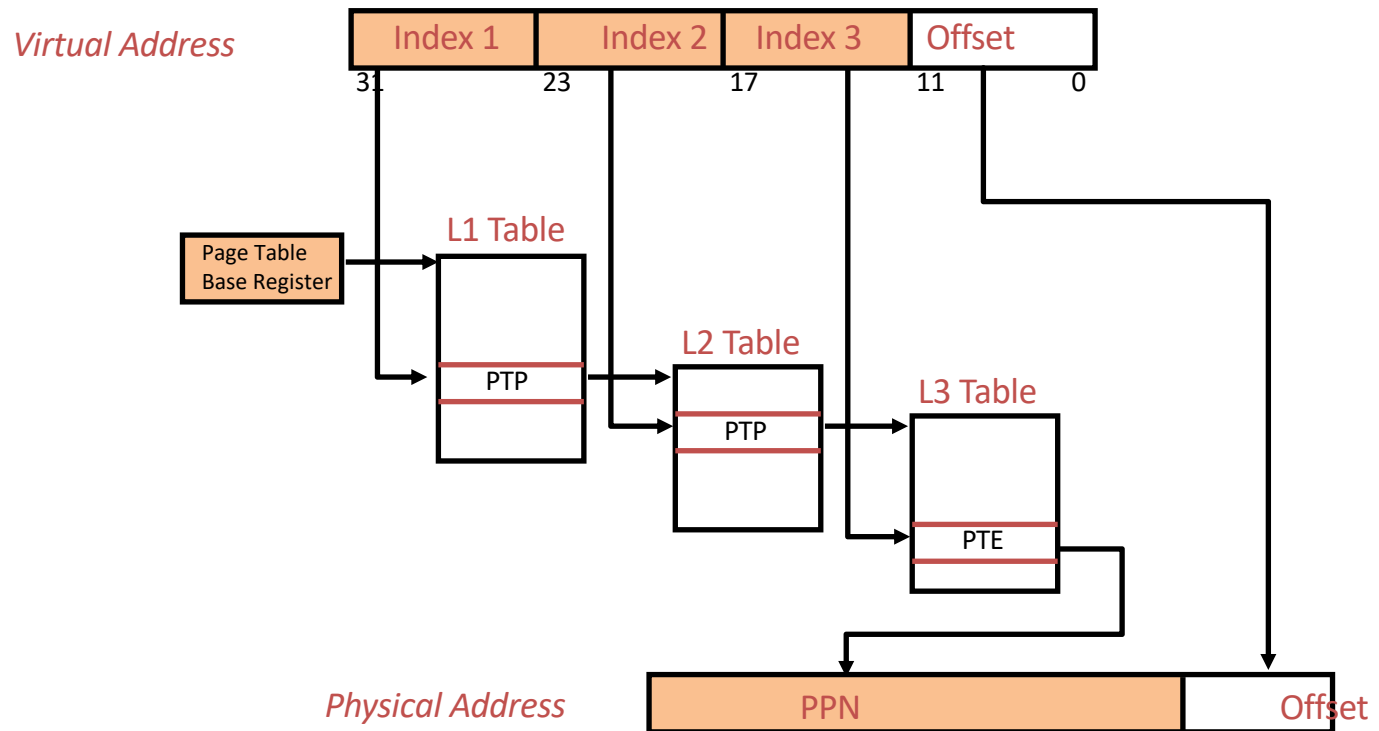
- $P = 2^p =$ page size (bytes).
- $N = 2^n =$ Virtual-address limit
- $M = 2^m =$ Physical-address limit



Page offset bits do not change with translation

Hierarchical Page Tables

- Virtual address space is *sparsely* populated



Translation Lookaside Buffer

- It is just another cache!
 - Holds VPN->PPN mappings to accelerate address translation
 - Can play all the tricks we did with data caches
 - Associativity
 - Replacement policy
 - Multiple levels
- TLB Miss -> Page Table walk

Pipelining

- Overlaps execution of multiple instructions: Pipeline parallelism
- Visualization
 - Instruction flow diagram
 - Resource usage diagram
- Hazard: an instruction cannot execute because
 - Resource is not ready: structural hazard
 - Data value is not ready: data hazard
 - PC is not ready: control hazard

Strategies to resolve hazards

- Stall
- Bypass
- Speculate
- Do something else

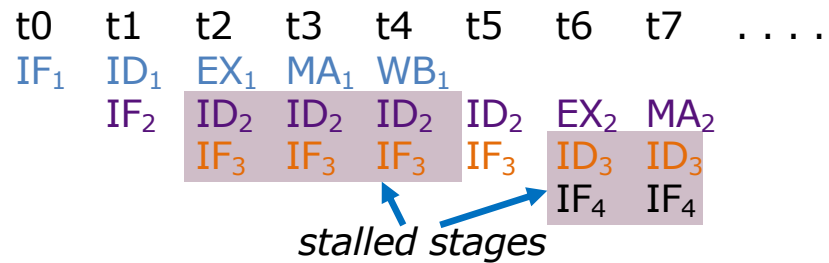
Instruction Flow Diagram

Assume no Bypassing:

I1: $r1 \leftarrow r0 + 10$
 I2: $r3 \leftarrow M[r1 + 10]$
 I3: $r4 \leftarrow r3 + 1$

... *time*

(I₁) $r1 \leftarrow r0 + 10$
 (I₂) $r3 \leftarrow M[r1 + 10]$
 (I₃) $r4 \leftarrow r3 + 1$
 (I₄)
 (I₅)



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

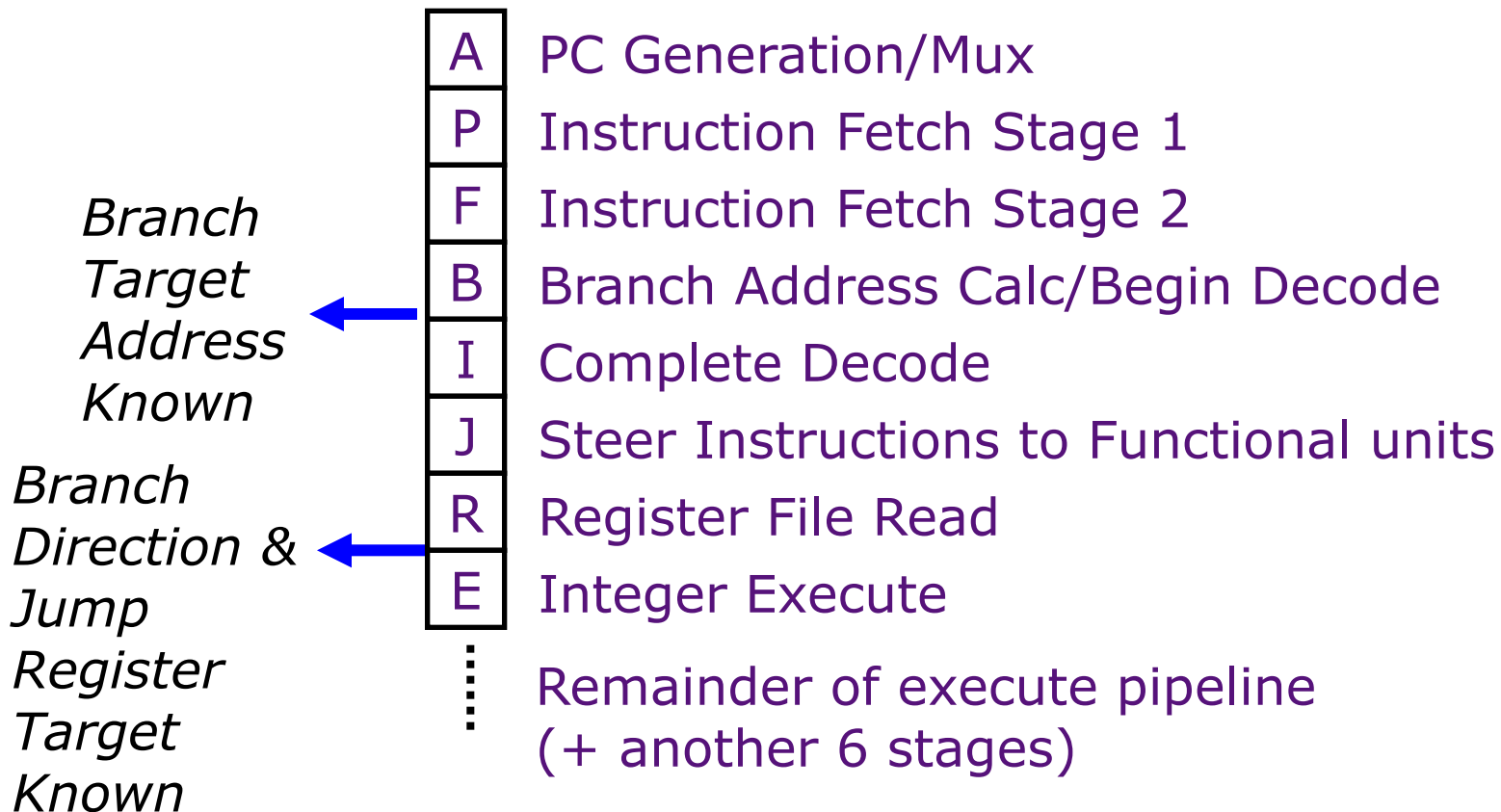
- Strategy: find something else to do
- Difference from in-order issue
 - More hazards to consider (e.g., WAR and control)
- Techniques typically combined with OOO issue
 - Register renaming
 - Critical since it reduces/eliminates WAR and WAW hazards
 - In-order commit
 - Critical since it simplifies 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
- 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

Branch prediction

- To reduce the control flow penalty

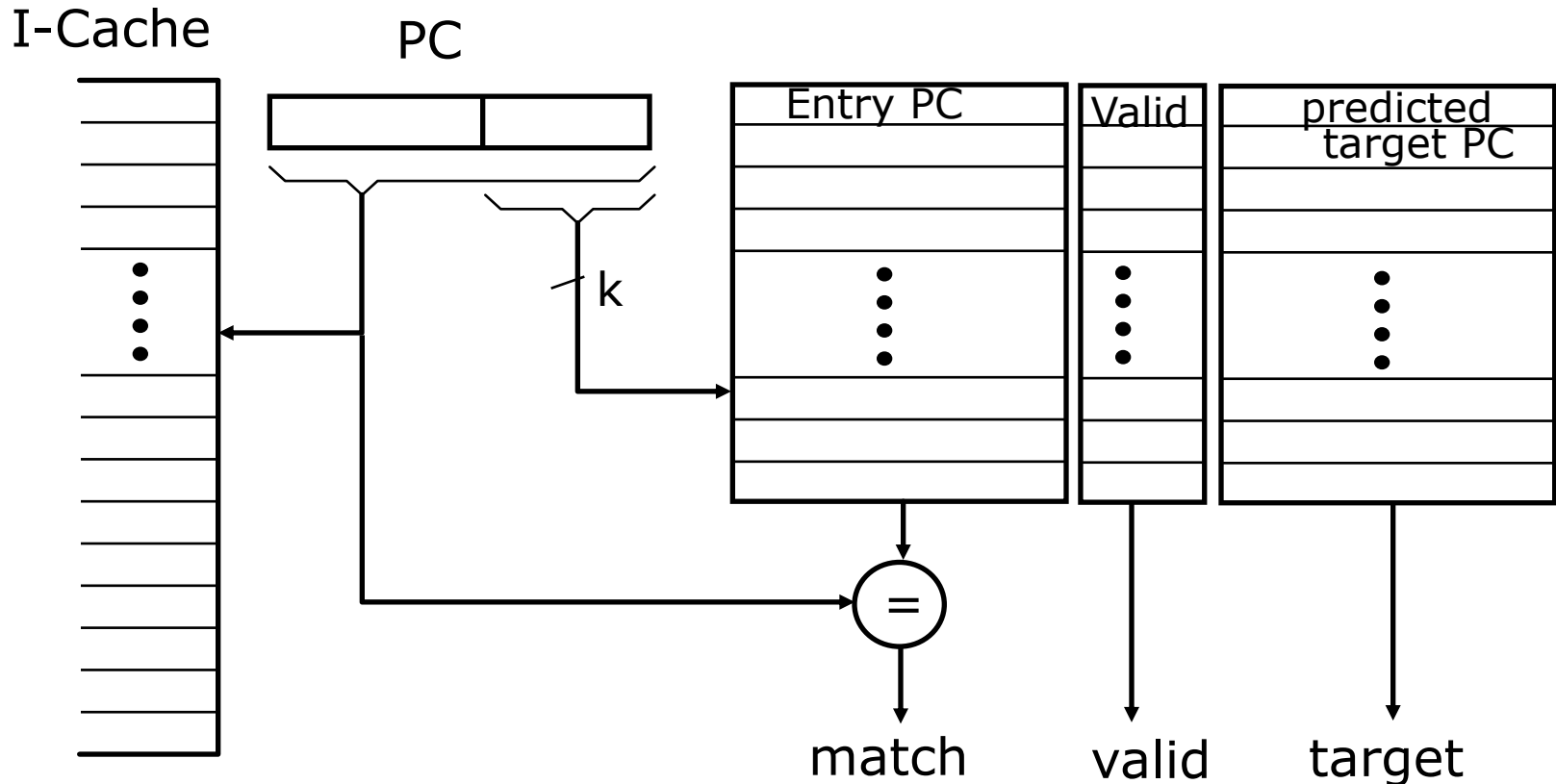


Predicting Branch Direction

- 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)

Predicting Branch Target

- Tight loop to produce next PC every cycle
- Example: 2^k entry direct-mapped BTB



Questions?

- Caches
- Virtual Memory
- Pipelining
- Complex Pipelines
 - Scoreboarding
 - Register Renaming
 - Branch Prediction
 - OoO Issue & Reorder Buffer