Quiz 1 Review

Ryan Lee

(Adapted from prior course offerings)

Quiz 1 Logistics

- Time: 1pm-2:30pm EDT on Friday, October 15^{th}
- Location: 32-141
- Covered Materials: L01-09
- Additional handout will be provided
  - Data-in-ROB style OoO Processor
- Closed book, no calculators.

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
  - Specultive value management

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.

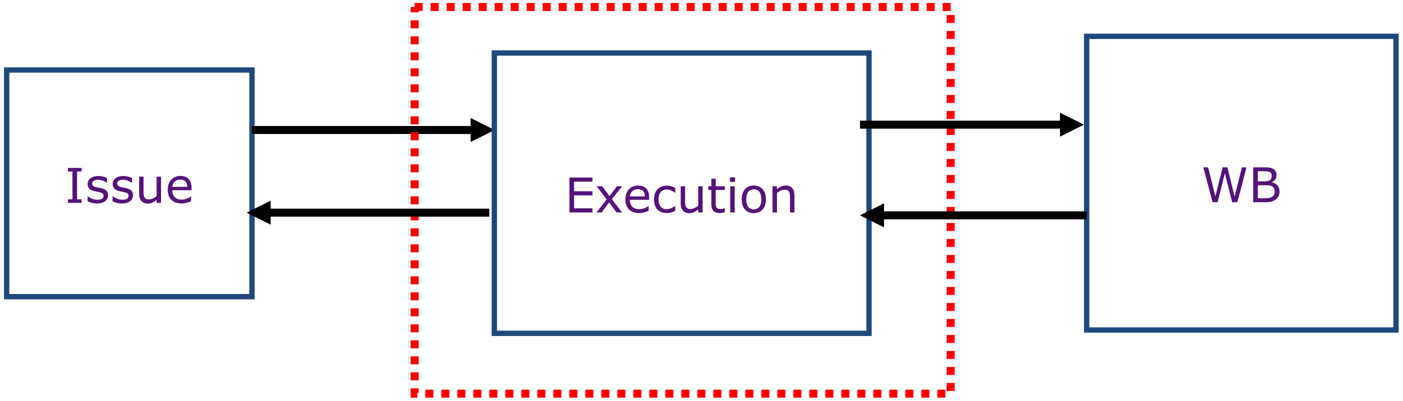
- Try out Problem set 1 & EDSACjr-based problems.

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

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



Example:

4 floating point registers
8 cycles per floating point operation

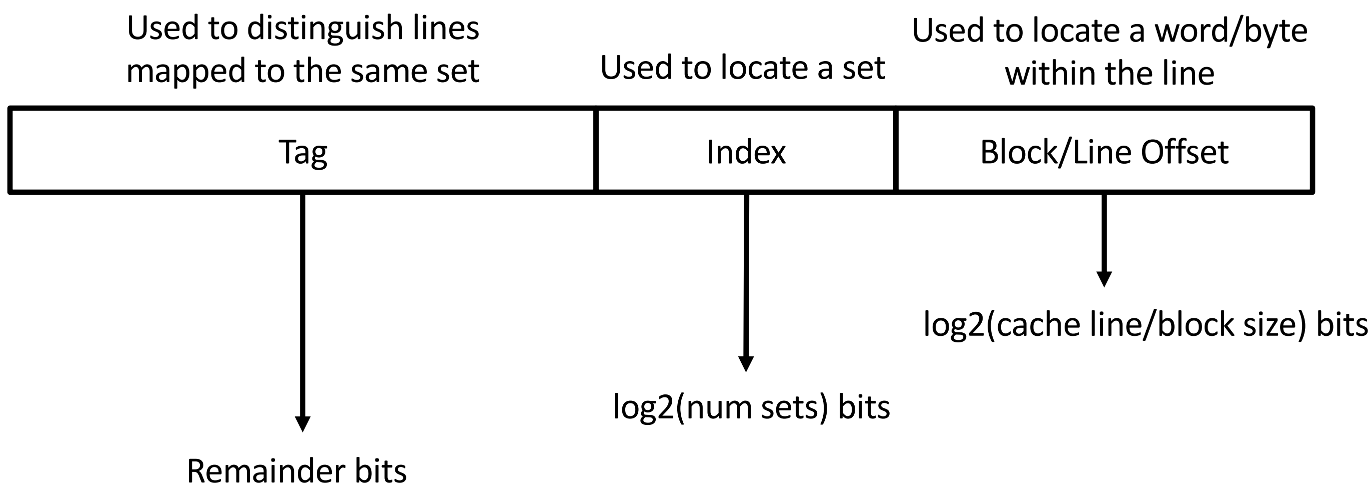
$\Rightarrow$ *½ issues per cycle!*

Caches

- Performance metrics
  - AMAT = hit time + miss rate * miss penalty

- Design options
  - # of sets, # of ways
  - Block size
  - Replacement policy
  - Inclusivity and exclusivity
  - More to cover in the future lectures

Caches


Used to distinguish lines mapped to the same set
Used to locate a set
Used to locate a word/byte within the line
Tag
Index
Block/Line Offset
log2(cache line/block size) bits
log2(num sets) bits
Remainder bits

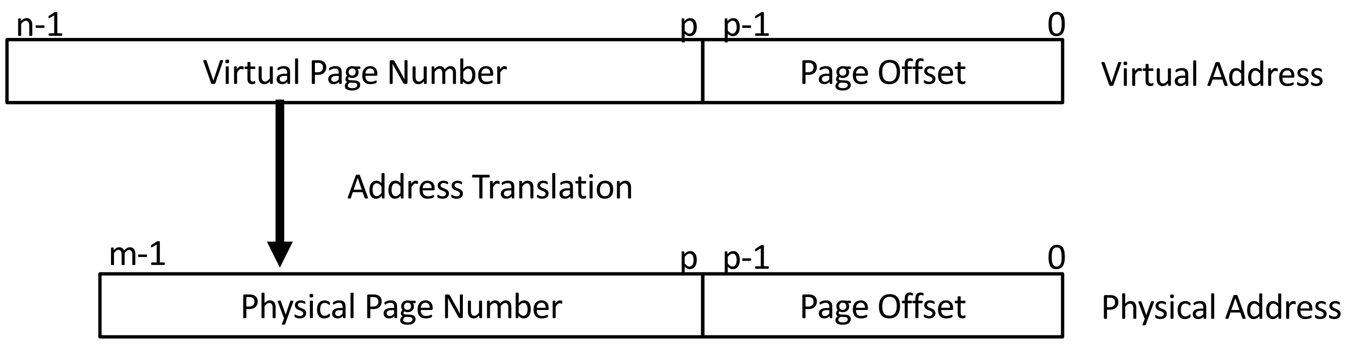
Virtual Memory

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

Address Translation

Parameters

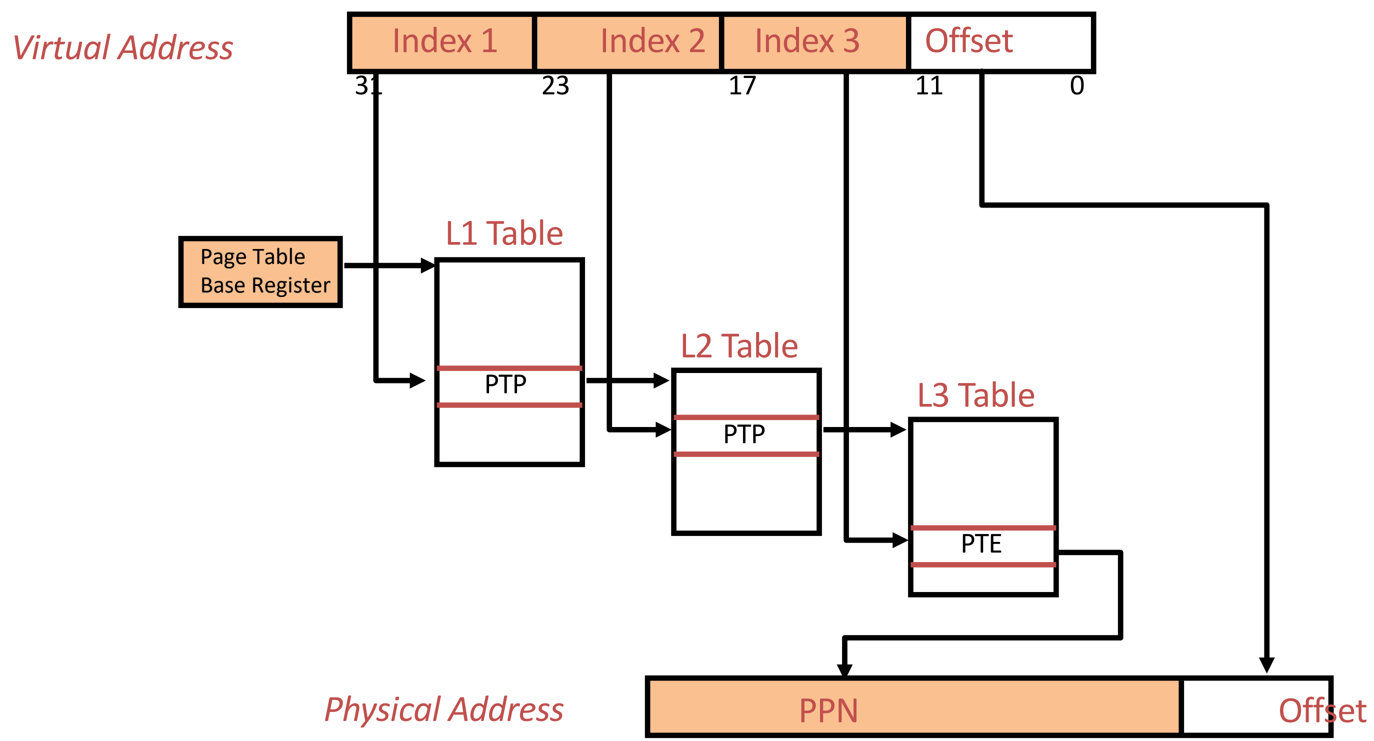
- $P = 2^p$ = page size (bytes).
- $N = 2^n$ = Virtual-address limit
- $M = 2^m$ = Physical-address limit



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 <- r0 + 10
I2: r3 <- M[r1 + 10]
I3: r4 <- r3 + 1
...

time	t0	t1	t2	t3	t4	t5	t6	t7	. . . .
(I_1) r1 $\leftarrow$ r0 + 10	IF_1	ID_1	EX_1	MA_1	WB_1				
(I_2) r3 $\leftarrow$ M[r1 + 10]		IF_2	ID_2	ID_2	ID_2	ID_2	EX_2	MA_2	
(I_3) r4 $\leftarrow$ r3 + 1			IF_3	IF_3	IF_3	IF_3	ID_3	ID_3	
(I_4)							IF_4	IF_4	
(I_5)									

stalled stages

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

	A	PC Generation/Mux
	P	Instruction Fetch Stage 1
	F	Instruction Fetch Stage 2
Branch Target Address Known ←	B	Branch Address Calc/Begin Decode
	I	Complete Decode
	J	Steer Instructions to Functional units
Branch Direction & Jump Register Target Known ←	R	Register File Read
	E	Integer Execute
	⋮	Remainder of execute pipeline (+ another 6 stages)

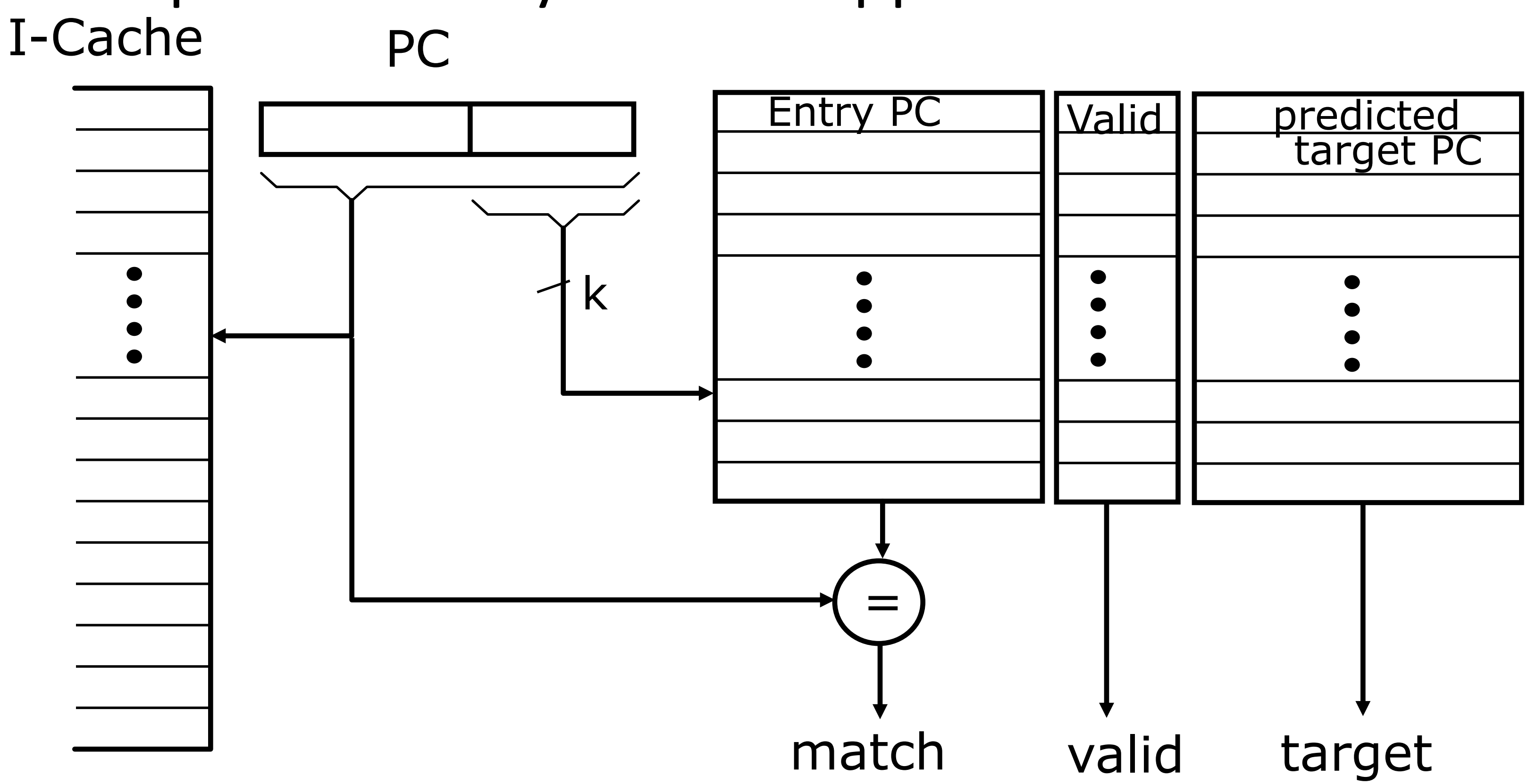
Predicting Branch Direction

- 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



Speculative Value Management

- When do we do speculation?
  - Branch prediction
  - Assume no exceptions/interrupts
  - …

- 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

Questions?

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