18-344 Recitation 4

09/26/2025

Outline

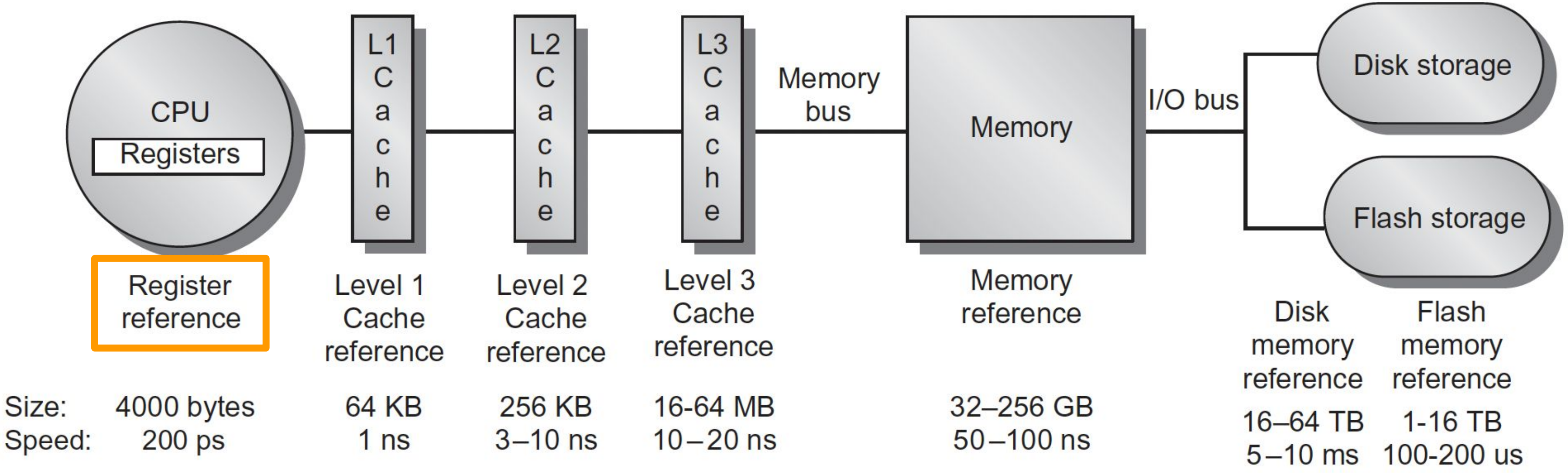
1. Logistics
2. Review
3. Homework 1 postmortem

Logistics

- Lab 0 and Homework 1 grades have been released
- Lab 1 released
    - Code freeze: September 28 (Sunday) (in 2 days)
        - Submit (push a commit) on GitHub Classroom
    - Report submission: October 1 (Wednesday) (in 5 days)
        - Submit on Gradescope
- Homework 3 released, due September 29 (Monday) (in 3 days)
    - Covers materials from lectures 7-9
        - Mostly caching
- Homework 2 grades to be released around this weekend/early next week

Review

The memory hierarchy



Memory hierarchy for server

Computer Architecture: A Quantitative Approach 6th Edition, Hennesy and Patterson

Memory access time

Generally,

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

Caches’ 3 C’s

- Compulsory miss: The first access to a block address is always a miss
- Capacity miss: Cache is too small to hold everything long enough so they can be reused
  - Think of it as a non-compulsory miss that would occur in an ideal fully-associative cache
- Conflict miss: Miss to a previously visited block that was evicted due to another block address being mapped to the same cache block
  - If a miss isn’t a compulsory or a capacity miss, it’s a conflict miss.

Cache replacement policies

- Round-robin
- Least-recently used (LRU)

Cache replacement policies

- Round-robin
- Least-recently used (LRU)

```
accessCacheLRU(access a){
  for each block in cache, b:
    if b != a.block:
      LRU_Age[b]++
  LRU_Age[b] = 0
}


findBlockLRU(){
  return argmax(b,LRU_Age)
}
```

Cache replacement policies

- Round-robin
- Least-recently used (LRU)
    - Expensive to implement
        - Area & Power cost to store ages
        - Time & Energy cost to update ages and identify the block to evict

Cache replacement policies

- Round-robin
- Least-recently used (LRU)
  - Expensive to implement
    - Area & Power cost to store ages
    - Time & Energy cost to update ages and identify the block to evict
- Bit-pseudo-least-recently used (BPLRU)
  - Evict a block that was definitely not most recently used, decent approximation of LRU

Cache replacement policies

- Round-robin
- Least-recently used (LRU)
  - Expensive to implement
    - Area & Power cost to store ages
    - Time & Energy cost to update ages ar
- Bit-pseudo-least-recently used (BPLRU
  - Evict a block that was definitely not most re

```
accessCachePLRU(access a){
  MRU_Bit[a.block] = 1
  if ++MRU_BitSum == setSize:
    for each block in cache, b:
      MRU_Bit[b] = 0
    MRU_BitSum = 0
}


findBlockLRU(){
  for i in 0..setSize:
    if !MRU_Bit[i]:
      return block(i);
}
```

Cache replacement policies

- Round-robin
- Least-recently used (LRU)
  - Expensive to implement
    - Area & Power cost to store ages
    - Time & Energy cost to update ages and identify the block to evict
- Bit-pseudo-least-recently used (BPLRU)
  - Evict a block that was definitely not most recently used, decent approximation of LRU
  - Not as expensive as LRU

Cache replacement policies

- Round-robin
- Least-recently used (LRU)
    - Expensive to implement
        - Area & Power cost to store ages
        - Time & Energy cost to update ages and identify the block to evict
- Bit-pseudo-least-recently used (BPLRU)
    - Evict a block that was definitely not most recently used, decent approximation of LRU
    - Not as expensive as LRU
- Belady’s MIN algorithm (MIN)
    - Optimally evict a block that has the longest reuse distance

Cache replacement policies

- Round-robin
- Least-recently used (LRU)
    - Expensive to implement
        - Area & Power cost
        - Time & Energy cos
- Bit-pseudo-least-recent
    - Evict a block that was de
    - Not as expensive as LRU
- Belady's MIN algorithm
    - Optimally evict a block th

```
findBlockMIN(){
    //init reuse distances
    for each block in cache, b:
        RD[b] = 0; RD_done[b] = false;
    //look forward in the execution trace
    for each access, a, forward in execution trace:
    //increment reuse distance for each block not already seen
        for each block in cache, b:
            if RD_done[b] == false:
                RD[b]++;
        RD_done[a.block] = true
    //MIN finds the block with maximum RD
    return argmax(b,RD[b])
```

Cache replacement policies

- Round-robin
- Least-recently used (LRU)
  - Expensive to implement
    - Area & Power cost
    - Time & Energy cos
- Bit-pseudo-least-recentl
  - Evict a block that was de
  - Not as expensive as LRU
- Belady's MIN algorithm
  - Optimally evict a block th

```
findBlockMIN(){
    //init reuse distances
    for each block in cache, b:
        RD[b] = 0; RD_done[b] = false;
    //look forward in the execution trace
    for each access, a, forward in execution trace:
    //increment reuse distance for each block not already seen
        for each block in cache, b:
            if RD_done[b] == false:
                RD[b]++;
        RD_done[a.block] = true
    //MIN finds the block with maximum RD
    return argmax(b,RD[b])
```

Cache replacement policies

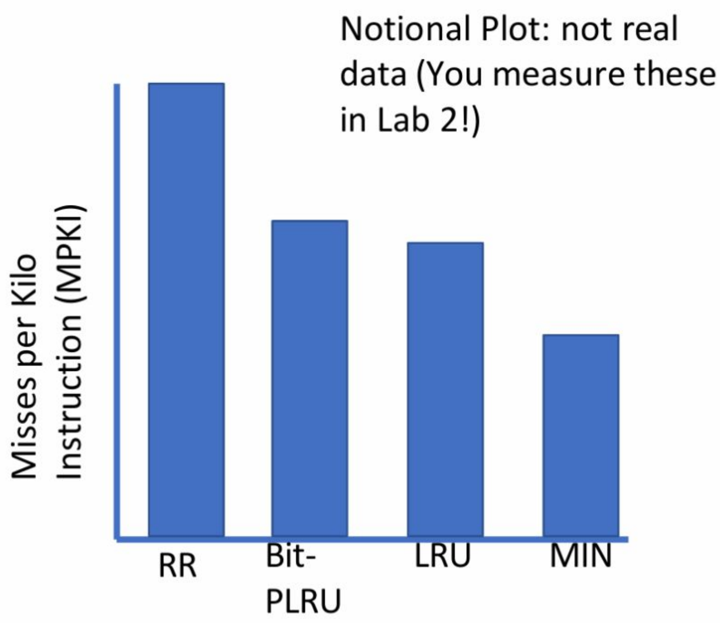
- Round-robin
- Least-recently used (LRU)
  - Expensive to implement
    - Area & Power cost to store ages
    - Time & Energy cost to update ages and identify the block to evict
- Bit-pseudo-least-recently used (BPLRU)
  - Evict a block that was definitely not most recently used, decent approximation of LRU
  - Not as expensive as LRU
- Belady’s MIN algorithm (MIN)
  - Optimally evict a block that has the longest reuse distance
  - Not realistic to implement

Cache replacement policies

- Round-robin
- Least-recently used (LRU)
  - Expensive to implement
    - Area & Power cost to store ages
    - Time & Energy cost to update ages and identify the block to evict
- Bit-pseudo-least-recently used (BPLRU)
  - Evict a block that was definitely not most recently used, decent approximation of LRU
  - Not as expensive as LRU
- Belady's MIN algorithm (MIN)
  - Optimally evict a block that has the longest reuse distance
  - Not realistic to implement

Capacity miss: Think of it as a non-compulsory miss that would occur in an ***ideal*** fully-associative cache.

Comparing cache replacement policies



RR: log(set size) bits per set to track next to evict, no action on access

Bit-PLRU: 1 MRU bit per block + log(set size) bits per set (or equivalent logic) to detect all set,
Clear bits on access if all bits set

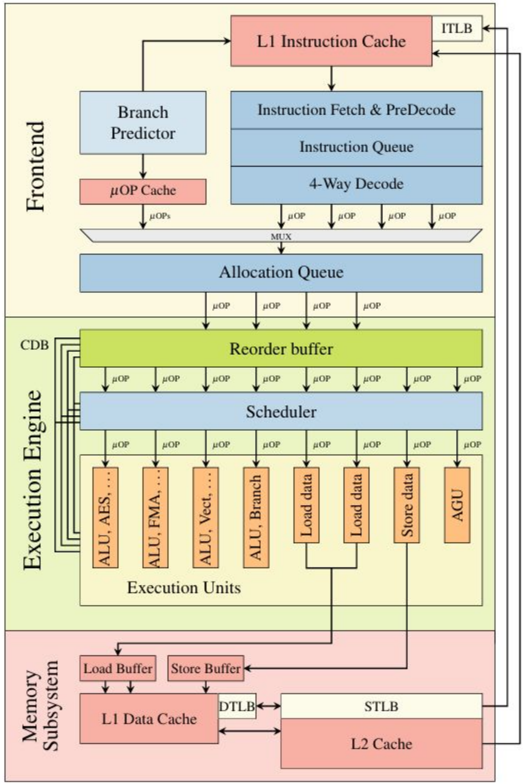
LRU: 1 age per block + logic to track max.
Update (set size - 1) ages on any access

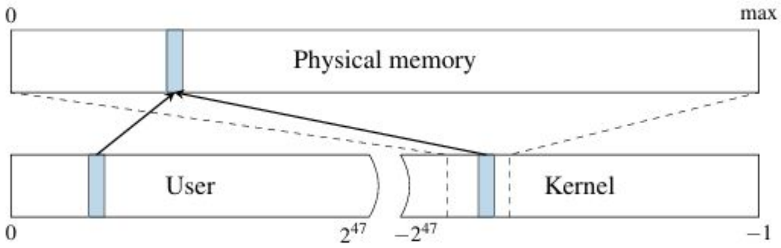
MIN: unimplementable, requires future knowledge of execution trace.

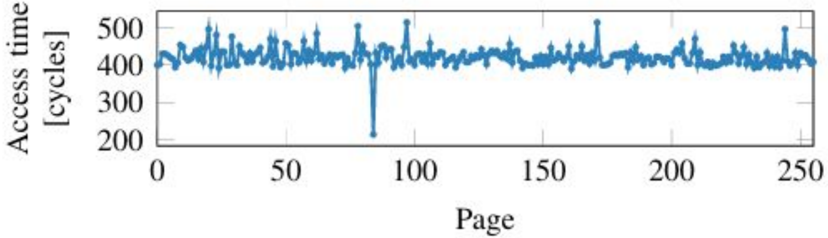
Caching optimizations

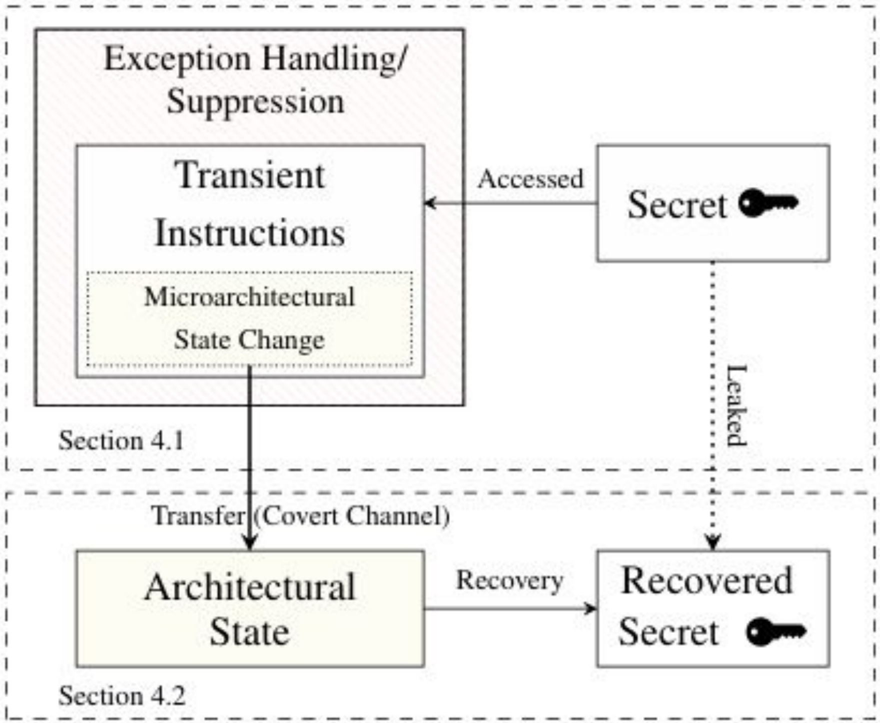
- Victim cache
  - (Usually) small, fully-associative cache that tracks evicted blocks.
- Stream buffer
  - A buffer that prefetches successive blocks from a lower level cache to a higher level cache upon a miss.
- Write buffer
  - Single-cycle operation to buffer a write request before it is issued to the cache.
  - Ordering challenges.
- Scratchpad
  - Software controlled memory with explicit, scratch-pad-private physical memory space.

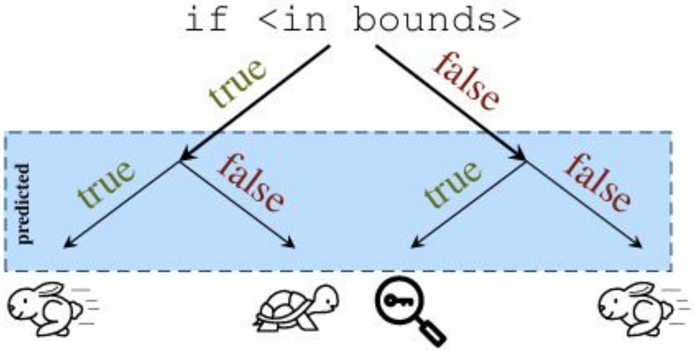
and – we’ll talk about these briefly


Frontend
L1 Instruction Cache
ITLB
Branch Predictor
Instruction Fetch & PreDecode
Instruction Queue
4-Way Decode
μOP Cache
μOPs
μOP
MUX
Allocation Queue
Execution Engine
CDB
Reorder buffer
Scheduler
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Memory Subsystem
Load Buffer
Store Buffer
L1 Data Cache
DTLB
STLB
L2 Cache

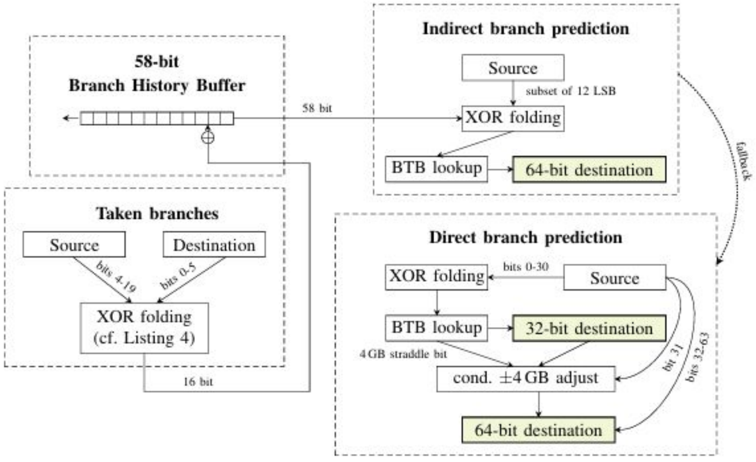

0
max
Physical memory
User
Kernel
0
2^{47}
-2^{47}
-1


Access time [cycles]
500
400
300
200
0
50
100
150
200
250
Page

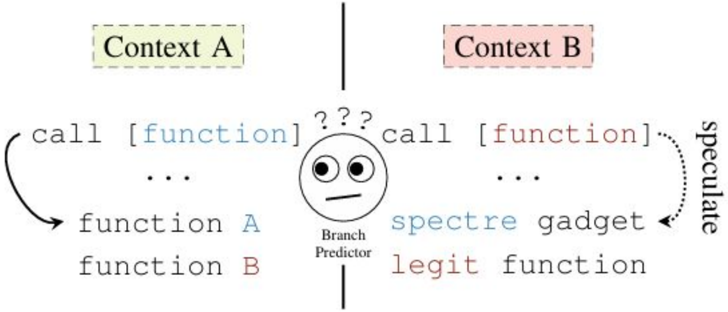

Exception Handling/ Suppression
Transient Instructions
Microarchitectural State Change
Accessed
Secret
Leaked
Section 4.1
Transfer (Covert Channel)
Architectural State
Recovery
Recovered Secret
Section 4.2



if <in bounds>
true
false
predicted
true
false
true
false



58-bit
Branch History Buffer
58 bit
Indirect branch prediction
Source
subset of 12 LSB
XOR folding
BTB lookup
64-bit destination
fallback
Taken branches
Source
Destination
bits 4-19
bits 0-5
XOR folding
(cf. Listing 4)
16 bit
Direct branch prediction
XOR folding
bits 0-30
Source
BTB lookup
32-bit destination
4 GB straddle bit
cond. ±4 GB adjust
bit 31
bits 32-63
64-bit destination



Context A
Context B
call [function]
...
function A
function B
? ? ?
Branch
Predictor
call [function]
...
spectre gadget
legit function
speculate

Homework 1 postmortem

– all of you did a pretty good job

Common questions

Q1.2
3 Points

Your friend proposes a
operations by 30%. Cal

Answer in nanosecond

Speedup w/ Example

"You friend proposes an optimization which would speed up ***all*** operations by 30%"

$$X = 1.3 = \frac{T_{\text{original}}}{T_{\text{optimized}}}$$