CS152 Discussion Section 3

Memory Hierarchy

Week of Feb 5
Spring 2024

Administrivia: Week 3

- **Lab1 due Friday Feb 9**
- **Problem Set 1 due Wednesday Feb 7**
- **Problem Set 2 will be out Wednesday Feb 7**

Feedback form! - tinyurl.com/152feedback

Agenda

- Review: Exceptions
- Review: More Complex Pipelines
- Review: Cache Organization
- Review: Different Eviction Policies
- Review: Cache Optimization
- Questions and Office Hours

Exceptions

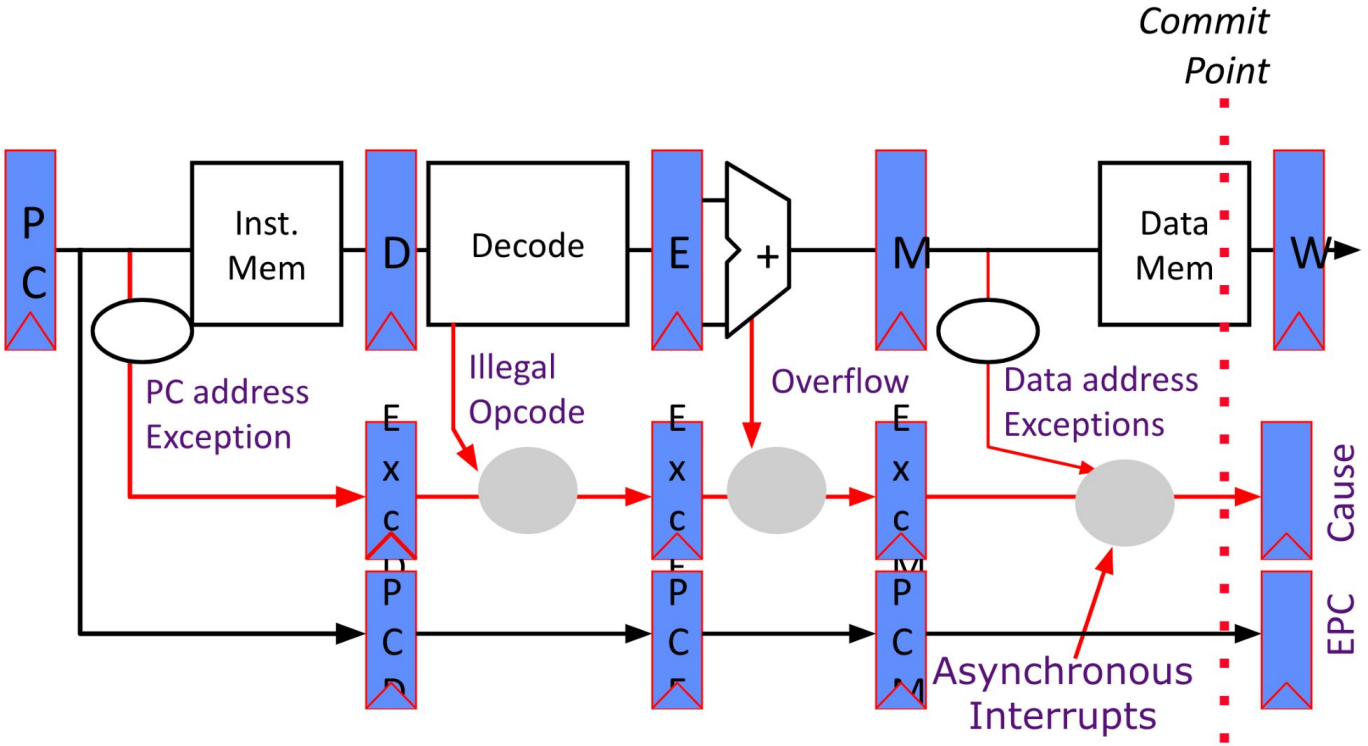
Precise Exception Definition:

> All instructions prior to the exception in program order have committed, and none of the instructions after (and including the faulting instruction) appear to have started.

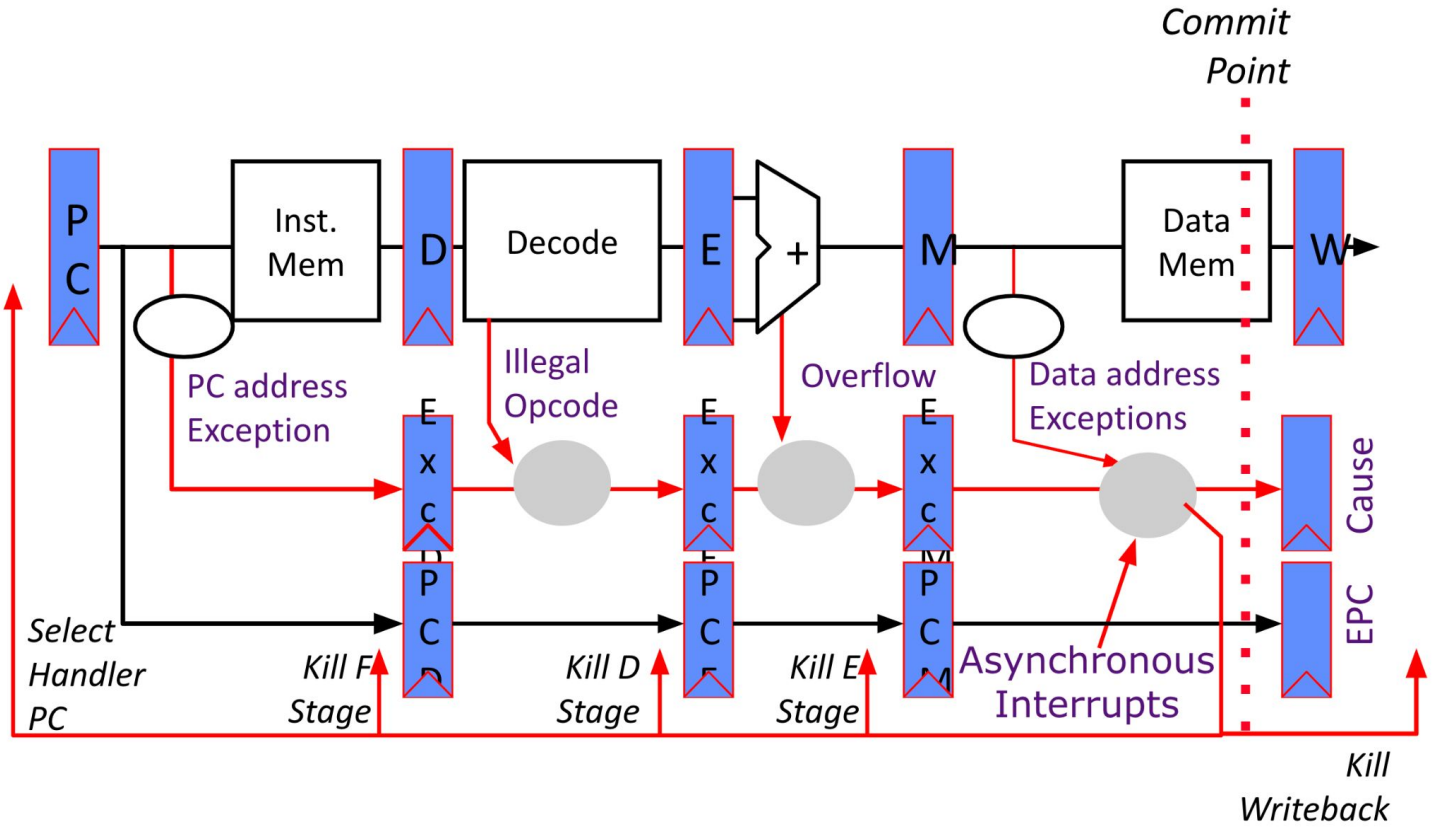
- Why are precise exceptions useful?
  - Architectural state matches programmer's model of sequential execution
  - Deterministic debugging
  - Clean restartability without exposing microarchitectural state

- Why might one not want to always implement precise exceptions?
  - Introduces hardware complexity
  - Specific application does not need to or cannot recover from fatal exceptions

Exceptions


Commit
Point
P
C
Inst.
Mem
D
Decode
E
+
M
Data
Mem
W
PC address
Exception
Illegal
Opcode
Overflow
Data address
Exceptions
E
x
c
P
C
Cause
EPC
Asynchronous
Interrupts

Exceptions


Commit
Point
P
C
Inst.
Mem
D
Decode
E
+
M
Data
Mem
W
PC address
Exception
Illegal
Opcode
Overflow
Data address
Exceptions
E
x
c
Cause
P
C
EPC
Select
Handler
PC
Kill F
Stage
Kill D
Stage
Kill E
Stage
Asynchronous
Interrupts
Kill
Writeback

Q1: Precise Exceptions

- Describe how program execution time may be affected by adding support for precise exceptions
  - Remember, *execution time = instructions/program * cycles/instruction * time/cycle*
- Compare and contrast precise exception handling with branch misprediction

Q1: Precise Exceptions

- Describe how program execution time may be affected by adding hardware support for precise exceptions
    - Remember, *execution time = instructions/program * cycles/instruction * time/cycle*

Instructions/Program: Decrease. Without hardware support for precise exceptions, may need more instructions to restore microarchitectural state

Cycles/Instruction: None (precise exception information carried in parallel)

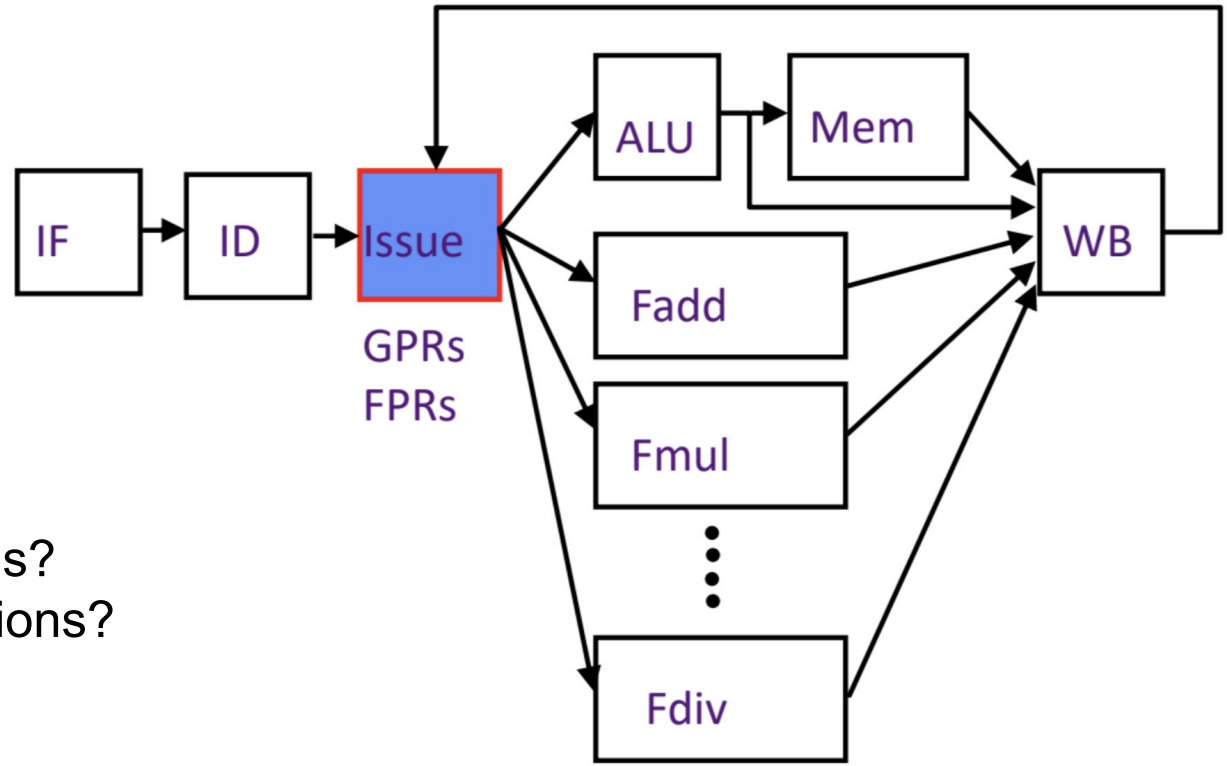
Time/Cycle: Increase (increased hardware complexity may be on critical path)

- Compare and contrast precise exception handling with branch misprediction

Exception: update Cause and EPC registers, flush pipeline, inject handler PC into Fetch stage

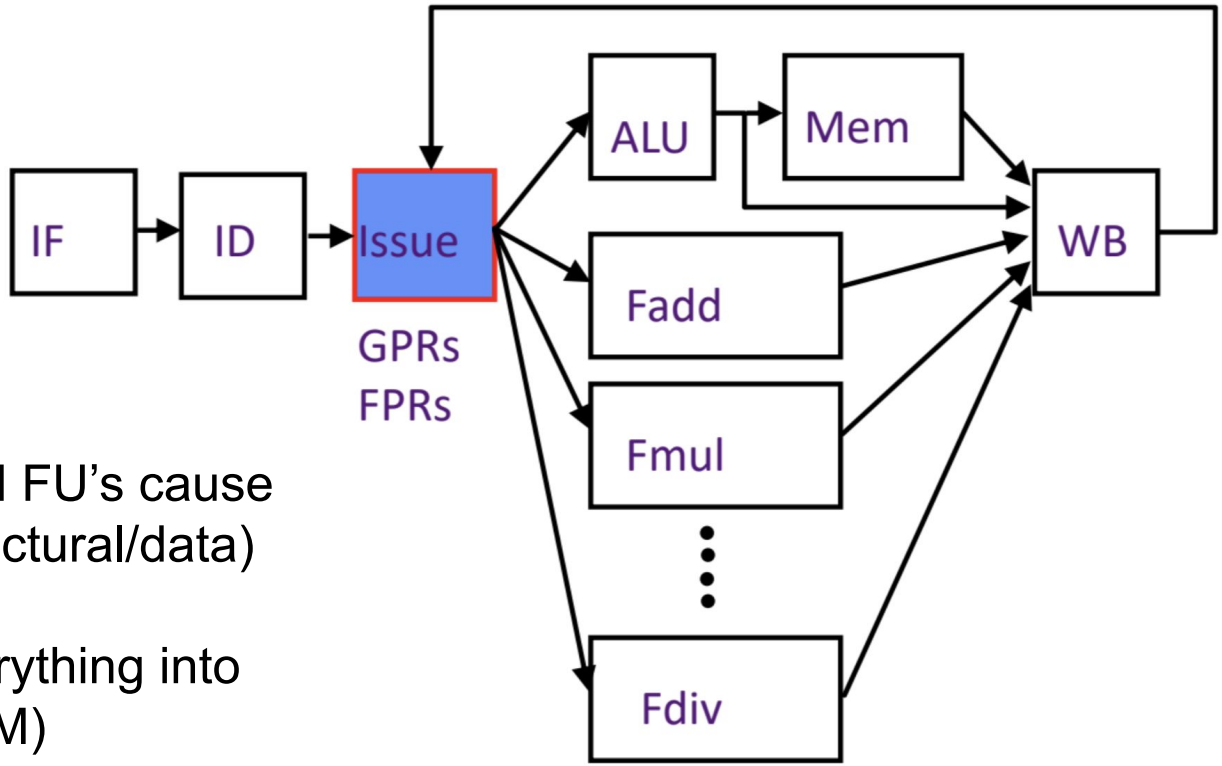
Branch mispredict: kill instructions in F and D phases and use calculated target PC

More Complex Pipelines



Hazards?
Exceptions?

More Complex Pipelines

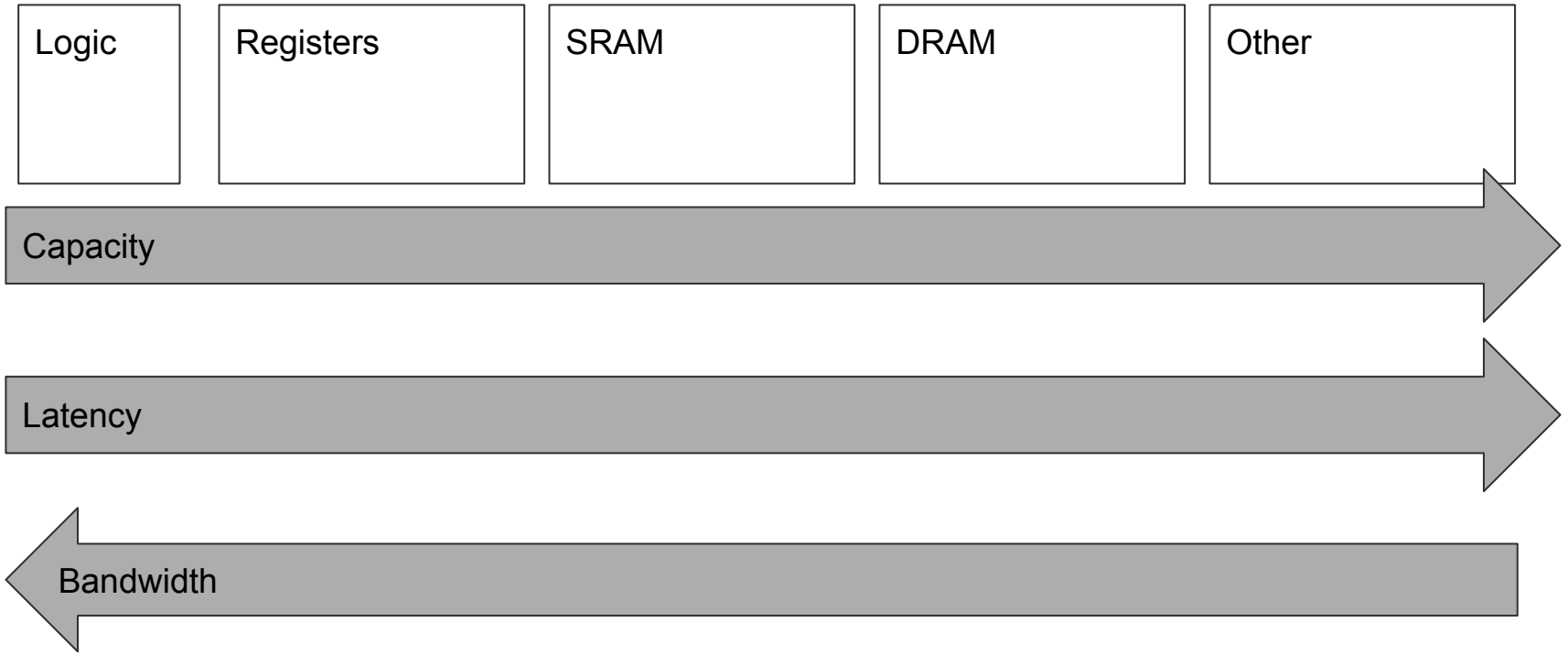


Hazards? Unpipelined FU's cause hazards in E/WB (structural/data)

Exceptions? Stall everything into commit point (End of M)

Memory Hierarchy

Where should instruction data go?
Where should program data go?



Why Caches?

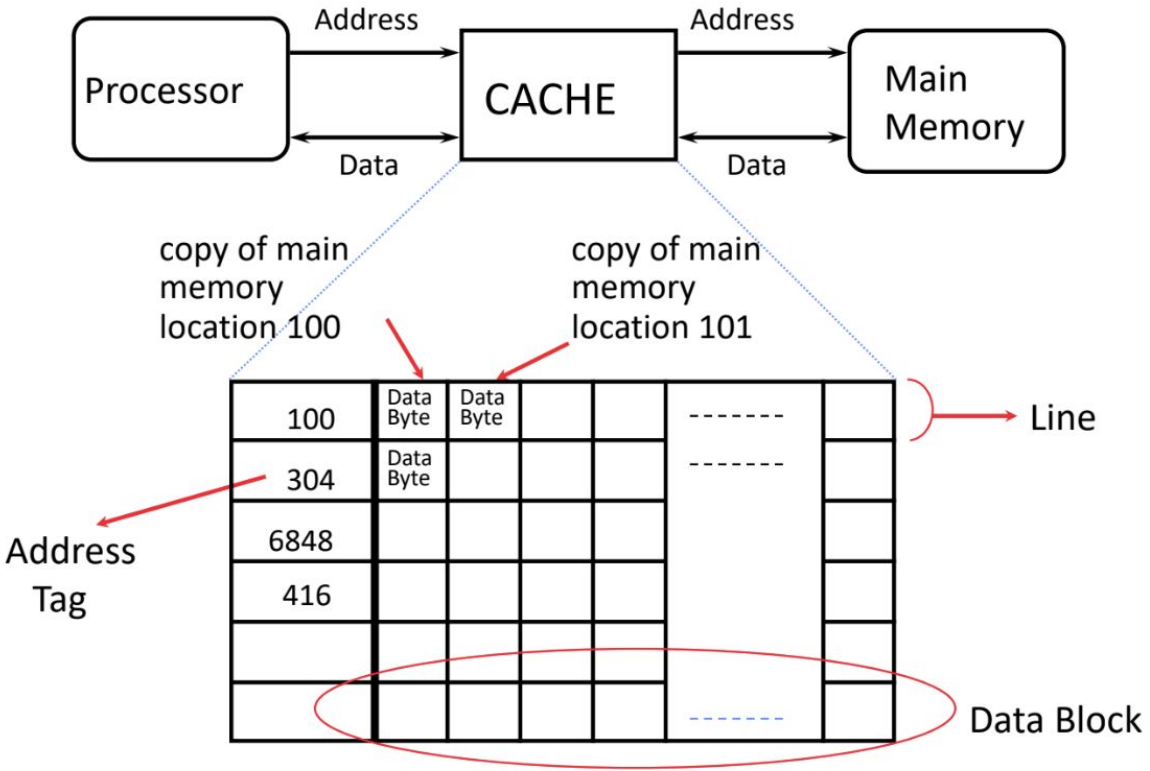
Why Caches?

- Temporal Locality
- Spatial Locality

Why Caches?

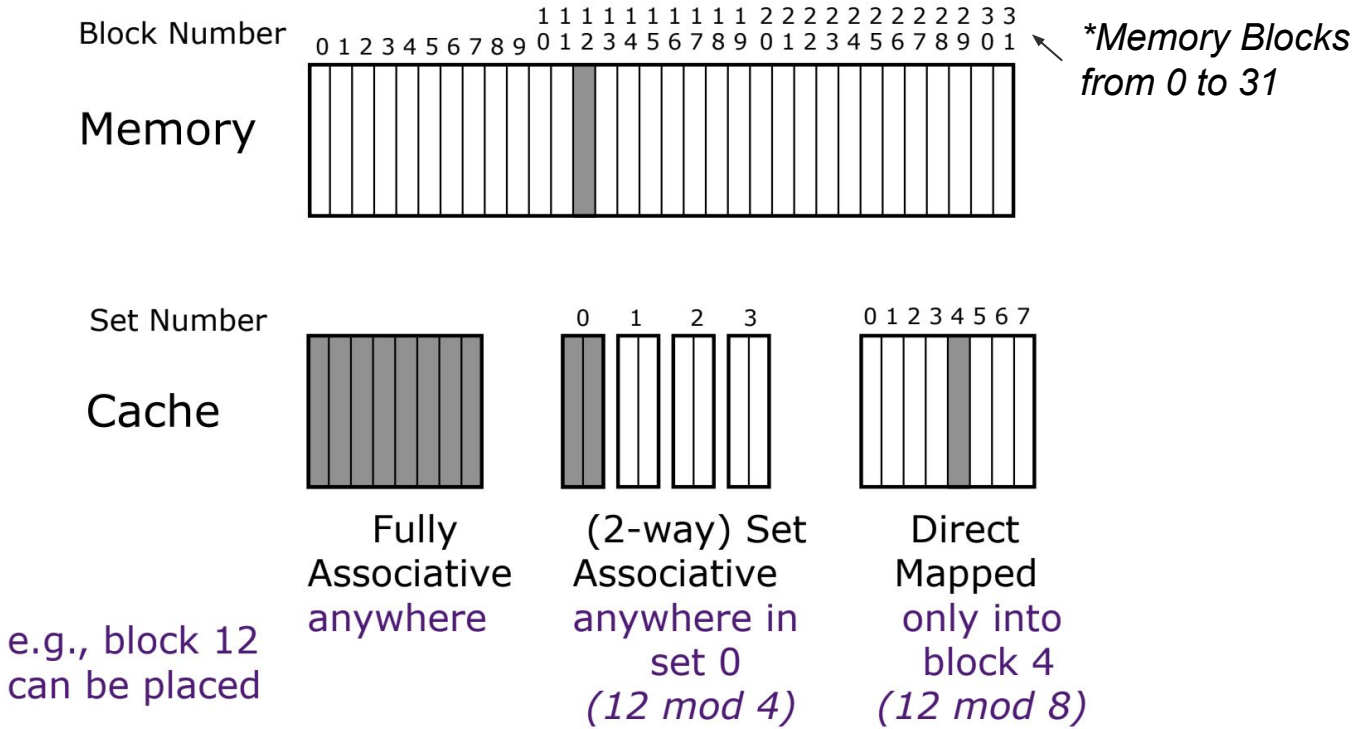
- Temporal Locality
  - Recently accessed locations are more likely to be accessed again
  - Examples:
    - Adding to counter
    - Matrix multiplication
- Spatial Locality
  - Locations near recently accessed locations are more likely to be accessed
  - Examples:
    - for(i=0; i<10; i++) {sum = sum + **array[i]**;}

Simple Cache



Placement Policy

Nice Illustration



Q2: Cache Organization

Consider a 1 KiB 4-way set-associative cache with 32-byte cache lines. The address is 12 bits wide. How are the address bits partitioned?

Address = Tag : Index : Offset

- Tag:
- Index:
- Offset:

Q2: Cache Organization

Consider a 1 KiB 4-way set-associative cache with 32-byte cache lines. The address is 12 bits wide. How are the address bits partitioned?

Address = Tag : Index : Offset

- Tag:
  Addr[11:8], 4 bits

- Index:
  Cache block = 32 bytes
  3 bits, addr[7:5]

- Offset:
  log2(32) bits = 5
  addr[4:0]

Q3: Replacement Policy

- Random
  - Why is this not as bad as it sounds?
- Least-Recently Used (LRU)
- Not-Most-Recently Used (NMRU)

Plus many more possibilities…

- Pseudo Least-Recently Used (PLRU)
- First-In First-Out (FIFO)

Replacement Policy

Suppose we see the following stream of accesses where A, B, C, D, and E represent unique addresses from different lines that all map to the same set:

A, B, C, D, B, A, E

Assume the cache has four ways and all lines in the set are initially invalid. When address E is accessed, one of the cache lines must be evicted. For each replacement policy, which line gets evicted?

- FIFO
- NMRU
- LRU
- PLRU

FIFO (First-In-First-Out)

State: next line to replace

On replacement, increment state

Assume state starts at 0

Access	Way0	Way1	Way2	Way3	State after Access
A	A				1
B		B			2
C			C		3
D				D	0
B		hit			0
A	hit				0
E	E				1

NMRU (Not Most Recently Used)

Almost same as FIFO

State: next line to replace

On replacement OR hit to line-to-be replaced, increment state

Assume state starts at 0

Access	Way0	Way1	Way2	Way3	State after Access
A	A				1
B		B			2
C			C		3
D				D	0
B		hit			0
A	hit				1
E		E			2

LRU (Least Recently Used)

State: list of least recently accessed ways

Initial state: [3, 2, 1, 0]

On hit, move hit way to back of state list

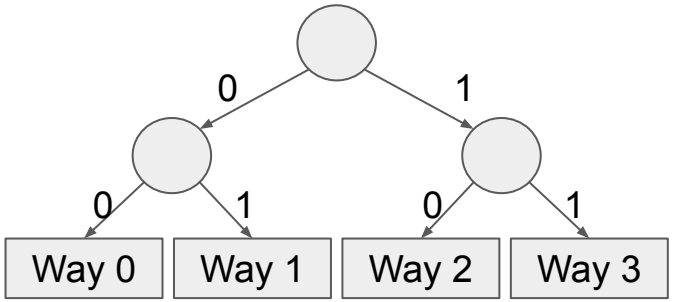
To replace, pop from front of state list

Access	Way0	Way1	Way2	Way3	State after Access
A	A				[0,3,2,1]
B		B			[1,0,3,2]
C			C		[2,1,0,3]
D				D	[3,2,1,0]
B		hit			[1,3,2,0]
A	hit				[0,1,3,2]
E			E		

Pseudo-LRU

Represent tree as 0(0)(0)

Initial state is 0(0)(0)



Access	Way0	Way1	Way2	Way3	State after Access
A	A				1(1)(0)
B			B		0(1)(1)
C		C			1(0)(1)
D				D	0(0)(0)
B			hit		0(0)(1)
A	hit				1(1)(1)
E				E	0(1)(0)

4-way Pseudo LRU

time

	Cache Tags (hex)										
Address (hex)	**Set 0**					**Set 1**					**Hit?**
	Way 0	**Way 1**	**Way 2**	**Way 3**	**PLRU State**	**Way 0**	**Way 1**	**Way 2**	**Way 3**	**PLRU State**	
208	2	inv	inv	inv	110	inv	inv	inv	inv	000	no
22C						2				110	no
41C			4		011						no
604		6			101						no
320								3		011	no
214	hit				111						yes
310				3	010						no
50C		5			100						
404			hit		001						yes

Causes of Cache Misses

- Compulsory
- Capacity
- Conflict

Causes of Cache Misses

- Compulsory
  - First reference to a line. Will always happen
  - To improve: increase line size (but increased conflict misses and miss penalty)
- Capacity
  - Cache is too small, so requested line was evicted
  - To improve: increase cache size (but increased hit time)
- Conflict
  - Occur due to placement policy (would not occur in a fully associative cache)
  - To improve: increase cache size and increase associativity (but increased hit time)

Write Policies

- Cache Hit
- Cache Miss

Write Policies

- Cache Hit
  - Write-through - write to $ + memory
  - Write-back - write to $ only, evictions spawn writes
- Cache Miss
  - No-write-allocate - write to memory
  - Write-allocate - fetch into $ then write to memory

- Common combinations
  - Write-through + no-write-allocate
  - Write-back + write-allocate

Write Policies

- Cache Hit
  - Write-through - write to $ + memory
  - Write-back - write to $ only, evictions spawn writes
- Cache Miss
  - No-write-allocate - write to memory
  - Write-allocate - fetch into $ then write to memory

- Common combinations
  - Write-through + no-write-allocate
  - Write-back + write-allocate

Why do we care about the optimizations?

Any issues with multiple CPU's accessing memory?

Q4: Cache Optimizations

Average memory access time (AMAT) = Hit time + Miss rate x Miss penalty

Increase / Decrease / No Change

Technique	Hit Time	Miss Penalty	Miss Rate	Hardware Complexity
Smaller, simpler caches				
Multi-level caches				
Smart replacement policy				
Pipelined writes				
Write buffer				
Sub-blocks (sector cache)				

Cache Optimizations

Increase / Decrease / No Change

Technique	Hit Time	Miss Penalty	Miss Rate	Hardware Complexity
Smaller, simpler caches	Decrease	No change	Increase	Decrease
Multi-level caches	No change	Decrease	Decrease	Increase
Smart replacement policy	May increase	No change	Decrease	Increase
Pipelined writes	Decrease	No change	No change	Increase
Write buffer	No change	Decrease	No change	Increase
Sub-blocks (sector cache)	No change	Decrease	Increase	Increase

Cache Optimizations

Increase / Decrease / No Change

Technique	Hit Time	Miss Penalty	Miss Rate	Hardware Complexity
Code optimization				
Compiler prefetching				
Hardware prefetching (stream buffer)				

Cache Optimizations

Increase / Decrease / No Change

Technique	Hit Time	Miss Penalty	Miss Rate	Hardware Complexity
Code optimization	No change	No change	Decrease	No change
Compiler prefetching	No change	No change	Decrease	No change
Hardware prefetching (stream buffer)	No change	Increase	Decrease	Increase