RISC Design

Memory System

Virendra Singh

Professor

Computer Architecture and Dependable Systems Lab

Department of Electrical Engineering

Indian Institute of Technology Bombay

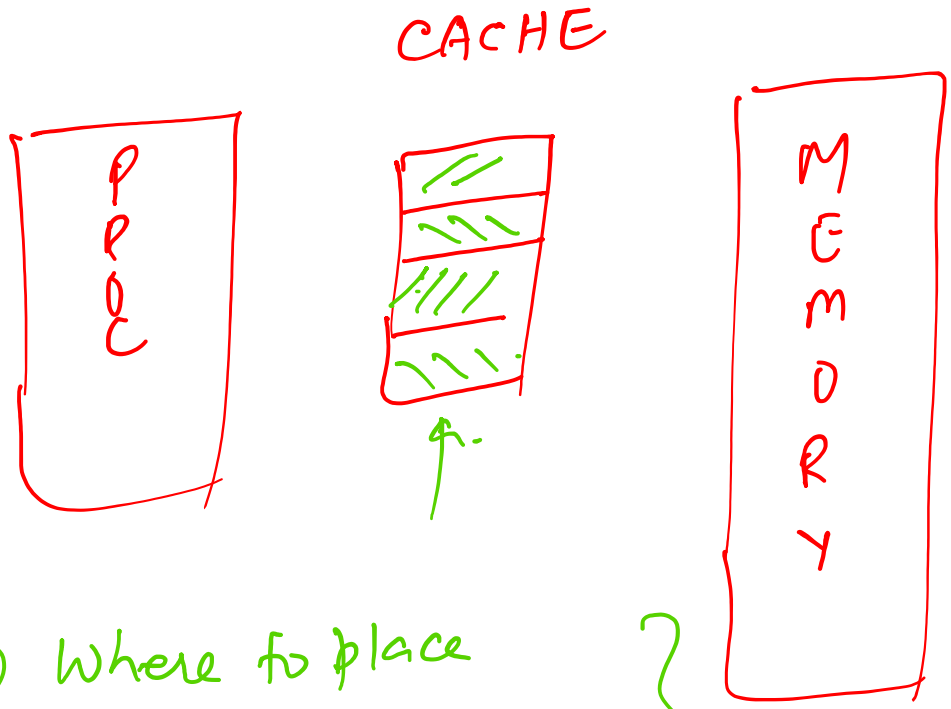
http://www.ee.iitb.ac.in/~viren/

E-mail: viren@ee.iitb.ac.in

EE-739: Processor Design

Lecture 27 (31 March 2021)

CADSL

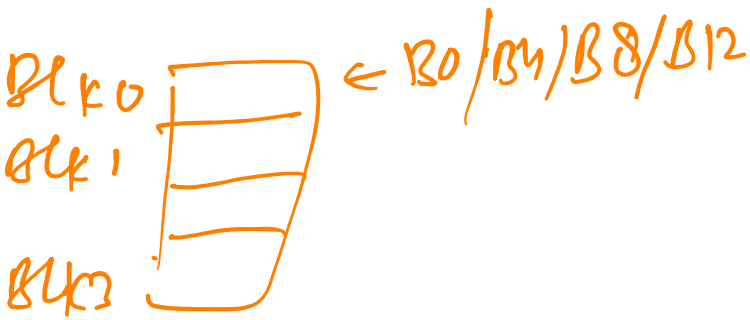


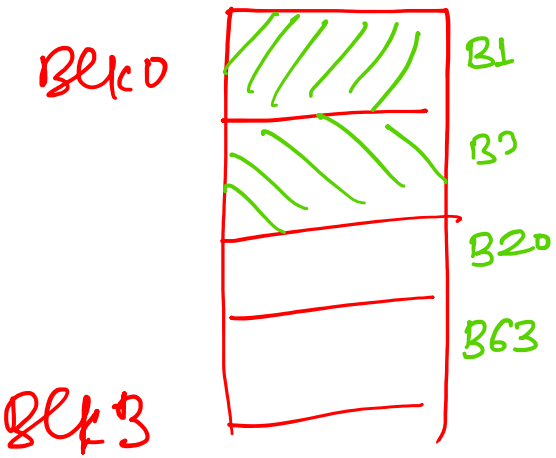
(1) Where to place
(2) How to identify
(3) Whom to evict
(4) When to write

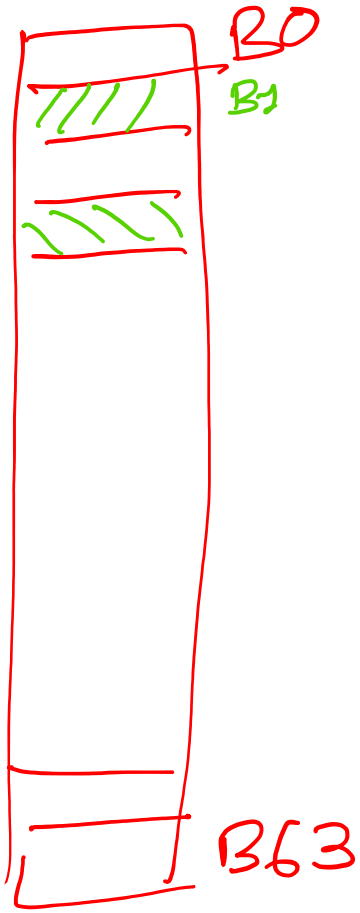
Whom to evict

Placement ← Direct mapped
Fully Associative
Set Associative.

DM – mapping is fixed – only evict the DM block







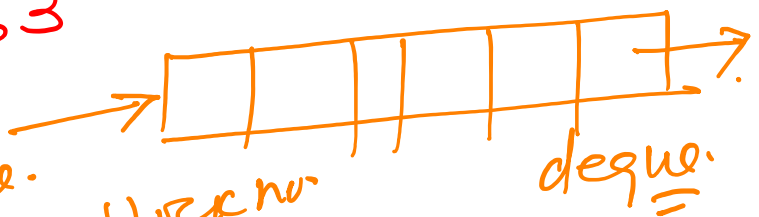
B62 → where to place

Replace a block which is not likely to be used in future

POLICY

① Random.

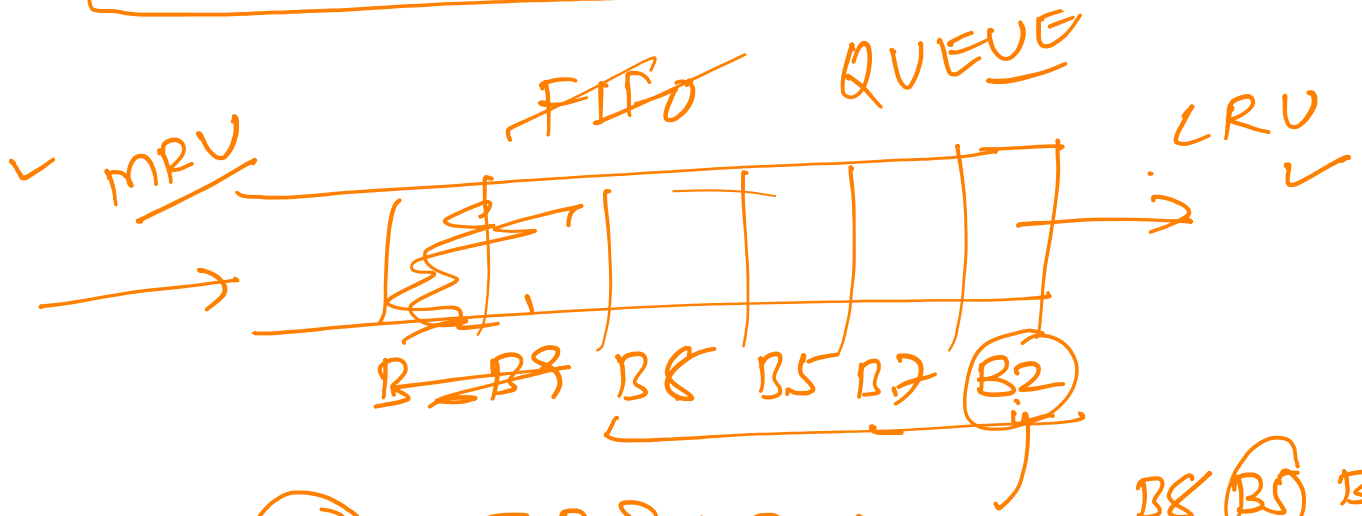
② FIFO -



(insert the block no. when it is fetched)

B62 B63 B2 B20 ← B63 B2 B20 B1 →

(3) Least Recently Use (OR LRU)



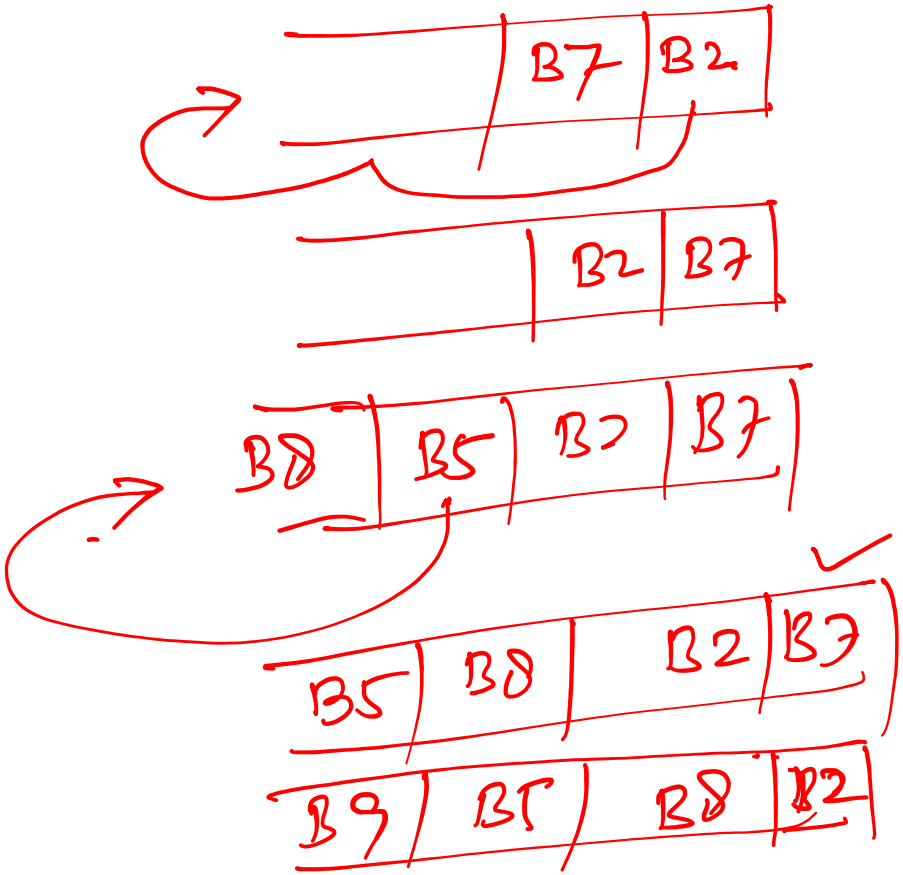
B2 B7 B2 B5 B8 B5 ⇒

B8 B5 B2 B7

B5 B8 B2 B7

Approximation of LRU
FIFO but not MRU

B2 B7 B2 B5 B8 B6 B9 LRU



LRU

2 block.

Set Associative memory

2 way SA memory ✓

↳ bit ✓

Replacement

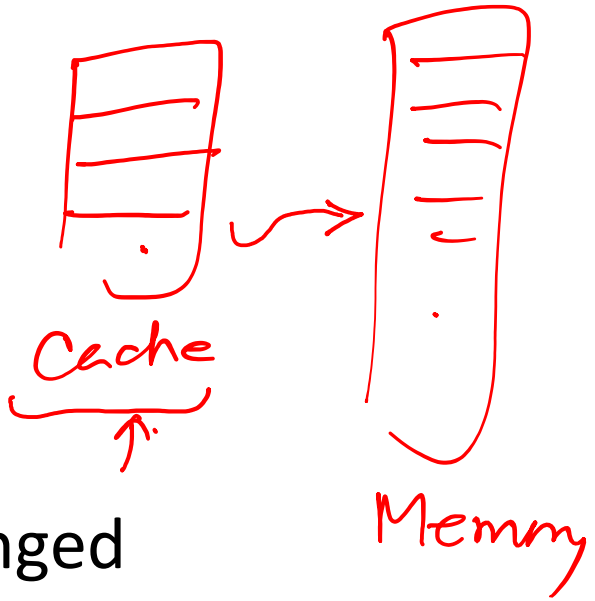
- Cache has finite size
  - What do we do when it is full?
- Analogy: desktop full?
  - Move books to bookshelf to make room
- Same idea:
  - Move blocks to next level of cache

Replacement

- How do we choose *victim*?
  - Verbs: *Victimize, evict, replace, cast out*
- Several policies are possible
  - FIFO (first-in-first-out)
  - LRU (least recently used)
  - NMRU (not most recently used)
  - Pseudo-random (yes, really!)
- Pick victim within *set* where a = *associativity*
  - If a <= 2, LRU is cheap and easy (1 bit)
  - If a > 2, it gets harder
  - Pseudo-random works pretty well for caches

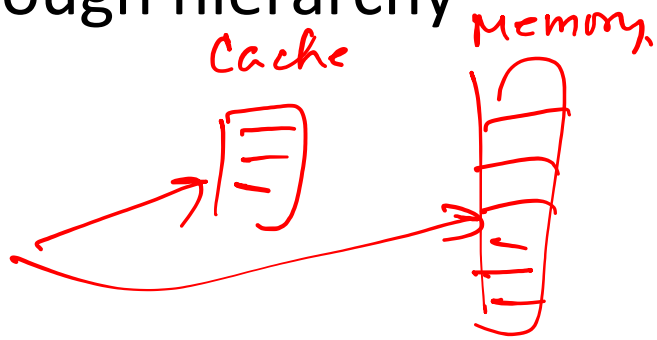
Write Policy

- Memory hierarchy
  - 2 or more copies of same block
    - Main memory and/or disk
    - Caches
- What to do on a write?
  - Eventually, all copies must be changed
  - Write must *propagate* to all levels



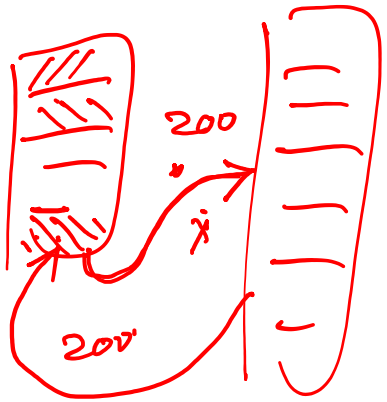
Write Policy

- Easiest policy: *write-through*
- Every write propagates directly through hierarchy
  - Write in L1, L2, memory, disk (?!?)
- Why is this a bad idea?
  - Very high bandwidth requirement
  - Remember, large memories are slow
- Popular in real systems only to the L2
  - Every write updates L1 and L2
  - Beyond L2, use *write-back* policy



Write Policy

- Most widely used: *write-back*
- Maintain *state* of each line in a cache
  - Invalid – not present in the cache
  - Clean – present, but not written (unmodified)
  - Dirty – present and written (modified)
- Store state in tag array, next to address tag
  - Mark dirty bit on a write
- On eviction, check dirty bit
  - If set, write back dirty line to next level
  - Called a *writeback* or *castout*



flag
modified / not modified
dirty
clean

Write Policy

- Complications of write-back policy
  - Stale copies lower in the hierarchy
  - Must always check higher level for dirty copies before accessing copy in a lower level
- Not a big problem in uniprocessors
  - In multiprocessors: *the cache coherence problem*
- I/O devices that use DMA (direct memory access) can cause problems even in uniprocessors
  - Called coherent I/O
  - Must check caches for dirty copies before reading main memory

Cache Example

- 32B Cache: <BS=4,S=4,B=8>
  - o=2, i=2, t=2; 2-way set-associative
  - Initially empty
  - Only tag array shown on right
- Trace execution of:

Reference	Binary	Set/Way	Hit/Miss

Tag Array

Tag0	Tag1	LRU
		0
		0
		0
		0

Cache Example

- 32B Cache: <BS=4,S=4,B=8>
  - o=2, i=2, t=2; 2-way set-associative
  - Initially empty
  - Only tag array shown on right
- Trace execution of:

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss

Tag Array

Tag0	Tag1	LRU
		0
		0
10		1
		0

Cache Example

- 32B Cache: <BS=4,S=4,B=8>
  - o=2, i=2, t=2; 2-way set-associative
  - Initially empty
  - Only tag array shown on right
- Trace execution of:

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss
Load 0x2B	101011	2/0	Hit

Tag Array

Tag0	Tag1	LRU
		0
		0
10		1
		0

Cache Example

- 32B Cache: <BS=4,S=4,B=8>
  - o=2, i=2, t=2; 2-way set-associative
  - Initially empty
  - Only tag array shown on right
- Trace execution of:

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss
Load 0x2B	101011	2/0	Hit
Load 0x3C	111100	3/0	Miss

Tag Array

Tag0	Tag1	LRU
		0
		0
10		1
11		1

Cache Example

- 32B Cache: <BS=4,S=4,B=8>
  - o=2, i=2, t=2; 2-way set-associative
  - Initially empty
  - Only tag array shown on right
- Trace execution of:

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss
Load 0x2B	101011	2/0	Hit
Load 0x3C	111100	3/0	Miss
Load 0x20	100000	0/0	Miss

Tag Array

Tag0	Tag1	LRU
10		1
		0
10		1
11		1

Cache Example

- 32B Cache: <BS=4,S=4,B=8>
  - o=2, i=2, t=2; 2-way set-associative
  - Initially empty
  - Only tag array shown on right
- Trace execution of:

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss
Load 0x2B	101011	2/0	Hit
Load 0x3C	111100	3/0	Miss
Load 0x20	100000	0/0	Miss
Load 0x33	110011	0/1	Miss

Tag Array

Tag0	Tag1	LRU
10	11	0
		0
10		1
11		1

Cache Example

- 32B Cache: <BS=4,S=4,B=8>
  - o=2, i=2, t=2; 2-way set-associative
  - Initially empty
  - Only tag array shown on right
- Trace execution of:

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss
Load 0x2B	101011	2/0	Hit
Load 0x3C	111100	3/0	Miss
Load 0x20	100000	0/0	Miss
Load 0x33	110011	0/1	Miss
Load 0x11	010001	0/0 (lru)	Miss/Evict

Tag Array

Tag0	Tag1	LRU
01	11	1
		0
10		1
11		1

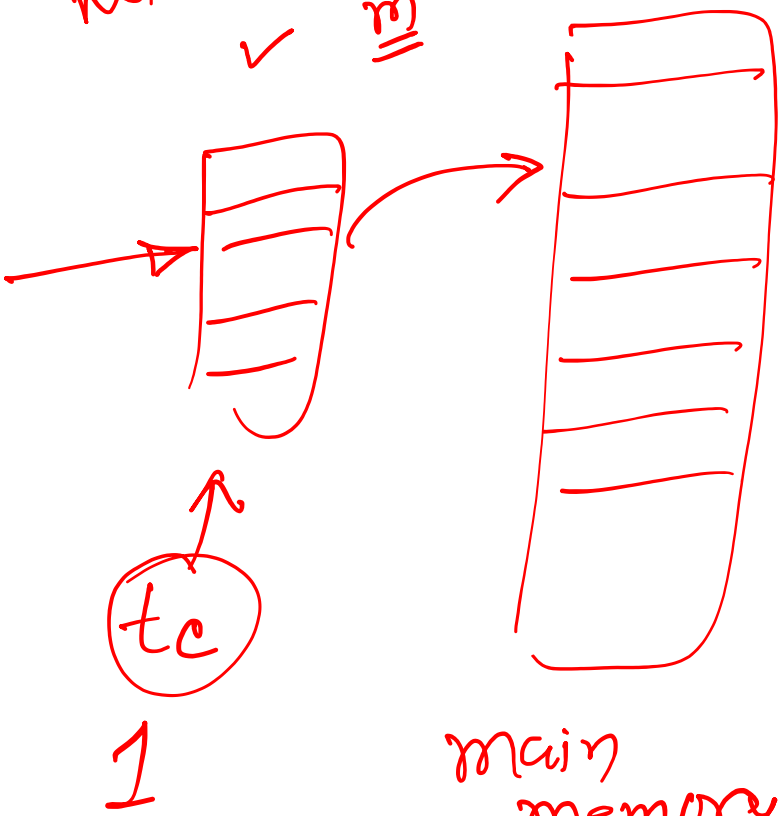
Cache Example

- 32B Cache: <BS=4,S=4,B=8>
  - o=2, i=2, t=2; 2-way set-associative
  - Initially empty
  - Only tag array shown on right
- Trace execution of:

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss
Load 0x2B	101011	2/0	Hit
Load 0x3C	111100	3/0	Miss
Load 0x20	100000	0/0	Miss
Load 0x33	110011	0/1	Miss
Load 0x11	010001	0/0 (lru)	Miss/Evict
Store 0x29	101001	2/0	Hit/Dirty

Tag Array

Tag0	Tag1	LRU
01	11	1
		0
10 d		1
11		1



$$tav = tc + m \cdot tm.$$

$tav = 1 + .1 \times 200$

$= 1 + .1 \times 200 = 1 + 20$

$= 21$

$200 \longrightarrow 21$

$h = 99\%.$ $m = 1\%.$

100

$tav = 1 + .01 \times 200$

$= 1 + 2 = 3$

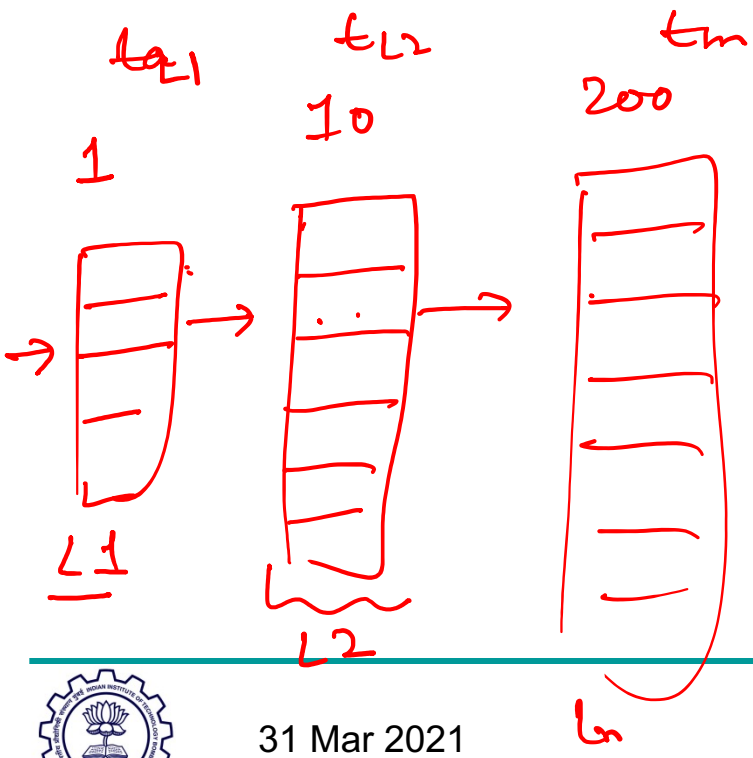
10%

$h = 90\%.$ $m = 10\%.$

Memory Optimization

$$\frac{\text{Misses}}{\text{Instruction}} = \frac{\text{Miss rate} \times \text{Memory accesses}}{\text{Instruction count}} = \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}}$$

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



reduce

reduce miss penalty

$t_{av} = t_{L1} + m_{L1} \cdot t_{L2} + m_{L1} \cdot m_{L2} \cdot t_m$

$m_{L1} = m_{L2} = 10\%$

$= 1 + .1 \times 10 + .1 \times .1 \times 200$

$= 1 + 1 + 2 = 4$

Thank You