19. Caches: Direct Mapped

EECS 370 – Introduction to Computer Organization – Fall 2020

Satish Narayanasamy

EECS Department
University of Michigan in Ann Arbor, USA

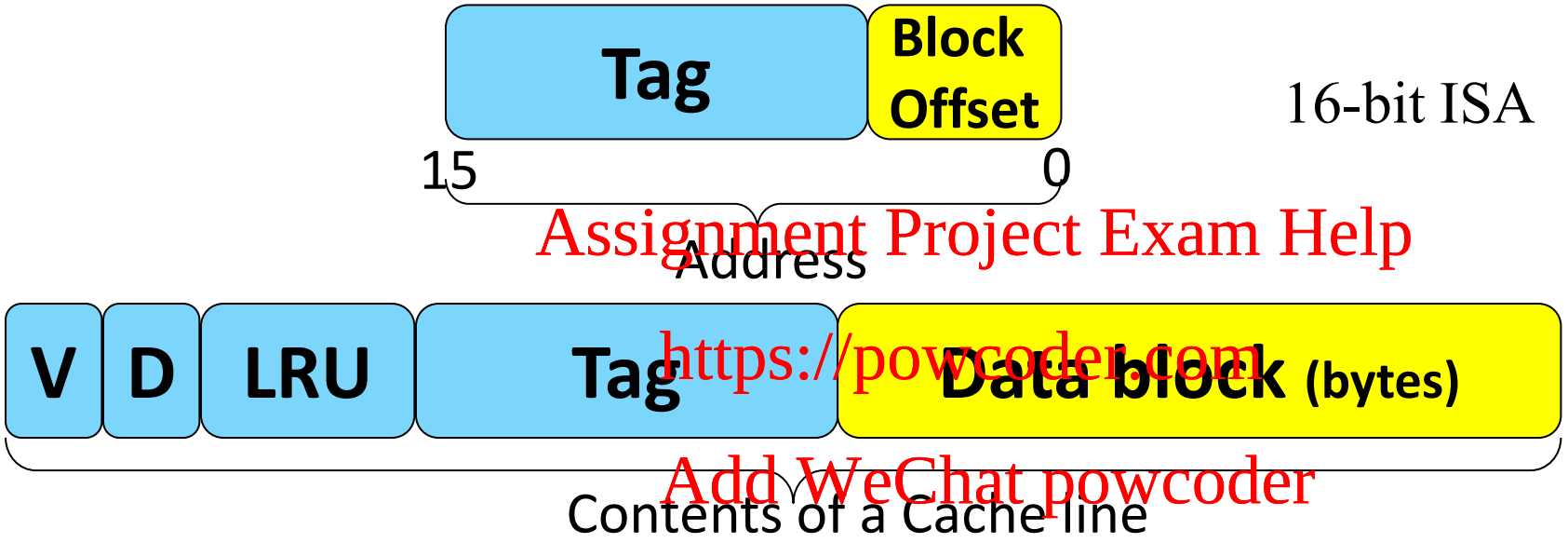
Announcements

Upcoming deadlines:

HW4	due Nov 10th
Project 3	due Nov. 12th

Recap: Cache Blocks and Write policy

Review: Cache Organization



Cache blocks:

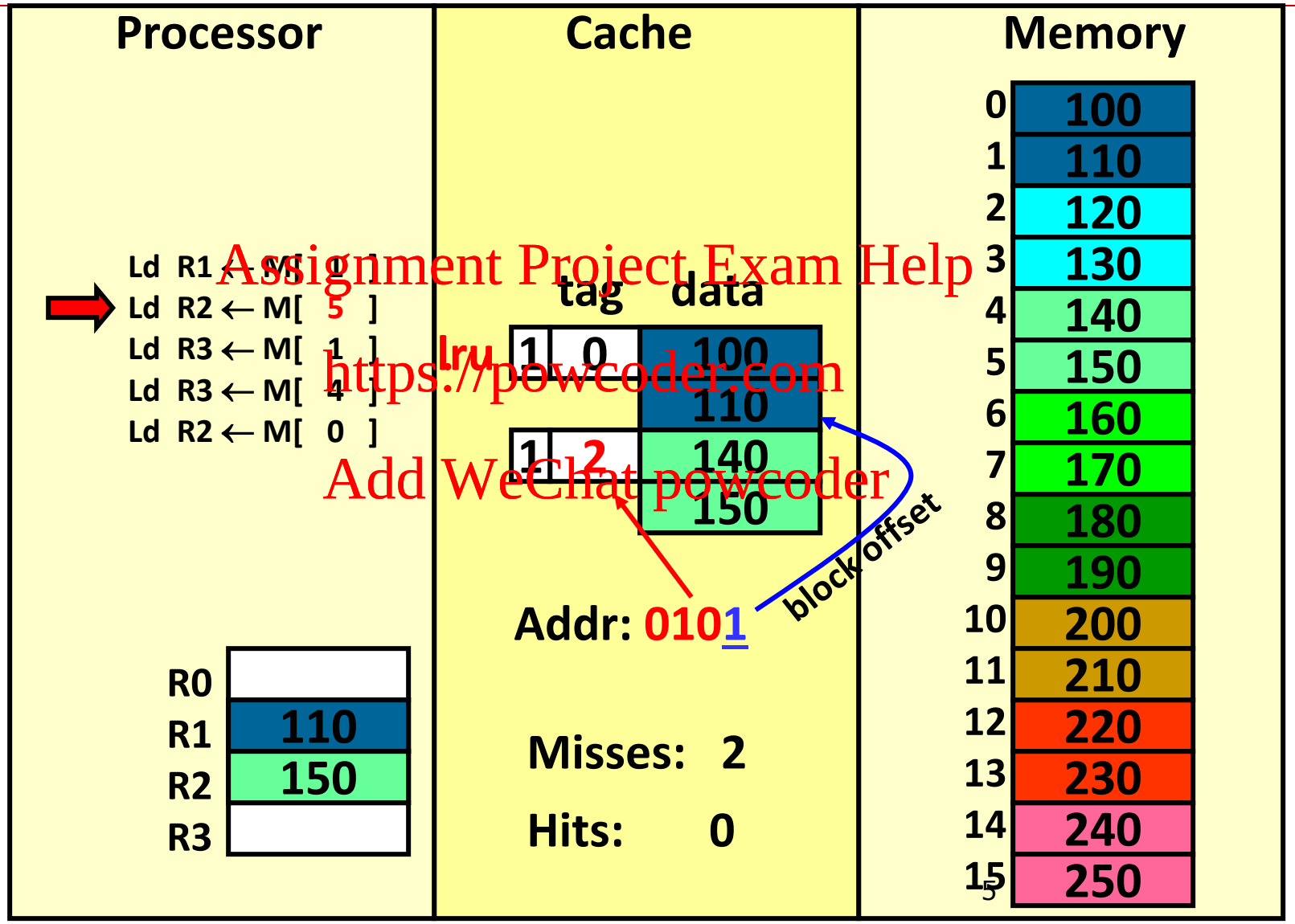
Captures spatial locality (increase cache hit rate)

Reduces tag overhead (number and size of tags)

Need not store block offset in the cache line

Determine byte to be read/written from the address directly

Review: How to find tag from address?



Review: Writes

Write-allocate vs. no-write-allocate caches

Policy that decides what to do with a cache-miss on a store instruction.

Write-allocate: First bring data from memory into the cache, then write

No-write-allocate: do not bring data in the cache, just write directly to the memory, not to the cache

Review: Writes

Write-through vs. write-back caches

Policy that decides when to **write to cache vs. memory vs. both**

Write-through: write to both cache and memory

Write-back: write only to cache, keep track of dirty cache line, write to memory when dirty cache line is evicted

Review: Writes

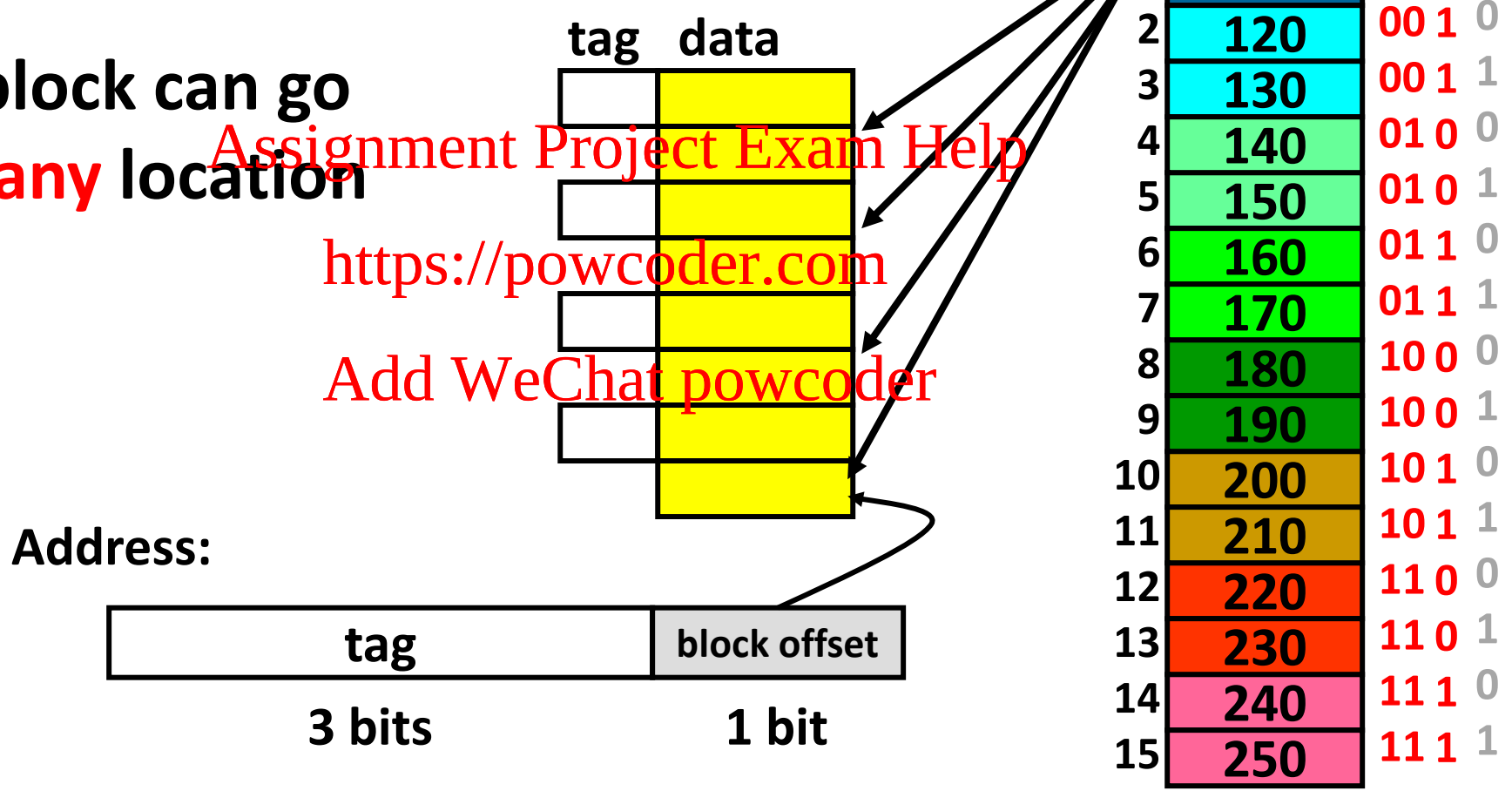
Store w No Allocate	Write-Back	Write-Through
Hit?	Write Cache	Write to Cache + Memory
Miss?	Write to Memory	Write to Memory
Replace block?	If evicted block is dirty, write to Memory	Do Nothing

Store w Allocate	Write-Back	Write-Through
Hit?	Write Cache	Write to Cache + Memory
Miss?	Read from Memory to Cache, Allocate to LRU block Write to Cache	Read from Memory to Cache, Allocate to LRU block Write to Cache + Memory
Replace block?	If evicted block is dirty, write to Memory	Do Nothing

Direct Mapped Caches

Fully-associative caches

A block can go to **any** location



Fully-associative caches

We designed a fully-associative cache

- A memory location can be copied to any cache line.
- We check every cache tag to determine whether the data is in the cache.

This approach can be too slow sometimes

- Parallel tag searches are expensive and can be slow. Why?

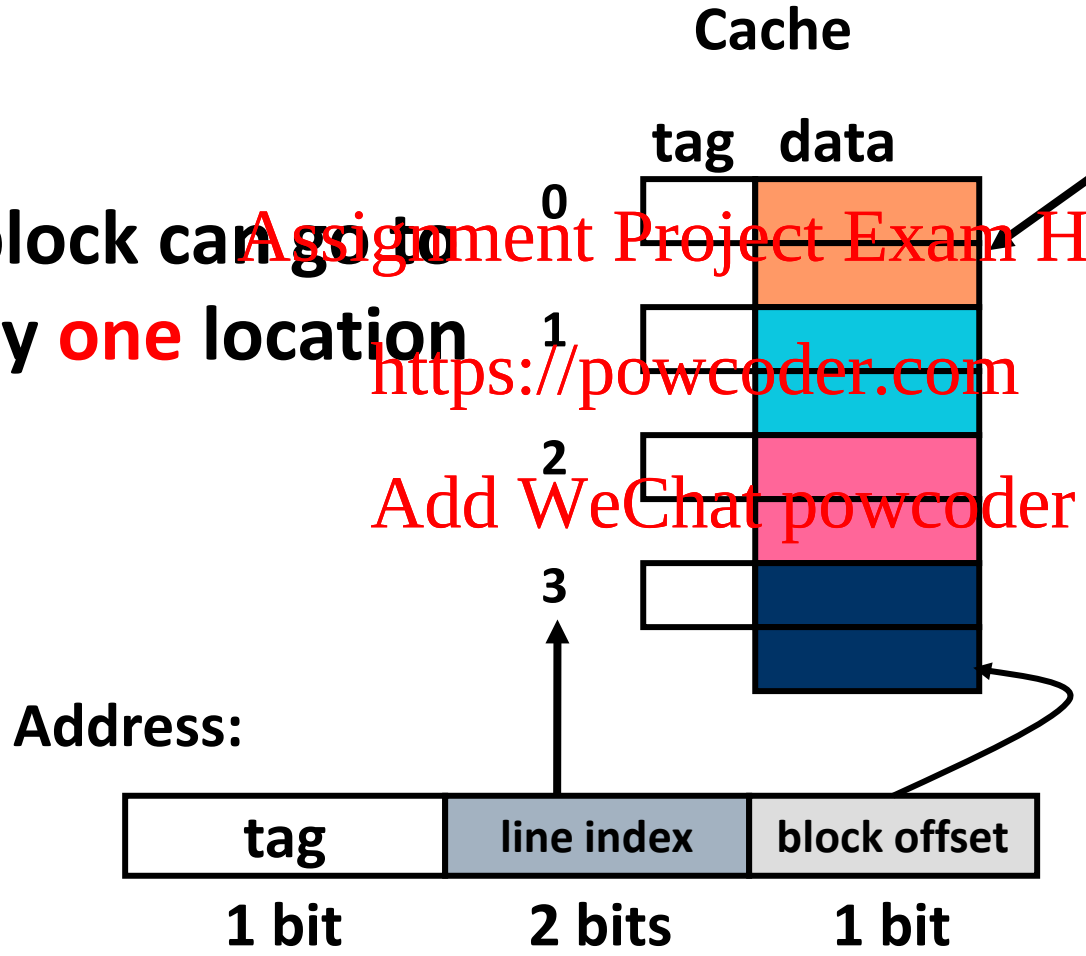
Direct mapped caches

We can redesign the cache to eliminate the requirement for parallel tag lookups

- Direct mapped caches partition memory into as many regions as there are cache lines
- Each memory region maps to a single cache line in which data can be placed
- You then only need to check a single tag – the one associated with the region the reference is located in

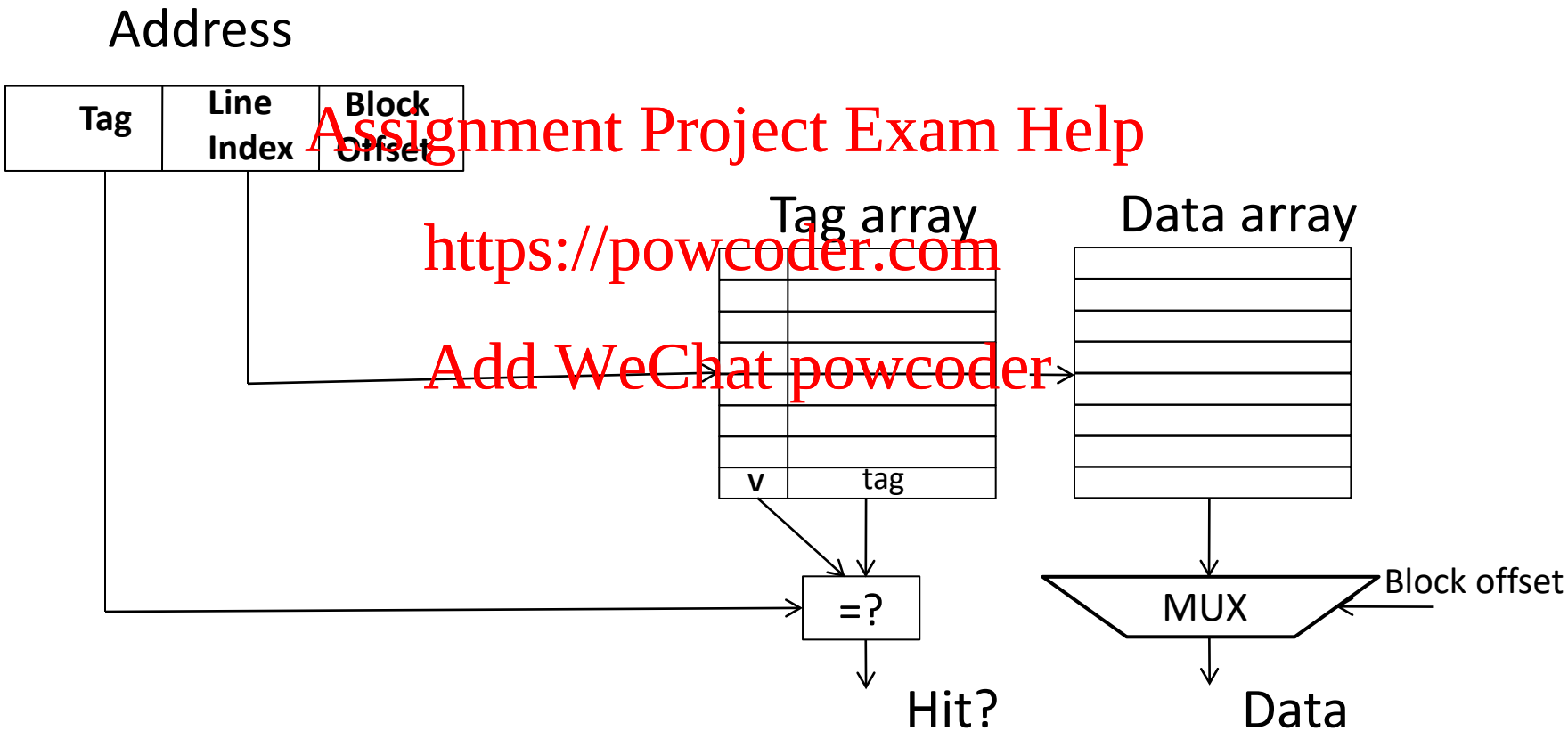
Mapping memory to cache (Direct-mapped)

A block can go to only **one** location



	Memory	Tag	Index	Block_offset
0	78	0	00	0
1	29	0	00	1
2	120	0	01	0
3	123	0	01	1
4	71	0	10	0
5	150	0	10	1
6	162	0	11	0
7	173	0	11	1
8	18	1	00	0
9	21	1	00	1
10	33	1	01	0
11	28	1	01	1
12	19	1	10	0
13	200	1	10	1
14	210	1	11	0
15	225	1	11	1

Direct-mapped cache: Placement & Access



Direct mapped caches

Two blocks in memory that map to the same cache index cannot be present in the cache at the same time (conflict)

One index → one entry

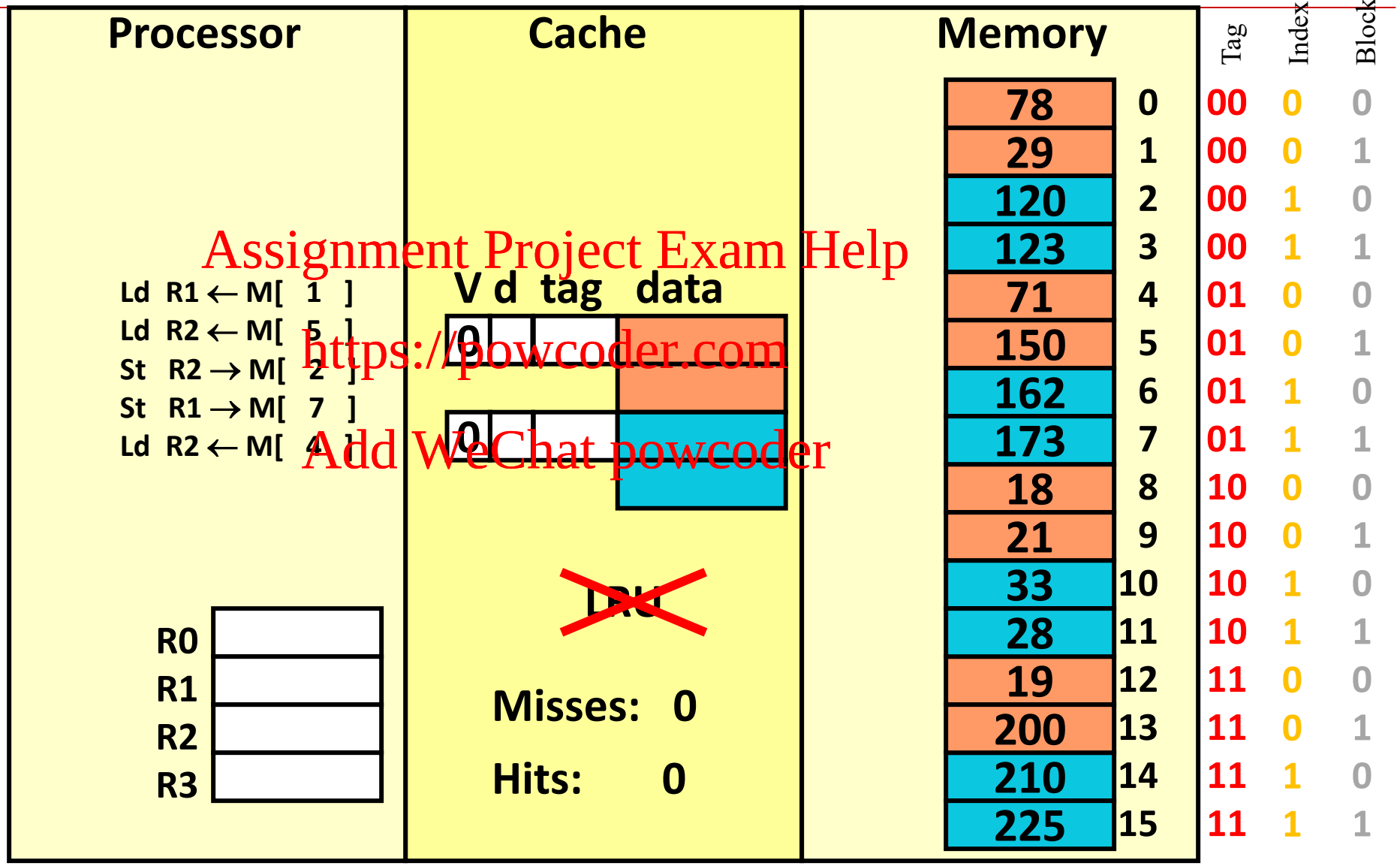
Can lead to 0% hit rate if more than one block accessed in an interleaved manner map to the same index

Assume addresses A and B have the same index bits but different tag bits

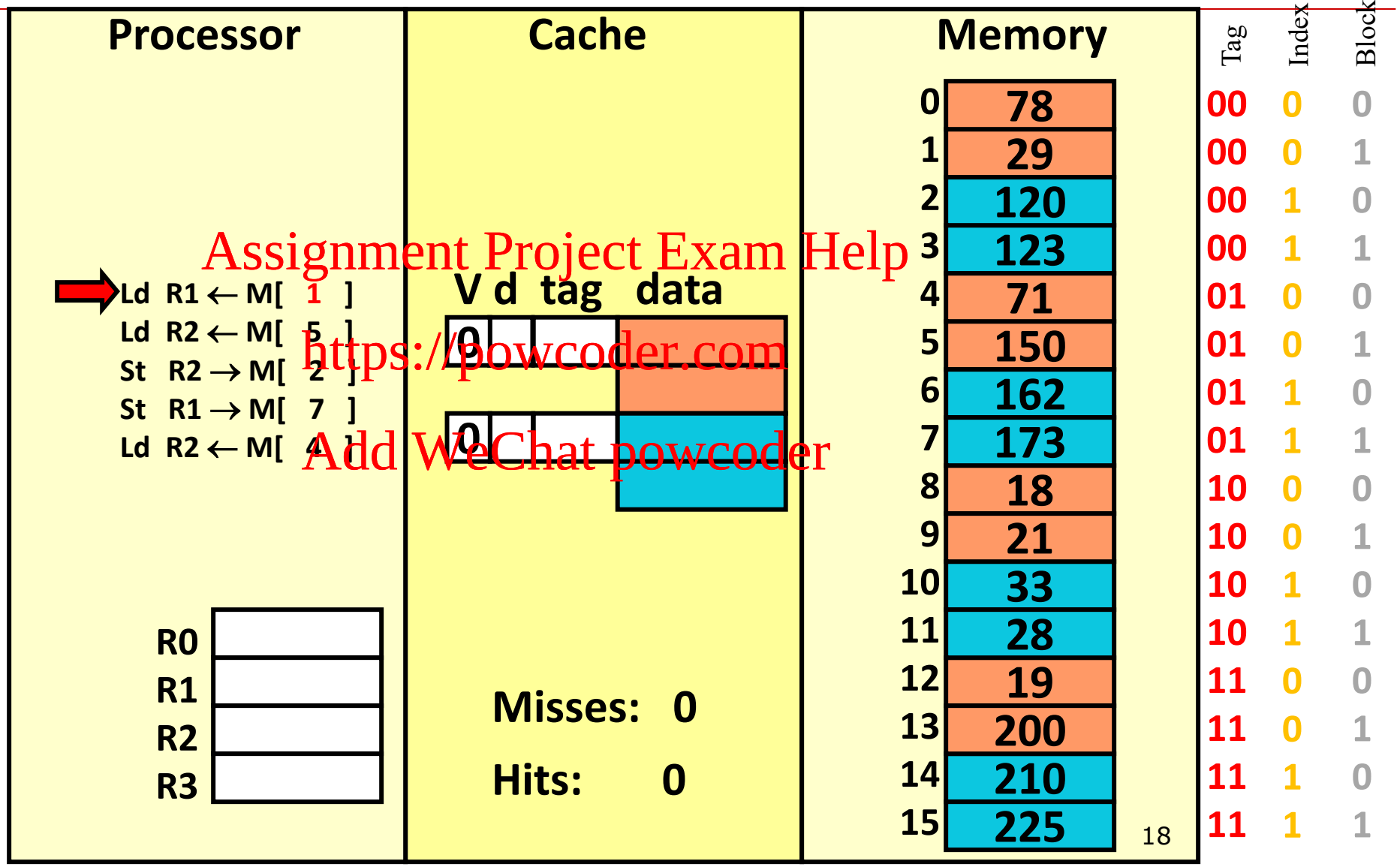
A, B, A, B, A, B, A, B, …

All accesses are conflict misses

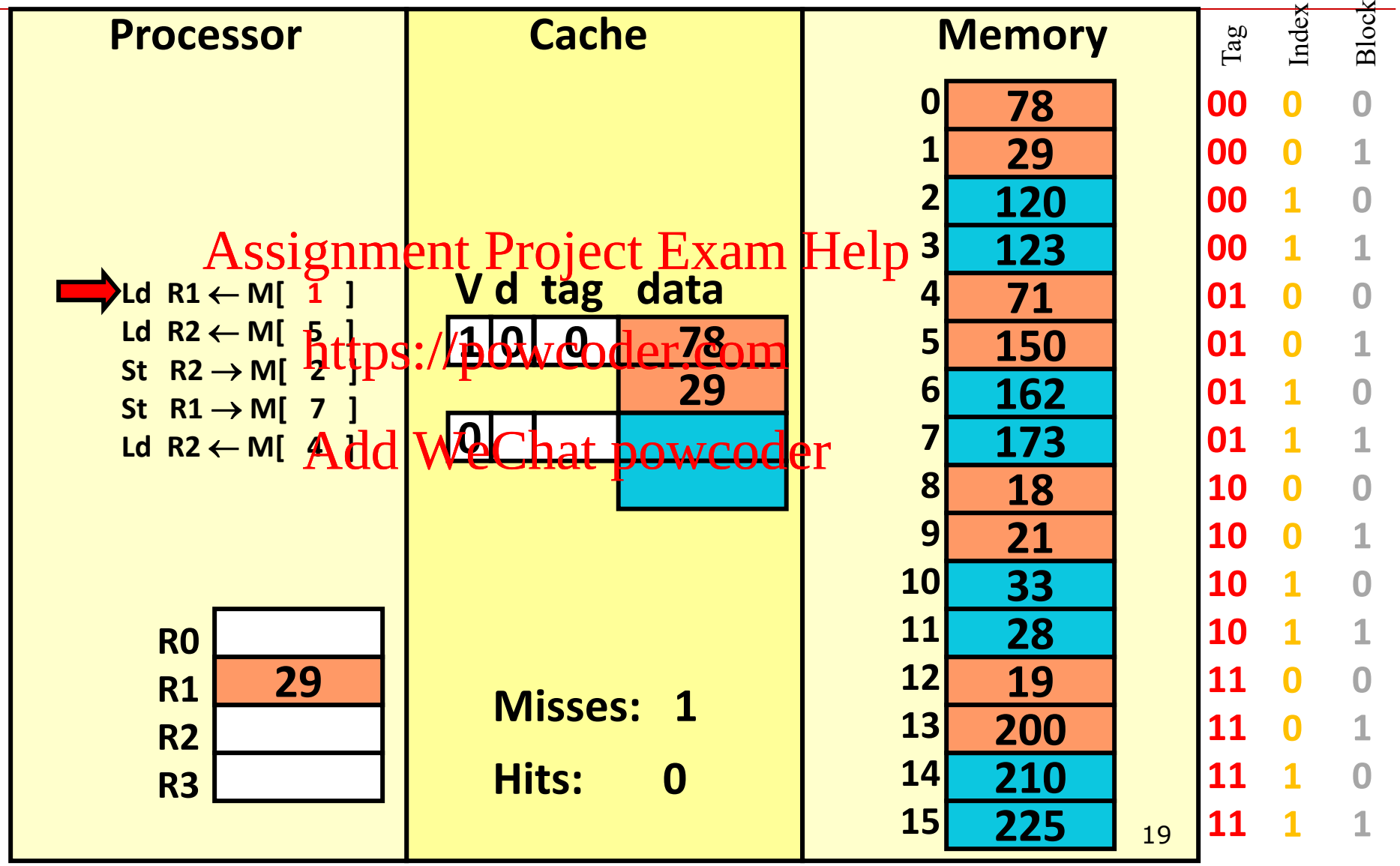
Direct-mapped cache



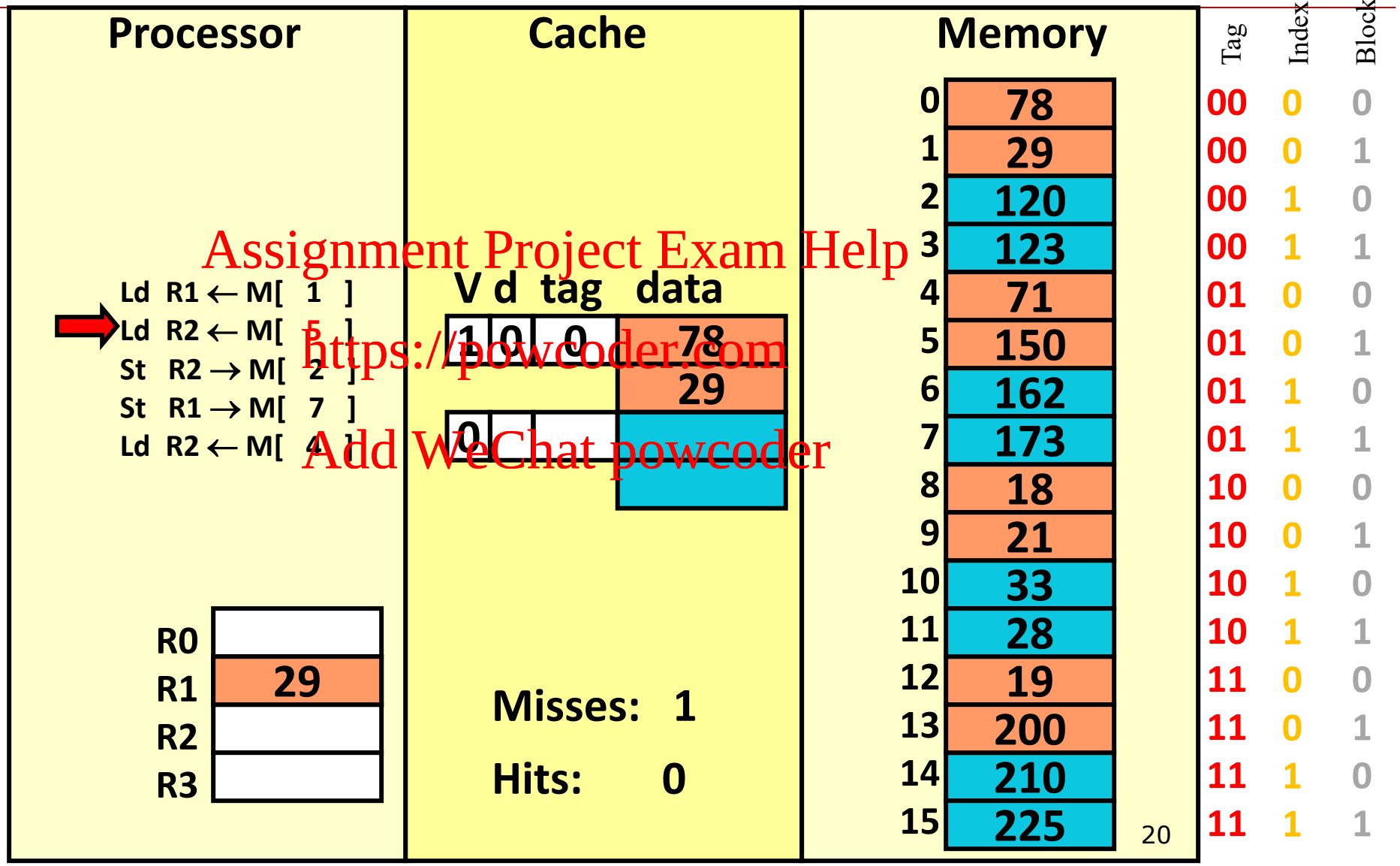
Direct-mapped (REF 1)



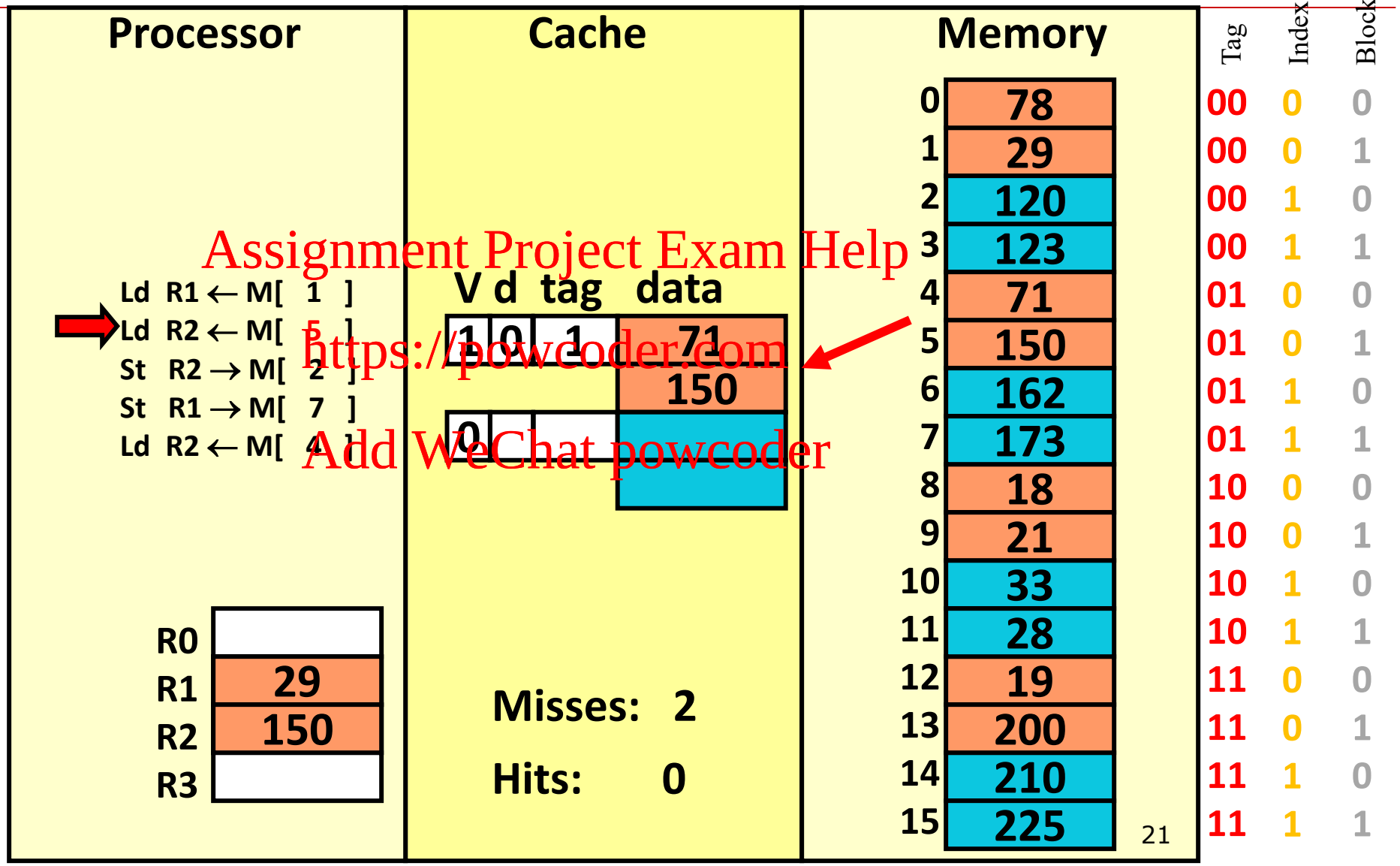
Direct-mapped (REF 1)



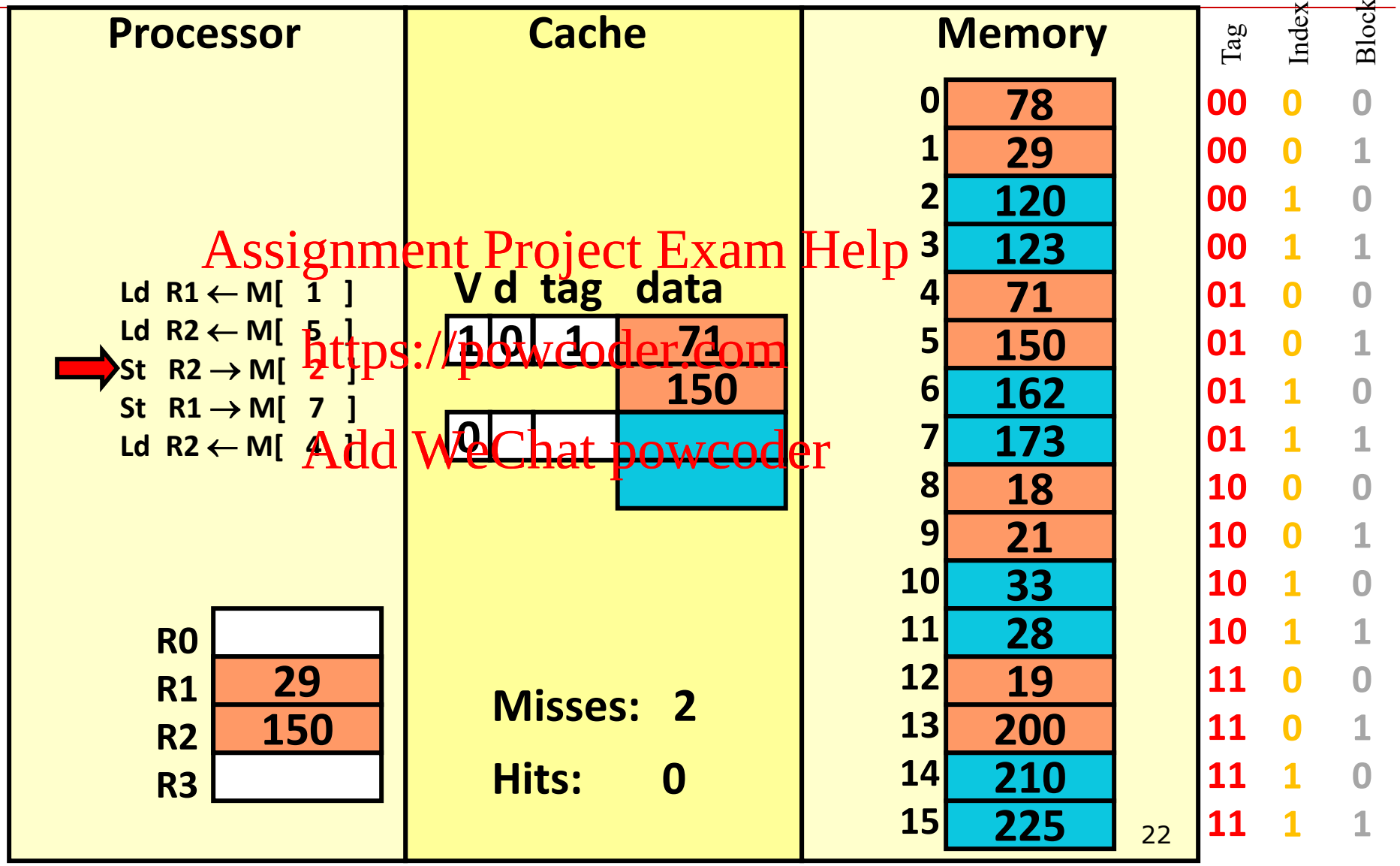
Direct-mapped (REF 2)



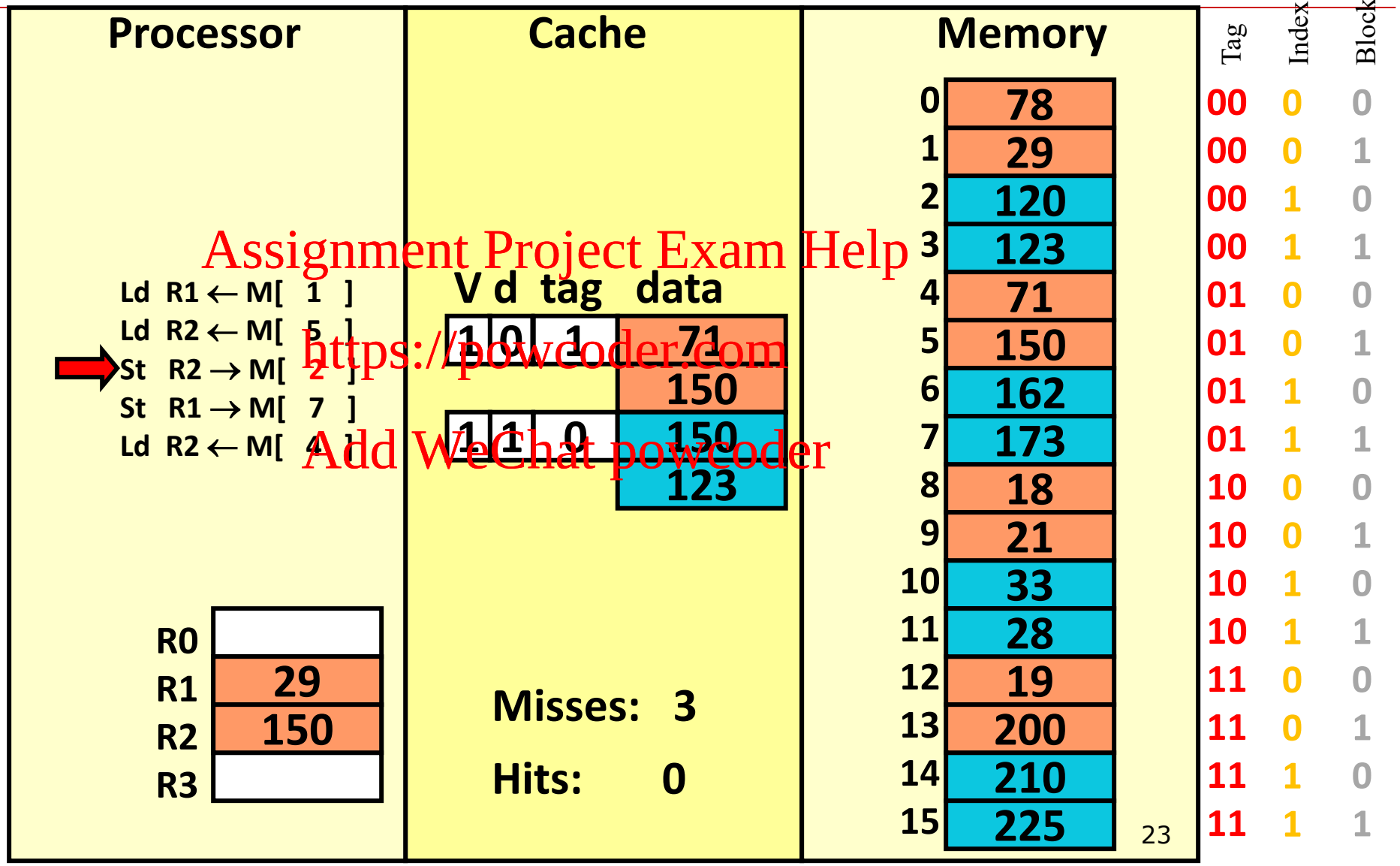
Direct-mapped (REF 2)



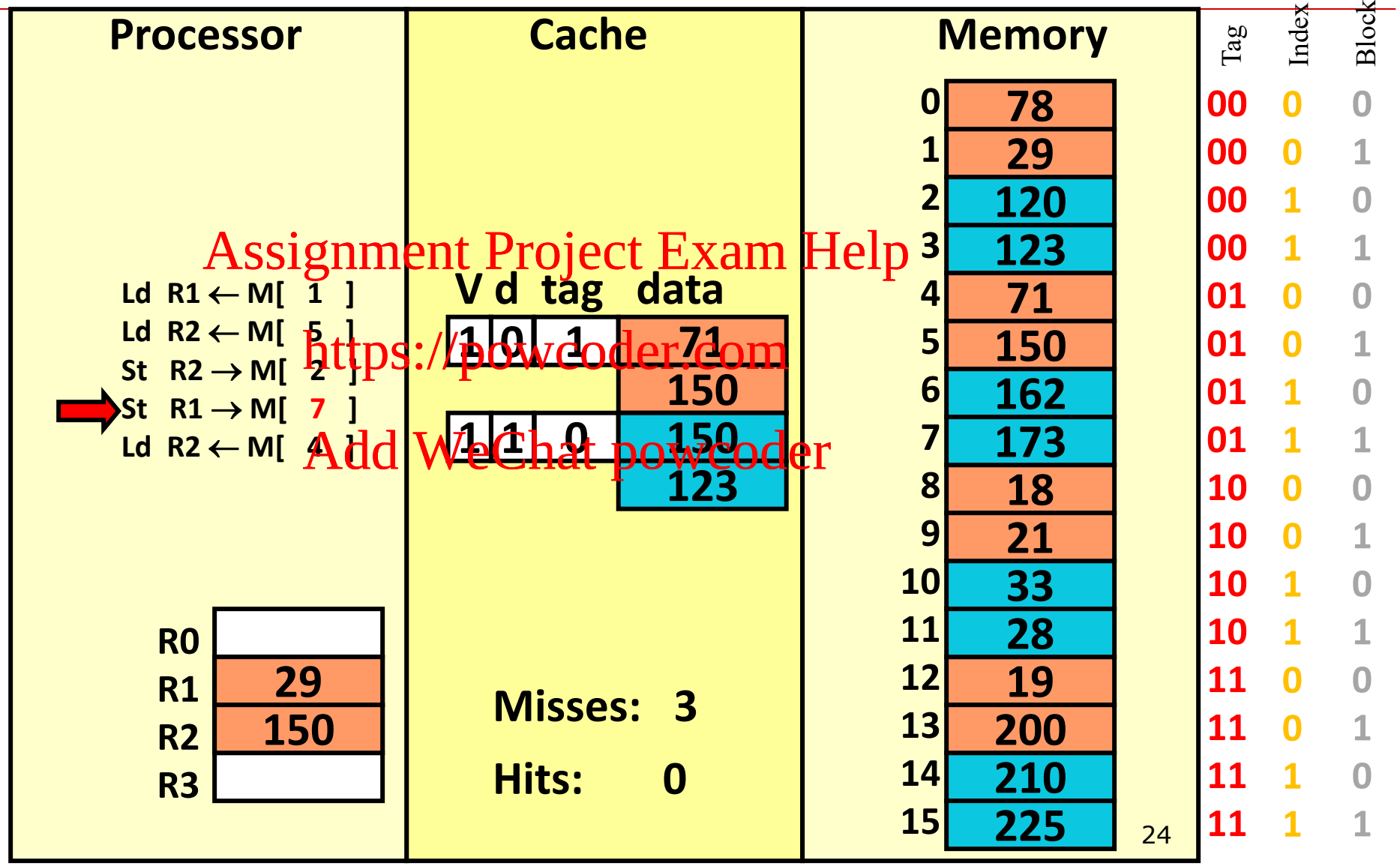
Direct-mapped (REF 3)



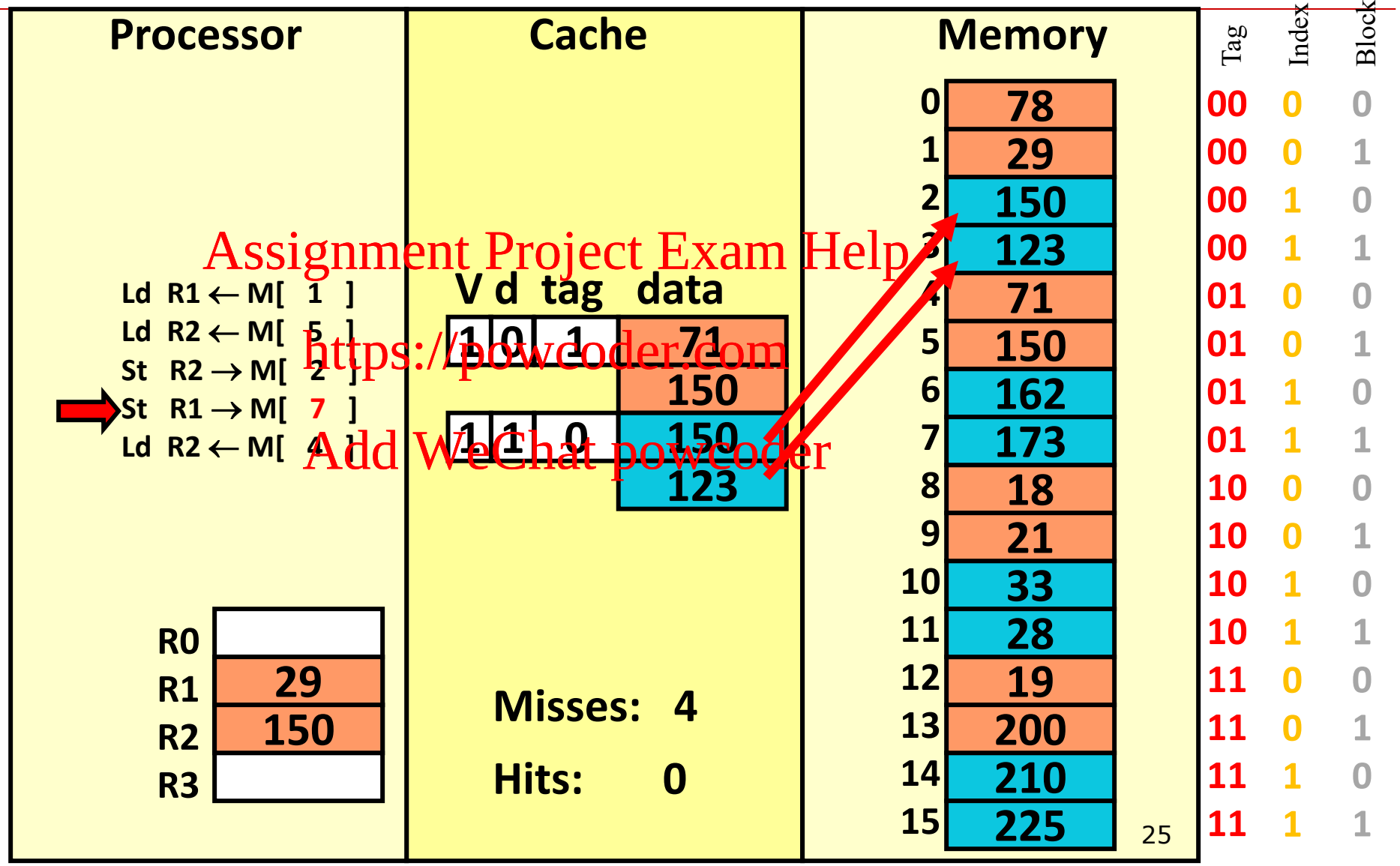
Direct-mapped (REF 3)



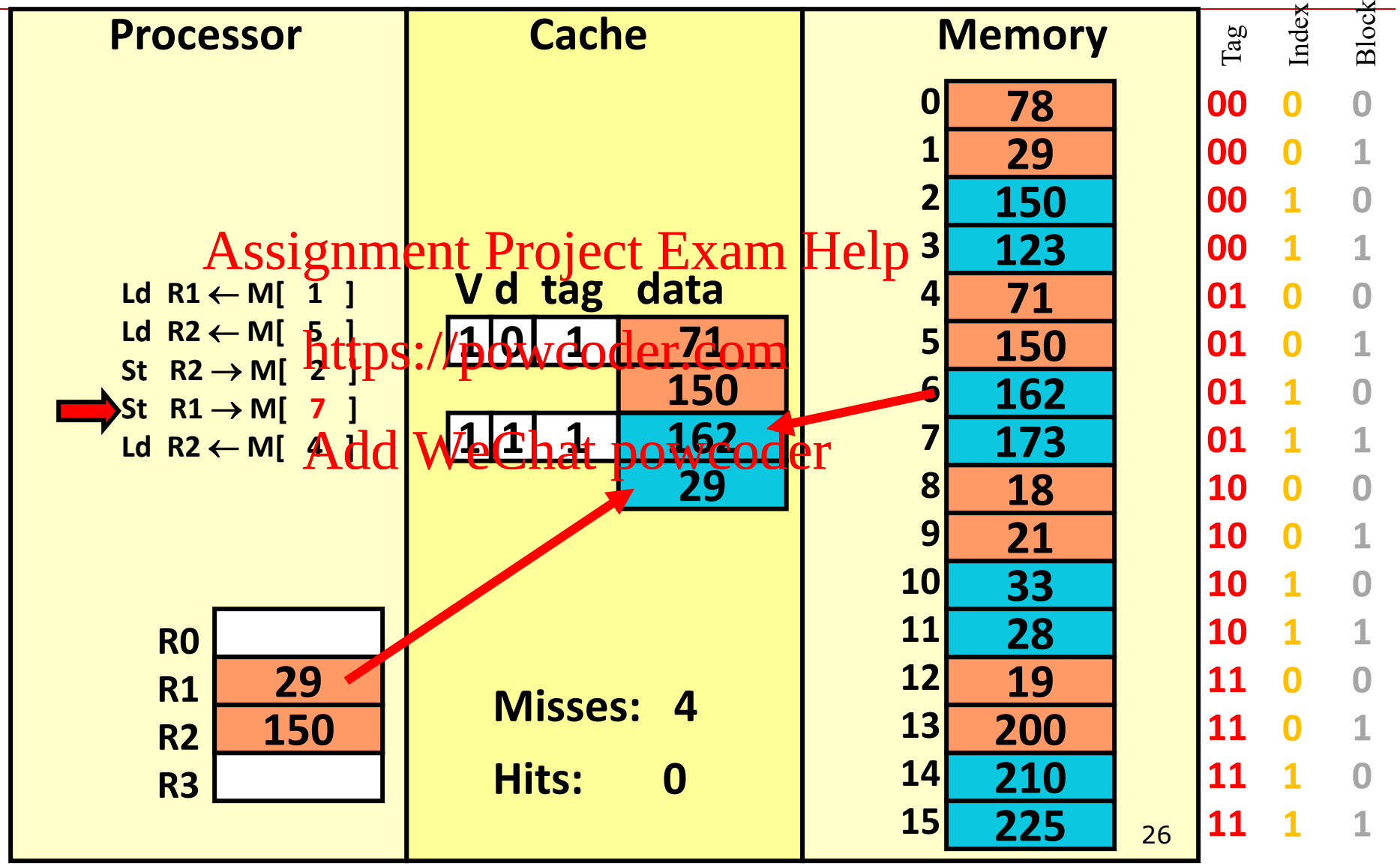
Direct-mapped (REF 4)



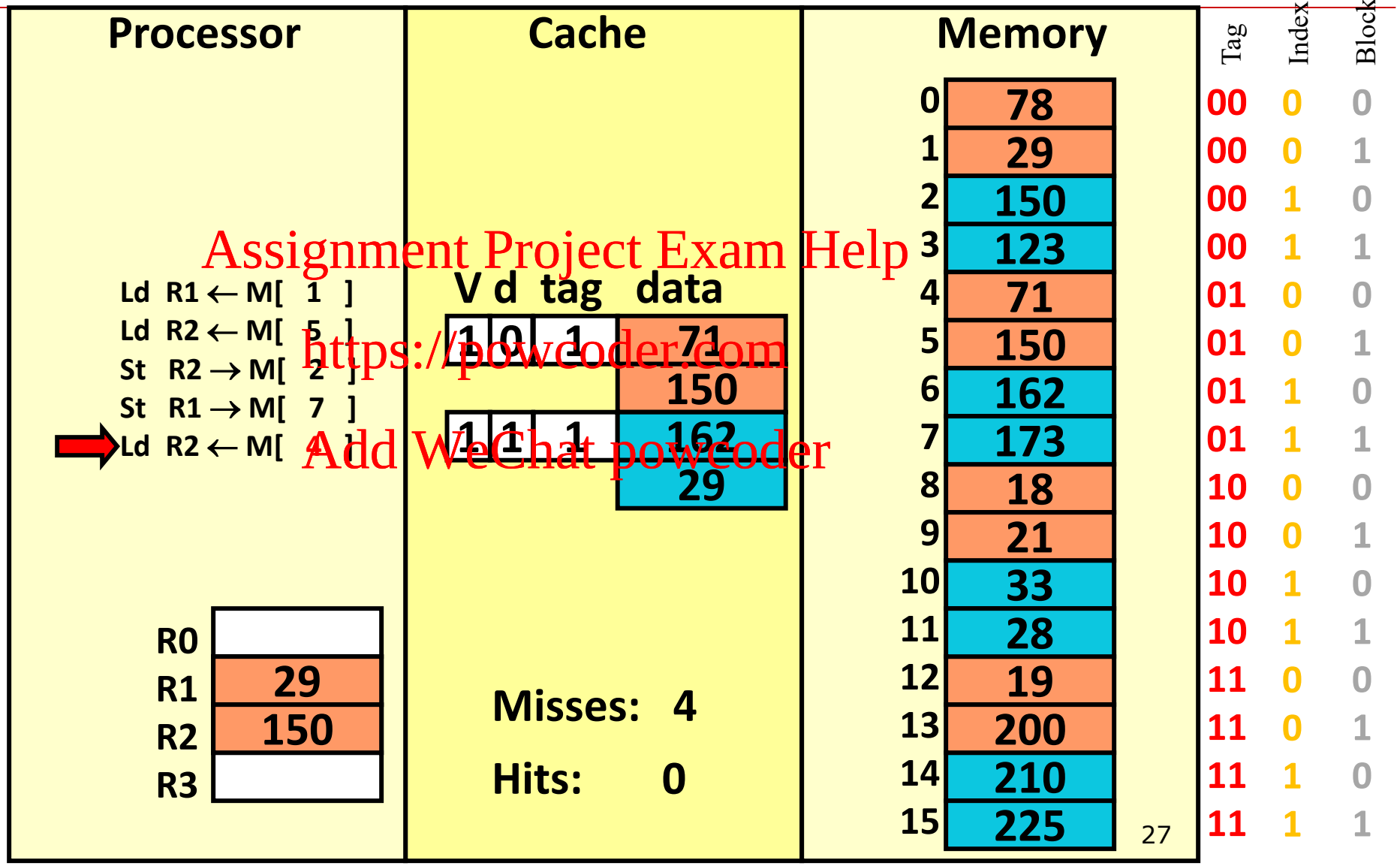
Direct-mapped (REF 4)



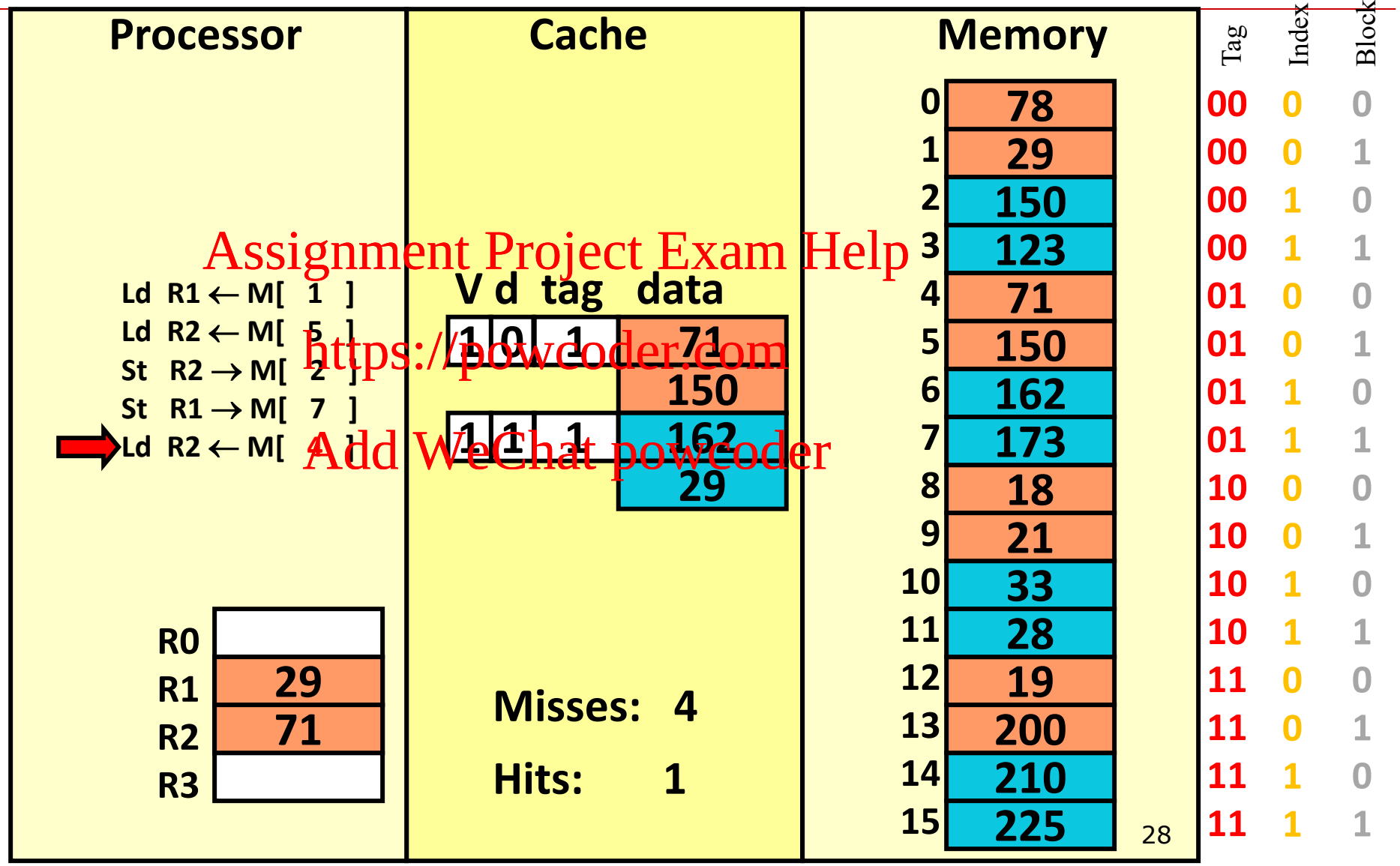
Direct-mapped (REF 4)



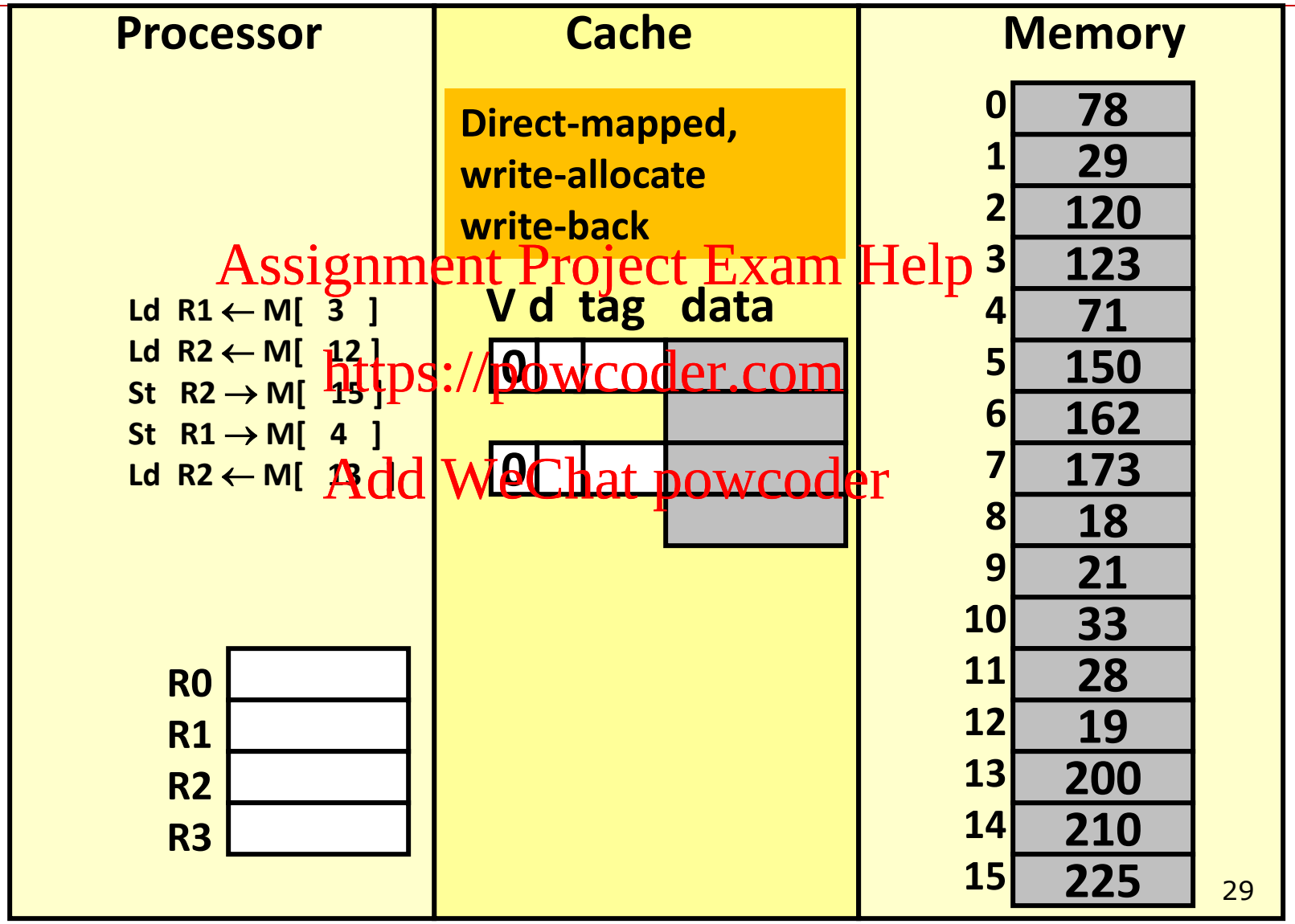
Direct-mapped (REF 5)



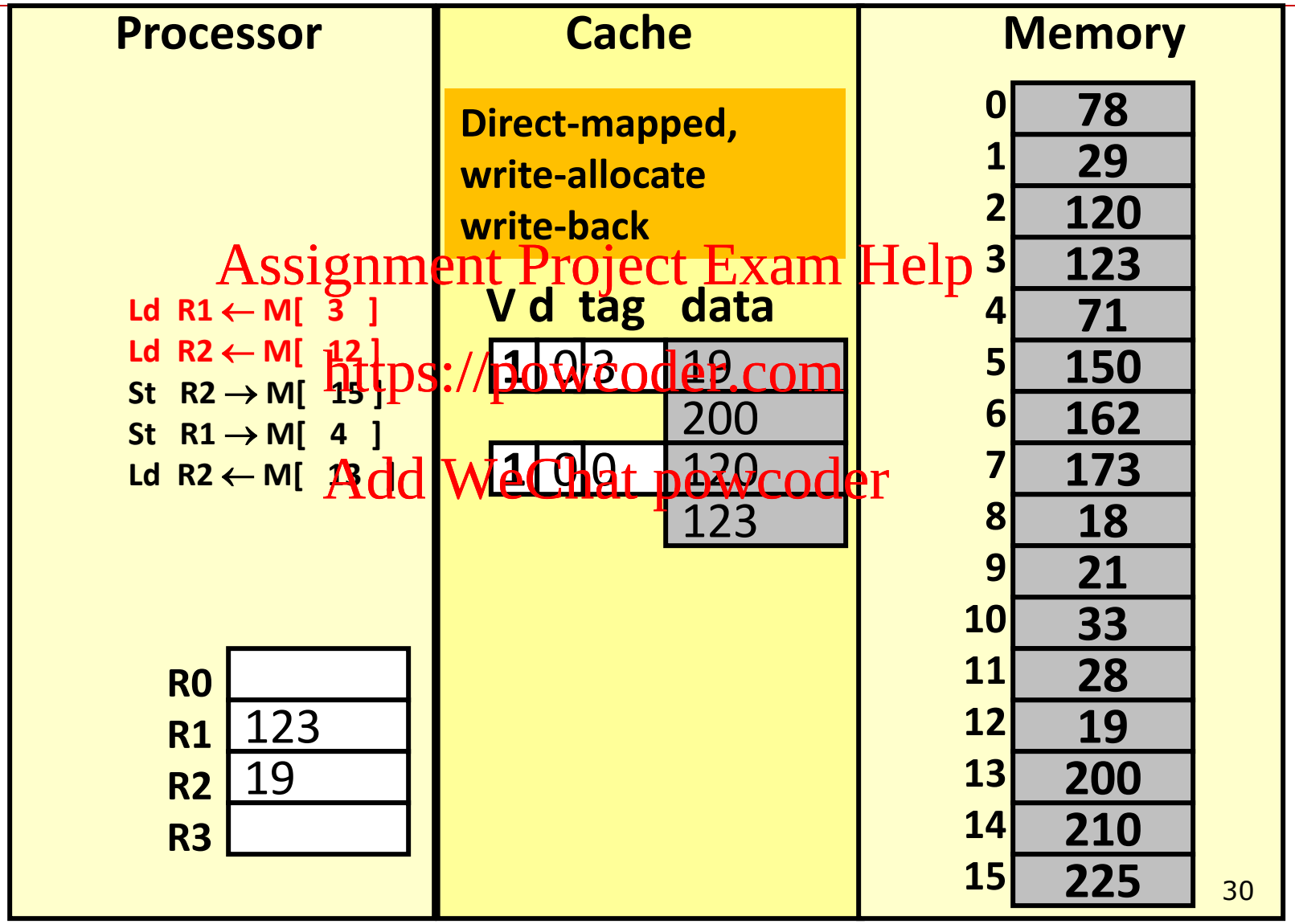
Direct-mapped (REF 5)



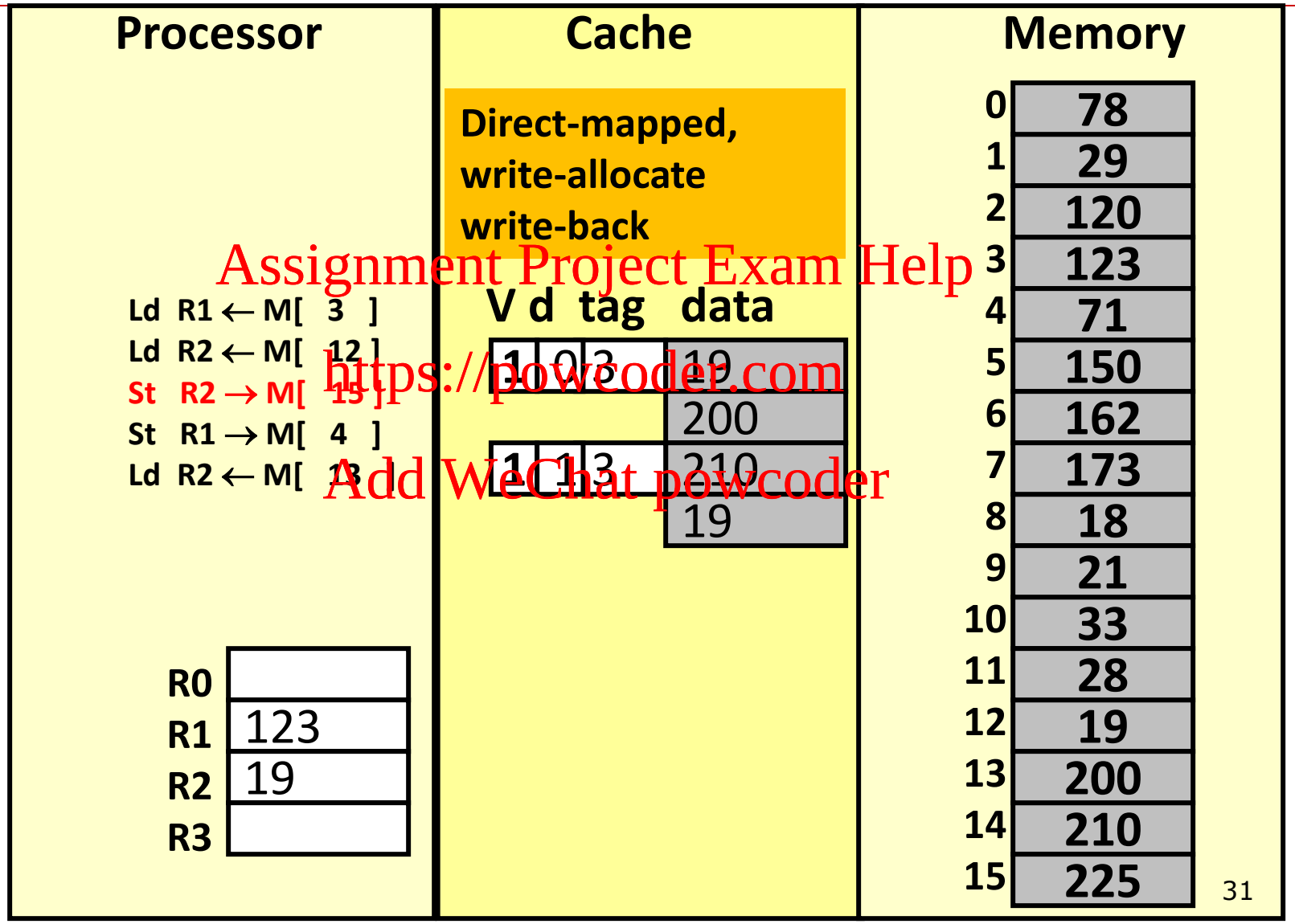
Class Problem – What is the state of the cache after executing the following instruction sequence?



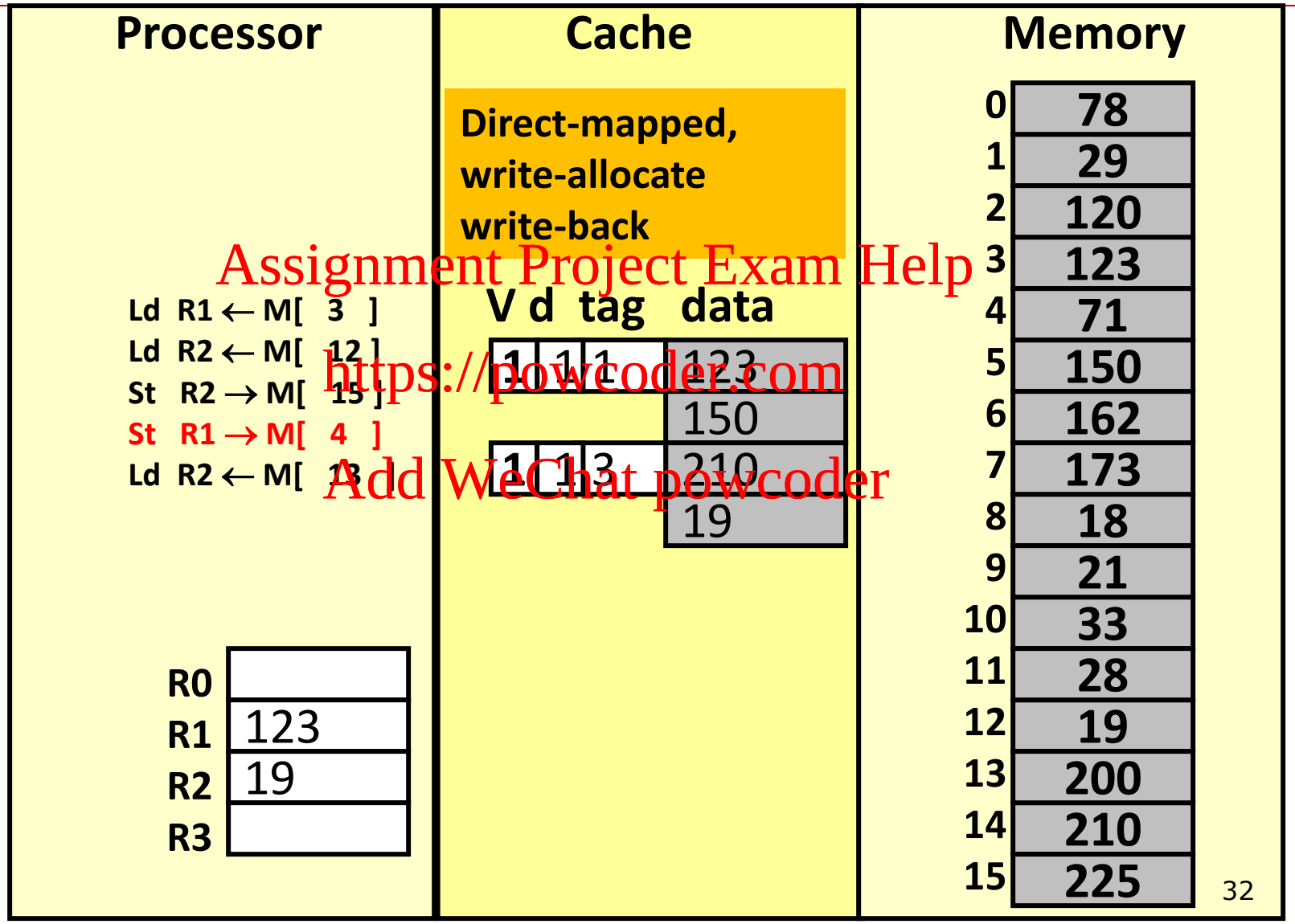
Class Problem – What is the state of the cache after executing the following instruction sequence?



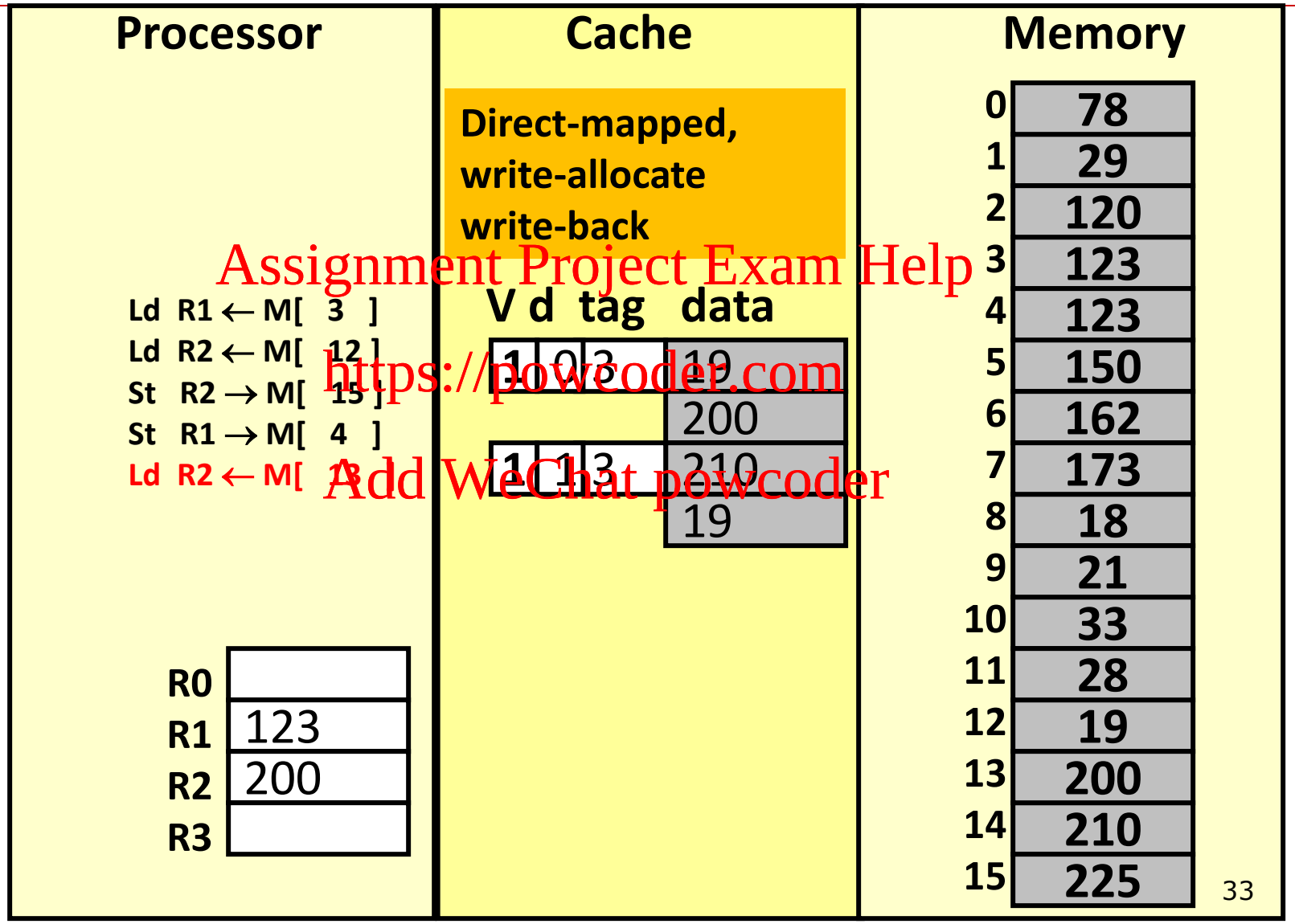
Class Problem – What is the state of the cache after executing the following instruction sequence?



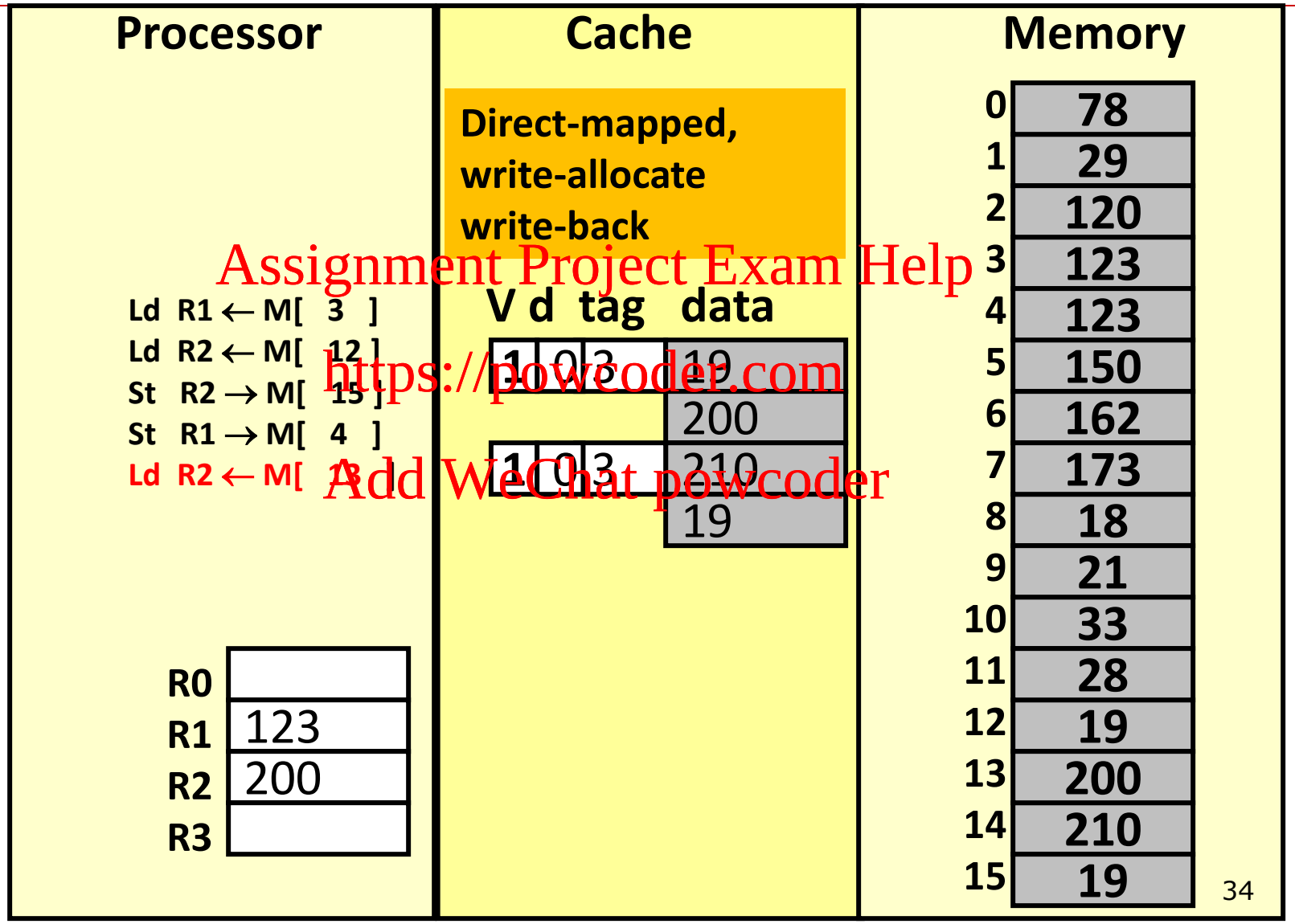
Class Problem – What is the state of the cache after executing the following instruction sequence?



Class Problem – What is the state of the cache after executing the following instruction sequence?



Class Problem – What is the state of the cache after executing the following instruction sequence?



Class Problem

How many tag bits are required for:

32-bit address, byte addressed, direct-mapped 32k cache, 128 byte block size, write-back

What are the overheads of this cache?

Class Problem

How many tag bits are required for:

32-bit address, byte addressed, direct-mapped 32k cache, 128 byte block size, write-back

 # Bytes in block = 128 => Block offset size = 7 bits (*byte addressable*)

 # Lines = 32k / 128 = 256 => Line index = 8 bits

Tag bits = 32 – 7 – 8 = 17 bits

What is the overhead of this cache?

17 bits (Tag) + 1 bit (Valid) + 1 bit (Dirty) = 19 bits / line

19 bits / line * 256 lines = 4864 bits

4864 bits / 32KB = 1.9% overhead

What about cache for instructions?

Instructions should be cached as well

We have two choices:

1. Treat instruction fetches as normal data and allocate cache lines when fetched
2. Create a second cache (called the I-cache) which caches instructions only

How do you know which cache to use?

What are advantages of a separate ICache?

Integrating Caches into a Pipeline

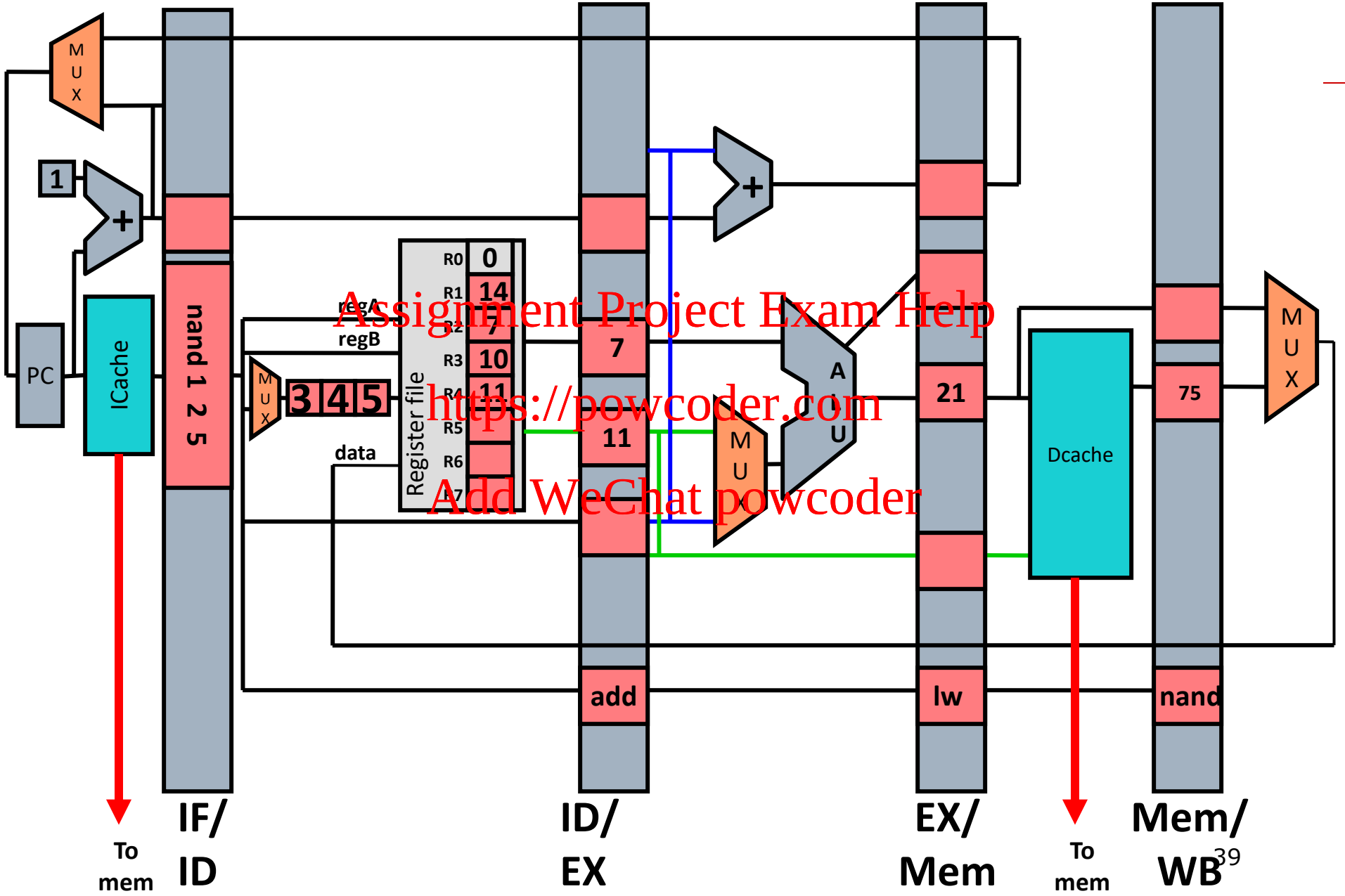
How are caches integrated into a pipelined implementation?

- Replace instruction memory with Icache
- Replace data memory with Dcache

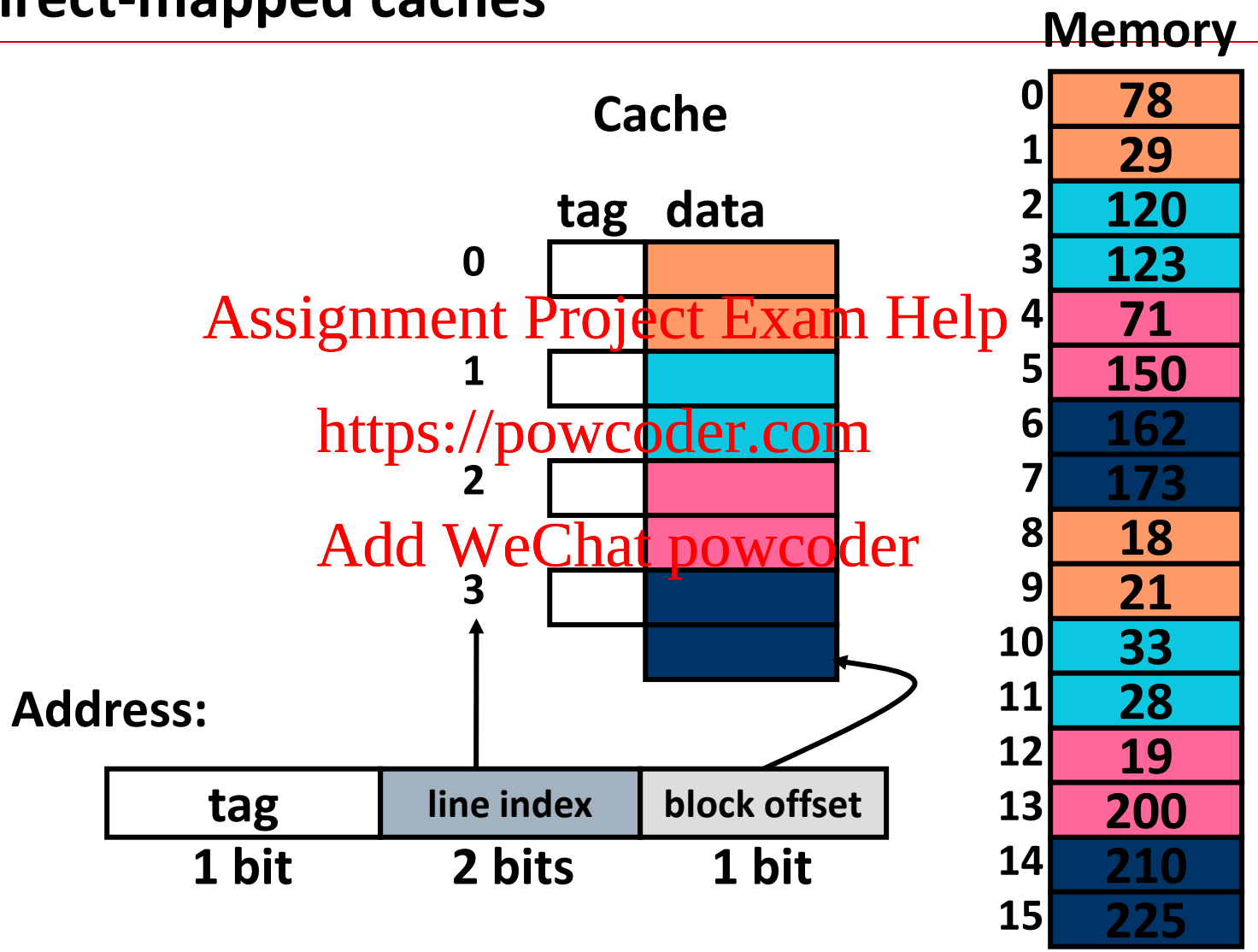
Issues:

- Memory accesses now have variable latency
- Both caches may miss at the same time

LC2K Pipeline with Caches



Summary: Direct-mapped caches



Next lecture: Get the advantage of both...

Set associative caches:

- Partition memory into regions
  - like direct mapped but fewer partitions
- Associate a region to a set of cache lines
  - Check tags for all lines in a set to determine a HIT

Treat each line in a set like a small fully associative cache

- LRU (or LRU-like) policy generally used

Set-associative cache