20. Caches: Set Associative

EECS 370 – Introduction to Computer Organization – Fall 2020

Satish Narayanasamy

EECS Department
University of Michigan in Ann Arbor, USA

Announcements

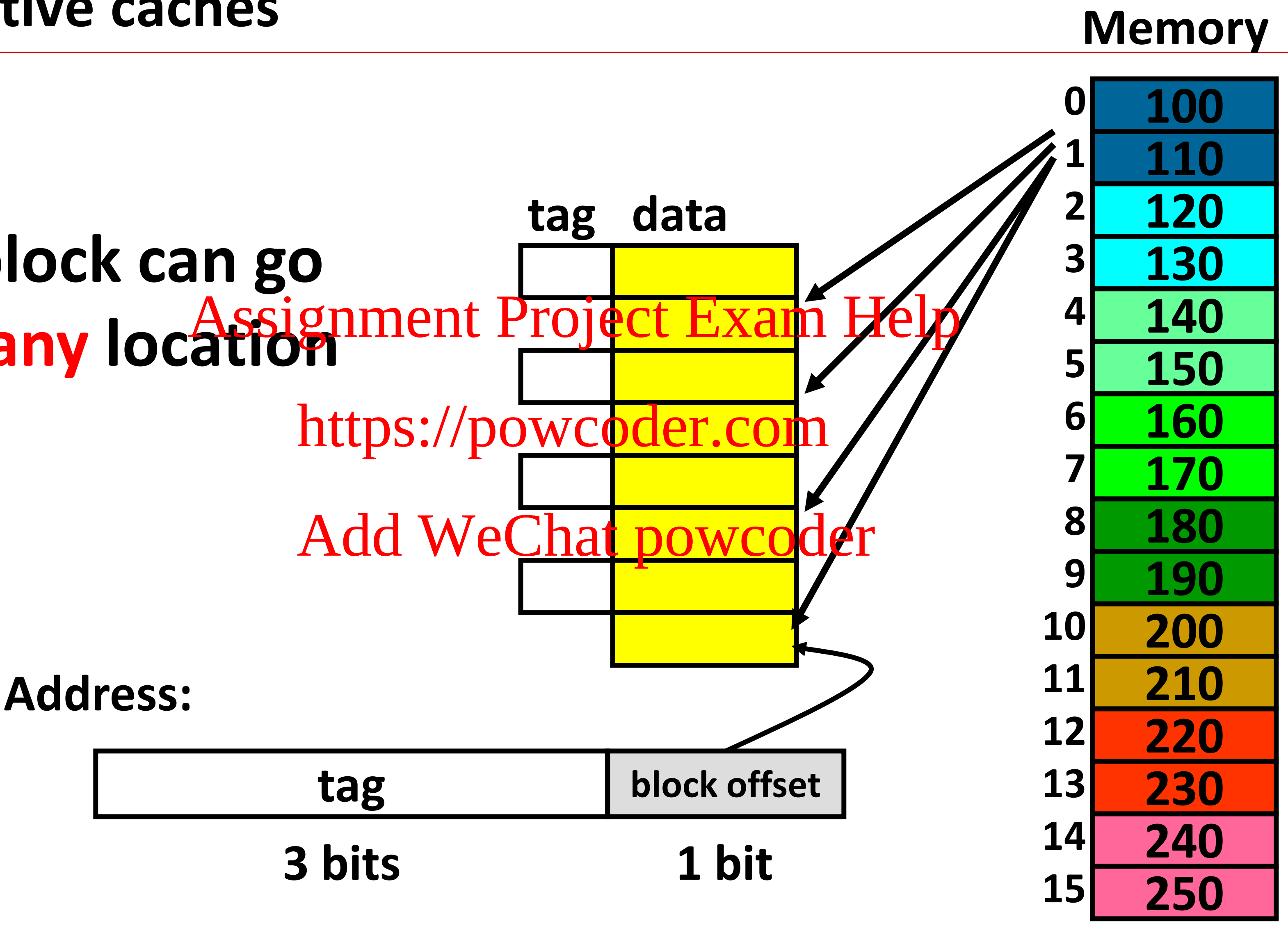
Upcoming deadlines:

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

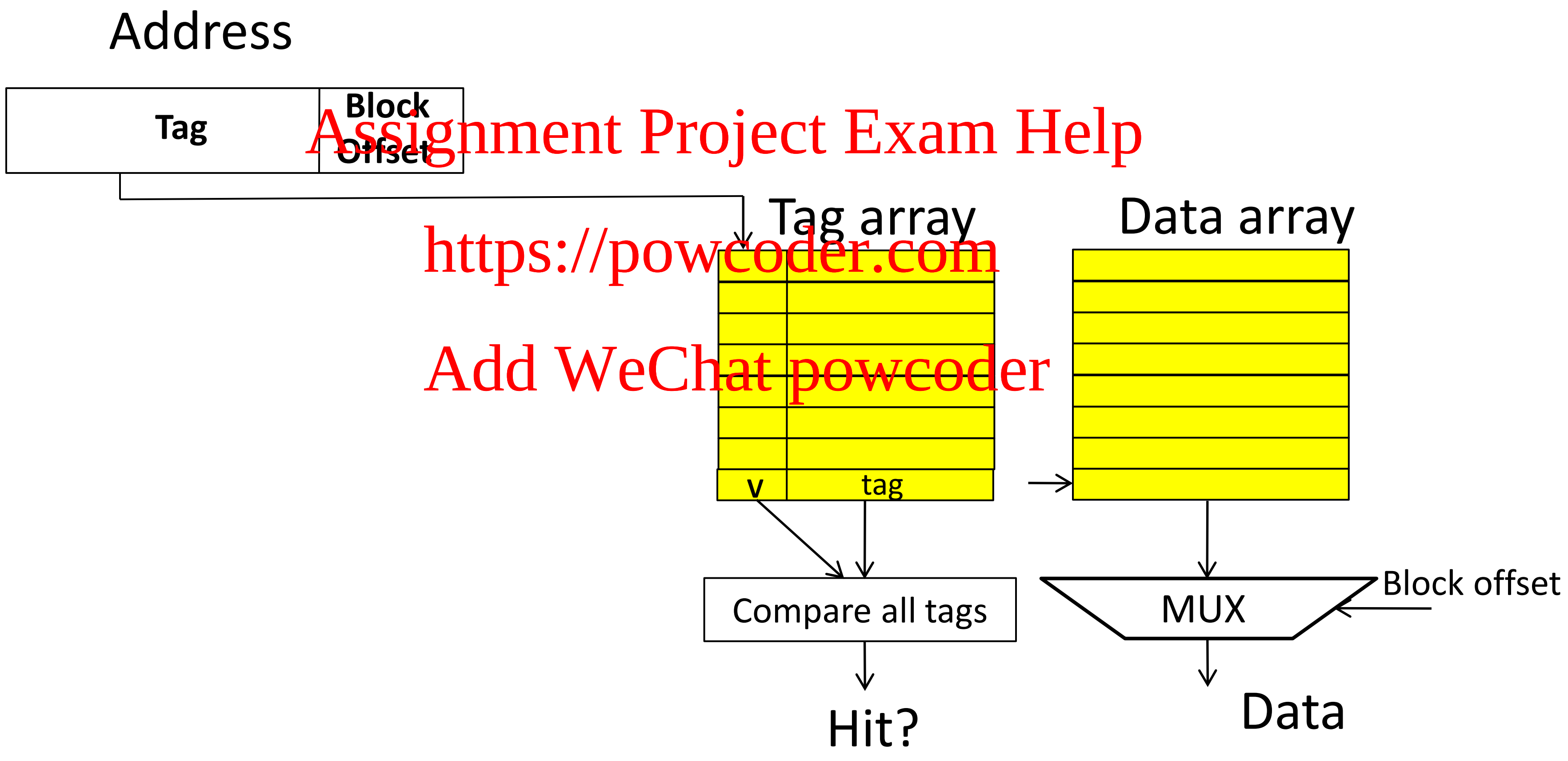
Grading policy: Best of two

Fully-associative caches

A block can go
to **any** location

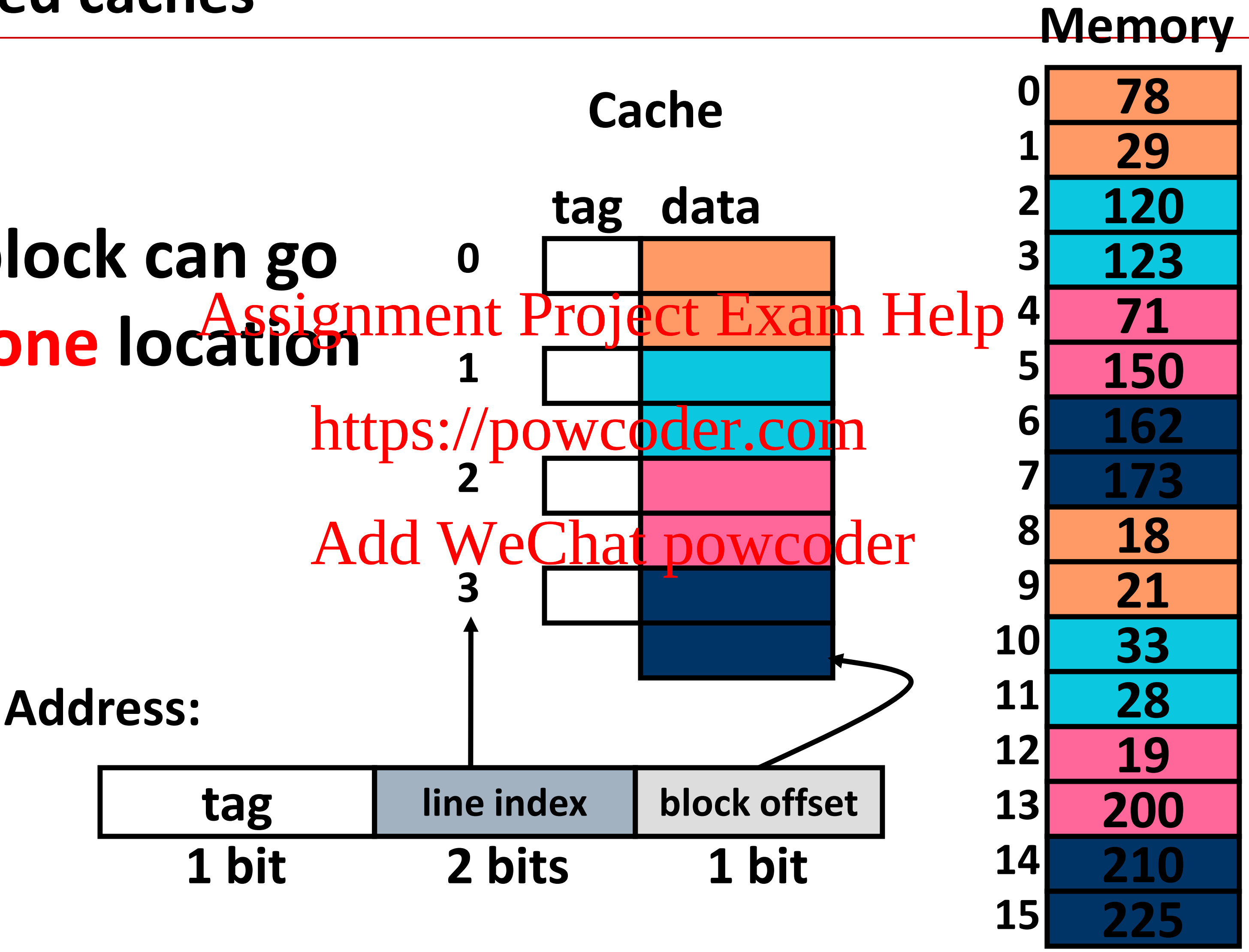


Fully-associative cache: Placement & Access

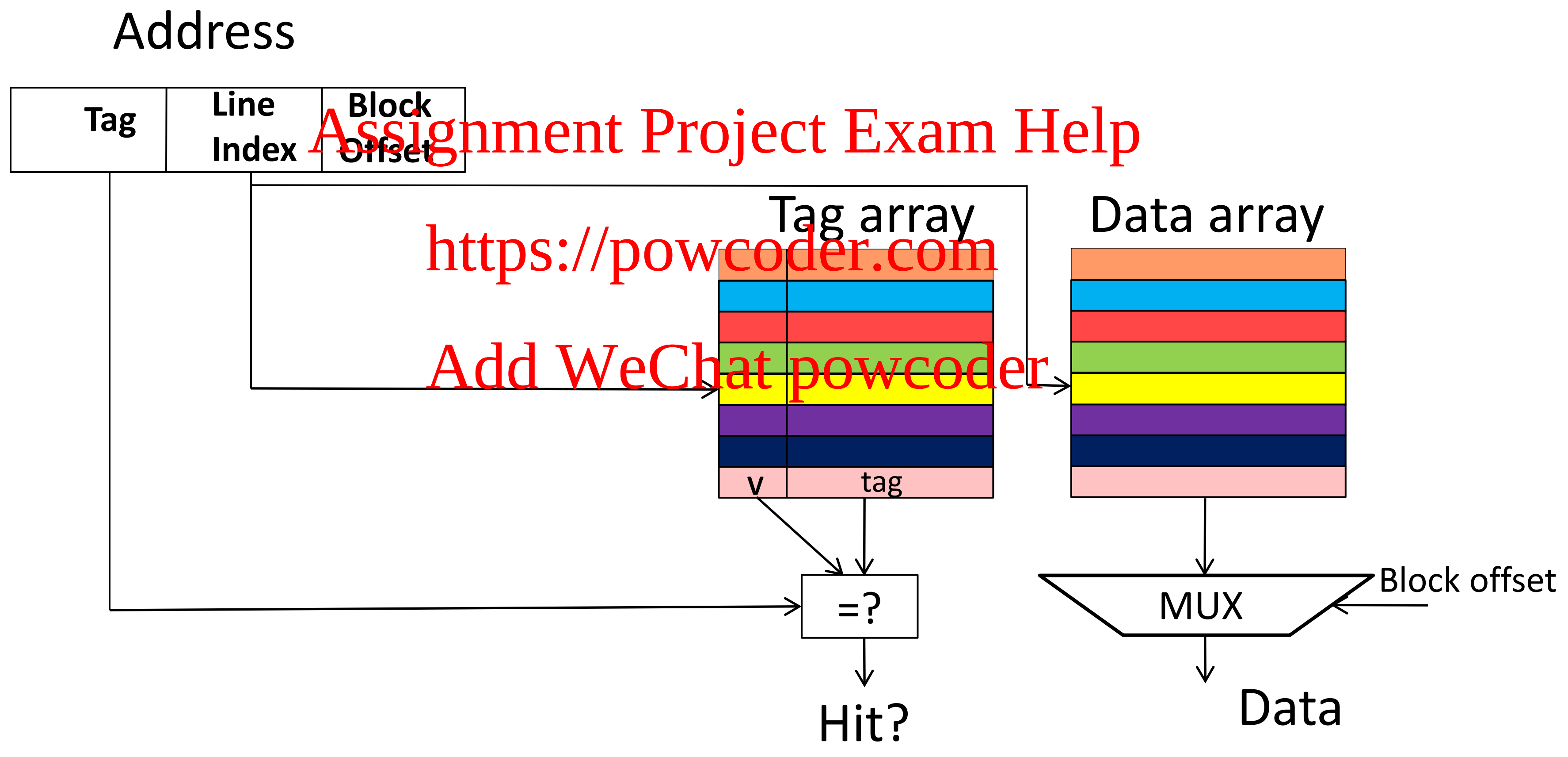


Direct-mapped caches

A block can go to one location



Direct-mapped cache: Placement & Access



This lecture

Set Associative Caches

Idea

Illustration

3C problem

The middle ground...

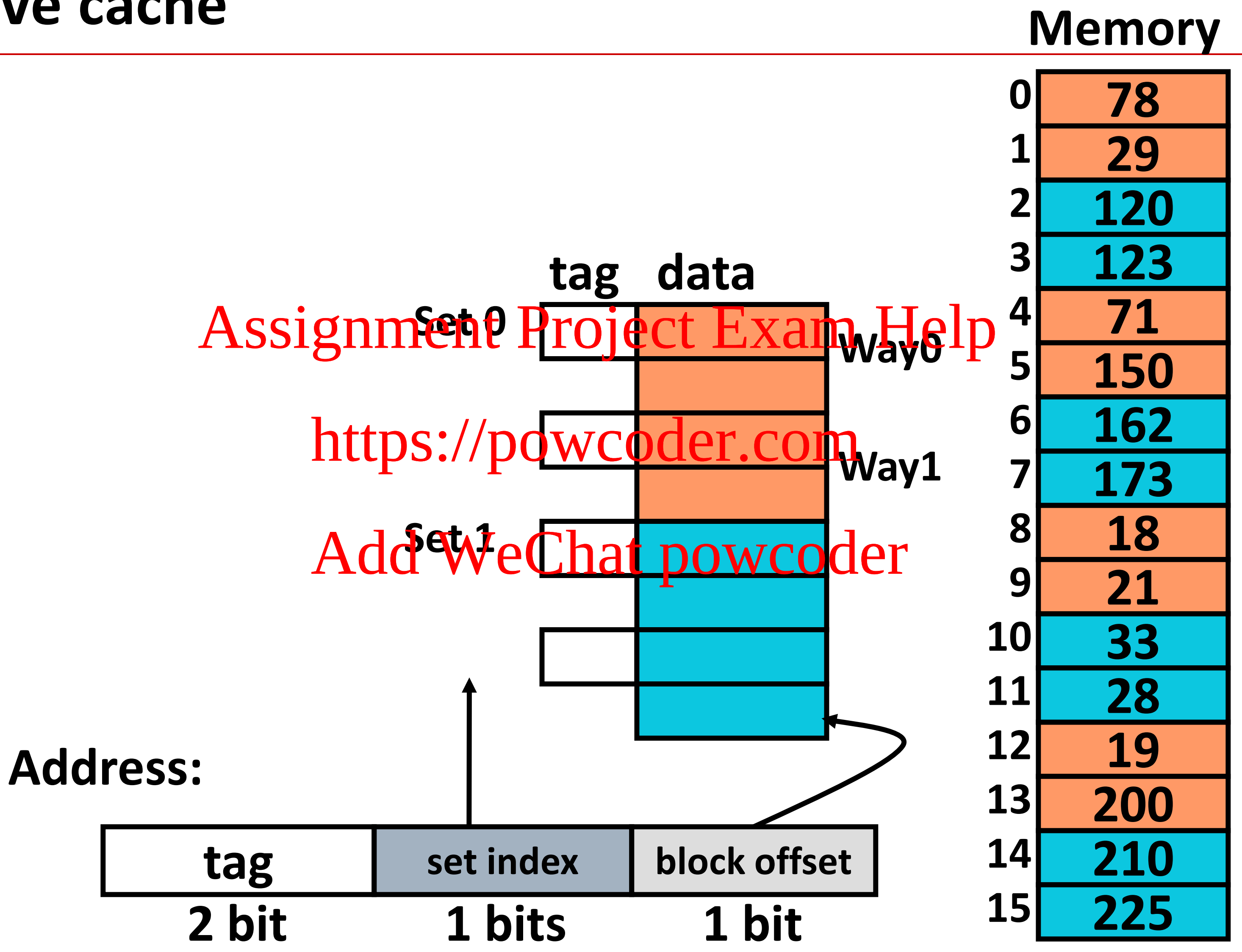
Set associative caches

- 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

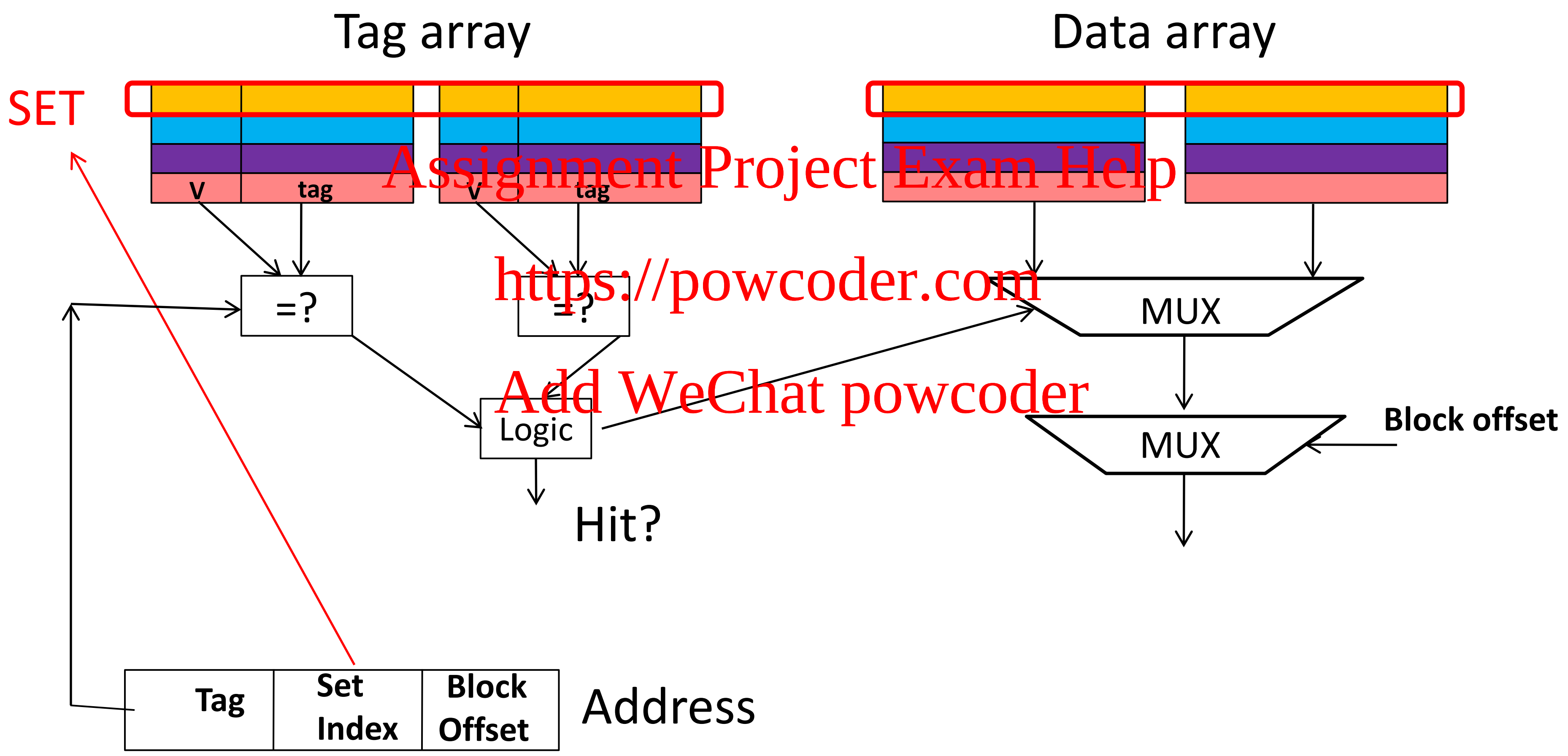
Treat each line in a set like a small fully associative cache

- LRU (or LRU-like) policy generally used

Set-associative cache



Set-associative cache: Placement & Access

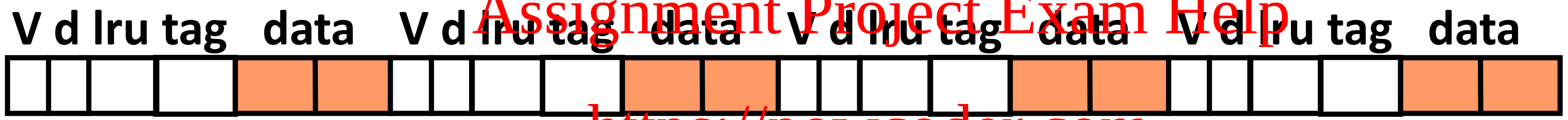


Cache Organization Comparison

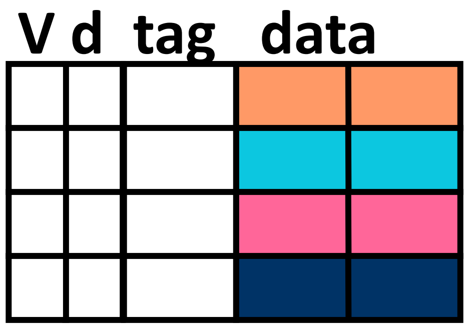
Cache size = 8 bytes (for all caches)
Block size = 2 bytes
#blocks = 4

Fully associative

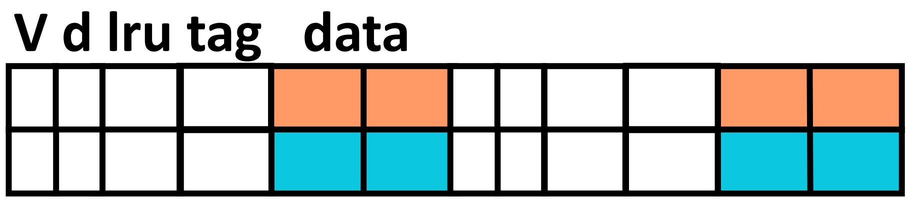
# blocks per set = all blocks = 4 in this example;
so, also correct to view this cache as 4-way associative


V d lru tag data V d lru tag data V d lru tag data V d lru tag data

Direct mapped: (#blocks per set = 1)



2-way associative (#blocks per set = 2)



Cache Organization: Equations

Block

#blocks = cache size / block_size

#cache lines = #blocks

block_offset_size = log2(#block_size)

Set

#sets = #lines/ #ways = #lines / (lines per set)

Direct-mapped:	#sets =	#lines/1	
2-way associative:	#sets =	#lines/2	
n-way associative:	#sets =	#lines/n	
fully-associative:	#sets =	1	(all lines are in 1 set)

set_index_size = log2(#sets)

Tag size = address size – set_index_size – block_offset_size

Class Problem 1

For a 32-bit address and 16KB cache with 64-byte blocks, show the breakdown of the address for the following cache configuration:

A) fully associative cache

B) 4-way set associative cache

C) Direct-mapped cache

Class Problem 1 (Solution)

For a 32-bit address and 16KB cache with 64-byte blocks, show the breakdown of the address for the following cache configuration:

#lines = 16 KB / 64 byte = 256
Block_offset_size = log(block_size) = log(64)= 6 bits

A) fully associative cache

Tag = 32 - 6 = 26 bits

B) 4-way set associative cache

#sets = #lines / ways = #lines/ 4 = 64
Set Index size = log(64) = 6 bits

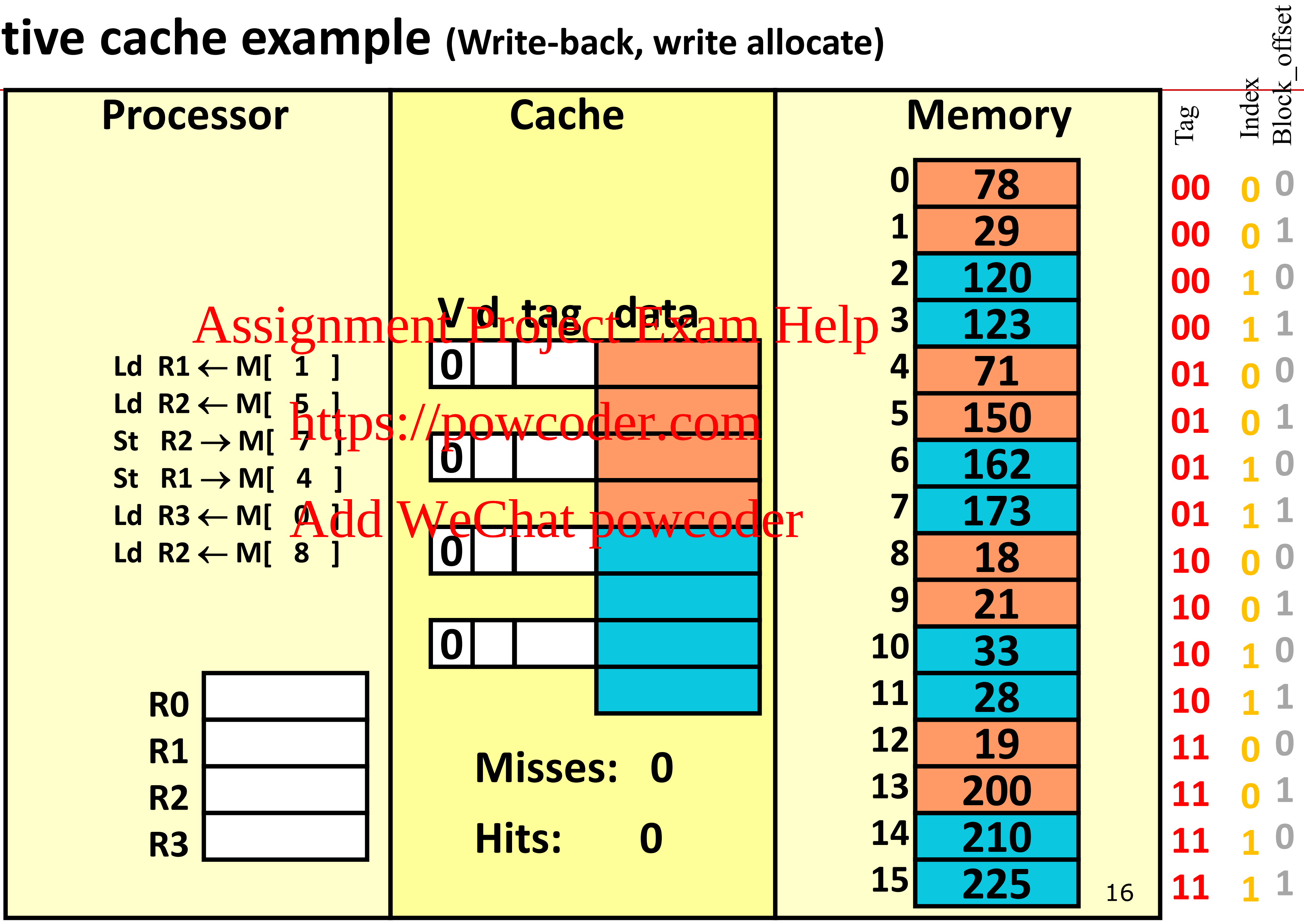
Tag = address_size – set_index_size – block_offset_size
= 32 - 6 - 6
= 20 bits

C) Direct-mapped cache

#sets = #lines / ways = #lines/1 = 256
Set_index_size = log(#sets) = log(256) = 8 bits
Tag = 32 – 6 – 8 = 18 bits

Set Associative Caches: Illustration

Set-associative cache example (Write-back, write allocate)



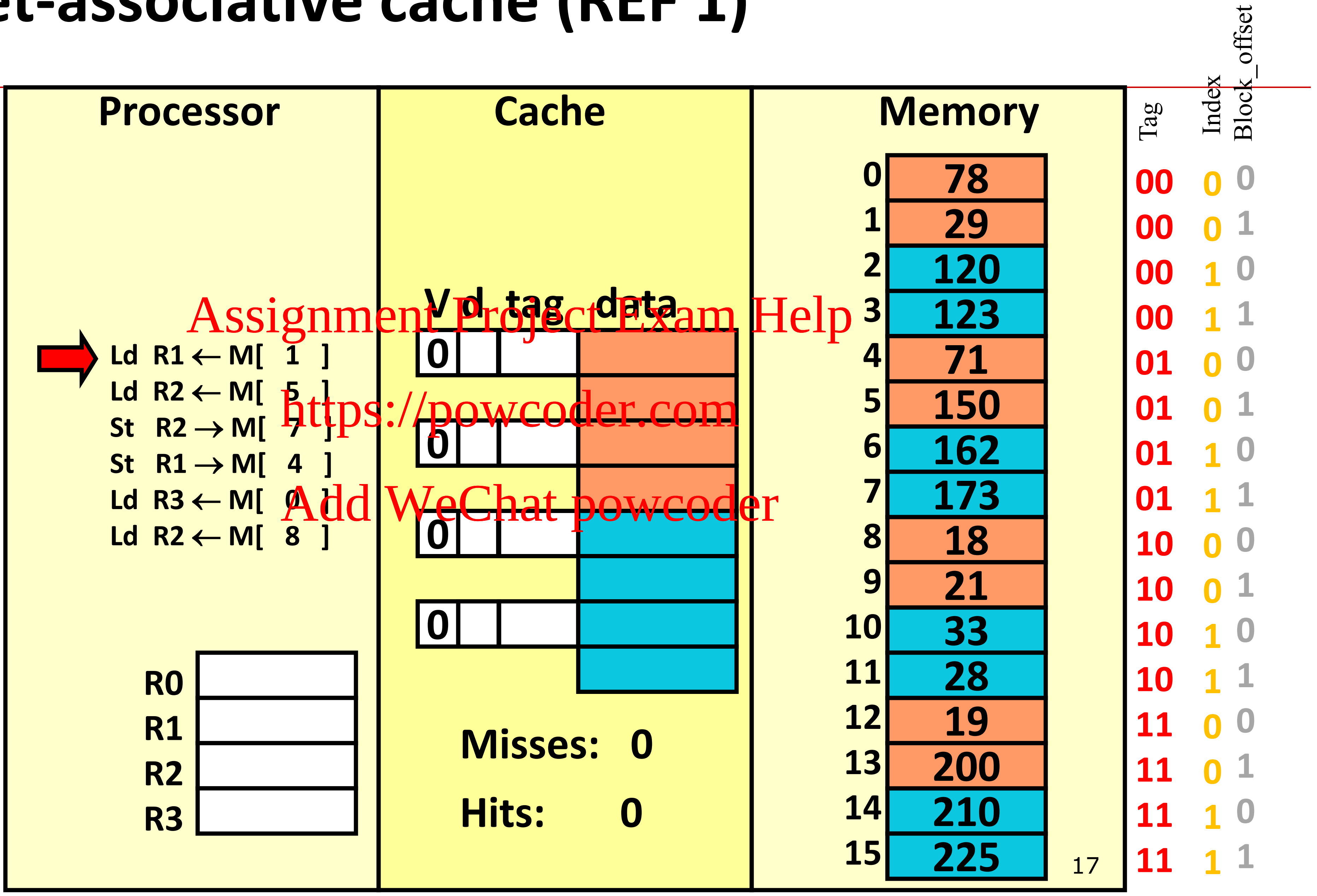
```
Ld  R1 ← M[  1  ]
Ld  R2 ← M[  5  ]
St  R2 → M[  7  ]
St  R1 → M[  4  ]
Ld  R3 ← M[  0  ]
Ld  R2 ← M[  8  ]
```

Misses: 0

Hits: 0

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

Set-associative cache (REF 1)



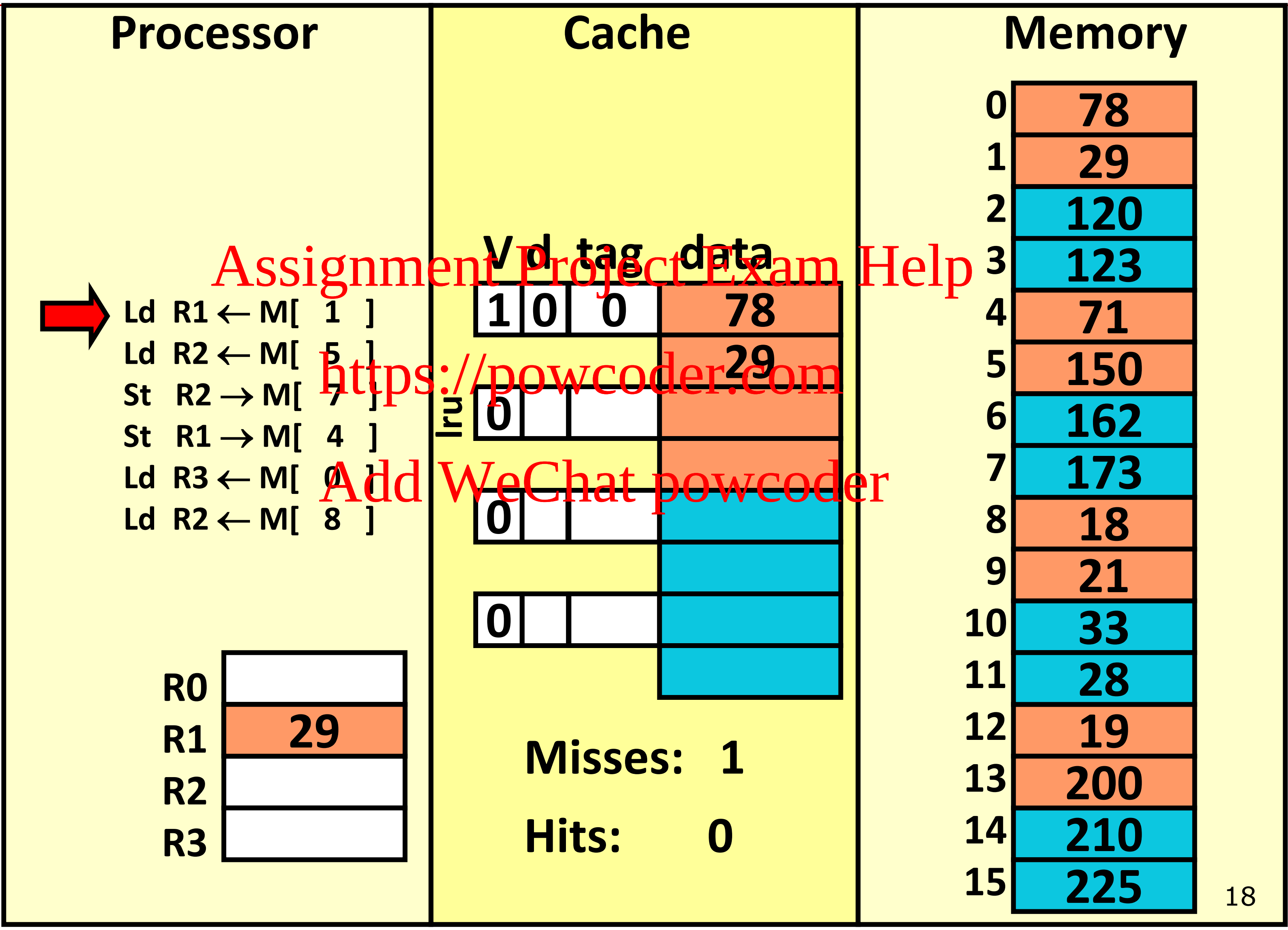
```
Ld  R1 ← M[  1  ]
Ld  R2 ← M[  5  ]
St  R2 → M[  7  ]
St  R1 → M[  4  ]
Ld  R3 ← M[  0  ]
Ld  R2 ← M[  8  ]
```

Misses: 0

Hits: 0

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

Set-associative cache (REF 1)



```
Ld  R1 ← M[  1  ]
Ld  R2 ← M[  5  ]
St  R2 → M[  7  ]
St  R1 → M[  4  ]
Ld  R3 ← M[  0  ]
Ld  R2 ← M[  8  ]
```

R0	
R1	29
R2	
R3	

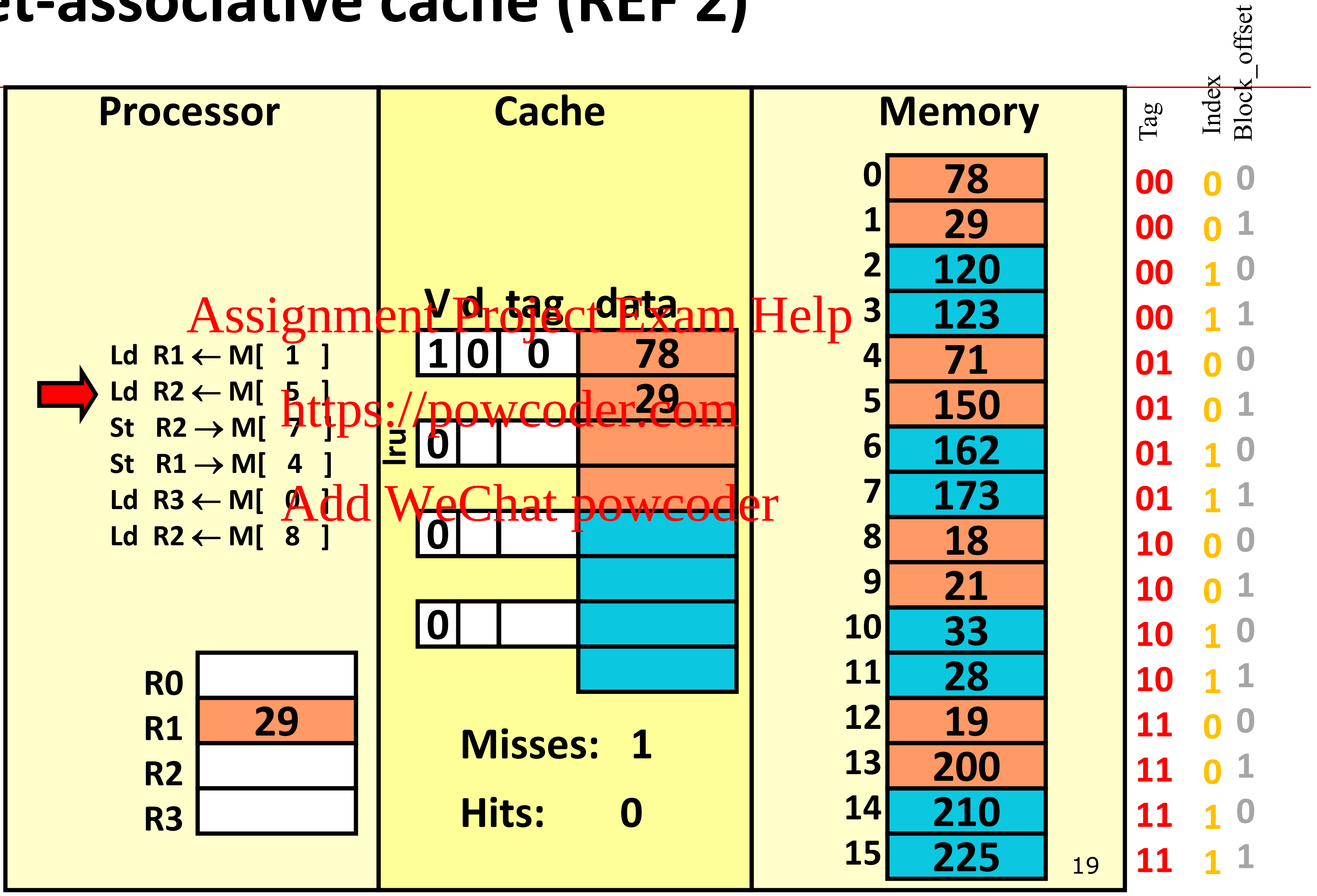
V	d	tag	data
1	0	0	78
			29
0			
0			
0			

Misses: 1

Hits: 0

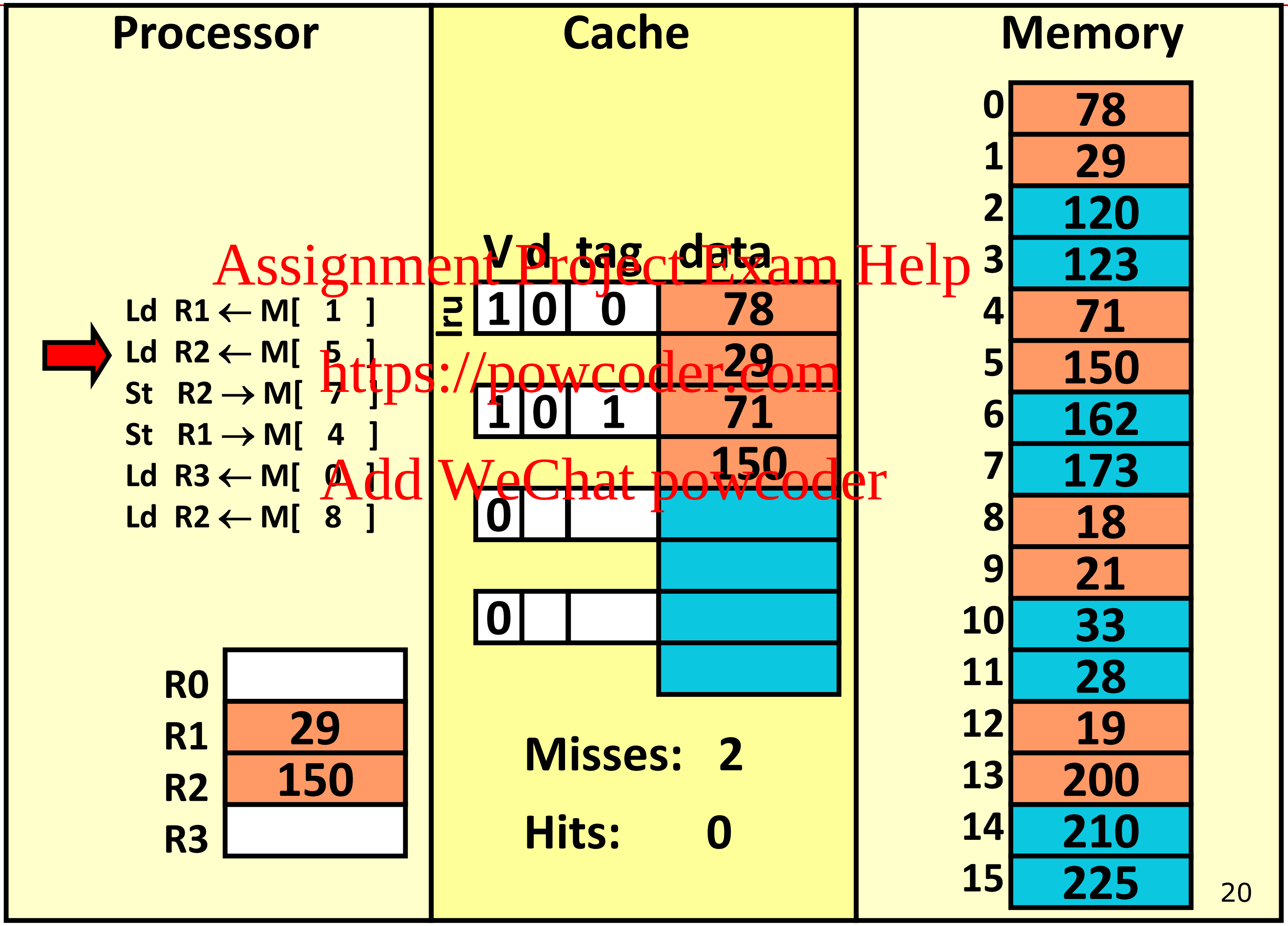
	Memory	Tag	Index	Block_offset
0	78	00	0	0
1	29	00	0	1
2	120	00	1	0
3	123	00	1	1
4	71	01	0	0
5	150	01	0	1
6	162	01	1	0
7	173	01	1	1
8	18	10	0	0
9	21	10	0	1
10	33	10	1	0
11	28	10	1	1
12	19	11	0	0
13	200	11	0	1
14	210	11	1	0
15	225	11	1	1

Set-associative cache (REF 2)



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

Set-associative cache (REF 2)



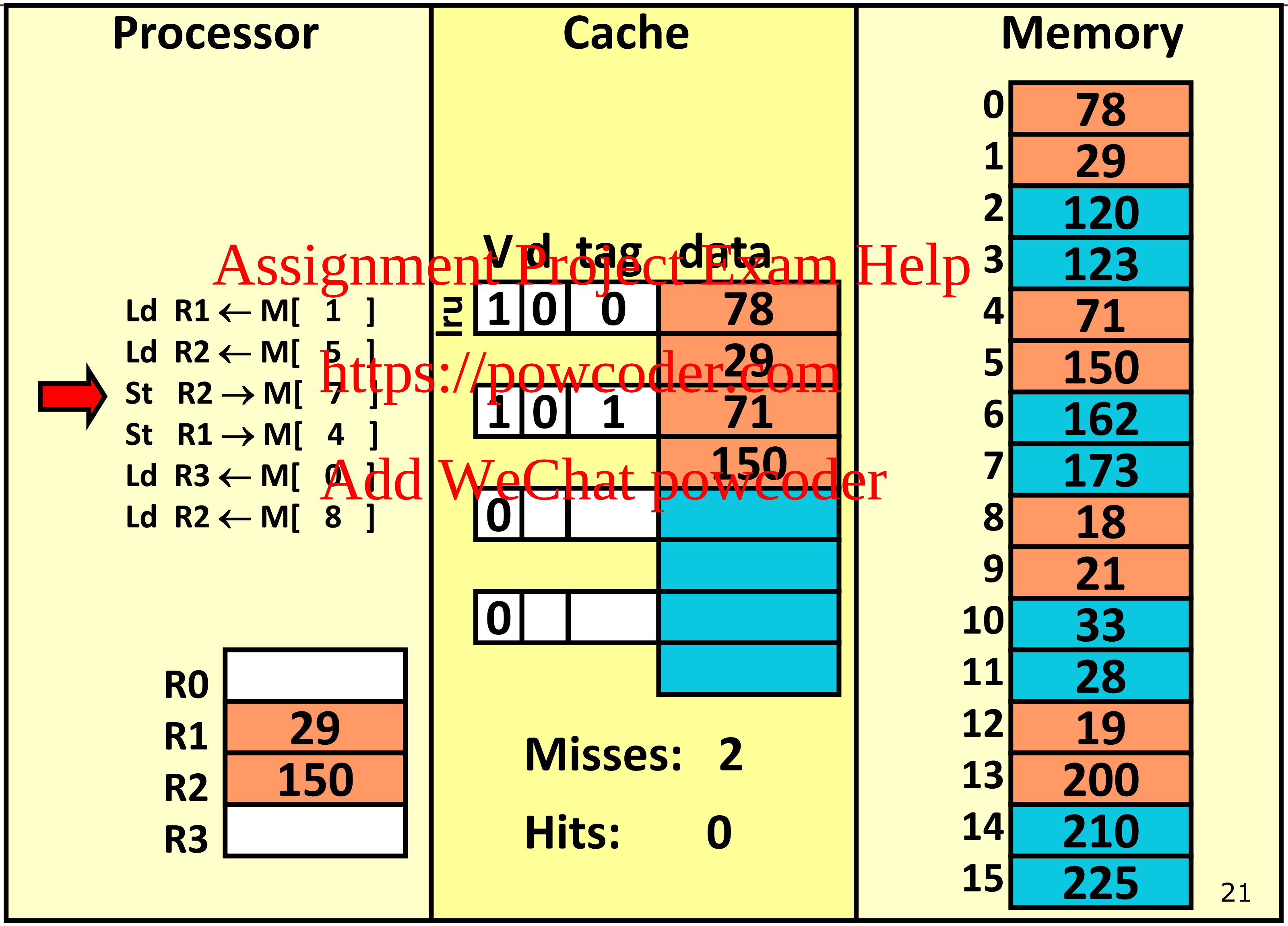
```
   Ld  R1 ← M[  1  ]
➡  Ld  R2 ← M[  5  ]
   St  R2 → M[  7  ]
   St  R1 → M[  4  ]
   Ld  R3 ← M[  0  ]
   Ld  R2 ← M[  8  ]
```

Register	Value
R0	
R1	29
R2	150
R3	

V	d	tag	data
1	0	0	78
			29
1	0	1	71
			150
0			
0			

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

Set-associative cache (REF 3)



```
Ld  R1 ← M[  1  ]
Ld  R2 ← M[  5  ]
St  R2 → M[  7  ]
St  R1 → M[  4  ]
Ld  R3 ← M[  0  ]
Ld  R2 ← M[  8  ]
```

R0	
R1	29
R2	150
R3	

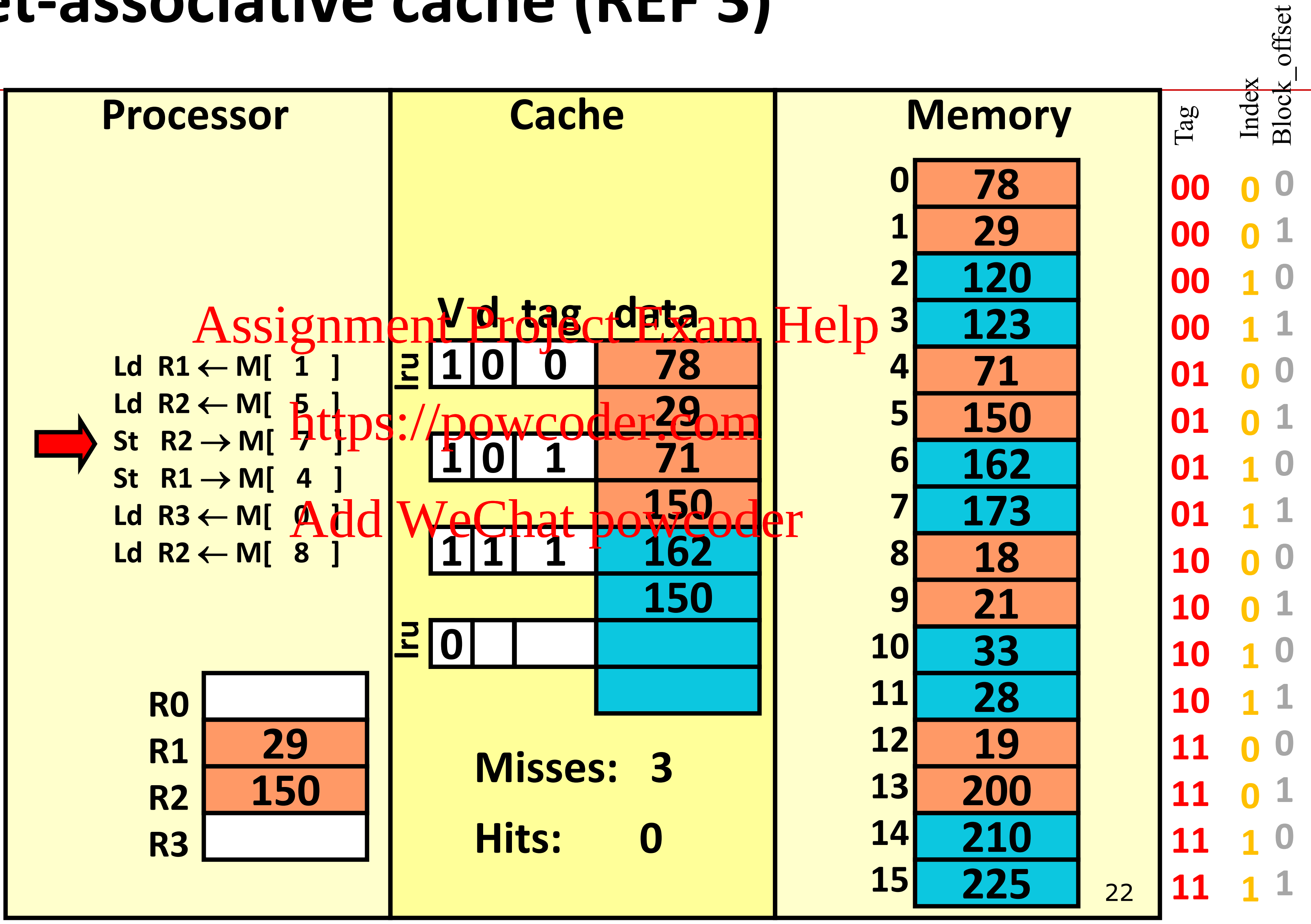
V	d	tag	data
1	0	0	78
			29
1	0	1	71
			150
0			
0			

Misses: 2

Hits: 0

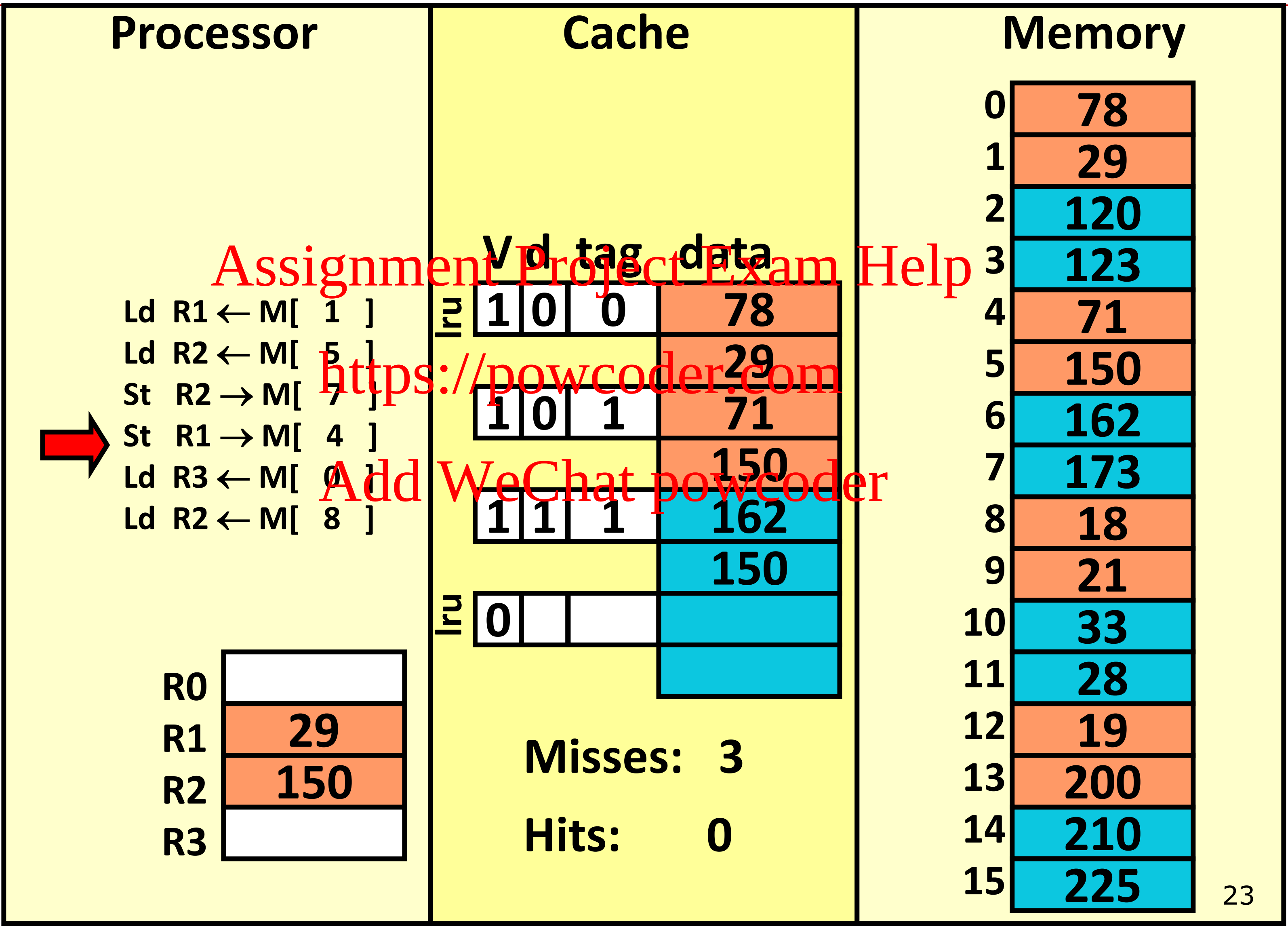
	Memory	Tag	Index	Block_offset
0	78	00	0	0
1	29	00	0	1
2	120	00	1	0
3	123	00	1	1
4	71	01	0	0
5	150	01	0	1
6	162	01	1	0
7	173	01	1	1
8	18	10	0	0
9	21	10	0	1
10	33	10	1	0
11	28	10	1	1
12	19	11	0	0
13	200	11	0	1
14	210	11	1	0
15	225	11	1	1

Set-associative cache (REF 3)



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

Set-associative cache (REF 4)



```
    Ld  R1 ← M[  1  ]
    Ld  R2 ← M[  5  ]
    St  R2 → M[  7  ]
⇒   St  R1 → M[  4  ]
    Ld  R3 ← M[  0  ]
    Ld  R2 ← M[  8  ]
```

R0	
R1	29
R2	150
R3	

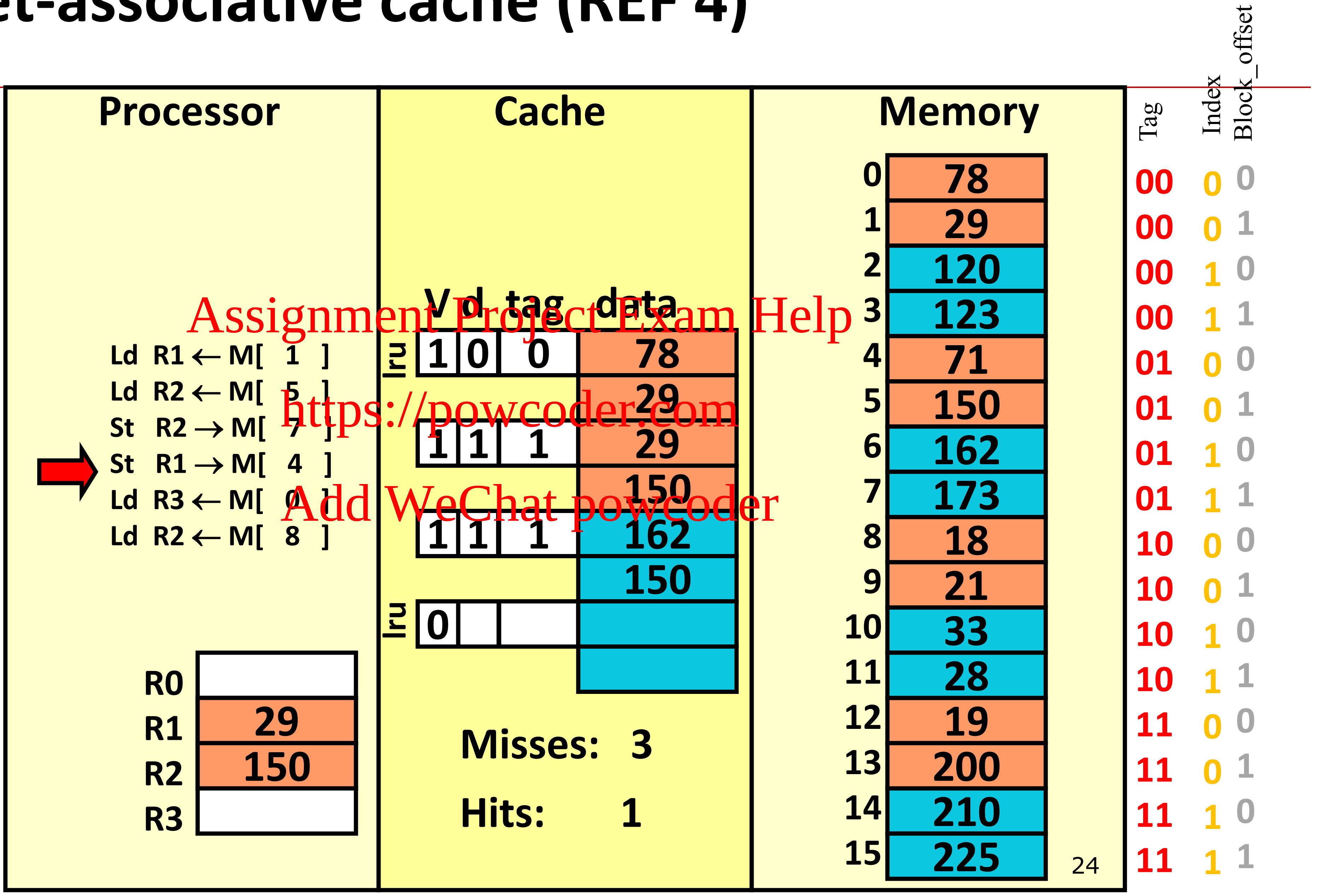
	V	d	tag	data
lru	1	0	0	78
				29
	1	0	1	71
				150
	1	1	1	162
				150
lru	0			

Misses: 3

Hits: 0

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

Set-associative cache (REF 4)



Processor

```
    Ld  R1 ← M[  1  ]
    Ld  R2 ← M[  5  ]
    St  R2 → M[  7  ]
➡   St  R1 → M[  4  ]
    Ld  R3 ← M[  0  ]
    Ld  R2 ← M[  8  ]
```

R0	
R1	29
R2	150
R3	

Cache

	V	d	tag	data
lru	1	0	0	78
				29
	1	1	1	29
				150
	1	1	1	162
				150
lru	0			

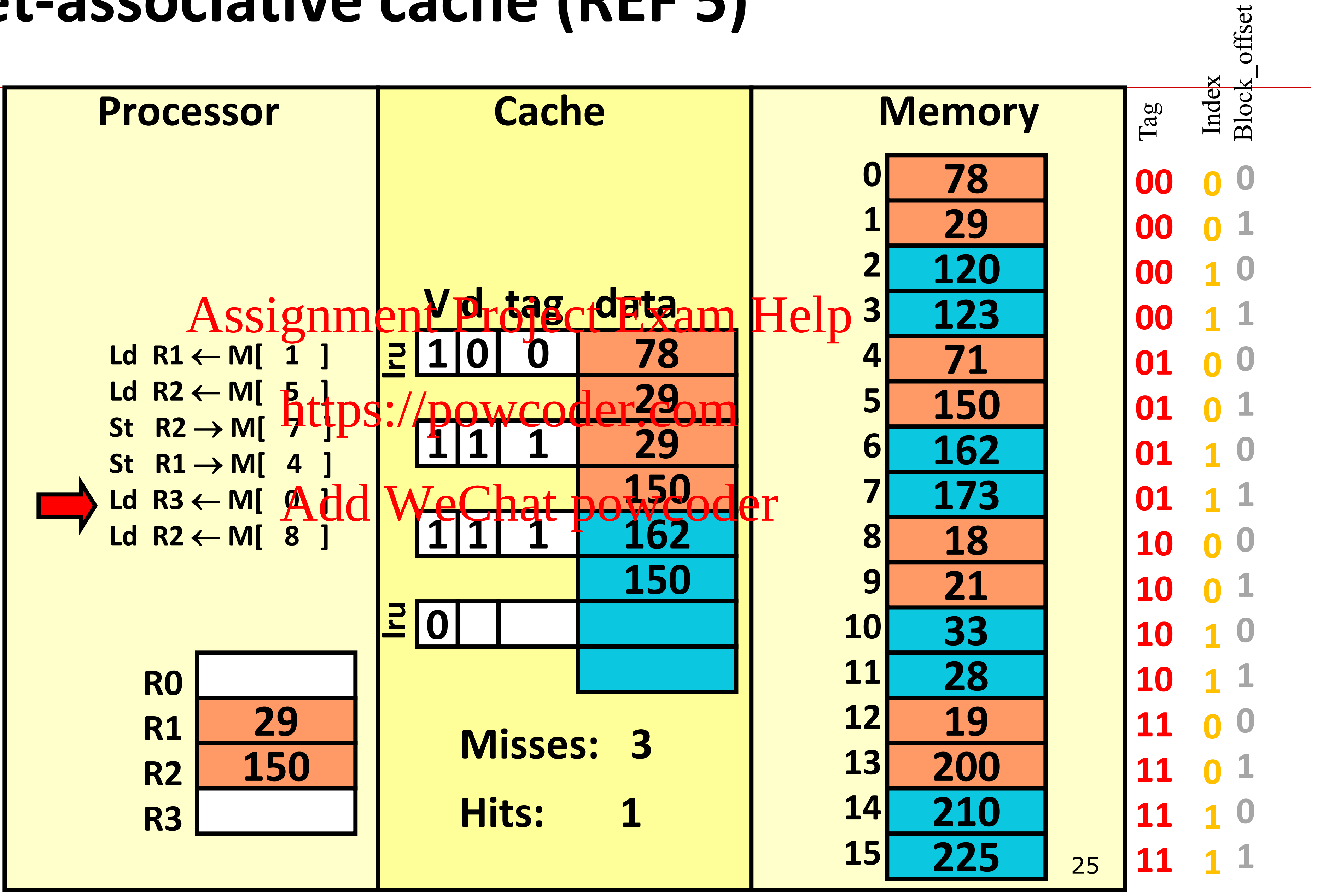
Misses: 3

Hits: 1

Memory

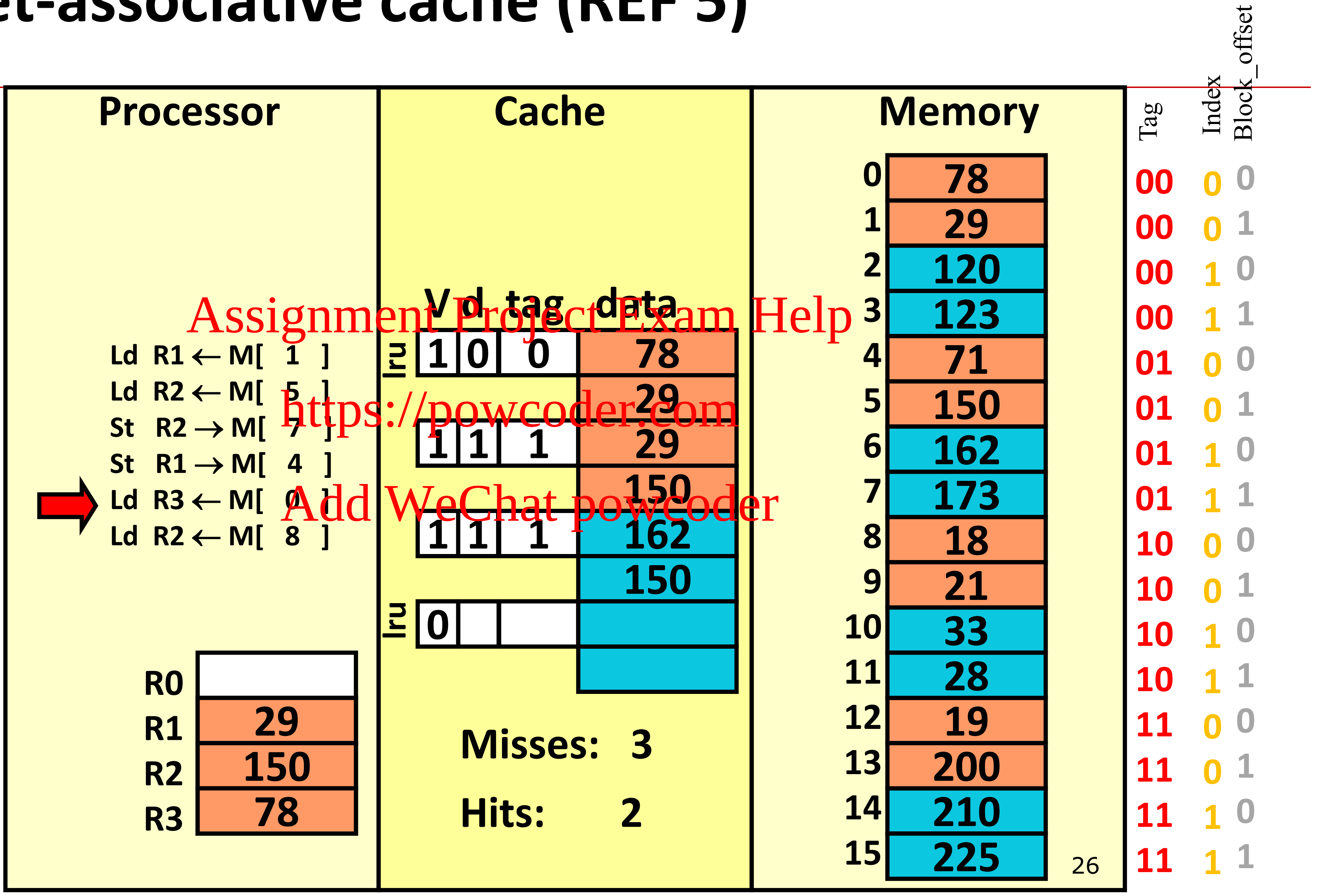
		Tag	Index	Block_offset
0	78	00	0	0
1	29	00	0	1
2	120	00	1	0
3	123	00	1	1
4	71	01	0	0
5	150	01	0	1
6	162	01	1	0
7	173	01	1	1
8	18	10	0	0
9	21	10	0	1
10	33	10	1	0
11	28	10	1	1
12	19	11	0	0
13	200	11	0	1
14	210	11	1	0
15	225	11	1	1

Set-associative cache (REF 5)

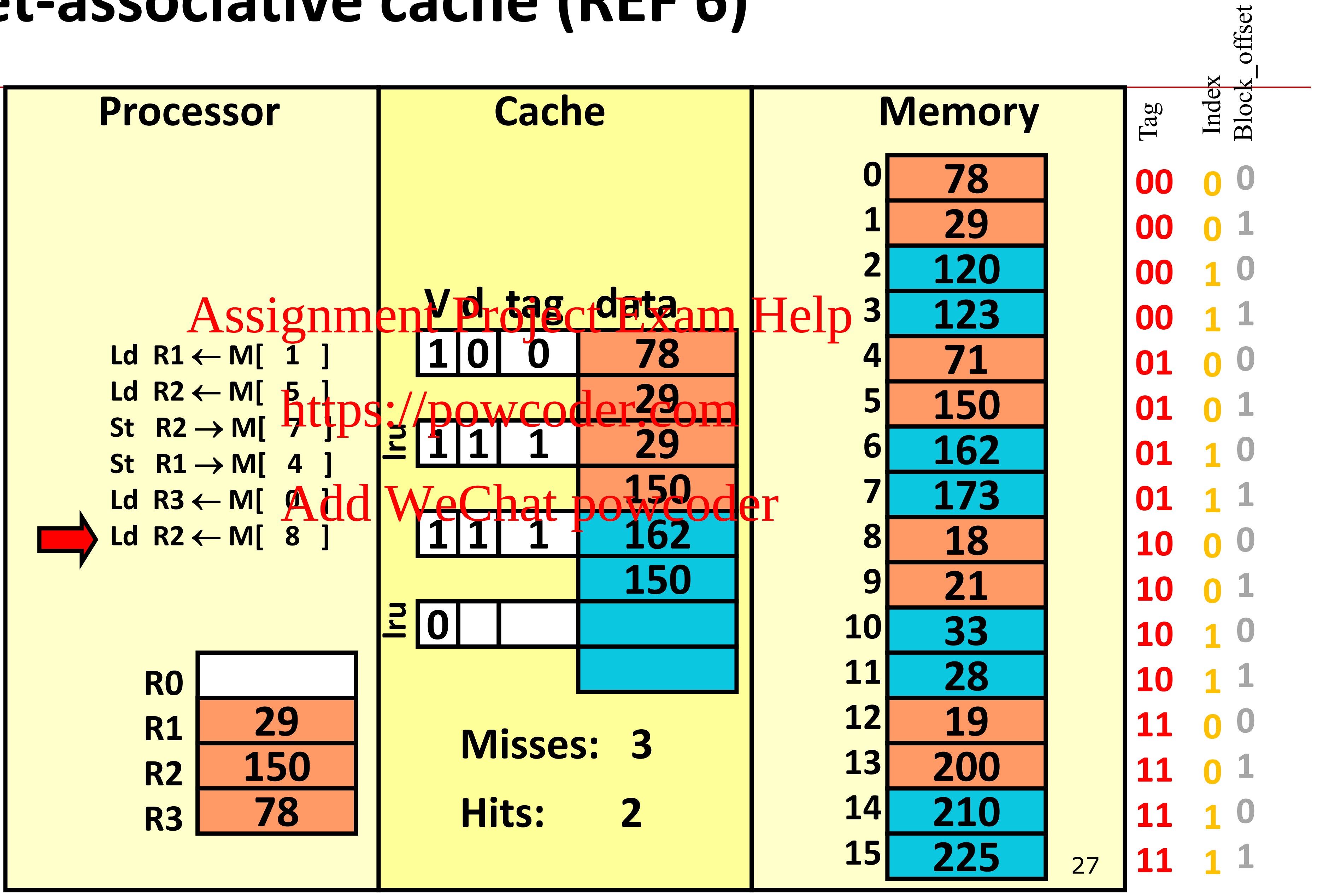


Address	Memory	Tag	Index	Block_offset
0	78	00	0	0
1	29	00	0	1
2	120	00	1	0
3	123	00	1	1
4	71	01	0	0
5	150	01	0	1
6	162	01	1	0
7	173	01	1	1
8	18	10	0	0
9	21	10	0	1
10	33	10	1	0
11	28	10	1	1
12	19	11	0	0
13	200	11	0	1
14	210	11	1	0
15	225	11	1	1

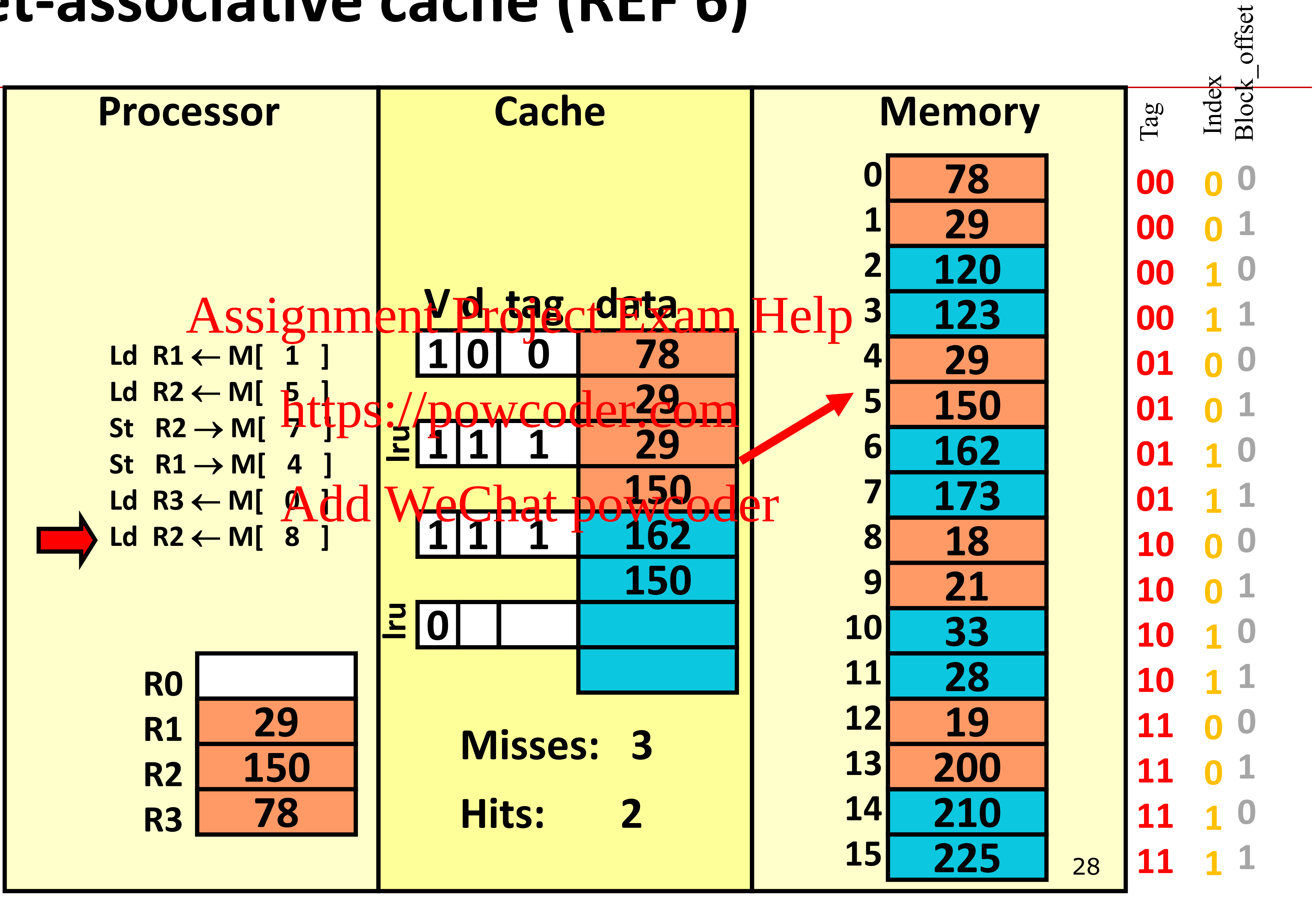
Set-associative cache (REF 5)



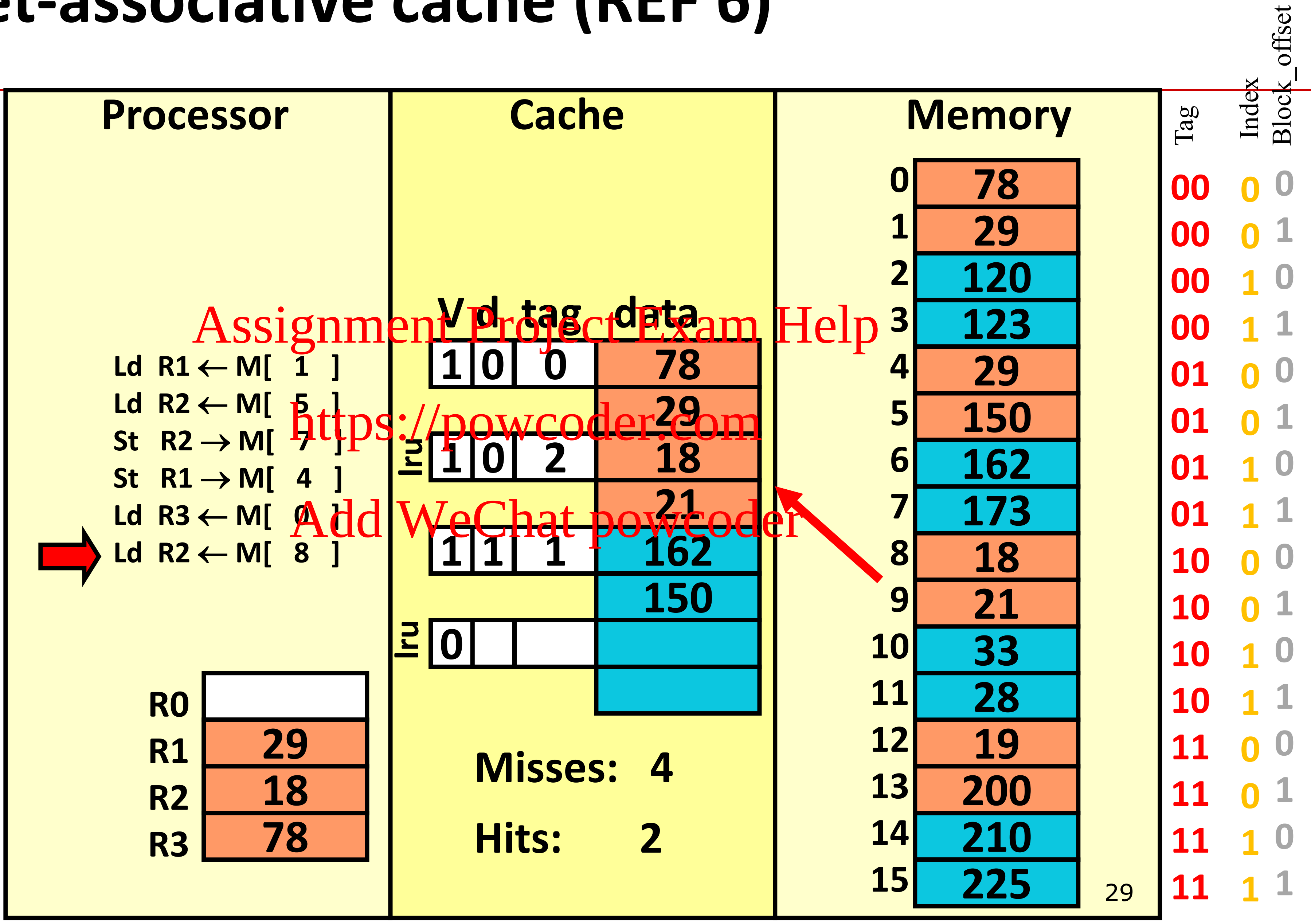
Set-associative cache (REF 6)



Set-associative cache (REF 6)



Set-associative cache (REF 6)



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

Reasons for cache misses a.k.a. The 3C's of Cache Misses

Compulsory miss

First reference to any block will always miss

Also sometimes called a **"cold start"** miss

Capacity miss

Cache is too small to hold all the data

Would have had a hit with an infinite cache

Conflict miss

Would have had a hit with a fully associative cache

Classifying Cache Misses

Can we classify a cache miss into one of the following?

- **Compulsory miss**
- **Capacity miss**
- **Conflict miss**

Yes! Simulate three different caches

Simulate with a cache of unlimited size (cache size = memory size)

– Any misses must be **compulsory misses**

Simulate again with a fully associative cache of the intended size

- Any new misses must be **capacity misses**

Simulate a third time, with the actual intended cache

- Any new misses must be **conflict misses**

Fixing cache misses

Compulsory misses

- First reference to an address
- No way to completely avoid these
- Reduce by **increasing block size** **(spatial locality)**
- This reduces the total number of blocks

Capacity misses

- Would have a hit with a large enough cache
- Reduce by **building a bigger cache**

Conflict misses

- Would have had a hit with a fully associative cache
- Cache does not have enough associativity
- Reduce by **increasing associativity**

3 C's Sample Problem

Consider a cache with the following configuration:

write-allocate

Cache size = 64 bytes

Block size = 16 bytes

2-way associative.

16-bit byte-addressable ISA. (address size is 16 bits).

LRU replacement policy.

Assume the cache is empty at the start.

For the following memory accesses, indicate whether the reference is a hit or miss, and the type of a miss (compulsory, conflict, capacity)

3 C’s Practice Problem – Address sequence

Address

0x00

0x14

0x27

0x08

0x38

0x4A

0x18

0x27

0x0F

0x40

3 C's Practice Problem – Simulate infinite cache

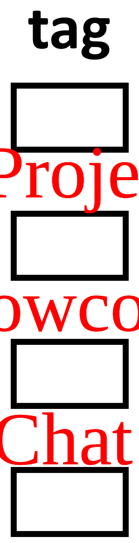
Address

Tag	Block_offset
0x0	0
0x1	4
0x2	7
0x0	8
0x3	8
0x4	A
0x1	8
0x2	7
0x0	F
0x4	0

3 C’s Practice Problem – Simulate fully associative cache

Address

Tag	Block_offset
0x0	0
0x1	4
0x2	7
0x0	8
0x3	8
0x4	A
0x1	8
0x2	7
0x0	F
0x4	0



3 C's Practice Problem – Simulate given set associative cache

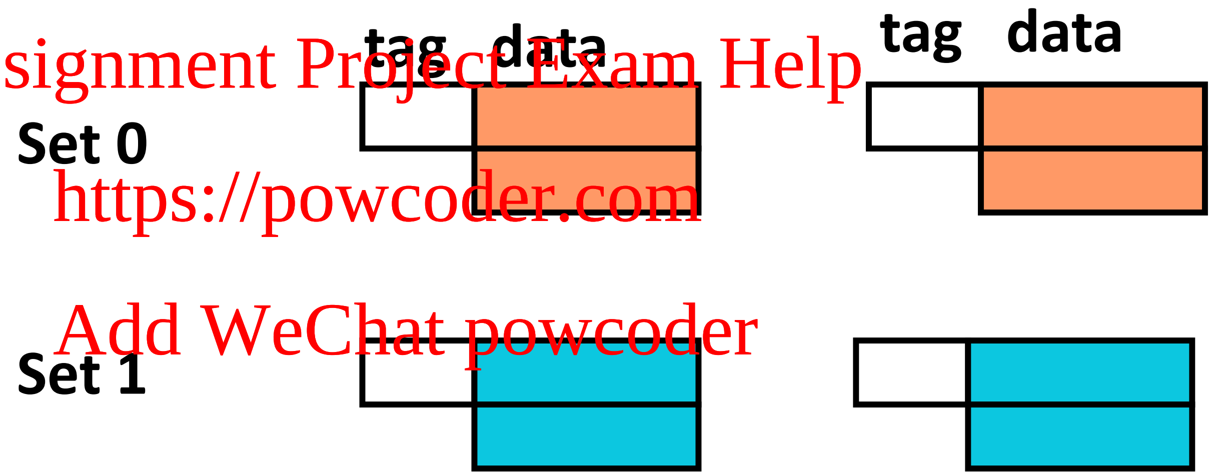
Address

Tag

Set index

0x0	0
0x1	4
0x2	7
0x0	8
0x3	8
0x4	A
0x1	8
0x2	7
0x0	F
0x4	0

Block_offset



3 C's Practice Problem – 3 C's

Address	Infinite	FA	SA	3Cs
0x00	M	M	M	
0x14	M	M	M	
0x27	M	M	M	
0x08	H	H	H	
0x38	M	M	M	
0x4A	M	M	M	
0x18	H	M	H	
0x27	H	M	M	
0x0F	H	M	M	
0x40	H	H	M	

3 C's Practice Problem – 3 C's

Address	Infinite	FA	SA	3Cs
0x00	M	M	M	Compulsory
0x14	M	M	M	Compulsory
0x27	M	M	M	Compulsory
0x08	H	H	H	---
0x38	M	M	M	Compulsory
0x4A	M	M	M	Compulsory
0x18	H	M	H	---
0x27	H	M	M	Capacity
0x0F	H	M	M	Capacity
0x40	H	H	M	Conflict

Cache Parameters vs. Miss Rate

Cache Size

Block Size

Associativity

Replacement policy

Questions to ask

Can block size be not power of 2?

Can number of sets be not power of 2?

Can number of ways be not power of 2?

- Can we have 3-way set associative cache?

Cache Size

Cache size in the total data (not including tag) capacity

- bigger can exploit temporal locality better
- not ALWAYS better

Too large a cache adversely affects hit & miss latency

- smaller is faster => bigger is slower

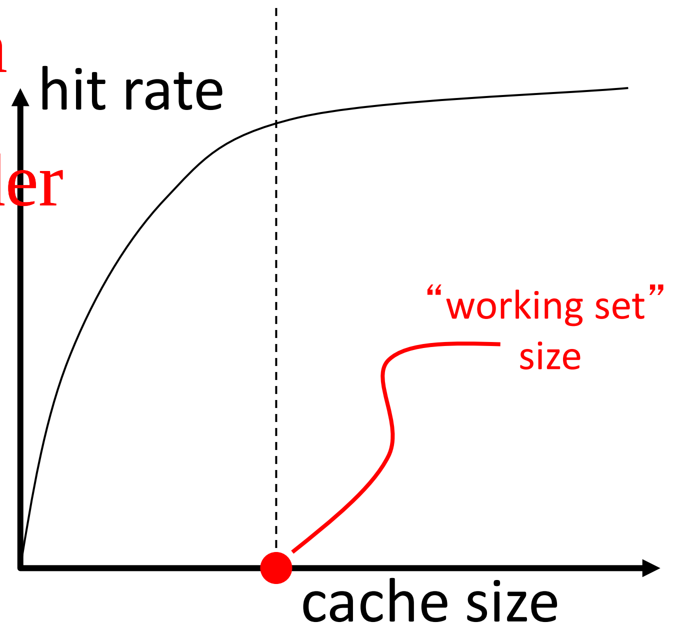
- access time may degrade critical path

Too small a cache

- doesn't exploit temporal locality well
- useful data replaced often

Working set: the whole set of data executing application references

- **Within a time interval**



Block size (also called Line size)

Block size is the data that is associated with an address tag

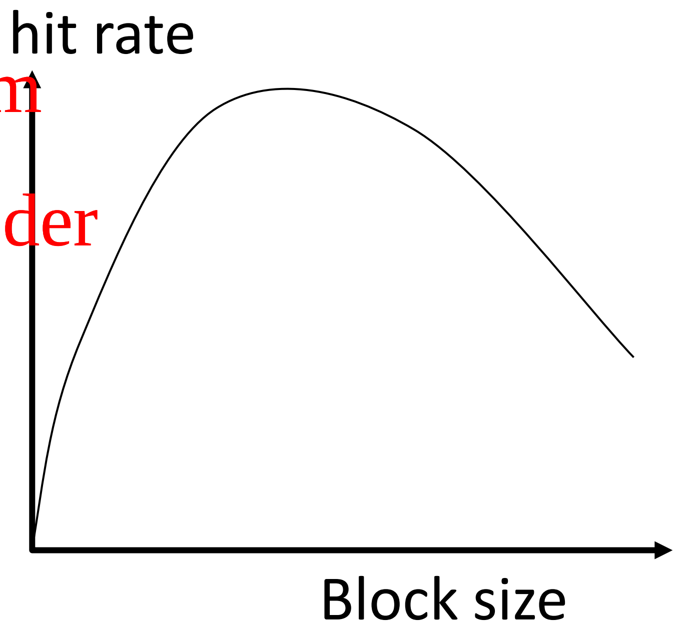
- Sub-blocking: A block divided into multiple pieces (each with V bit)
  - Can improve "write" performance

Too small blocks

- don't exploit spatial locality well
- have larger tag overhead

Too large blocks

- too few total # of blocks
  - likely-useless data transferred
  - Extra bandwidth/energy consumed



Associativity

How many blocks map to the same set (same set index)?

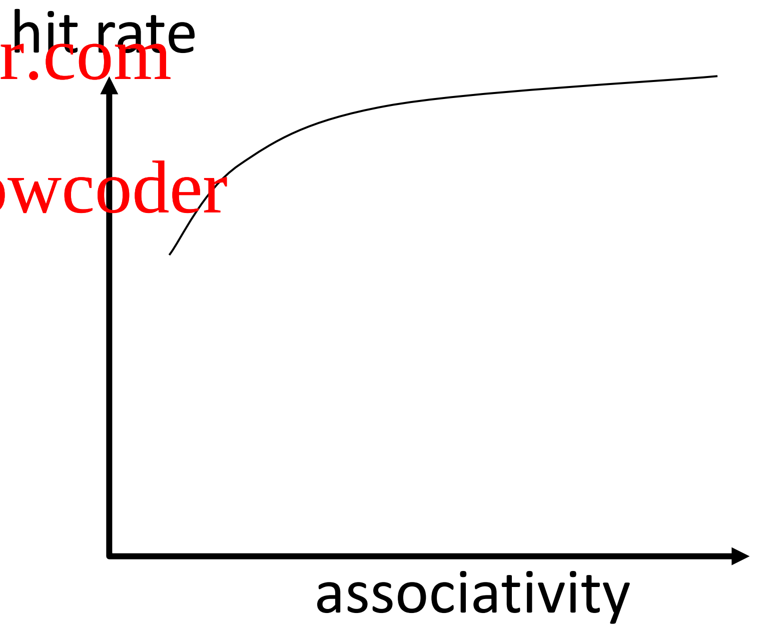
Larger associativity

- lower miss rate, less variation among programs
- diminishing returns

Smaller associativity

- lower cost
- faster hit time
  - Especially important for L1 caches

Power of 2 associativity?


hit rate
associativity