Associative Cache and Snoop Cache

Ver. 3

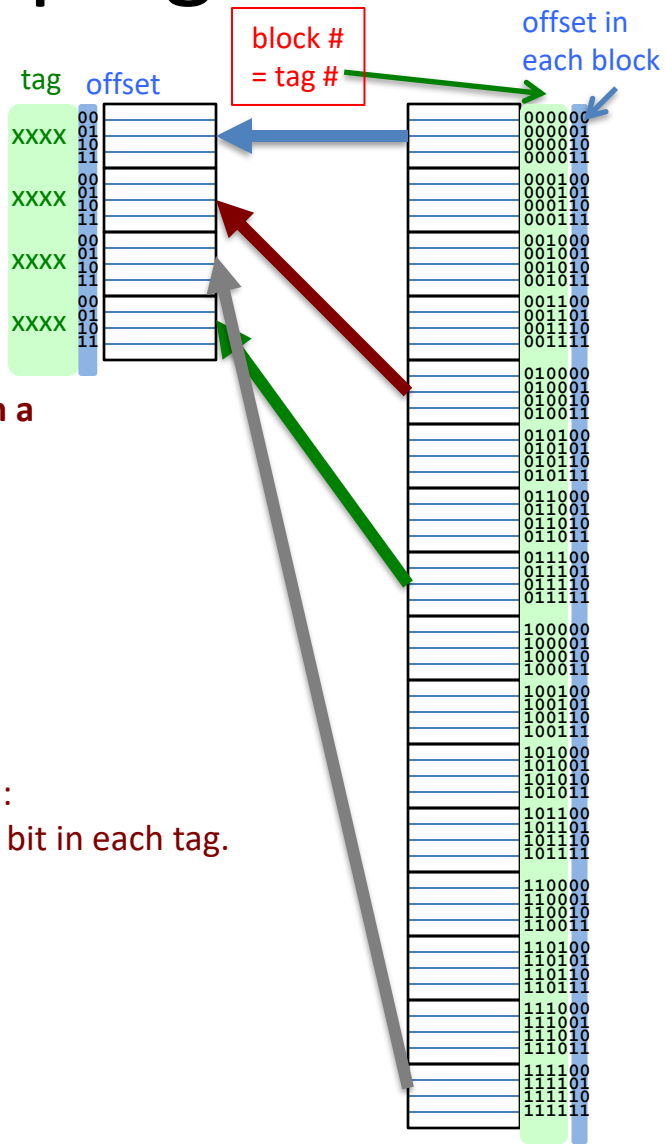
Professor: Yang Peng

Topics

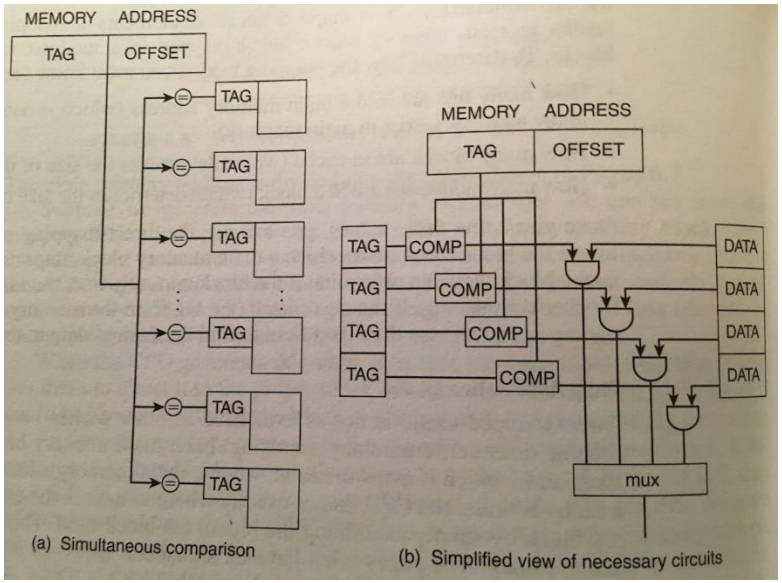
- Associative Mapping
  - Full associative
  - Set associative
- Cache Write Policies
  - Write-through, Write-back, Write-around, and Write-allocate
- Elements of Cache Design
  - Replacement algorithms
  - Multi-level caches

Associative Mapping

Tag (4)	Offset (2)



How to find the correspondence between memory and cache given a memory block?
Compare it with every single cache tag!



We need:
the valid bit in each tag.

Associative Mapping

Divide the address into tag, block and offset bits

Memory size	M bytes
Cache size	C bytes
Cache block size	B bytes

Tag bits: log(M/B)	Offset bits: log B

Memory size	64 bytes
Cache size	16 bytes
Cache block size	4 bytes

Tag (4)	Offset (2)

Compute address 47's tag and offset

47_{10} = 0x2F = 2_101111

1011 11

Tag = 11, Offset = 3

Memory size	512KB
Cache size	32KB
Cache block size	64B

Tag (13)	Offset (6)

Compute address 278407's tag and offset

278407_{10} = 0x43F87 = 2_100 0011 1111 1000 0111

1000011111110 000111

Tag = 0x10FE = 4350, Offset = 7

Memory size	4GB
Cache size	64KB
Cache block size	64B

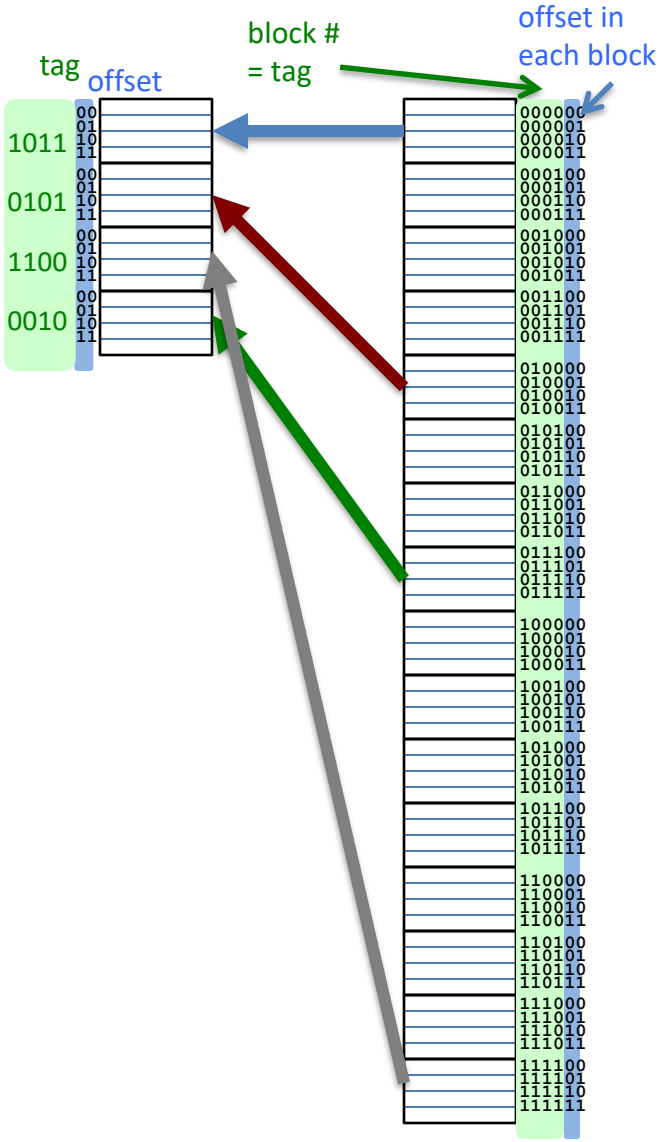
Tag (16)	Offset (6)

#tags = 4GB / 64B = 4 * 2^{30} / 2^6 = 2^{26} = 2^6

#offset = 32 – 26 = 6

Find hit/miss in an associative cache

1. Given a memory address, get its tag and offset.
2. Search the cache tag to find the block whose tag matches the memory address tag.
3. If found the matching tag (valid), then hit, otherwise miss.
4. When hit, then find the content in the cache block which is at the offset location.



Examples:

Suppose

Tag (4)	Offset (2)

Ex1. Can memory address 011111 be found in the cache?

Ex2. How about 001001?

Problem on Associative Cache

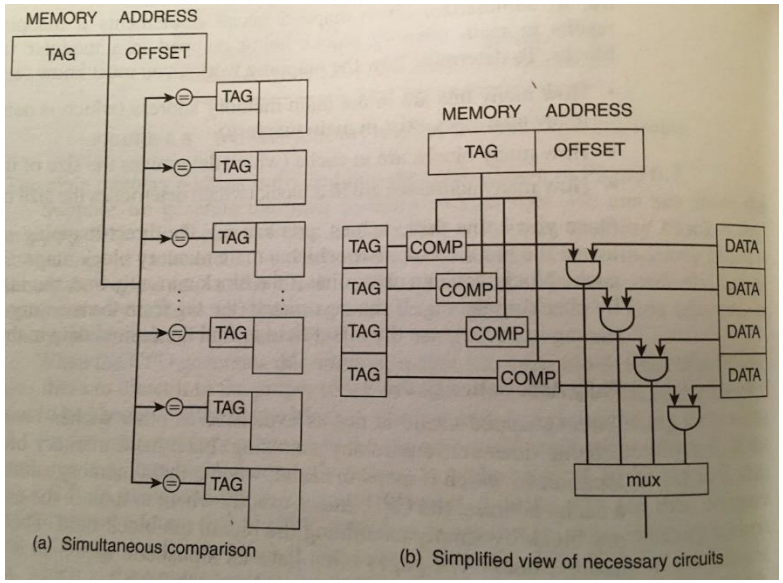
Recall tag and offset bits calculation on p2:

- Memory size 512KB.
- Associative Cache size is 32KB.
- The cache block size is 64 bytes.

➔ Number of Tags for a main memory : 512KB/64 =8K ➔ number of tag bit is $\log_2 2^{13} = 13$!

Tag (13)	Offset (6)

8196 comparators!

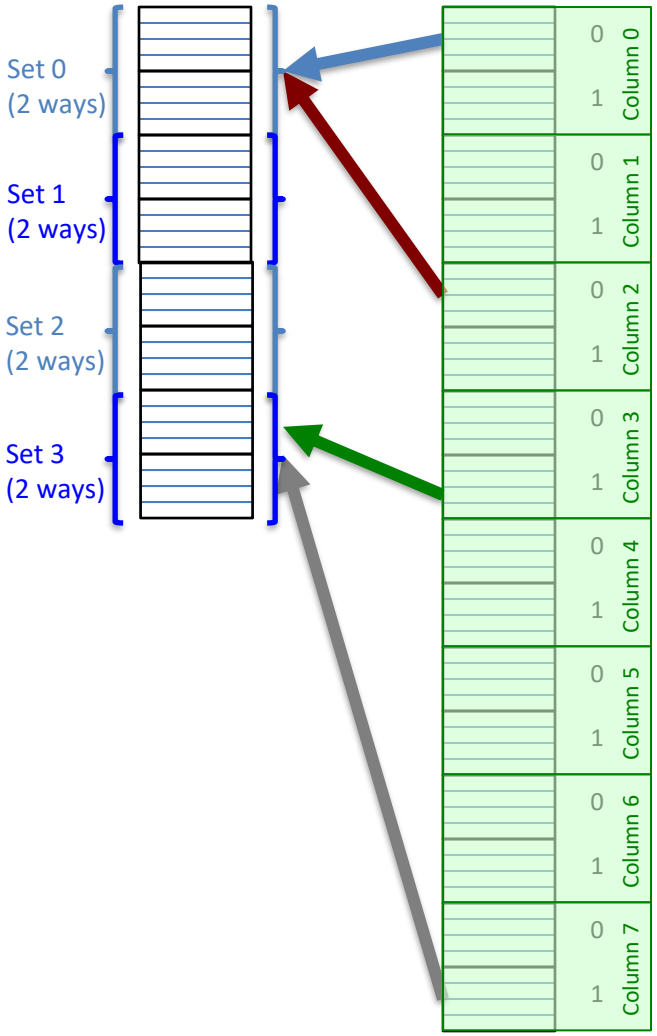


1. Should scan all the tags in a cache to find the matching tag (= the corresponding block).
   - **Complex hardware** is needed for address comparisons if main memory is large
2. When the cache is full, what should be removed (victim block) when a new block should be loaded into cache?
   - Need replacement algorithm (will cover later)

Set-Associative Cache

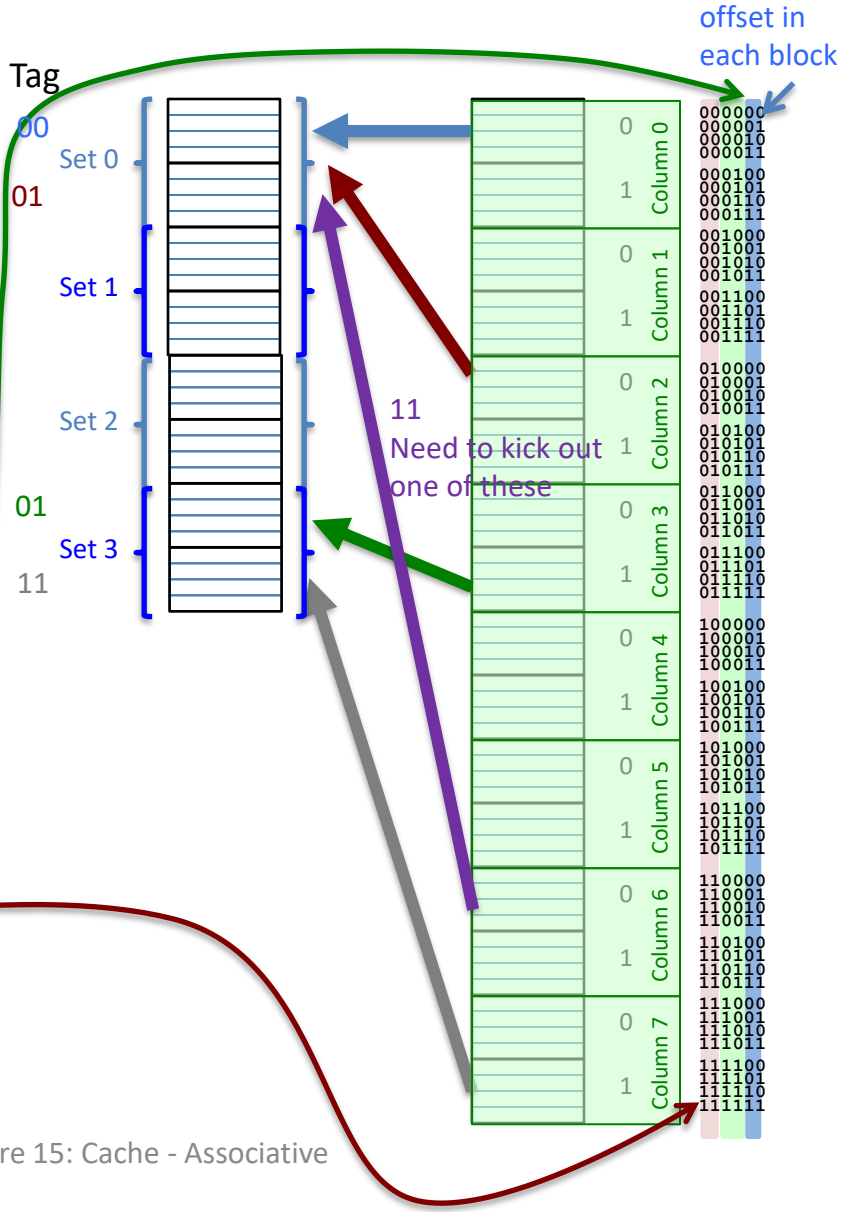
- Or called **N-way set associative** cache.
- **Combines the properties of the direct-mapped and associative cache into one system**
  - Direct mapped Cache is very restrictive
  - Associative Cache is very expensive (should search)
  - So, combine those two→ Set Associative Cache
  - Most used in modern processors (**4-way set associative cache**)
    - *The example on this page: 2-way set associative cache →→→→*
- Equivalent to a multiple-column direct mapping
    - *The example on this page: a 2-column direct mapping →→→→*
- **N-way** set associative: each set contains **N number of blocks**
    - *The example on this page: each set contains 2 blocks.*

Two-Way Set Associative Mapping



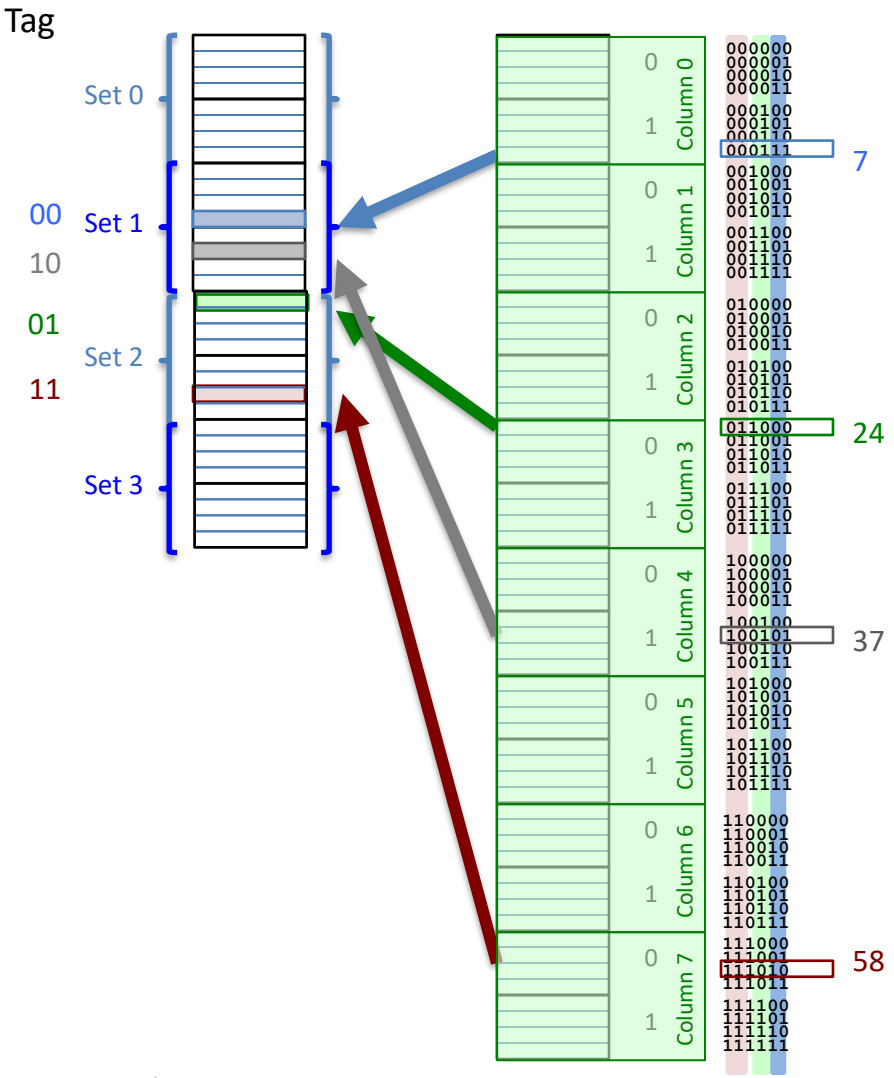
2-way Set-associative Cache

- Suppose a byte-addressable main memory system is 64 bytes capacity.
- A 2-way set associative cache has 32 bytes capacity
- A block (cache line) has 4 bytes
  - Offset bits in block: 2
- How many sets does cache have?
  - Four sets: sets 0, 1, 2, and 3
- Then, which bits in each memory address should indicate the corresponding set#?
  - Bit #3 and #2
- Then, how many bits should be allocated to represent a tag or a column #?
  - 2 bits



2-way Set-associative cache

- We want to bring data at
  - Address 7 (2_000111)
  - Address 58 (2_111010)
  - Address 24 (2_011000)
  - Address 37 (2_100101)



N-Way Set Associative Cache

Divide the address into tag, block and offset bits

Memory size	M bytes
Set associative cache size	W-way C bytes
Cache block size	B bytes

Tag bits: log(M/(C/W))	Set bits: log((C/B)/W)	Offset bits: log B

Memory size	64 bytes
Cache size 2 ways	32 bytes
Cache block size	4 bytes

Tag (2)	Set (2)	Offset (2)

#tags: 64 / (32 / 2) = 4 = 2^2
#sets: (32/4)/2 = 4 = 2^2
#offset: 4 = 2^2

Compute address 47's tag, block and offset
47_{10} = 0x2F = 2_101111
10 11 11
Tag = 2, Block = 3, Offset = 3

Memory size	512KB
Cache size 2 ways	32KB
Cache block size	64B

Tag (5)	Set (8)	Offset (6)

#tags: 512 * 2^{10} / (32 * 2^{10} / 2) = 512 / 16 = 32 = 2^5
#sets: (32K/64)/2 = 512 / 2 = 256 = 2^8
#offset: 64 = 2^6

Compute address 278407's tag, block and offset
278407_{10} = 0x43F87 = 2_100 0011 1111 1000 0111
10000 11111110 000111
Tag = 16, Block = 254, Offset = 7

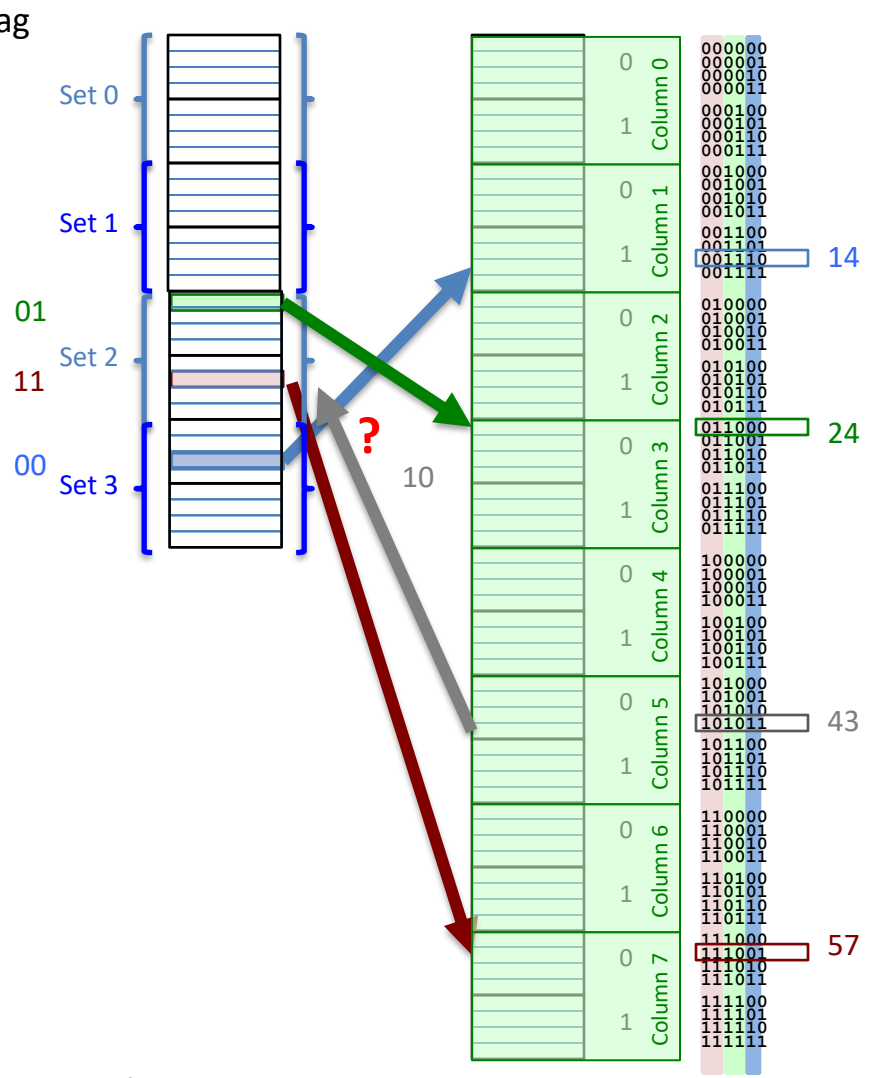
Memory size	4GB
Cache size 4 ways	64KB
Cache block size	64B

Tag (18)	Set (8)	Offset (6)

#tags = 4G / (64KB / 4) = 2^{32} / (2^{16} / 4) = 2^{32} / 2^{14} = 2^{18}
#sets = (64KB / 64) / 4) = (2^{16} / 2^6) / 4 = 2^{10} / 4 = 2^8
#offset = 64B = 2^6

Find Hit/Miss in Set Associative Cache

- Given a memory address (57 = 2_111001)
  - Get tag(11), set(10), offset(01)
  - Go to the set 2 in the cache, search for tag (11).
  - If the tag matches, then hit, retrieve data in offset(01 = 1)
- Given a memory address (14 = 2_001110)
  - Get tag(00), set(11), offset(10)
  - Go to the set 3 in the cache, search for tag (00).
  - If the tag matches, then hit, retrieve data in offset(10 = 2)
- Given a memory address (43 = 2_101011)
  - Get tag(10), set(10), offset(11)
  - Go to the set 2 in the cache, search for tag (10)
  - If the tag does not match, then miss, bring data from memory address 43.
  - **HOW!? → Cache line replacement**



Cache Block Replacement

- Replacement Algorithm
  - FIFO
  - Random
  - LRU
- LRU: the most popular
  - Need a timestamp on when the corresponding cache block was accessed.
  - Choose a victim whose timestamp is the oldest.
  - Very expensive:
    - To provide each cache block with a time stamp
    - To compare these time stamps: O(N) in N-way set associative

https://www.sciencedirect.com/topics/computer-science/set-associative-cache

LRU bit	LRU block
0	0th
1	1st

LRU bits in O(log N): 2 way: 1 bit

LRU bit0	LRU bit1	LRU block
0	0	0th
0	1	1st
1	0	2nd
1	1	3rd

LRU bits in O(log N): 4 way: 2 bits

In-Class Exercise

A certain RISC processor has an on-chip cache with the following specifications:

- 32-bit wide address and data busses
- On-chip instruction cache is 64K bytes, organized as a 4-way set associative
- Cache line (block size) = 64 bytes
- Average cache hit rate = 95%
- Instructions located in cache execute in 1 clock cycle
- For this memory design burst accesses from main memory requires an address set-up time of 5 clock cycles, and then all subsequent burst fetches from main memory require 2 clock cycles per memory fetch.

Q) What is the effective execution time in nanoseconds for this RISC processor if the clock frequency is 100 MHz?

From the lecture notes we recall that:

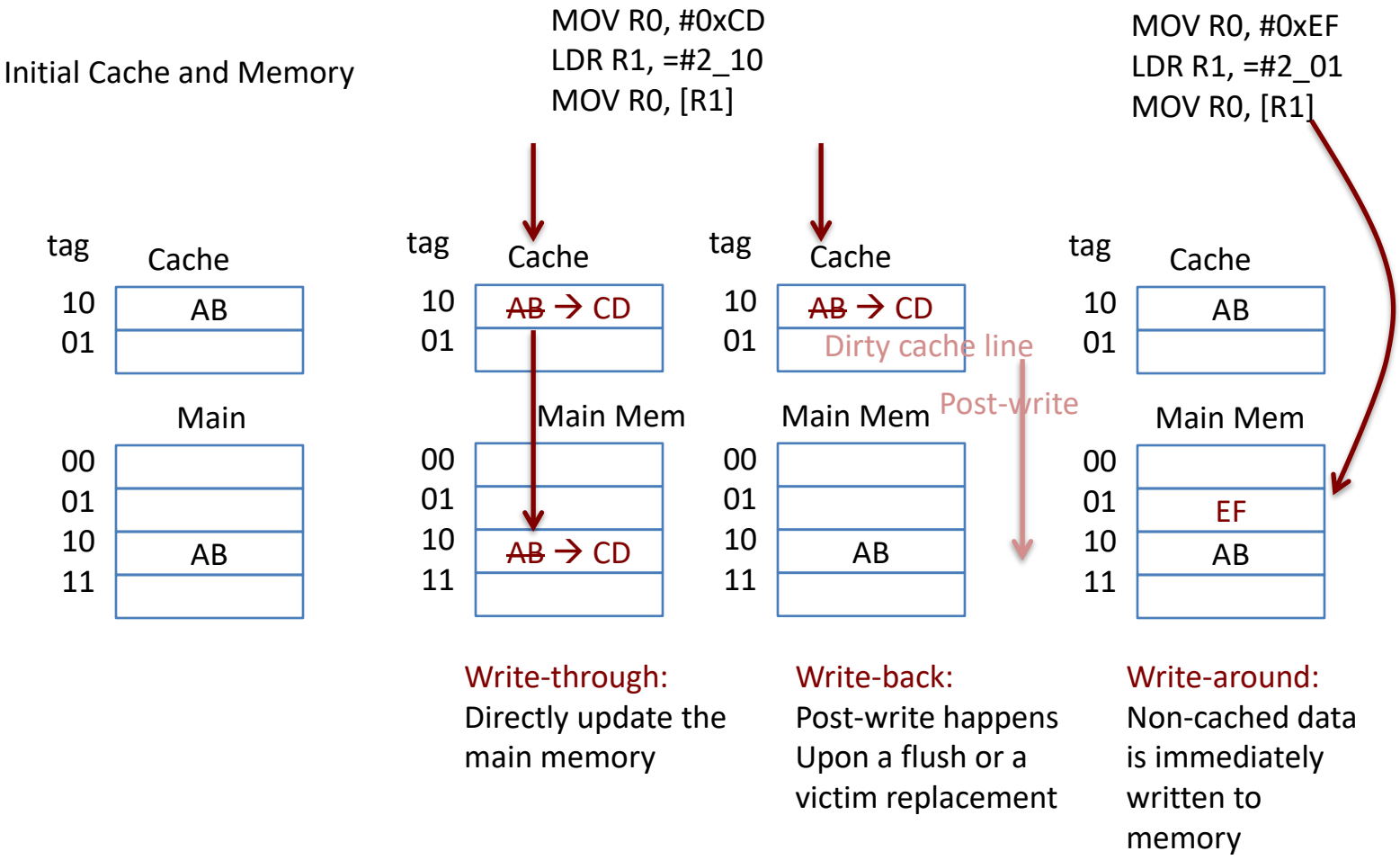
- 1 clock cycle = 1/100MHz = 10 ns
- Effective execution time = *hit rate * hit time + miss rate * miss penalty*
- *Hit rate: 0.95: Miss rate: 0.05*
- *Hit time = 1 clock cycle * 10 ns*
- *Miss time = (burst memory access + read all 64 bytes (in a block))*10ns*

  = (5 + 2 (64/4))*10 ns = 370 ns*

Note: If we have a cache miss, then the processor must refill one line in the cache. Since this is a 32-bit processor, we can read 4 bytes at once. Thus, we must read main memory 64/4 or 16 times. Since each external memory fetch requires 2 cycles, the memory read requires 16x2 = 32 clock cycles. However, we also have the overhead of setting up the address. This takes 5 clock cycles. Each clock cycle is 10 ns, so the miss penalty causes a total time delay of:

- Effective execution time = 0.95*10 + 0.05*370 = 9.5 + 18.5 **= 28 ns**

Cache Write Strategies



Elements of Cache Design

- Mapping function
  - Direct
  - Associative
  - Set Associative
- Write Policy
  - Write through – Previously L1
  - Write back – Now L1, L2, and L3
- Replacement Algorithm
  - Least recently used (LRU)
  - First in first out (FIFO)
  - Least frequently used (LFU)
  - Random
- Cache Address:
  - Logical
  - Physical – Now L1, L2, and L3
- Cache Size
- Line Size – 64B~128B
- Number of Caches
  - Single or two level – L1, L2, and L3
  - Unified or split
- Snoop cache
  - Write invalidate
  - Inclusive cache

Summary

- Two types of locality: spatial and temporal
- Cache contents include data, tags, and attribute bits (valid and LRU)
- Spatial locality demands larger block sizes (but may cause thrashing in multi cores.)
- Because miss penalty is increasing (processors getting faster than memory), modern processors use set-associative caches
- Multi-level caches used to reduce miss penalty (1st level on-chip)
- Memory system should be designed to support caches