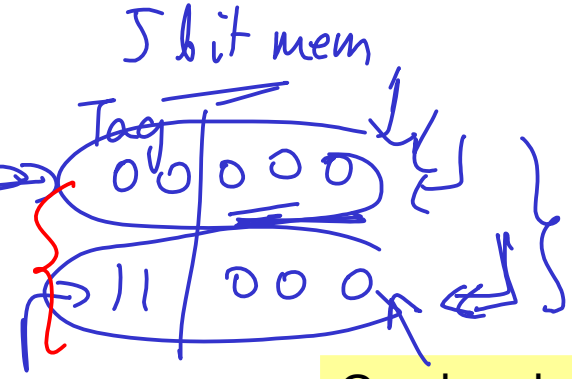
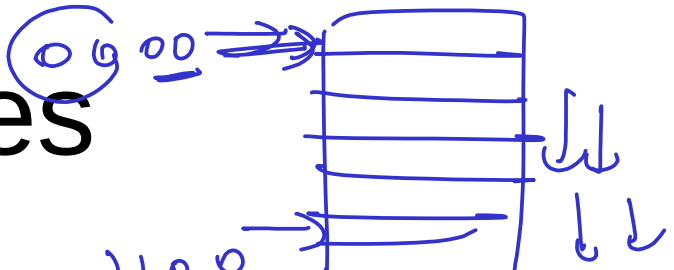
→ Block size Tag | Index | Byte offset.

Reducing cache misses

- More Flexible Placement of Blocks
- Direct mapped --> there is a direct mapping from any block address in memory to a single location in the upper level of the hierarchy
- Are there other ways of placing blocks?

Types of caches





One-way set associative

(direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Can be placed in only one block

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Within a given set, it can be placed in either of the 2 blocks

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Within a given set, it can be placed in either of the 4 blocks

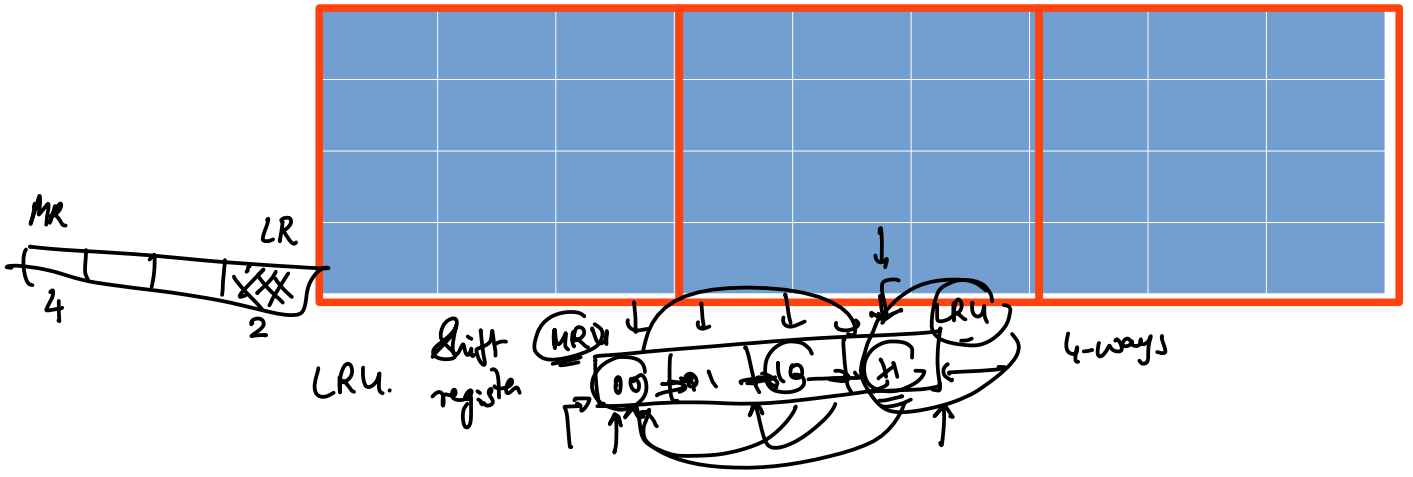
Can be placed in any block

Eight-way set associative (fully associative)

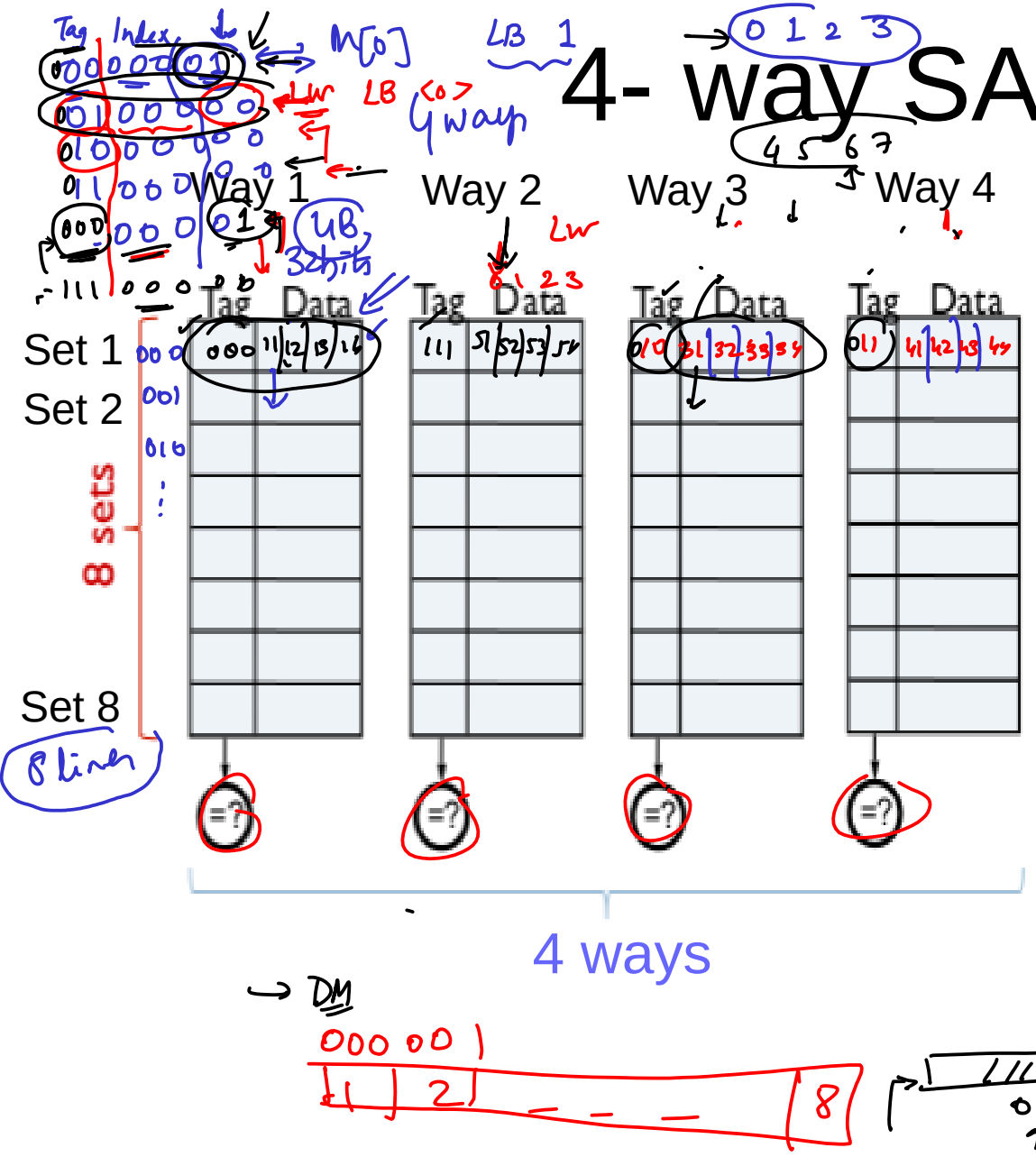
Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Set associative- Parking space analogy

- If we have 100 parking spaces we can divide them into 10 sets of 10 spaces each
- You have a choice of any place in your set
- Find your parking place within the set



4- way SA cache



Row = # Set

Columns = # Ways

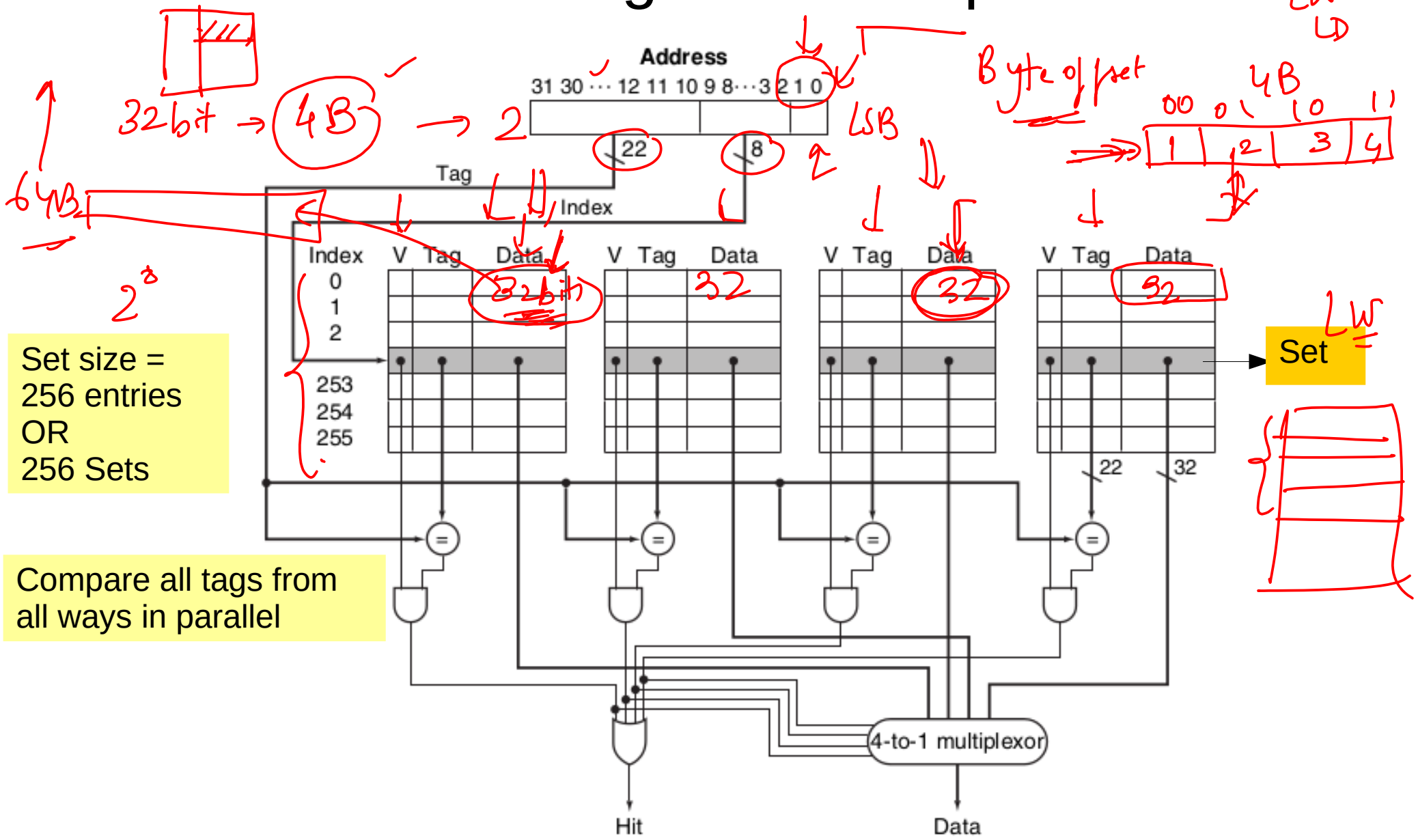
Set size = No of entries / set --> 4-way = 4 entries/set

# of tags = # of ways

Compare all tags from all ways in parallel

Similar to 4 parallel DM caches

Implementation of a 4-way 256-set SA cache addressing one word per block



Set size = 256 entries OR 256 Sets

Compare all tags from all ways in parallel

Example 3

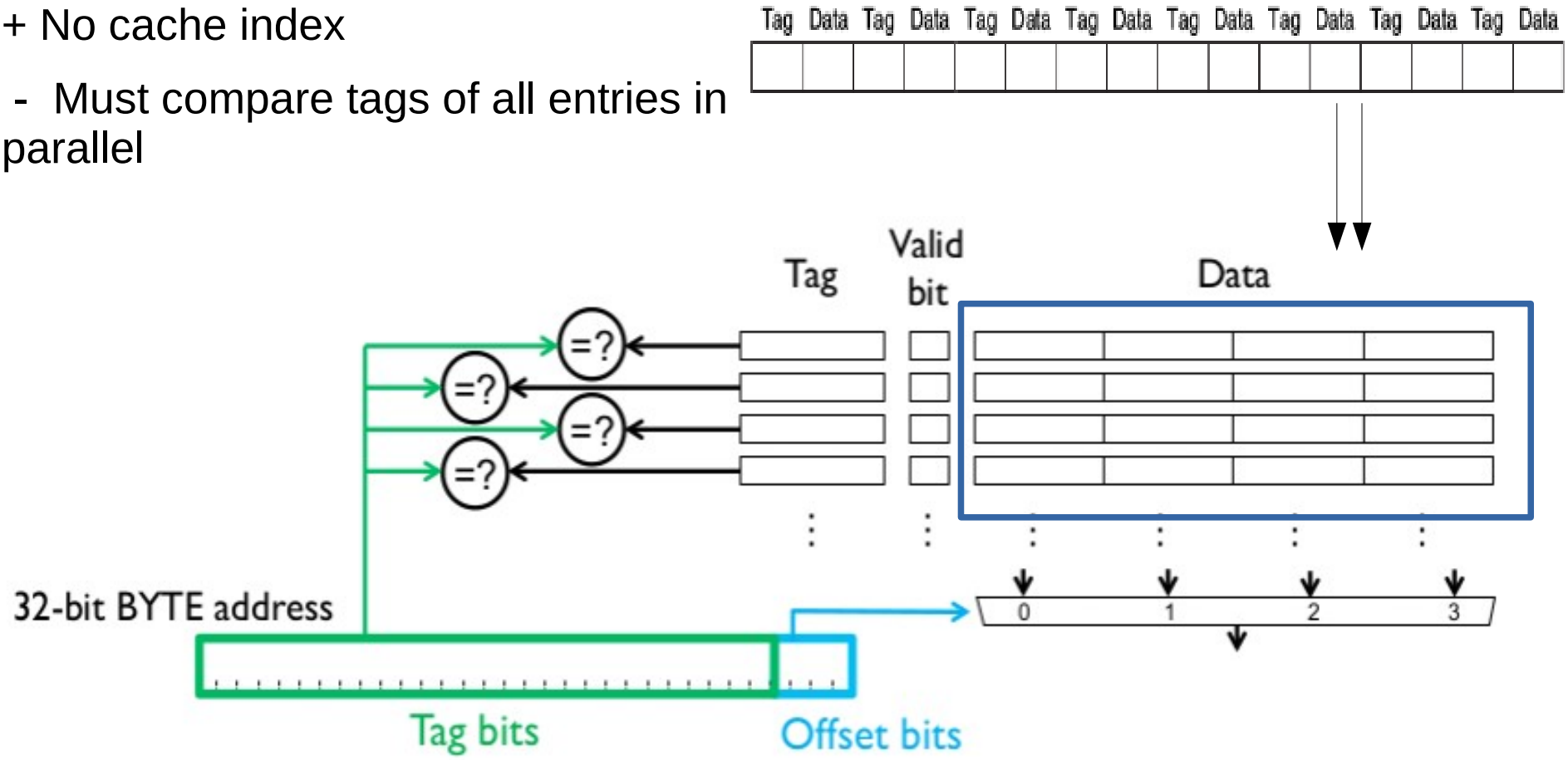
- A cache is **4-way set-associative** and has 64 KB data. Each block contains 32 bytes. The address is 32 bits wide.
  - How many sets does it have?
  - What are the sizes of the tag, index, and block offset fields?

Solution

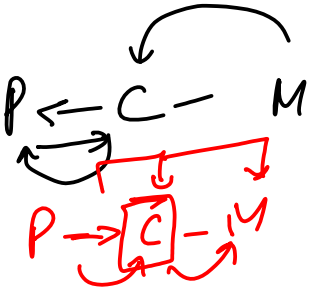
- # bits in block offset = 5 (since each block contains 2^5 bytes)
- # blocks in cache = 64×1024 / 32 = 2048 (2^11)
- # sets in cache = 2048 / 4 = 512 (2^9) sets
  - So # bits in index field = 9
- # bits in tag field = 32 – 5 - 9 = 18

Fully associative

- + Any address can be in any location
- + No cache index
- - Must compare tags of all entries in parallel



Summary of Loads/Stores



Search the cache for the address

Hit

Miss

Load/Read: Return the data to the processor from cache

Store/Write: Update the relevant word or line in the cache (write back) Update cache and the memory (write through)

Load/Read: Bring the missing line in to the cache – **<u>May need to replace a line in cache</u>** --> then return the data to the processor

Store/Write: Allocate or No allocate and update the relevant memory line in the cache/memory

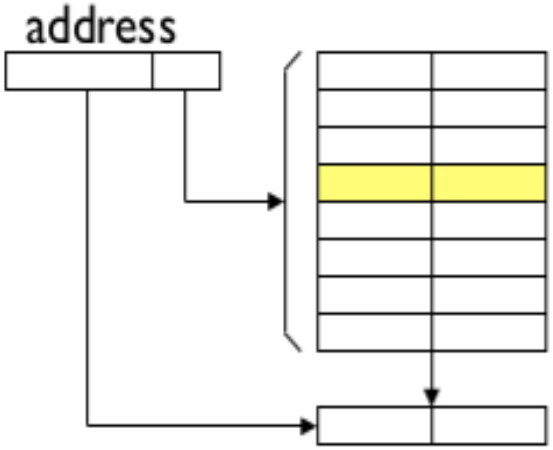
Replacement algorithms

No choice in a direct mapped cache

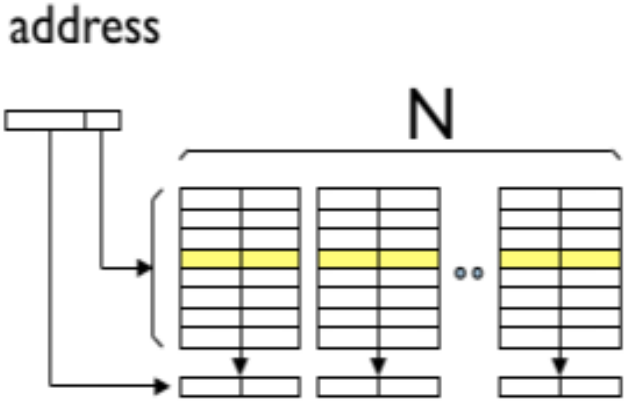
In an associative cache, which block from set should be evicted when the set becomes full?

- Random
- Least-Recently Used (LRU)
  - LRU cache state must be updated on every access
  - True implementation only feasible for small sets (2-way)
  - Maintain LRU bits to keep track of which one is the most recent
- First-In, First-Out (FIFO)
  - Used in highly associative caches

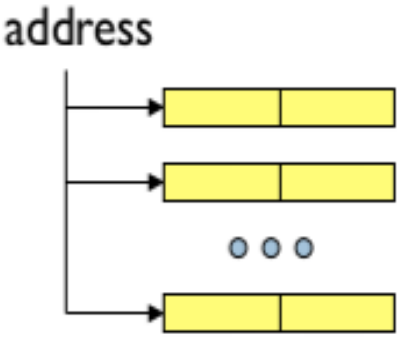
Summary



- Compare addr with only one tag
- Location X can be in exactly one cache line



- Compare addr with only N tags parallely
- Location X can be in one set, but in any of the N cache lines belonging to that set



- Compare addr with only all tags parallely
- Location X can be in any cache line

Example 4

- Assume a 4 line DM cache consisting of four one-word blocks. Find the number of misses for the following sequence:

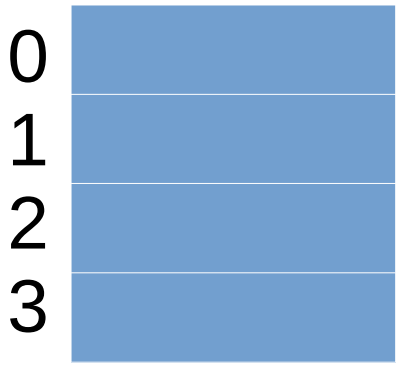
  LW $t0, 0x00

  LW $t0, 0x08

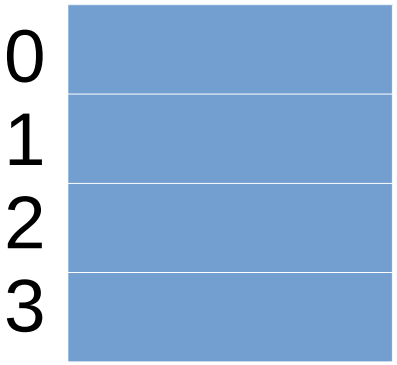
  LW $t0, 0x00

  LW $t0, 0x06

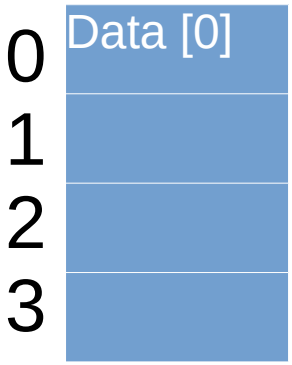
  LW $t0, 0x08

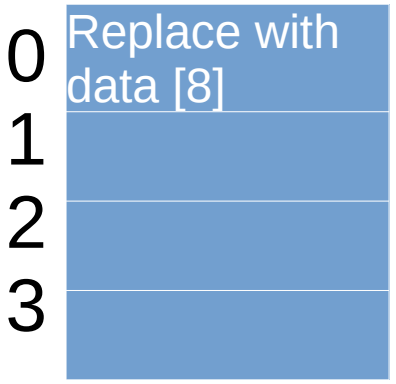


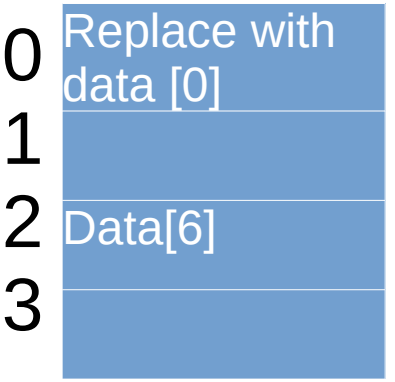
Solution

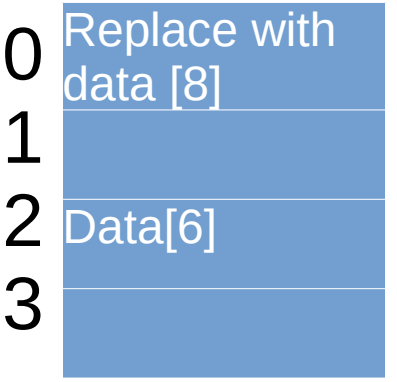


0 - (0 mod 4) – Block 0 - Miss
8 - (8 mod 4) – Block 0 - Miss
0 - (0 mod 4) – Block 0 - Miss
6 - (6 mod 4) – Block 2 - Miss
8 - (8 mod 4) – Block 0 - Miss









Example 5

- Assume a **2-way 2-line SA cache** consisting of four one-word blocks. Find the number of misses for:

  LW $t0, 0x00

  LW $t0, 0x08

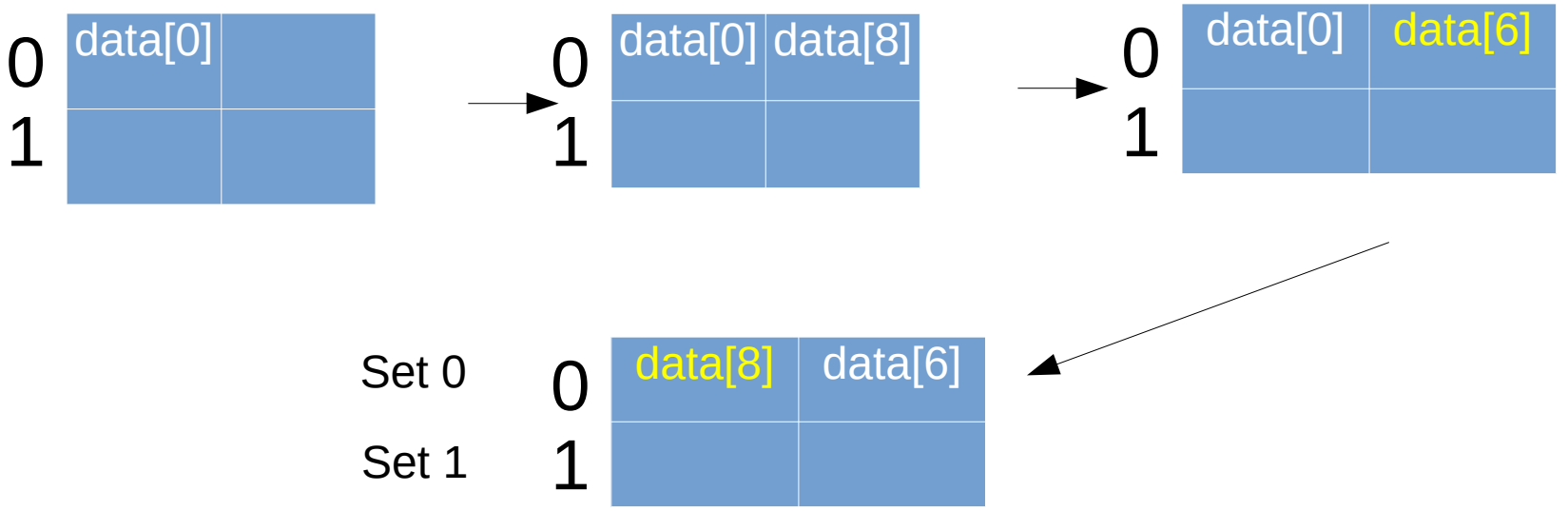
  LW $t0, 0x00

  LW $t0, 0x06

  LW $t0, 0x08

Solution

0 - (0 mod 2) – Set 0 - Miss
8 - (8 mod 2) – Set 0 - Miss
0 - (0 mod 2) – Set 0 - Hit
6 - (6 mod 2) – Set 0 - Miss
8 - (8 mod 2) – Set 0 - Miss



Example 6

- Assume a **FA cache** consisting of four one-word blocks. Find the number of misses for

  LW $t0, 0x00

  LW $t0, 0x08

  LW $t0, 0x00

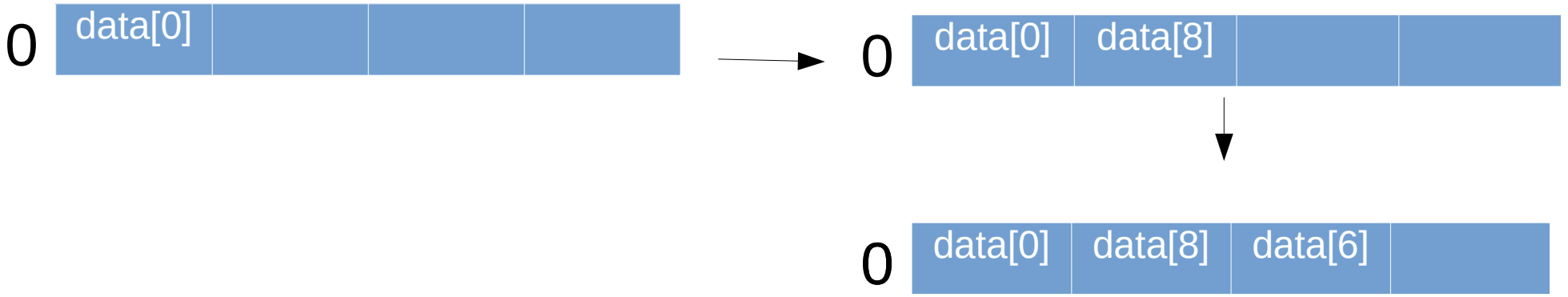
  LW $t0, 0x06

  LW $t0, 0x08

Solution

Set 0

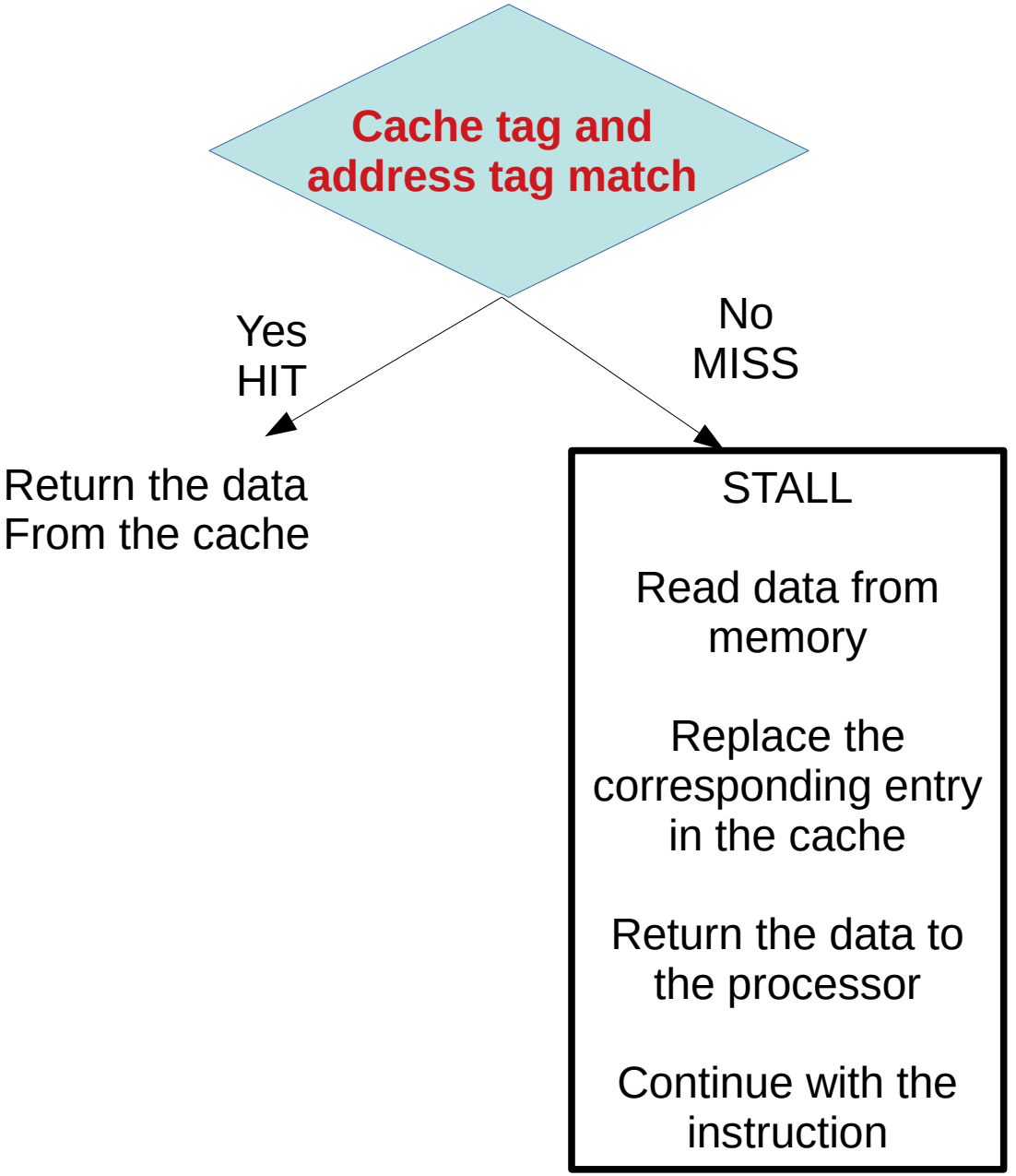
0 - Miss
8 - Miss
0 - Hit
6 - Miss
8 - Hit



Misses

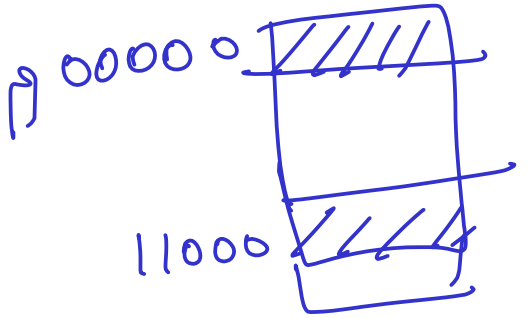
- Compulsory—The first access to a block is not in the cache, so the block must be brought into the cache.
  - **(Misses in infinite cache)**
- Capacity—If the cache cannot contain all the blocks needed during execution of a program
  - **(Misses due to size of cache)**
- Conflict/Collision—If the block-placement strategy is set associative or direct mapped, conflict misses due to too many blocks mapping to same set.
  - **(Misses due to associativity and size of cache)**

Cache read example



Handling a cache **read** miss

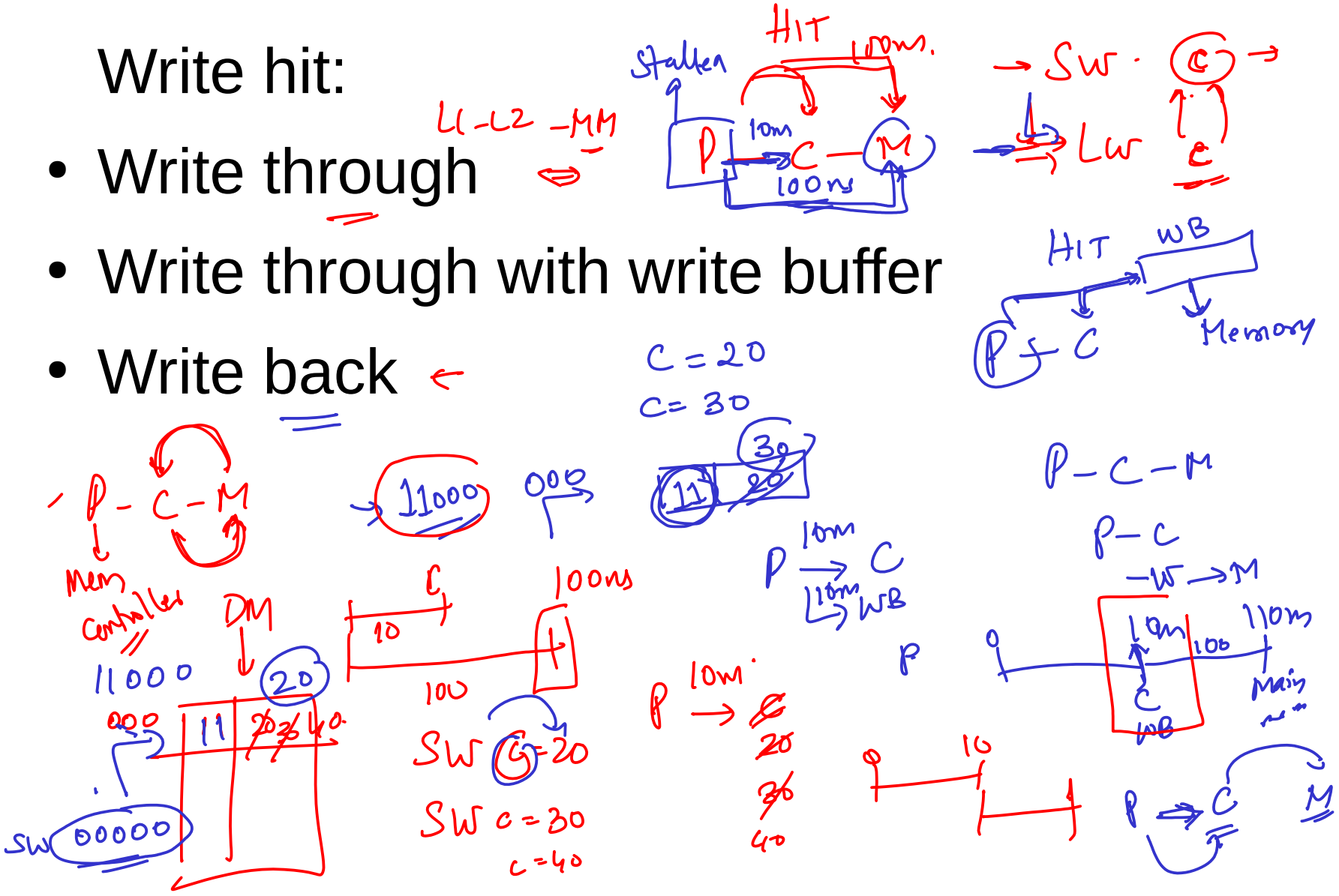
- Send the original PC value (current PC – 4) to the memory.
- Perform a read and wait for the memory to complete its access.
- Write the cache entry
  - Put the data from memory in the data portion of the entry, writing the upper bits of the address into the tag field, and turning the valid bit on.
- Restart the instruction execution at the first step, which will refetch the instruction, this time finding it in the cache.



Stores/Cache writes

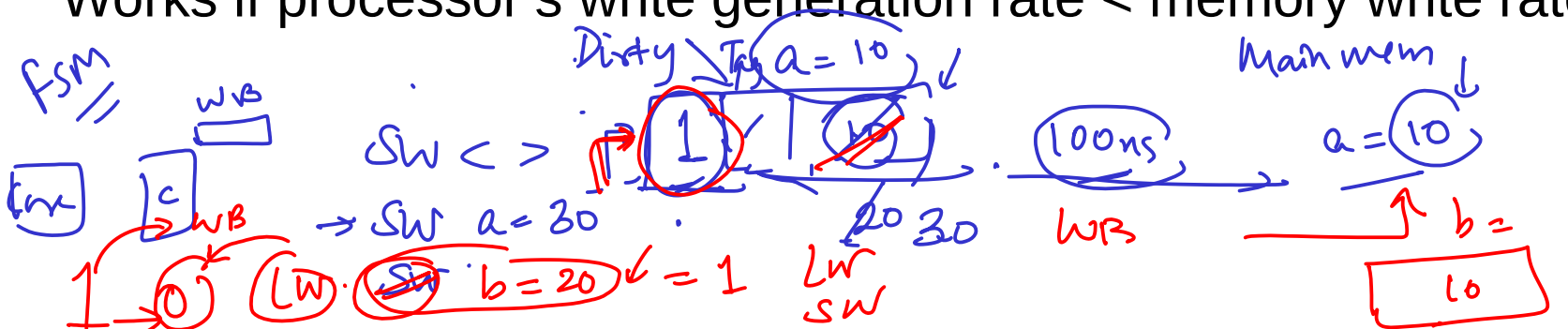
Write hit:

- Write through
- Write through with write buffer
- Write back



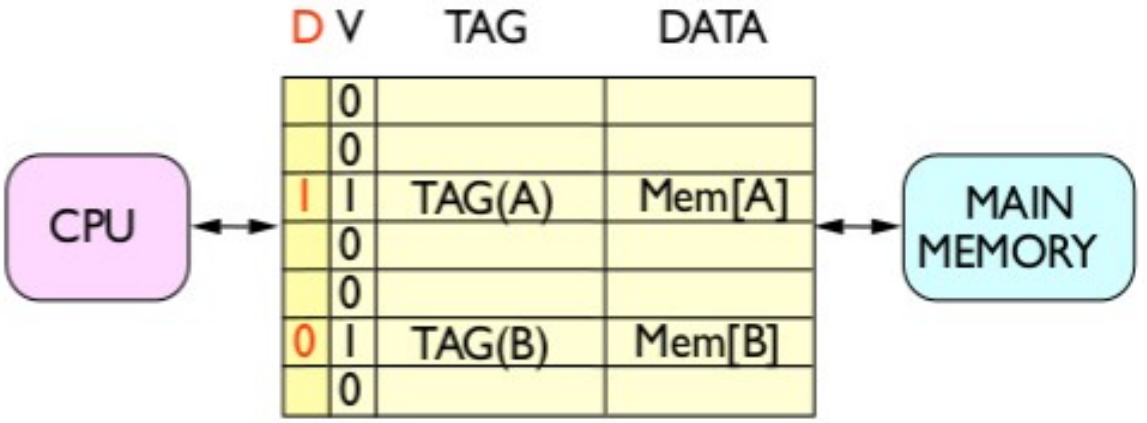
Write through

- Assume a store instruction
- Write through always updates both the cache and the next lower level of the memory hierarchy
  - - Causes delays in writing to the slower memory
  - - Performance
  - + Data is always consistent between the two
- Alternate: Use a write buffer to hold the data while data is being written to main memory or next level and then erase it
  - Processor stalls if the buffer is full
  - Works if processor's write generation rate < memory write rate



Write back

- Updating values only to the block in the cache
- Write the modified data to the lower level when the block is replaced/evicted.
  - + Performance
  - - Use an additional dirty bit to figure out which block was written to in the cache, but not yet written to in the memory



Cache write miss

- Miss Allocate: Fetch the line, and update the word in it
- Miss No allocate: Cache is not affected. Cache line is not allocated. Store goes to memory
- Combinations used
  - **Write back + write miss Allocate**
  - Write through + write miss No-allocate

Example 6

Size of Tags versus Set Associativity

- Assuming a cache of 4096 blocks, a 4-word block size, and a 32-bit address, find the total number of tag bits for caches that are Direct mapped.
- Find the total number of sets and tag bits for four-way set associative, and fully associative caches.

Solution

Direct mapped:

- 16 bytes per block --> 4 bit offset
- 4096 lines --> 12 bit index
- 32 – (4+ 12) = 16 bit tag
- No of tag bits = 16 * 4k = 64k tag bits

Solution

2-way SA:

- 16 bytes per block --> 4 bit offset
- 4096 blocks --> gets divided into 2 ways = 2048 sets --> 11 index bits
- 32 – (4+ 11) = 17 bit tag
- No of tag bits = 17 * 2 * 2048 tag bits

Solution

4-way SA:

- 16 bytes per block --> 4 bit offset
- 4096 blocks --> gets divided into 4 ways = 1024 sets --> 10 index bits (Index bits have reduced)
- 32 – (4+ 10) = 18 bit tag (Tag bits have increased)
- No of tag bits = 18 * 4 * 2048 tag bits

Solution

FA:

- 16 bytes per block --> 4 bit offset
- 4096 blocks --> 1 set of 4096 blocks --> No index bits
- 32 – (4) = 28 bit tag (Tag bits have increased)
- No of tag bits = 28* 1 * 4096 tag bits

Example 7

- Show the hits and misses and final cache contents for a fully associative cache with four-word blocks and a total size of 8 words. Assume LRU replacement.
  - Assume the sequence: Word address: 2,3,10,11,16,21,0 - addresses are in decimal

Solution

- 2: Miss: 0-3
- 3: Hit
- 10: 8-11, 0-3
- 11: Hit
- 0: Hit
- 21: 20-23, 0-3
- 1: Hit

Refer to https://inst.eecs.berkeley.edu/~cs61cl/fa08/labs/lab22.html

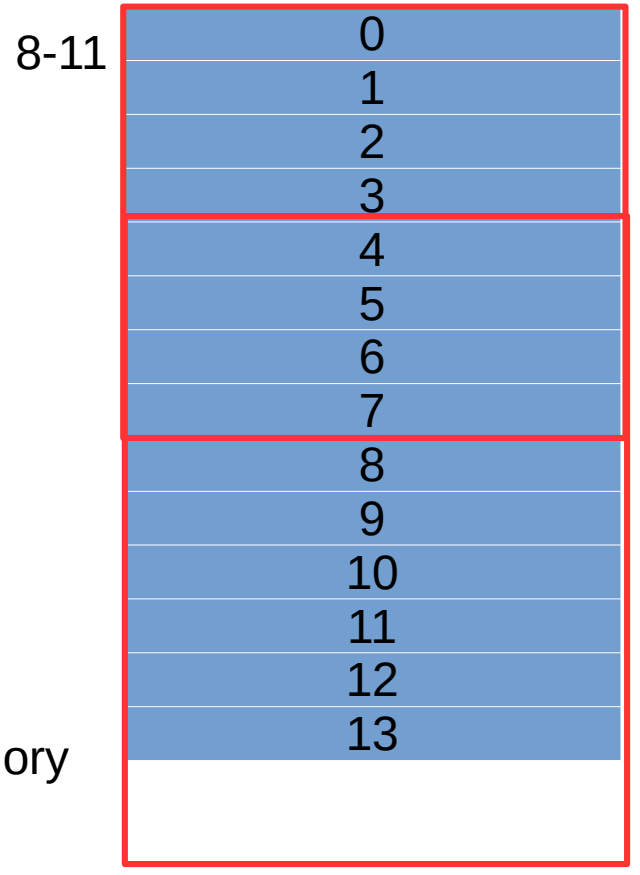
Example 8

- Show the hits and misses and final cache contents for a fully associative cache with 2-word blocks and a total size of 8 words. Assume LRU replacement.

Way 0 Way 1

Similarly we will have 2 more ways
To make the total size 8 words

Assume addresses:
4: 00000100, 0: 00000000, 8: 00001000
How many bytes of data will you fetch from memory for each address?



https://inst.eecs.berkeley.edu/~cs61cl/fa08/labs/lab22.html

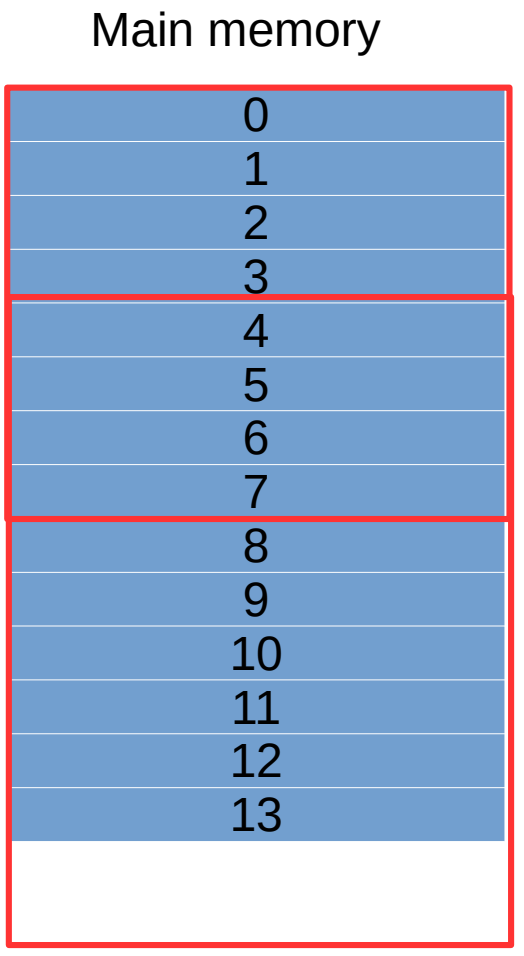
Example 8

- Show the hits and misses and final cache contents for a fully associative cache with 2-word blocks and a total size of 8 words. Assume LRU replacement.
  - Assume addresses

    4: 00000100, 0: 00000000, 8: 00001000

    Since it is a 2-word block, we have to

    fetch 2 words (2 words = 8 bytes).
  - When you get address 4, what will you fetch?
  - Divide the address by 8.
  - 0/8 and 4/8 yield = 0--> fetch these words
  - 4/8 and 8/8 yield 0 and 1 as quotients. So, 4 and 8 will index into different lines



https://inst.eecs.berkeley.edu/~cs61cl/fa08/labs/lab22.html

Acknowledgements

- MITx- 6.004- Computation Structures-
  - https://www.youtube.com/watch?v=hg0dTS10uSk&list=PLDSlqjcPpoL64CJdF0Qee5oWqGS6we_Yu&index=13
  - https://www.youtube.com/watch?v=-XqdkA931ag&list=PLDSlqjcPpoL64CJdF0Qee5oWqGS6we_Yu&index=14
- http://home.ku.edu.tr/comp303/public_html/Lecture15.pdf

Acknowledgements

- UCB- CS 252
  - Course page: http://inst.eecs.berkeley.edu/~cs152/sp18/
  - Memory: http://inst.eecs.berkeley.edu/~cs152/sp18/lectures/L05-Memory.pdf
  - http://inst.eecs.berkeley.edu/~cs152/sp18/lectures/L06-MemoryII.pdf

- Hennessey and Patterson
- Tech Review pages