Cache Memory

Muhammad Tahir

Lecture 19-20

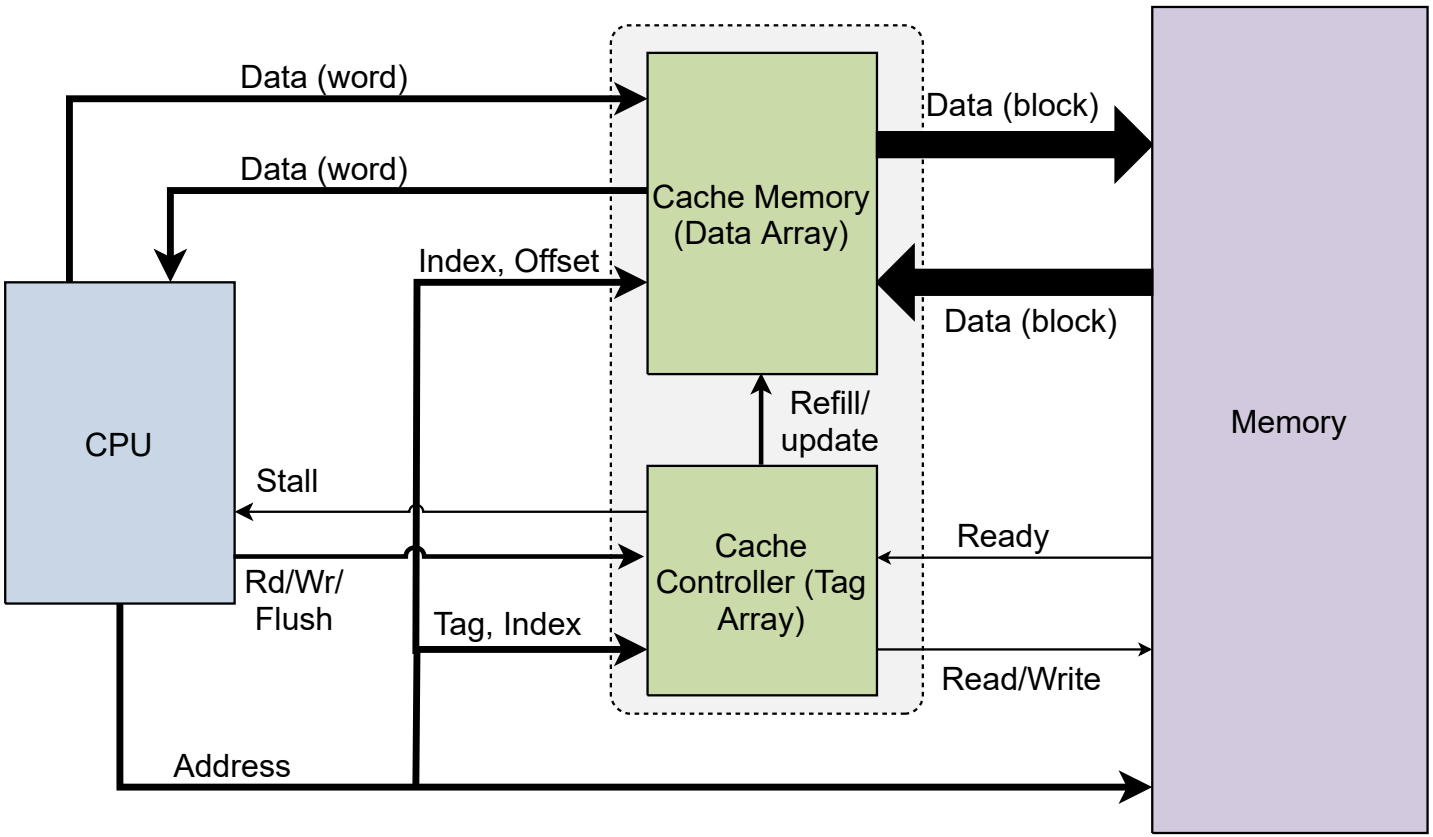
Electrical Engineering Department
University of Engineering and Technology Lahore

Contents

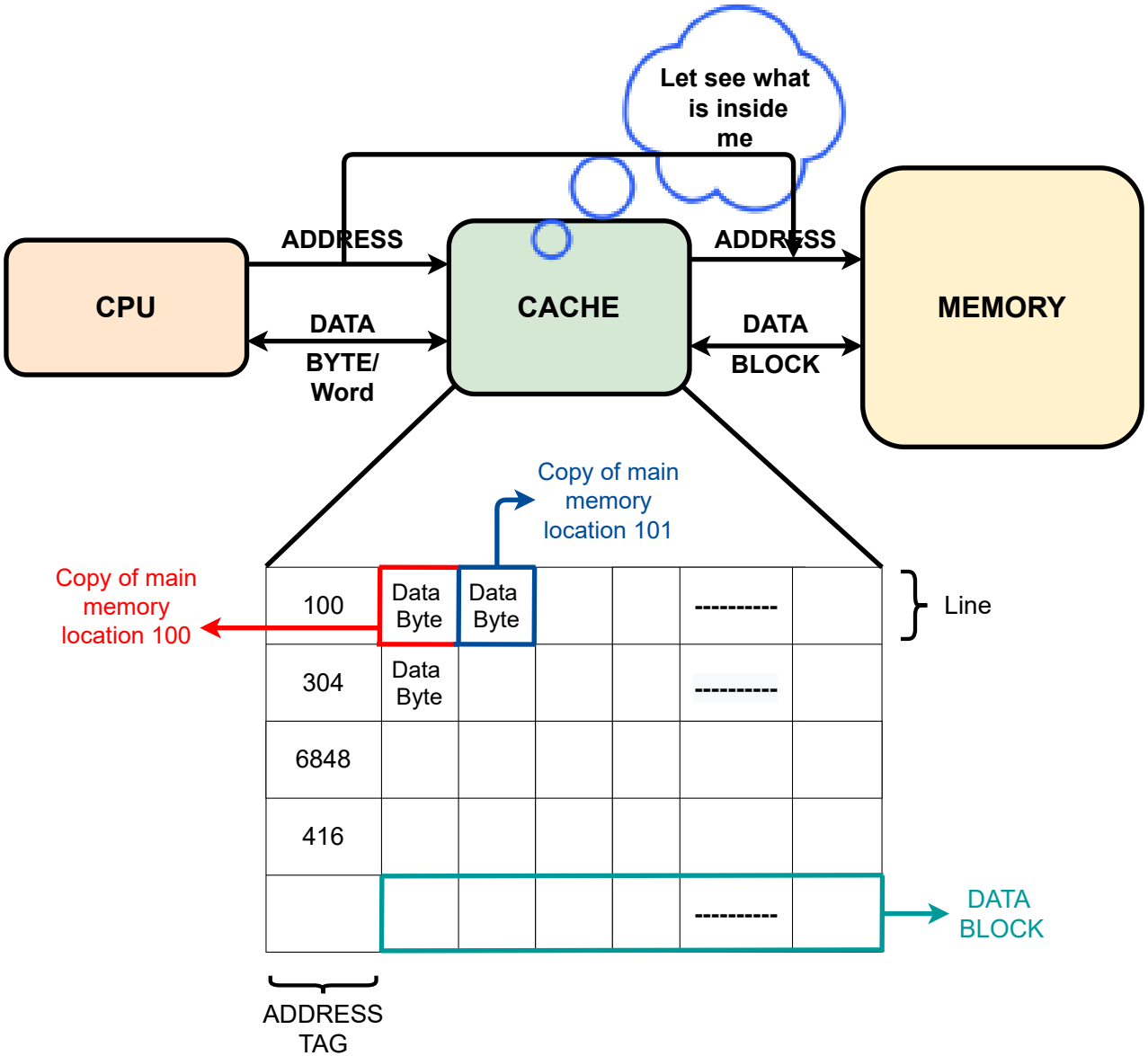
What is Cache Memory

- A `Cache`:
  - is a small but fast memory leading to reduced latency
  - holds an identical copy of most frequently used segments from main memory
  - provides faster access or reduced latency
  - is transparent to the user
  - can have multiple levels
- A split cache is the one which has separate data and instruction caches

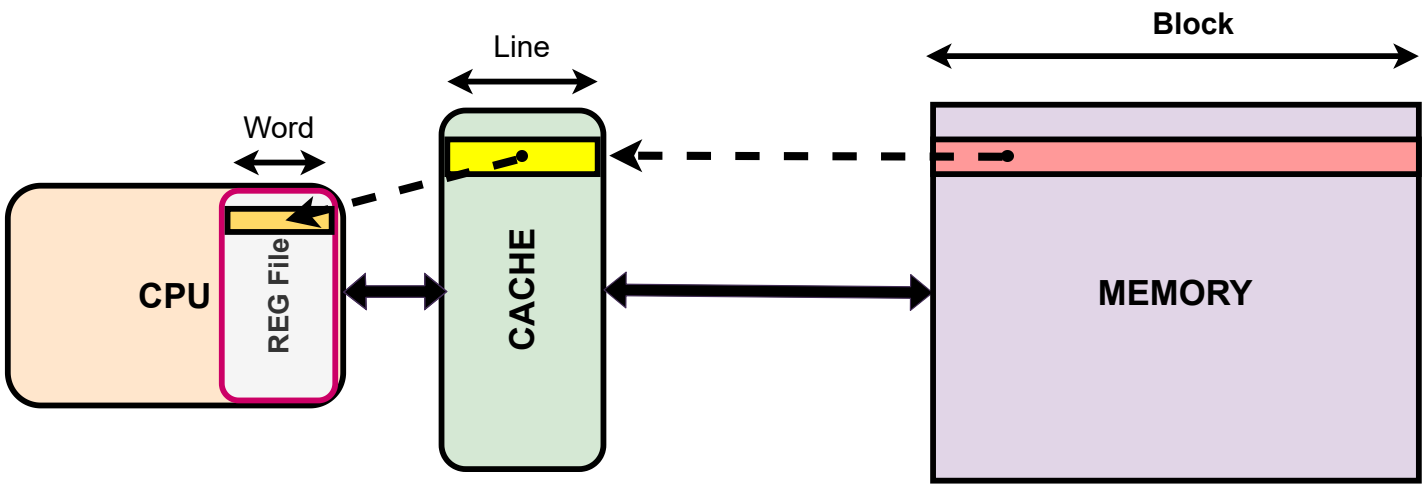
Cache Structures



Cache Structures cont'd



Cache Structures cont'd

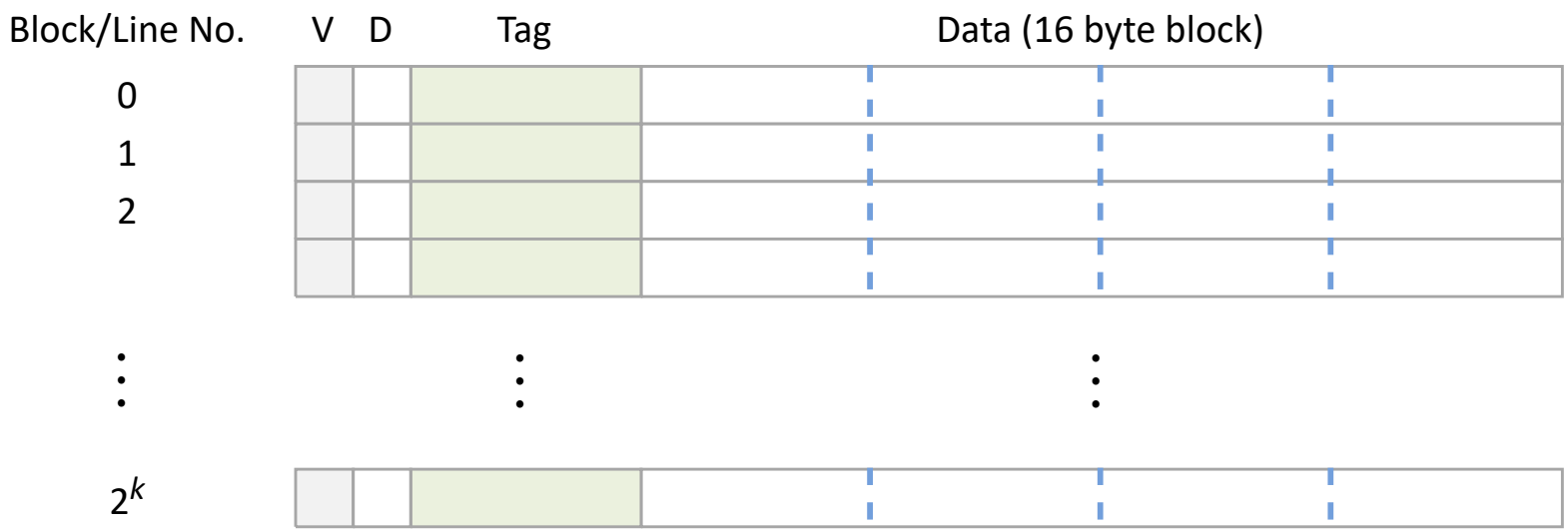


Caching Terminology

- Capacity (C): Number of bytes that the cache can store
- Cache Block/Line: Unit of storage in the cache. For block size of b bytes, the cache has $B = C/b$ blocks
- Cache Hit/Miss: Whether data (while executing an instruction) is found (hit) or not found (miss) in the cache
  - Read Miss: Data is not found in the cache when performing a read operation
  - Write Miss: When writing data to cache, the corresponding block is not in the cache

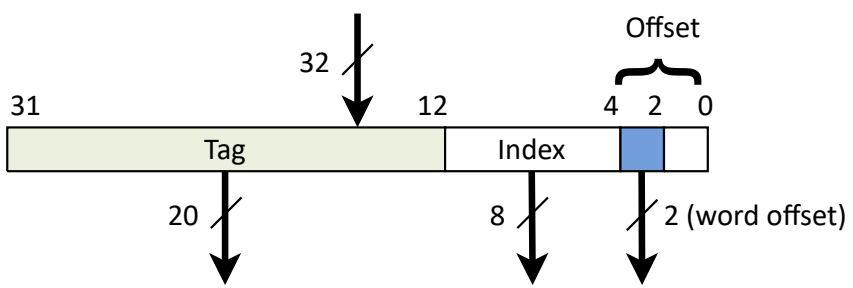
Cache Blocks

- Each cache block or cache line has a tag field to find whether the requested data is already in the cache or not
- A cache block has a valid bit to determine whether the data in the block is valid or not
- There is a dirty bit to mark whether the block is modified while in cache (dirty) or not modified (clean)



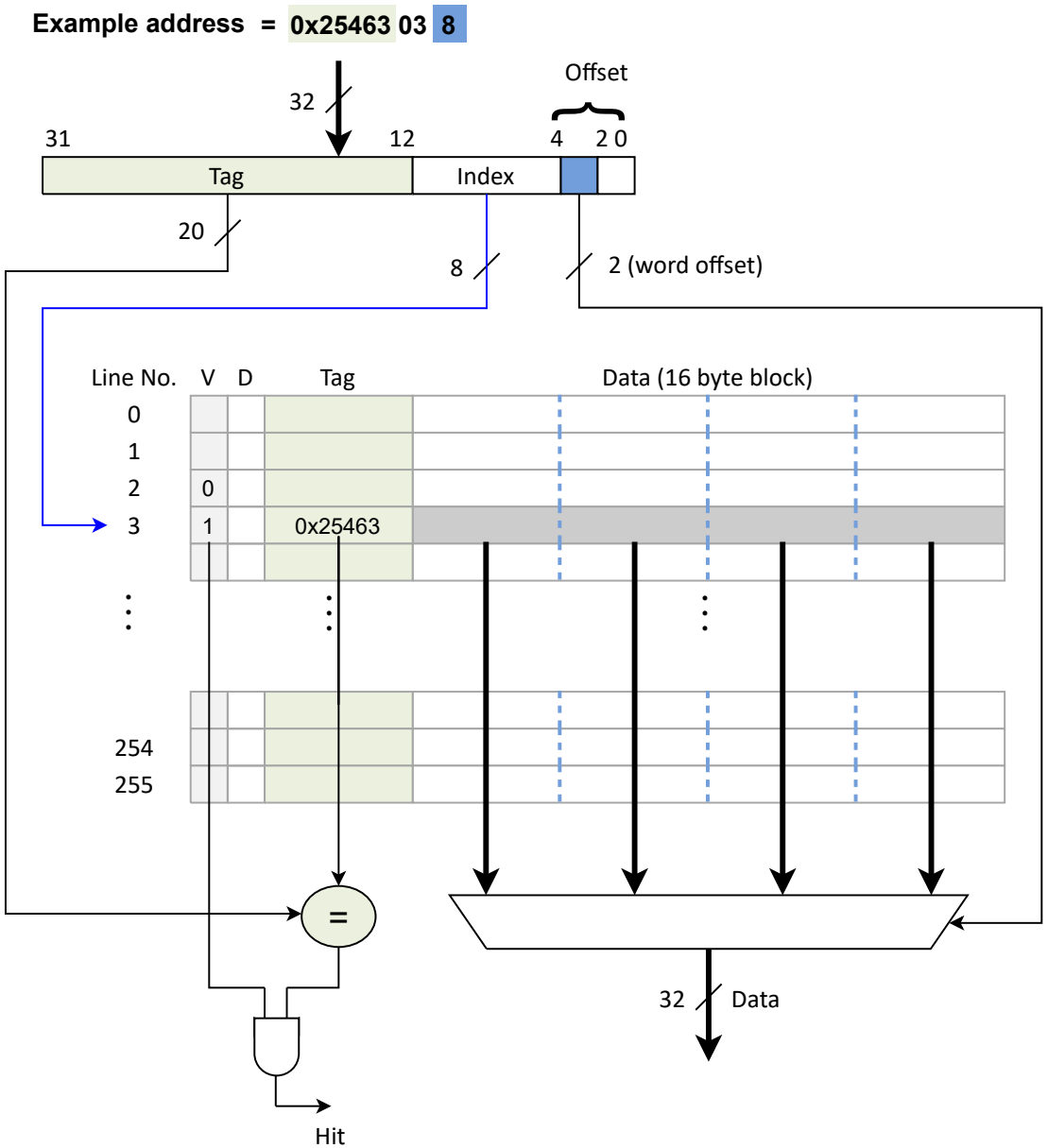
Cache Addressing

- Logically memory (main) is divided into multiple blocks
- Each block from main memory maps to a cache line
- The selection of cache line is determined by the index field in the address
- The tag field is used to determine the presence/absence of data in the cache
- The tag and index fields collectively form the block address
- Which data element to select from the cache line, is determined by the offset field

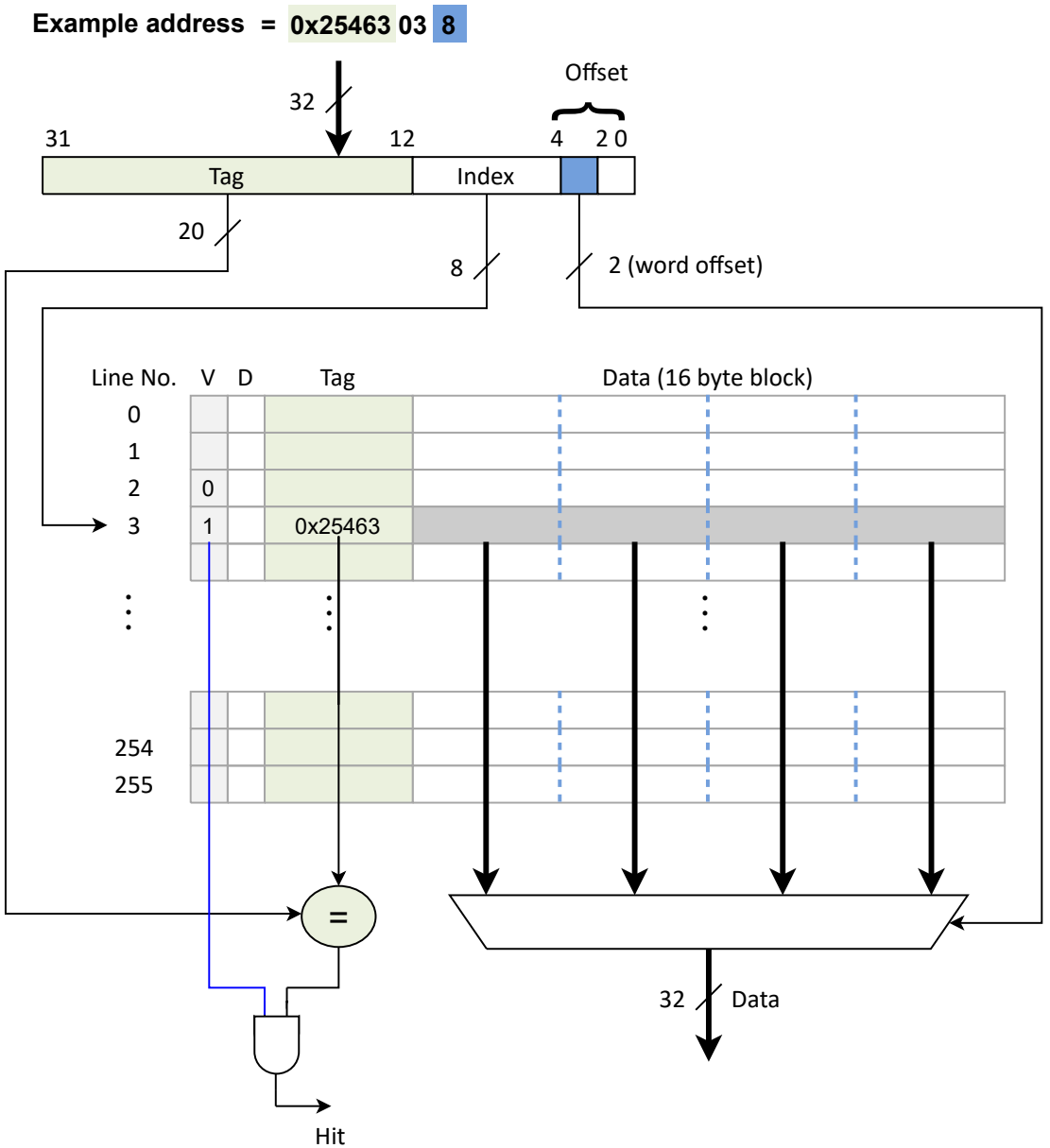


Identifying the Cache Contents

Step1: Locate cache line with ***index*** field



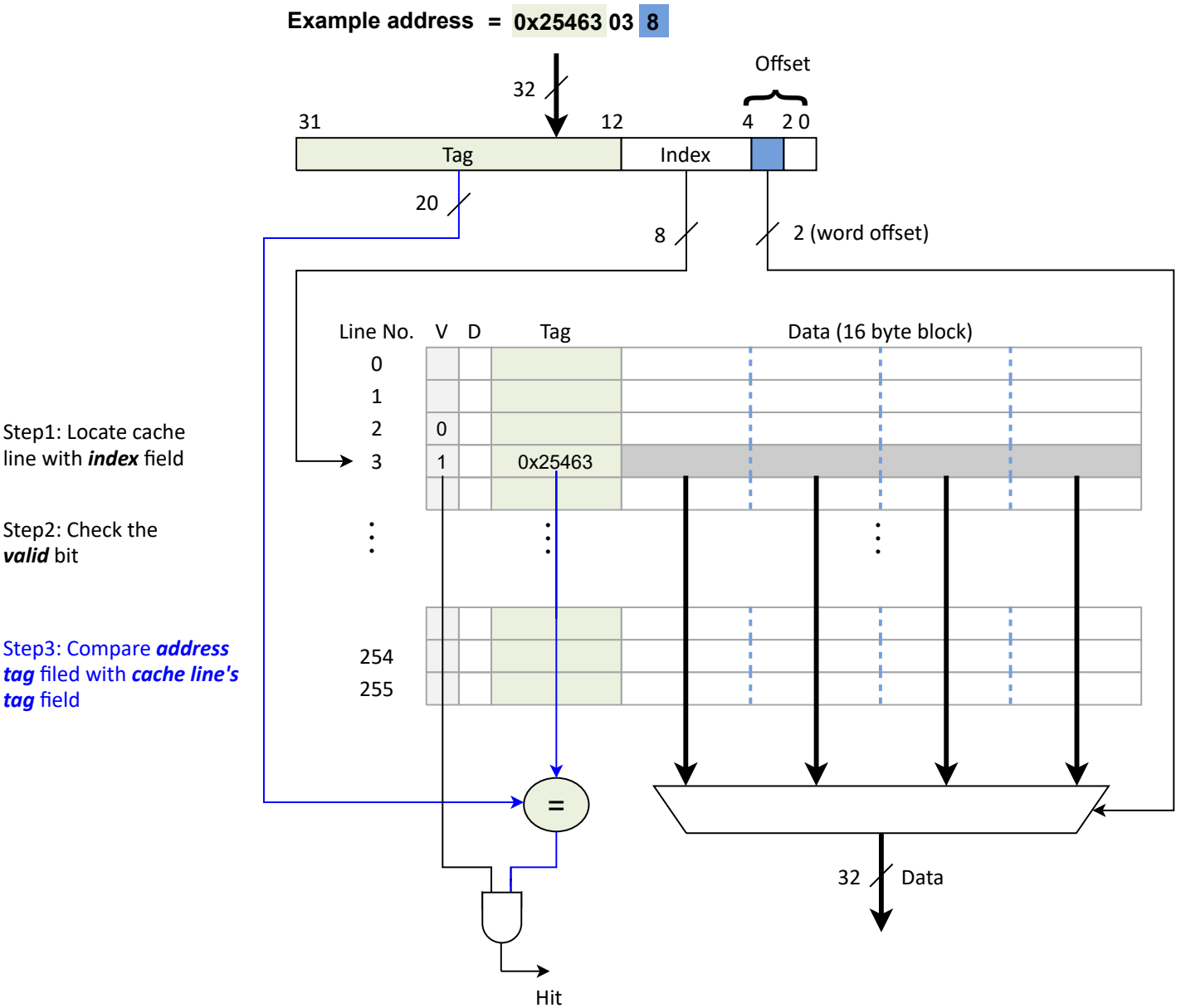
Identifying the Cache Contents Cont'd



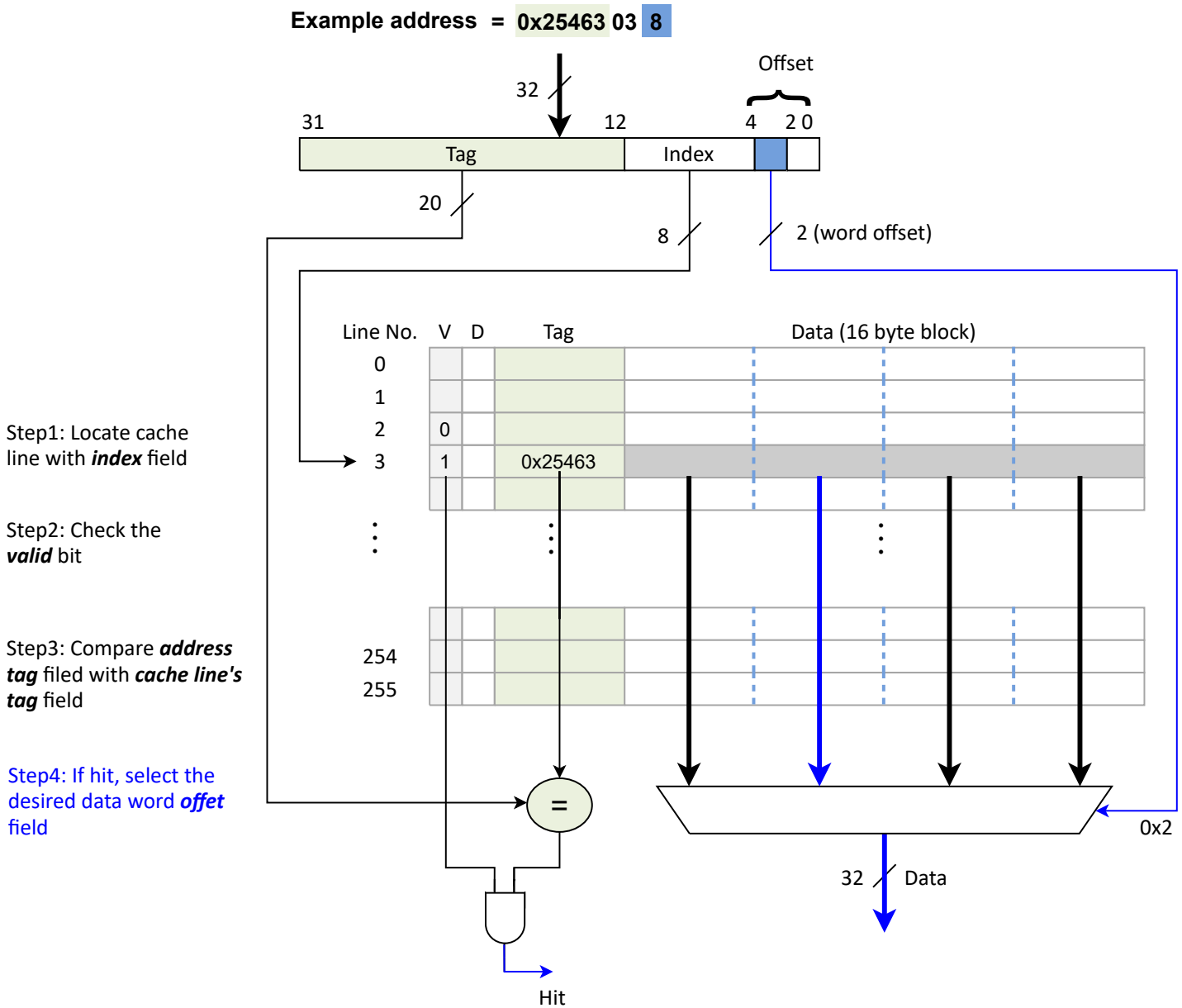
Step1: Locate cache line with ***index*** field

Step2: Check the ***valid*** bit

Identifying the Cache Contents Cont'd



Identifying the Cache Contents Cont'd



Caching Design Decisions

- Placement Policy: Where and how to place a block in the cache
- Replacement Policy: Which data/block to remove to bring in a new block
- Write Policy: How do we manage main memory data consistency when performing write operations
- Finding Data: How do you determine that the requested information is available in the cache or not?

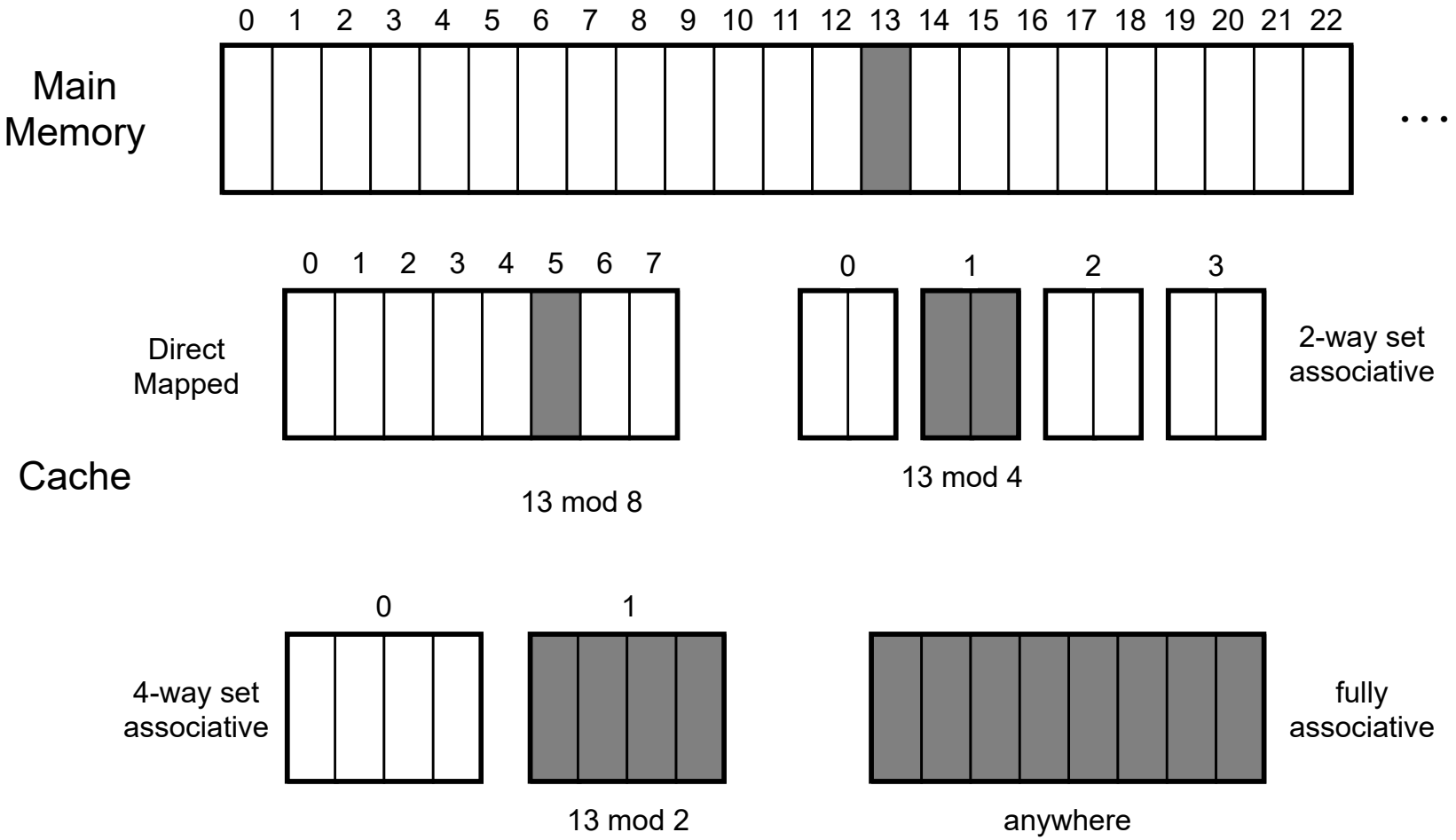
Placement Policies

- Direct Mapped
  - A block can only go in a single line of the cache – has many sets but each set can hold only one block
  - Only single tag needs to be checked against block number
  - Different blocks can go to the same line
  - A particular block can only go to one line (may lead to conflict misses)

Placement Policies Cont'd

- Set Associative
  - A block can be placed in any of the N lines (belonging to the indexed set)
  - There are M sets and each set contains N blocks ($B = M \times N$)
  - $N > 1$ but less than the total number of lines in the cache
- Fully Associative
  - Any block can be placed in any line
  - All tags are compared against incoming block number
  - A special case of set associative cache where N is equal to number of cache lines

Spectrum of Associativity



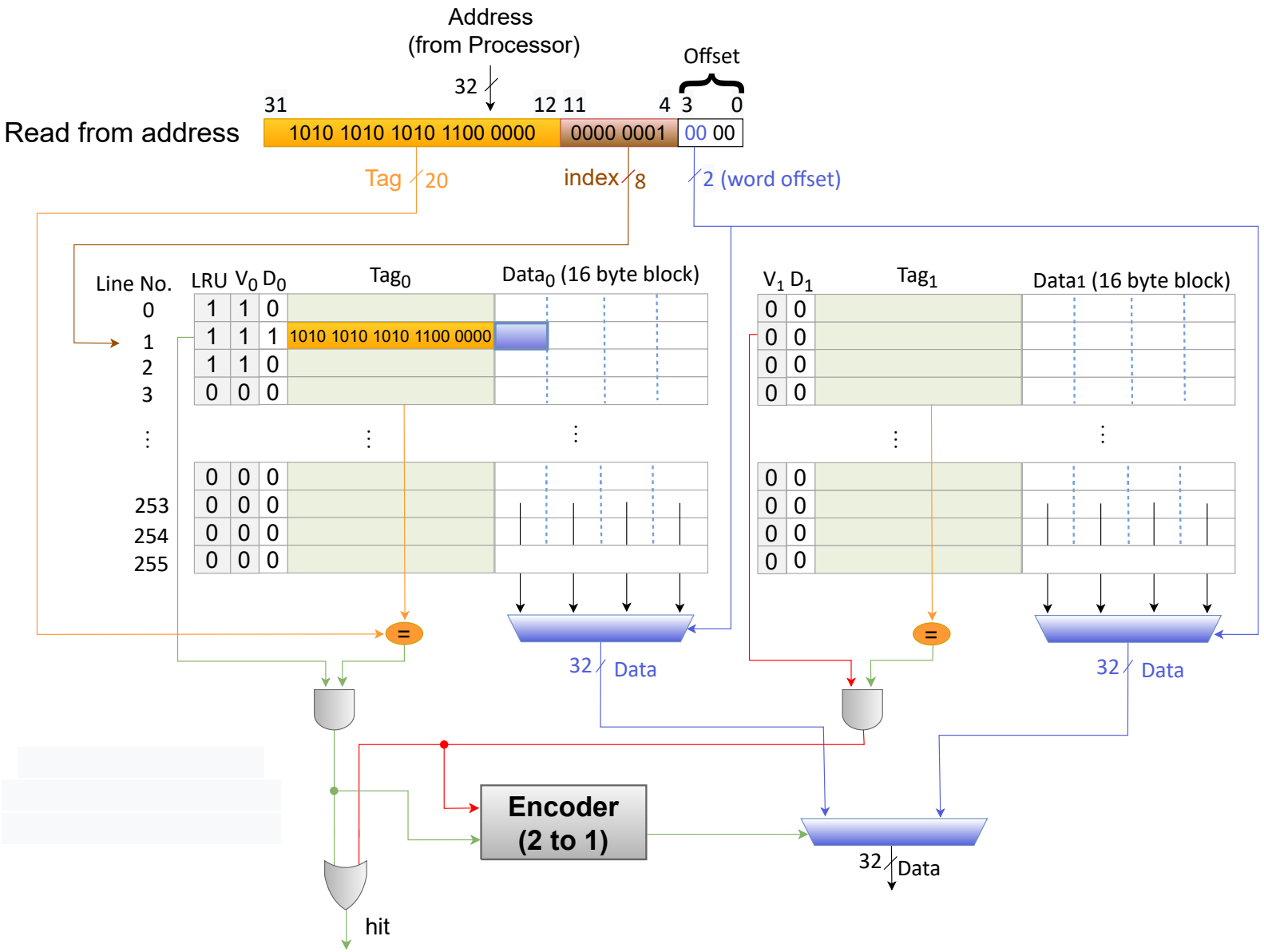
Direct Mapped Cache

- Look in only one place
  - Fast
  - Cheap (only one comparator, one valid bit checker)
  - Energy efficient
- Block must go to one place
  - If we access block A B A B A B... and both map to the same cache line (0 & 8 in the previous example)
  - We will always get a miss because of conflicts even though there could be other lines available

Set Associative Cache

- N-way set associative when a particular block can be in one of N lines
- A cache is dived into multiple sets and each set can have multiple lines
- A block can go to any of the lines within a set

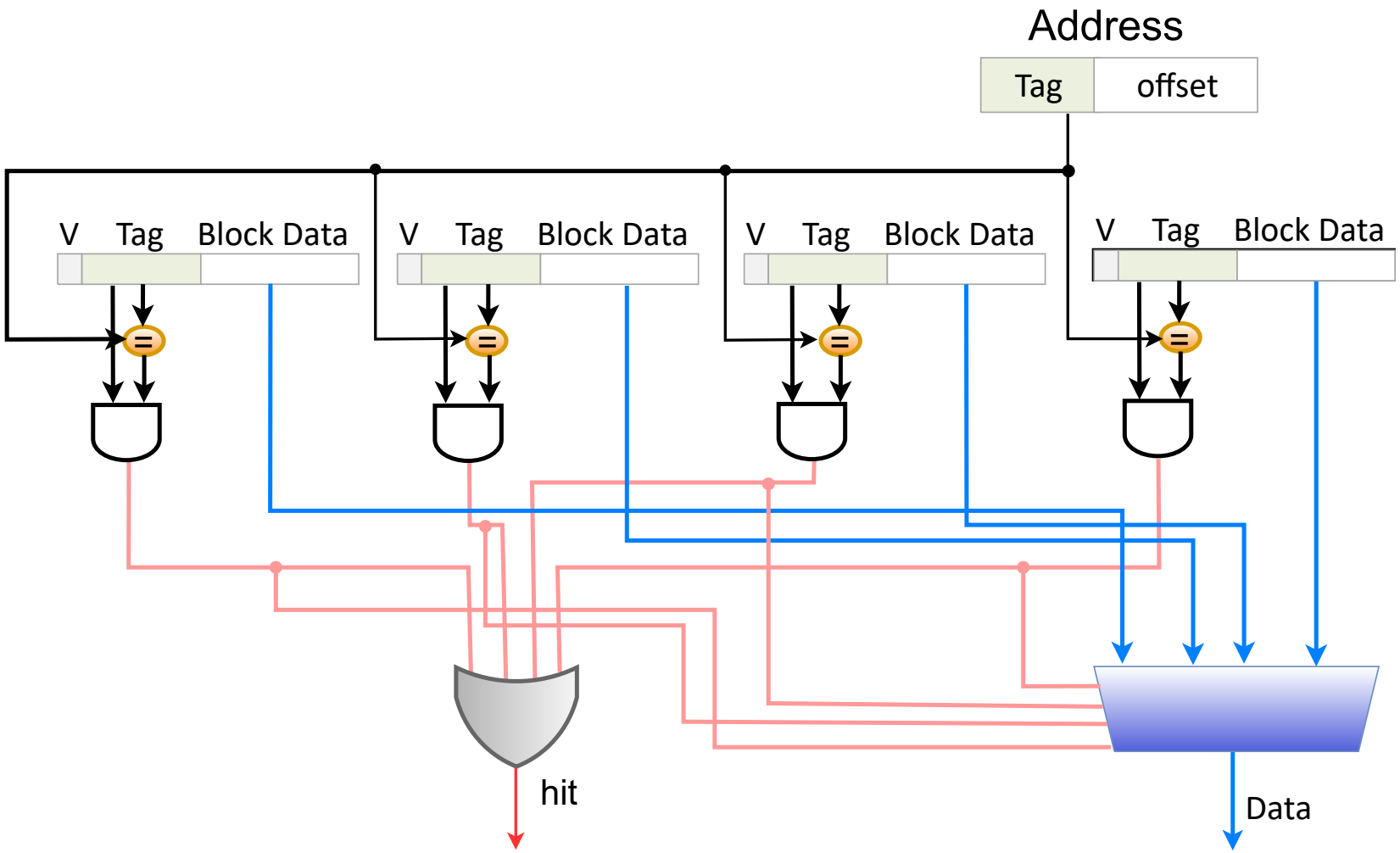
Set Associative Cache Cont'd



Fully Associative Cache

- Any block maps to any line of the cache
- A cache with 8 lines will have offset bits 5...0
- There will be no Index bits since block can be anywhere
- Have to check all the tags

Fully Associative Cache Cont'd



Set Associative Cache Example

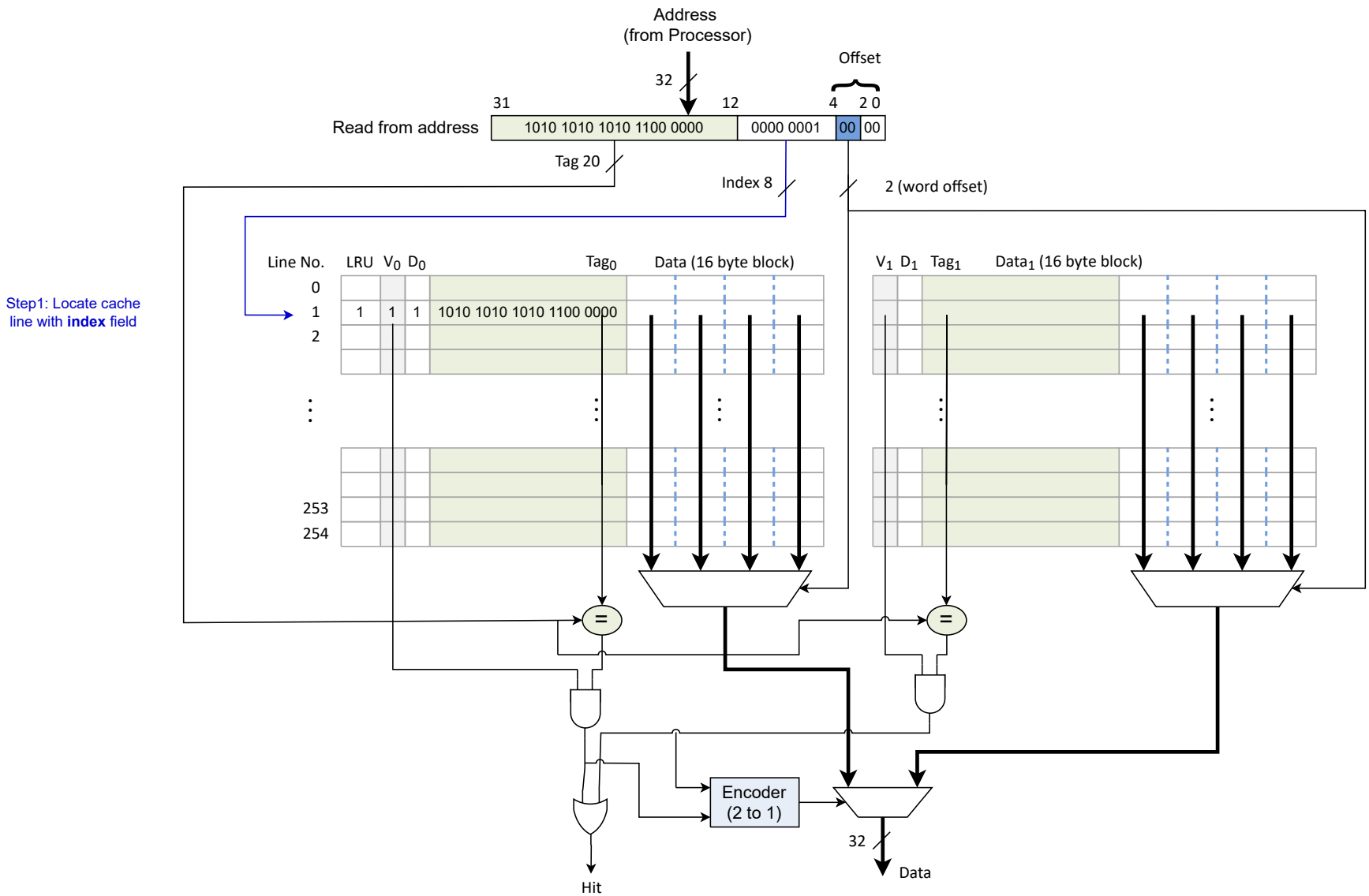
Consider a processor with the following attributes:

- A two-way set associative cache with 256 cache lines
- 16-byte cache blocks
- An LRU cache replacement policy

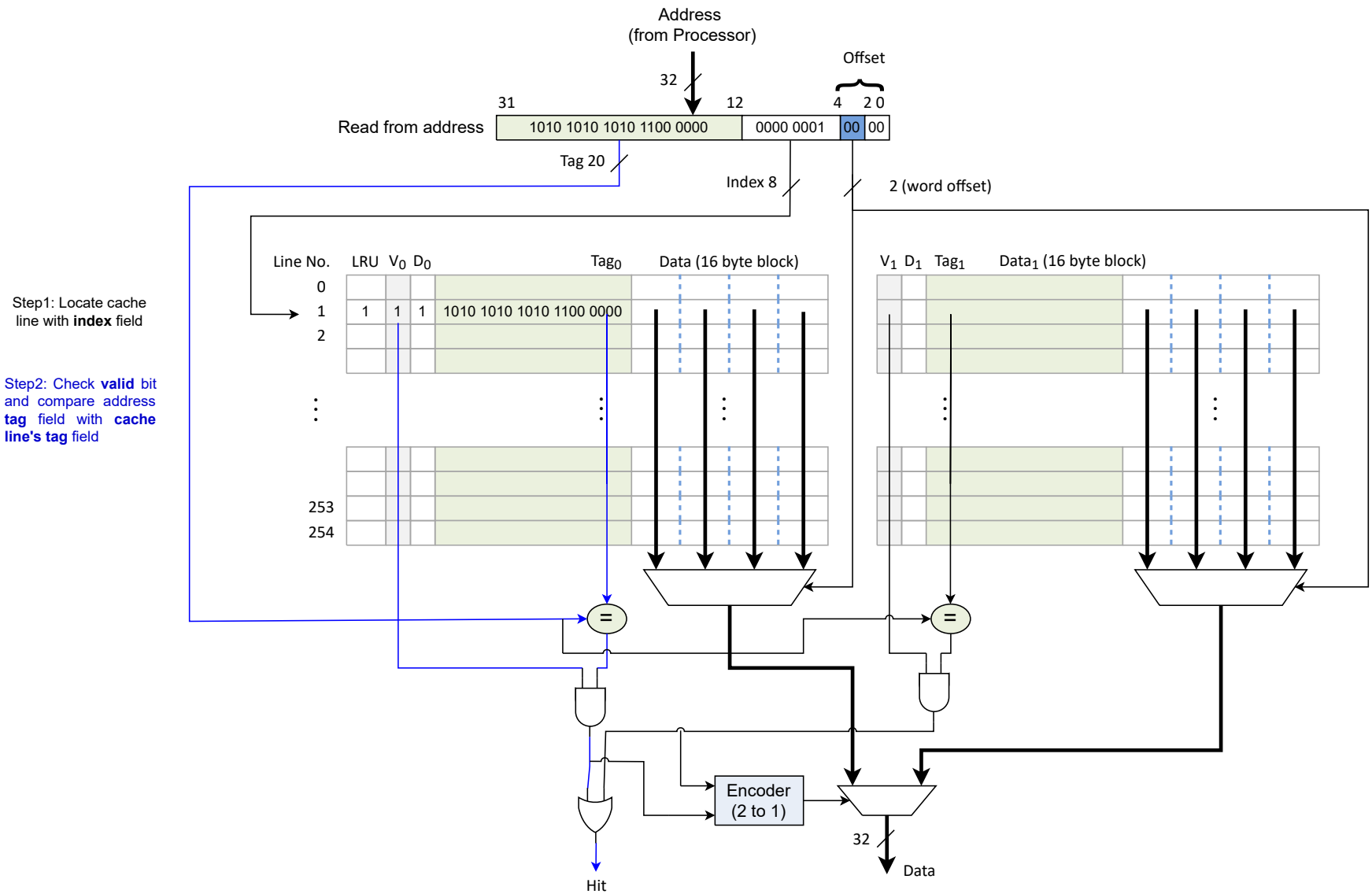
Assume the program performs the following memory accesses

- read from address 0xAAAC0010
- read from address 0xAAAC3010
- read from address 0xAAAC3010
- write to address 0xAAACA010

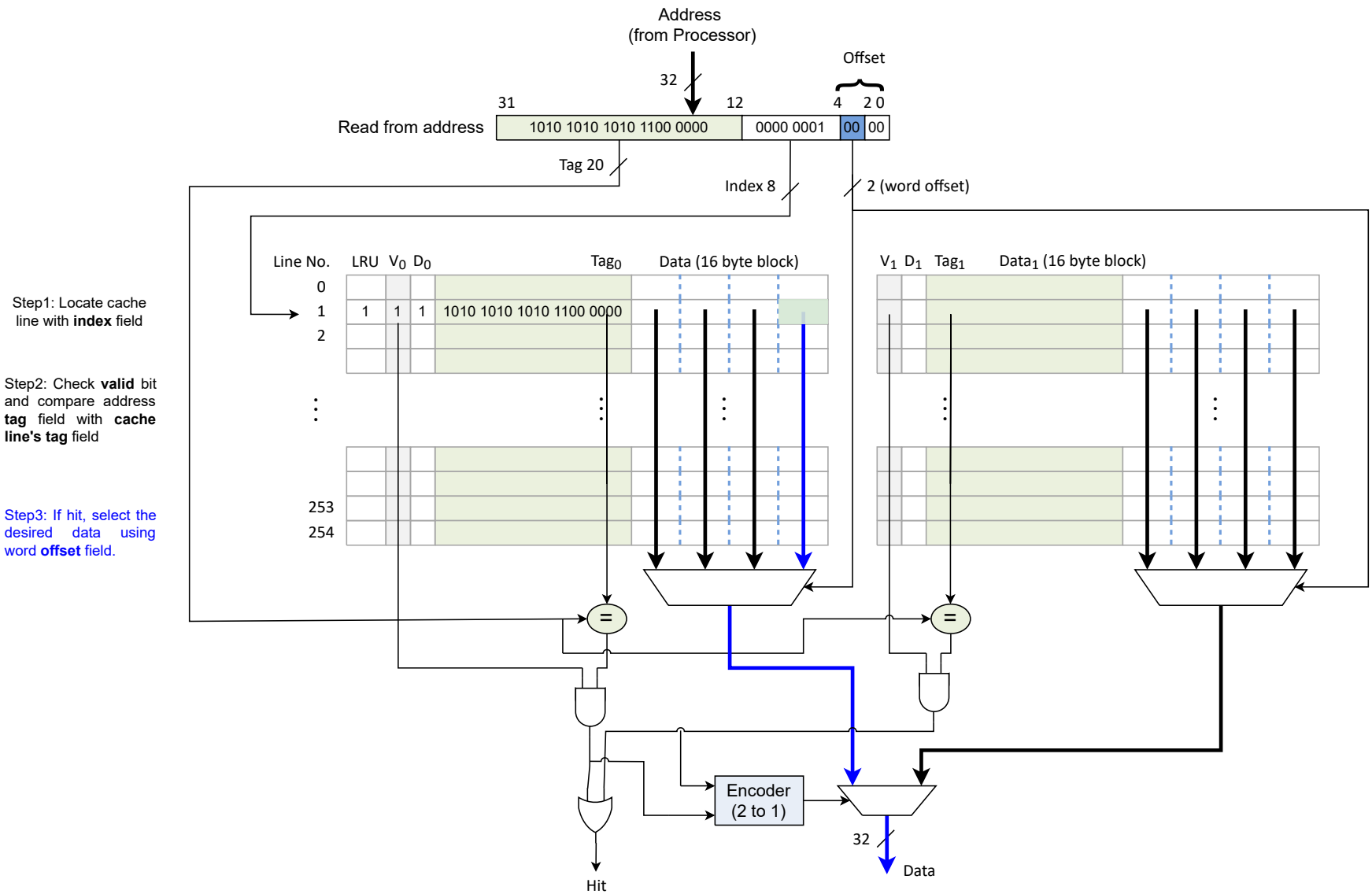
Set Associative Cache: Read Hit Example



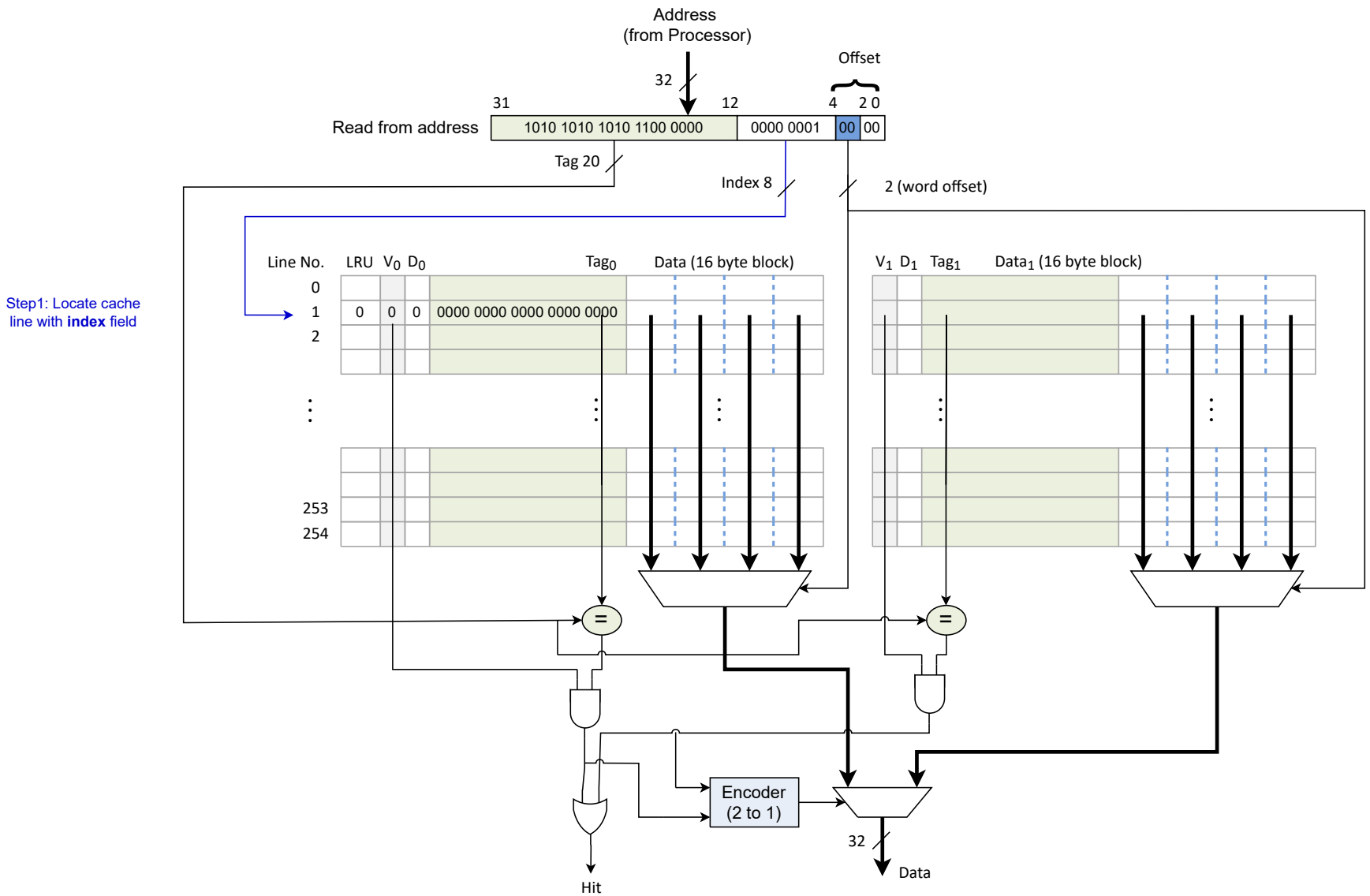
Set Associative Cache: Read Hit Example Cont'd



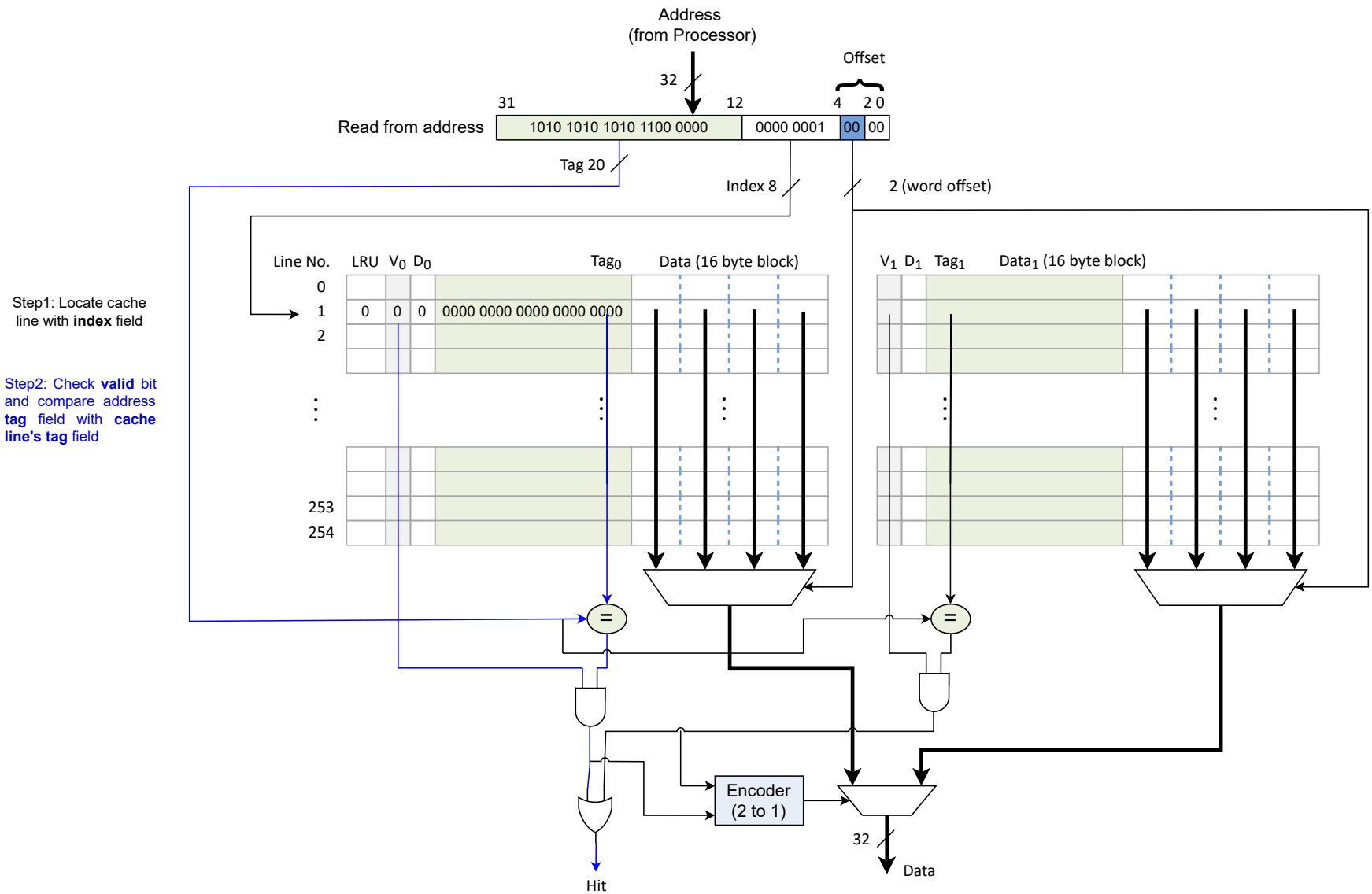
Set Associative Cache: Read Hit Example Cont'd



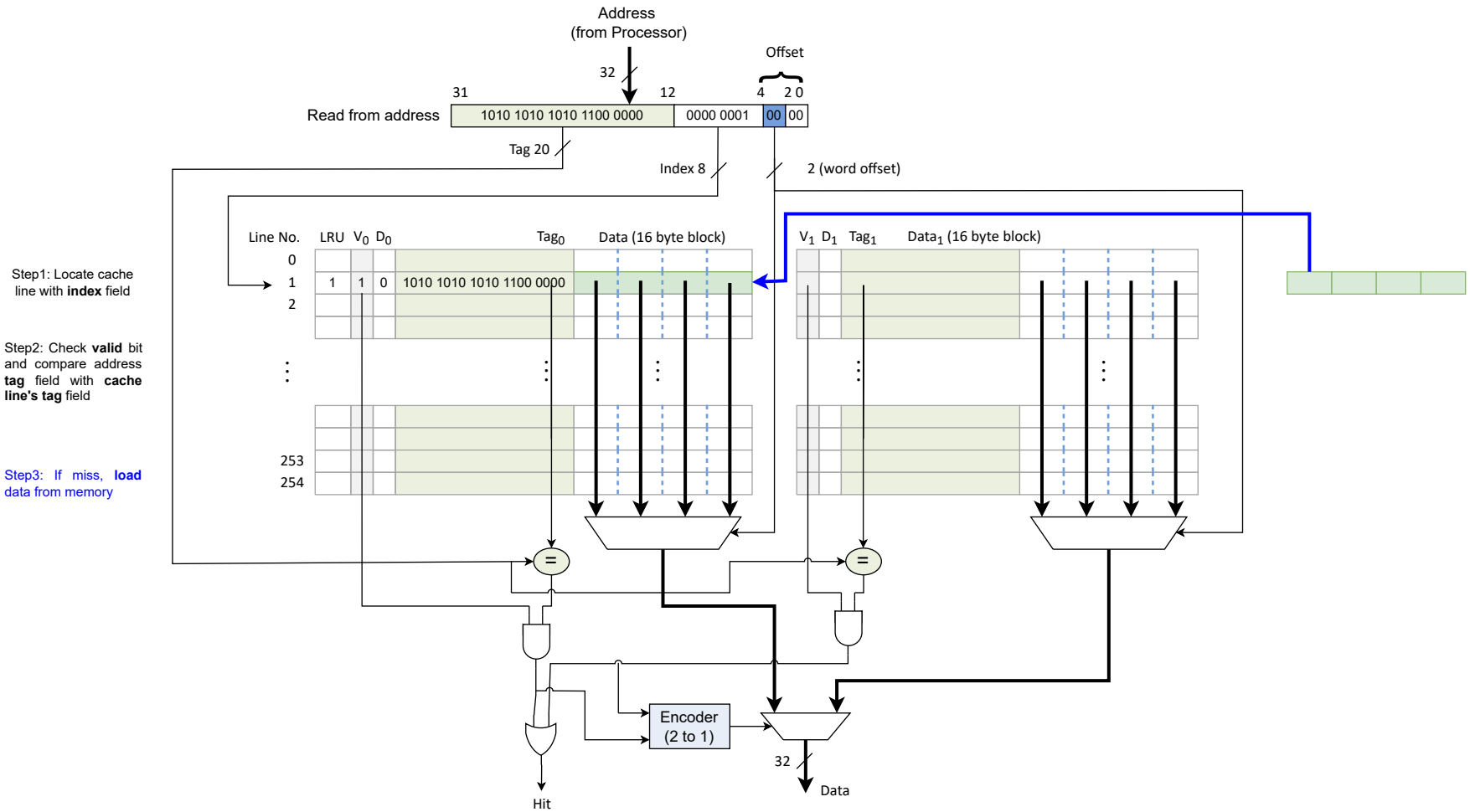
Set Associative Cache: Read Miss Example



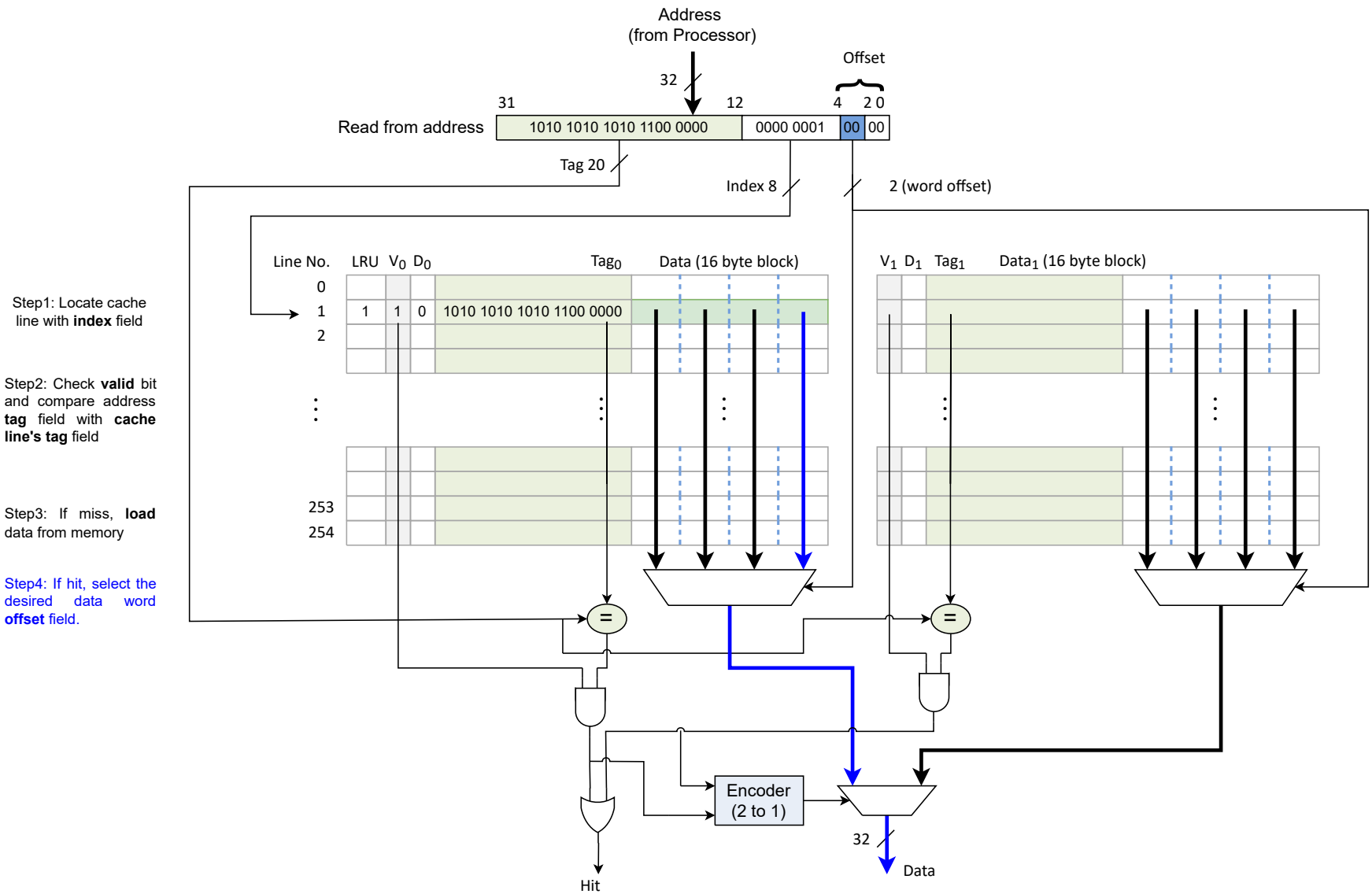
Set Associative Cache: Read Miss Example Cont'd



Set Associative Cache: Read Miss Example Cont'd



Set Associative Cache: Read Miss Example Cont'd



Cache Misses

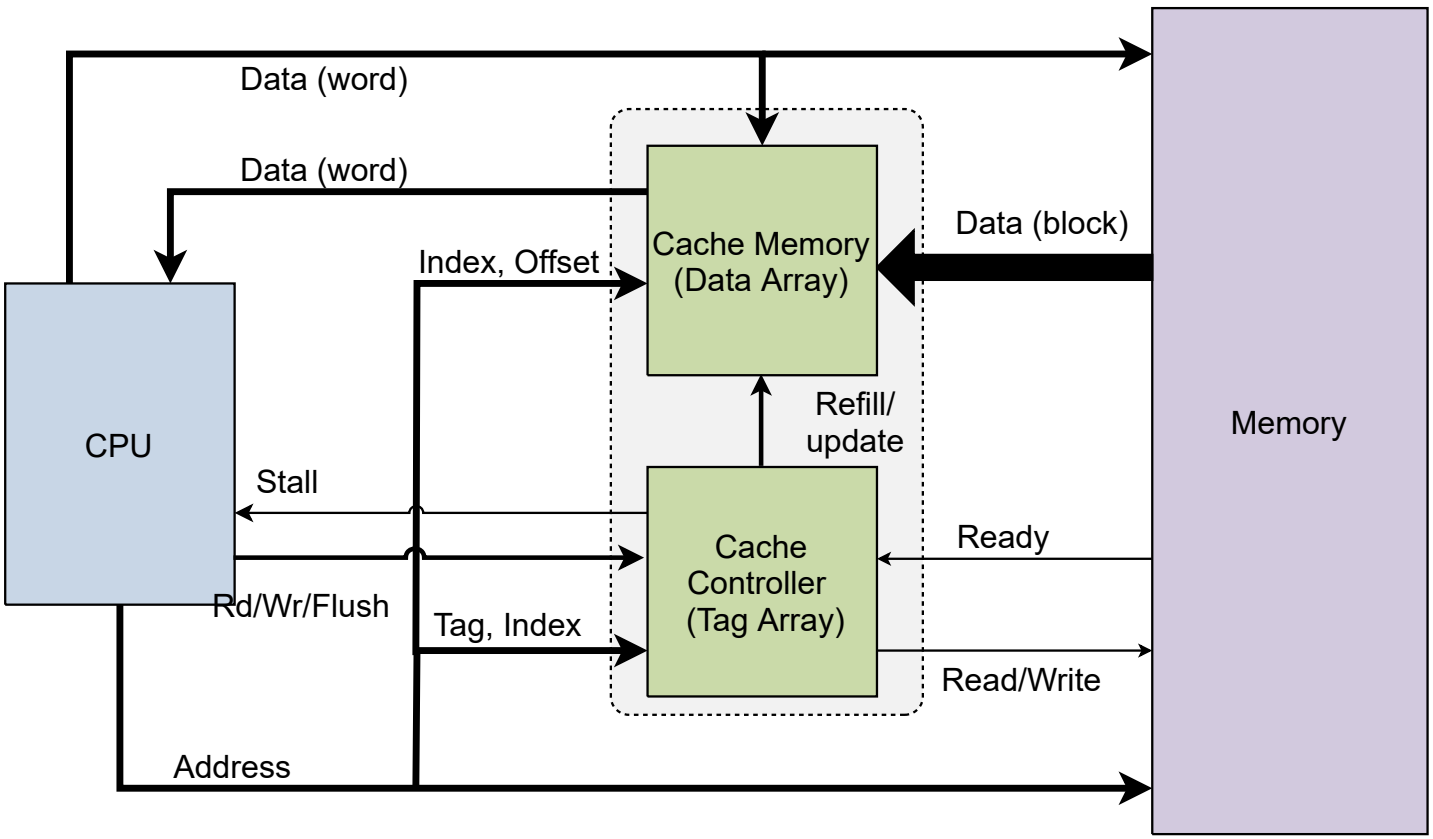
- Compulsory misses: Occurs when data is accessed first time
- Capacity Misses:
  - Cache is too small to hold all data of interest at one time
  - Assuming cache is full, accessing data X (not in cache), must evict data Y
  - **Capacity miss** occurs if program tries to access Y again
- Conflict misses: data of interest maps to same location in cache

Replacement Policy

- Random
- Least Recently Used (LRU)
  - LRU cache state must be updated on every access
  - True implementation only feasible for small sets (2-way)
  - Pseudo-LRU binary tree often used for 4-8 way
- First In, First Out (FIFO) a.k.a. Round-Robin
  - Used in highly associative caches
- Not Least Recently Used (NLRU)
  - FIFO with exception for most recently used block or blocks

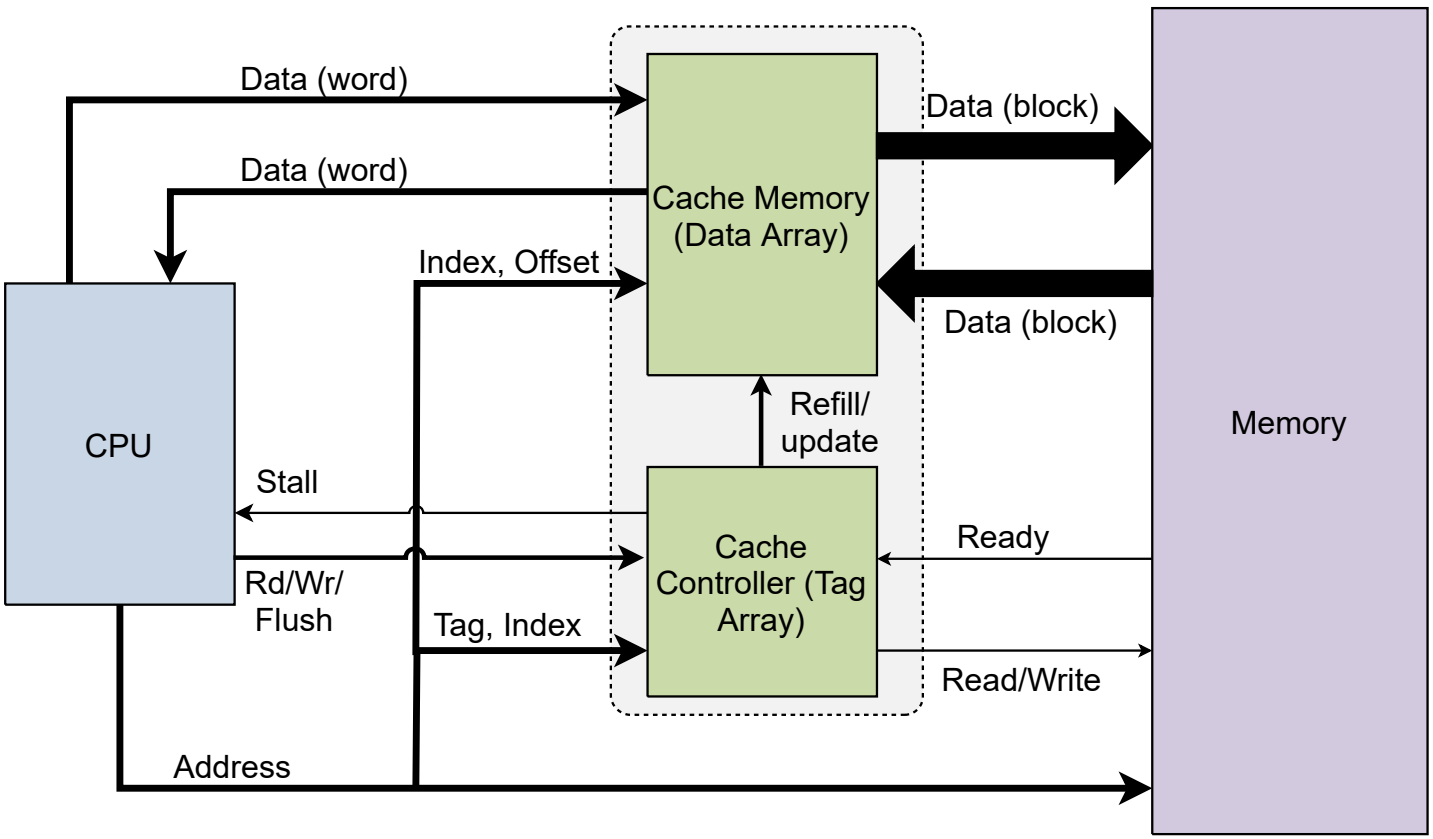
Cache Write Policies

- Write-through cache: Update both cache and (main) memory (rarely used now)



Cache Write Policies Cont'd

- Write-back cache: Only write to the cache. Write to the (main) memory only when the block is replaced



Cache Write Policies Cont'd

- Allocate policy
  - Do we allocate an entry in the cache for blocks we write on a write miss?
    - Write-allocate: Brings in a block when we have a write miss. Most caches today are of this type because in general there is some locality between writes and read.
    - No-write-allocate: Brings in a block only on a read miss
  - Most caches today are write-allocate with write-back

Cache Performance Parameters

- Cache hit time
  - Time between sending address and data returning from cache
- Cache miss rate
  - Number of cache misses divided by number of accesses
- Cache miss penalty
  - Extra processor stall caused by next-level cache/memory access
- Cache miss time/latency
  - Time between sending address and data returning from next-level cache/memory to processor

Cache Performance Analysis

- Average access time when using cache memory:

$$AMAT = HitTime + (MissRate) \times (MissPenalty)$$

OR

$$AMAT = (1 - MissRate) \times Hittime + MissRate \times MissTime$$

where

$$MissTime = HitTime + MissPenalty$$

Want the AMAT to be low

- Hit time to be low – small and fast cache
- Miss rate to be low – large and/or smart cache
- Miss penalty to be low – main memory access time

Suggested Reading

- Read relevant sections of Chapter 5 of [Patterson and Hennessy, 2021].
- Read Appendix B of [Patterson and Hennessy, 2019].

Acknowledgment

- Preparation of this material was partly supported by Lampro Mellon Pakistan.

References

Patterson, D. and Hennessy, J. (2021).

Computer Organization and Design RISC-V Edition: The Hardware Software Interface, 2nd Edition.

Morgan Kaufmann.

Patterson, D. and Hennessy, J. (6th Edition, 2019).

Computer Architecture: A Quantitative Approach.

Morgan Kaufmann.