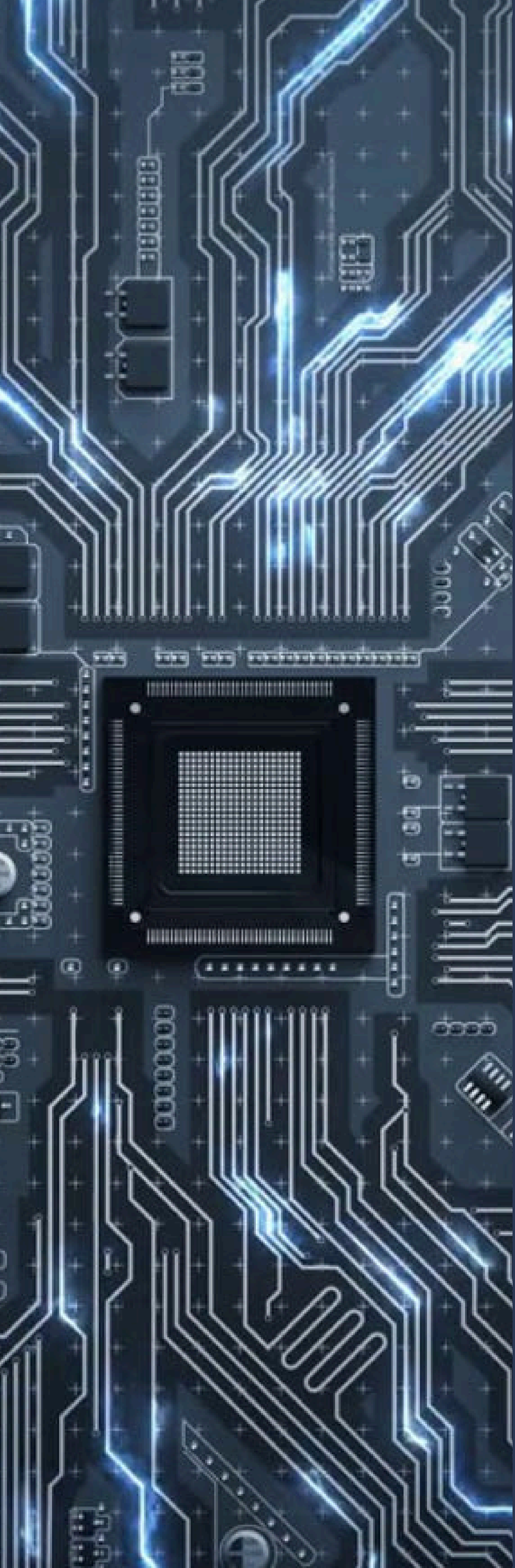


2024

CS203 PROJECT

Direct Mapping Cache Design

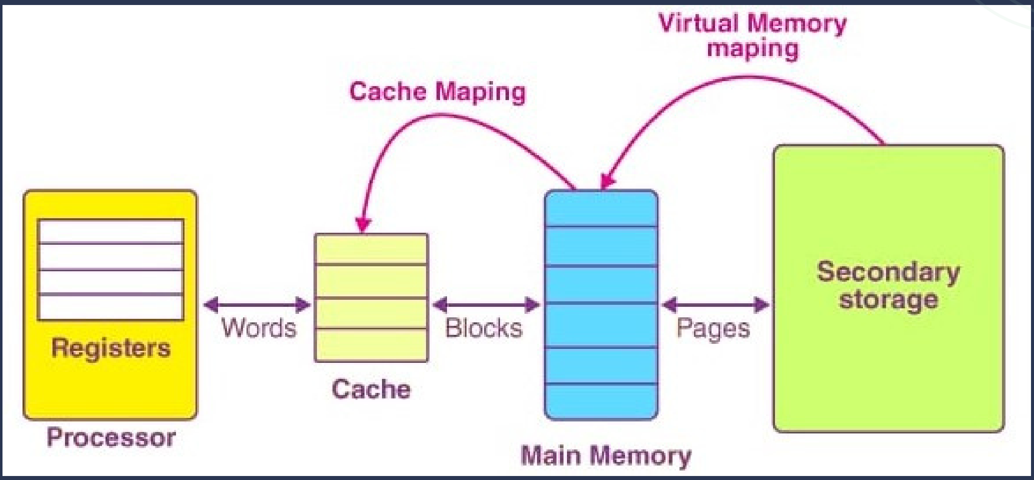
PRESENTED BY

NITIN KUMAR	2023CSB1141
HARDIK GARG	2023CSB1121
HARSH RAI	2023CSB1345
AASHISH SINGH	2023CSB1093
PARTH KULKARNI	2023CSB1142
ARPIT	2023CSB1099

TABLE OF CONTENTS

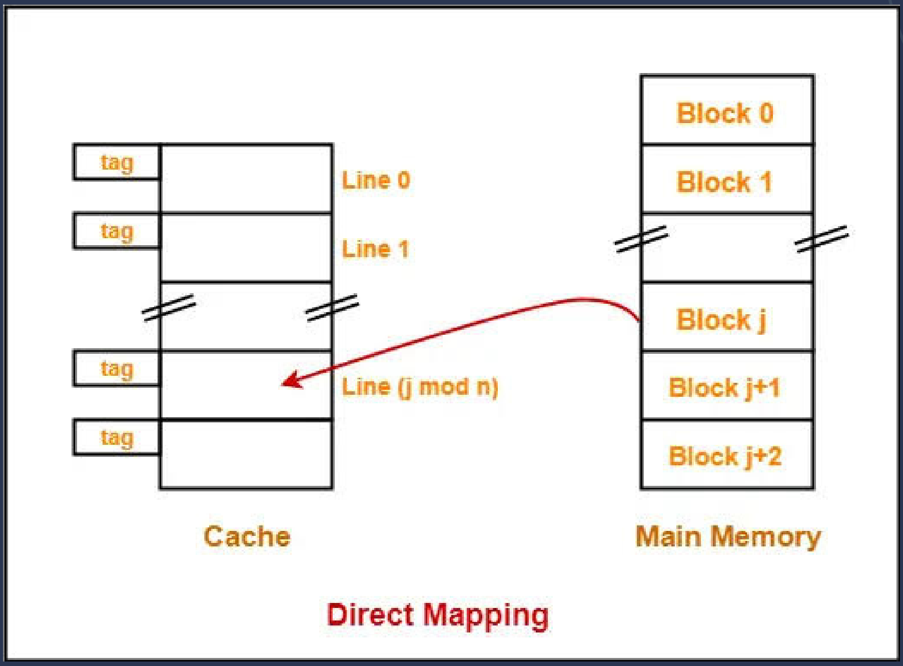
Abstract	1
Theory	2
Design Decisions	5
Challenges Faced	7
Performance Metrics	8
Bibliography	8

ABSTRACT



A direct-mapped cache in a computer system is a type of memory cache that uses a specific method for storing data to improve the speed of data access. In this Verilog project, our objective is to design a cache that maps memory addresses directly to cache lines. Let's go over the key concepts, specifications, and functionality.

THEORY



A cache is a smaller, faster memory that stores copies of frequently accessed data from a larger, slower memory, such as main memory (RAM). When a processor needs data, it first checks the cache. If the data is found in the cache (a "cache hit"), it can be accessed more quickly than if it had to be fetched from main memory.

If the data is not in the cache (a "cache miss"), it is fetched from the main memory and then stored in the cache for future access.
In a direct-mapped cache, each memory address maps to exactly one location (or "line") in the cache. This is unlike fully associative caches, where data can be stored in any cache line, or set-associative caches, which offer a compromise between the two.

Each cache line in a direct-mapped cache is structured to include three primary fields:

- Tag: A unique identifier used to determine which block of memory the data belongs to.
- Data: The actual information stored in the cache line.
- Valid Bit: A flag indicating whether the data in the cache line is valid and usable.

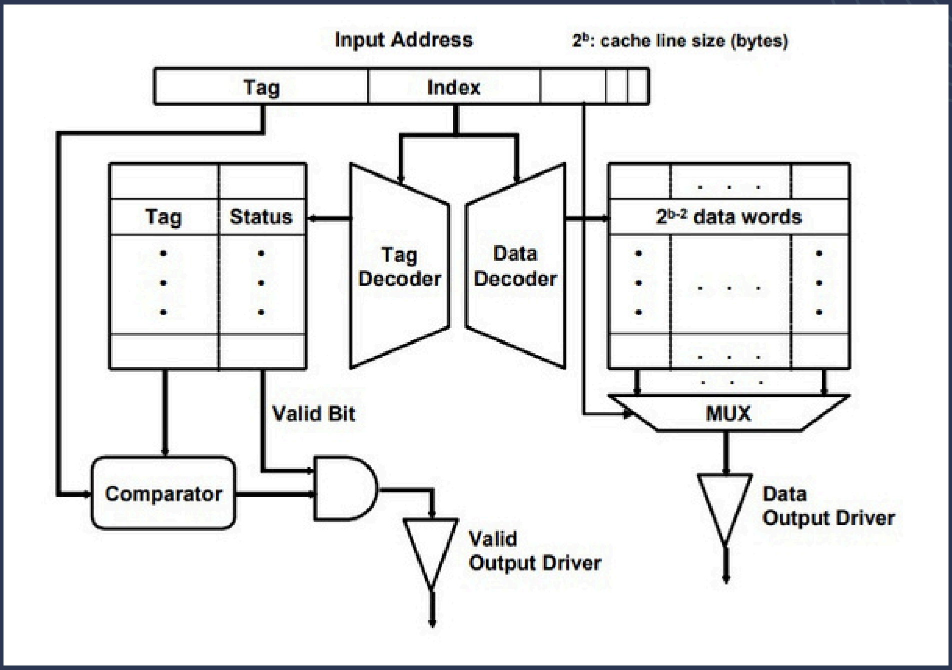
The size of these fields depends on parameters like the cache size, block size, and address width.
The process of accessing data in a direct-mapped cache begins with indexing into the cache using the address’s index field. The tag from the address is then compared with the tag stored in the indexed cache line. Simultaneously, the valid bit is checked.

If both the tags match and the valid bit is set, it is a cache hit, allowing the processor to quickly retrieve the data. Otherwise, a cache miss occurs, prompting the system to fetch the data from the main memory, update the cache line with the new tag and data, and set the valid bit.
This system supports both read and write operations. Reads involve checking for hits or misses and updating the cache on misses.

Writes can follow either a write-through or write-back policy, balancing speed and consistency between the cache and main memory.

Despite its simplicity and speed, direct-mapped caching can suffer from frequent cache misses when multiple memory blocks map to the same cache line—a drawback known as conflict misses. However, its straightforward design makes it a popular choice for applications where simplicity and predictability are critical.

DESIGN DECISIONS



The design consists of a single main module with parameters for DATA_WIDTH,TAG_WIDTH, and CACHE_SIZE, which are fixed as per the assignment specifications. The module operates on control signals—clock, reset, read, and write—to perform cache controller operations. Inputs to the module also include the tag and data_in, while outputs include data_out, along with counters for hits and misses.

The main memory is simulated as a byte array, its size is determined by the TAG_WIDTH, with each entry storing data of size DATA_WIDTH. The cache is a smaller array divided into multiple lines, each also of size DATA_WIDTH, and organised for direct mapping.
On reset, all internal variables and memory are initialised to zero.

During a read operation, the module checks if the requested address data is present in the cache by matching the tag and ensuring the valid bit is set. If a cache hit occurs, the stored data is returned; otherwise, a cache miss fetches data from the main memory, updates the cache, and returns the fetched data. Write operations can be performed independently, directly updating the cache.

In the testbench, the main memory is preloaded with test data, while the cache starts empty. Repeated read and write operations simulate typical memory behavior, populating the cache over time and tracking performance through hit and miss counters.

CHALLENGES FACED

<u>1. Mapping Mechanism Between Main Memory and Cache</u>

Mapping main memory blocks to cache lines was challenging, especially in determining block sizes for the tag, index, and offset. Multiple blocks mapping to the same cache line added complexity. In a direct mapping mechanism, this is addressed by using the modulus of the tag with the total number of cache lines.

<u>2. Repeated Access to Test Hit-Miss Ratio</u>

Testing the hit-miss ratio by repeatedly accessing the same address required careful handling. On a cache miss, data had to be fetched from main memory and written correctly to the cache line to avoid repeated misses.

<u>3. Handling Cache Misses</u>

Managing cache misses was essential, as missing data had to be loaded from main memory into the cache. For a cache miss, the system had to retrieve data from the main memory using the tag and load it into the appropriate cache line.

<u>4. Understanding the Basic Structure of a Direct-Mapped Cache</u>

Understanding a direct-mapped cache was challenging, particularly extracting the tag and index from the address, determining their bit sizes, and correctly mapping them to cache lines and main memory. Proper comprehension of these elements was essential to building the cache structure.

PERFORMANCE METRICS

For testing, we designed three test cases, each consisting of 20 sample addresses. The main memory in each test case was initialized to a random hexadecimal value, and the cache was initially empty.

The hit and miss ratio for all of them were as follows:

	Hits	Misses	Delay Time
Test Case 1:	16	4	75ns
Test Case 2:	10	10	95ns
Test Case 3:	16	4	65ns

Note: Since the cache was initially empty, a certain number of misses were required to fill the cache initially. Test cases are available on the GitHub repo (https://github.com/2023csb1141/CS203_Project-2024-)

BIBLIOGRAPHY

https://www.eecs.harvard.edu/cs146-246/cs146-lecture15.pdf

https://csg.csail.mit.edu/6.888Yan/slides/review/L16-Caches.pdf

https://web.stanford.edu/class/archive/cs/cs107/cs107.1174/lect18.pdf