**IUBAT - International University of Business Agriculture and Technology**

**Final Assessment**

CSC 247

**Submitted By**

Name: Sumaiya Haq

ID: 23303137

Section: I

Semester: Summer 2025

Department of Computer Science and Engineering

**Submitted To**

Kazi Johir Raihan Suny

Lecturer, IUBAT

Department of Computer Science and Engineering

| Date of Perform | : 31-08-2025 |
| --- | --- |
| Date of submission | : 10-09-2025 |

## Part A – Cache Mapping

### 1) Working principles of cache mapping techniques:

**Direct-mapped cache**: Every memory block corresponds to a single cache line, which is the structure of the cache. The line is chosen by the mapping using the lower bits of the block number (index). Because each block can only be placed in one location, lookup is quick and easy with this design. Conflict misses are a drawback; even when other cache lines are empty, two frequently used blocks that map to the same line will keep evicting one another.

**Fully associative cache:**Any memory block can be positioned in any cache line thanks to the fully associative cache. The tag of the requested block must be compared to the tags kept in each cache line during a lookup, which is typically completed in parallel. Since there is no set mapping, this reduces conflict misses; however, it necessitates additional hardware (comparators) and may be slower or more costly for large caches. Small caches, such as TLBs or specialized buffers, are frequently fully associative.

**Set-associative cache:** The cache is divided into sets, with multiple lines (ways) in each set. Using index bits, a memory block maps to a single set, but it can occupy any space inside that set. An N-way set-associative cache balances the flexibility of fully associative caches with the ease of use of direct-mapped caches. In comparison to direct mapping, it minimizes conflicts while maintaining a lower hardware cost than fully associative. To determine which line to evict, replacement policies (LRU, FIFO, etc.) are used in every set.

## A.2 Number of tag, index, and block offset bits :

## Given that,

Cache size = 16 KB = 16384 bytes

Block size = 64 bytes = 64 B

Main memory size = 1 MB = 1048576 bytes

**1) Number of blocks:**

**Number of cache blocks** = Cache size / Block size = 16384 / 64 = 256

**Number of main-memory blocks** = Main memory size / Block size

= 1048576 / 64 = 16384

**2) Address bits and field breakdown:**

**Total address bits** = log2(Main memory size in bytes) = log2(1048576)

= 20 bits

**Block offset bits** = log2(Block size in bytes) = log2(64) = 6 bits

**Index bits** = log2(Number of cache blocks) = log2(256) = 8 bits

**Tag bits** = Total address bits - Index bits - Block offset bits = 20 - 8 - 6

= 6 bits

**3) Interpretation:**

1. The block offset (6 bits) identifies the specific byte within a block.
2. The index (8 bits) selects the cache line.
3. The tag (6 bits) identifies which memory block is stored in the cache line.

### Cache line allocation table for blocks 25 to 35

| Block (decimal) | Block (binary) | Block mod 4 | Cache line (block mod 256) |
| --- | --- | --- | --- |
| 25 | 11001 | 1 | 25 |
| 26 | 11010 | 2 | 26 |
| 27 | 11011 | 3 | 27 |
| 28 | 11100 | 0 | 28 |
| 29 | 11101 | 1 | 29 |
| 30 | 11110 | 2 | 30 |
| 31 | 11111 | 3 | 31 |
| 32 | 100000 | 0 | 32 |
| 33 | 100001 | 1 | 33 |
| 34 | 100010 | 2 | 34 |
| 35 | 100011 | 3 | 35 |

**Here**,

1. We used 14-bit binary representation for the block number because the main memory has 16384 blocks (log2(16384)=14).

2. 'Block mod 4' shows how the block number maps modulo 4, which is useful when considering a 4-way set-associative structure (blocks per set).

3. For direct mapping to 256 lines, the cache line is block mod 256. For blocks 25..35 all cache lines equal block numbers because they are less than 256.

## Part B – Replacement Algorithms

3.

Given that,

Reference sequence : 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

Cache size: 4 blocks .

We simulate FIFO, LRU, LFU.

Total references = 12.

### FIFO (First-In-First-Out)

| Reference | Cache state (contents) | Hit/Miss |
| --- | --- | --- |
| 2 | [2] | Miss |
| 3 | [2, 3] | Miss |
| 2 | [2, 3] | Hit |
| 1 | [2, 3, 1] | Miss |
| 5 | [2, 3, 1, 5] | Miss |
| 2 | [2, 3, 1, 5] | Hit |
| 4 | [2, 4, 1, 5] | Miss |
| 5 | [2, 4, 1, 5] | Hit |
| 3 | [2, 4, 3, 5] | Miss |
| 2 | [2, 4, 3, 5] | Hit |
| 5 | [2, 4, 3, 5] | Hit |
| 2 | [2, 4, 3, 5] | Hit |

FIFO

1.hits = 5

2.misses = 7

3.Hit ratio = 5/12 = 41.67%

Miss ratio = 58.33%

**LRU (Least Recently Used)**

| Reference | Cache state (contents) | Hit/Miss |
| --- | --- | --- |
| 2 | [2] | Miss |
| 3 | [2, 3] | Miss |
| 2 | [2, 3] | Hit |
| 1 | [2, 3, 1] | Miss |
| 5 | [2, 3, 1, 5] | Miss |
| 2 | [2, 3, 1, 5] | Hit |
| 4 | [2, 4, 1, 5] | Miss |
| 5 | [2, 4, 1, 5] | Hit |
| 3 | [2, 4, 3, 5] | Miss |
| 2 | [2, 4, 3, 5] | Hit |
| 5 | [2, 4, 3, 5] | Hit |
| 2 | [2, 4, 3, 5] | Hit |

LRU,

1. hits = 6

2.misses = 6.

Hit ratio = 6/12 = 50%

Miss ratio = 50%

### LFU (Least Frequently Used)

| Reference | Cache state (contents) | Hit/Miss |
| --- | --- | --- |
| 2 | [2] | Miss |
| 3 | [2, 3] | Miss |
| 2 | [2, 3] | Hit |
| 1 | [2, 3, 1] | Miss |
| 5 | [2, 3, 1, 5] | Miss |
| 2 | [2, 3, 1, 5] | Hit |
| 4 | [2, 4, 1, 5] | Miss |
| 5 | [2, 4, 1, 5] | Hit |
| 3 | [2, 4, 3, 5] | Miss |
| 2 | [2, 4, 3, 5] | Hit |
| 5 | [2, 4, 3, 5] | Hit |
| 2 | [2, 4, 3, 5] | Hit |

LFU hits = 7

misses = 5

Hit ratio = 7/12 = 58.33%

MIss ratio = 41.67%

## Part C – Computer Performance Impact

Given:

Cache hit time = 2 ns

Cache miss penalty = 50 ns

AMAT Formula:

AMAT=Hit time+(Miss rate×Miss penalty)

| Algorithm | Hits | Miss | Hit Ratio | Miss ratio | AMAT Calculation | AMAT (ns) |
| --- | --- | --- | --- | --- | --- | --- |
| FIFO | 5 | 7 | 41.67% | 58.33% | 2+(0.5833×50)2+  (0.5833×50) | 31.165 |
| LRU | 6 | 6 | 50% | 50% | 2+(0.5×50)2+(0.5×50) | 27 |
| LFU | 7 | 5 | 58.33% | 41.67% | 2+(0.4167×50)2+(0.4167×50) | 22.835 |

## Recommendation and Justification :

## Based on the simulations and AMAT calculations:

## FIFO is simple to implement but resulted in the highest miss rate (58.33%) and the longest AMAT (31.165 ns). It is not ideal for workloads with temporal locality.

## LRU achieved a 50% hit rate and an AMAT of 27 ns. It effectively leverages recent access patterns and is widely used in practice due to its balance between performance and implementation complexity.

## LFU yielded the best hit rate (58.33%) and the lowest AMAT (22.835 ns). It is excellent for workloads with stable, repetitive access patterns. However, it requires additional hardware for frequency counters and may retain stale data.

## Recommendation: For general-purpose workloads, LRU is recommended. It offers a good balance between hit rate, hardware complexity, and adaptability to varying access patterns. LFU is suitable for specialized applications with predictable reuse, while FIFO may be used in extremely resource-constrained environments

## Conclusion

Cache design and replacement policies significantly impact system performance. While LFU achieved the highest hit rate in this specific sequence, LRU remains the most practical choice for general use due to its consistent performance and manageable complexity. Understanding workload characteristics is essential for selecting the appropriate caching strategy.