FR. Conceicao Rodrigues College of Engineering
Department of Computer Engineering

10 .Direct Mapped Cache, Fully Associative Cache, Set associative cache

1. Course, Subject & Experiment Details

Academic Year	**2023-24**	**Estimated Time**	**Experiment No. 11– 02 Hours**
Course & Semester	**S.E. (Computers) – Sem. III**	**Subject Name**	**Digital Logic & Computer Organization and Architecture**
Chapter No.	**5**	**Chapter Title**	**Memory Organization**
Experiment Type	**Software**	**Subject Code**	**CSC304**
Roll No.	**9914**	**Date of Performance**	**21-10-2023**

Rubrics

Timeline (2)	**Practical Skill & Applied Knowledge (4)**	**Output (4)**	**Total (10)**

2. Aim & Objective of Experiment

1. Understanding behaviour of Direct, associative and set associative cache from working module

2. Designing a Direct, associative and set associative cache for given parameters

3. Problem Statement

Show the memory address representation for the main memory and Ram in bits and Solve for tag size,search time for the following configuration using Direct, associative and 2 way set associative mapping: a) MM=128words , RAM=16words ,Block size= 4words

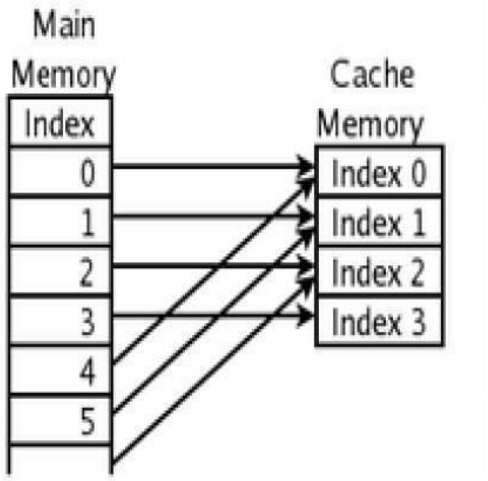
4. Software Required

Cache Simulator

https://www3.ntu.edu.sg/home/smitha/ParaCache/Paracache/dmc.html

5. **Brief Theoretical Description**

DIRECT MAPPED CACHE



DEFINITIONS

- **TAG** – distinguished one cache memory block with another
- **INDEX** – identifies the cache block
- **OFFSET** – points to desired data in cache block

INSTRUCTION BREAKDOWN

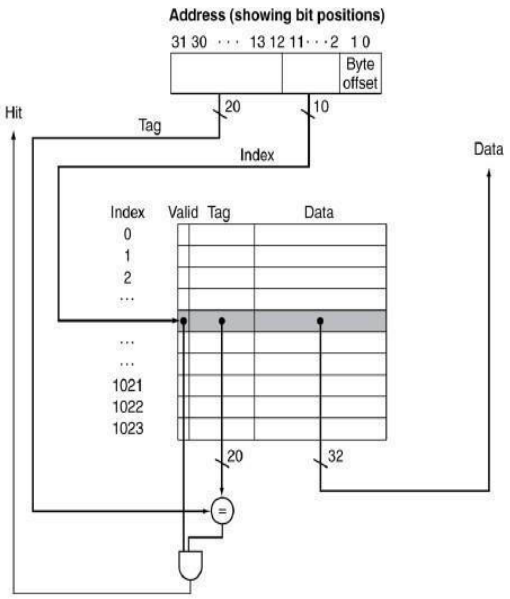
Each of the address of load instruction is broken into three parts: tag, index and offset.
Main memory size = M , cache size = C and offset bits = Z results in the following (right figure).

Length of load instruction	: $\log_2(M)$	**bits**
OFFSET	: z	**bits**
INDEX	: $\log_2(C) - z$	**bits**
TAG	: $\log_2(M) - \log_2(C)$	**bits**
CACHE BLOCKS	: $\log_2(C)$	**blocks**

DIRECT MAPPED CACHE

HOW IT WORKS

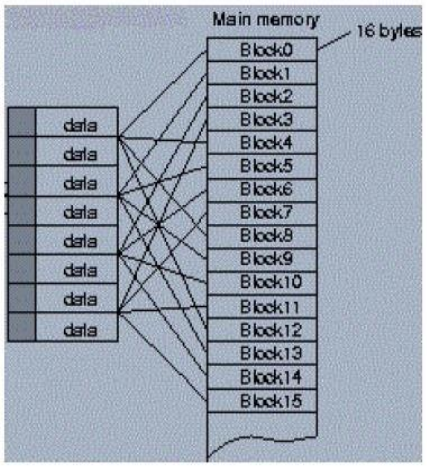
- The requested address is broken down into **tag, index,** and **offset.**
- Cache table with corresponding **index** will be examined.
  - If valid bit of the index is equals to 0, **cache miss** is obtained
  - Else tag bit of the requested address will be compared with tag bit in cache table
    - If tag is matched, **cache hit** is obtained
    - Else **cache miss** is obtained
- When **cache hit** is obtained, data from cache table will be returned. Else, data will be retrieved from main memory.



PRO / CONS

Direct mapping is simple and inexpensive to implement, but if a program accesses 2 blocks that map to the same line repeatedly, the cache begins to thrash back and forth reloading the line over and over again leads to high miss rate.

FULLY ASSOCIATIVE CACHE



DEFINITIONS

- **TAG** – distinguished one cache memory block with another
- **OFFSET** – points to desired data in cache block

INSTRUCTION BREAKDOWN

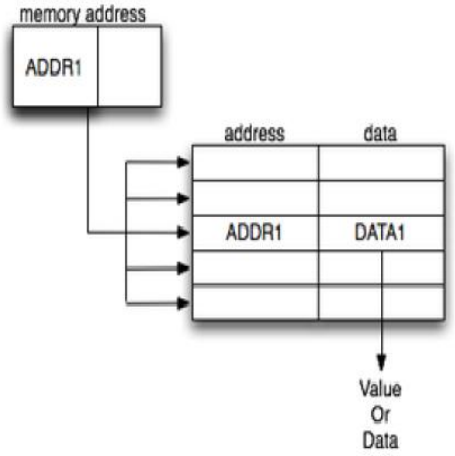
Each of the address of load instruction is broken into three parts: tag and offset.
Main memory size = M , cache size = C and offset bits = Z results in the following (right figure).

Length of load instruction	: $\log_2(M)$	**bits**
OFFSET	: z	**bits**
TAG	: $\log_2(M) - \log_2(C)$	**bits**
CACHE BLOCKS	: $\log_2(C)$	**blocks**

FULLY ASSOCIATIVE CACHE

HOW IT WORKS

- The requested address is broken down into **tag** and **offset.**
- Requested **tag** will be searched through cache with valid bit
  - If there is **tag** matched with one of the index in the cache table, **cache hit** is obtained
  - Else, **cache miss** is obtained
- When **cache hit** is obtained, data from cache table will be returned. Else, data will be retrieved from main memory.

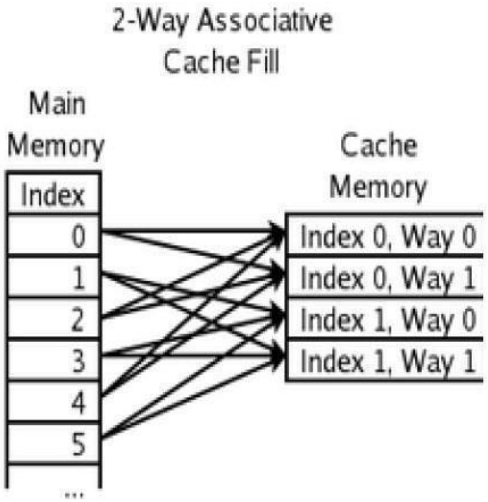


PRO / CONS

Fully Associative is the most efficient utilisation of cache blocks, yet it is expensive to transverse through the cache to find each requested tag. Since there is no specified slot for each instruction, an algorithm of replacement policy must be designed along with implementation of fully associative cache.

[N-WAY] SET ASSOCIATIVE CACHE

N-way set associative cache combines the idea of direct mapped cache and fully associative cache. It has direct mapped principle to an index, yet fully associative concept within the index. **An index** contains **N** blocks of way.



INSTRUCTION BREAKDOWN

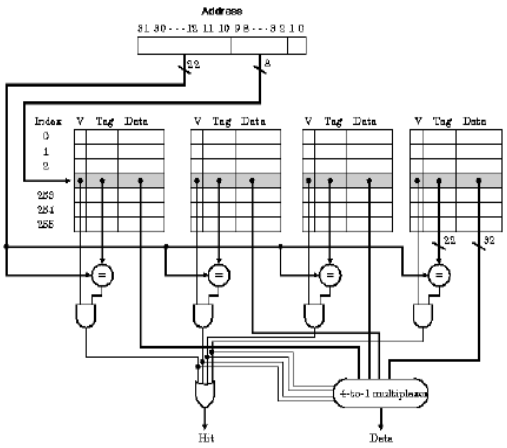
Each of the address of load instruction is broken into three parts: tag, index and offset.
N-way set associative, Main memory size = M , cache size = C and offset bits = Z results in the following (right figure).

Length of load instruction	: $\log_2(M)$	**bits**
OFFSET	: Z	**bits**
INDEX	: $\log_2(C) - z - \log_2(N)$	**bits**
TAG	: $\log_2(M) - \log_2(C) + \log_2(N)$	**bits**
CACHE BLOCKS	: $\log_2(C)$	**blocks**

[N-WAY] SET ASSOCIATIVE CACHE

HOW IT WORKS

- The requested address is broken down into **tag, index** and **offset.**
- Cache table with corresponding **index** will be examined. **N-ways** of cache blocks will be transversed.
  - If one of the **way** has the requested index, a **cache hit** will be obtained
  - Else **cache miss** is obtained
- When **cache hit** is obtained, data from cache table will be returned. Else, data will be retrieved from main memory.

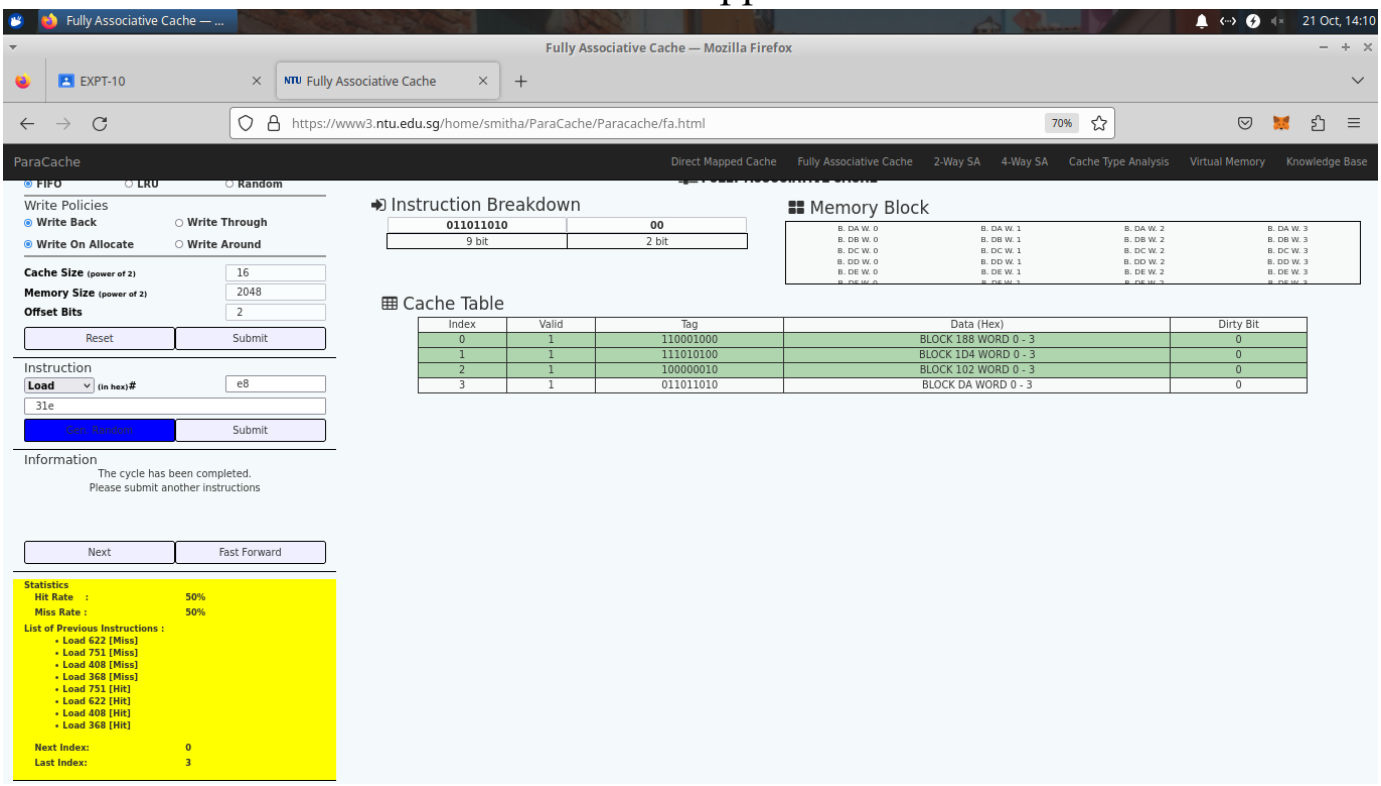


PRO / CONS

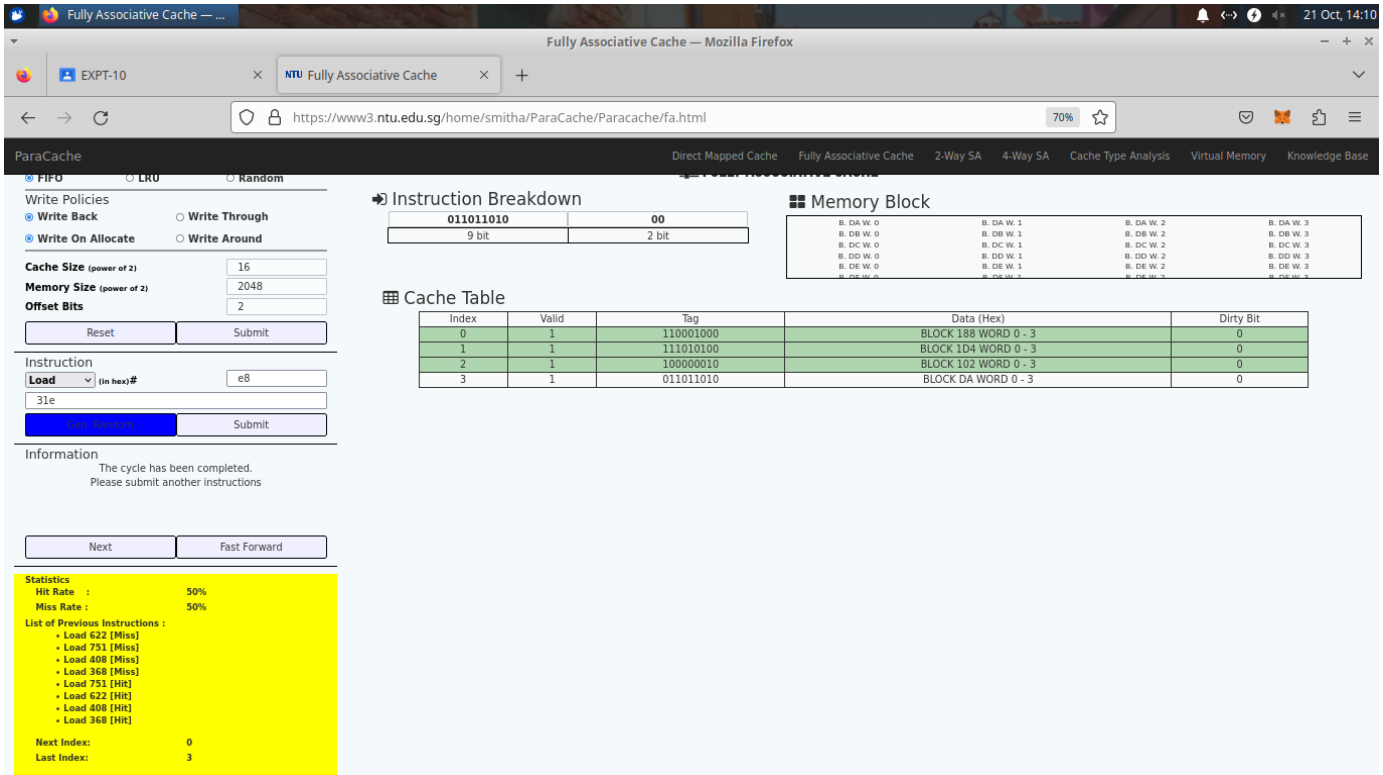
Combining direct mapped and fully associative principle is seen as the most balanced way to obtain high hit rate meanwhile maintaining the resources cost. However, N-Way associative could be hard for initial implementation as various concepts is involved.

6. ***Attach the screenshot:***

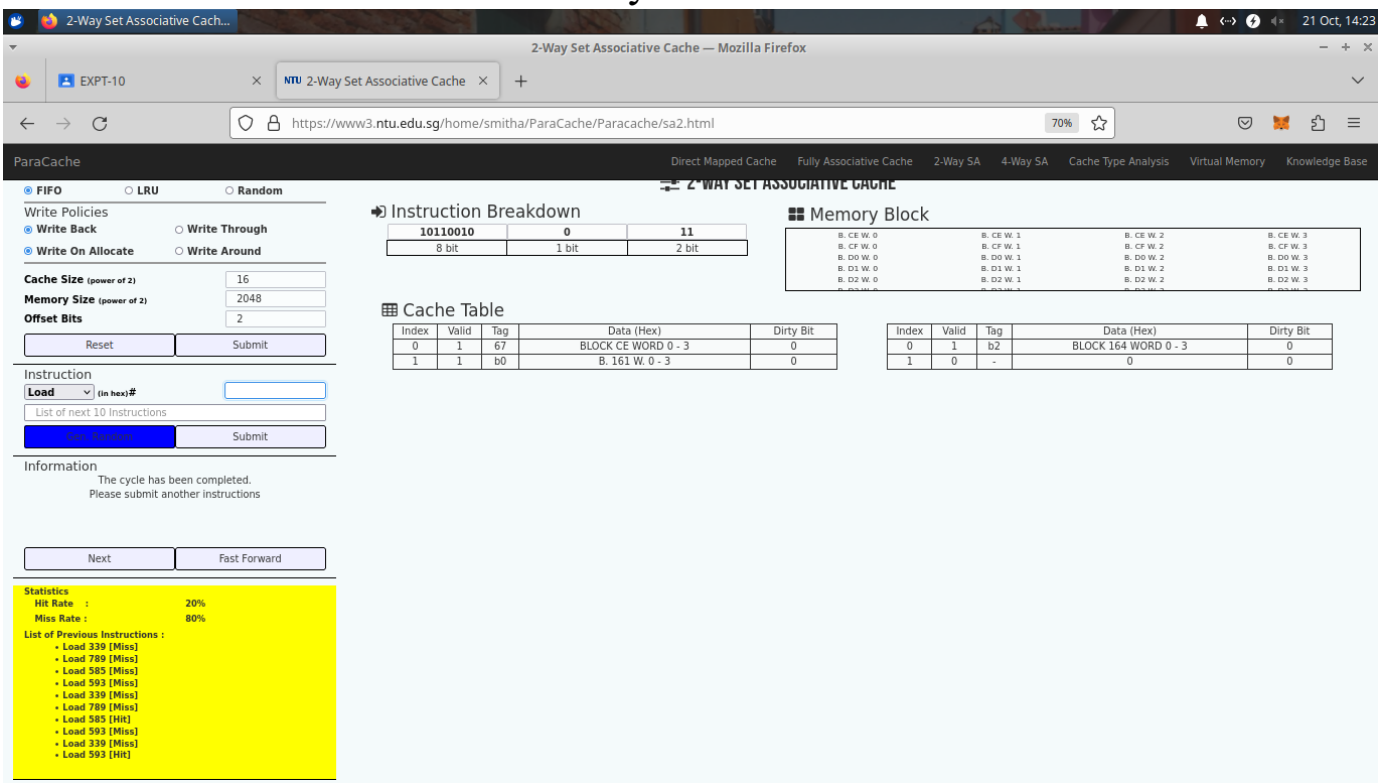
Direct Mapped Cache



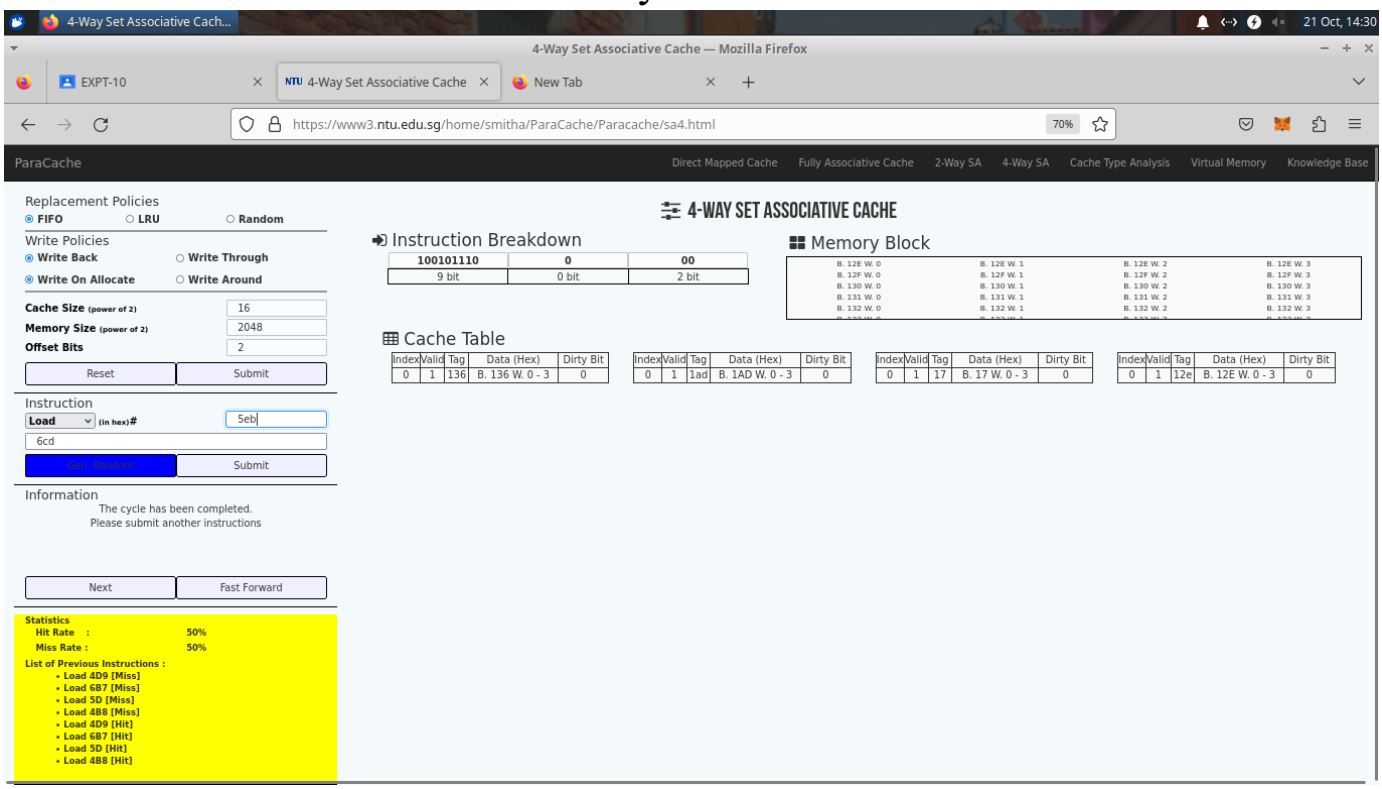
Fully Associative



2-Way Set Associative



4-Way Set Associative



7. **Conclusion:**

We understood of the behavior of different cache mapping techniques: Direct Mapped, Fully Associative, and Set Associative Caches. We designed these cache structures based on given parameters and explored their characteristics.