Cache controller

What is cache memory:

It is general principal that, "*Large memories are slow, small memories are fast"*.

Main memory is slow because of large size. To increase microprocessor read and write frequency, we need some intermediate memory, which should be small but fast. This memory is called cache memory.

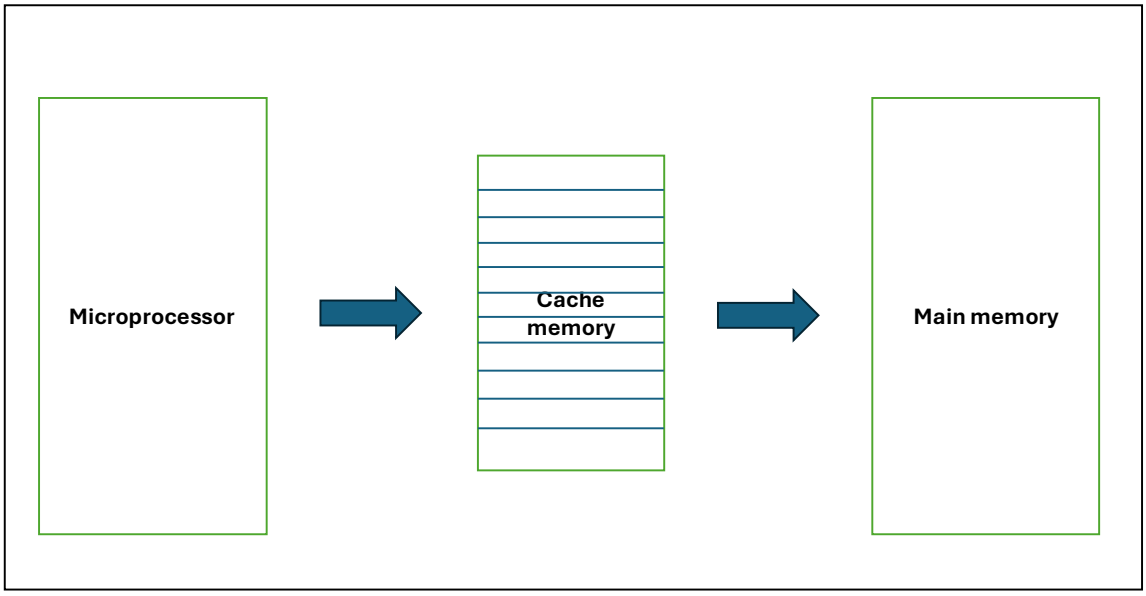

Fig 01: Cache in microprocessor system

The microprocessor first searches for data in this small memory. This search operation is fast because of the small volume. If required data is not available in small memory, it then goes to main memory and searches data over there.

If the required data is found out in the cache memory, it is called *'cache hit'*. If data is not available in cache memory, then it is called *'cache miss'*.

Cache coherency:

It is common practice that microprocessors will need data closer to the previous data it has read or written. That is why we place block of data in cache from main memory so that maximum cache hit occurs.

There are two types of coherency:

1) Spatial coherency:
2) –

Working of cache memory:

When microprocessors need some data, it gives cache read signal and address for the data. If data is available against that address, cache asserts the read hit signal and return microprocessor the required data. But if the data is not available at the provided address, cache asserts the cache miss signal and requests the main memory for the required data. The main memory returns it a block of data according to the cache coherency policy. After this data is successfully loaded in cache, the required line of data is sent to the microprocessor.

When microprocessors need to write some data in cache, it gives write signal, address and data to be written, if that address is already present in cache, write hit asserts and microprocessor data is simply overwritten over that address. If that address is not present in the cache, write miss asserts and block of data (with addresses) is overloaded from main memory to cache and then the data from microprocessor is placed at the provided address (This is 'write allocate' policy and we will discuss that latterly).

Cache addressing:

The address from the microprocessor is divided into three parts (size depending upon the cache architecture)

1) Index: This part decides the line/row of cache memory to/from which write/read occur.
2) Tag: This part is the unique number for specific data set.
3) Offset: This part is used to select byte in word or words in multiple words (depending upon the size of the data line)

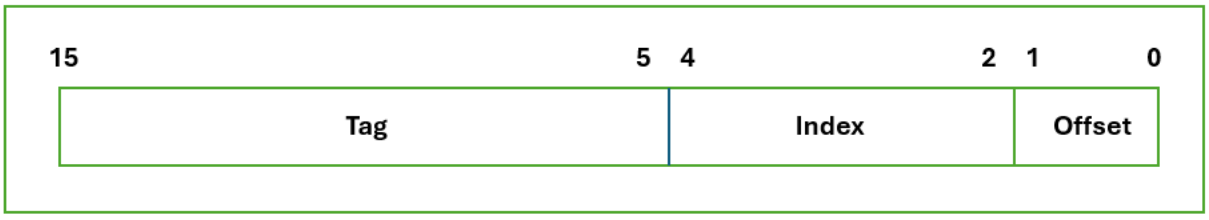

Fig 02: Address partitioning

The index bits are decided based on the number of lines in cache block (in case of set associative or direct mapped).

Cache placement policies:

Direct mapped:

I direct mapped cache, the tag in the address is matched with the cache tag at the specified index. If it is there, then the cache is hit otherwise miss. The advantage of this policy is that there is no comparator (only one) and the disadvantage is high no of the cash miss.

Fully associative:

In fully associative cache, the tag in the address is matched with each line of cache. If matches with any line, then cache is hit otherwise miss. The advantage is high no cache hit, and the disadvantage is high no of comparators.

Set associative:

In set associative cache, the cache is divided into ‘n’ number of blocks. Each block has ‘m’ no lines. Based on index, one block is chosen. In that block, tag is compared with each line, if tag of address matches with any line, then cache is hit otherwise miss. Using this policy, we decrease the number of comparators and cache thrashing (misses).

Cache write policies:

Write through:

In cache write through policy, whenever new data is written into cache, that data will also be written/updated in main memory at the same clock cycle.

Write back:

in cache write back policy, data is written in cache but not updated in main memory in the same cycle. But when the same data needed to be erased/evacuated, it will first be updated in main memory and then evacuated from cache. Its advantage is less time elapses in data transfer to main memory.

Cache replacement policies:

FIFO:

In FIFO policy, the first line used will be evacuated first (if needed). This policy will be used in a fully associative or set associative cache.

LRU:

In LRU policy, The Lease recent updated line will be evacuated first (if needed). This policy will be used in a fully associative or set associative cache.

Specifications for my Project:

General Cache Parameters

- Address Size: 16-bit memory addresses.
- Cache Size: 8 Cache Lines (for both fully associative and 4-way set-associative).
- Cache Line Width: 32 bits (4 bytes per line).
- Total Cache Storage:
    - Fully Associative: 8 lines × 4 bytes = 32 bytes total.
    - 4-Way Set-Associative: 4 sets × 2 lines/set × 4 bytes = 32 bytes total.
- Fully Associative Cache

Since there are only 8 cache lines, the tag needs to be:

- Tag = 16-bit address (all bits used as tag, no index).
- Least recent used Replacement policy.
- Write- through + No write allocate policy.

Block Diagram and signals:

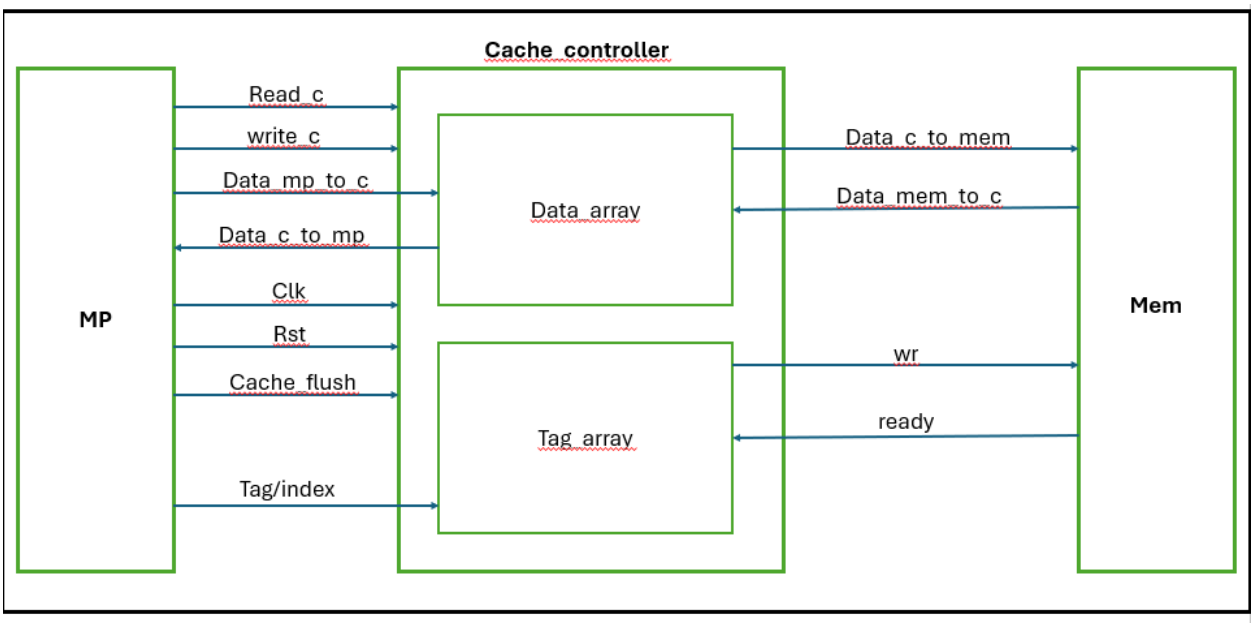

Fig 03: Main block Diagram and input/output signals

Finite state machine:

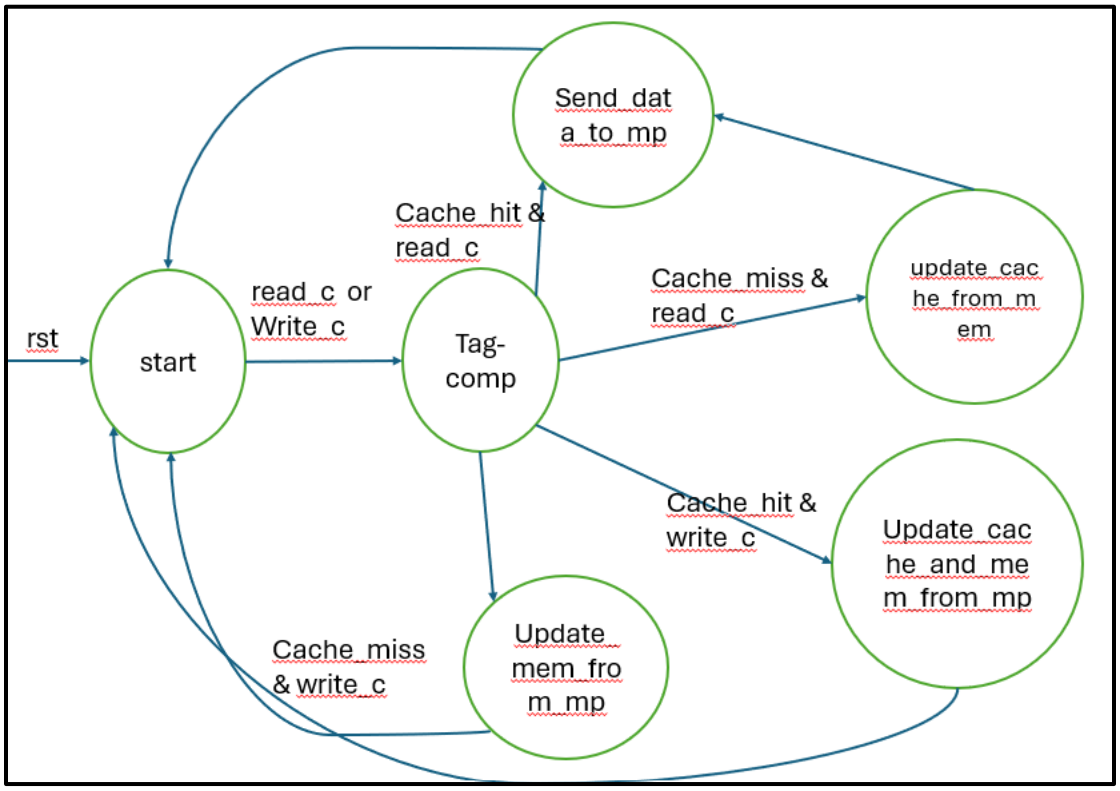

Fig 04: Finite state machine

Simulation results:

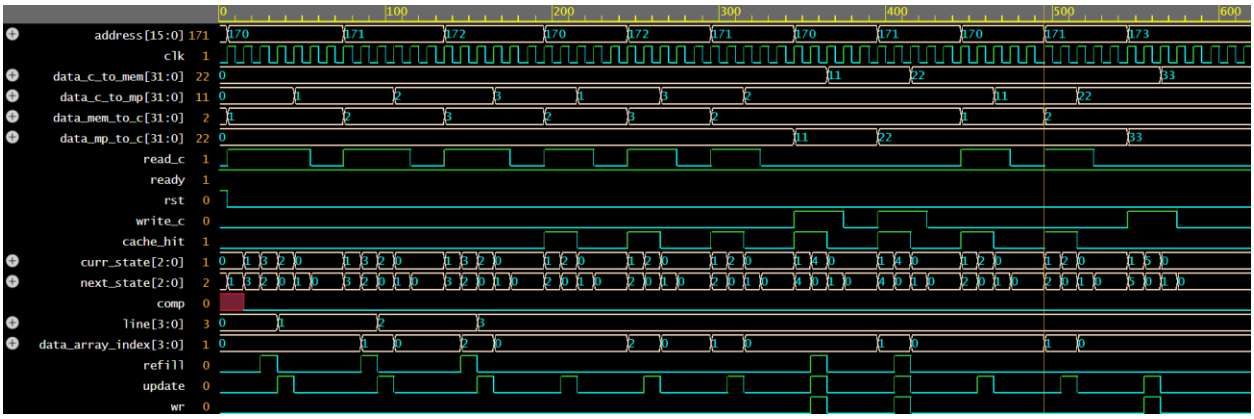

Fig 05: EdaPlayground simulation waveform

Project link:

‘edaplyground’ link is given as below.

https://www.edaplayground.com/x/PjST