PROJECT REPORT: TWO – LEVEL CACHE CONTROLLER WITH SET ASSOCIATIVE MAPPING

12 Aug 24

Made by: -

Amit Chaudhari

23M1141

Department of Electrical Engineering

IIT Bombay

Introduction:

In modern computing systems, the performance and efficiency of memory access are critical for achieving high processing speeds. A two-level cache controller plays a vital role in optimizing data retrieval times, ensuring that the central processing unit (CPU) has quick access to frequently used data. Caching mechanisms are designed to reduce the time it takes to access data from the main memory, thereby improving overall system performance.

This project focuses on the design and implementation of a two-level cache controller utilizing set-associative mapping. Set-associative mapping is a compromise between direct-mapped and fully associative caching, offering a balanced approach that reduces the likelihood of cache misses while maintaining manageable complexity in cache design. In a set-associative cache, the cache is divided into several sets, each containing multiple lines (or blocks) where data can be stored. The cache controller uses the memory address to determine the specific set to search, and within that set, it checks for the presence of the required data. This method allows for multiple potential locations for a single data item, which reduces the chances of collision and increases the hit rate.

Techniques Used:

1. Set Associativity: A cache mapping technique where each block of memory can be placed in any one of a fixed number of locations (a set), reducing the likelihood of collisions compared to direct mapping.

2. Look Through: A cache access method where data is first searched in the cache; if it's not found (a miss), the request is passed on to the next level of memory or main memory.

3. Write Back: A caching technique where data modifications are only written to the cache and updated in main memory when the cache line is evicted, reducing the number of write operations to memory.

4. Write Allocate: A cache policy where on a write miss, the data block is first loaded into the cache from main memory, and then the write operation is performed on the cache.

5. Exclusive Cache: A caching strategy where data is stored only in one cache level (either L1 or L2, but not both simultaneously), ensuring that each cache level contains unique data to maximize the overall cache capacity.

6. Least Recently Used: A cache replacement policy that evicts the data block that has not been accessed for the longest period of time, assuming that recently used data is more likely to be used again soon.

Specifications:

These specifications can be easily modified by modifying parameters.

	L1 cache	L2 cache
Block size	4 bytes	4 bytes
Number of blocks	32	128
Associativity	2-way	4-way
Number of sets	16	32
Total size	128 bytes	512 bytes
Tag size	26 bits	25 bits
Set index	4 bits	5 bits
LRU overhead per block	1 bit	2 bits
Latency	1 clock cycles	4 clock cycles

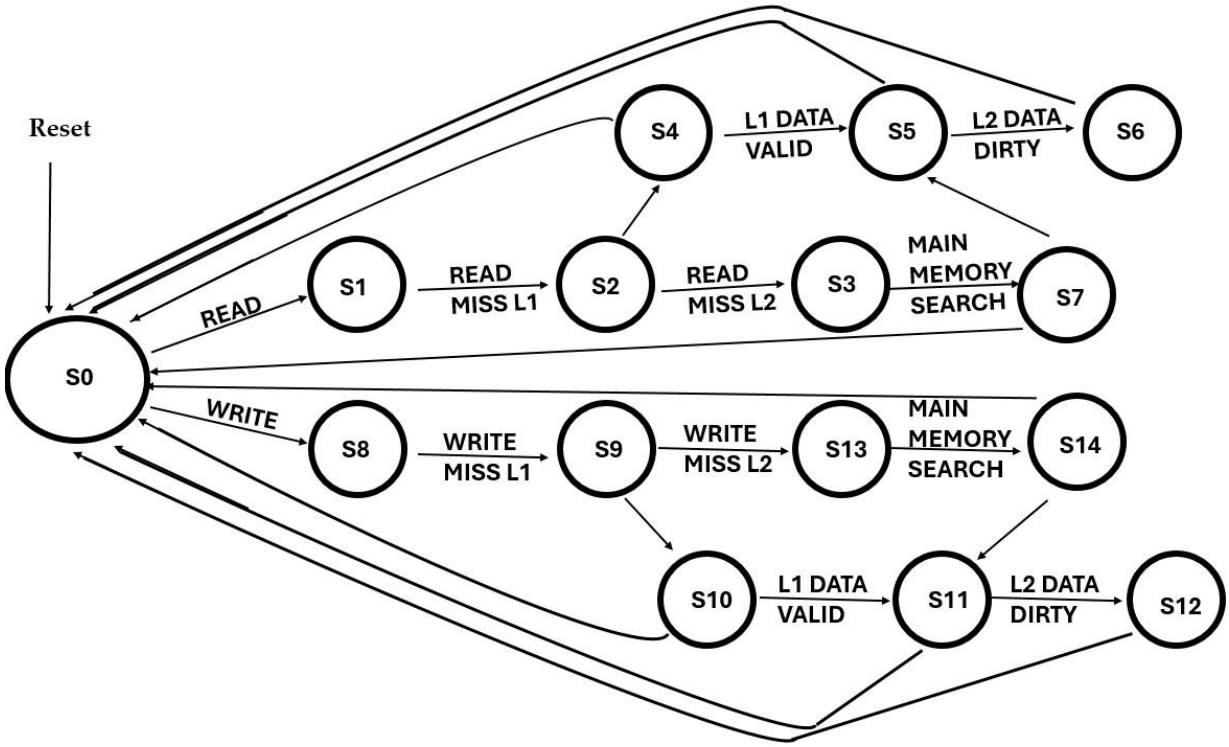
Address size **:** 32 bits

Byte offset **:** 2 bits

Main memory size **:** 1024*4 = 4kB

Main memory latency: 10 clock cycles

Finite State Machine for cache controller:



State definitions:

S0: Decode whether processor requests read or write and stores the operation mode, address and data (if write operation) in a buffer.

S1: Search for requested data in L1 cache in appropriate set. Set hit1 signal if found. Update LRU bits.

S2: Search for requested data in L2 cache in appropriate set. Set hit2 signal if found. Update LRU bits. Write data in a buffer.

S3: Search for requested data in main memory. Write requested data in a buffer.

S4: Upgrade L2 data to L1, if LRU data in L1 is valid, put it in a buffer.

S5: Place L1 evicted data into L2, if LRU data in L2 is dirty, put it in a buffer.

S6: Update L2 evicted data in main memory.

S7: Upgrade data from main memory into L1, if LRU data in L1 is valid, put it in a buffer.

S8: Search for requested data in L1 cache in appropriate set. Set hit1 signal if found. Update LRU bits. Write data in the found block.

S9: Search for requested data in L2 cache in appropriate set. Set hit2 signal if found. Update LRU bits. Write data in a buffer.

S10: Search for requested address in main memory.

S11: Upgrade data from main memory into L1, if LRU data in L1 is valid, put it in a buffer. Write data in the block.

S12: Upgrade L2 data to L1, if LRU data in L1 is valid, put it in a buffer. Write data in the block.

S13: Place L1 evicted data into L2, if LRU data in L2 is dirty, put it in a buffer.

S14: Update L2 evicted data in main memory.