2 LEVEL CACHE

The code shows the implementation of a two level cache system

INPUT FORMAT

1. The type of cache mapping: (DIRECT MAPPING), (FULLY ASSOCIATIVE), (SET ASSOCIATIVE)
2. The number of cache lines (should be of the power 2)
3. The block size (should be of the power 2)
4. Select from the following option:
    a. WRITE (ADDRESS) (VALUE): WRITE command followed by integer address and value to be stored at the address
    b. READ (ADDRESS): READ command followed by the address from where the data needs to be read.
    c. CACHE: CACHE command to print the contents of cache
    d. PCACHE: PCACHE command to print the contents of primary cache
    e. SCACHE: SCACHE command to print the contents of secondary cache
    f. CLEAR: CLEAR command to clear the screen contents
    g. EXIT: EXIT command to exit from the program

Note: For k-way set associative the value of k is also needed to be input

OUTPUT FORMAT

1. Output for WRITE command:
    a. Cache tag hit/Cache tag miss in primary/secondary cache : Depending on cache hit or cache miss the either of four combinations will be printed.
    b. Indicator that flags if the cache is full and the value which is deleted according to the mapping chosen. This is also depicted for both the cache
2. Output for READ command:
    a. Cache tag hit/Cache tag miss: Depending on cache hit or cache miss the either of the two will be printed for both primary and secondary cache.
    b. Indicator that flags if the cache is full and the value which is deleted according to the mapping chosen.
    c. The output of the query: Address location -> value at the address
    d. Gives an exception for the query if the address doesn't exist in the main memory.
3. Output for CACHE command:
    a. Prints the cache contents of both the cache with the block number along with additional information which depends on the type of mapping chosen. It prints the block number followed by memory in the block
        i. Fully associative: Cache lines
        ii. Direct Mapping: Cache lines

        iii. Set associative mapping: Set number and cache lines in the set in which the blocks are present
    b. Prints Cache empty if there is nothing in the cache
4. Output for PCACHE command:
    a. Prints the contents of the Primary cache of size S/2 with block number along with additional information which depends on the type of mapping chosen.
        i. Fully associative: Cache lines
        ii. Direct Mapping: Cache lines
        iii. Set associative mapping: Set number and cache lines in the set in which the blocks are present
5. Output for SCACHE command:
    a. Prints the contents of the secondary cache of size S with block number along with additional information which depends on the type of mapping chosen
        i. Fully associative: Cache Lines
        ii. Direct Mapping: Cache lines
        iii. Set associative mapping: Set number and cache lines in the set in which the blocks are present

FUNCTIONING OF CACHE SYSTEM

The following explains the functioning of the cache system that is implemented for different types of mapping

Type of mapping and their cache implementation:

1. **Fully associative mapping:** For the WRITE operation, the input of address and value is given at console. The corresponding block tag and data offset is calculated and stored. Then the primary cache is checked for the block tag in the cache. If the cache is found in the primary cache, it is a cache hit. If the block already contains the memory address, the value is updated. If not, then the value is added to the array. If the block is not found in primary cache but found in secondary cache, “cache hit in secondary cache” is printed. The value is removed from the secondary cache and inserted into the primary cache. If the primary cache is full then applying the LRU policy, the element is popped from the cache accordingly. If not, then the block removed from the secondary cache is added to the primary cache. In the case an element is removed from the primary cache, it is added to the secondary cache taking into consideration whether the cache is full.

2. **Direct mapping:** For WRITE operation, the input of address and value is given at console. The corresponding block tag and the cache line in which the block needs to go is calculated. If the block already exists at the cache line, then it is checked for the memory address. If it already contains the memory address, the value is updated. If not, the value is added to the block, showing the cache hit. If the block is not present in the primary cache, but it is present in the secondary cache at the block tag, it is fetched from

the secondary cache and put into the primary cache. If the index already contains a block, the older block is replaced by the newer block. The older block is added to the corresponding index in the secondary cache taking into account whether another block exists at that index or not. Similarly is done for READ command

3. **K-way set associative mapping:** For WRITE operation, the address and value are given as input on the console. The address is broken into a block address, data offset, and the set where the memory needs to be appended. The corresponding set is searched for the block following fully associative mapping inside the set in the primary cache. If the block is found, then it is a cache hit else cache miss in the primary cache. If the block is found in the secondary cache, then the corresponding primary cache set is calculated, and the set is updated in the primary cache. If there is an element pop from the primary cache, then it is fitted in the secondary cache by calculating the appropriate set and then appending it to the set's cache line. If there is a cache miss in both the caches, then the new block is added to primary cache following which the block delete (if all cache lines inside the set are full) is added to secondary cache. The same strategy is followed for the READ command with an exception that the value is not updated or appended.

CODE DOCUMENTATION:

CLASS NAME	DATA MEMBERS	MEMBER FUNCTIONS
Memory	1. memoryVal 2. memoryAdd 3. memoryTag 4. dataTag 5. timeStamp	1. printMemory()
Block	1. blockSize 2. blockTag 3. crntMemoryCount 4. memoryArr	1. insertInMemoryArr() 2. updateInMemoryArr() 3. isInBlock() 4. getFromBlock() 5. printBlock()
Fully Associative	1. cacheLinesPrimary 2. cacheLinesSecondary 3. blockSize 4. cachePrimary 5. cacheSecondary 6. crntCacheSizePrimary 7. crntCacheSizeSecondary 8. localTimeCheckingPrimary	1. isInPrimaryCache() 2. isInSecondaryCache() 3. updateLocalTimeCheckingPrimary() 4. updateLocalTimeCheckingSecondary() 5. writeToCache() 6. readCache()

	9. localTimeCheckingSecondary	7. printPrimaryCache() 8. printSecondaryCache() 9. printCache()
DirectMapping	1. cacheLinesPrimary 2. cacheLinesSecondary 3. blockSize 4. crntCacheSizePrimary 5. crntCacheSizeSecondary 6. cachePrimary 7. cacheSecondary	1. isInPrimaryCache() 2. isInSecondaryCache() 3. isIndexInPrimaryCache() 4. isIndexInSecondaryCache() 5. writeToCache() 6. readCache() 7. printPrimaryCache() 8. printSecondaryCache() 9. printCache()
SetObject	1. cacheLines 2. blockSize 3. setSize 4. LocalTimeChecking 5. crntCacheSize 6. setCache	1. isInCache() 2. updateLocalTimeCheckingArr() 3. writeInSetCache() 4. readFromSetCache() 5. printCache()
SetAssociative	1. cacheLinesPrimary 2. cacheLinesSecondary 3. blockSize 4. K 5. numberOfSetsPrimary 6. numberOfSetsSecondary 7. crntSetCountPrimary 8. crntSetCountSecondary 9. cacheLinesPrimarySet 10. cacheLinesSecondarySet 11. cachePrimary 12. cacheSecondary	1. isBlockInPrimaryCache() 2. isBlockInSecondaryCache() 3. arrangeInSecondaryCache() 4. writeToCache() 5. readCache() 6. printPrimaryCache() 7. printSecondaryCache() 8. printCache()

ERRORS REPORTED:

1. The input block size and cache lines should be of power 2
2. If the cache is empty and there is a read command then it will give error
3. If the cache is empty and print command is given then it will give error depending upon which cache is printed

LEAST RECENTLY USED POLICY(LRU): According to LRU policy, the memory block, which is the oldest in the cache is deleted first in case the cache is full, and there is a command request. The oldest block is the one that was created or updated before the other blocks in the cache. This can be understood concerning timestamps. The block with the smallest timestamp is the least recently used in the cache. LRU policy can also be understood using FIFO that is first in first out or a queue.

ASSUMPTIONS:

1. Both the cache has same type of mapping
2. The value of k is same in k-way set associative mapping