SINGLE LEVEL CACHE

The code shows the implementation of a single 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. CLEAR: CLEAR command to clear the screen contents
    e. 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: Depending on cache hit or cache miss the either of two will be printed.
    b. Indicator that flags if the cache is full and the value which is deleted according to the mapping chosen.
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.
    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 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 line in the set in which the block is present
    b. Prints Cache empty if there is nothing in the cache

<u>FUNCTIONING OF CACHE SYSTEM</u>

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 cache is checked for the block tag obtained. If the block tag is found in the cache, then it is a cache hit, and the value is inserted or updated in the block. If the block tag is not found, then it is a cache miss. If the cache has space, then the value is appended in the space with appropriate labeling of block tags. If the cache is full, then using the LRU policy, the value is deleted from the cache, and a new value is inserted in the place of the deleted value. For the READ operation, the address is taken as input. The corresponding block tag is found, and the block is searched in the cache. If it is found in the cache, it is a cache hit. The value is printed corresponding to the memory location. If it is not in the cache, then the value is looked up in the main memory, and if found, it is pushed into the cache. The output is printed, and the cache is updated.

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 tag is found at that cache line, it is considered as a cache hit. The value is inserted or updated in the block accordingly. If the block tag is not found, then it is a cache miss. If the cache line where the block needs to be appended is empty, then the block is just added there with appropriate memory contents. If the block already contains another block, the block is replaced by the new entry. For READ operation, if the block exists in the cache line, then it is a cache hit, and output is printed. If not, then the block is fetched from the main memory and written in the cache.

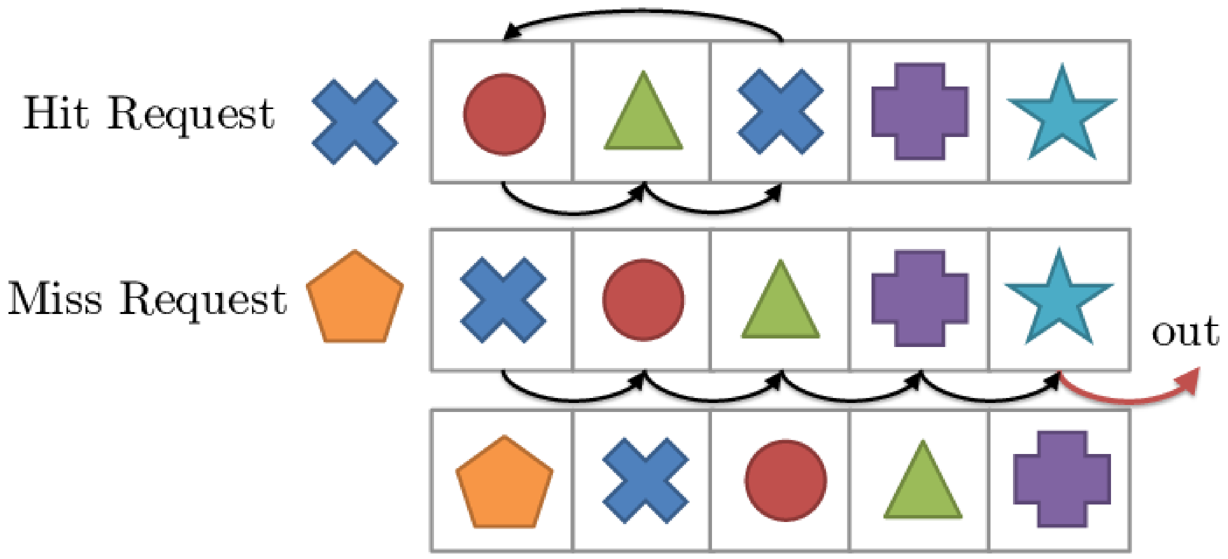
3. **K-way Set Associative Mapping:** For WRITE operation, the address and value are given as input on the console. The address is broken into a block address, data offset, and the set where the memory needs to be appended. The corresponding set is searched for the block following fully associative mapping inside the set. If the block is found, then it is a cache hit else cache miss. The procedure inside the set is the same as that of fully associative mapping. In the READ operation, the address is fed as an input. The address is looked up in the set, and if not present, it is fetched from the main memory. If the address is not assigned, it gives an error exception.

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()
MainMemory	1. mainMemoryDict	1. isInMainMemory() 2. addToMainMemory() 3. getFromMainMemory 4. updateInMainMemory 5. printMainMemory()
FullyAssociative	1. cacheLines 2. blockSize 3. cacheSize 4. crntCacheSize 5. cache	1. isInCache() 2. writeToCache() 3. readCache() 4. printCache()
DirectMapping	1. cacheLines 2. blockSize 3. cacheSize 4. crntCacheSize 5. cache	1. isInCache() 2. isIndexInCache() 3. writeToCache() 4. readCache() 5. printCache()
SetObject	1. cacheLines 2. blockSize 3. setSize 4. LocalTimeChecking 5. crntCacheSize 6. setCache	1. isInCache() 2. updateLocalTimeChecking Arr() 3. writeInSetCache() 4. readFromSetCache() 5. printCache()
SetAssociative	1. cacheLines 2. blockSize 3. K 4. numberOfSets	1. isSetNumInCache() 2. writeToCache() 3. readCache() 4. printCache()

	5. crntSetCount 6. cache	
GLOBAL	1. TimeChecking 2. MainMemoryArr	1. UpdateInTimeCheckingArr 2. isBlockInTimeChecking 3. Main()

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.



RESOURCES:
https://www.google.com/search?q=lru+policy&rlz=1C1CHBF_enIN876IN876&source=lnms&tbm=isch&sa=X&ved=2ahUKEwihkJTA-8TpAhXg4jgGHcRYBjoQ_AUoAXoECAwQAw#imgrc=5G_3qWChp7_V2M

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

ASSUMPTION

1. Main Memory is of size 4 mb