Cache Project

Nandika Jain - 2019064 (Sec-A, Group-8)

The project implements 3 different types of cache mapping:

1. Direct mapping
2. Associative Memory Mapping
3. n-way Associative Memory Mapping

- **How to run the program?**
    - Open terminal in the directory containing the source file.
    - Enter command: python3 CacheProject.py or py CacheProject.py
    - If you have changed the name of the file from CacheProject.py to something else, you will have to use that instead of CacheProject.
    - python3 or py is the environment variable to compile and run using python. If you have a different variable, you will have to use that
    - The code will be executed with the above command.
    - The program uses a library tabulate from python. If the program is giving errors before starting execution, you need to have this library installed on your system. Follow the instructions on https://pypi.org/project/tabulate/ to do so.

- **Assumptions**
    - The program will keep asking the user to enter the number of cache lines until it receives a valid input, that is, a power of 2.
    - The program will keep asking the user to enter the block size until it receives a valid input, that is, a power of 2.
    - The user enters the size of the block in bytes.
    - The size of one word is taken to be 4 bytes.
    - Word length is assumed to be 32 bits,i.e. it is a 32 bit system with 32 bit addressing.
    - Size of the block entered by the user should be greater than or equal to 4 since the size of each word is taken to be 4 bytes.
    - The value of n entered by the user in a n-way Associative Memory Mapping is a power of 2.
    - The number of queries given by the user is in decimal form.
    - The user enters the address to be read or written in binary form and it is converted to decimal form.

- When the cache structure is printed, in a fully associative and n-way set associative, the address is converted to decimal and it also prints the value of the LRU counter.
- If the cache line is not written to, it gives a value of None.
- If a user enters an address greater than 32 bits, the higher bits with index greater than 31 are considered as an overflow and not taken into account for computing the tag address.
- If a user enters an address less than 32 bits, the MSB's are all considered to be 0 and addressing is done in accordance to that.
- An invalid query is also counted as a query.
- Once a user writes to the cache line, that is to a word in the cache line, the rest of the data in the particular block is initialized to 0. So in case of a read operation, if any word in the block is written to, not necessarily the word in the address, it shows a Cache Hit since the concept of cache hit and miss is based on the presence of the required block in the Cache and in the real cache some garbage values are always present.
- LRU Replacement Policy used for n-way Associative Memory Cache and Associative Memory Mapping:
  The concept of LRU(Least Recently Used) is used for replacement in the n-way and fully associative set mapping. When a new block is added, it has the maximum LRU counter,i.e. the last one to be evicted due to the idea of the temporal locality that if a block was used recently, it is likely to be used in the near future as well. Overwriting/Writing to a different word in a block or reading words from the same block increments the LRU counter.

- **Driver**
  The driver function first asks the user for the number of cache lines and block size.
  Then the user has the option to choose from one of the 3 cache mappings.

- **Mappings**
  For each mapping, the user first has to enter the number of queries they want to have.
  For each query, they have the choice to either read data, write data or print the entire cache. Invalid choices are counted as queries.
  For n-way Associative mapping, the user additionally has to input the value of n which should be a power of 2.

- **Queries**
  For read and write operations, the user must give the address of the data in binary format. For write operations, the data to be written must be in decimal format.

- **Output**
  For each query of type **read**, the program prints “Cache Miss!” if the block of the corresponding address provided by the user is not present. If the block is present there, the program prints “Cache Hit!”, followed by the data stored at that address.
  For each query of type **write**, the program takes in the data to be written and writes it to the desired block. In case of a replacement, it prints the tag(in decimal) of the block being replaced.
  For each query of type **print**, the program prints the entire cache structure which includes the data, tag and the cache line where it is present. In case of fully associative and n-way cache mapping, it prints the value of the LRU counter along with the tag.

- **Errors Handled**
  - It handles the error where the user doesn’t enter CL or B in powers of 2 by asking the user to enter another value until a valid input is given.
  - It handles the error where the user enters the block size less than 4 bytes, since a word is taken to be of 4 bytes. It handles this error by asking for input until a valid input is given by the user.
  - It handles the error of an invalid choice entered by the user at the time of selecting the type of cache mapping desired.
  - It handles the error of an invalid query at the time of a particular mapping.
  - If the user enters an address of greater than 32 bits, it doesn’t throw an error and considers the lower 32 bits and considers the rest as an overflow.
  - If the user enters an address less than 32 bits, it considers the higher bits to be 0’s till the 32nd bit and doesn’t throw any errors.

- **Files**
  CacheProject.py - The main file of the project. Implements the project using Python 3.
  Documentation_Cache.pdf - Contains instructions on how to run the program and explains the code.

Brief explanation of the three mapping techniques that have been implemented:

Direct Mapped Cache :

In direct mapping, the data in the address inputted by the user can only reside in one specific cache line which is done by computing the cache line through the address. In case another address has the same mapping to the same cache line, it will replace the existing block even though there might be space in the rest of the cache. Thus, **hit ratio here is low** but locating the cache line where the desired block would be present is done in **constant time**.

Associative Memory Mapping :

In associative memory mapping, the address inputted by the user can reside in any block. It is not mapped to any particular cache line. The block goes in that cache line where the cache is empty. If the cache is full, it replaces the block which was least recently used and has the highest priority due to the idea of temporal locality. It has a **higher hit ratio** due to no strict mapping of an address to a cache line but locating the required cache line containing the block **takes more time than direct mapped cache**.

n-way Associative Memory Mapping :

N-way set associative memory mapping is a combination of direct mapped and associative memory mapping which tries to eliminate the drawbacks of both the mapping schemes. Instead of having exactly one cache line that is getting mapped onto by an address, it contains various sets having 'n' number of cache lines. Each address gets mapped to a fixed set where there are 'n' cache lines where the address can reside anywhere. In case the set is full, the block replaces that block in the set which was least used in the past and the new block has the highest priority within the set due to temporal locality. Set associative cache mapping thus **combines the best of direct and associative cache mapping techniques.**

Note: When n=CL in a n-way set associative memory mapping, it behaves as Associative Memory Mapping and when n=1 in a set associative memory mapping, it behaves as Direct Mapped Cache

Here is an explanation of what each of our function does:

1. ***isPowerOf2:***
   Parameters: a
   Return type: bool
   Used by driver function
   Checks whether the parameter *a* is a power of 2 or not. Used to check whether

the provided block size and cache lines are valid or not.

2. ***binaryToDecimal:***
   Parameters: b
   Return type: int/long
   Used by driver function
   Converts the inputted address into decimal format.

3. ***whatPowerOf2:***
   Parameters: a
   Return type: int/long
   Used by all cache mappings
   Returns $\log_2(a)$

4. ***DM_read:***
   Parameters: tagArray, *dataArray*, *address*, *blockOffset*, *cacheLineOffset*
   Return type: void
   Reads the data written at address *address* in our cache.
   If data is present at *address,* prints “Cache hit!” followed by the data found.
   Otherwise prints “Cache Miss!”

5. ***DM_write:***
   Parameters: tagArray, *dataArray*, *address*, *data*, *blockOffset*, *cacheLineOffset*
   Return type: void
   Writes *data* provided by the user at the cache address of *address*. In case of replacement, it prints the tag of the block it replaced in decimal.

6. ***DM_fullCache:***
   Parameters: dataArray, CL, tagArray
   Return type: void
   Prints our entire cache.

7. ***directMappedCache:***
   Parameters: CL, B
   Return type: void
   The driver function for the implementation of Direct Mapped Cache.

8. ***AM_write:***
   Parameters: tagArray, *dataArray*, *blockOffset*, *address*, *data*, *CL, word_each_block*

Return type: void
Writes *data* provided by the user at the cache address of *address* and implements LRU replacement policy. In case of a replacement, it prints the tag of the block it replaced in decimal.

9. ***AM_read:***
Parameters: tagArray, *dataArray*, *blockOffset*, *address*, *CL*
Return type: void
Reads the data written at address *address* in our cache.
If data is present at *address,* prints “Cache hit!” followed by the data found.
Otherwise prints “Cache Miss!” and increments the LRU counter on access of a block.

10. ***AM_printCache:***
Parameters: dataArray,CL,tagArray
Return type: void
Prints our entire cache along with the LRU counter values in the tag part.

11. ***associativeMemoryCache:***
Parameters: CL, B
Return type: void
The driver function for the implementation of Associative Mapped Cache.

12. ***n_way_AM_write:***
Parameters: tagArray, *dataArray*, *address*, *blockOffset, SetOffset, data,block_each_set*
Return type: void
Writes *data* provided by the user at the cache address of *address* and implements LRU replacement policy. In case of a replacement, it prints the tag of the block it replaced in decimal.

13. ***n_way_AM_read:***
Parameters: tagArray, *dataArray*, *address, blockOffset, SetOffset, block_each_set*
Return type: void
Reads the data written at address *address* in our cache.
If data is present at *address,* prints “Cache hit!” followed by the data found.
Otherwise prints “Cache Miss!” and increments the LRU counter on access of a block.

14. ***n_way_AM_print:***
    Parameters: dataArray, CL, tagArray , n
    Return type: void
    Prints our entire cache along with LRU counter values in the tag part.

15. ***n_way_AssociativeMemoryCache:***
    Parameters: CL, B, n
    Return type: void
    The driver function for the implementation of Associative Mapped Cache.