**Cache Simulator** is coded in Java 1.8

**Running a Cache Simulator**

* At line number 25, change the path for the cache configuration file.

data = **new** String(Files.*readAllBytes*(Paths.*get*("D:\\Studies\\Java Eclipse\\CacheSimulator\\src\\example1.cfg" )));

* At line number 52, change the path for the trace file.

trace = **new** String(Files.*readAllBytes*(Paths.*get*("D:\\Studies\\Java Eclipse\\CacheSimulator\\src\\example1.trc" )));

* Compile and Run the program.

**Hit Rates with different cache configuration**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **S.N.** | **Cache Configuration** | | | **No. of address accessed** | **No. of Hits** | **No. of Misses** | **Hit Rate** |
| **Cache Size** | **Block Size** | **Associativity** |
| 1 | 32768 | 64 | 4 | 695521 | 591570 | 103951 | 0.85054 |
| 2 | 16384 | 64 | 4 | 695521 | 591570 | 103951 | 0.85054 |
| 3 | 32768 | 128 | 4 | 695521 | 598958 | 96563 | 0.86116 |
| 4 | 32768 | 64 | 16 | 695521 | 591570 | 103951 | 0.85054 |
| 5 | 32768 | 64 | 32 | 695521 | 591570 | 103951 | 0.85054 |

**Implementation of the cache simulator**

**Data Structure used for the cache block:** Linked list of Linked List of Strings

**Data Structure used to hold cache access counter:** Linked list of Linked List of Integer

**Implementation of Cache Simulator**

1. Initialize linked list of strings (**Cache**) to zero (“0”)

Cache is empty at first

1. Initialize linked list of integers (**Counters**) to 0.
2. Increment all non-zero **Counter** value by 1 every time we access the new trace.
3. Using index bits of the trace, we calculate the set number. Then we will check if any position in the set is empty or not. If there any empty space in set we can put new trace in the empty space and the corresponding count in the **Counter** will be made zero. And, increment the value of miss by 1.
4. If the values in the calculated set are not empty, we can compare the trace in set with new trace. If it matches, there will be hit thus increase hit by 1 and set corresponding count in **Counter** to zero.
5. If there is a miss, we increase the value of miss by 1. Then we count the maximum value of **Counter** in that set and put that new trace in the position with the maximum count. And initialize the corresponding count by zero.
6. Finally we calculate, hit rate = hit / No. of accesses.

**LRU Implementation**

* **Counter** plays the main role for LRU implementation
* **Counter** are set to zero at the very beginning.
* **Counter** are at the specified set position are made zero (0) again if there is a hit or a value is placed in an empty slot.
* When there is a miss, slot in a set with maximum count value in the counter will be replaced with the new trace.
* After replacement, **counter** value at that position will be set to zero.