Assignment - 2
Submission Date: 10th June 2023
CS1023: Software Development Fundamentals

Simulate a set-associative cache memory and show the following results after every run.
total misses, total hits, set-wise total misses, and set-wise total hits.

Input to the program through command line parameters are Cache size (P), Associativity (W), Block size (64B always), and the trace file path. ***You must use Java for this assignment.***

A. Description:
Assume a set-associative cache memory whenever I mention cache in this document. Read the complete document and only start programming after understanding the complete document. You can ask me any concepts related to cache memory. Consider **B** as **bytes** in this document.

A.1 Basic Concepts:
A set associative cache can be considered as a 2D array where the rows can be considered as sets and columns as ways. A cache block always maps to a fixed set based on the set-index bits calculated from its 32-bit address. The address of a block can be divided into ***<tag, set-index, offset>***. The offset bits are required for uniquely identifying a word within the block. The total bits in offset are ***log2(block size)***. For a 64B block size the offset is 6 bits. The set-index bits can be calculated as ***log2(total sets)***.

Given the block size, associativity and cache size, you can calculate the number of sets available in the cache. In a set-associative cache, each entry (in the 2D array) represents a block. If the block size is **64**B and associativity is **W**, the size of single set is **W*64**B. Now, if the size of the cache is **P**KB, then the total sets in the cache is: **P*1024/W*64**. Multiplying P with 1024 is for converting KB into B.

When a processor requests a block with its 32-bit address, the block is searched in the cache. The entire cache is not searched, only a specific set, based on the set-index bits of the requested address is searched. If the block is found in the cache then it is called a hit otherwise miss. In case of miss, the block is fetched from main memory and stored in the cache. To store a newly incoming block in the cache, an existing block may need to be replaced from the cache. While searching for a block in the cache, the tag part of its block address is searched for a match with the tags of the block present in the cache. A tag match means, cache hit.

Trace is the sequence of the block addresses requested by a processor. Assume that the trace has three addresses **a1, a2,** and **a1**. It means the processor has first requested **a1**, then **a2**, and then **a1** again. ***Each address is represented as a 32-bit address where the first six bits (from the right) are always zero.***

A.2 How to simulate the cache:
Design the cache with appropriate java collections. To design the cache, the program first calculates the number of sets required and the set-index bits. The program must have some configuration parameters, for Cache Size, Associativity, and block size. These parameters should be allowed to pass through command line parameters. Which means your program should be able to simulate a cache memory of any size. ***Keep the block size fixed to 64B.***

Once the cache is designed, read the trace file, which also needs to be supplied as command line parameters. After reading the trace file, save the addresses in a list or some other java collection. Please note that the trace must have repeated block addresses. ***You need to first create 2-3 trace files of multiple 32-bits addresses by yourself.***

Read one address at a time from the trace, find out which set it maps. Search that particular set in the cache for the block. If tag matches, increment the hit counter. Otherwise, increment the miss counter and place the newly requested block in the cache. If the set has no free location available, you must replace an existing block from the set. ***Initially use LRU as the replacement policy.***