**PES University Electronic City Campus**

Hosur Rd, Konappana Agrahara, Electronic City, Bengaluru, Karnataka 560100



A Project Report on

*“Implementation of CUDA program for matrix multiplication*

*&  
Configuring data caches in RIPES for given specifications”*

Submitted by:

* **AMEER SOHAIL (PES2UG20EC138)**
* **DARSHAN (PES2UG20EC147)**
* **BHUVANESH GOWDA R(PES2UG20EC140)**

Under the guidance of:

**Prof.shashidhar Tantry  
Associate Professor,**

**Department of Electronics and Communication Engineering**

**PES UNIVERSITY**

Electronic City, Bangalore-560100

2022 - 2023

**ABSTRACT**

1. Given two large matrices A and B, represented as two-dimensional arrays of size N x N, where N is a positive integer. You need to implement a CUDA program to perform matrix multiplication of A and B, and store the result in a new matrix C, also of size N x N.
2. Configuring the data cache in the given specification on RIPES. Running the matrix multiplier code to simulate each data cache configuration and report cache size, hit rate and write-backs.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO** | **TITLE** | **PAGE NO.** |
| **1.** | **OBJECTIVE** | 03 |
| **2.** | **METHODOLOGY** | 04 |
| **3.** | **IMPLEMENTATION** | 05-06 |
| **4.** | **RESULTS** | 07-13 |
| **5.** | **CONCLUSION** | 14 |

**CHAPTER 1**

**OBJECTIVE**

1. Implementing a CUDA kernel that performs matrix multiplication on the GPU. The kernel should take the input matrices A and B, and compute the resulting matrix C. Each thread in the CUDA grid should compute one element of the output matrix C, by performing the dot product of the corresponding row in A and column in B. Also implementing appropriate memory management, including allocation of memory on the GPU, transferring the input matrices A and B to the GPU memory, and transferring the resulting matrix C back to the CPU memory for further processing or storage.
2. Configuring the data cache in the given specification on RIPES. Running the matrix multiplier code to simulate each data cache configuration and report cache size, hit rate and write-backs.
3. Configure a 64-entry 8-word direct mapped cache.
4. Configure a 16-entry 4-word 4-way set-associative cache.
5. Configure a 16-entry 2-word 4-way set-associative cache with write-through.
6. Configure a 64-entry 8-word fully associative cache with Least Recently Used replacement policy and report the numbers. Change the replacement policy to Random and report the numbers for the same cache.
7. Configure a 32-entry 2-word direct-mapped cache and plot a graph using the plot configuration with numerator as Hits and denominator as Access count. Explain why the number of hits increases and decreases back down before increasing again.
8. Configure a 16-entry 4-word 2-way set-associative cache with write-back and write allocate and report the numbers. For the same configuration of cache, use write-through and no write allocate and report the numbers. Explain the differences between the two cache configurations

**CHAPTER 2**

**METHODOLOGY**

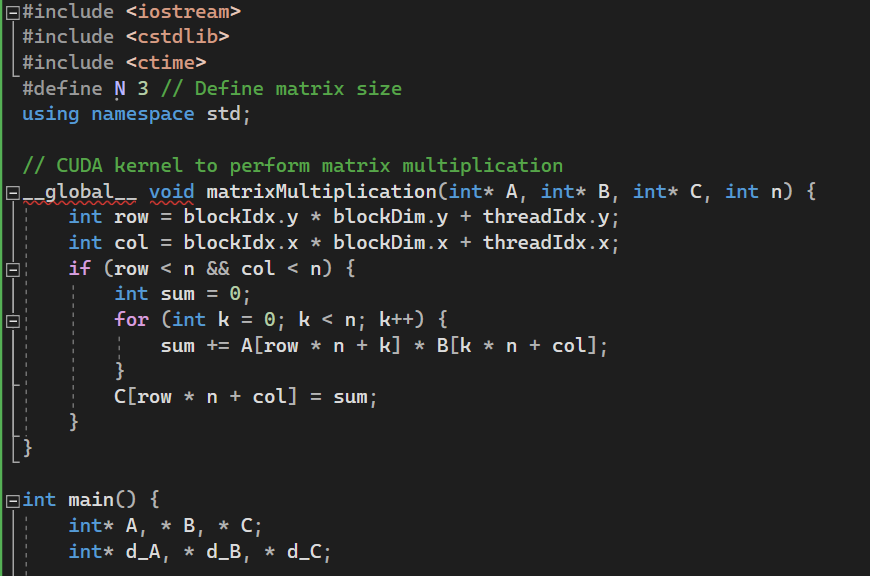
Implementation of CUDA kernel or program on visual studio to study the memory allocation on the GPU, exploring the parallel processing capablities and effectively utilizing the CUDA threads and blocks

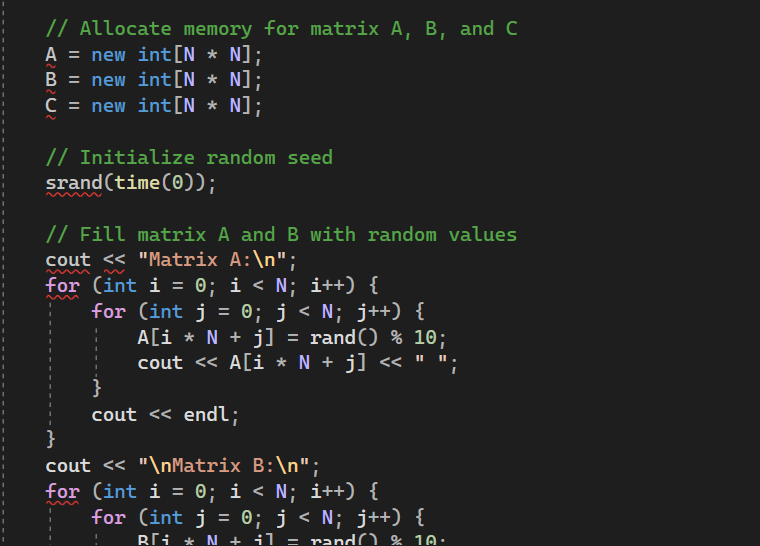
Using RIPES for configuring caches of different specifications and approaches such as writeback or writethrough, going through the plot of the configuration to observe the hits and misses

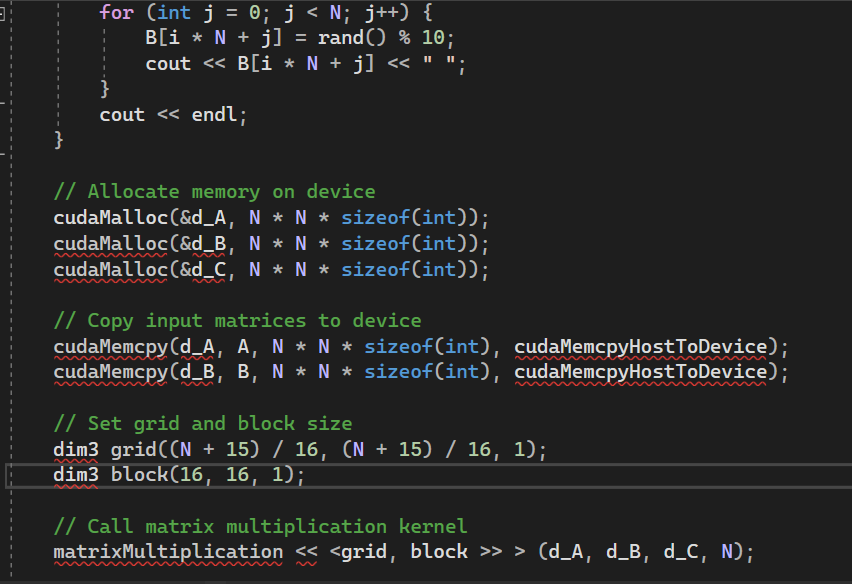
**CHAPTER 4**

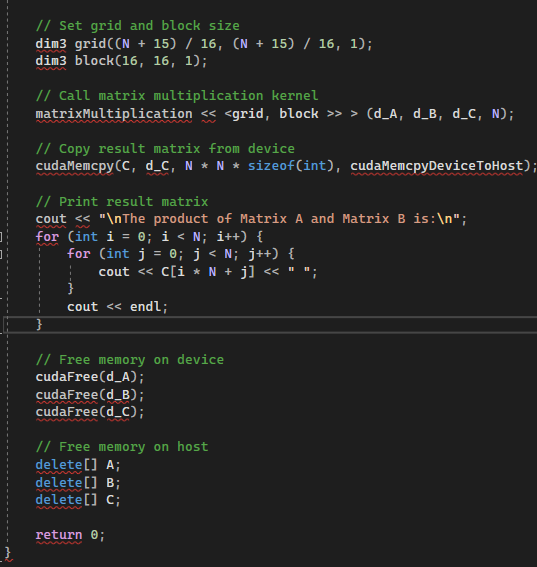
**IMPLEMENTATION**

**CUDA program:**



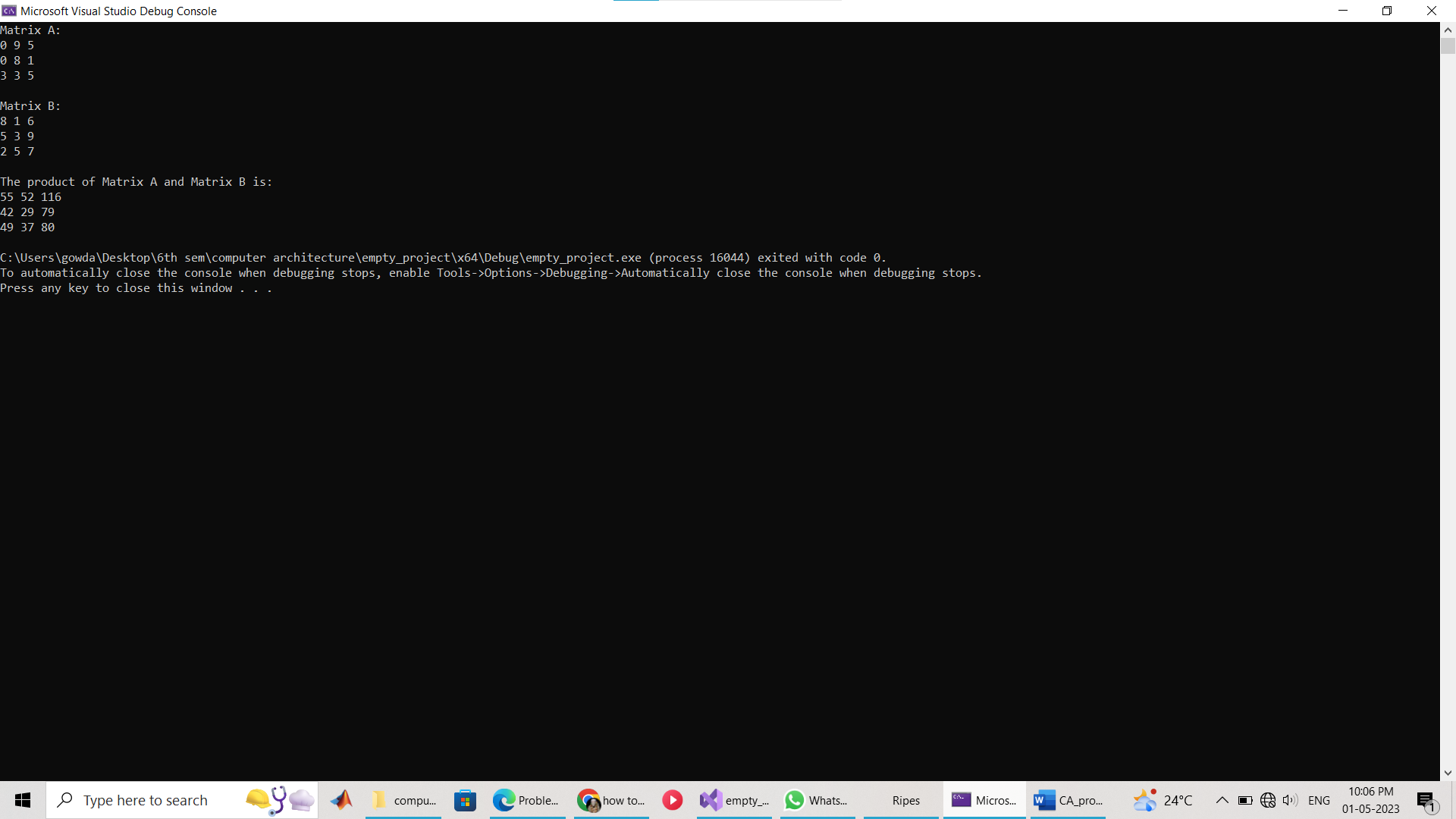






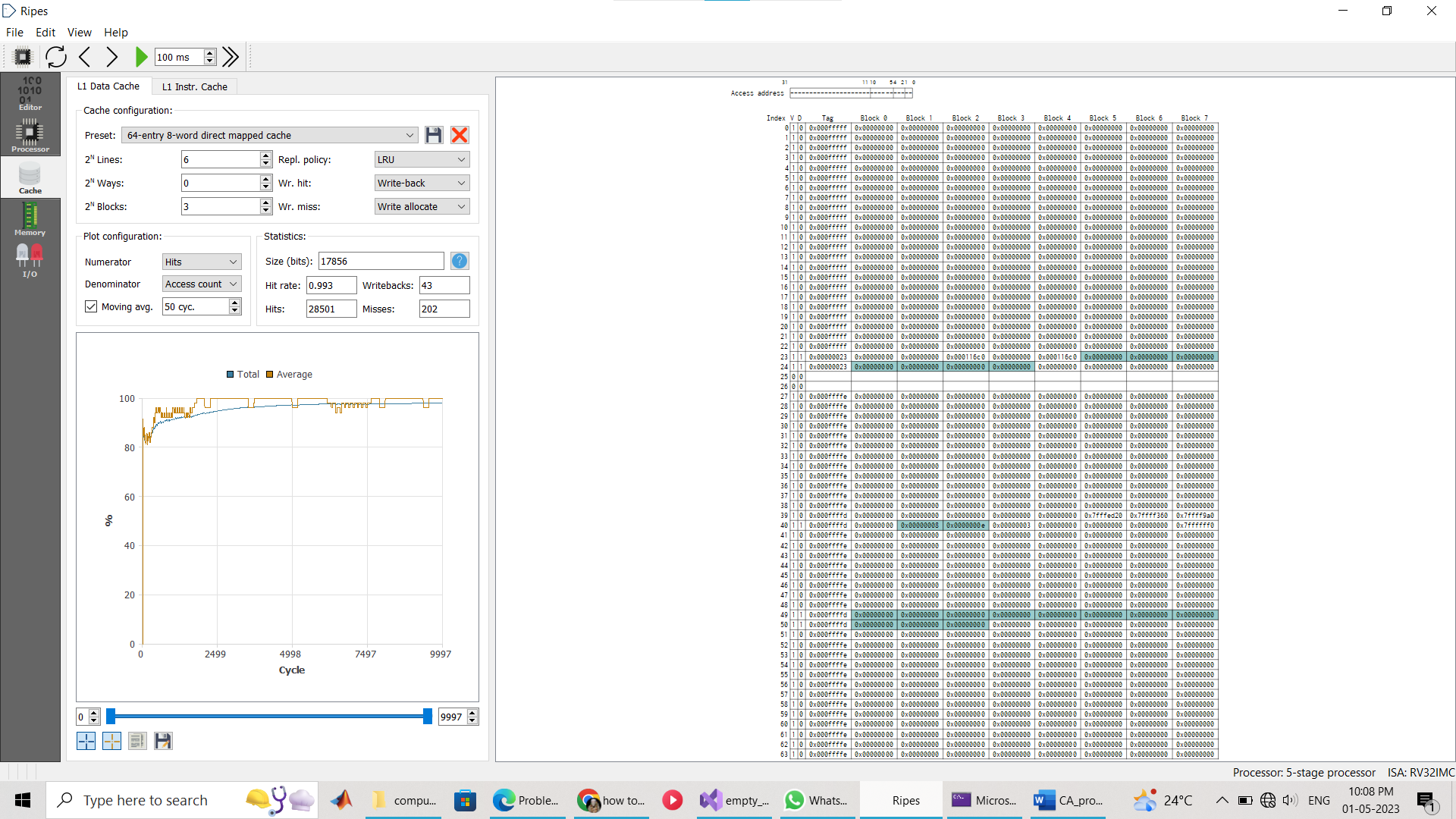
**RESULTS**

**PART 1- CUDA:**

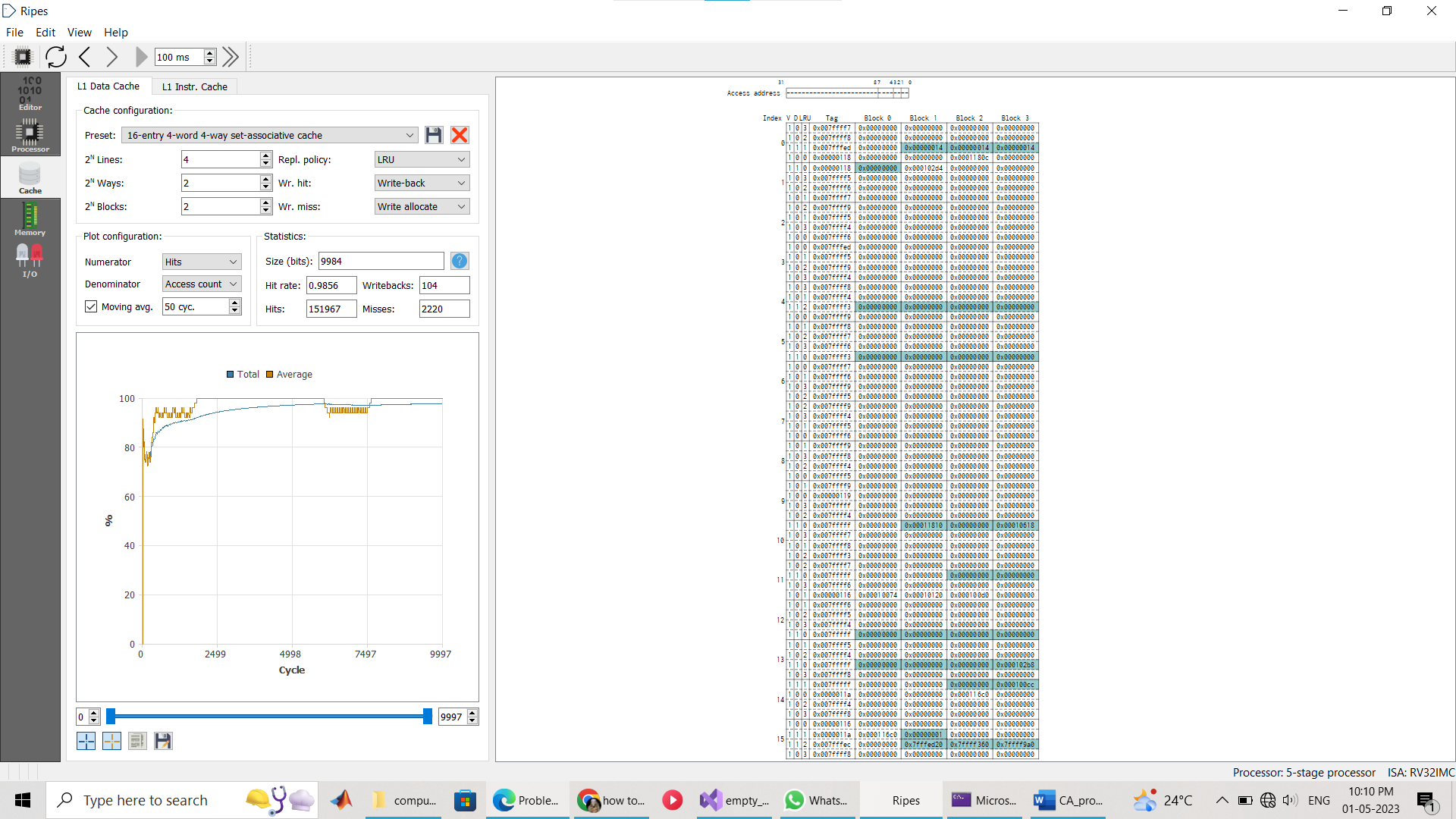
****

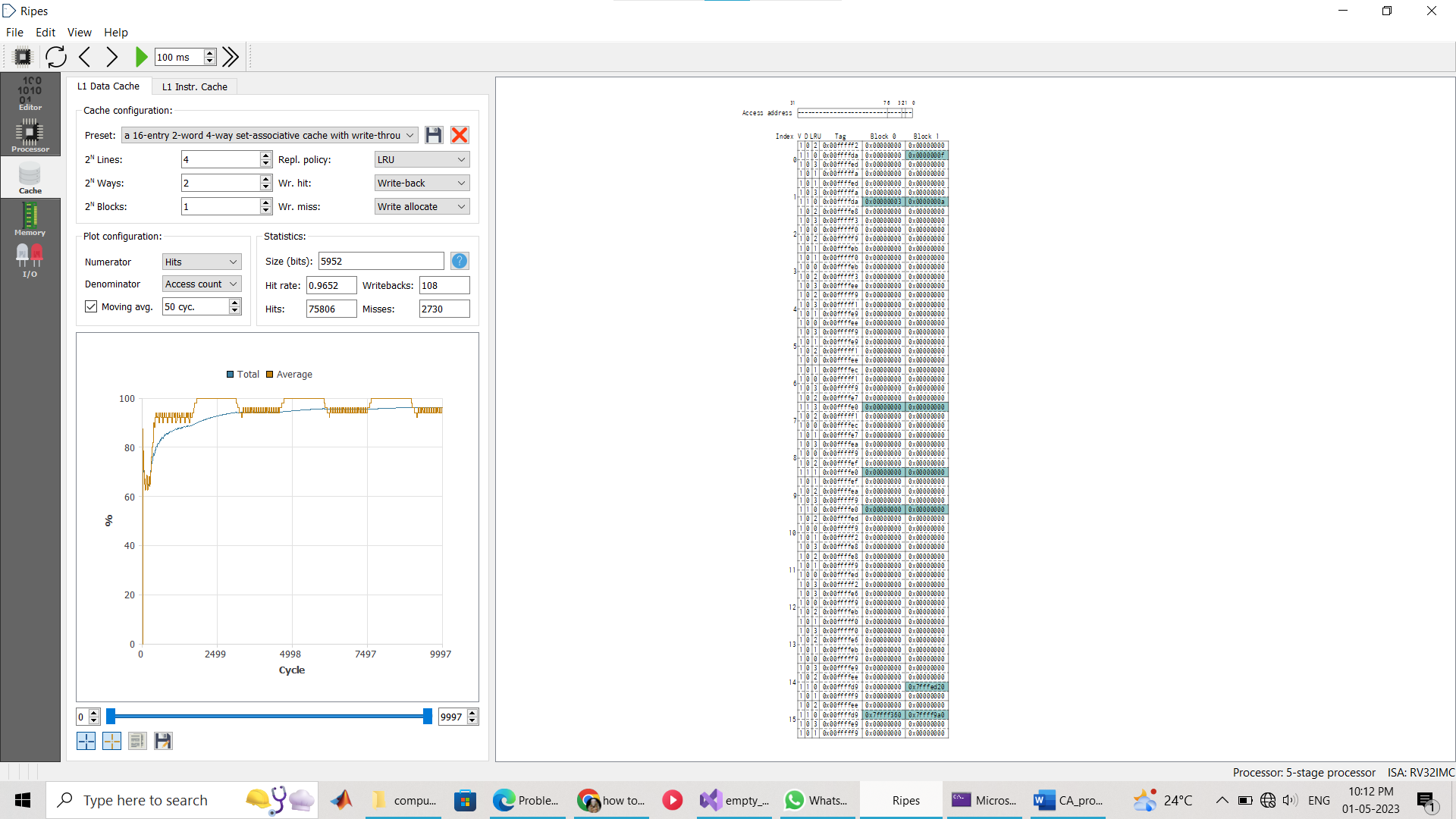
**PART 2 - RIPES:**

**Q1)**

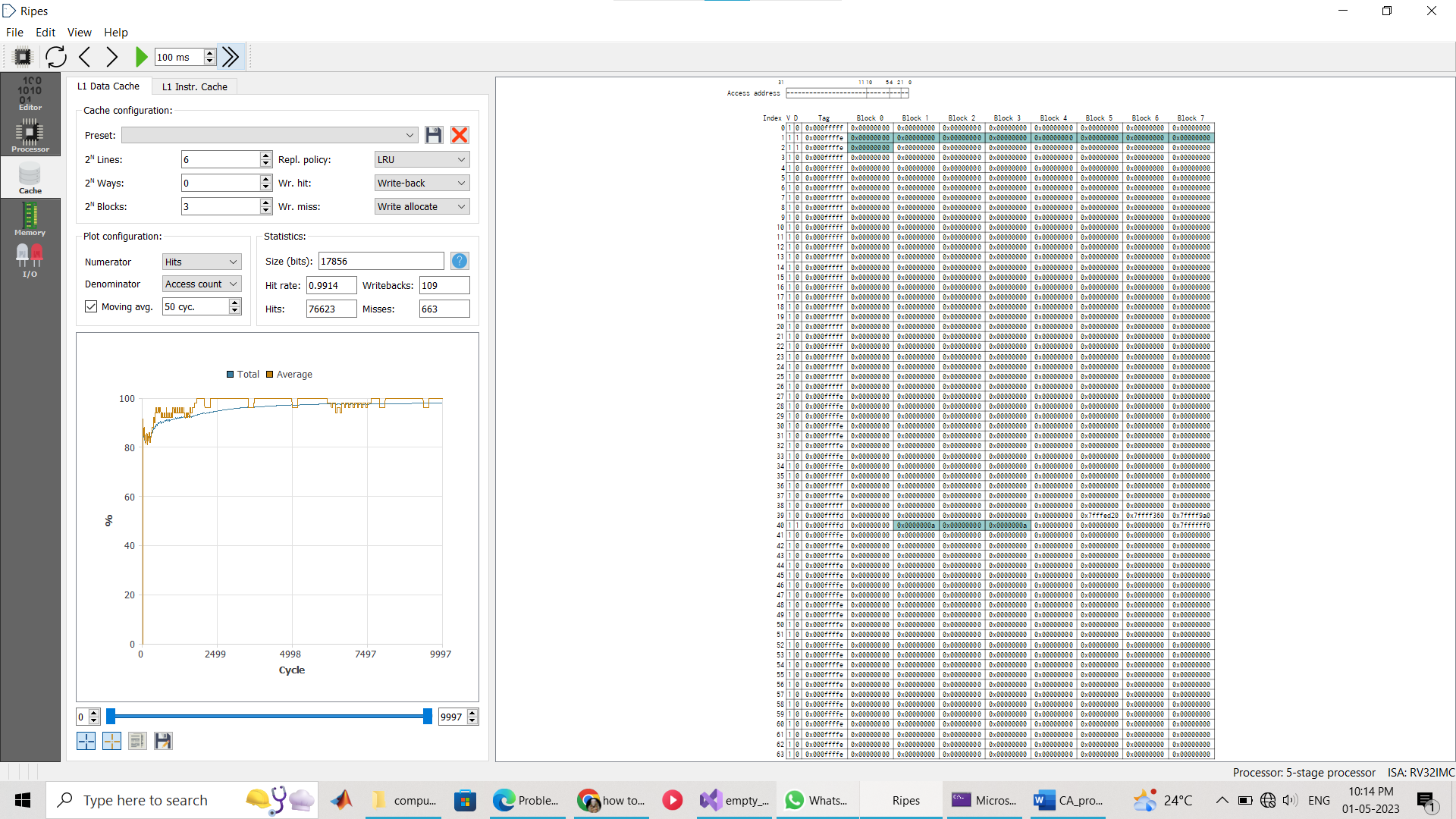
****

**Q2)**

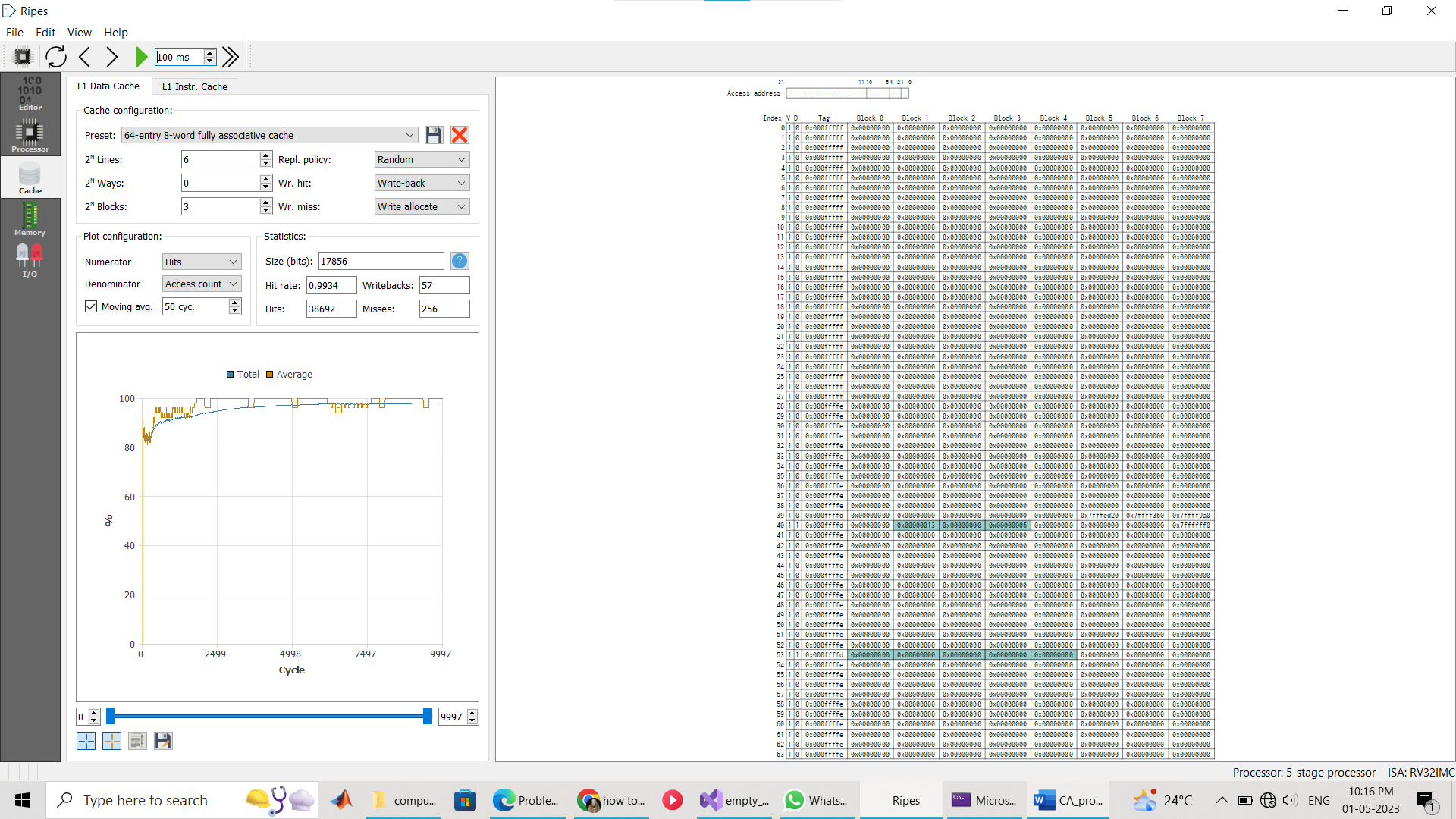
****

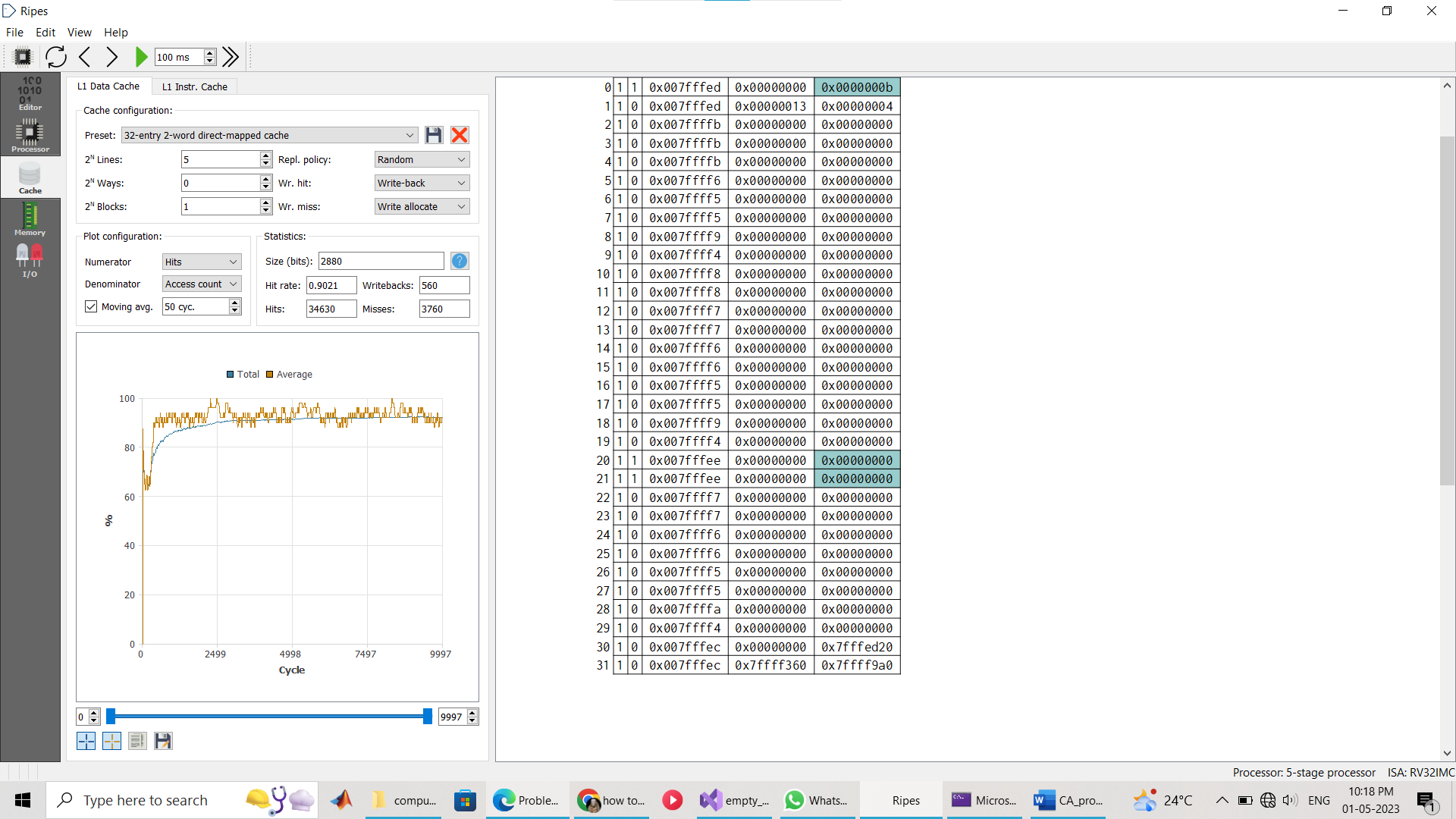
**Q3)**

**4a)**

****

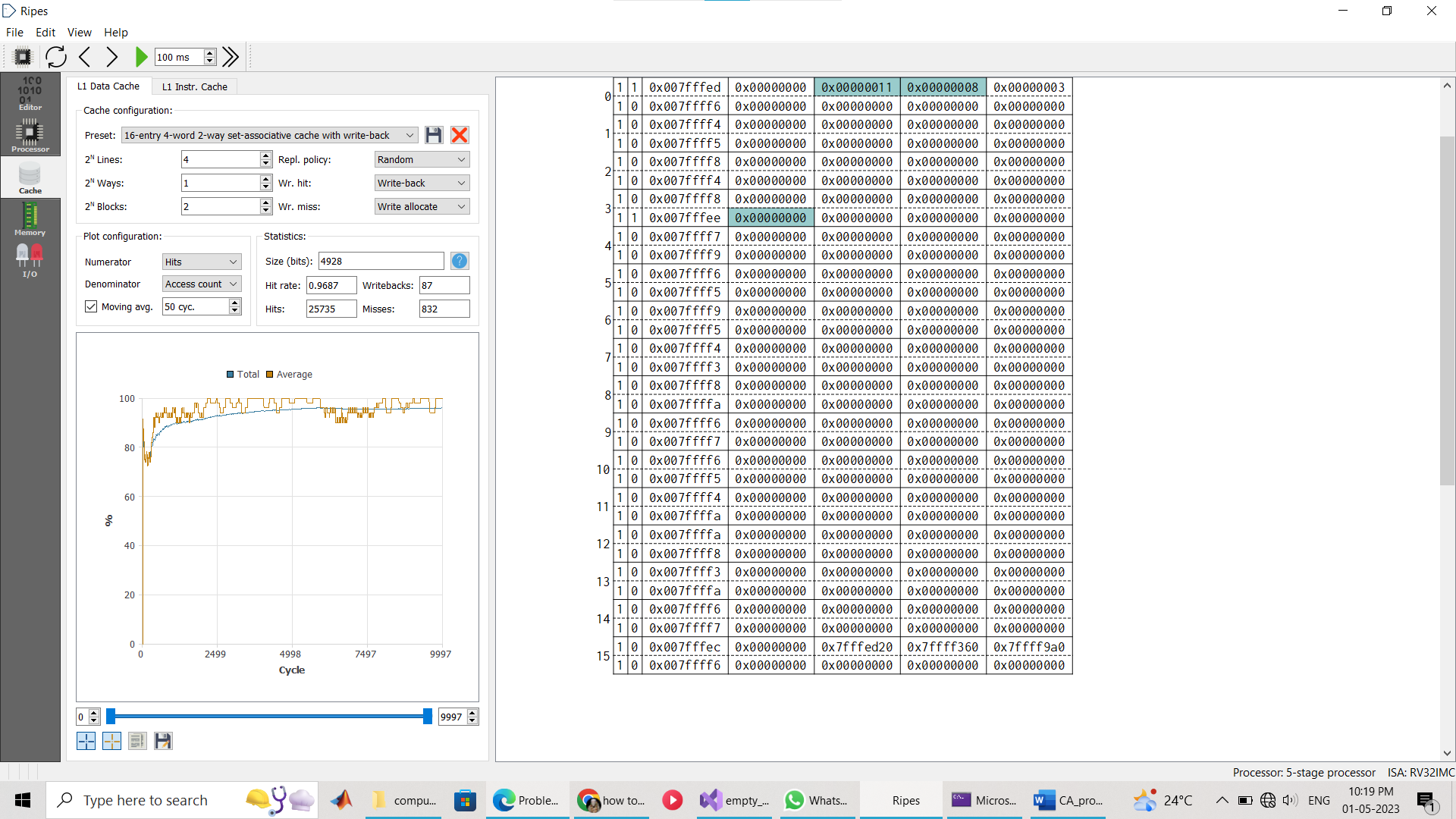
**4b)**

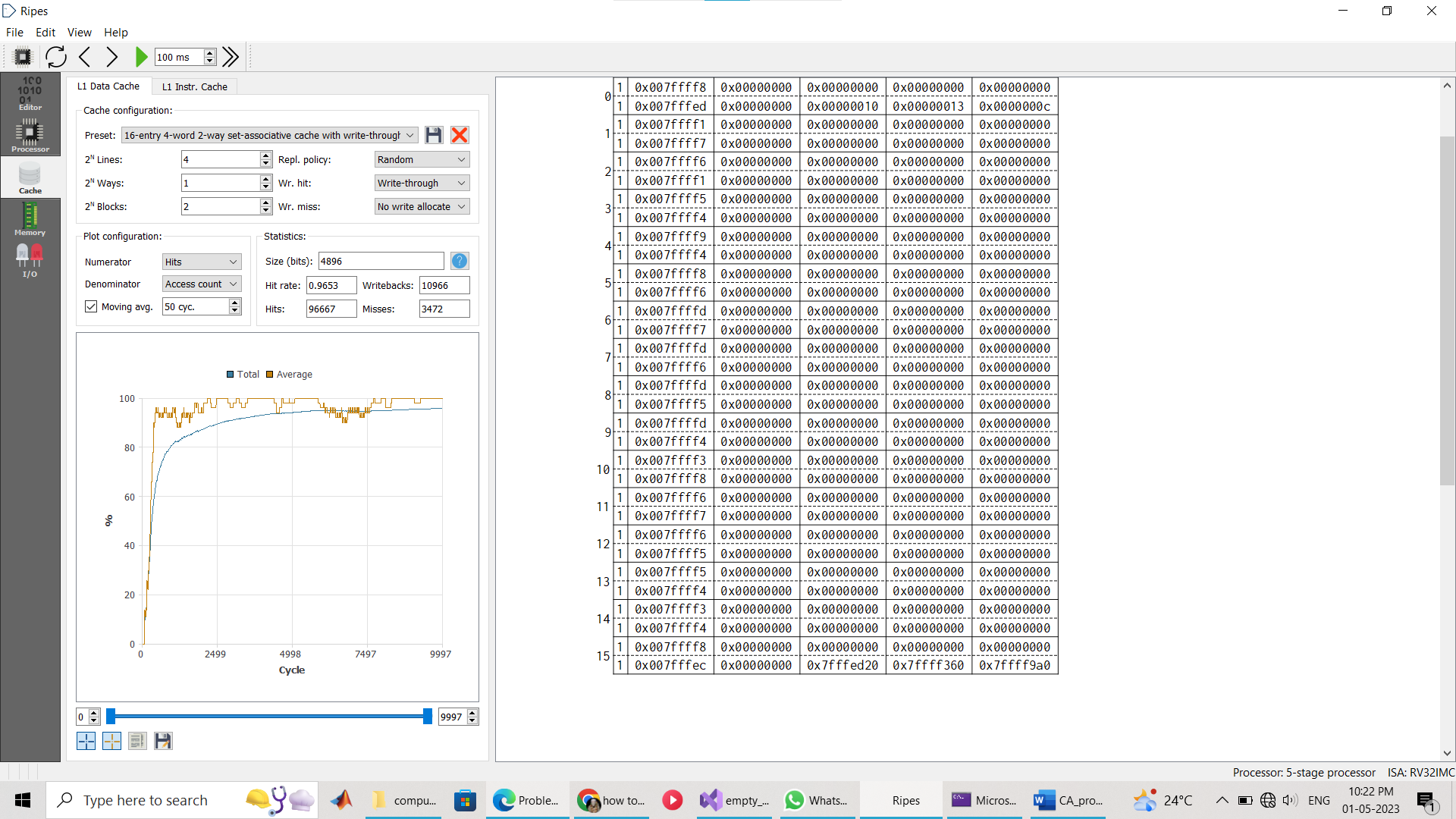
****

**5)  
**

In a Cache with write-through and no-write-allocate policies, write operations are immediately written both to the cache and main memory. No-write-allocate means that when a write operation misses in the cache, the write is directly performed in main memory without bringing the block into the cache.

**6a)**

****

**6b) **

In a 32-entry 2-word direct-mapped cache, the number of hits increases and decreases back down before increasing again because of cache conflicts. When a program accesses memory blocks that map to the same cache line, the cache can only hold one of the blocks at a time, so it will evict one and fetch the other, resulting in a cache miss. As the program continues to access these memory blocks, the cache will alternate between holding one block and the other, resulting in alternating hits and misses on the cache line. Therefore, the number of hits may increase up to a maximum of 16 per cache line (since there are only 16 cache lines available for each set of two memory blocks that map to the same cache line), but will eventually decrease due to evictions and cache misses, before increasing again when the program accesses the memory blocks that were evicted earlier.

**CONCLUSION**

We notice that the CUDA kernel can be implemented to perform various operations on the GPU making use of memory management, taking advantage of the parallel processing capabilities of the GPU. Minimizing data transfers between CPU and GPU, and effectively utilizing the CUDA threads and blocks to achieve maximum throughput and performance.

Different data caches can be configured using RIPES, of different sizes for the cache and different approaches such as direct mapping, set associative and fully associative. Difference in the hits, hit rate, misses, miss rates and other parameters as well is studied and compared.