**­­­­Computer Architecture Lab Report Week 12**

**Full name: Nguyễn Hồng Phúc**

**Student ID: 20225659**

# A. Row-major

*What was the final cache hit rate?* \_\_\_\_\_\_\_\_\_\_\_\_\_. With each miss, a block of 4  
words are written into the cache. In a row-major traversal, matrix elements are  
accessed in the same order they are stored in memory. Thus each cache miss is  
followed by 3 hits as the next 3 elements are found in the same cache block. This is  
followed by another miss when Direct Mapping maps to the next cache block, and the  
patterns repeats itself. So 3 of every 4 memory accesses will be resolved in cache.

A screenshot of a computer

Description automatically generated

9. Given that explanation, what do you predict the hit rate will be if the block size isincreased from 4 words to 8 words? \_\_\_\_\_\_\_\_\_\_\_\_\_\_. Decreased from 4 words to 2words? \_\_\_\_\_\_\_\_\_\_\_

**Current Scenario with 4 Words:**

* For every block (4 words), 1 miss and 3 hits.
* Hit rate: 3/4​ or 75%.

**New Scenario with 8 Words:**

* For every block (8 words), 1 miss and 7 hits.
* Let's break it down: The first access to a new block (8 words) causes a miss. The next 7 accesses are hits because they are within the same block.
* Hit rate calculation:
  + Total accesses per block = 8.
  + Misses per block = 1.
  + Hits per block = 7.
  + Hit rate = 7/8 or 87.5%.

**Decreasing Block Size from 4 Words to 2 Words**

**New Scenario with 2 Words:**

* For every block (2 words), 1 miss and 1 hit.
* Let's break it down: The first access to a new block (2 words) causes a miss. The next access is a hit because it is within the same block.
* Hit rate calculation:
  + Total accesses per block = 2.
  + Misses per block = 1.
  + Hits per block = 1.
  + Hit rate = 1/2​ or 50%.

**Summary**

* **Hit rate with block size of 8 words:** 87.5%.

A screenshot of a computer

Description automatically generated

* **Hit rate with block size of 2 words:** 50%.

A screenshot of a computer

Description automatically generated

# B. Column-major

*What was the cache performance for this program?* \_\_\_\_\_\_\_\_\_\_\_\_. The problem is  
the memory locations are now accessed not sequentially as before, but each access is  
16 words beyond the previous one (circularly). With the settings we've used, no two  
consecutive memory accesses occur in the same block so every access is a miss.

### Cache Behavior in Column-Major Order

Given the column-major traversal:

* For (col = 0; col < 16; col++)
  + For (row = 0; row < 16; row++)
    - data[row][col] = value++;

### Memory Access Pattern

* The matrix is stored in a linear array in memory, with each row of 16 integers stored consecutively.
* Accesses to data[row][col] in column-major order result in jumps of 16 words (integers) between consecutive accesses.
* For example, accessing data[0][0] is followed by accessing data[1][0], which is 16 integers away in memory, and so on.

### Cache Performance Analysis

Assume each cache block holds 4 words (integers), and we are using a direct-mapped cache:

1. **Initial Setup**:
   * For each column access, starting with data[0][0], the cache misses and loads the block containing data[0][0], data[0][1], data[0][2], and data[0][3].
2. **Subsequent Accesses**:
   * Access to data[1][0] results in another cache miss, loading the block containing data[1][0], data[1][1], data[1][2], and data[1][3].
   * This pattern continues, with each access being 16 words apart, leading to each access mapping to a different cache block.
   * Given the pattern, no two consecutive memory accesses fall within the same cache block.

### Conclusion on Cache Performance

* **Cache Misses**: Since each access in column-major order is 16 words apart and each cache block contains only 4 words, every access results in a cache miss. No two consecutive accesses map to the same cache block.
* **Hit Rate**: The hit rate is effectively 0% because every access is a cache miss.

### Summary

### The cache performance for this program, given the settings and traversal pattern, is:

* **Cache Miss Rate**: 100%
* **Cache Hit Rate**: 0%

A screenshot of a computer

Description automatically generated

13. Change the block size to 16. Note this will reset the tool.  
14. Create a second instance of the Cache Simulator by once again selecting **Data Cache  
Simulator** from the **Tools** menu. Adjust the two frames so you can view both at the  
same time. Connect the new tool instance to MIPS, change its block size to 16 and  
change its number of blocks to 16.  
15. Re-run the program. *What is the cache performance of the original tool instance?*\_\_\_\_\_\_\_\_\_\_\_\_. Block size 16 didn't help because there was still only one access to  
each block, the initial miss, before that block was replaced with a new one. *What is  
the cache performance of the second tool instance?* \_\_\_\_\_\_\_\_\_\_\_\_. At this point,  
the entire matrix will fit into cache and so once a block is read in it is never replaced.  
Only the first access to a block results in a miss.

Để phân tích hiệu suất bộ nhớ đệm, chúng ta sẽ thực hiện các bước sau:

1. **Tạo một instance của Cache Simulator**:
   * Chọn **Data Cache Simulator** từ menu Tools.
   * Kết nối instance mới này với MIPS.
   * Thay đổi kích thước khối (block size) thành 16.
   * Thay đổi số lượng khối (number of blocks) thành 16.
2. **Chạy chương trình và quan sát hiệu suất bộ nhớ đệm của instance gốc**:
   * Khi kích thước khối là 4 từ (mặc định), duyệt ma trận theo thứ tự cột sẽ dẫn đến mỗi lần truy cập là một lỗi bộ nhớ đệm (cache miss).
   * Điều này xảy ra vì mỗi truy cập cách nhau 16 từ và mỗi khối chỉ chứa 4 từ, dẫn đến không có hai truy cập nào nằm trong cùng một khối bộ nhớ đệm.
   * **Hiệu suất bộ nhớ đệm của instance gốc**: 0% hit rate (tỷ lệ trúng là 0%).
3. **Chạy chương trình với kích thước khối 16 từ và số lượng khối 16**:
   * Với kích thước khối là 16 từ, mỗi khối bộ nhớ đệm có thể chứa toàn bộ một cột của ma trận 16x16.
   * Tuy nhiên, vì mỗi lần truy cập chỉ truy cập một từ trong khối, sau lần truy cập đầu tiên, khối đó sẽ bị thay thế bởi một khối mới cho lần truy cập tiếp theo.
   * **Hiệu suất bộ nhớ đệm của instance thứ hai**: Vẫn là 0% hit rate (tỷ lệ trúng là 0%).
4. **Tăng số lượng khối lên 256**:
   * Nếu tăng số lượng khối lên 256, toàn bộ ma trận sẽ nằm gọn trong bộ nhớ đệm.
   * Mỗi lần truy cập đầu tiên vào một khối sẽ dẫn đến lỗi bộ nhớ đệm, nhưng sau đó, khối sẽ không bị thay thế vì bộ nhớ đệm đủ lớn để chứa toàn bộ ma trận.
   * **Hiệu suất bộ nhớ đệm sau khi tăng số lượng khối**:
     + Chỉ lần truy cập đầu tiên vào mỗi khối dẫn đến lỗi (16 khối đầu tiên).
     + Sau đó, tất cả các truy cập đều là trúng.
     + Với 256 từ trong bộ nhớ đệm và 16 lần lỗi ban đầu, tỷ lệ trúng sẽ là:
       - Số lần truy cập: 256 (16x16).
       - Số lần trúng: 240.
       - **Tỷ lệ trúng**: ≈93.75%.

### Tóm tắt

* **Hiệu suất bộ nhớ đệm của instance gốc (kích thước khối 4 từ)**: 0% hit rate.
* **Hiệu suất bộ nhớ đệm của instance thứ hai (kích thước khối 16 từ)**: 0% hit rate.
* **Hiệu suất bộ nhớ đệm sau khi tăng số lượng khối lên 256**: Khoảng 93.75% hit rate.

A screenshot of a computer

Description automatically generated

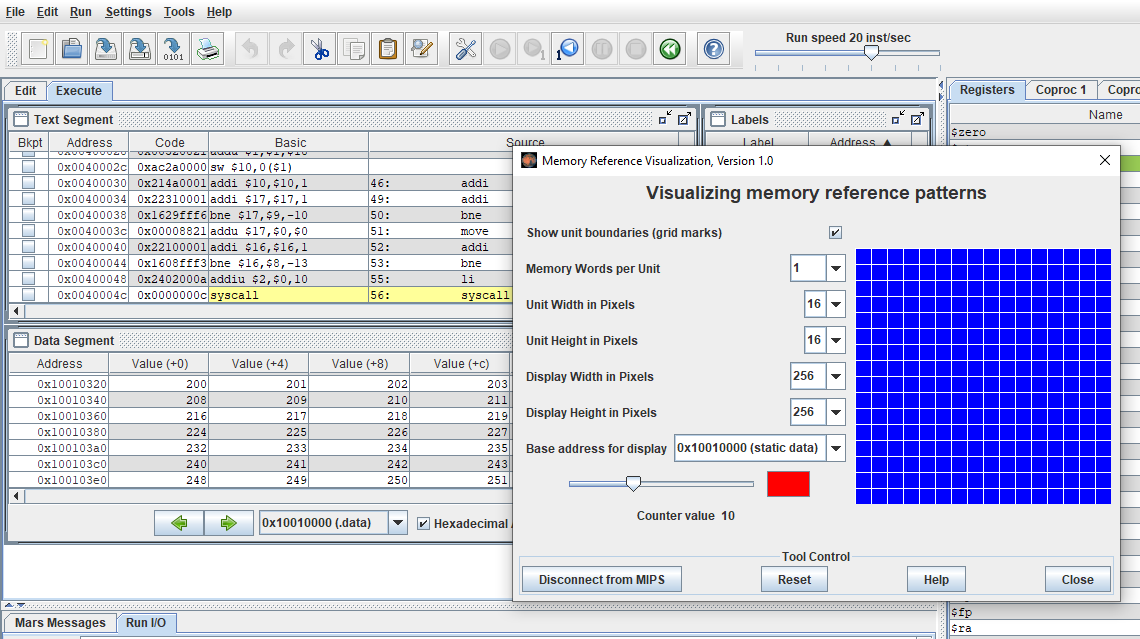
# C. The Memory Reference Visualization tool

Với chương trình row-major khi đang chạy:

A screenshot of a computer

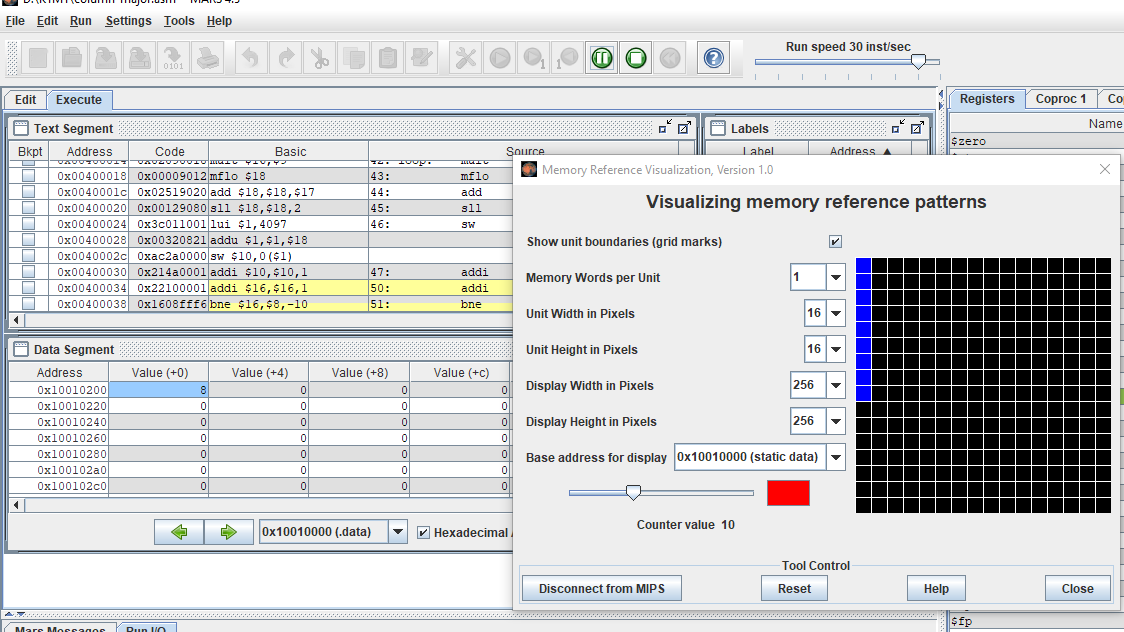
Description automatically generated

Khi đã chạy xong:



Với chương trình column-major:

Trước khi chạy xong



Sau khi chạy xong:

A screenshot of a computer

Description automatically generated

Với chương trình Fibonacci:

Trước khi chạy xong:

A screenshot of a computer

Description automatically generated

Sau khi chạy xong:

A screenshot of a computer

Description automatically generated