

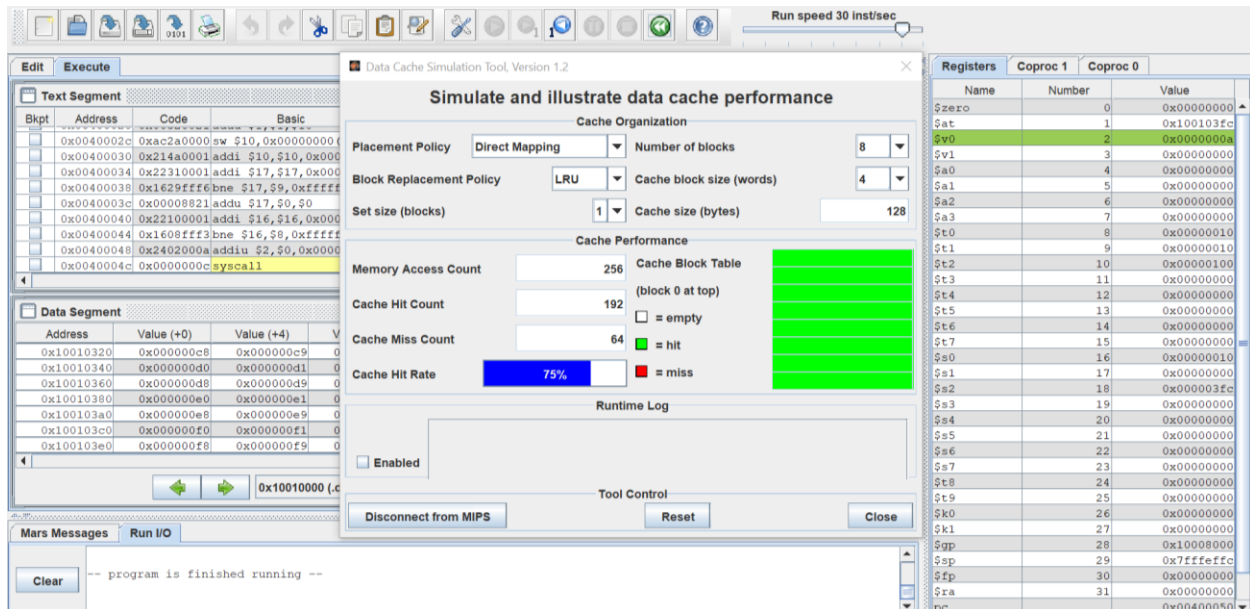
Báo cáo Thực hành KTMT buổi 13

Họ và tên: Nguyễn Đức Phú

MSSV: 20215116

Part 1: Running the Data Cache Simulator tool

8. What was the final cache hit rate? => 75%



9. Thay đổi block size và dự đoán kết quả

- Tăng từ 4 lên 8: Với block size là 8 → mỗi lần cache miss thì 7 phần tử sau đó sẽ được tìm thấy tại cache
⇒ Tỷ lệ dự đoán: $\frac{7}{8}$ (= 87.5%)
- Giảm từ 4 xuống 2: Với block size là 2 → mỗi lần cache miss thì 1 phần tử sau đó sẽ được tìm thấy tại cache
⇒ Tỷ lệ dự đoán: $\frac{1}{2}$ (= 50%)

10. Chạy và kiểm chứng

Cache Organization

Placement Policy

Direct Mapping

Number of blocks

8

Block Replacement Policy

LRU

Cache block size (words)

8

Set size (blocks)

1

Cache size (bytes)

256

Cache Performance

Memory Access Count

256

Cache Block Table

(block 0 at top)

☐ = empty

☒ = hit

☐ = miss

Cache Hit Count

224

Cache Miss Count

32

Cache Hit Rate

88%

Con số 88% là do làm tròn từ 87.5%

Cache Organization

Placement Policy

Direct Mapping

Number of blocks

8

Block Replacement Policy

LRU

Cache block size (words)

2

Set size (blocks)

1

Cache size (bytes)

64

Cache Performance

Memory Access Count

256

Cache Block Table

(block 0 at top)

☐ = empty

☒ = hit

☐ = miss

Cache Hit Count

128

Cache Miss Count

128

Cache Hit Rate

50%

12. Chạy với file `column-major.asm`

What was the cache performance for this program? => 0%

The screenshot shows the 'Data Cache Simulation Tool, Version 1.2' interface. The 'Execute' window is active, displaying a 'Text Segment' with memory addresses, codes, and basic information. The 'Cache Organization' section shows 'Direct Mapping' and 'LRU' replacement policy. The 'Cache Performance' section shows a 'Cache Block Table' with a red bar chart indicating hits and misses. The 'Runtime Log' section shows a list of events. The 'Registers' section shows the state of Coprocessor 1 and Coprocessor 0.

14. Chạy thử song song

The image displays two instances of the 'Data Cache Simulation Tool, Version 1.2' window, illustrating the effect of different cache replacement policies on hit rate.

Left Window (Successful Simulation):

- Title:** Simulate and illustrate data cache performance
- Cache Organization:**
 - Placement Policy: Direct Mapping
 - Number of blocks: 16
 - Block Replacement Policy: LRU
 - Cache block size (words): 16
 - Set size (blocks): 1
 - Cache size (bytes): 1024
- Cache Performance:**
 - Memory Access Count: 256
 - Cache Hit Count: 240
 - Cache Miss Count: 16
 - Cache Hit Rate: 94%
 - Cache Block Table: A green bar chart representing the hit/miss status for each access.
- Runtime Log:** Enabled (checkbox).
- Tool Control:** Disconnect from MIPS, Reset, Close.

Right Window (Failed Simulation):

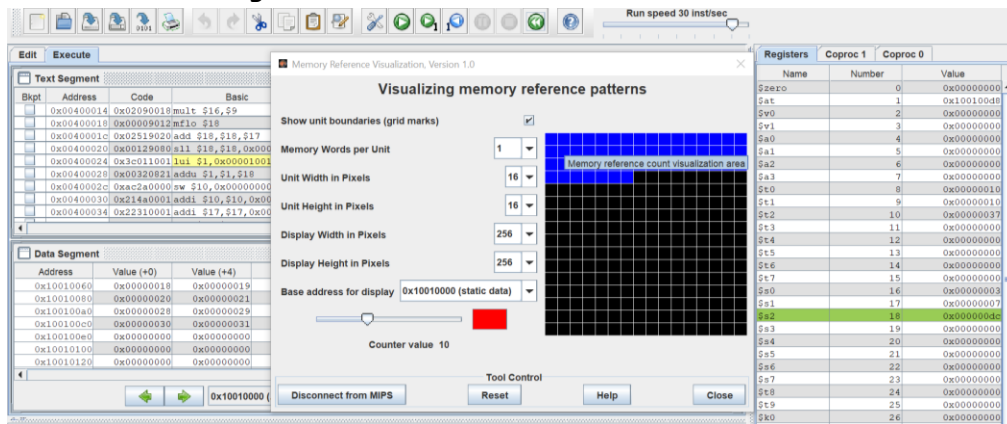
- Title:** Simulate and illustrate data cache performance
- Cache Organization:**
 - Placement Policy: Direct Mapping
 - Number of blocks: 8
 - Block Replacement Policy: LRU
 - Cache block size (words): 16
 - Set size (blocks): 1
 - Cache size (bytes): 512
- Cache Performance:**
 - Memory Access Count: 256
 - Cache Hit Count: 0
 - Cache Miss Count: 256
 - Cache Hit Rate: 0%
 - Cache Block Table: A red bar chart representing the hit/miss status for each access.
- Runtime Log:** Enabled (checkbox).
- Tool Control:** Disconnect from MIPS, Reset, Close.

15. Nhận xét

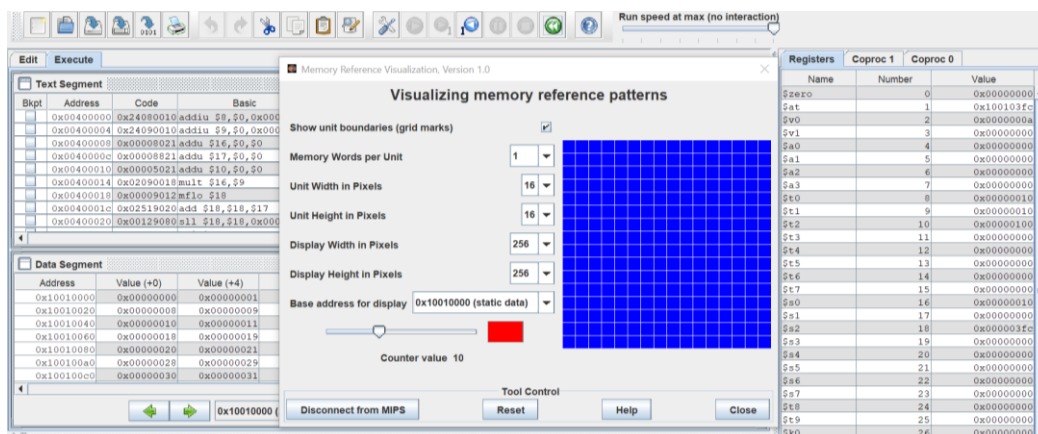
- Nếu chỉ tăng block size lên 16 sẽ không giúp cải thiện hiệu suất do vẫn chưa phù hợp với kích thước ma trận
 - ⇒ Các block ban đầu bị thay thế trước khi chúng được truy cập ở phần sau của chương trình
 - ⇒ Cache hit rate: 0%
- Nếu tăng số block lên 16 kích thước ma trận sẽ vừa với bộ đệm
 - ⇒ Các block không bị thay thế trong suốt chương trình
 - ⇒ Chỉ có lần truy cập đầu tiên với mỗi block bị miss
 - ⇒ Cache hit rate: 15/16 ($\approx 94\%$)

Part 2: The Memory Reference Visualization tool

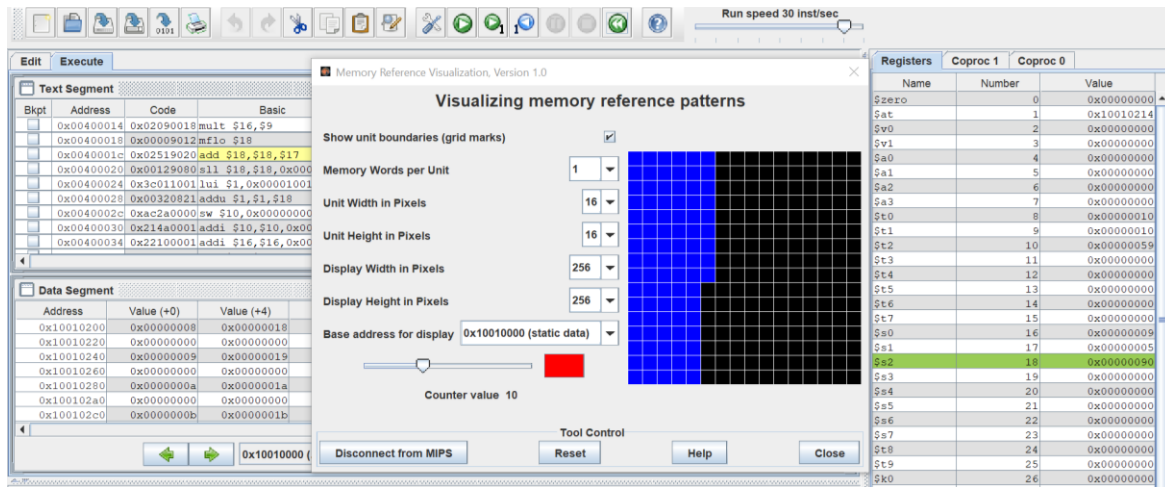
1. Với file `row-major.asm`



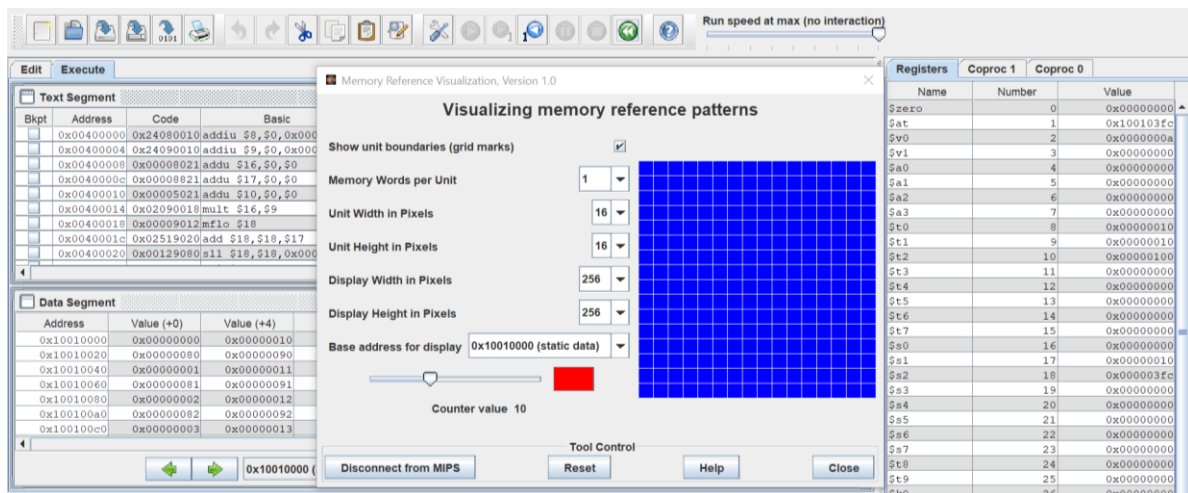
- ⇒ Bộ nhớ được truy cập bằng cách duyệt qua từng cột của lần lượt các hàng 1, 2, 3, ...
- ⇒ Khi chạy xong toàn bộ chương trình toàn bộ các phần tử được truy cập và mỗi phần tử được truy cập đúng một lần



2. Với file `column-major.asm`

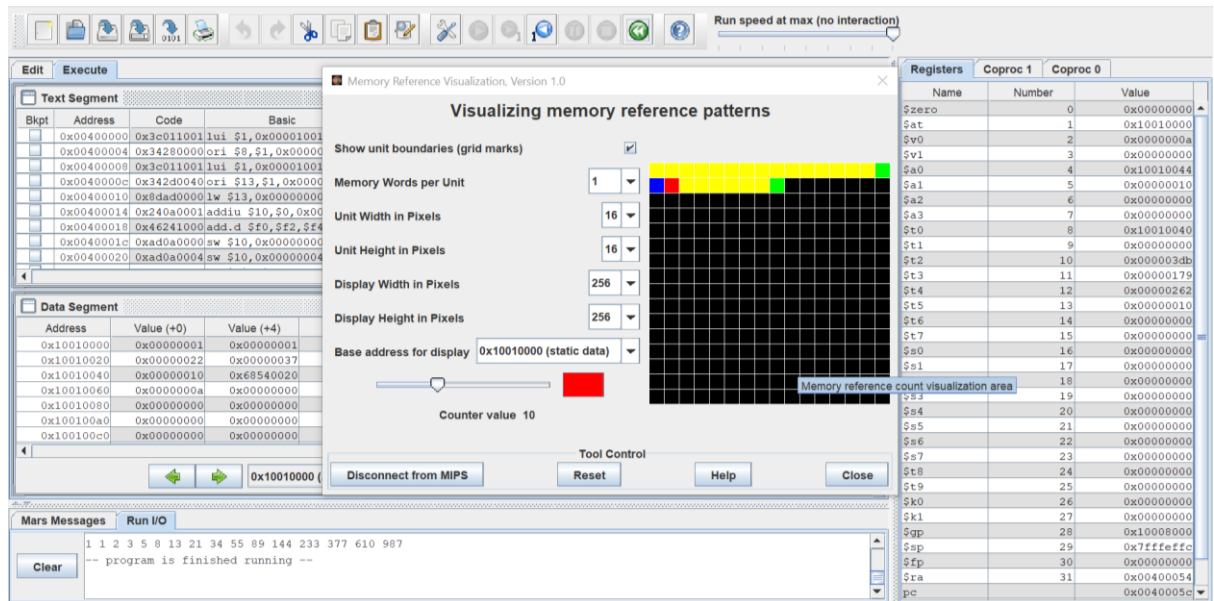


- ⇒ Bộ nhớ được truy cập bằng cách duyệt qua từng hàng của lần lượt các cột 1, 2, 3, ...
- ⇒ Khi chạy xong toàn bộ chương trình toàn bộ các phần tử được truy cập và mỗi phần tử được truy cập đúng một lần



3. Với file `fibonacci.asm`

- Khi chạy toàn bộ chương trình:



- Khi chỉ chạy phần tính dãy fibonacci (bỏ qua print):

