

# REPORT WEEK 12

Họ và tên: Nguyễn Văn Hưng

MSSV: 20225634

Lớp: Thực hành kiến trúc máy tính – Mã lớp: 147799

## I. Assignment 1:

### Running the Data Cache Simulator tool

1. Close any MIPS programs you are currently using.
2. Open the program **row-major.asm** from the `Examples` folder. This program will traverse a 16 by 16 element integer matrix in row-major order, assigning elements the values 0 through 255 in order. It performs the following algorithm:

```
for (row = 0; row < 16; row++)  
for (col = 0; col < 16; col++)  
    data[row][col] = value++;
```

Code:

```
#####  
#####  
# Row-major order traversal of 16 x 16 array of  
words.  
# Pete Sanderson  
# 31 March 2007  
#  
# To easily observe the row-oriented order, run the  
Memory Reference  
# Visualization tool with its default settings over this  
program.  
# You may, at the same time or separately, run the  
Data Cache Simulator  
# over this program to observe caching performance.  
Compare the results  
# with those of the column-major order traversal  
algorithm.  
#  
# The C/C++/Java-like equivalent of this MIPS  
program is:  
# int size = 16;  
# int[size][size] data;  
# int value = 0;  
# for (int row = 0; row < size; row++) {  
#     for (int col = 0; col < size; col++) {  
#         data[row][col] = value;
```

```

#     value++;
# }
# }
#
# Note: Program is hard-wired for 16 x 16 matrix. If
# you want to change this,
#     three statements need to be changed.
#     1. The array storage size declaration at "data:"
#     needs to be changed from
#         256 (which is 16 * 16) to #columns * #rows.
#     2. The "li" to initialize $t0 needs to be changed
#     to new #rows.
#     3. The "li" to initialize $t1 needs to be changed
#     to new #columns.
#
# .data
data: .word 0 : 256    # storage for 16x16
matrix of words
# .text
li    $t0, 16        # $t0 = number of rows
li    $t1, 16        # $t1 = number of columns
move  $s0, $zero     # $s0 = row counter
move  $s1, $zero     # $s1 = column counter
move  $t2, $zero     # $t2 = the value to be
stored
# Each loop iteration will store incremented $t1
# value into next element of matrix.
# Offset is calculated at each iteration. offset = 4 *
# (row*#cols+col)
# Note: no attempt is made to optimize runtime
# performance!
loop: mult  $s0, $t1    # $s2 = row * #cols (two-
instruction sequence)
      mflo  $s2        # move multiply result from
lo register to $s2
      add  $s2, $s2, $s1 # $s2 += column counter
      sll  $s2, $s2, 2   # $s2 *= 4 (shift left 2 bits)
for byte offset
      sw   $t2, data($s2) # store the value in matrix
element
      addi $t2, $t2, 1   # increment value to be
stored
# Loop control: If we increment past last column,
# reset column counter and increment row counter
#     If we increment past last row, we're
# finished.

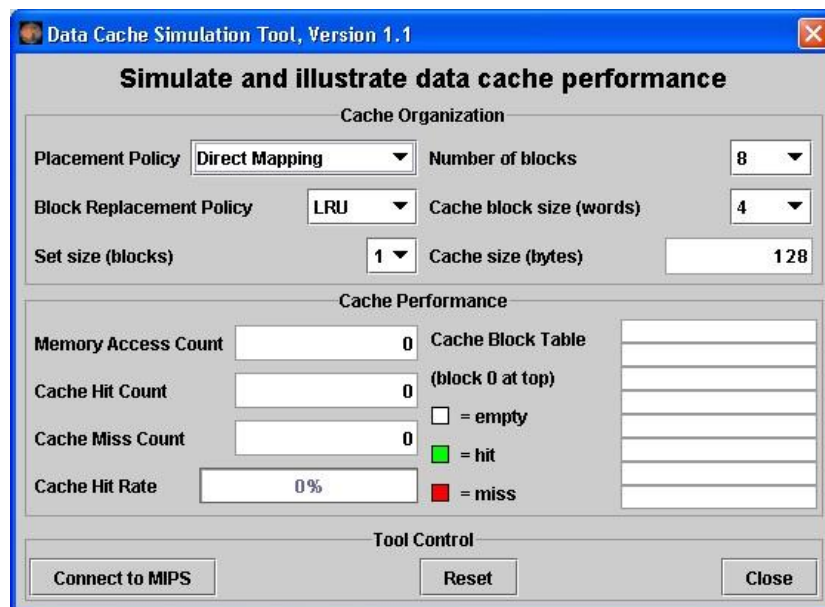
```

```

        addi    $s1, $s1, 1    # increment column
counter
        bne     $s1, $t1, loop # not at end of row so loop
back
        move    $s1, $zero    # reset column counter
        addi    $s0, $s0, 1    # increment row counter
        bne     $s0, $t0, loop # not at end of matrix so
loop back
# We're finished traversing the matrix.
        li      $v0, 10        # system service 10 is exit
        syscall                # we are outta here.

```

3. Assemble the program.
4. From the **Tools** menu, select **Data Cache Simulator**. A new frame will appear in the middle of the screen.





This is a MARS Tool that will simulate the use and performance of cache memory when the underlying MIPS program executes. Notice its three major sections:

- *Cache Organization:* You can use the combo boxes to specify how the cache will be configured for this run. Feel free to explore the different settings, but the default is fine for now.
- *Cache Performance:* With each memory access during program execution, the simulator will determine whether or not that access can be satisfied from cache and update the performance display accordingly.
- *Tool Control:* These buttons perform generic control functions as described by their labels.

- Click the tool's **Connect to MIPS** button. This causes the tool to register as an observer of MIPS memory and thus respond during program execution.
- Back in MARS, adjust the **Run Speed slider** to 30 instructions per second. It is located at the right side of the toolbar. This slows execution so you can watch the Cache Performance animation.



- In MARS, run the program using the **Run** toolbar button , the menu item or keyboard shortcut. Watch the Cache Performance animate as it is updated with every access to MIPS memory.
- What was the final cache hit rate? **75%*** . 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.

 Data Cache Simulation Tool, Version 1.2

### Simulate and illustrate data cache performance

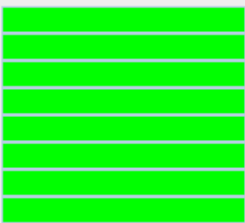
**Cache Organization**

Placement Policy: Direct Mapping Number of blocks: 8

Block Replacement Policy: LRU Cache block size (words): 4

Set size (blocks): 1 Cache size (bytes): 128

**Cache Performance**

Memory Access Count: 256 Cache Block Table: 

Cache Hit Count: 192 (block 0 at top)

Cache Miss Count: 64 ☐ = empty

Cache Hit Rate: 75% ☒ = hit

☐ = miss

**Runtime Log**

☐ Enabled

**Tool Control**

Disconnect from MIPS Reset Close

9. Given that explanation, *what do you predict the hit rate will be if the block size is increased from 4 words to 8 words?* **88%.**  
*Decreased from 4 words to 2 words?* **50%.**

The image shows two side-by-side windows of the 'Data Cache Simulation Tool, Version 1.2'. Both windows are titled 'Simulate and illustrate data cache performance'.

**Left Window (8 words block size):**


- 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 Hit Count: 224
  - Cache Miss Count: 32
  - Cache Hit Rate: 88%
  - Cache Block Table: (block 0 at top) 8 green bars, 0 empty, 0 miss.
- Runtime Log: Disabled
- Tool Control: Disconnect from MIPS, Reset, Close

**Right Window (2 words block size):**

- 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: 768
  - Cache Hit Count: 384
  - Cache Miss Count: 384
  - Cache Hit Rate: 50%
  - Cache Block Table: (block 0 at top) 8 green bars, 0 empty, 0 miss.
- Runtime Log: Disabled
- Tool Control: Disconnect from MIPS, Reset, Close

10. Verify your predictions by modifying the block size and re-running the program from step 7.

*NOTE:* when you modify the Cache Organization, the performance values are automatically reset (you can always use the tool's **Reset** button).

*NOTE:* You have to **reset**  the MIPS program before you can re-run it. *NOTE:* Feel free to adjust the **Run Speed slider** to maximum speed anytime you want.

11. Repeat steps 2 through 10 for program **column-major.asm** from the Examples folder. This program will traverse a 16 by 16 element integer matrix in column-major order, assigning elements the values 0 through 255 in order. It performs the following algorithm:

```
for (col = 0; col < 16; col++)
for (row = 0; row < 16; row++)
data[row][col] = value++;
```

*NOTE:* You can leave the Cache Simulator in place, move it out of the way, or close it. It will not interfere with the actions needed to open, assemble, or run this new program and will remain connected to MIPS memory. If you do not close the tool, then skip steps 4 and 5.

Code:

```
#####
#
# Column-major order traversal of 16 x 16 array of words.
# Pete Sanderson
# 31 March 2007
```

```

#
# To easily observe the column-oriented order, run the Memory Reference
# Visualization tool with its default settings over this program.
# You may, at the same time or separately, run the Data Cache Simulator
# over this program to observe caching performance. Compare the results
# with those of the row-major order traversal algorithm.
#
# The C/C++/Java-like equivalent of this MIPS program is:
#   int size = 16;
#   int[size][size] data;
#   int value = 0;
#   for (int col = 0; col < size; col++) {
#       for (int row = 0; row < size; row++) {
#           data[row][col] = value;
#           value++;
#       }
#   }
#
# Note: Program is hard-wired for 16 x 16 matrix. If you want to change this,
#       three statements need to be changed.
#       1. The array storage size declaration at "data:" needs to be changed from
#          256 (which is 16 * 16) to #columns * #rows.
#       2. The "li" to initialize $t0 needs to be changed to the new #rows.
#       3. The "li" to initialize $t1 needs to be changed to the new #columns.
#
.data
data: .word 0 : 256    # 16x16 matrix of words
.text
li    $t0, 16        # $t0 = number of rows
li    $t1, 16        # $t1 = number of columns
move  $s0, $zero     # $s0 = row counter
move  $s1, $zero     # $s1 = column counter
move  $t2, $zero     # $t2 = the value to be stored
# Each loop iteration will store incremented $t1 value into next element of
# matrix.
# Offset is calculated at each iteration. offset = 4 * (row*#cols+col)
# Note: no attempt is made to optimize runtime performance!
loop: mult  $s0, $t1   # $s2 = row * #cols (two-instruction sequence)
      mflo  $s2       # move multiply result from lo register to $s2
      add   $s2, $s2, $s1 # $s2 += col counter
      sll   $s2, $s2, 2  # $s2 *= 4 (shift left 2 bits) for byte offset
      sw    $t2, data($s2) # store the value in matrix element
      addi  $t2, $t2, 1  # increment value to be stored
# Loop control: If we increment past bottom of column, reset row and increment
# column


```

```

#           If we increment past the last column, we're finished.
addi    $s0, $s0, 1    # increment row counter
bne     $s0, $t0, loop # not at bottom of column so loop back
move    $s0, $zero     # reset row counter
addi    $s1, $s1, 1    # increment column counter
bne     $s1, $t1, loop # loop back if not at end of matrix (past the last
column)
# We're finished traversing the matrix.
li      $v0, 10        # system service 10 is exit
syscall                # we are outta here.

```

12. What was the cache performance for this program? **0%**. 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.

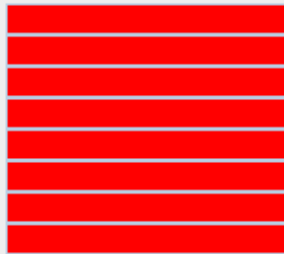
 Data Cache Simulation Tool, Version 1.2 ✕

### Simulate and illustrate data cache performance

**Cache Organization**

Placement Policy	Direct Mapping ▼	Number of blocks	8 ▼
Block Replacement Policy	LRU ▼	Cache block size (words)	4 ▼
Set size (blocks)	1 ▼	Cache size (bytes)	128

**Cache Performance**

Memory Access Count	256	Cache Block Table	
Cache Hit Count	0	(block 0 at top)	
Cache Miss Count	256	<input type="checkbox"/> = empty <input checked="" type="checkbox"/> = hit <input checked="" type="checkbox"/> = miss	
Cache Hit Rate	0%		

**Runtime Log**

☐ Enabled

**Tool Control**

Disconnect from MIPS
Reset
Close

13. Change the block size to 16. Note this will reset the tool.

**Data Cache Simulation Tool, Version 1.2**

**Simulate and illustrate data cache performance**

**Cache Organization**

Placement Policy: Direct Mapping (dropdown)  
 Block Replacement Policy: LRU (dropdown)  
 Set size (blocks): 1 (dropdown)  
 Number of blocks: 8 (dropdown)  
 Cache block size (words): 16 (dropdown)  
 Cache size (bytes): 512 (text box)

**Cache Performance**

Memory Access Count: 256 (text box)  
 Cache Hit Count: 0 (text box)  
 Cache Miss Count: 256 (text box)  
 Cache Hit Rate: 0% (text box)

Cache Block Table (block 0 at top):  
☐ = empty  
☒ = hit  
☒ = miss

**Runtime Log**

☐ Enabled

**Tool Control**

Disconnect from MIPS (button)   Reset (button)   Close (button)

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.

**Data Cache Simulation Tool, Version 1.2**

**Simulate and illustrate data cache performance**

**Cache Organization**

Placement Policy: Direct Mapping (dropdown)  
 Block Replacement Policy: LRU (dropdown)  
 Set size (blocks): 1 (dropdown)  
 Number of blocks: 16 (dropdown)  
 Cache block size (words): 16 (dropdown)  
 Cache size (bytes): 1024 (text box)

**Cache Performance**

Memory Access Count: 256 (text box)  
 Cache Hit Count: 240 (text box)  
 Cache Miss Count: 16 (text box)  
 Cache Hit Rate: 94% (text box)

Cache Block Table (block 0 at top):  
☐ = empty  
☒ = hit  
☒ = miss

**Runtime Log**

☐ Enabled

**Tool Control**

Disconnect from MIPS (button)   Reset (button)   Close (button)

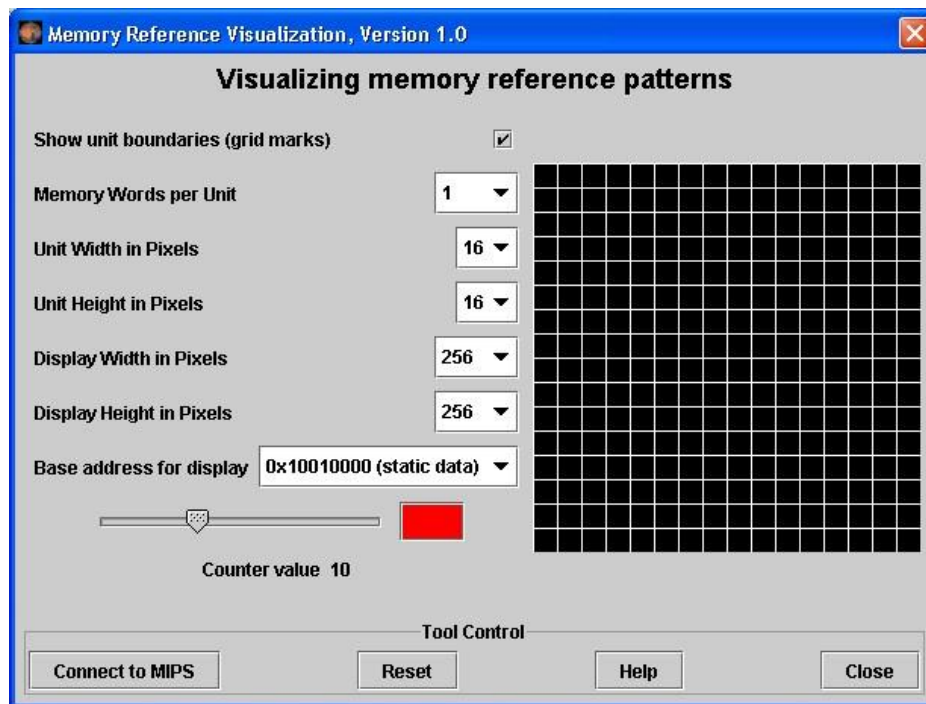


15. Re-run the program. *What is the cache performance of the original tool instance?* **0%**. Block size 16 didn't help because there was still only one access to each block, the initial miss, before that block was replaced with a new one. *What is the cache performance of the second tool instance?* **94%**. 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.

## II. Assignment 2:

### The Memory Reference Visualization tool

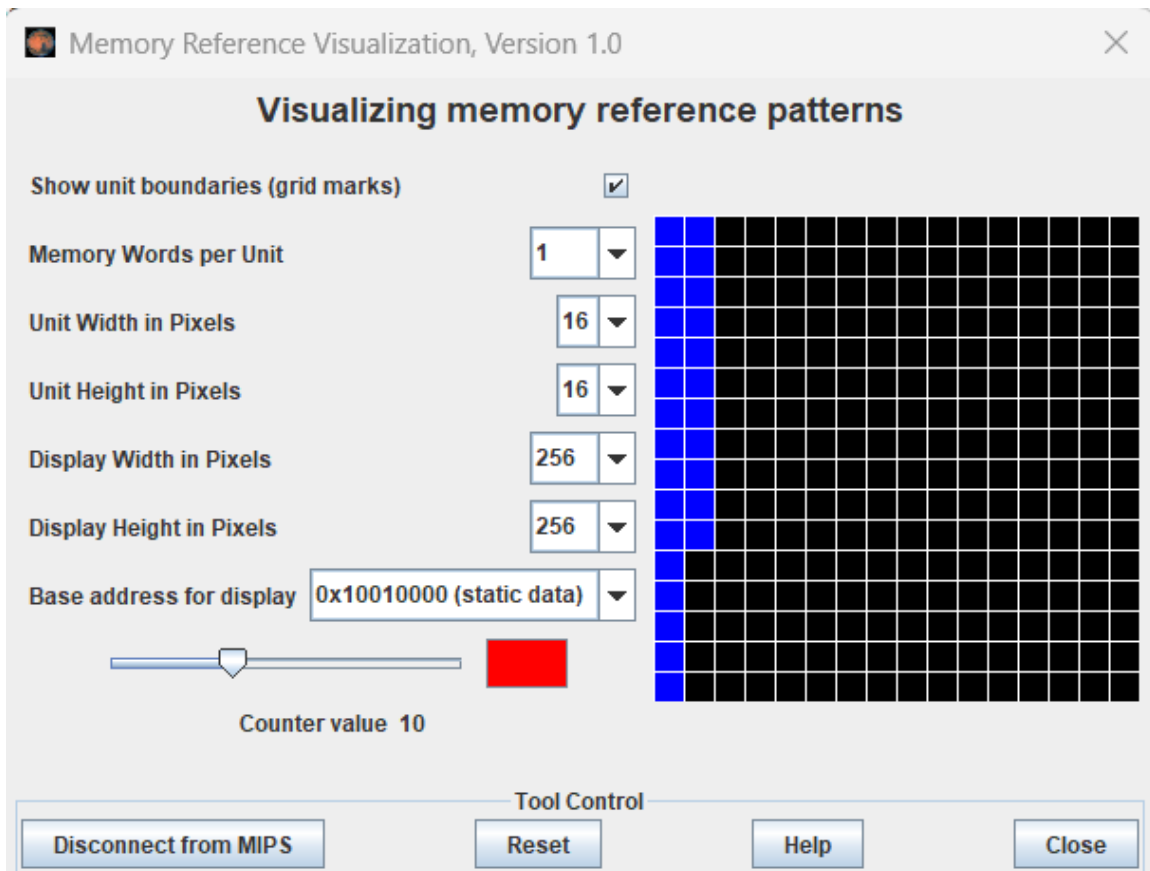
1. Open the program **row-major.asm** from the `Examples` folder if it is not already open.
2. Assemble the program.
3. From the **Tools** menu, select **Memory Reference Visualization**. A new frame will appear in the middle of the screen.



This tool will paint a grid unit each time the corresponding MIPS memory word is referenced. The base address, the first static data segment (`.data` directive) word, corresponds to the upper-left grid unit. Address correspondence continues in rowmajor order (left to right, then next row down).

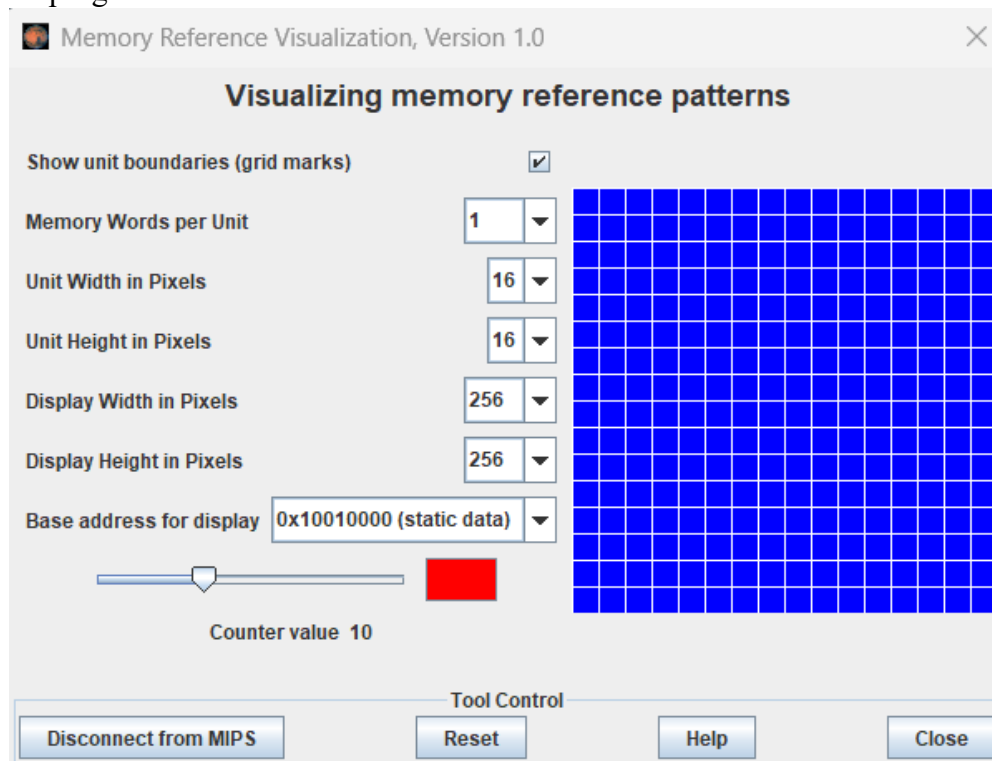
The color depends on the number of times the word has been referenced. Black is 0, blue is 1, green is 2, yellow is 3 and 4, orange is 5 through 9, red is 10 or higher. View the scale using the tool's slider control. You can change the color (but not the reference count) by clicking on the color patch.

4. Click the tool's **Connect to MIPS** button. This causes the tool to register as an observer of MIPS memory and thus respond during program execution.
5. Back in MARS, adjust the **Run Speed slider** to 30 instructions per second.
6. Run the program. Watch the tool animate as it is updated with every access to MIPS memory. *Feel free to stop the program at any time.*

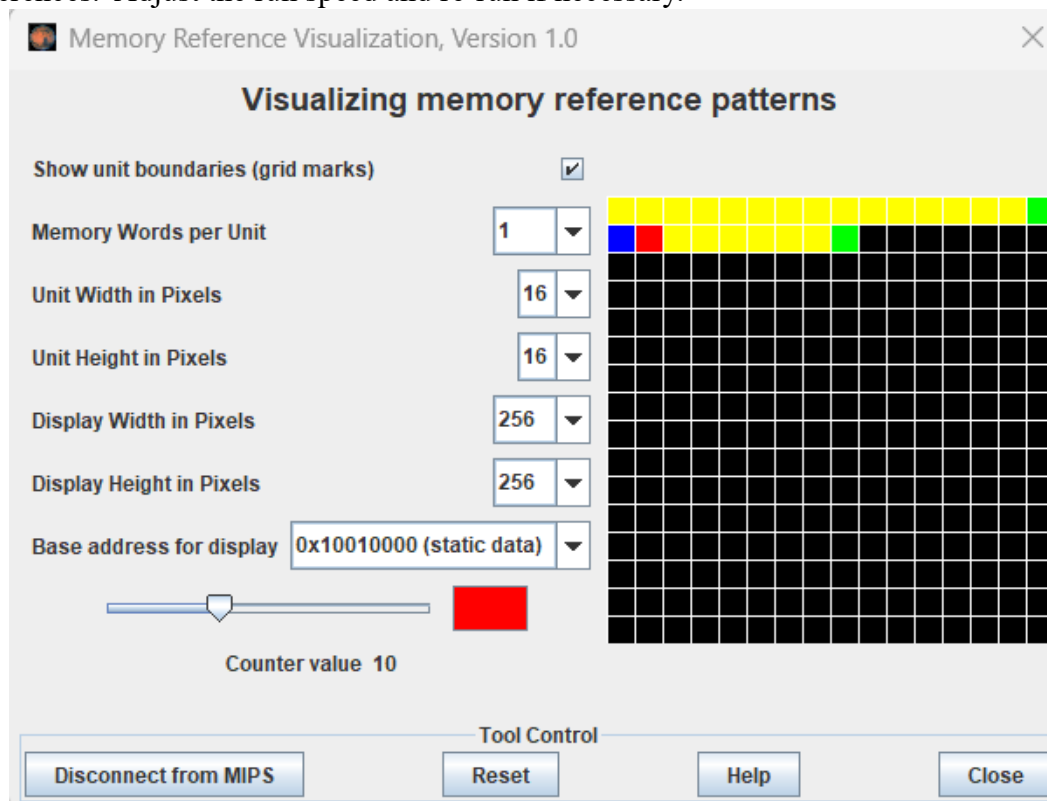


7. Hopefully you observed that the animation sequence corresponded to the expected memory access sequence of the row-major.asm program. *If you have trouble seeing the blue, reset the tool, move the slider to position 1, change the color to something brighter, and re-run.*

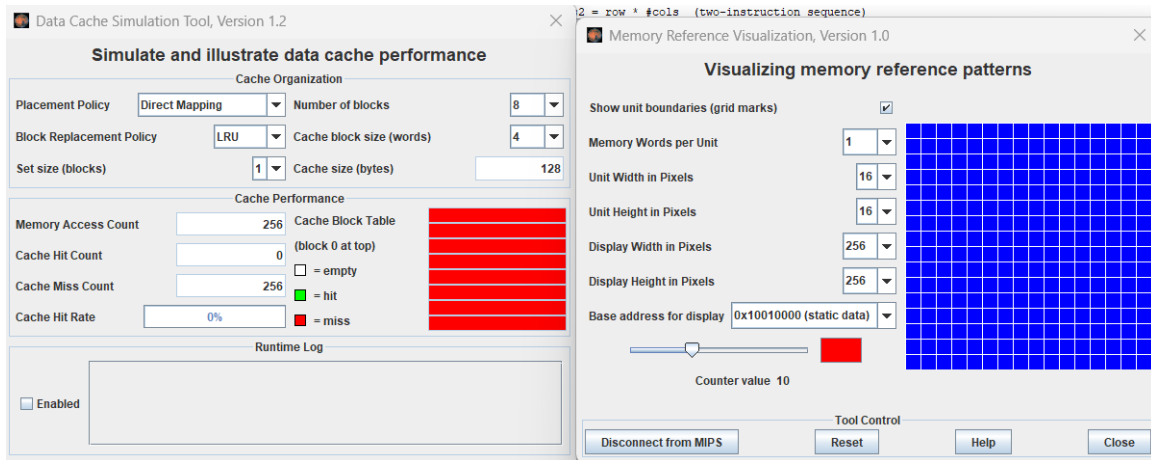
8. Repeat steps 2 through 7, for **column-major.asm**. You should observe that the animation sequence corresponded to the expected memory access sequence of this program.



9. Repeat again for **fibonacci.asm** to observe the animated pattern of memory references. Adjust the run speed and re-run if necessary.



10. (Optional) Create a new instance of the Data Cache Simulator. Move the two frames around so you can see both. Connect the cache simulator to MIPS and reset the Memory Reference Visualization. Re-run the program. This exercise illustrates that two different tools can be used simultaneously.



The Memory Reference Visualization tool could be useful in an operating systems course to illustrate spatial and temporal locality and memory reference patterns in general.