Lab 12 Report
Nguyễn Khánh Nam
20225749

Code: row-major.asm

```
#############################################################################
###########
#  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; col < 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.
```

Code: colum-major

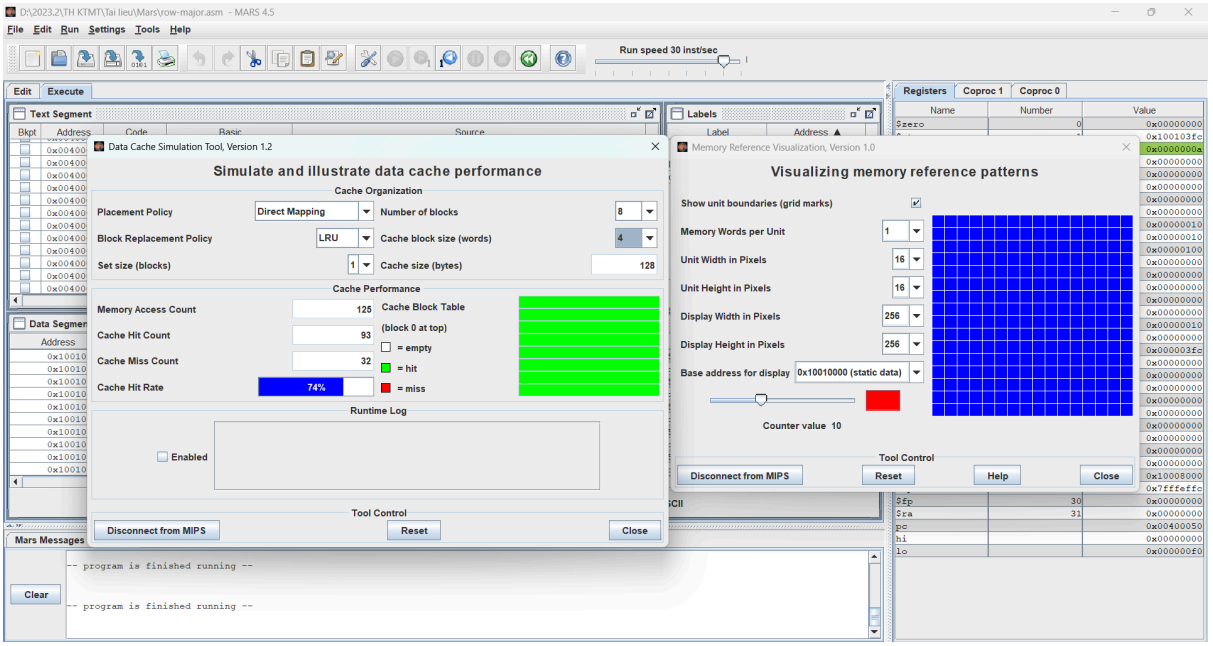
```
################################################################
#
#  Column-major order traversal of 16 x 16 array of words.
#  Pete Sanderson
#  31 March 2007
#
#  To easily observe the column-oriented order, run the Memory Reference
#  Visualization tool with its default settings over this program.
#  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.
```

Result:

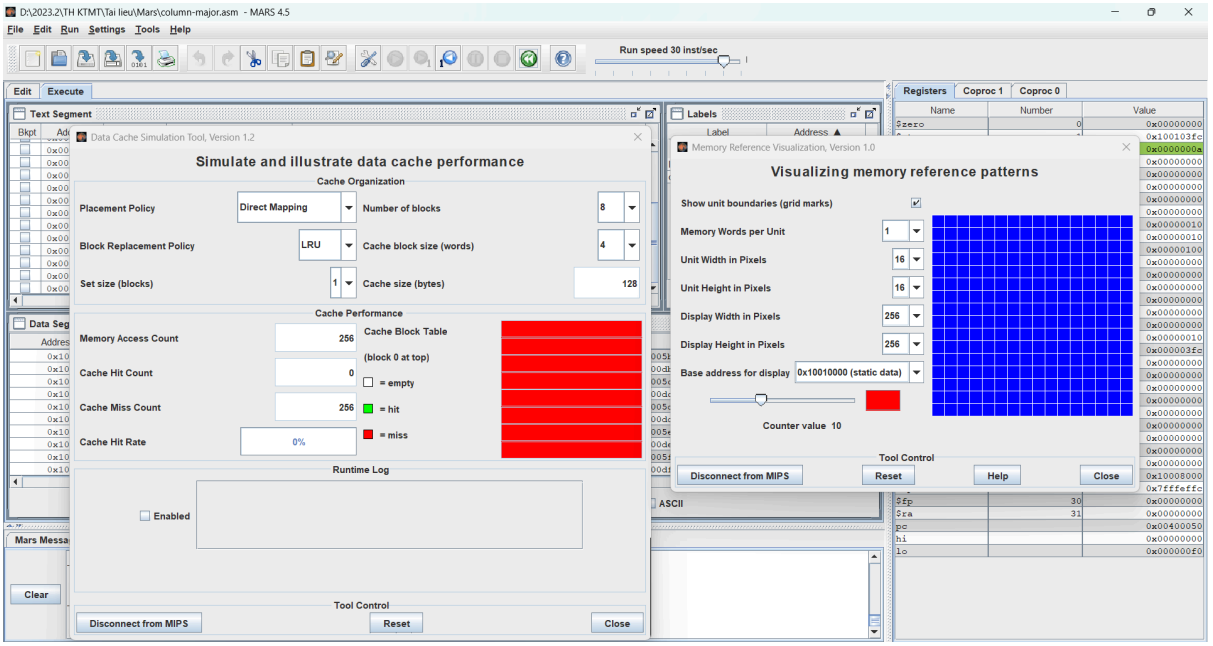
row-major:



- The final cache hit rate: ¾
    + Mỗi lần cache miss, sẽ có 1 khối 4 words được đọc lưu vào cache.
    + Trong truy vấn theo hàng, các phần tử mảng cũng được truy cập lần lượt theo đúng thứ tự lưu nên trong 1 khối có 4 word thì sẽ có 3 words trùng đã được đọc trong khối từ trước đó trong lần truy cập vào ô tiếp theo.
    + Sau đó đến ô thứ 5 -> cache miss nên tự lưu trước 4 ô từ ô thư 5.
    + Lặp lại như trên.

- Increase block size to 8 words: cache hits = ⅞
- Decrease to 2 words: cache hits = ½

column-major

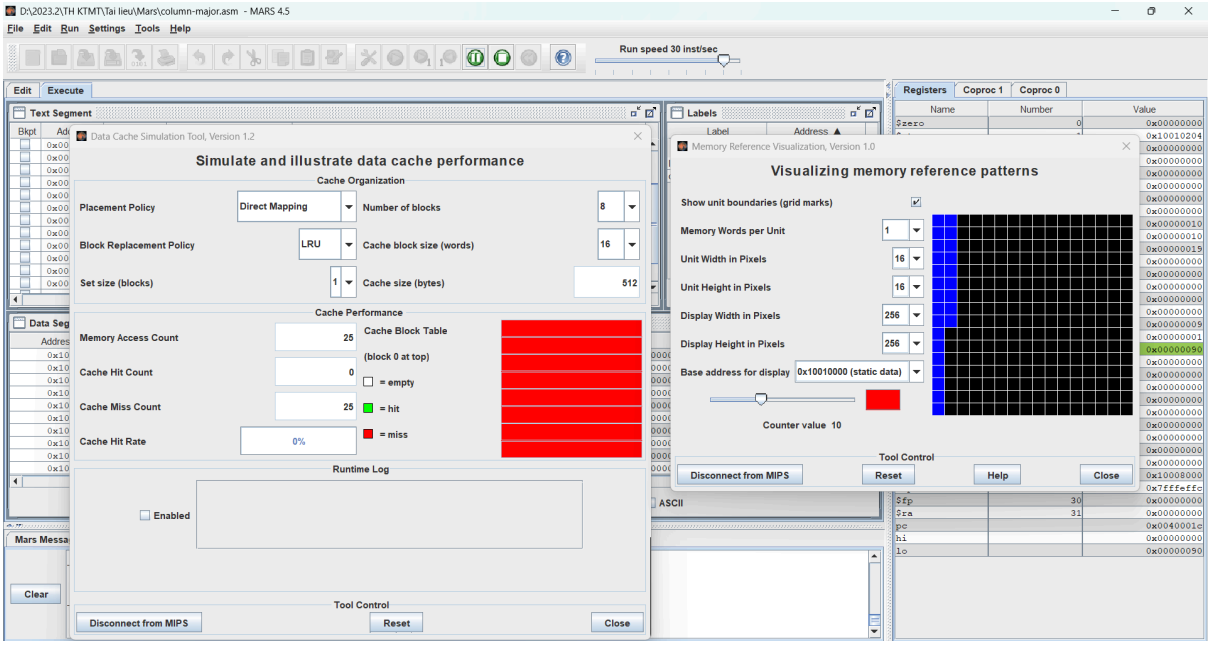


- The final cache hit rate: 0/4
    + Mỗi lần cache miss, sẽ có 1 khối 4 words được đọc lưu vào cache.
    + Do truy cập dữ liệu theo cột nên sẽ liên tục bị cache miss do 1 khối 4 words được lưu theo hàng mỗi lần cache miss.

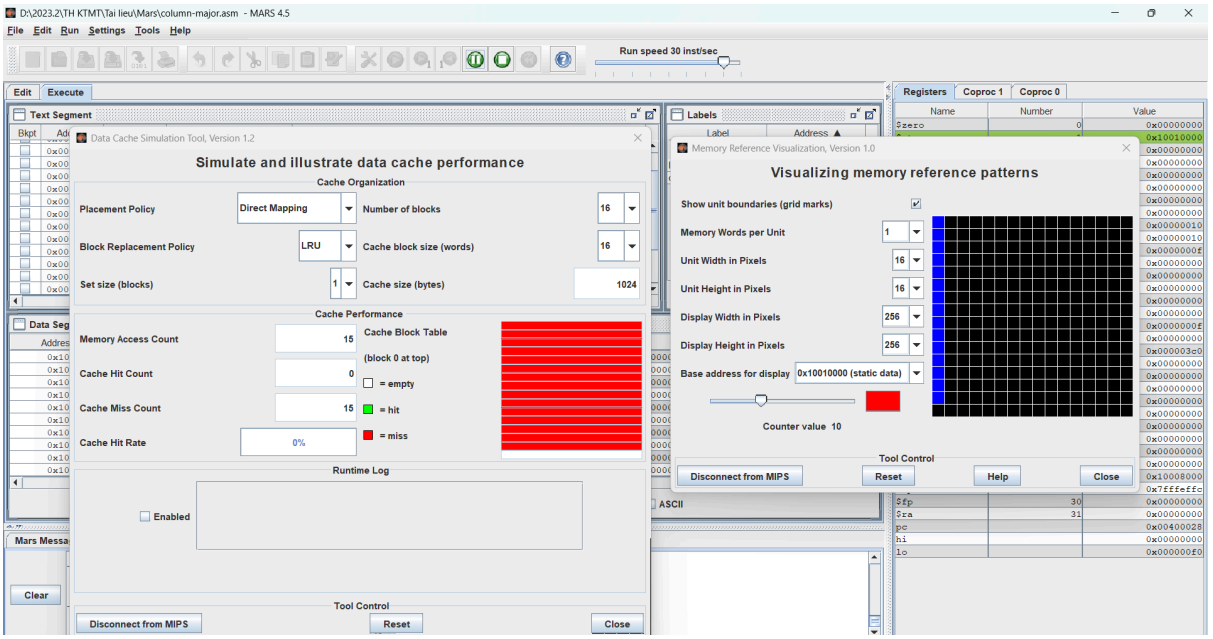
- Increase block size to 8 words: cache hits = 0
- Decrease to 2 words: cache hits = 0

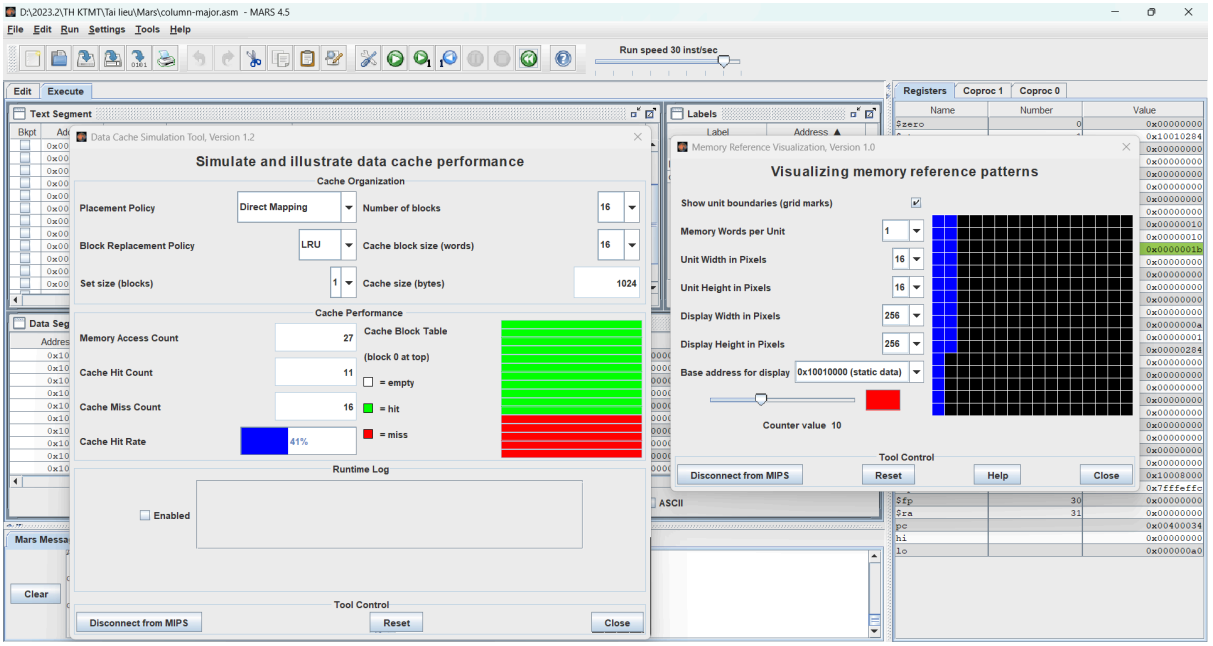
- The cache performance for this program: Truy cập từng words theo cột nên mỗi word được truy cập sẽ cách word trong block được lưu vào cache 16 words -> cache miss (dữ liệu trong block bị thay thế dù chưa được truy cập)

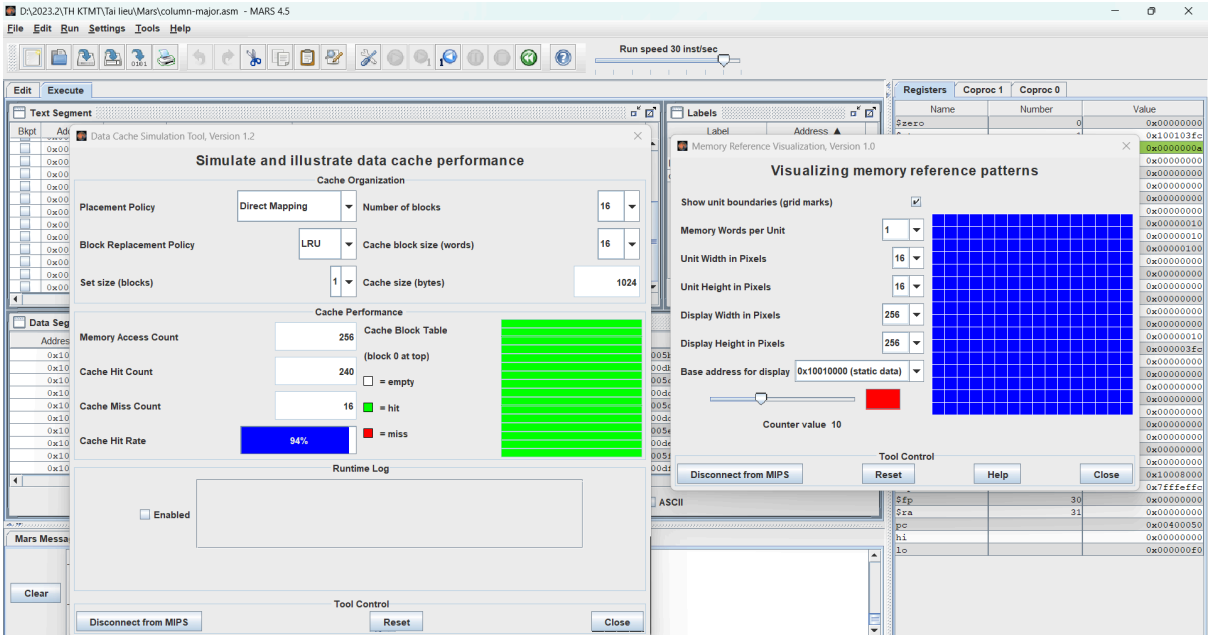
- Block size tăng lên 16 nhưng số lượng block vẫn là 8 nên không đủ để cho dữ liệu truy cập được cache hits. Do chưa kịp truy cập tới dữ liệu cần dùng thì đã bị thay thế sau khi hết 8 block



- Block size tăng lên 16 cùng số lượng Block cũng được tăng lên 16 mà mảng 16x16 nên các block sẽ được lưu hết vào cache có đủ số lượng block 16 words cho cả mảng nên sau khi duyệt hết cột đầu tiên từ cột thứ 2 sẽ có cache hits lần lượng các block.




D:\2023.2\TH KTMT\Tai lieu\Mars\column-major.asm - MARS 4.5
File Edit Run Settings Tools Help
Run speed 30 inst/sec
Edit Execute
Text Segment
Data Cache Simulation Tool, Version 1.2
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
27
Cache Hit Count
11
Cache Miss Count
16
Cache Hit Rate
41%
Cache Block Table
(block 0 at top)
= empty
= hit
= miss
Runtime Log
Enabled
Tool Control
Disconnect from MIPS
Reset
Close
Labels
Memory Reference Visualization, Version 1.0
Visualizing memory reference patterns
Show unit boundaries (grid marks)
Memory Words per Unit
Unit Width in Pixels
Unit Height in Pixels
Display Width in Pixels
256
Display Height in Pixels
256
Base address for display
0x10010000 (static data)
Counter value 10
Help
Registers
Coproc 1
Coproc 0
Name
Number
Value
ASCII
Mars Messa
Clear


D:\2023.2\TH KTMT\Tai lieu\Mars\column-major.asm - MARS 4.5
File Edit Run Settings Tools Help
Run speed 30 inst/sec
Edit Execute
Text Segment
Data Cache Simulation Tool, Version 1.2
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
(block 0 at top)
= empty
= hit
= miss
Runtime Log
Enabled
Tool Control
Disconnect from MIPS
Reset
Close
Labels
Memory Reference Visualization, Version 1.0
Visualizing memory reference patterns
Show unit boundaries (grid marks)
Memory Words per Unit
Unit Width in Pixels
Unit Height in Pixels
Display Width in Pixels
256
Display Height in Pixels
256
Base address for display
0x10010000 (static data)
Counter value 10
Help
Registers
Coproc 1
Coproc 0
Name
Number
Value
ASCII
Mars Messa
Clear