**Lab Report  
Lab 12: Cache memory**

|  |  |
| --- | --- |
| Student ID | Nguyễn Đức Anh |
| Student Name | 20235890 |

**Assignment 1:**

**A,**

**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

         la         a0, data

         mv     s0, zero     # $s0 = row counter

         mv     s1, zero     # $s1 = column counter

         mv     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:

         mul      s2, s0, t1       # $s2 = row \* #cols  (two-instruction sequence)

         add      s2, s2, s1  # $s2 += column counter

         slli      s2, s2, 2    # $s2 \*= 4 (shift left 2 bits) for byte offset

         add    s2, a0, s2

         sw       t2, 0(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

         mv     s1, zero     # reset column counter

         addi     s0, s0, 1    # increment row counter

         bne      s0, t0, loop # not at end of matrix so loop back

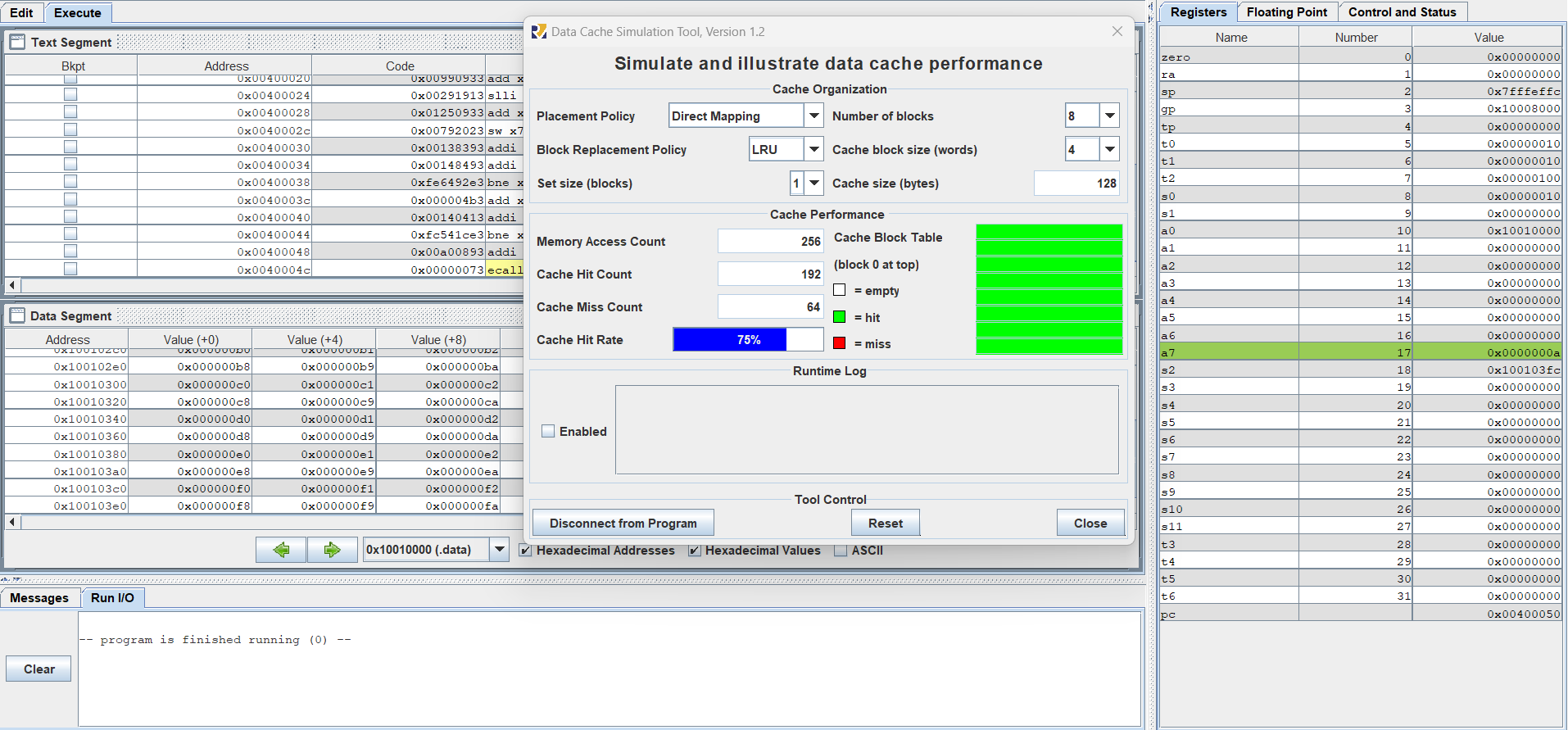
#  We're finished traversing the matrix.

         li       a7, 10        # system service 10 is exit

         ecall                 # we are outta here.

* The final cache hit rate is 75% on block size 4 words.
* The final cache hit rate is 87.5% (approximately 88%) on block size 8 words.
* The final cache hit rate is 50% on block size 2 words.

**Result:** (Input, output for each case; is the result same as the theory, etc…)



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**B,**

**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

         la     a0, data

         mv     s0, zero     # $s0 = row counter

         mv     s1, zero     # $s1 = column counter

         mv     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:

         mul     s2, s0, t1       # $s2 = row \* #cols  (two-instruction sequence)

         add      s2, s2, s1  # $s2 += col counter

         slli      s2, s2, 2    # $s2 \*= 4 (shift left 2 bits) for byte offset

         add        s2, a0, s2

         sw       t2, 0(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

         mv     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       a7, 10        # system service 10 is exit

         ecall                 # we are outta here.

* The final cache hit rate is 0% on block size 4 words.
* The final cache hit rate is 0% on block size 16 words.
* The final cache hit rate is 94% on number of block 16 and block size 16 words.

**Result:** (Input, output for each case; is the result same as the theory, etc…)

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Assignment 2:**

**A,**

**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

         la         a0, data

         mv     s0, zero     # $s0 = row counter

         mv     s1, zero     # $s1 = column counter

         mv     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:

         mul      s2, s0, t1       # $s2 = row \* #cols  (two-instruction sequence)

         add      s2, s2, s1  # $s2 += column counter

         slli      s2, s2, 2    # $s2 \*= 4 (shift left 2 bits) for byte offset

         add    s2, a0, s2

         sw       t2, 0(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

         mv     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       a7, 10        # system service 10 is exit

         ecall                 # we are outta here.

**Result:** (Input, output for each case; is the result same as the theory, etc…)

A computer screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**B,**

**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

         la     a0, data

         mv     s0, zero     # $s0 = row counter

         mv     s1, zero     # $s1 = column counter

         mv     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:

         mul     s2, s0, t1       # $s2 = row \* #cols  (two-instruction sequence)

         add      s2, s2, s1  # $s2 += col counter

         slli      s2, s2, 2    # $s2 \*= 4 (shift left 2 bits) for byte offset

         add        s2, a0, s2

         sw       t2, 0(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

         mv     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       a7, 10        # system service 10 is exit

         ecall                 # we are outta here.

**Result:** (Input, output for each case; is the result same as the theory, etc…)

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**C,**

**Code:**

# Compute first twelve Fibonacci numbers and put in array, then print

.data

      fibs: .word   0 : 16        # "array" of 12 words to contain fib values

      size: .word  16             # size of "array"

.text

      la   t0, fibs        # load address of array

      la   t5, size        # load address of size variable

      lw   t5, 0(t5)      # load array size

      li   t2, 1           # 1 is first and second Fib. number

      #add.d $f0, $f2, $f4

      sw   t2, 0(t0)      # F[0] = 1

      sw   t2, 4(t0)      # F[1] = F[0] = 1

      addi t1, t5, -2     # Counter for loop, will execute (size-2) times

loop:

      lw   t3, 0(t0)      # Get value from array F[n]

      lw   t4, 4(t0)      # Get value from array F[n+1]

      add  t2, t3, t4    # $t2 = F[n] + F[n+1]

      sw   t2, 8(t0)      # Store F[n+2] = F[n] + F[n+1] in array

      addi t0, t0, 4      # increment address of Fib. number source

      addi t1, t1, -1     # decrement loop counter

      bgtz t1, loop        # repeat if not finished yet.

      la   a0, fibs        # first argument for print (array)

      add  a1, zero, t5  # second argument for print (size)

      jal  print            # call print routine.

      li   a7, 10          # system call for exit

      ecall               # we are out of here.

#########  routine to print the numbers on one line.

.data

      space:.asciz  " "          # space to insert between numbers

      head: .asciz  "The Fibonacci numbers are:\n"

.text

print:

      add  t0, zero, a0  # starting address of array

      add  t1, zero, a1  # initialize loop counter to array size

      la   a0, head        # load address of print heading

      li   a7, 4           # specify Print String service

      ecall               # print heading

out:

      lw   a0, 0(t0)      # load fibonacci number for syscall

      li   a7, 1           # specify Print Integer service

      ecall               # print fibonacci number

      la   a0, space       # load address of spacer for syscall

      li   a7, 4           # specify Print String service

      ecall               # output string

      addi t0, t0, 4      # increment address

      addi t1, t1, -1     # decrement loop counter

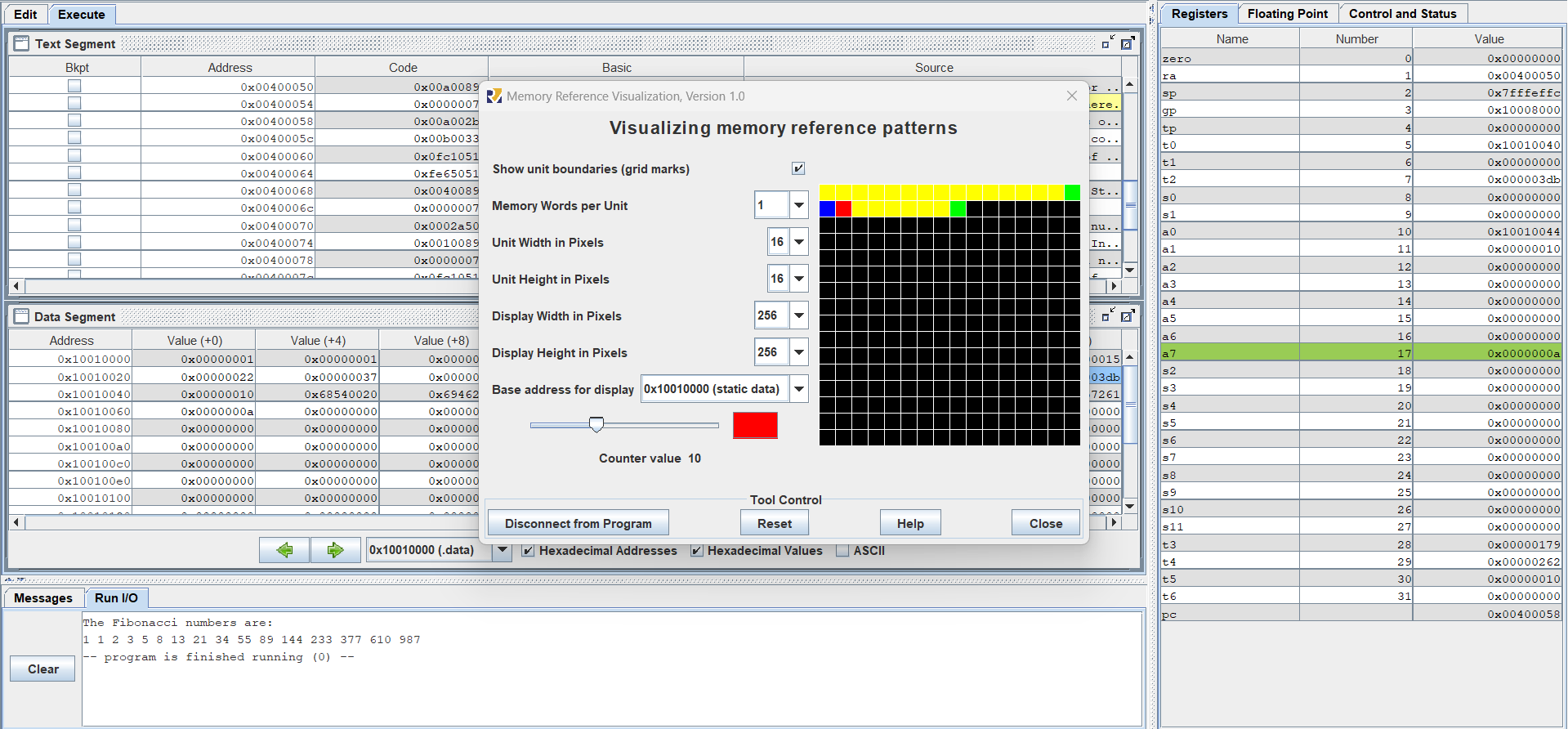
      bgtz t1, out         # repeat if not finished

      jr   ra              # return

**Result:** (Input, output for each case; is the result same as the theory, etc…)

A computer screen shot of a computer

Description automatically generated

****