TUHH
Technische Universität Hamburg

HAMBURG UNIVERSITY OF TECHNOLOGY

PROBLEM-BASED LEARNING

Advanced System-on-Chip Design

author
@tuhh.de

Documentation

Dozent
Dipl.-Ing. Wolfgang BRANDT

January 31, 2017

Contents

List of Figures

List of Tables

Listings

1 Introduction

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

2 Task 1 - Introduction Set Architecture of the MIPS-Processor

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

3 Task 2 - VHDL Introduction

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

4 Task 3 - MIPS Extension

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Table 1: Cache Simulation of Column Major

Placement (Policy)	Cache Block Size (Words)	Cache Hit Count	Cache Miss Count	Cache Hit Rate
Direct Mapping	2	0	256	0
Direct Mapping	4	0	256	0
Direct Mapping	8	0	256	0
Direct Mapping	16	0	256	0
2-Way Set Associative	2	0	256	0
2-Way Set Associative	4	0	256	0
2-Way Set Associative	8	0	512	0
2-Way Set Associative	16	0	256	0
4-Way Set Associative	2	0	256	0
4-Way Set Associative	4	0	256	0
4-Way Set Associative	8	0	256	0
4-Way Set Associative	16	0	256	0

Table 2: Cache Simulation of Row Major

Placement (Policy)	Cache Block Size (Words)	Cache Hit Count	Cache Miss Count	Cache Hit Rate
Direct Mapping	2	128	128	50
Direct Mapping	4	192	64	75
Direct Mapping	8	224	32	88
Direct Mapping	16	240	16	94
2-Way Set Associative	2	128	128	50
2-Way Set Associative	4	192	64	75
2-Way Set Associative	8	224	32	88
2-Way Set Associative	16	240	16	94
4-Way Set Associative	2	128	128	50
4-Way Set Associative	4	192	64	75
4-Way Set Associative	8	224	16	88
4-Way Set Associative	16	240	16	94

5 Task 4 - Caches

5.1 Cache Simulation - Results

The two assembler programs *row-major.asm* and *column-major.asm* has been used for the cache simulation. 1 contains the results regarding the file *column-major.asm* and 2 illustrates the results of *row-major.asm*.
TODO Interpretation

5.2 Design a Finite State Machine for the Cache

Table 3: Overview - FSM States

Abbreviation	Name	CPU Request Mode	Description
IDLE	-	-	-
CW	COMPARE WRITE	Write Request	-
CMW	CACHE MISS WRITE	Write Request	-
WBW	WRITE BACK WRITE	Write Request	-
WCW	WRITE CACHE WRITE	Write Request	-
CR	COMPARE READ	Read Request	-
CMR	CACHE MISS READ	Read Request	-
WBR	WRITE BACK READ	Read Request	-
WCR	WRITE CACHE READ	Read Request	-

Table 4: Overview - FSM Inputs

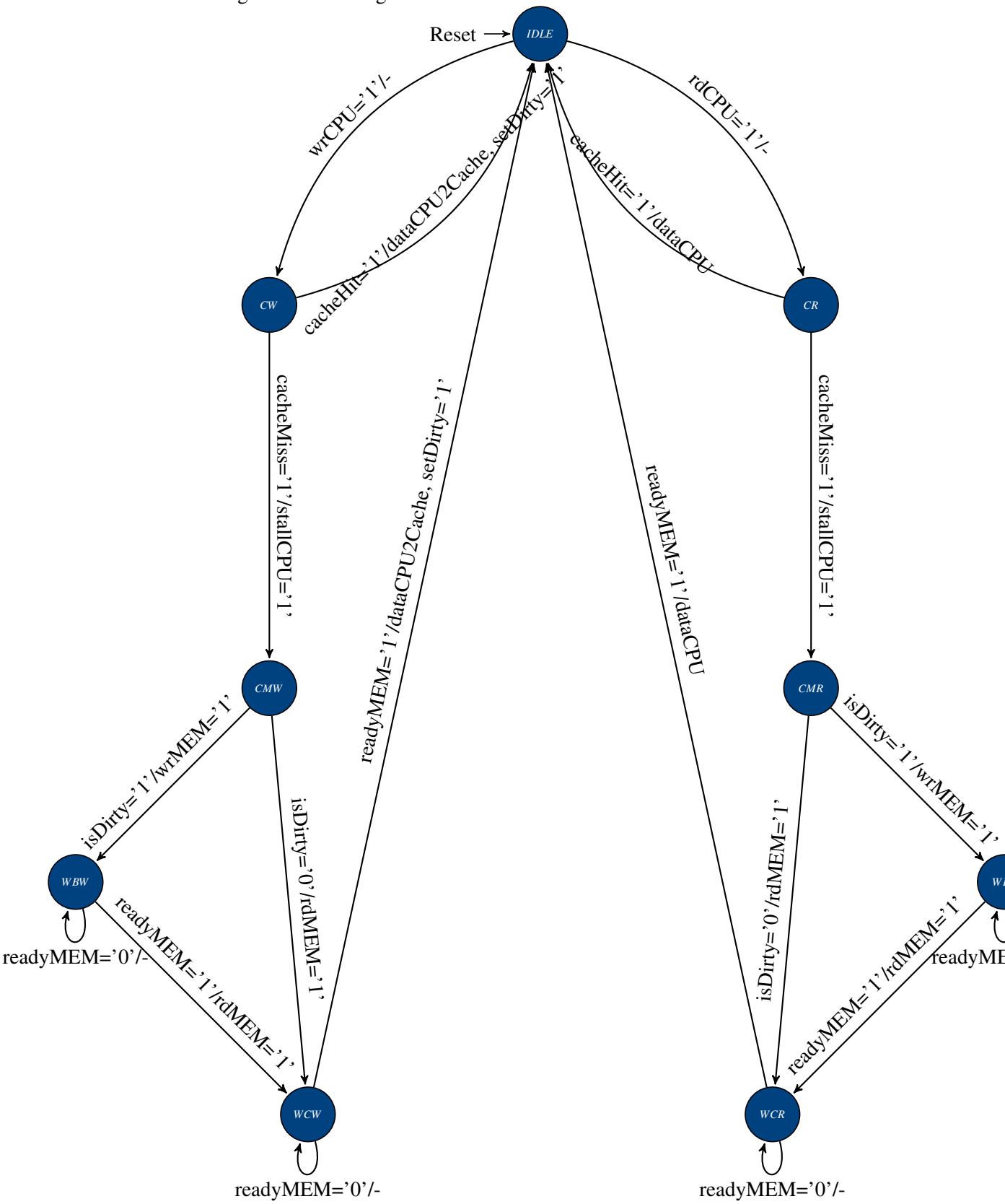
Abbreviation	Name	Description
rdCPU	CPU Read Request	-
wrCPU	CPU Write Request	-
cacheMiss	Cache Miss	-
cacheHit	Cache Hit	-
readyMEM	Write-Back is resolved	-
isDirty	Cache Block is dirty	-

In figure 1 the state diagram of the cache controller is illustrated. The state diagram represents a Mealy automaton. The state space of the state machine is given in table 3. Besides the state machine inputs are listed in table 4 and the state machine outputs are shown in table 5. A sketch of the state diagram is printed in figure 2.

Table 5: Overview - FSM Outputs

Abbreviation	Name	Description
stallCPU	Stall Processor	-
setDirty	Set Dirty Bit (Modified) Bit	-
wrMEM	Write To Memory	Write Replaced Block To Memory
dataCPU	Read Data Into CPU	-
rdMEM	Read Cache Block Into Cache From Memory	-
dataCPU2Cache	Write Data Into Cache	-

Figure 1: State diagram of the cache controller.



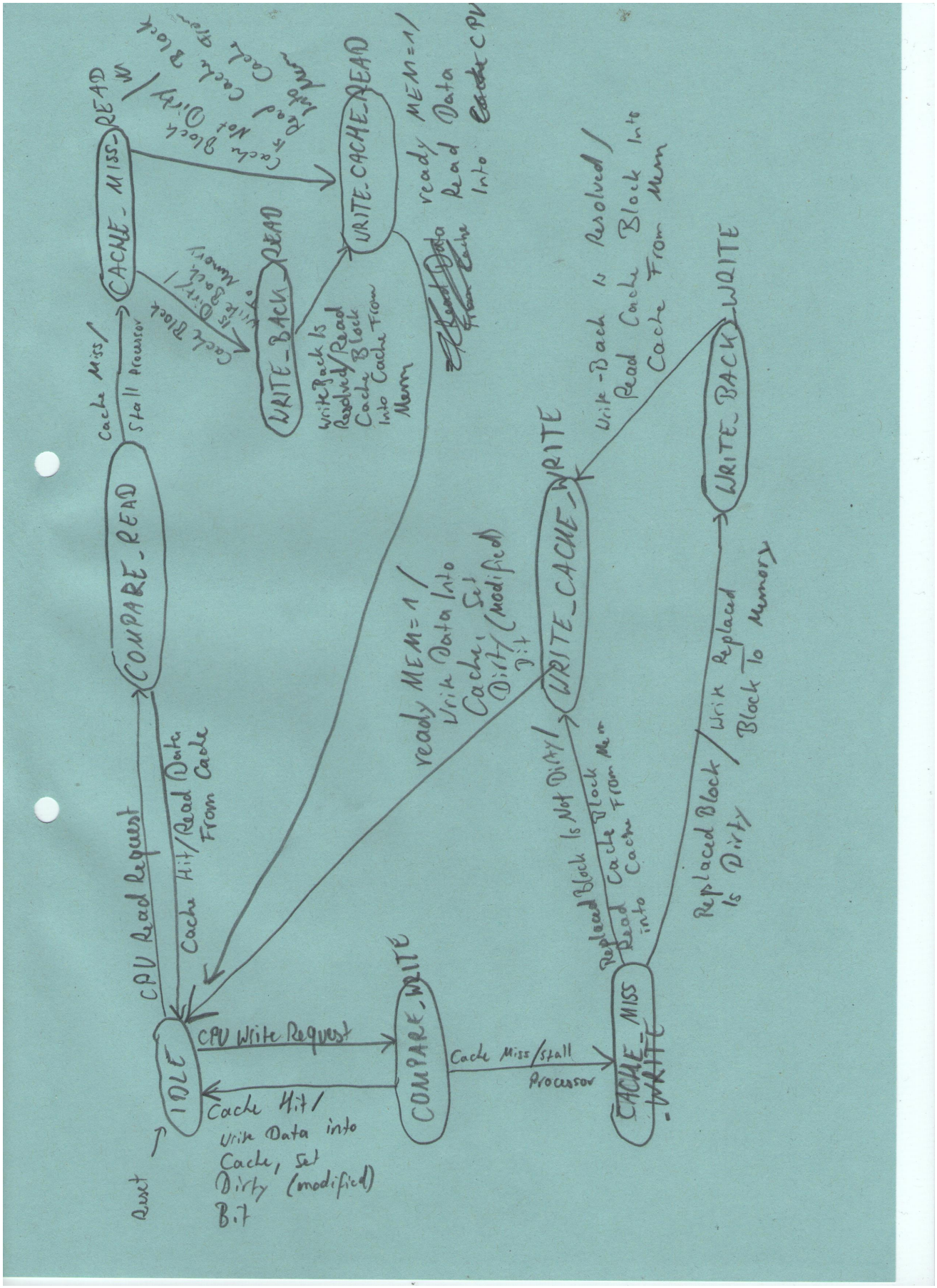
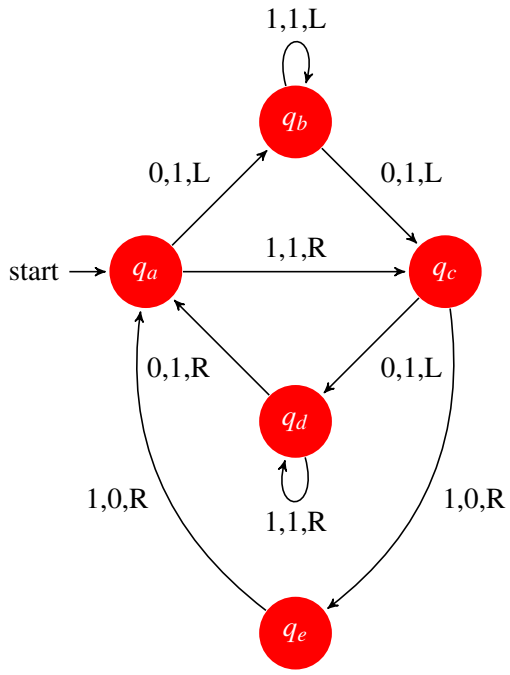


Figure 2: Sketch of Mealy Automata - Cache Controller

6 Appendix

6.1 Finite State Machine - Example



6.2 Implementation

```
#################################################################
#
#  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
```

```
57          syscall                 # we are outta here.
```

Listing 1: column-major.asm

```
1  ###############################################################################
2  #  Row-major order traversal of 16 x 16 array of words.
3  #  Pete Sanderson
4  #  31 March 2007
5  #
6  #  To easily observe the row-oriented order, run the Memory Reference
7  #  Visualization tool with its default settings over this program.
8  #  You may, at the same time or separately, run the Data Cache Simulator
9  #  over this program to observe caching performance.  Compare the results
10 #  with those of the column-major order traversal algorithm.
11 #
12 #  The C/C++/Java-like equivalent of this MIPS program is:
13 #     int size = 16;
14 #     int[size][size] data;
15 #     int value = 0;
16 #     for (int row = 0; col < size; row++) {
17 #        for (int col = 0; col < size; col++) }
18 #           data[row][col] = value;
19 #           value++;
20 #        }
21 #     }
22 #
23 #  Note: Program is hard-wired for 16 x 16 matrix.  If you want to change this,
24 #        three statements need to be changed.
25 #        1. The array storage size declaration at "data:" needs to be changed
       from
26 #           256 (which is 16 * 16) to #columns * #rows.
27 #        2. The "li" to initialize $t0 needs to be changed to new #rows.
28 #        3. The "li" to initialize $t1 needs to be changed to new #columns.
29 #
30           .data
31 data:     .word     0 : 256       # storage for 16x16 matrix of words
32           .text
33           li        $t0, 16       # $t0 = number of rows
34           li        $t1, 16       # $t1 = number of columns
35           move      $s0, $zero    # $s0 = row counter
36           move      $s1, $zero    # $s1 = column counter
37           move      $t2, $zero    # $t2 = the value to be stored
38 #  Each loop iteration will store incremented $t1 value into next element of
       matrix.
39 #  Offset is calculated at each iteration. offset = 4 * (row*#cols+col)
40 #  Note: no attempt is made to optimize runtime performance!
41 loop:     mult      $s0, $t1      # $s2 = row * #cols  (two-instruction sequence)
42           mflo      $s2           # move multiply result from lo register to $s2
43           add       $s2, $s2, $s1 # $s2 += column counter
44           sll       $s2, $s2, 2   # $s2 *= 4 (shift left 2 bits) for byte offset
45           sw        $t2, data($s2) # store the value in matrix element
46           addi      $t2, $t2, 1   # increment value to be stored
47 #  Loop control: If we increment past last column, reset column counter and
       increment row counter
48 #                If we increment past last row, we're finished.
49           addi      $s1, $s1, 1   # increment column counter
50           bne       $s1, $t1, loop # not at end of row so loop back
51           move      $s1, $zero    # reset column counter
52           addi      $s0, $s0, 1   # increment row counter
53           bne       $s0, $t0, loop # not at end of matrix so loop back
54 #  We're finished traversing the matrix.
55           li        $v0, 10       # system service 10 is exit
56           syscall                 # we are outta here.
```

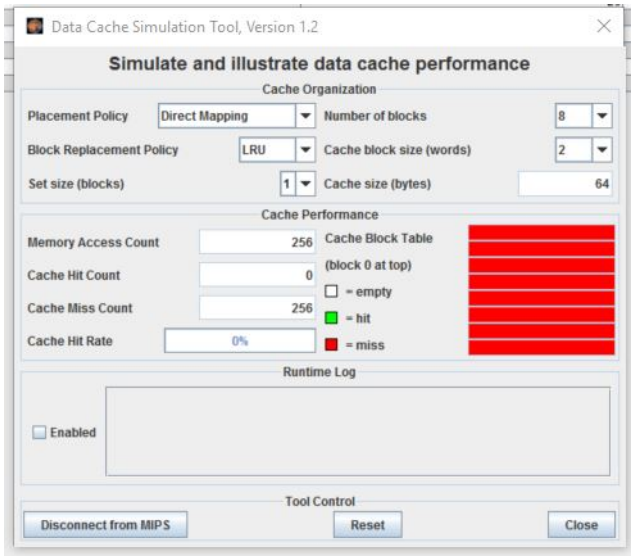


Figure 3: Column Major, Direct Mapping, Cache Block Size 2

Listing 2: row-major.asm

6.3 Cache Results - Snapshots

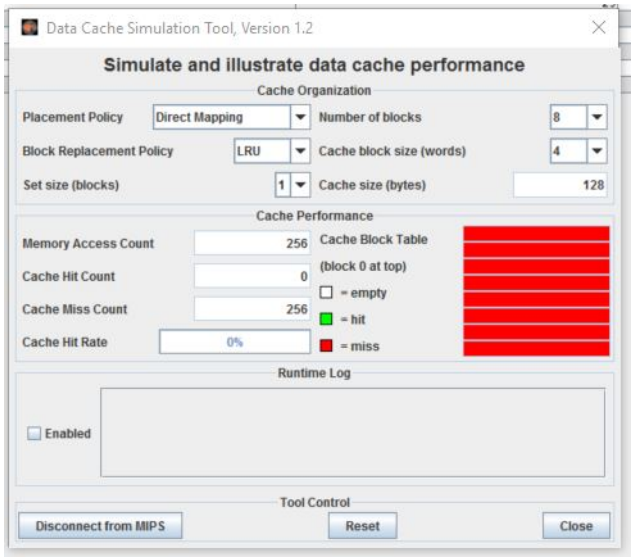


Figure 4: Column Major, Direct Mapping, Cache Block Size 4

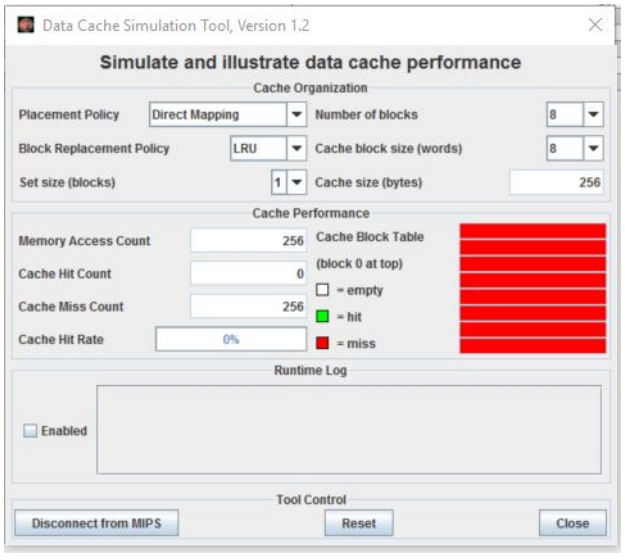


Figure 5: Column Major, Direct Mapping, Cache Block Size 8

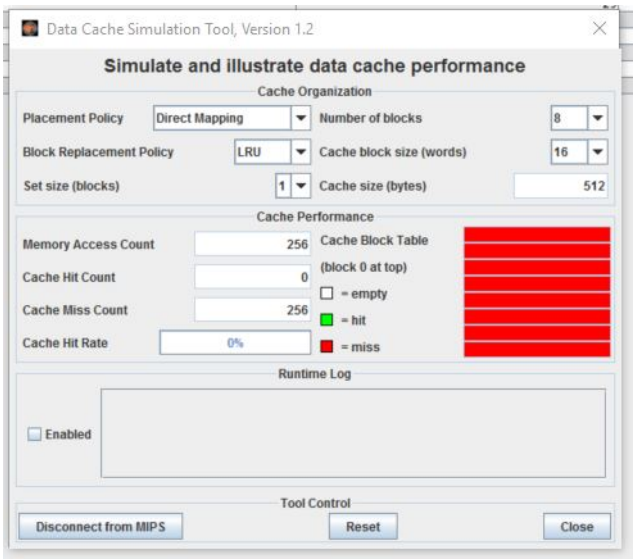


Figure 6: Column Major, Direct Mapping, Cache Block Size 16

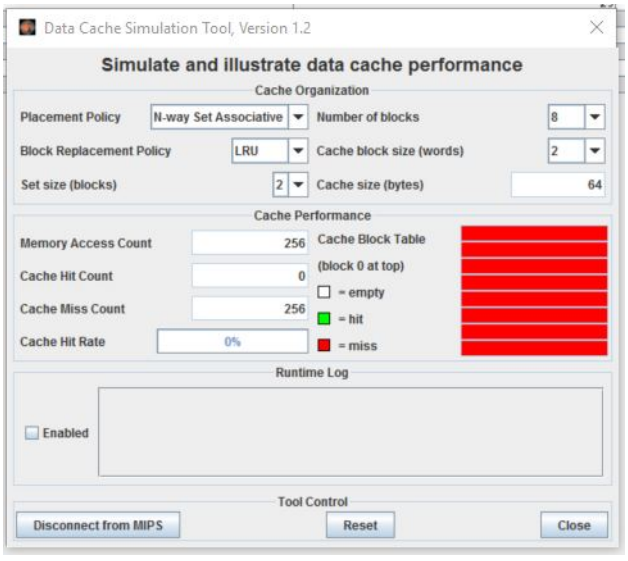


Figure 7: Column Major, 2-Way Associative, Cache Block Size 2

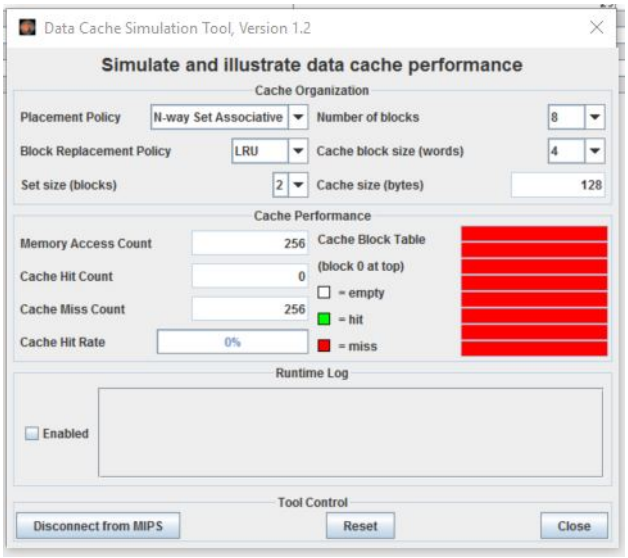


Figure 8: Column Major, 2-Way Associative, Cache Block Size 4

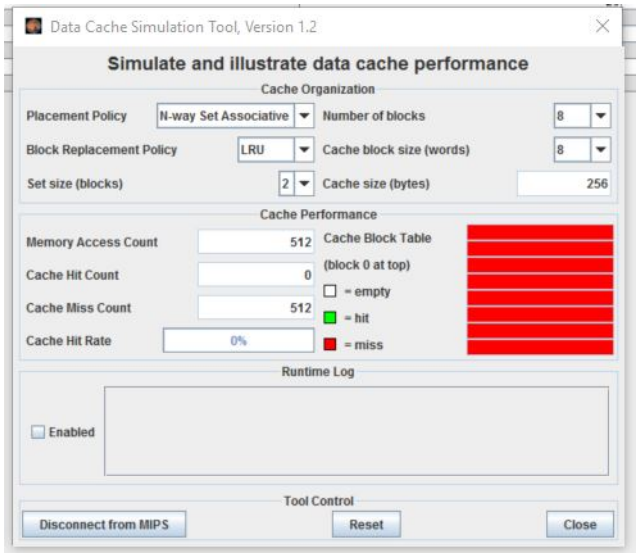


Figure 9: Column Major, 2-Way Associative, Cache Block Size 8

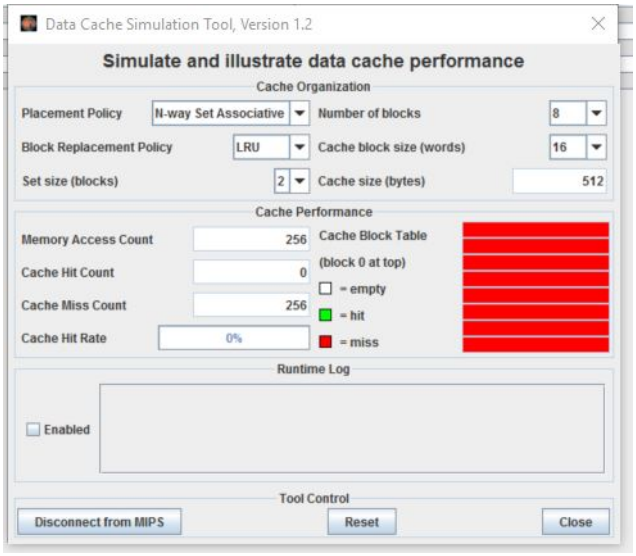


Figure 10: Column Major, 2-Way Associative, Cache Block Size 16

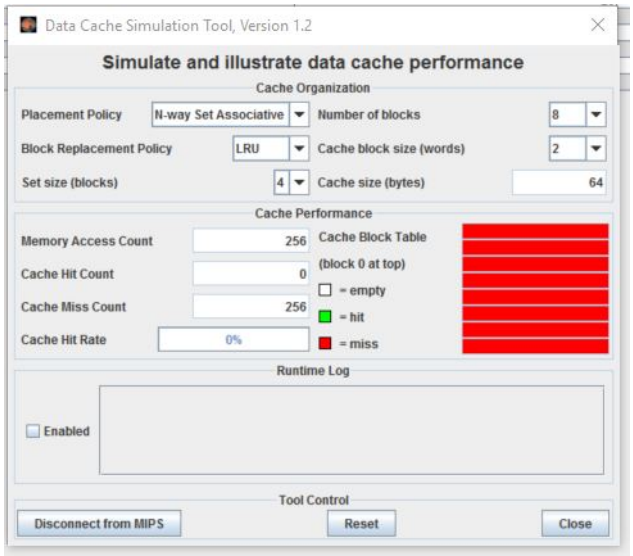


Figure 11: Column Major, 4-Way Associative, Cache Block Size 2

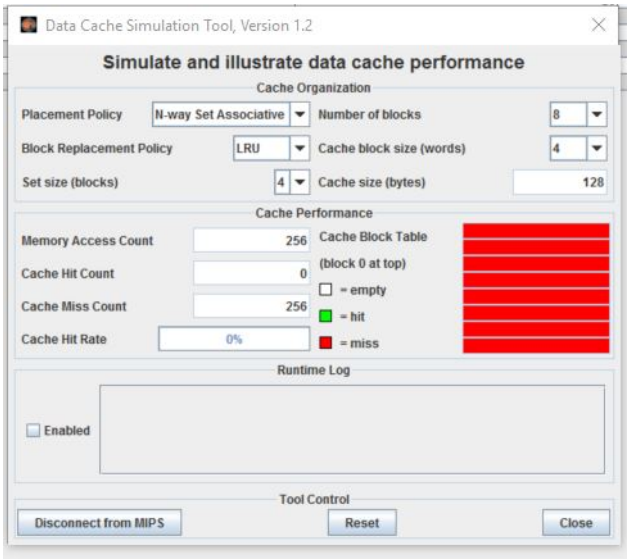


Figure 12: Column Major, 4-Way Associative, Cache Block Size 4

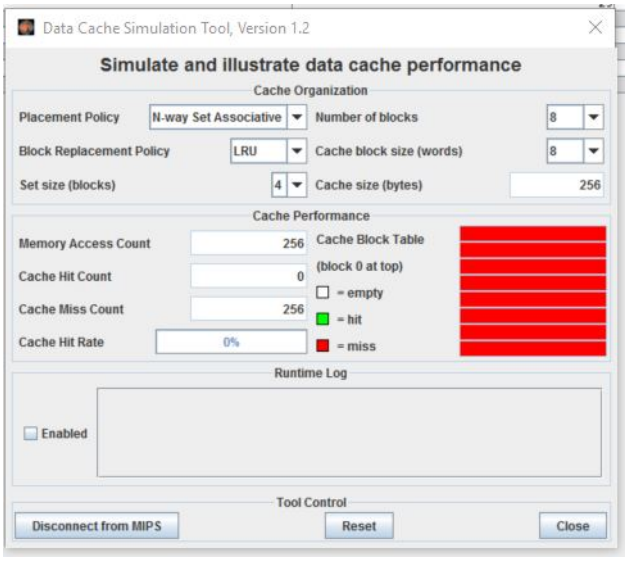


Figure 13: Column Major, 4-Way Associative, Cache Block Size 8

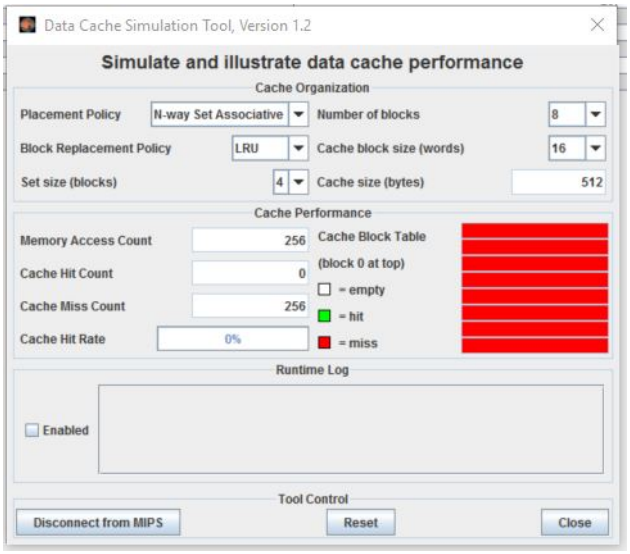


Figure 14: Column Major, 4-Way Associative, Cache Block Size 16

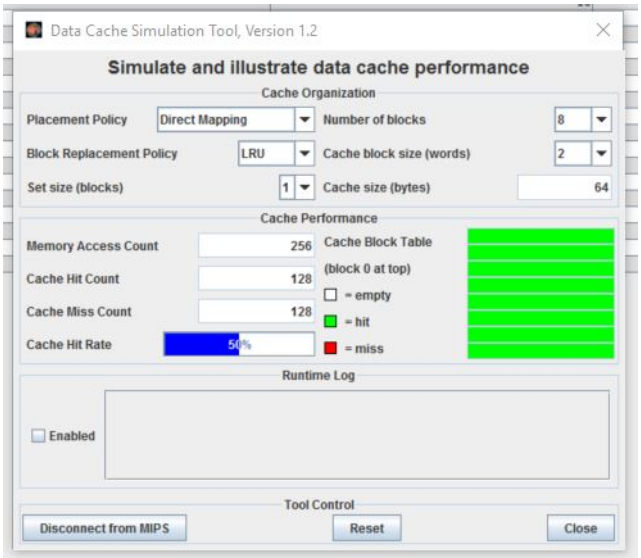


Figure 15: Row Major, Direct Mapping, Cache Block Size 2

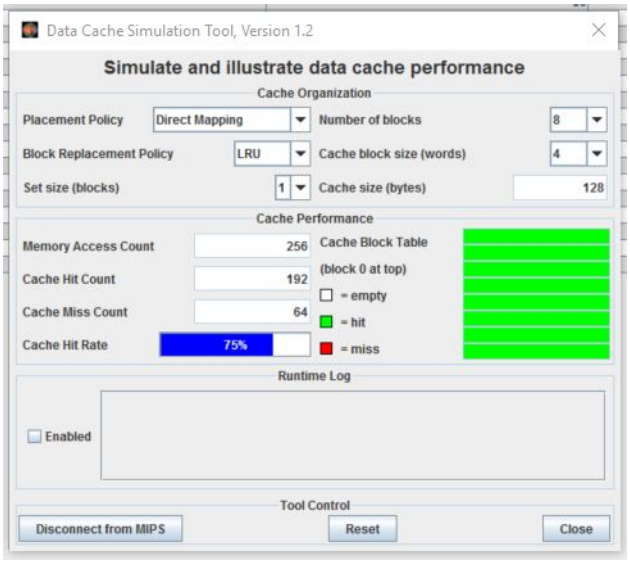


Figure 16: Row Major, Direct Mapping, Cache Block Size 4

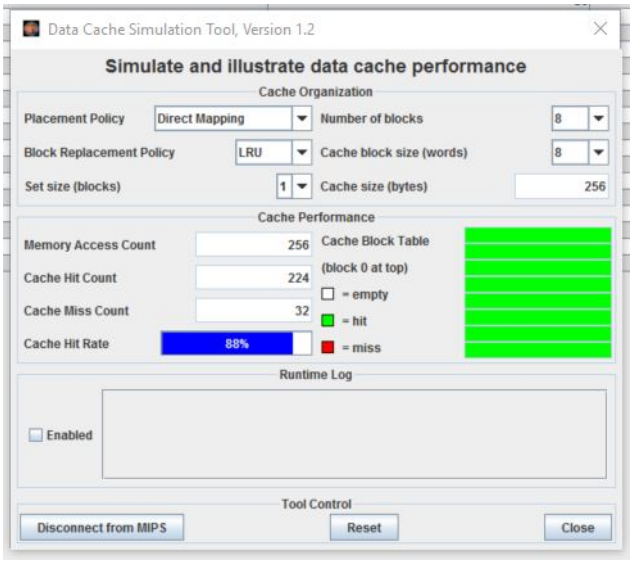


Figure 17: Row Major, Direct Mapping, Cache Block Size 8

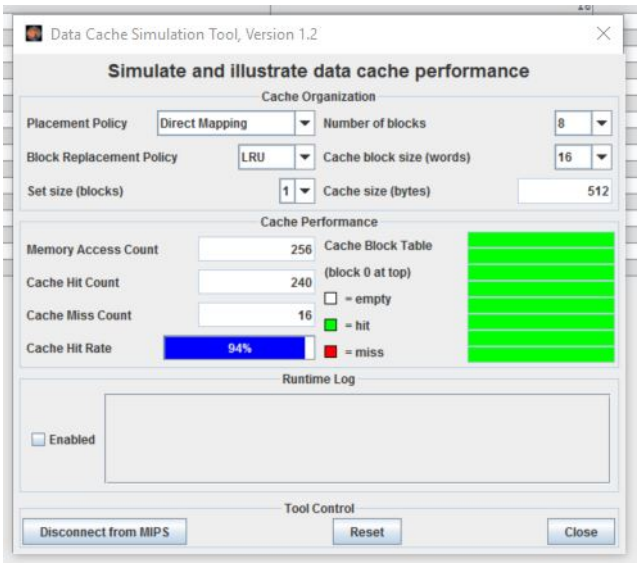


Figure 18: Row Major, Direct Mapping, Cache Block Size 16

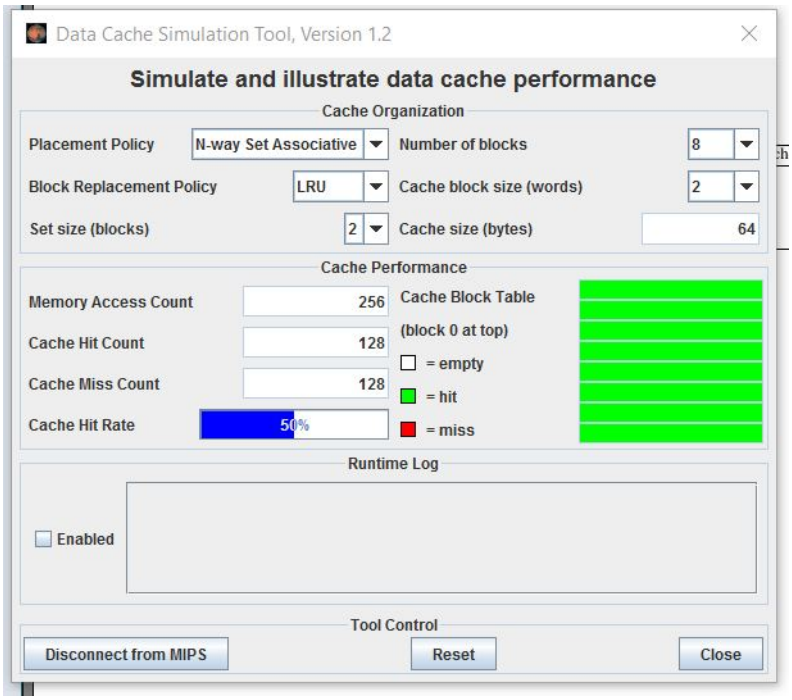


Figure 19: Row Major, 2-Way Associative, Cache Block Size 2

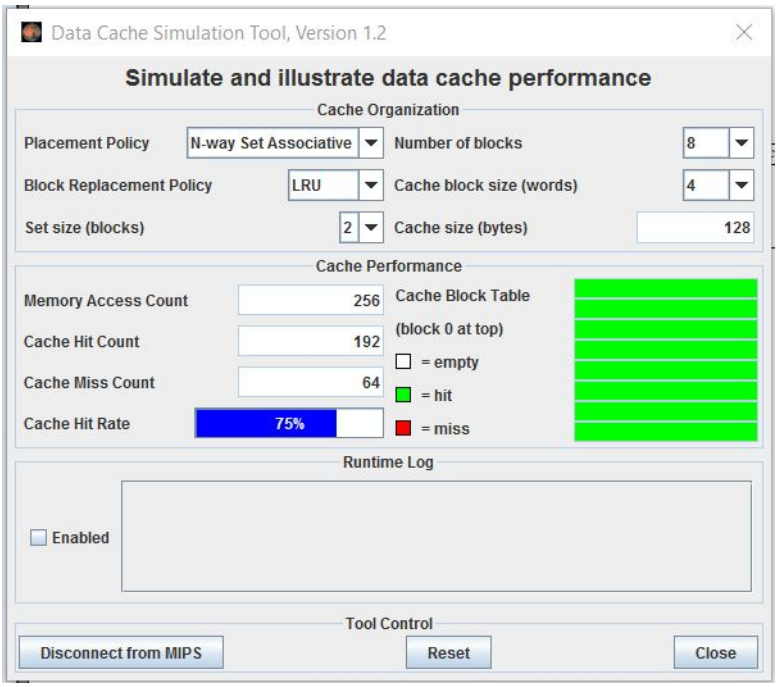


Figure 20: Row Major, 2-Way Associative, Cache Block Size 4

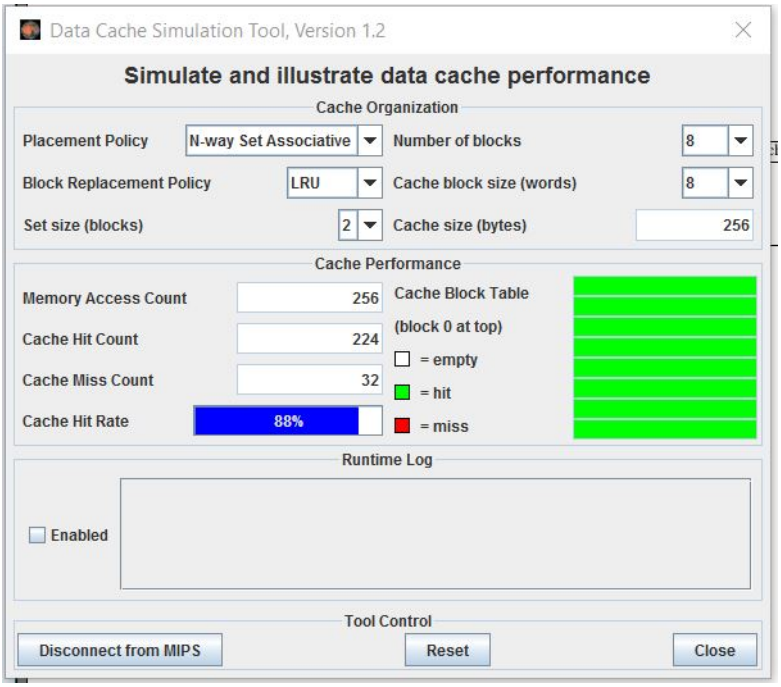


Figure 21: Row Major, 2-Way Associative, Cache Block Size 8

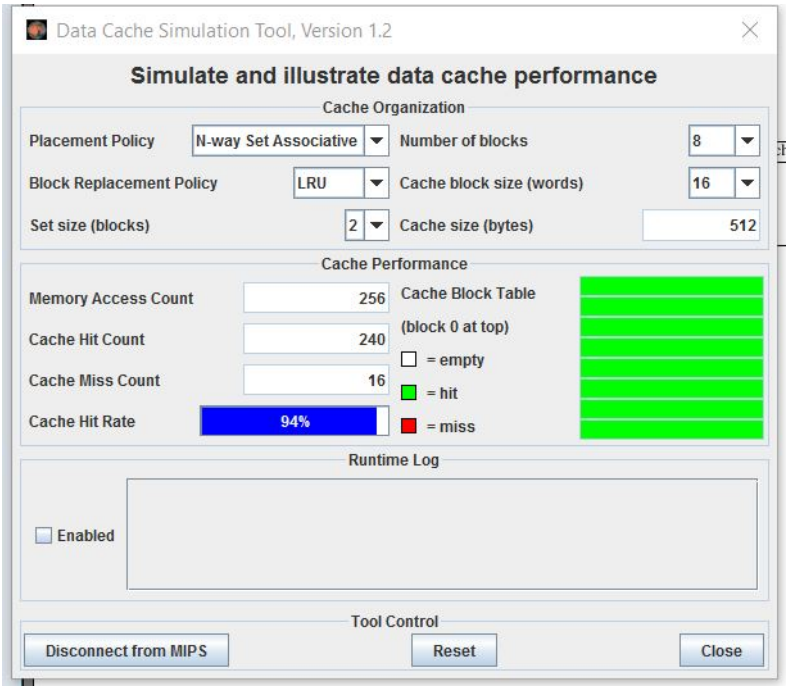


Figure 22: Row Major, 2-Way Associative, Cache Block Size 16

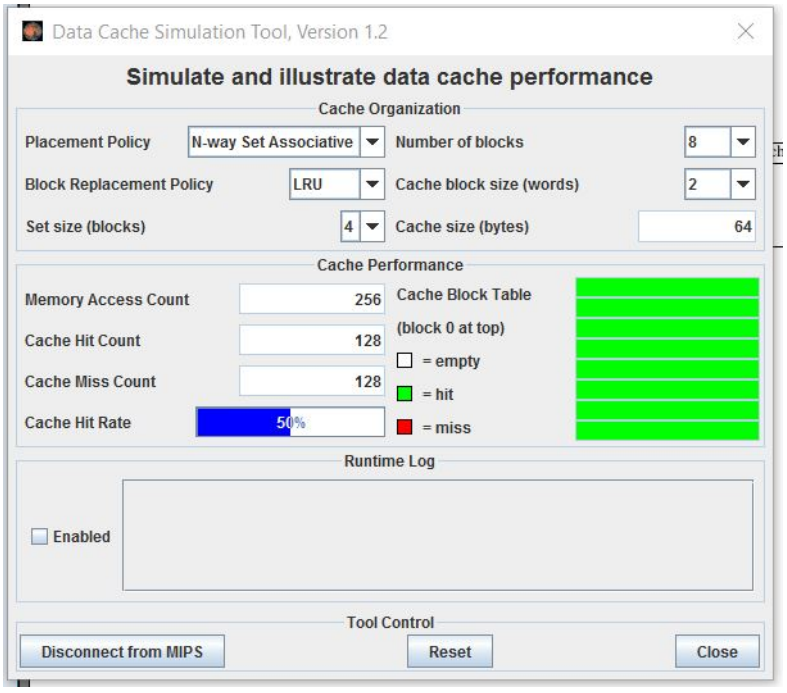


Figure 23: Row Major, 4-Way Associative, Cache Block Size 2

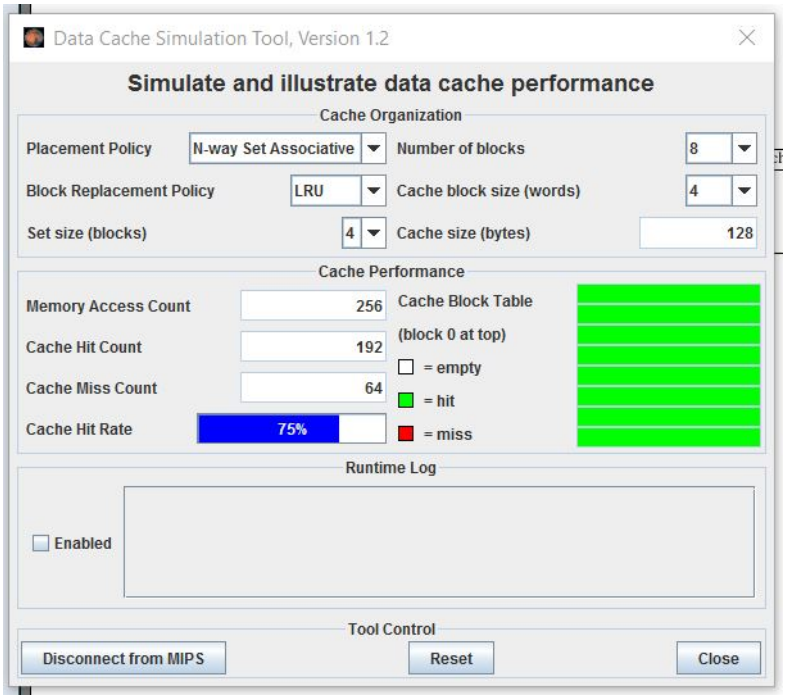


Figure 24: Row Major, 4-Way Associative, Cache Block Size 4

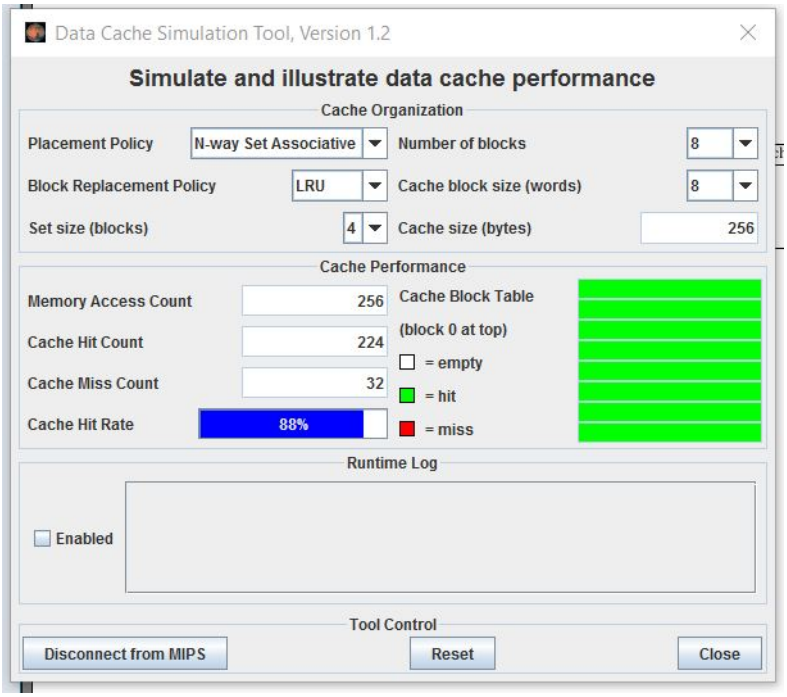


Figure 25: Row Major, 4-Way Associative, Cache Block Size 8

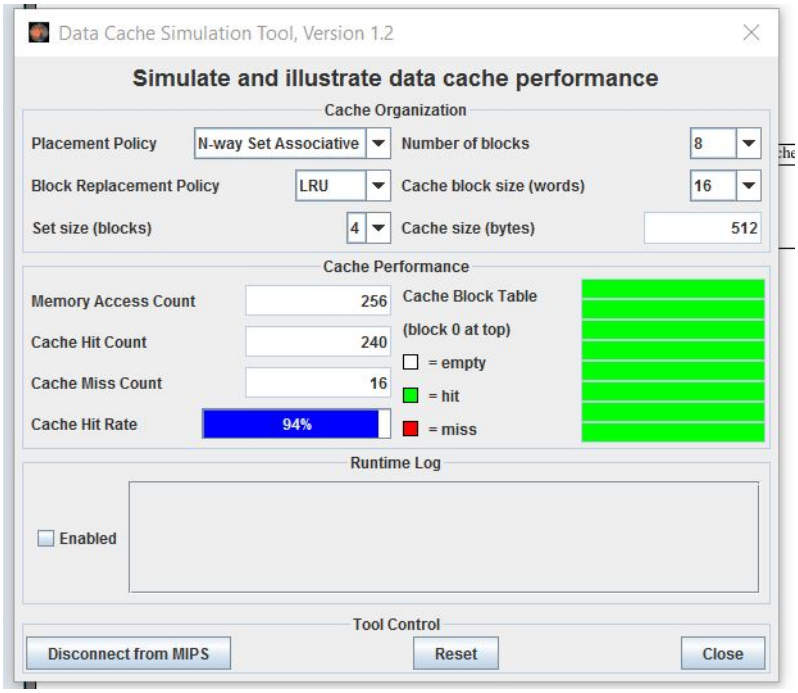


Figure 26: Row Major, 4-Way Associative, Cache Block Size 16