Microprocessor System Design

ECE – 485/585

Fall 2018

Final Project – Cache Simulation

Vinitha Baddam <vbaddam@pdx.edu>

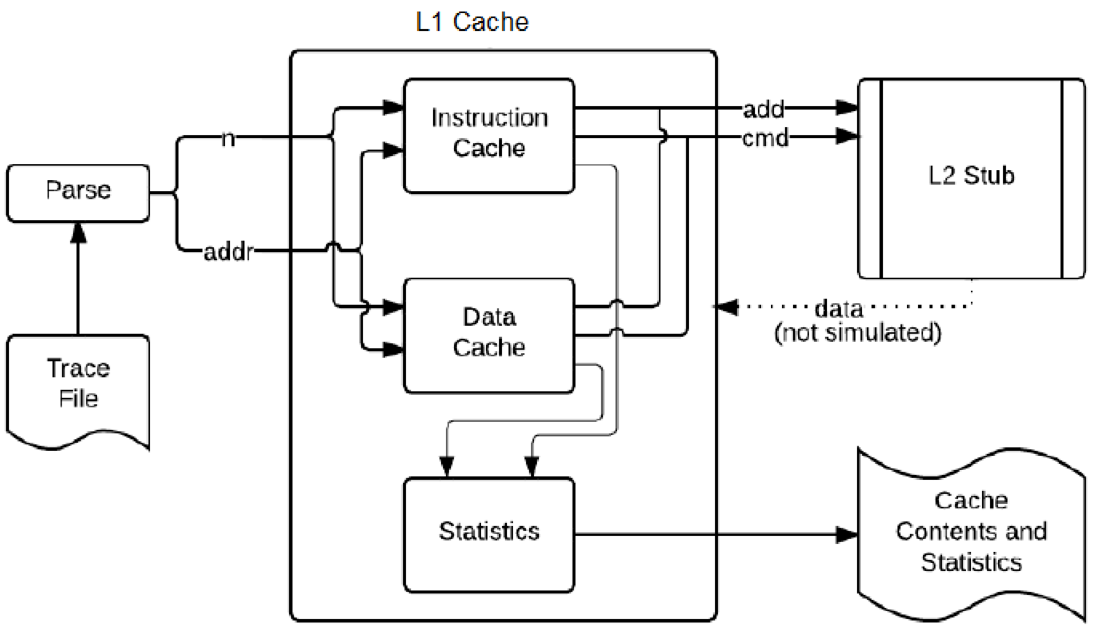
Michael Bourquin <bourquin@pdx.edu>

Hima Ethakota <hima@pdx.edu>

Contents

1. OVERVIEW:

This project requires the simulation of a split L1 cache for a 32-bit processor with having multiple processors. The data cache is 8-way set associative having 16k sets and 64-byte lines and the instruction cache is 4-way set associative having 16k sets and 64-byte lines. The cache employs MESI protocol to maintain cache coherence. The L1 data cache is write-back using write allocate and is write-back except for the first write to a line which is write-through. Both caches employ LRU replacement policy and are backed by a shared L2 cache. The number of reads, writes, hits, misses and the hit rate are calculated and recorded.



2. SPECIFICATIONS

- The 32-bit address is broken into the offset bits, index bits and tag bits. The data cache is 8-way set associative having 16K sets and 64-byte lines and the instruction cache is a 4-way set associative having 16K sets and 64-byte lines.

  31:20 = 12-bit tag

  19:6 = 14-bit index

  5:0 = 6-bit offset
- We give a command to perform various functions in L1 and L2 cache as the project specifies. The output is simulated and the cache contents are displayed in the report/statistics. There is a command for print which will display the contents in the cache that comprise of offset bits, index bits, tag bits, MESI and LRU bits. For the data cache as it is 8-way associative, we have 4 bits for LRU and for the instruction cache being 4-way associative, we have 2 bits for the LRU. MESI bits are represented as 2 bits.
- When printing the contents and state of the cache in response to a 9 in the trace file, we show only the valid lines in the cache along with way and appropriate state and LRU bits.
- Following is the trace file format

  n <address>

  Where n is

  0 - Read data request to L1 data cache

  1 - Write data request to L1 data cache

  2 - Instruction fetch (a read request to L1 instruction cache)

  3 - Invalidate command from L2

  4 - Data request from L2 (in response to snoop)

  8 - Clear the cache and reset all state (and statistics)

  9 - Print contents and state of the cache (allow subsequent trace activity)

  And the address will be a hex value.

  There will be no address for commands (n) 8 and 9
- In order to maintain inclusivity and implement the MESI protocol the L1 caches may have to communicate with the shared L2 cache. To simulate this, you should display the following messages (where <address> is a hexadecimal address).
  - ➢ Return data to L2 <address>

In response to a 4 in the trace file your cache should signal that it's returning the data for that line (if present and modified)

- ➢ Write to L2 <address>

This operation is used to write back a modified line to L2 upon eviction from the L1 cache. It is also used for an initial write through when a cache line is written for the first time so that the L2 knows it's been modified and has the correct data

- ➢ Read from L2 <address>

  This operation is used to obtain the data from L2 on an L1 cache miss

- ➢ Read for Ownership from L2 <address>

  This operation is used to obtain the data from L2 on an L1 cache write miss

- Maintain and report the following key statistics of cache usage for each cache and display them upon completion of execution of each trace:
  - ➢ Number of cache reads
  - ➢ Number of cache writes
  - ➢ Number of cache hits
  - ➢ Number of cache misses
  - ➢ Cache hit ratio

3. ASSUMPTIONS

- In case of designing the L1 cache, the following assumptions were made:
- The cache employs MESI protocol and also follow inclusivity hierarchy to maintain cache coherence. We use 2 bits to represent the MESI bits in the caches.

- LRU replacement policy is used to update the cache after every transaction takes place. The LRU policy we use is the counter method.
- There is no data hence the cache contents would display the LRU, MESI, tag, index and offset bits.
- The data cache is a write through cache in the first transaction and from the next transaction is a write back one.
- All read and write operations are referred to single byte locations.
- A read request from L2 cache can be because of i) Read miss in other processor's cache ii) Write miss in other processor's cache, hence we assume that the processors employ write allocate.
- The MESI bits which are of 2-bit size are encoded as:
    - ➢ Invalid = 2'b00
    - ➢ Exclusive = 2'b01
    - ➢ Shared = 2'b10
    - ➢ Modified = 2'b11
- The LRU replacement policy is done using the counter method and the most recently used bit is encoded as 111 for data cache (11 for instruction cache) and the least recently used bit is encoded as 000 for data cache (00 for instruction cache). The bits higher than the most recently used bits are decremented by 1 and then updated accordingly.
- In our simulation we have considered two mode output.
- MODE=0: In this mode, our simulation displays only the required summary of usage statistics and responses to 9s in the trace file and nothing else
- MODE=1: In this mode, our simulation should display everything from mode 0 but also display the communication messages to the L2 as described in the specification and nothing else.

4. SOURCE CODE:

The Verilog source code for our project is reproduced below:

4.1 tb_CacheSim.v

```
/*
// ECE 485/585: Microprocessor System Design
// Final Project
// Fall 2018
// File : tb_CacheSim.v (Test Bench)
// Authors : Vinitha Baddam, Michael Bourquin and Hima Ethakota
// Description : This module reads a stimulus file
*/

`define AddressBits 32  //Addressbits

//+TRACE=latest.txt +MODE=0

module tb_CacheSim();

    //clock parameters
    parameter CLOCK_CYCLE = 20;
    parameter CLOCK_WIDTH = CLOCK_CYCLE/2;

    parameter TRUE = 1'b1;
    parameter FALSE = 1'b0;
    parameter RESET = 4'd8; // for reset command

    reg clk; //clock
    integer fptr; // the file handle
    reg done;  // trace fle processing status
    reg [3:0] command; // command n from trace file
    reg [`AddressBits-1:0] address; // address from trace file
    reg  [8*100:0] filename; //string trace file name
    reg mode; //output  mode
    integer result;

    CACHE_SIMULATION cache_sim(
            .clk(clk),
            .command(command),
            .address(address),
            .mode(mode),
            .done(done)
            );

    initial
    begin
            clk = FALSE;
            done = FALSE;

            // Check to make sure that a TRACE file was provided
            if($value$plusargs("TRACE=%s", filename) == FALSE)
            begin
                    $display("Please enter a valid trace file name on plusargs");
                    $finish;
```

```
		end

		// If it was, open the file
		fptr = $fopen(filename , "r");

		if($value$plusargs("MODE=%d",  mode) == FALSE)
		begin
			$display("Please enter a valid mode on plusargs!");
			$finish;
		end

		// simulate initial reset
		#CLOCK_WIDTH clk = FALSE;
		command = RESET; //for reset
		address = 32'b0;
		#CLOCK_WIDTH clk = TRUE;

		// While there are lines left to be read :
		while(!$feof(fptr))
		begin
			// Parse the line
			#CLOCK_WIDTH clk = FALSE;
			result = $fscanf(fptr,"%d", command);

			//check if the command is 8 or 9
			if(command != 8 && command != 9)
			begin
				result = $fscanf(fptr,"%h", address);
			end
			#CLOCK_WIDTH clk = TRUE;
		end

		// Close the file, and finish up
		$fclose(fptr);

		#CLOCK_WIDTH clk = FALSE;
		done = TRUE; // set done to true to print statistics
		#CLOCK_WIDTH clk = TRUE;
		$stop;
	end
endmodule
```

4.2 CacheSimulation.v

```
/*
// ECE 485/585: Microprocessor System Design
// Final Project
// Fall 2018
// File : CacheSimulation.v
// Authors : Vinitha Baddam, Michael Bourquin and Hima Ethakota
// Description : This module makes a call to all other modules
*/

`define AddressBits 32  //Addressbits

module CACHE_SIMULATION(
    input clk,
    input [3:0] command,
    //input address,
    input [`AddressBits-1:0] address,
    input mode,
    input done
    );

    //valid commands from trace file
    parameter READ = 4'd0;
    parameter WRITE = 4'd1;
    parameter INSTRUCTION_FETCH = 4'd2;
    parameter INVALIDATE = 4'd3;
    parameter SNOOP = 4'd4;
    parameter RESET = 4'd8;
    parameter PRINT = 4'd9;

    //signals for statistics
    wire [31:0]
            d_read_hit,
            d_read_miss,
            d_reads,
            d_write_hit,
            d_write_miss,
            d_writes,
            i_hit,
            i_miss,
            i_reads;

    //To call data cache
    DATA_CACHE d_cache(
            .clk(clk),
            .command(command),
            .address(address),
            .mode(mode),
            .DC_Read_Hit(d_read_hit),
            .DC_Read_Miss(d_read_miss),
            .DC_Reads(d_reads),
            .DC_Write_Hit(d_write_hit),
            .DC_Write_Miss(d_write_miss),
            .DC_Writes(d_writes)
            );
    //To call instruction cache
    INSTRUCTION_CACHE i_cache(
            .clk(clk),
            .command(command),
            .address(address),
            .mode(mode),
            .IC_Read_Hit(i_hit),
            .IC_Read_Miss(i_miss),
            .IC_Reads(i_reads)
            );

    //To print statistics on done status
    STATISTICS stats(
            .done(done),
            .DC_Read_Hit(d_read_hit),
            .DC_Read_Miss(d_read_miss),
            .DC_Reads(d_reads),
            .DC_Write_Hit(d_write_hit),
            .DC_Write_Miss(d_write_miss),
            .DC_Writes(d_writes),
            .IC_Read_Hit(i_hit),
            .IC_Read_Miss(i_miss),
            .IC_Reads(i_reads)
            );

endmodule
```

4.3 InstructionCache.v

```
/*
// ECE 485/585: Microprocessor System Design
// Final Project
// Fall 2018
// File : InstructionCache.v
// Authors : Vinitha Baddam, Michael Bourquin and Hima Ethakota
// Description : This module is for all the instruction cache operations
*/

`define InstructionCacheWay 4  //InstructionCacheWay
`define InstructionCacheSet 16*1024  //InstructionCacheset
`define CacheLineSize 64  //linelength
`define AddressBits 32  //AddressBits

module INSTRUCTION_CACHE(
    input clk,
    input [3:0] command,
    input [`AddressBits-1:0] address,
    input mode,
    output reg [31:0] IC_Read_Hit = 32'b0,
    output reg [31:0] IC_Read_Miss = 32'b0,
    output reg [31:0] IC_Reads = 32'b0
    );

    //MESI protocal
    parameter
            Invalid = 2'b00,
            Exclusive = 2'b01,
            Shared = 2'b10,
            Modified = 2'b11;

    //valid commands from trace file
    parameter
            //READ = 4'd0,
            //WRITE = 4'd1,
            INSTRUCTION_FETCH = 4'd2,
            INVALIDATE = 4'd3,
            //SNOOP = 4'd4,
            RESET = 4'd8,
            PRINT = 4'd9;

    //Instruction cache index, offset, tag and LRU bits
    parameter
            IC_IndexBits = $clog2(`InstructionCacheSet),
            IC_OffsetBits = $clog2(`CacheLineSize),
            IC_TagBits = `AddressBits -(IC_OffsetBits+IC_IndexBits),
            IC_LRUBits = $clog2(`InstructionCacheWay),
            MESI_Size = 2;

    //InstructionCache
    reg [IC_TagBits + IC_LRUBits + MESI_Size-1:0] InstructionCache[0:`InstructionCacheWay-1]
    [0:`InstructionCacheSet-1];
    //Cache Instruction Line
    reg [IC_TagBits + IC_LRUBits + MESI_Size-1:0] InstructionLine;

    reg [1:0] MESI_Bits;
    reg [IC_OffsetBits-1:0] Offset;
    reg [IC_IndexBits-1:0] Index;
    reg [IC_TagBits-1:0] Tag;

    //for instruction cache
    reg [IC_OffsetBits-1:0] IC_Offset;
    reg [IC_IndexBits-1:0] IC_Index;
    reg [IC_TagBits-1:0] IC_Tag;
    reg [IC_LRUBits-1:0] IC_LRU;

    //These are used to print the cache content
    reg [IC_TagBits-1:0] print_Tag [0:`InstructionCacheWay];
    reg [IC_LRUBits-1:0] print_LRU [0:`InstructionCacheWay];
    reg [1:0] print_MESI [0:`InstructionCacheWay];
```

```
    parameter TRUE = 1'b1;
    parameter FALSE = 1'b0;

    integer way, found, replaced, w, s, temp;
    reg print;

    //Execute below on clock positive edge
    always @(posedge clk)
    begin
            //Parsing address to get offset, index and tag bits
            Offset = address[IC_OffsetBits-1:0];
            Index = address[IC_OffsetBits+IC_IndexBits-1:IC_OffsetBits];
            Tag = address[`AddressBits-1:`AddressBits-IC_TagBits];

            //Initially found and replaced values will be false
            found=0;
            replaced=0;

            case(command)
            //2 instruction fetch (a read request to L1 instruction cache)
            INSTRUCTION_FETCH:
            begin
                    IC_Reads=IC_Reads+1;  //reads counter
                    //see if the tag of the fetch address matches any tag in those set lines
                    for(way=0;way<`InstructionCacheWay;way=way+1)
                    begin
                            InstructionLine = InstructionCache[way][Index];
            MESI_Bits = InstructionLine[IC_TagBits + IC_LRUBits + MESI_Size-1: IC_TagBits+
            IC_LRUBits];
                            IC_Tag = InstructionLine[IC_TagBits-1:0];

                            //if valid and tag match
            if((MESI_Bits != Invalid) && (IC_Tag == Tag))
            begin
                                    IC_Read_Hit=IC_Read_Hit+1;  //read hit counter
                                    found=1;          //data in cache found!
                                    IC_LRU = InstructionLine[IC_TagBits+IC_LRUBits-1:
IC_TagBits];
                InstructionLine[IC_TagBits+IC_LRUBits+MESI_Size-1:IC_TagBits +IC_LRUBits]
                = Shared;
                                    InstructionCache[way][Index] = InstructionLine;

                            end
                    end

                    // If tag is not found in the cache set from above
                    if(found==0)
                    begin
                            IC_Read_Miss=IC_Read_Miss+1;  //write miss counter
                            if(mode == 1)
                    $display("Read from L2 %h", address); //add lru and write

        //find an invalid way in set and put tag bits and set lru 111 and decreament lru for other
        ways by 1
                for(way=0;way<`InstructionCacheWay;way=way+1)
                begin
                    InstructionLine=InstructionCache[way][Index];
         MESI_Bits=InstructionLine[IC_TagBits+IC_LRUBits+ MESI_Size-1:IC_TagBits+
         IC_LRUBits];

                    if ((MESI_Bits == Invalid) && (found!=1))
                    begin
                        found=1; // invalid way found
          IC_LRU=InstructionLine[IC_TagBits+IC_LRUBits-1:IC_TagBits];

                        InstructionLine[IC_TagBits-1:0] = Tag;
          InstructionLine[IC_TagBits+IC_LRUBits+MESI_Size-1:IC_TagBits+
          IC_LRUBits] = Exclusive;
          InstructionCache[way][Index] = InstructionLine;
                    end
                end

                // if no invalid way, look for LRU=000 and check mesi bits
                if(found!=1)
                begin
                    for(way=0;way<`InstructionCacheWay;way=way+1)
                    begin
                        InstructionLine=InstructionCache[way][Index];
          MESI_Bits =InstructionLine[IC_TagBits+IC_LRUBits+ MESI_Size-1:IC_TagBits
          + IC_LRUBits];
          IC_LRU=InstructionLine[IC_TagBits+IC_LRUBits-1: IC_TagBits];

                        if (IC_LRU == 0 && replaced == 0)
                        begin
                            replaced = 1;
           // if mesi bit modified write to L2 and replace,ie, put tag bits there
                            if(MESI_Bits == Modified)
                            begin
                                Offset = 0;
                                if(mode == 1)
                $display("Write to L2 %h", {InstructionLine[IC_TagBits-1:0], Index,
                Offset});
                            end


IC_LRU=InstructionLine[IC_TagBits+IC_LRUBits-1:IC_TagBits];
                            InstructionLine[IC_TagBits-1:0] = Tag;
           InstructionLine[IC_TagBits+IC_LRUBits+MESI_Size-1:IC_TagBits+
           IC_LRUBits] = Exclusive;
                            InstructionCache[way][Index] = InstructionLine;
                        end
                        end
                        IC_LRU = 0;  //To replace '0'th LRU element
                    end
                end

                temp = UpdateInstruction_LRU(IC_LRU, Index);          // update lru
            end

            //3 invalidate command from L2
            INVALIDATE:
            begin
                // look up for line by going to set n comparing tags in ways
                for(way=0;way<`InstructionCacheWay;way=way+1)
                begin
                    InstructionLine = InstructionCache[way][Index];
            MESI_Bits = InstructionLine[IC_TagBits+IC_LRUBits+MESI_Size-1:IC_TagBits+
            IC_LRUBits];
                    IC_Tag = InstructionLine[IC_TagBits-1:0];

            if(Tag == IC_Tag) //if tag match
                    begin
                        MESI_Bits = Invalid; //if line found then set MESI_Bits as
invalid
                        IC_LRU = InstructionLine[IC_TagBits+IC_LRUBits-
1:IC_TagBits];
             InstructionLine[IC_TagBits+IC_LRUBits+MESI_Size-1:IC_TagBits+IC_LRUBits] =
             MESI_Bits;
                        InstructionLine[IC_TagBits-1:0] = 12'bx;
                        InstructionCache[way][Index] = InstructionLine;
                    end
                end

                temp = UpdateInstruction_LRU(IC_LRU, Index);          // update lru
            end

            //8 clear the cache and reset all state (and statistics)
            RESET:
            begin
                for (s=0; s<`InstructionCacheSet; s=s+1)
                begin
                    for (w=0; w<`InstructionCacheWay; w=w+1)
                    begin
                        InstructionLine=InstructionCache[w][s];
                        //set all the cache MESI bits to Invalid
             InstructionLine[IC_TagBits + IC_LRUBits + MESI_Size-1 : IC_TagBits +
             IC_LRUBits] = Invalid;
                        //Initially assign LRU bits to each cache line in a set to line
number
                        InstructionLine[IC_TagBits + IC_LRUBits-1 : IC_TagBits] = w;
                        //set tag bits to x
                        InstructionLine[IC_TagBits-1:0] = 12'bx;
```

```
                              InstructionCache[w][s] = InstructionLine;
                         end
                    end

                    //set all summary paramentes to '0' on reset
                    IC_Read_Hit = 32'b0;
                    IC_Read_Miss = 32'b0;
                    IC_Reads = 32'b0;
               end

               //9 print contents and state of the cache (allow subsequent trace activity)
               PRINT:
               begin
                    $display("_____________________________________");
                    $display("                                  ");
                    $display("    INSTRUCTION CACHE CONTENTS      ");
                    $display("_____________________________________");

                    for (s=0;s<`InstructionCacheSet;s=s+1)
                    begin
                         print = FALSE;
                         for (w=0;w<`InstructionCacheWay;w=w+1)
                         begin
                              InstructionLine = InstructionCache[w][s];
                 MESI_Bits = InstructionLine[IC_TagBits+IC_LRUBits+MESI_Size-1:IC_TagBits+
                 IC_LRUBits];
                              IC_LRU = InstructionLine[IC_TagBits+IC_LRUBits-
       1:IC_TagBits];
                              IC_Tag = InstructionLine[IC_TagBits-1:0];

                              if(print == TRUE || MESI_Bits != Invalid || IC_Tag != "x")
                              begin
                                   if(print == FALSE)
                                   begin
                                        print = TRUE;
                                        w = -1;
                                   end
                                   else
                                   begin
                                        print_MESI[w] = MESI_Bits;
                                        print_LRU[w] = IC_LRU;
                                        print_Tag[w] = IC_Tag;
                                   end
                              end
                         end
                         if(print == TRUE)
                         begin
                              $display("Set Index : %h", s);
                              $display("Way: 1          2          3          4");
                 $display("Tag:       %h       %h       %h       %h", print_Tag[0], print_Tag[1],
                 print_Tag[2], print_Tag[3]);
```

```
                $display("LRU:      %b      %b      %b      %b", print_LRU[0], print_LRU[1],
                print_LRU[2], print_LRU[3]);
                $display("MESI:     %s      %s      %s      %s", Get_MESI_ID(print_MESI[0]),
                Get_MESI_ID(print_MESI[1]), Get_MESI_ID(print_MESI[2]),
                Get_MESI_ID(print_MESI[3]));
                                $display("       ** END OF SET **         ");
                                $display("-----------------------------------");
                        end
                end
                $display("  END OF INSTRUCTION CACHE CONTENTS ");
                $display("___________________________________");
        end

        endcase
    end

    //function to update LRU values of each line in a set based on line reference
    function UpdateInstruction_LRU;
        input [IC_LRUBits-1:0] LRU;
        input [IC_IndexBits-1:0] index;

        integer w;

        begin
            //update the LRU bits of each cache line in a set
            for (w=0; w<`InstructionCacheWay; w=w+1)
            begin
                InstructionLine = InstructionCache[w][index];
                IC_LRU = InstructionLine[IC_TagBits+IC_LRUBits-1:IC_TagBits];
                //if it is a most recently used/refered cache line then set the LRU bits to
111
                if(IC_LRU == LRU)
                begin
                    InstructionLine[IC_TagBits+IC_LRUBits-1:IC_TagBits] =
`InstructionCacheWay-1;
                    InstructionCache[w][index] = InstructionLine;
                end
                //LRU bits higher than the most recently used bits are decremented by 1
                else if(IC_LRU > LRU)
                begin
                    InstructionLine[IC_TagBits+IC_LRUBits-1:IC_TagBits] =
IC_LRU-1;
                    InstructionCache[w][index] = InstructionLine;
                end
            end
        end
    endfunction

    //function to get MESI ids for a given 2 bit MESI value
    function [8:0] Get_MESI_ID;
        input [1:0] MESI;
```

```
		begin
			case(MESI)
			2'b00: Get_MESI_ID = "I";
			2'b01: Get_MESI_ID = "E";
			2'b10: Get_MESI_ID = "S";
			2'b11: Get_MESI_ID = "M";
			endcase
		end
	endfunction

endmodule
```

4.4 DataCache.v

```
/*
// ECE 485/585: Microprocessor System Design
// Final Project
// Fall 2018
// File : DataCache.v
// Authors : Vinitha Baddam, Michael Bourquin and Hima Ethakota
// Description : This module is for all the data cache operations
*/

`define DataCacheWay 8  //DataCacheway
`define DataCacheSet 16*1024  //DataCacheset
`define CacheLineSize 64  //linelength
`define AddressBits 32  //AddressBits

module DATA_CACHE(
	input clk,
	input [3:0] command,
	input [`AddressBits-1:0] address,
	input mode,
	output reg [31:0] DC_Read_Hit = 32'b0,
	output reg [31:0] DC_Read_Miss = 32'b0,
	output reg [31:0] DC_Reads = 32'b0,
	output reg [31:0] DC_Write_Hit = 32'b0,
	output reg [31:0] DC_Write_Miss = 32'b0,
	output reg [31:0] DC_Writes = 32'b0
	);

	//MESI protocal
	parameter
		Invalid = 2'b00,
		Exclusive = 2'b01,
		Shared = 2'b10,
            Modified = 2'b11;

  //valid commands from trace file
  parameter
            READ = 4'd0,
            WRITE = 4'd1,
            //INSTRUCTION_FETCH = 4'd2,
            INVALIDATE = 4'd3,
            SNOOP = 4'd4,
            RESET = 4'd8,
            PRINT = 4'd9;

  //Data cache index, offset, tag and LRU bits
  parameter
            DC_IndexBits = $clog2(`DataCacheSet),
            DC_OffsetBits = $clog2(`CacheLineSize),
            DC_TagBits = `AddressBits -(DC_OffsetBits+DC_IndexBits),
            DC_LRUBits = $clog2(`DataCacheWay),
            MESI_Size = 2;

//Data Cache
reg [DC_TagBits + DC_LRUBits + MESI_Size-1:0] DataCache[0:`DataCacheWay-1]
[0:`DataCacheSet-1];
//Cache Data Line
  reg [DC_TagBits + DC_LRUBits + MESI_Size-1:0] DataLine;

  reg [1:0] MESI_Bits;
  reg [DC_OffsetBits-1:0] Offset;
  reg [DC_IndexBits-1:0] Index;
  reg [DC_TagBits-1:0] Tag;

  //for data cache
  reg [DC_OffsetBits-1:0] DC_Offset;
  reg [DC_IndexBits-1:0] DC_Index;
  reg [DC_TagBits-1:0] DC_Tag;
  reg [DC_LRUBits-1:0] DC_LRU;

  //These are used to print the cache content
  reg [DC_TagBits-1:0] print_Tag [0:`DataCacheWay];
  reg [DC_LRUBits-1:0] print_LRU [0:`DataCacheWay];
  reg [1:0] print_MESI [0:`DataCacheWay];

  parameter TRUE = 1'b1;
  parameter FALSE = 1'b0;

  integer way, found, replaced, w, s, temp;
  reg print;

  //Execute below on clock positive edge
  always @(posedge clk)
  begin
```

```
            //Parsing address to get offset, index and tag bits
            Offset = address[DC_OffsetBits-1:0];
            Index = address[DC_OffsetBits+DC_IndexBits-1:DC_OffsetBits];
            Tag = address[`AddressBits-1:`AddressBits-DC_TagBits];

            //Initially found and replaced values will be false
            found=0;
            replaced=0;

            case(command)
            //0 read data request to L1 data cache
            READ:
            begin
                DC_Reads=DC_Reads+1; //reads counter

                //see if the tag of the read address matches any tag in those set lines
                for(way=0;way<`DataCacheWay;way=way+1)
                begin
                    DataLine = DataCache[way][Index];
            MESI_Bits = DataLine[DC_TagBits+DC_LRUBits+MESI_Size-1:DC_TagBits+
            DC_LRUBits];
                    DC_Tag = DataLine[DC_TagBits-1:0];

            //if valid and tag match
                    if((MESI_Bits != Invalid) && (DC_Tag == Tag))
                    begin
                        DC_Read_Hit=DC_Read_Hit+1;  //read hit counter
                        DC_LRU=DataLine[DC_TagBits+DC_LRUBits-
      1:DC_TagBits];
                        found=1;        //data in cache found!

                        if (MESI_Bits == Exclusive)
                        begin
                            MESI_Bits = Shared;  //change the MESI bits to Shared
      if it was Exclusive
                        end

              DataLine[DC_TagBits+DC_LRUBits+MESI_Size-1:DC_TagBits+DC_LRUBits] =
              MESI_Bits;
                        DataCache[way][Index] = DataLine;
                    end
                end

                // If tag is not found in the cache set from above
                if(found==0)
                begin
                    DC_Read_Miss=DC_Read_Miss+1;  //read miss counter

                    if(mode == 1)
                        $display("Read from L2 %h", address); //add lru and write
            //find an invalid way in set and put tag bits and set lru 111 and decreament lru for other
            ways by 1
                        for(way=0;way<`DataCacheWay;way=way+1)
                        begin
                              DataLine = DataCache[way][Index];
              MESI_Bits = DataLine[DC_TagBits+DC_LRUBits+MESI_Size-1:DC_TagBits+
              DC_LRUBits];

                              if ((MESI_Bits == Invalid) && (found!=1))
                              begin
                                    found=1; // invalid way found
                                    DC_LRU = DataLine[DC_TagBits+DC_LRUBits-
1:DC_TagBits];
                                    DataLine[DC_TagBits-1:0] = Tag;
                DataLine[DC_TagBits+DC_LRUBits+MESI_Size-1:DC_TagBits+DC_LRUBits]
                = Exclusive;
                                    DataCache[way][Index] = DataLine;
                              end
                        end
                        // if no invalid way, look for LRU=000 and check mesi bits
                        if(found!=1)
                        begin
                              for(way=0;way<`DataCacheWay;way=way+1)
                              begin
                                    DataLine = DataCache[way][Index];
                MESI_Bits = DataLine[DC_TagBits+DC_LRUBits+MESI_Size-1:
                DC_TagBits+DC_LRUBits];
                                    DC_LRU = DataLine[DC_TagBits+DC_LRUBits-
1:DC_TagBits];

                                    if (DC_LRU == 0 && replaced == 0)
                                    begin
                                          replaced = 1;
                                          // if mesi bit modified write to L2 and replace,ie,
put tag bits there
                                          if(MESI_Bits == Modified)
                                          begin
                                                Offset = 0;
                                                if(mode == 1)
                        $display("Write to L2 %h", {DataLine[DC_TagBits-1:0], Index,
                        Offset});
                                          end

                                          DC_LRU =
DataLine[DC_TagBits+DC_LRUBits-1:DC_TagBits];
                                          DataLine[DC_TagBits-1:0] = Tag;
                  DataLine[DC_TagBits+DC_LRUBits+MESI_Size-1:DC_TagBits+
                  DC_LRUBits] = Exclusive;
                                          DataCache[way][Index] = DataLine;
                                    end
                              end
```

```
                    DC_LRU = 0; //To replace '0'th LRU element
                end
            end

            temp = UpdateData_LRU(DC_LRU, Index);
        end

        //1 write data request to L1 data cache
        WRITE:
        begin
            DC_Writes=DC_Writes+1;  //writes counter
            //see if the tag of the write address matches any tag in those set lines
            for(way=0;way<`DataCacheWay;way=way+1)
            begin
                DataLine = DataCache[way][Index];
        MESI_Bits = DataLine[DC_TagBits+DC_LRUBits+MESI_Size-1:DC_TagBits+
        DC_LRUBits];
                DC_Tag = DataLine[DC_TagBits-1:0];

                //if valid and tag match
                if((MESI_Bits != Invalid) && (DC_Tag == Tag))
                begin
                    DC_Write_Hit=DC_Write_Hit+1;  //write hit counter
                    DC_LRU = DataLine[DC_TagBits+DC_LRUBits-
1:DC_TagBits];
                    found=1;        // cache line to be written found!

                    if(MESI_Bits == Shared)
                    begin
                        if(mode == 1)
                            $display("Write        to L2 %h", address);
                        MESI_Bits = Exclusive;
                    end
                    else if (MESI_Bits == Exclusive)
                    begin
                        MESI_Bits = Modified;
                    end
                    else if (MESI_Bits == Modified)
                    begin
                        MESI_Bits = Modified;
                    end
         DataLine[DC_TagBits+DC_LRUBits+MESI_Size-1:DC_TagBits+DC_LRUBits] =
         MESI_Bits;
                    DataCache[way][Index] = DataLine;
                end
            end
            // If tag is not found in the cache set from above
            if(found==0)
            begin
                DC_Write_Miss=DC_Write_Miss+1;  //write miss counter
```

```
                        if(mode == 1)
                                $display("Read for Ownership from L2 %h", address); //add lru
and write

            //find an invalid way in set and put tag bits and set lru 111 and decreament lru for other
            ways by 1
                        for(way=0;way<`DataCacheWay;way=way+1)
                        begin
                                DataLine=DataCache[way][Index];
                MESI_Bits=DataLine[DC_TagBits+DC_LRUBits+MESI_Size-1:DC_TagBits+
                DC_LRUBits];

                                if((MESI_Bits == Invalid) && (found!=1))
                                begin
                                        found=1; // invalid way found
                                        DC_LRU = DataLine[DC_TagBits+DC_LRUBits-
1:DC_TagBits];
                                        DataLine[DC_TagBits-1:0] = Tag;
                    DataLine[DC_TagBits+DC_LRUBits+MESI_Size-1:DC_TagBits+DC_LRUBits]
                    = Exclusive;
                                        DataCache[way][Index] = DataLine;
                                        // <write data to this line>
                                        //first write is write through
                                        if(mode == 1)
                                                $display("Write         to L2 %h ",address);
                                end
                        end
                        // if no invalid way, look for LRU=000 and check mesi bits
                        if(found!=1)
                        begin
                                for(way=0;way<`DataCacheWay;way=way+1)
                                begin
                                        DataLine = DataCache[way][Index];
                    MESI_Bits = DataLine[DC_TagBits+DC_LRUBits+MESI_Size-1:DC_TagBits+
                    DC_LRUBits];
                                        DC_LRU = DataLine[DC_TagBits+DC_LRUBits-
1:DC_TagBits];

                                        if (DC_LRU == 0 && replaced == 0)
                                        begin
                                                replaced = 1;
                                                // if mesi bit modified write to L2 and replace,ie,
put tag bits there
                                                if(MESI_Bits == Modified)
                                                begin
                                                        Offset = 0;
                                                        if(mode == 1)
                        $display("Write to L2 %h", {DataLine[DC_TagBits-1:0], Index,
                        Offset});
                                                end
                                        DC_LRU =
DataLine[DC_TagBits+DC_LRUBits-1:DC_TagBits];
                                        DataLine[DC_TagBits-1:0] = Tag;
                    DataLine[DC_TagBits+DC_LRUBits+MESI_Size-1:DC_TagBits+
                    DC_LRUBits] = Modified;
                                        DataCache[way][Index] = DataLine;
                                    end
                                end
                                DC_LRU = 0; //To replace '0'th LRU element
                            end
                        end

                        temp = UpdateData_LRU(DC_LRU, Index);       // update lru
                    end

                    //3 invalidate command from L2
                    INVALIDATE:
                    begin
                        // look up for line by going to set and comparing tags in ways
                        for(way=0;way<`DataCacheWay;way=way+1)
                        begin
                            DataLine = DataCache[way][Index];
                    MESI_Bits = DataLine[DC_TagBits+DC_LRUBits+MESI_Size-1:DC_TagBits+
                    DC_LRUBits];
                            DC_Tag = DataLine[DC_TagBits-1:0];

                            if(Tag == DC_Tag) //if tag match
                            begin
                                MESI_Bits = Invalid; //if line found then set MESI_Bits as
invalid
                                DC_LRU = DataLine[DC_TagBits+DC_LRUBits-
1:DC_TagBits];
                      DataLine[DC_TagBits+DC_LRUBits+MESI_Size-1:DC_TagBits+DC_LRUBits] =
                      MESI_Bits;
                                DataLine[DC_TagBits-1:0] = 12'bx;
                                DataCache[way][Index] = DataLine;
                            end
                        end

                        temp = UpdateData_LRU(DC_LRU, Index);       // update lru
                    end

                    //4 data request from L2 (in response to snoop)
                    SNOOP:
                    begin
                        //look up for line with mesi modified
                        for(way=0;way<`DataCacheWay;way=way+1)
                        begin
                            DataLine = DataCache[way][Index];
                    MESI_Bits = DataLine[DC_TagBits+DC_LRUBits+MESI_Size-1:DC_TagBits+
                    DC_LRUBits];
```

```
                        DC_Tag = DataLine[DC_TagBits-1:0];

                        if((MESI_Bits == Modified) && (Tag == DC_Tag)) //if valid and tag
match
                        begin
                                found=1;        // modified cache line found!
                                if(mode == 1)
                                        $display("Return data to L2 %h",address); //Return data
to L2 <address>
                        end
                        if(Tag == DC_Tag)
                        begin
                                found=1;        // modified cache line found!
                                MESI_Bits = Invalid; //if found then change to invalid
                                DC_LRU = DataLine[DC_TagBits+DC_LRUBits-
1:DC_TagBits];
        DataLine[DC_TagBits+DC_LRUBits+MESI_Size-1:DC_TagBits+DC_LRUBits] =
        MESI_Bits;
                                DataLine[DC_TagBits-1:0] = 12'bx;
                                DataCache[way][Index] = DataLine;
                        end
                end

                temp = UpdateData_LRU(DC_LRU, Index);   // update lru
        end

        //8 clear the cache and reset all state (and statistics)
        RESET:
        begin
                for (s=0; s<`DataCacheSet; s=s+1)
                begin
                        for (w=0; w<`DataCacheWay; w=w+1)
                        begin
                                DataLine=DataCache[w][s];
                                //set all the cache MESI bits to Invalid
        DataLine[DC_TagBits + DC_LRUBits + MESI_Size-1 : DC_TagBits +
        DC_LRUBits] = Invalid;
                                //Initially assign LRU bits to each cache line in a set to line
number
                                DataLine[DC_TagBits + DC_LRUBits-1 : DC_TagBits] = w;
                                //set tag bits to x
                                DataLine[DC_TagBits-1:0] = 12'bx;
                                DataCache[w][s] = DataLine;
                        end
                end

                //set all summary paramentes to '0' on reset
                DC_Read_Hit = 32'b0;
                DC_Read_Miss = 32'b0;
                DC_Reads = 32'b0;
                DC_Write_Hit = 32'b0;
                DC_Write_Miss = 32'b0;
                DC_Writes = 32'b0;
        end

        //9 print contents and state of the cache (allow subsequent trace activity)
        PRINT:
        begin
    $display("____________________________________________________________
________");
                $display("                                                    ");
                $display("                    DATA CACHE CONTENTS                    ");
    $display("________________________________________________________________
______");

                for (s=0;s<`DataCacheSet;s=s+1)
                begin
                        print = FALSE;
                        for (w=0;w<`DataCacheWay;w=w+1)
                        begin
                                DataLine = DataCache[w][s];
            MESI_Bits = DataLine[DC_TagBits+DC_LRUBits+MESI_Size-1:DC_TagBits+
            DC_LRUBits];
                                DC_LRU = DataLine[DC_TagBits+DC_LRUBits-
1:DC_TagBits];
                                DC_Tag = DataLine[DC_TagBits-1:0];
                                if(print == TRUE || MESI_Bits != Invalid || DC_Tag != "x")
                                begin
                                        if(print == FALSE)
                                        begin
                                                print = TRUE;
                                                w = -1;
                                        end
                                        else
                                        begin
                                                print_MESI[w] = MESI_Bits;
                                                print_LRU[w] = DC_LRU;
                                                print_Tag[w] = DC_Tag;
                                        end
                                end
                        end
                        if(print == TRUE)
                        begin
                                $display("Set Index : %h", s);
                                $display("Way: 1        2        3        4        5        6
7    8");
            $display("Tag:      %h      %h      %h      %h      %h      %h      %h      %h",
            print_Tag[0], print_Tag[1], print_Tag[2], print_Tag[3], print_Tag[4], print_Tag[5],
            print_Tag[6], print_Tag[7]);
            $display("LRU:      %b      %b      %b      %b      %b      %b      %b      %b",
            print_LRU[0], print_LRU[1], print_LRU[2], print_LRU[3], print_LRU[4],
            print_LRU[5], print_LRU[6], print_LRU[7]);
```

```
                $display("MESI:    %s      %s      %s      %s      %s      %s      %s      %s",
                Get_MESI_ID(print_MESI[0]), Get_MESI_ID(print_MESI[1]),
                Get_MESI_ID(print_MESI[2]), Get_MESI_ID(print_MESI[3]),
                Get_MESI_ID(print_MESI[4]), Get_MESI_ID(print_MESI[5]),
                Get_MESI_ID(print_MESI[6]), Get_MESI_ID(print_MESI[7]));
                                $display("                ** END OF SET **                ");
                                $display("--------------------------------------------------------------
-");
                        end
                    end
                    $display("              END OF DATA CACHE CONTENTS              ");
        $display("_______________________________________________________________
____");
            end

            endcase
    end

    //function to update LRU values of each line in a set based on line reference
    function UpdateData_LRU;
            input [DC_LRUBits-1:0] LRU;
            input [DC_IndexBits-1:0] index;

            integer w;

            begin
                    //update the LRU bits of each cache line in a set
                    for (w=0; w<`DataCacheWay; w=w+1)
                    begin
                            DataLine = DataCache[w][index];
                            DC_LRU = DataLine[DC_TagBits+DC_LRUBits-1:DC_TagBits];
                            //if it is a most recently used/refered cache line then set the LRU bits to
111
                            if(DC_LRU == LRU)
                            begin
                                    DataLine[DC_TagBits+DC_LRUBits-1:DC_TagBits] =
`DataCacheWay-1;
                                    DataCache[w][index] = DataLine;
                            end
                            //LRU bits higher than the most recently used bits are decremented by 1
                            else if(DC_LRU > LRU)
                            begin
                                    DataLine[DC_TagBits+DC_LRUBits-1:DC_TagBits] =
DC_LRU-1;
                                    DataCache[w][index] = DataLine;
                            end
                    end
            end
    endfunction

    //function to get MESI ids for a given 2 bit MESI value
    function [8:0] Get_MESI_ID;
            input [1:0] MESI;

            begin
                    case(MESI)
                    2'b00: Get_MESI_ID = "I";
                    2'b01: Get_MESI_ID = "E";
                    2'b10: Get_MESI_ID = "S";
                    2'b11: Get_MESI_ID = "M";
                    endcase
```

4.5 Statistics.v

```
/*
// ECE 485/585: Microprocessor System Design
// Final Project
// Fall 2018
// File : Statistics.v
// Authors : Vinitha Baddam, Michael Bourquin and Hima Ethakota
// Description : This module is for printing statistics
*/


module STATISTICS(
    input done,
    input [31:0] DC_Read_Hit,
    input [31:0] DC_Read_Miss,
    input [31:0] DC_Reads,
    input [31:0] DC_Write_Hit,
    input [31:0] DC_Write_Miss,
    input [31:0] DC_Writes,
    input [31:0] IC_Read_Hit,
    input [31:0] IC_Read_Miss,
    input [31:0] IC_Reads
    );

    //Execute below when done is true
    always @(posedge done)
    begin
            $display("Data Cache Usage Statistics:");
            $display("Number of cache reads                 : %d", DC_Reads);
            $display("Number of cache writes        : %d", DC_Writes);
            $display("Number of cache hits          : %d", DC_Read_Hit + DC_Write_Hit);
            $display("Number of cache misses        : %d", DC_Read_Miss + DC_Write_Miss);
      $display("Cache       hit ratio          : %.2f%% \n", (DC_Reads + DC_Writes) != 0 ? 100.00
      * (DC_Read_Hit + DC_Write_Hit)/(DC_Reads + DC_Writes) : 0);

            $display("Instruction Cache Usage Statistics:");
            $display("Number of cache reads                 : %d", IC_Reads);
            $display("Number of cache hits          : %d", IC_Read_Hit);
            $display("Number of cache misses        : %d", IC_Read_Miss);
    $display("Cache       hit ratio          : %.2f%% \n", IC_Reads != 0 ?
    100.00*(IC_Read_Hit)/(IC_Reads) : 0);
  end

endmodule
```

5. TEST CASES WITH OUTPUT

5.1 explained.txt

```
0 984DE132
0 116DE12F
0 100DE130
0 999DE12E
0 645DE10A
0 846DE107
0 211DE128
0 777DE133
9
0 999DE132
1 116DE123
1 666DE135
1 333DE12C
0 846DE10C
0 777DE136
1 ABCDE128
0 116DE101
1 100DE101
1 AAADE101
1 EDCDE101
4 AAADE101
```

9

Output: MODE=0

```
# ________________________________________
#
#    INSTRUCTION CACHE CONTENTS
# ________________________________________
#   END OF INSTRUCTION CACHE CONTENTS
# ________________________________________
# __________________________________________________________________________
#
#                    DATA CACHE CONTENTS
# __________________________________________________________________________
# Set Index : 00003784
# Way:        1              2              3              4              5              6
       7              8
# Tag:                 984     116     100     999     645     846     211     777
# LRU:                 000     001     010     011     100     101     110     111
# MESI:       E              E              E              E              E              E
       E              E
#                     ** END OF SET **
# --------------------------------------------------------------
#                  END OF DATA CACHE CONTENTS
# __________________________________________________________________________
# ________________________________________
#
#    INSTRUCTION CACHE CONTENTS
# ________________________________________
#   END OF INSTRUCTION CACHE CONTENTS
# ________________________________________
# __________________________________________________________________________
#
#                    DATA CACHE CONTENTS
# __________________________________________________________________________
# Set Index : 00003784
# Way:        1              2              3              4              5              6
       7              8
# Tag:                 edc     116     333     xxx             abc     846     100     777
# LRU:                 110     100     000     111     011     001     101     010
# MESI:       M              M              M              I              M              S
       M              S
#                     ** END OF SET **
# --------------------------------------------------------------
#                    END OF DATA CACHE CONTENTS
# ____________________________________________________________________________
# Data Cache Usage Statistics:
# Number of cache reads                 :       12
# Number of cache writes                :        7
# Number of cache hits                  :        5
# Number of cache misses    :       14
# Cache        hit ratio                                   : 26.32%
#
# Instruction Cache Usage Statistics:
# Number of cache reads                 :        0
# Number of cache hits                  :        0
# Number of cache misses    :        0
# Cache        hit ratio                                   : 0.00%
#
# ** Note: $stop    : N:/tb_CacheSim.v(89)
#    Time: 480 ns  Iteration: 0  Instance: /tb_CacheSim
```

5.2 reset.txt

```
       0 984DE132
0 116DE12F
0 100DE130
0 999DE12E
0 645DE10A
0 846DE107
0 211DE128
0 777DE133
9
8
9
```

Output: MODE=0

```
       # _________________________________________
#
#    INSTRUCTION CACHE CONTENTS
# _________________________________________
#   END OF INSTRUCTION CACHE CONTENTS
# _________________________________________
# ____________________________________________________________________________
#
#                      DATA CACHE CONTENTS
# ______________________________________________________________________
# Set Index : 00003784
# Way:       1              2              3              4              5              6
      7              8
# Tag:                 984     116     100     999     645     846     211     777
# LRU:                 000     001     010     011     100     101     110     111
# MESI:      E              E              E              E              E              E
      E              E
#                 ** END OF SET **
# ------------------------------------------------------------------
#               END OF DATA CACHE CONTENTS
# ______________________________________________________________________
# ______________________________________
#
#    INSTRUCTION CACHE CONTENTS
# ______________________________________
#   END OF INSTRUCTION CACHE CONTENTS
# ______________________________________
# ______________________________________________________________________
#
#                 DATA CACHE CONTENTS
# ______________________________________________________________________
#               END OF DATA CACHE CONTENTS
# ______________________________________________________________________
# Data Cache Usage Statistics:
# Number of cache reads              :       0
# Number of cache writes             :       0
# Number of cache hits               :       0
# Number of cache misses    :       0
# Cache       hit ratio                          : 0.00%
#
# Instruction Cache Usage Statistics:
# Number of cache reads              :       0
# Number of cache hits               :       0
# Number of cache misses    :       0
# Cache       hit ratio                          : 0.00%
#
# ** Note: $stop    : N:/tb_CacheSim.v(89)
#    Time: 260 ns  Iteration: 0  Instance: /tb_CacheSim
```

5.3 EmptyFile.txt

Output: MODE=0

```
# Data Cache Usage Statistics:
```

```
# Number of cache reads                    :         0
# Number of cache writes                   :         0
# Number of cache hits                     :         0
# Number of cache misses     :         0
# Cache         hit ratio                               : 0.00%
#
# Instruction Cache Usage Statistics:
# Number of cache reads                    :         0
# Number of cache hits                     :         0
# Number of cache misses     :         0
# Cache         hit ratio                               : 0.00%
#
# ** Note: $stop    : N:/tb_CacheSim.v(89)
#    Time: 60 ns  Iteration: 0  Instance: /tb_CacheSim
```

5.4 InstructionFetch.txt

```
2 984DE132
2 116DE12F
2 100DE130
2 999DE12E
9
2 645DE10A
2 846DE107
2 211DE128
9
```

Output:

```
# ______________________________________________________________________
#
#                      DATA CACHE CONTENTS
# ______________________________________________________________________
#                    END OF DATA CACHE CONTENTS
# ______________________________________________________________________
# ______________________________________
#
#     INSTRUCTION CACHE CONTENTS
# ______________________________________
# Set Index : 00003784
# Way:          1                  2                  3                  4
# Tag:                  984       116       100       999
# LRU:                  00                  01                  10                  11
# MESI:         E                    E                    E                    E
#       ** END OF SET **
# -----------------------------------
#   END OF INSTRUCTION CACHE CONTENTS
# ________________________________________
# __________________________________________________________________________
#
#                       DATA CACHE CONTENTS
# __________________________________________________________________________
#                     END OF DATA CACHE CONTENTS
# __________________________________________________________________________
# ________________________________________
#
#    INSTRUCTION CACHE CONTENTS
# ________________________________________
# Set Index : 00003784
# Way:          1                    2                    3                    4
# Tag:                   645      846      211      999
# LRU:                   01                  10                  11                  00
# MESI:         E                    E                    E                    E
#       ** END OF SET **
# -----------------------------------
#   END OF INSTRUCTION CACHE CONTENTS
# ________________________________________
# Data Cache Usage Statistics:
# Number of cache reads                 :        0
# Number of cache writes                :        0
# Number of cache hits                  :        0
# Number of cache misses    :        0
# Cache        hit ratio                              : 0.00%
#
# Instruction Cache Usage Statistics:
# Number of cache reads                 :        7
# Number of cache hits                  :        0
# Number of cache misses    :        7
# Cache        hit ratio                              : 0.00%
#
# ** Note: $stop    : N:/tb_CacheSim.v(89)
#    Time: 220 ns  Iteration: 0  Instance: /tb_CacheSim
```