**Microprocessor System Design**

**ECE – 485/585**

**Fall 2018**

**Final Project – Cache Simulation**

Vinitha Baddam <[vbaddam@pdx.edu](mailto:vbaddam@pdx.edu)>

Michael Bourquin <[bourquin@pdx.edu](mailto:bourquin@pdx.edu)>

Hima Ethakota <hima@pdx.edu>

# Contents

1. OVERVIEW 2

2. SPECIFICATIONS 4

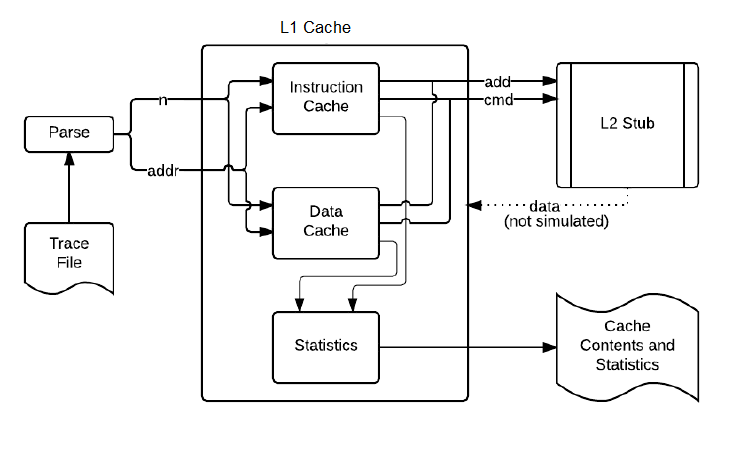
3. ASSUMPTIONS 4

4. SOURCE CODE 5

5. TEST CASES WITH OUTPUT 27

# **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 shows 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*

*\*/*

**`defin**e 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