

1 memtest.v

```

////////////////////////////////////////////////////////////////
// ECE 485/585: Microprocessor System Design
// Portland State University – Fall 2012
// Final Project:
//
// File:      memtest.v (Test Bench)
// Authors:
// Description:
//
// This module reads in a stimulus file provided by the
// command line and passes commands to the cache.
//
////////////////////////////////////////////////////////////////

module test ();

    parameter CLOCK_CYCLE = 20;
    parameter CLOCK_WIDTH = CLOCK_CYCLE/2;
    parameter TRUE = 1'b1;
    parameter FALSE = 1'b0;

    reg Clock;
    integer file;      // the file handle
    reg done;
    reg [3:0] command;
    reg [31:0] value;
    reg [9000:0] filename;
    wire [25:0] add_out;
    wire [1:0] cmd_out;
    integer count;

    PROJECT project (
        .clk (Clock),
        .n(command),
        .add_in(value),
        .done(done),
        .add_out(add_out),
        .cmd_out(cmd_out)
    );

    L_NEXT l_next (
        .add_in(add_out),
        .cmd_in(cmd_out)
    );

    initial
    begin
        Clock = FALSE;
        done = FALSE;

        // Check to make sure that a stimulus file was provided
        if ($value$plusargs("stimulus=%s", filename) == FALSE)

```

```
        begin
            $display("ERROR: _No_Stimulus_specified._Please_specify\
+stimulus=<filename>_to_start.");
            $finish;
        end

        // If it was, open the file
        file = $fopen(filename, "r");
        count = 2;

        // simulate initial reset
        #CLOCK_WIDTH Clock = FALSE;
        command = 4'd8;
        #CLOCK_WIDTH Clock = TRUE;

        // While there are lines left to be read:
        while (count > 1)
            begin
                // Parse the line
                #CLOCK_WIDTH Clock = FALSE;
                count = $fscanf(file, "%d_%x", command, value);
                #CLOCK_WIDTH Clock = TRUE;
            end

        // Close the file, and finish up
        $fclose(file);
        done = TRUE;
    end
endmodule
```

2 PROJECT.v

```

/////////////////////////////////////////////////////////////////
// ECE 485/585: Microprocessor System Design
// Portland State University – Fall 2012
// Final Project:
//
// File:    PROJECT.v (Top-level wrapper module for project)
// Authors:
// Description:
//
/////////////////////////////////////////////////////////////////
module PROJECT(
    input clk ,
    input clear ,
    input [3:0] n,
    input [31:0] add_in ,
    input done ,
    output reg [25:0] add_out ,
    output reg [1:0] cmd_out
);

    // valid commands from tracefile
    parameter RESET      = 4'd8;
    parameter INVALIDATE = 4'd3;
    parameter READ       = 4'd0;
    parameter WRITE      = 4'd1;
    parameter INST_FETCH = 4'd2;
    parameter PRINT      = 4'd9;

    // signals from file to caches
    wire [31:0] i_add, d_add;
    assign i_add = add_in;
    assign d_add = add_in;

    // signals between caches and next-level cache
    wire [1:0] l2_i_cmd, l2_d_cmd;
    wire [25:0] l2_i_add, l2_d_add;

    // signals to/from stats
    wire [31:0] i_hit;
    wire [31:0] d_hit;
    wire [31:0] i_miss;
    wire [31:0] d_miss;
    wire [31:0] i_reads;
    wire [31:0] d_reads;
    wire [31:0] d_writes;

    //mux the L2 outputs
    always @(n)
    begin
        if(n == INST_FETCH)
        begin
            add_out = l2_i_add;

```

```
        cmd_out = l2_i_cmd;
    end

    else

    begin
        add_out = l2_d_add;
        cmd_out = l2_d_cmd;
    end
end

INS_CACHE i_cache (
    .clk(clk),
    .n(n),
    .add_in(add_in),
    .add_out(l2_i_add),
    .cmd_out(l2_i_cmd),
    .hit(i_hit),
    .miss(i_miss),
    .reads(i_reads)
);

DATA_CACHE d_cache (
    .n(n),
    .add_in(add_in),
    .clk(clk),
    .add_out(l2_d_add),
    .cmd_out(l2_d_cmd),
    .hit(d_hit),
    .miss(d_miss),
    .reads(d_reads),
    .writes(d_writes)
);

STATS stats(
    .print(done),
    .ins_reads(i_reads),
    .ins_hit(i_hit),
    .ins_miss(i_miss),
    .data_reads(d_reads),
    .data_writes(d_writes),
    .data_hit(d_hit),
    .data_miss(d_miss)
);

endmodule
```

3 INS_CACHE.v

```

////////////////////////////////////////////////////////////////
// ECE 485/585: Microprocessor System Design
// Portland State University - Fall 2012
// Final Project:
//
// File:    INS_CACHE.v (Instruction Cache)
// Authors:
// Description:
//
////////////////////////////////////////////////////////////////
`define SETS 1024*16
`define WAYS 2
`define SETBITS 14
`define TAGBITS 12

module INS_CACHE(
    // INPUTS
    input [3:0]  n,          // from trace file
    input [31:0] add_in,     // from trace file
    input clk ,

    // OUTPUTS
    output reg [25:0] add_out = 26'bZ, // to next-level cache
    output reg [1:0]  cmd_out = NOP,   // to next-level cache
    output reg [31:0] hit      = 32'b0, // to statistics module
    output reg [31:0] miss     = 32'b0, // to statistics module
    output reg [31:0] reads    = 32'b0 // to statistics module
);

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

// instruction cache only repends to following values of n:
parameter RESET      = 4'd8;
parameter INVALIDATE = 4'd3;
parameter INST_FETCH = 4'd2;
parameter PRINT      = 4'd9;

// instruction cache sends following commands to next-level cache
parameter READ_OUT   = 2'b01;
parameter NOP        = 2'b00;

// CACHE ELEMENTS
// LRU: 1 bit per set. Encoding: 1 = Way 1 is LRU. 0 = Way 0 is LRU
reg LRU ['SETS-1:0];
// Valid: 1 bit per way. Encoding: 1 = way is valid, 0 = not valid
reg Valid ['SETS-1:0]['WAYS-1:0];
// Tag: Tag is of size TAGBITS. One tag per way.
reg ['TAGBITS-1:0] Tag ['SETS-1:0]['WAYS-1:0];

// loop counters
integer set_cnt, way_cnt;

```

```

// internal
reg done = 1'b0;

// assignments
wire [11:0] curr_tag = add_in[31:20];
wire [13:0] curr_index = add_in[19:6];

always @(posedge clk)
begin
    add_out = 26'bZ; // always initialize address out to high-z
    cmd_out = NOP; // default to NOP, if a read happens, it will be updated
    done = FALSE; // and set internal done signal to false

    case(n)
        // RESET: iterates through all elements in the cache and sets
        // everything to 0. Also initializes hit/miss/read counters.
        RESET:
        begin
            hit = 32'b0;
            miss = 32'b0;
            reads = 32'b0;

            // for every set...
            for (set_cnt = 0; set_cnt < 'SETS; set_cnt = set_cnt + 1'b1)
            begin
                LRU[set_cnt] = 1'b0; // set the LRU to 0
                // for each way of set...
                for (way_cnt = 0; way_cnt < 'WAYS; way_cnt = way_cnt + 1'b1)
                begin
                    // clear valid and tag bits.
                    Valid [set_cnt][way_cnt] = FALSE;
                    Tag [set_cnt][way_cnt] = 'TAGBITS'b0;
                end
            end
        end

    end

// INVALIDATE: use address passed in with invalidate command as an
// index to a given line. Then, invalidate the line for which the
// stored tag equals the tag passed in add_in.
INVALIDATE:
begin
    for (way_cnt = 0; way_cnt < 'WAYS; way_cnt = way_cnt + 1'b1)
    begin
        if (!done)
        begin
            if (Tag[curr_index][way_cnt] == curr_tag)
            begin
                done = TRUE;
                Valid[curr_index][way_cnt] = FALSE;
            end
        end
    end
end
end

```

end

INST_FETCH:

begin

reads = reads + 1'b1; *// always increment read count*

*// First, look at both lines. if for either, the tags match
// and the line is valid, then the read was a hit. done
// is set to true, and execution will drop through the rest
// of the INST_FETCH routine.*

for (way_cnt = 0; way_cnt < 'WAYS; way_cnt = way_cnt + 1'b1)

begin

if (done == FALSE)

if (Tag[curr_index][way_cnt] == curr_tag &&
Valid[curr_index][way_cnt] == TRUE)

begin

LRU[curr_index] = ~way_cnt[0];

hit = hit + 1'b1;

done = TRUE;

end

else ;

end

// at this point, if done is still false, then the fetch was not a hit.

if (done == FALSE)

miss = miss + 1'b1;

*// Next, look at both lines. If either is empty then
// do a read and put result in the empty line, then set
// done to true, and execution will drop through the rest of
// the INST_FETCH routine.*

for (way_cnt = 0; way_cnt < 'WAYS; way_cnt = way_cnt + 1'b1)

begin

if (done == FALSE)

if (Valid[curr_index][way_cnt] == FALSE)

begin

// set L_NEXT command/address

add_out = add_in[31:6]; *// perform read*

cmd_out = READ_OUT; *// perform read*

Tag[curr_index][way_cnt] = curr_tag;

Valid[curr_index][way_cnt] = TRUE;

LRU[curr_index] = ~way_cnt[0];

done = TRUE;

end

end

*// Reaching this point means an eviction is needed because the
// instruction fetch was a miss, and there was no empty line
// in which to put the incoming read. So evict the LRU*

if (done == FALSE)

begin

// set L_NEXT command/address

```

        add_out          = add_in[31:6]; // perform read
        cmd_out          = READ_OUT;    // perform read

        Tag[curr_index][LRU[curr_index]] = curr_tag;
        Valid[curr_index][LRU[curr_index]] = TRUE;
        LRU[curr_index] = ~LRU[curr_index];
    end
end

PRINT:
begin
    // print header
    $display("\n-----_INSTRUCTION_CACHE_CONTENTS_-----");
    $display("_Index_ | _LRU_ | _V[0] | Tag[0] | _V[1] | Tag[1]");
    // cycle through all of the ways within a set
    for (way_cnt = 0; way_cnt < 'SETS; way_cnt = way_cnt+1)
    begin
        // print out the whole set if there are any valid lines
        if (Valid[way_cnt][0] | Valid[way_cnt][1])
        begin
            $display("_%4h_ | _%d_ | _%d_ | _%3h_ | _%d_ | _%3h",
                way_cnt['SETBITS-1:0],
                LRU[way_cnt],
                Valid[way_cnt][0],
                // print X's if invalid
                Valid[way_cnt][0] ? Tag[way_cnt][0] : 'TAGBITS'hX,
                Valid[way_cnt][1],
                // print X's if invalid
                Valid[way_cnt][1] ? Tag[way_cnt][1] : 'TAGBITS'hX
            );
        end
    end
    $display("----_END_OF_INSTRUCTION_CACHE_CONTENTS_----\n");
end

default: ; // commands this module doesn't respond to
endcase
end

endmodule

```


4 DATA_CACHE.v

```

////////////////////////////////////////////////////////////////
// ECE 485/585: Microprocessor System Design
// Portland State University - Fall 2012
// Final Project:
//
// File: DATA_CACHE.v (Data Cache)
// Authors:
// Description:
//
//
//
////////////////////////////////////////////////////////////////
`define SETS 1024*16
`define WAYS 4
`define SETBITS 14
`define TAGBITS 12

module DATA_CACHE(
    // INPUTS
    input [3:0] n, // from trace file
    input [31:0] add_in, // from trace file
    input clk,

    // OUTPUTS
    output reg [25:0] add_out = 26'bZ, // to next-level cache
    output reg [1:0] cmd_out = NOP, // to next-level cache
    output reg [31:0] hit = 32'b0, // to statistics module
    output reg [31:0] miss = 32'b0, // to statistics module
    output reg [31:0] reads = 32'b0, // to statistics module
    output reg [31:0] writes = 32'b0 // to statistics module
);

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

// data cache only responds to following values of n
parameter RESET = 4'd8;
parameter INVALIDATE = 4'd3;
parameter READ = 4'd0;
parameter WRITE = 4'd1;
parameter PRINT = 4'd9;

// data cache sends following commands to next-level cache
parameter READ_OUT = 2'b01;
parameter WRITE_OUT = 2'b10;
parameter RW_OUT = 2'b10; // Read with intent to write
parameter NOP = 2'b00;

// CACHE ELEMENTS
// LRU: 6 bits per set.
reg [5:0] LRU ['SETS-1:0];
// Valid: 1 bit per way. Encoding: 1 = way is valid, 0 = not valid

```

```

reg Valid [‘SETS-1:0] [‘WAYS-1:0];
// Tag: Tag is of size TAGBITS. One tag per way.
reg [11:0] Tag [‘SETS-1:0] [‘WAYS-1:0];

// loop counters
integer set_cnt, way_cnt;

// internal
reg done = FALSE;
reg [1:0] lru_way; // temp variable, holds output from decode_lru
reg [5:0] lru_calc_in; // temp variable, holds output from next_lru

// assignments
wire [11:0] curr_tag = add_in[31:20];
wire [13:0] curr_index = add_in[19:6];

always @(posedge clk)
begin
    add_out = 26'bZ;
    cmd_out = NOP;
    done = FALSE;

    case (n)
    RESET:
        // clear all Valid bits in the Data Cache and
        // reset the statistics counters
        begin
            hit = 32'b0;
            miss = 32'b0;
            reads = 32'b0;
            writes = 32'b0;

            // for every set
            for (set_cnt = 0; set_cnt < ‘SETS; set_cnt = set_cnt + 1'b1)
            begin
                LRU[set_cnt] = 6'b0;
            // for all ways
            for (way_cnt = 0; way_cnt < ‘WAYS; way_cnt = way_cnt + 1'b1)
            begin
                Valid [set_cnt][way_cnt] = FALSE;
                Tag [set_cnt][way_cnt] = 24'b0;
            end
            end
        end

    INVALIDATE:
        begin
            // when an invalidate command is passed in, check to see if
            // any line in the cache matches the address passed in, if
            // it does, clear the Valid bit for that line.
            for (way_cnt = 0; way_cnt < ‘WAYS; way_cnt = way_cnt + 1'b1)
            begin
                if (done == FALSE)
                begin

```

```

        if (Tag[curr_index][way_cnt] == curr_tag)
        begin
            Valid[curr_index][way_cnt] = FALSE;
            done = TRUE;
        end
    end
end
end

READ:
begin
    // increment the number of total reads since reset occurred
    reads = reads + 1'b1;
    // search the ways within the set, if there is a hit, update the LRU
    // and increment the hit counter
    for (way_cnt = 0; way_cnt < 'WAYS; way_cnt = way_cnt + 1'b1)
    begin
        if (done == FALSE)
        begin
            if (Tag[curr_index][way_cnt] == curr_tag &&
                Valid[curr_index][way_cnt] == TRUE)
            begin
                lru_calc_in      = next_lru(LRU[curr_index], way_cnt[1:0]);
                LRU[curr_index]  = lru_calc_in;
                hit               = hit + 1'b1;
                done              = TRUE;
            end
        end
    end
end

    // if there was no hit, increment the miss counter
    if (done == FALSE)
        miss = miss + 1'b1;

    // if there was no hit, check to see if there is an empty
    // line in the set, if not, evict the LRU of the line
    // and replace it with the newly read in value.
    for (way_cnt = 0; way_cnt < 'WAYS; way_cnt = way_cnt + 1'b1)
    begin
        if (done == FALSE)
        begin
            if (Valid[curr_index][way_cnt] == FALSE)
            begin
                add_out          = add_in[31:6];    // generate read
                cmd_out          = READ_OUT;        // generate read

                lru_calc_in      = next_lru(LRU[curr_index], way_cnt[1:0]);
                LRU[curr_index]  = lru_calc_in;
                Tag[curr_index][way_cnt] = curr_tag;
                Valid[curr_index][way_cnt] = TRUE;
                done              = TRUE;
            end
        end
    end
end
end

```

```

    if (done == FALSE)
begin
    add_out          = add_in[31:6];    // generate read
    cmd_out          = READ_OUT;        // generate read

    lru_way          = decode_lru(LRU[curr_index]);
    Tag[curr_index][lru_way]          = curr_tag;
    Valid[curr_index][lru_way]        = TRUE;
    lru_calc_in      = next_lru(LRU[curr_index], lru_way);
    LRU[curr_index]  = lru_calc_in;
end
end

WRITE:
begin
    // increment the number of total writes since reset occurred
    writes = writes + 1;

    // search the ways within the set, if there is a hit, update the LRU
    // and increment the hit counter
    for (way_cnt = 0; way_cnt < 'WAYS; way_cnt = way_cnt + 1'b1)
begin
    if (done == FALSE)
begin
        if (Tag[curr_index][way_cnt] == curr_tag &&
            Valid[curr_index][way_cnt] == TRUE)
begin
            // :: already have data ::

            // :: modify the data ::

            lru_calc_in      = next_lru(LRU[curr_index], way_cnt[1:0]);
            LRU[curr_index]  = lru_calc_in;
            hit              = hit + 1'b1;

            add_out          = add_in[31:6];    // write out to L2
            cmd_out          = WRITE_OUT;        // write out to L2
            done              = TRUE;
        end
    end
end

    // if there was no hit, increment the miss counter
    if (done == FALSE)
        miss = miss + 1'b1;

    // if there was no hit, check to see if there is an empty
    // line in the set, if not, evict the LRU of the line
    // and replace it with the newly read in value.
    for (way_cnt = 0; way_cnt < 'WAYS; way_cnt = way_cnt + 1'b1)
begin
    if (done == FALSE)
begin

```

```

    if (Valid[curr_index][way_cnt] == FALSE)
    begin
        add_out = add_in[31:6]; // read data w/ intent to mod
        cmd_out = RW_OUT;      // read data w/ intent to mod

        // :: modify the data ::

        lru_calc_in          = next_lru(LRU[curr_index], way_cnt[1:0]);
        LRU[curr_index]      = lru_calc_in;
        Tag[curr_index][way_cnt] = curr_tag;
        Valid[curr_index][way_cnt] = TRUE;
        done                 = TRUE;

        add_out              = add_in[31:6]; // write out to L2
        cmd_out              = WRITE_OUT;    // write out to L2
    end
end

    if (done == FALSE)
    begin
        add_out              = add_in[31:6]; // read in w/ intent to mod
        cmd_out              = RW_OUT;      // read in w/ intent to mod

        // :: modify the data ::

        lru_way              = decode_lru(LRU[curr_index]);
        Tag[curr_index][lru_way] = curr_tag;
        Valid[curr_index][lru_way] = TRUE;
        lru_calc_in          = next_lru(LRU[curr_index], lru_way);
        LRU[curr_index]      = lru_calc_in;

        add_out              = add_in[31:6]; // write out to L2
        cmd_out              = WRITE_OUT;    // write out to L2
    end
end

// Print all of the contents of the Data Cache
PRINT:
begin
    // print header
    $display("-----_DATA_CACHE_CONTENTS_-----");
    $display("_INDEX_ | _LRU_ | _V[0] | Tag[0] | _V[1] | Tag[1] | _V[2] | Tag[2] | _V[3] | Tag[3]");
    // cycle through all of the ways within a set
    for (set_cnt = 0; set_cnt < 'SETS; set_cnt = set_cnt+1)
    begin
        // print out the whole set if there are any valid lines
        if (Valid[set_cnt][3] | Valid[set_cnt][2] |
            Valid[set_cnt][1] | Valid[set_cnt][0] )
        begin
            $display("_%4h_ | _%d_ | _%d_ | _%3h_ | _%d_ | _%3h_ | _%d_ | _%3h_ | _%d_ | _%3h",
                set_cnt['SETBITS-1:0],
                decode_lru(LRU[set_cnt]),
                Valid[set_cnt][0],

```

```

Valid[set_cnt][0] ? Tag[set_cnt][0] : 'TAGBITS'hX,
Valid[set_cnt][1],
Valid[set_cnt][1] ? Tag[set_cnt][1] : 'TAGBITS'hX,
Valid[set_cnt][2],
Valid[set_cnt][2] ? Tag[set_cnt][2] : 'TAGBITS'hX,
Valid[set_cnt][3],
Valid[set_cnt][3] ? Tag[set_cnt][3] : 'TAGBITS'hX
    );
end
end
$display("-----_END_OF_DATA_CACHE_CONTENTS_-----");
end

default: ; // commands this module doesn't respond to
endcase
end

function [1:0] decode_lru;
input [5:0] lru_bits;
begin
    if      (!(|lru_bits[5:3])) decode_lru = 2'd0;
    else if (!(|lru_bits[2:1])) decode_lru = 2'd1;
    else if (! lru_bits[0])      decode_lru = 2'd2;
    else                        decode_lru = 2'd3;
end
endfunction

function [5:0] next_lru;
input [5:0] lru_bits;
input [1:0] way_accessed;
begin
    case (way_accessed)
        // Set the first 3 bits (this defines MRU 0)
        2'd0: next_lru = (lru_bits | 6'b111000);

        // Clear bit 0, Set bits 3 & 4 (MRU 1)
        2'd1: next_lru = ((lru_bits & 6'b011111) | 6'b000110);

        // Clear bits 1 & 3, Set bit 5 (MRU 2)
        2'd2: next_lru = ((lru_bits & 6'b101011) | 6'b000001);

        // Clear bits 2,4,5 (MRU 3)
        2'd3: next_lru = (lru_bits & 6'b110100);
    endcase
end
endfunction

endmodule

//this is not a firefox cache

```

5 STATS.v

```

/////////////////////////////////////////////////////////////////
// ECE 485/585: Microprocessor System Design
// Portland State University – Fall 2012
// Final Project:
//
// File:     STATS.v (Statistics Module aka It's a series of counters)
// Authors:
// Description:
//
/////////////////////////////////////////////////////////////////
module STATS(
    // INPUTS
    input  print, // mux to determine reads/writes
    input  [31:0] ins_reads ,
    input  [31:0] ins_hit ,
    input  [31:0] ins_miss ,

    input  [31:0] data_reads ,
    input  [31:0] data_writes ,
    input  [31:0] data_hit ,
    input  [31:0] data_miss
);

always @(posedge print)
begin
    $display("_STATISTICS:_");
    $display("_Hits_==_%d", data_hit + ins_hit);
    $display("_Miss_==_%d", data_miss + ins_miss);
    $display("_Reads_==_%d", data_reads + ins_reads);
    $display("_Writes_==_%d", data_writes);
    $display("_Hit_Ratio_==_%.1f%%", (data_reads + ins_reads + data_writes) == 0 ?
        0 : 100.0*(data_hit + ins_hit)/(data_reads + ins_reads + data_writes));
end
endmodule

```

6 Testbench Output

This directory contains test vectors for our simulated L1 cache. Due to the limited syntax of the trace file format, it is impossible to put comments about each test in its file. Therefore, the tests are described below:

Cached_Instr_Reads.trace: (PASS)

Multiple reads to the same instruction cache line

----- INSTRUCTION CACHE CONTENTS -----

Index	LRU	V[0]	Tag[0]	V[1]	Tag[1]
0000	1	1	000	0	xxx

--- END OF INSTRUCTION CACHE CONTENTS ---

----- DATA CACHE CONTENTS -----

INDEX	LRU	V[0]	Tag[0]	V[1]	Tag[1]	V[2]	Tag[2]	V[3]	Tag[3]
0000	1	1	000	0	xxx	0	xxx	0	xxx

----- END OF DATA CACHE CONTENTS -----

STATISTICS:

Hits	=	3
Miss	=	1
Reads	=	4
Writes	=	0
Hit Ratio	=	75.0%

Cached_Data_Reads.trace: (PASS)

Multiple reads to the same data cache line

----- INSTRUCTION CACHE CONTENTS -----

Index	LRU	V[0]	Tag[0]	V[1]	Tag[1]
0000	1	1	000	0	xxx

--- END OF INSTRUCTION CACHE CONTENTS ---

----- DATA CACHE CONTENTS -----

INDEX	LRU	V[0]	Tag[0]	V[1]	Tag[1]	V[2]	Tag[2]	V[3]	Tag[3]
0000	1	1	000	0	xxx	0	xxx	0	xxx

----- END OF DATA CACHE CONTENTS -----

STATISTICS:

Hits	=	3
Miss	=	1
Reads	=	4
Writes	=	0
Hit Ratio	=	75.0%

Interleaved_Read_Write.trace: (PASS)

Reads and Writes to the data cache

----- INSTRUCTION CACHE CONTENTS -----

Index	LRU	V[0]	Tag[0]	V[1]	Tag[1]
0000	1	1	000	0	xxx

--- END OF INSTRUCTION CACHE CONTENTS ---

----- DATA CACHE CONTENTS -----

INDEX	LRU	V[0]	Tag[0]	V[1]	Tag[1]	V[2]	Tag[2]	V[3]	Tag[3]
0000	1	1	000	0	xxx	0	xxx	0	xxx

----- END OF DATA CACHE CONTENTS -----

STATISTICS:

```
Hits    =          3
Miss    =          1
Reads   =          2
Writes  =          2
Hit Ratio = 75.0%
```

Same_Set_Instr.trace: (PASS)

Multiple reads from the same set in the instruction cache.

----- INSTRUCTION CACHE CONTENTS -----

Index	LRU	V[0]	Tag[0]	V[1]	Tag[1]
0000	0	1	000	1	001

--- END OF INSTRUCTION CACHE CONTENTS ---

----- DATA CACHE CONTENTS -----

INDEX	LRU	V[0]	Tag[0]	V[1]	Tag[1]	V[2]	Tag[2]	V[3]	Tag[3]
0000	0	1	000	1	001				

--- END OF DATA CACHE CONTENTS ---

STATISTICS:

```
Hits    =          0
Miss    =          2
Reads   =          2
Writes  =          0
Hit Ratio = 0.0%
```

Same_Set_Data.trace: (PASS)

Multiple reads from the same set in the data cache.

----- INSTRUCTION CACHE CONTENTS -----

Index	LRU	V[0]	Tag[0]	V[1]	Tag[1]
0000	0	1	000	1	001

--- END OF INSTRUCTION CACHE CONTENTS ---

----- DATA CACHE CONTENTS -----

INDEX	LRU	V[0]	Tag[0]	V[1]	Tag[1]	V[2]	Tag[2]	V[3]	Tag[3]
0000	0	1	000	1	001	1	002	1	003

--- END OF DATA CACHE CONTENTS ---

STATISTICS:

```
Hits    =          0
Miss    =          4
Reads   =          2
Writes  =          2
Hit Ratio = 0.0%
```

Instr_Conflict.trace: (PASS)

Enough reads to the same instruction cache set to cause an eviction.

----- INSTRUCTION CACHE CONTENTS -----

Index	LRU	V[0]	Tag[0]	V[1]	Tag[1]
0000	1	1	200	1	100

--- END OF INSTRUCTION CACHE CONTENTS ---

----- DATA CACHE CONTENTS -----

INDEX	LRU	V[0]	Tag[0]	V[1]	Tag[1]	V[2]	Tag[2]	V[3]	Tag[3]
0000	1	1	200	1	100				

--- END OF DATA CACHE CONTENTS ---

STATISTICS:

```

Hits    =          0
Miss    =          3
Reads   =          3
Writes  =          0
Hit Ratio = 0.0%

```

Data_Conflict.trace (PASS)

```

    Enough reads and writes to the same instruction cache set to
    cause an eviction

```

----- DATA CACHE CONTENTS -----

INDEX	LRU	V[0]	Tag[0]	V[1]	Tag[1]	V[2]	Tag[2]	V[3]	Tag[3]
0000	1	1	500	1	200	1	300	1	400

----- END OF DATA CACHE CONTENTS -----

STATISTICS:

```

Hits    =          0
Miss    =          5
Reads   =          3
Writes  =          2
Hit Ratio = 0.0%

```

Instr_Invalidate.trace (PASS)

```

    Multiple Reads Followed by Invalidate clears single line.

```

----- INSTRUCTION CACHE CONTENTS -----

Index	LRU	V[0]	Tag[0]	V[1]	Tag[1]
0000	0	1	001	1	002

----- END OF INSTRUCTION CACHE CONTENTS -----

----- INSTRUCTION CACHE CONTENTS -----

Index	LRU	V[0]	Tag[0]	V[1]	Tag[1]
0000	0	0	xxx	1	002

----- END OF INSTRUCTION CACHE CONTENTS -----

STATISTICS:

```

Hits    =          0
Miss    =          2
Reads   =          2
Writes  =          0
Hit Ratio = 0.0%

```

Data_Invalidate.trace (PASS)

```

    Multiple Reads Followed by Invalidate clears single line.

```

----- DATA CACHE CONTENTS -----

INDEX	LRU	V[0]	Tag[0]	V[1]	Tag[1]	V[2]	Tag[2]	V[3]	Tag[3]
0000	0	1	001	1	002	1	003	1	004

----- END OF DATA CACHE CONTENTS -----

----- DATA CACHE CONTENTS -----

INDEX	LRU	V[0]	Tag[0]	V[1]	Tag[1]	V[2]	Tag[2]	V[3]	Tag[3]
0000	0	1	001	0	xxx	1	003	1	004

----- END OF DATA CACHE CONTENTS -----

STATISTICS:

Hits = 0
 Miss = 4
 Reads = 4
 Writes = 0
 Hit Ratio = 0.0%

Instr_Clear.trace (PASS)

Read Followed by Clear empties data cache

----- INSTRUCTION CACHE CONTENTS -----

Index	LRU	V[0]	Tag[0]	V[1]	Tag[1]
048c	1	1	001	0	xxx

 ----- END OF INSTRUCTION CACHE CONTENTS -----

----- INSTRUCTION CACHE CONTENTS -----

Index	LRU	V[0]	Tag[0]	V[1]	Tag[1]

 ----- END OF INSTRUCTION CACHE CONTENTS -----

STATISTICS:

Hits = 0
 Miss = 0
 Reads = 0
 Writes = 0
 Hit Ratio = 0.0%

Data_Clear.trace (PASS)

Read followed by Clear empties data cache

----- DATA CACHE CONTENTS -----

INDEX	LRU	V[0]	Tag[0]	V[1]	Tag[1]	V[2]	Tag[2]	V[3]	Tag[3]
048c	1	1	001	0	xxx	0	xxx	0	xxx

 ----- END OF DATA CACHE CONTENTS -----

----- DATA CACHE CONTENTS -----

INDEX	LRU	V[0]	Tag[0]	V[1]	Tag[1]	V[2]	Tag[2]	V[3]	Tag[3]

 ----- END OF DATA CACHE CONTENTS -----

STATISTICS:

Hits = 0
 Miss = 0
 Reads = 0
 Writes = 0
 Hit Ratio = 0.0%

Instr_Invalidate_Read.trace: (PASS)

Read, Invalidate, Read uses cleared way as LRU

----- INSTRUCTION CACHE CONTENTS -----

Index	LRU	V[0]	Tag[0]	V[1]	Tag[1]
0400	0	1	001	1	002

 ----- END OF INSTRUCTION CACHE CONTENTS -----

----- INSTRUCTION CACHE CONTENTS -----

Index	LRU	V[0]	Tag[0]	V[1]	Tag[1]
0400	0	1	001	1	003

--- END OF INSTRUCTION CACHE CONTENTS ---

STATISTICS:

Hits = 0
 Miss = 3
 Reads = 3
 Writes = 0
 Hit Ratio = 0.0%

Data_Invalidate_Read.trace: (PASS)

Read, Invalidate, Read uses cleared way as LRU.

----- DATA CACHE CONTENTS -----

INDEX	LRU	V[0]	Tag[0]	V[1]	Tag[1]	V[2]	Tag[2]	V[3]	Tag[3]
0400	0	1	001	1	002	1	003	1	004

----- END OF DATA CACHE CONTENTS -----

----- DATA CACHE CONTENTS -----

INDEX	LRU	V[0]	Tag[0]	V[1]	Tag[1]	V[2]	Tag[2]	V[3]	Tag[3]
0400	0	1	001	1	002	1	005	1	004

----- END OF DATA CACHE CONTENTS -----

STATISTICS:

Hits = 0
 Miss = 5
 Reads = 5
 Writes = 0
 Hit Ratio = 0.0%