Electrical and Computer Engineering

Fariborz Maseeh College of Engineering and Computer Science

Portland State University

Winter – 2024

ECE 585 – Microprocessor System Design

Split L1 Cache Using MESI Protocol to ensure Cache Coherence

Team 15:
Sai Rohith Reddy Yerram - yerram@pdx.edu
Bhoomika Kilari - bhoo@pdx.edu
Vadnala Nikhitha - vadnala@pdx.edu
Kumar Durga Manohar Karna - kkarna@pdx.edu

Contents:

1. Objective
2. Introduction
3. L1 Split Cache
5. Cache Addressing
6. Trace File Format
7. MESI protocol
8. MESI State Diagram
9. Source Code
10. Output
11. Roles and Responsibilities
12. References

Objective:

Design and simulation of a split L1 cache for a new 32-bit processor which can be used with up to three other processors in a shared memory configuration. The system employs a MESI protocol to ensure cache coherence.

Introduction:

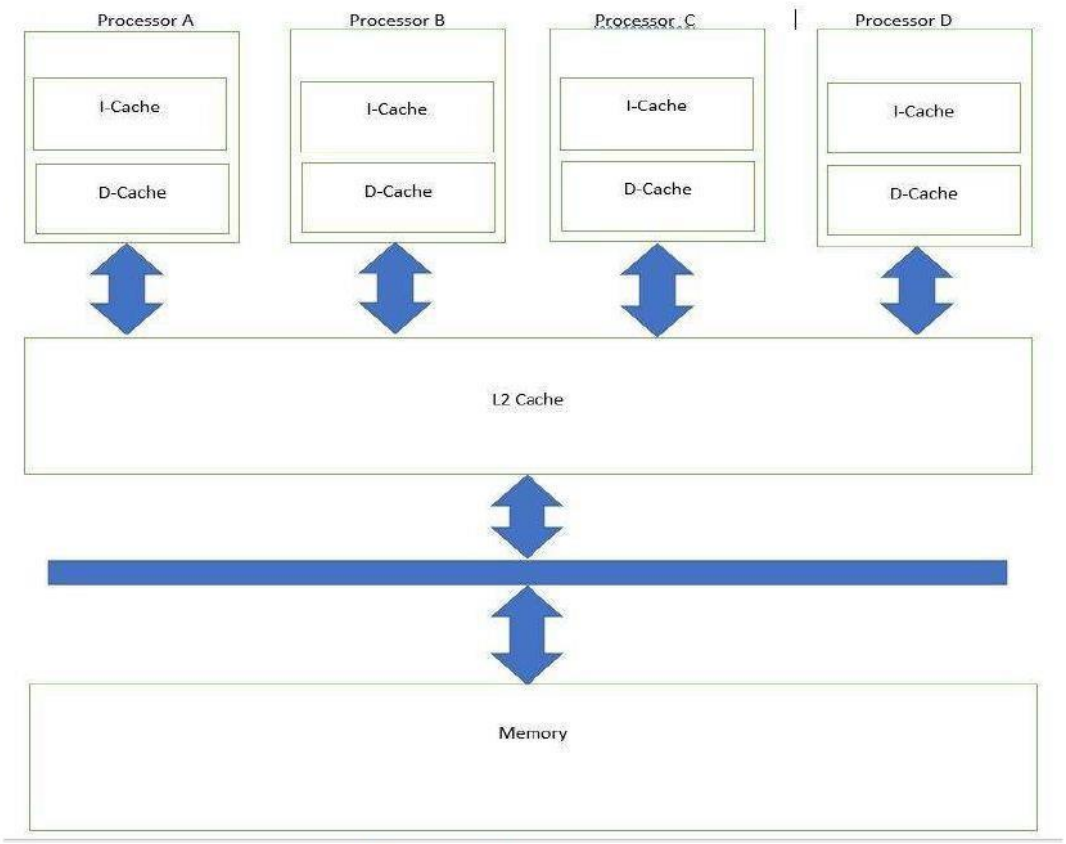


Figure 1: Block diagram

Cache is a high-speed data storage layer that stores a subset of data temporarily, typically for faster access. It is used to reduce data access latency, improve input/output (I/O) performance, and minimize the load on underlying data sources such as the CPU, memory, or storage devices. Caches can exist at various levels in a computing system, including CPU caches, disk caches, and web caches. They operate on the principle of locality, exploiting the tendency of programs to access the same data or instructions repeatedly or to access nearby data in a short period. Cache management involves strategies such as cache invalidation, which ensures that cached data remains consistent with the underlying data source, and cache replacement policies, which determine which items to remove from the cache when it reaches its capacity.

1. L1 Cache (Level 1 Cache):

- Located directly on the CPU chip.
- Small in size and fastest in terms of access time.
- Divided into separate instruction cache (L1i) and data cache (L1d) in many modern processors.
- Frequently accessed instructions and data are stored here for rapid access by the CPU cores.

2. L2 Cache (Level 2 Cache):

- Located on the CPU chip or very close to it.
- Larger in size compared to L1 cache but slower in access time.
- Serves as a secondary cache layer, supplementing the L1 cache.
- Typically shared among multiple CPU cores in multi-core processors.
- Stores additional instructions and data for quick access by the CPU cores.

3. L3 Cache (Level 3 Cache):

- Found on the CPU die or as a separate chip on the motherboard.
- Larger in size compared to L2 cache but slower in access time.
- Shared among multiple CPU cores and sometimes across multiple CPU sockets.
- Acts as a shared cache pool, providing a larger storage capacity for frequently accessed instructions and data.

4. Main Memory (RAM):

- Located off the CPU chip, typically on the motherboard.
- Much larger in size compared to cache but significantly slower in access time.
- Stores program instructions and data that cannot fit into the cache.
- Acts as a bridge between the fast cache and slower storage devices like solid-state drives (SSDs) or hard disk drives (HDDs).

Designing and simulating a L1 instruction cache is four-way set associative and consists of 16K sets and 64-byte lines. Your L1 data cache is eight-way set associative and consists of 16K sets of 64-byte lines. 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 cache hierarchy employs inclusivity is main goal.

Split L1 Cache:

Split L1 cache refers to a design where the Level 1 cache is divided into separate instruction and data caches. In this arrangement, the instruction cache holds instructions fetched from memory, while the data cache stores data accessed by the CPU. This separation allows for simultaneous access to instructions and data, potentially improving overall performance by reducing contention for cache access between instructions and data. Split L1 caches are commonly found in many modern processors, offering a balance between performance and efficiency in handling both instruction and data accesses.

1. Cache Coherence Protocols: Cache coherence protocols ensure that multiple caches in a shared-memory multiprocessor system have consistent views of memory by coordinating data updates and invalidations. Common protocols include MESI (Modified, Exclusive, Shared, Invalid), MOESI (Modified, Owned, Exclusive, Shared, Invalid), and MSI (Modified, Shared, Invalid).

2. L1 Cache Organization: L1 cache organization includes parameters like set-associativity, cache line size, and replacement policies, impacting its performance and efficiency. It directly influences the CPU's latency and throughput by determining how quickly data can be accessed by the processor.

3. Cache Hierarchy: Cache hierarchy refers to the arrangement of multiple levels of caches (L1, L2, L3) in modern processors, each serving as a buffer between the CPU and main memory. This hierarchical structure optimizes memory access times and overall system performance by prioritizing speed and proximity to the CPU for frequently accessed data.

4. Cache Write Policies: Cache write policies dictate how writes to cached data are managed: write-through writes to both cache and main memory simultaneously, while write-back initially writes only to the cache, impacting cache performance and system behaviour.

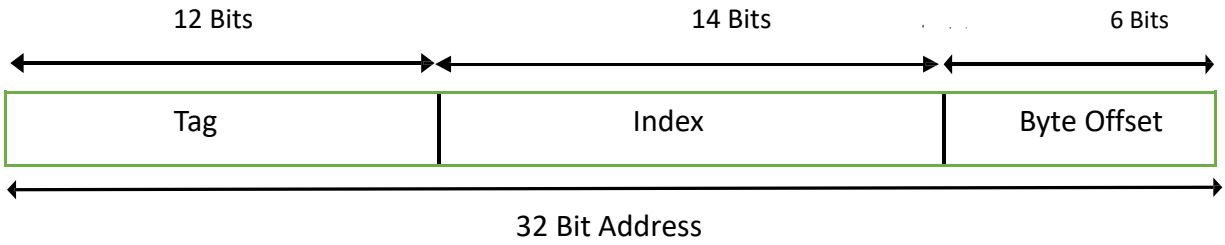
5. Cache Performance Analysis: Cache performance analysis involves evaluating metrics like cache hit rate, miss rate, and average memory access time to gauge the effectiveness of the cache in reducing memory latency, directly influencing overall system performance and efficiency.

6. Cache Replacement Algorithms: Cache replacement algorithms determine which cache line to evict when a new line needs to be fetched, with popular methods like LRU (Least Recently Used) and LFU (Least Frequently Used) aiming to maximize cache hit rates by prioritizing data likely to be accessed again soon.

Cache Addressing:

- Total number of Address bits: 32 bits.
- As each cache lines consists 64 bytes, the number of bits required for byte select are log2 (64) = 6 bits.
- The L1 Cache has 16k sets, the number of bits required to represent each set are log2 (16k) = log2 (2^14) = 14 bits.
- Therefore, the total number of Address Tag bits = Total Address bits – (bits required for byte select - bits required to represent each set)

  No. of Address Tag bits = 32 – (6+14) = 32 – 20 = 12 bits.



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)

MESI Protocol:

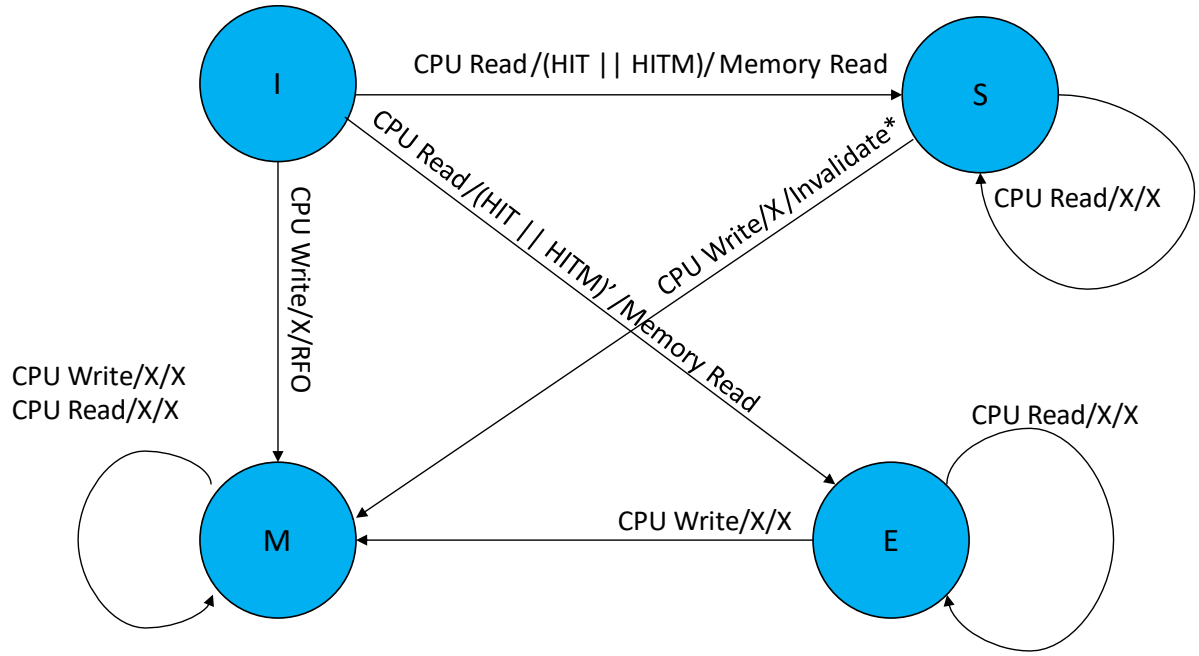
The MESI protocol, an acronym for Modified, Exclusive, Shared, and Invalid, is a widely used cache coherence protocol in multiprocessor systems. It ensures that multiple caches storing copies of the same memory location remain consistent with each other and with the underlying memory. The protocol operates by transitioning each cache line through four states—Modified, Exclusive, Shared, and Invalid—based on the actions performed on them, such as reads, writes, or cache misses.

- **Modified (M):** Indicates that the cache line is both present in the cache and has been modified from the value in main memory. This cache holds the only valid copy of the data, which must be written back to main memory before being shared or replaced.

- **Exclusive (E):** The cache line is present only in this cache and matches the main memory. It has not been modified, allowing the cache to modify the data without informing other caches (moving to the Modified state directly).

- **Shared (S):** Indicates that the cache line may be stored in multiple caches simultaneously and is consistent with main memory. Any modification requires the cache to first obtain exclusivity.

- **Invalid (I):** The cache line is not valid in the cache, indicating that it has been modified elsewhere or invalidated during a coherence operation.



Source Code:

```
module SplitL1Cache ;
        parameter Sets                  = 2**14;            // 16k sets
        parameter AddrBits              = 32;               // 32 Address bits
        parameter DWays         = 8;                        // 8-way Data Cache
        parameter IWays           = 4;                      // 4-way Instruction Cache
        parameter BytesperCacheLines        = 64;                   // 64 byte Cache lines

        localparam IndexBits            = $clog2(Sets);                 // Index bits
        localparam ByteOffsetBits               = $clog2(BytesperCacheLines);
                // byte select bits
        localparam TagBits                = (AddrBits)-(IndexBits+ByteOffsetBits);          //
Tag bits
        localparam DWayselectBits               = $clog2(DWays);                                //
Data way select bits
        localparam IWayselectBits               = $clog2(IWays);          // Instruction way
select bits

        logic   x;                                                      // Mode select
        logic   Hit;                                                // Indicates a Hit or a
Miss
        logic   NotValid;                                           // Indicates when Invalid
line is present
        logic   [3:0] n;                                    // Instruction code from Trace
file
        logic   [TagBits - 1 :0] Tag;                       // Tag
```

```
        logic   [ByteOffsetBits - 1 :0] Byte;   // Byte select
        logic   [IndexBits - 1   :0] Index;                 // Index
        logic   [AddrBits - 1 :0] Address;          // Address

        bit     [DWayselectBits - 1 :0]         Data_ways;                              // Data
ways
        bit     [IWayselectBits - 1 :0] Instruction_ways;               // Instruction ways
        bit     Flag;

        int             Trace;                                  // file descriptor
        int             TempDisplay;

        int DHitCounter         = 0;            //  Data Cache Hits count
        int DMissCounter        = 0;            //  Data Cache Misses count
        int DReadCounter        = 0;            //  Data Cache read count
        int DWriteCounter       = 0;            //  Data Cache write count

        int IHitCounter             = 0;                //  Instruction Cache Hits count
        int IMissCounter        = 0;            //  Instruction Cache Misses count
        int IReadCounter        = 0;            //  Instruction Cache read count

        real DHitRatio;                                 //  Data Cache Hit ratio
        real IHitRatio;                     //  Instruction Cache Hit Ratio

        longint CacheIterations = 0;    //  No.of Cache accesses

        // MESI states
        typedef enum logic [1:0]{
                                Invalid = 2'b00,
                                Shared              = 2'b01,
                                Modified        = 2'b10,
                                Exclusive       = 2'b11
                                } MESI;

        // L1 Data Cache
        typedef struct packed {
                                MESI MESI_bits;
                                bit [DWayselectBits-1:0]        LRUbits;
                                bit [TagBits      -1:0]     TagBits;
                                } DataCache;
DataCache [Sets-1:0] [DWays-1:0] L1DataCache;

        // L1 Instruction Cache
        typedef struct packed {
```

```
                                MESI MESI_bits;
                                bit [IWayselectBits-1:0]        LRUbits;
                                bit [TagBits     -1:0]    TagBits;
                                } InstructionCache;
InstructionCache [Sets-1:0][IWays-1:0] L1InstructionCache;



//  Read instructions from Trace File

initial   begin
          Cache_Clear();
      Trace = $fopen("traceFile.txt" , "r");
          if ($test$plusargs("USER_MODE"))
                        x=1;
          else
                x=0;
          while (!$feof(Trace))
          begin
      TempDisplay = $fscanf(Trace, "%h %h\n",n,Address);
      {Tag,Index,Byte} = Address;

                case (n) inside
                        4'd0:   Data_Read(Tag,Index,x);
                        4'd1:   DataWritetoL1DataCache (Tag,Index,x);
                        4'd2:   InstructionReadFromL1InstructionCache (Tag,Index,x);
                        4'd3:   SendInvalidateCommandFromL2Cache(Tag,Index,x);
                        4'd4:   DataRequestFromL2Cache (Tag,Index,x);
                        4'd8:   Cache_Clear();
                        4'd9:   DisplayCacheContentsMESIstates();
                endcase
          end
          $fclose(Trace);
          DHitRatio = (real'(DHitCounter)/(real'(DHitCounter) + real'(DMissCounter))) * 100.00;
          IHitRatio         = (real'(IHitCounter) /(real'(IHitCounter)  + real'(IMissCounter)))
*100.00;

          $display("---------Statistics for Data Cache----------");
          $display("Data Cache Reads= %d\n Data Cache Writes= %d\n Data Cache Hits= %d \n
Data Cache Misses= %d \n Data Cache Hit Ratio = %f\n", DReadCounter, DWriteCounter,
DHitCounter, DMissCounter, DHitRatio);

          $display("---------Instruction Cache Statistics----------");
```

```
	$display("Instruction Cache Reads = %d \nInstruction Cache Misses = %d
\nInstruction Cache Hits = %d \nInstruction Cache Hit Ratio =  %f \n",IReadCounter,
IMissCounter, IHitCounter, IHitRatio);
	$finish;

end

//Updating LRU Bits

task automatic UpdateLRUBits_data(logic [IndexBits-1:0]iIndex, ref bit [DWayselectBits-1:0]
Data_ways ); // Update LRU bits in DATA CACHE
	logic [DWayselectBits-1:0]temp;
	temp = L1DataCache[iIndex][Data_ways].LRUbits;

	for (int j = 0; j < DWays ; j++)
		L1DataCache[iIndex][j].LRUbits = (L1DataCache[iIndex][j].LRUbits > temp) ?
L1DataCache[iIndex][j].LRUbits - 1'b1 : L1DataCache[iIndex][j].LRUbits;

	L1DataCache[iIndex][Data_ways].LRUbits = '1;
endtask

task automatic UpdateLRUBits_ins(logic [IndexBits-1:0]iIndex, ref bit [IWayselectBits-1:0]
Instruction_ways ); // Update LRU bits in INSTRUCTION CACHE
	logic [IWayselectBits-1:0]temp;
	temp = L1InstructionCache[iIndex][Instruction_ways].LRUbits;

	for (int j = 0; j < IWays ; j++)
		L1InstructionCache[iIndex][j].LRUbits = (L1InstructionCache[iIndex][j].LRUbits
> temp) ? L1InstructionCache[iIndex][j].LRUbits - 1'b1 :
L1InstructionCache[iIndex][j].LRUbits;

	L1InstructionCache[iIndex][Instruction_ways].LRUbits = '1;
endtask


// Write Data to L1 Data Cache


task DataWritetoL1DataCache ( logic [TagBits -1 :0] Tag, logic [IndexBits-1:0] Index, logic x);

	DWriteCounter++ ;
	Data_Address_Valid (Index, Tag, Hit, Data_ways);

	if (Hit == 1)
```

```
	begin
		DHitCounter++ ;
		UpdateLRUBits_data(Index, Data_ways );
		if (L1DataCache[Index][Data_ways].MESI_bits == Shared)
		begin
			L1DataCache[Index][Data_ways].MESI_bits = Exclusive;
			if(x==1) $display("Write to L2 address %d'h%h" ,AddrBits,Address);
		end
		else if(L1DataCache[Index][Data_ways].MESI_bits == Exclusive)
			L1DataCache[Index][Data_ways].MESI_bits = Modified;
	end
	else
	begin
		DMissCounter++ ;
		If_Invalid_Data(Index , NotValid , Data_ways );

		if (NotValid)
		begin
			Data_Allocate_CacheLine(Index,Tag, Data_ways);
			UpdateLRUBits_data(Index, Data_ways );
			L1DataCache[Index][Data_ways].MESI_bits = Exclusive;
			if (x==1)
				$display("Read for ownership from L2 %d'h%h\nWrite to L2
address %d'h%h ",AddrBits,Address,AddrBits,Address);
		end
		else
		begin
			Eviction_DATA(Index, Data_ways);
			Data_Allocate_CacheLine(Index, Tag, Data_ways);
			UpdateLRUBits_data(Index, Data_ways );
			L1DataCache[Index][Data_ways].MESI_bits = Modified;
			if (x==1)
				$display("Read for ownership from L2
%d'h%h",AddrBits,Address);
		end
	end
endtask



//Read Data From L1 Cache


task Data_Read ( logic [TagBits-1 :0] Tag, logic [IndexBits-1:0] Index, logic x);
```

```
	DReadCounter++ ;
	Data_Address_Valid (Index,Tag,Hit,Data_ways);

	if (Hit == 1)
	begin
		DHitCounter++ ;
		UpdateLRUBits_data(Index, Data_ways );
		L1DataCache[Index][Data_ways].MESI_bits =
(L1DataCache[Index][Data_ways].MESI_bits == Exclusive) ? Shared :
L1DataCache[Index][Data_ways].MESI_bits ;
	end
	else
	begin
		DMissCounter++ ;
		NotValid = 0;
		If_Invalid_Data (Index , NotValid , Data_ways );

		if (NotValid)
		begin
			Data_Allocate_CacheLine(Index,Tag, Data_ways);
			UpdateLRUBits_data(Index, Data_ways );
			L1DataCache[Index][Data_ways].MESI_bits = Exclusive;

			if (x==1)
				$display("Read from L2 address %d'h%h" ,AddrBits,Address);
		end
		else
		begin
			Eviction_DATA(Index, Data_ways);
			Data_Allocate_CacheLine(Index, Tag, Data_ways);
			UpdateLRUBits_data(Index, Data_ways );
			L1DataCache[Index][Data_ways].MESI_bits = Exclusive;

			if (x==1)
				$display("Read from L2 address %d'h%h" ,AddrBits,Address);
		end
	end
endtask


//Instruction Fetch
```

```
task InstructionReadFromL1InstructionCache ( logic [TagBits -1 :0] Tag, logic [IndexBits-1:0]
Index, logic x);

        IReadCounter++ ;
        INSTRUCTION_Address_Valid (Index, Tag, Hit, Instruction_ways);

        if (Hit == 1)
        begin
                IHitCounter++ ;
                UpdateLRUBits_ins(Index, Instruction_ways );
                L1InstructionCache[Index][Instruction_ways].MESI_bits =
(L1InstructionCache[Index][Instruction_ways].MESI_bits == Exclusive) ? Shared :
L1InstructionCache[Index][Instruction_ways].MESI_bits      ;
        end
        else
        begin
                IMissCounter++ ;
                If_Invalid_INSTRUCTION(Index ,  NotValid , Instruction_ways );

                if (NotValid)
                begin
                        INSTRUCTION_Allocate_Line(Index,Tag, Instruction_ways);
                        UpdateLRUBits_ins(Index, Instruction_ways );
                        L1InstructionCache[Index][Instruction_ways].MESI_bits = Exclusive;
                        if (x==1)
                                $display("Read from L2 address %d'h%h" ,AddrBits,Address);
                end
                else
                begin
                        Eviction_INSTRUCTION(Index, Instruction_ways);
                        INSTRUCTION_Allocate_Line(Index, Tag, Instruction_ways);
                        UpdateLRUBits_ins(Index,  Instruction_ways );
                        L1InstructionCache[Index][Instruction_ways].MESI_bits = Exclusive;
                        if (x==1)
                                $display("Read from L2 address %d'h%h" ,AddrBits,Address);
                end
        end
endtask


//Data Request From L2 Cache

task DataRequestFromL2Cache ( logic [TagBits -1 :0] Tag, logic [IndexBits-1:0] Index, logic x);
// Data Request from L2 Cache
```

```
	Data_Address_Valid (Index, Tag, Hit, Data_ways);
	if (Hit == 1)
		case (L1DataCache[Index][Data_ways].MESI_bits) inside

			Exclusive:	L1DataCache[Index][Data_ways].MESI_bits = Shared;
			Modified :	begin
						L1DataCache[Index][Data_ways].MESI_bits =
Invalid;
						if (x==1)
							$display("Return data to L2 address
%d'h%h" ,AddrBits,Address);
					end
		endcase
endtask


//Address Valid task

task automatic Data_Address_Valid (logic [IndexBits-1 :0] iIndex, logic [TagBits -1 :0] iTag,
output logic Hit , ref bit [DWayselectBits-1:0] Data_ways );
	Hit = 0;

	for (int j = 0;  j < DWays ; j++)
		if (L1DataCache[iIndex][j].MESI_bits != Invalid)
			if (L1DataCache[iIndex][j].TagBits == iTag)
			begin
				Data_ways = j;
				Hit = 1;
				return;
			end
endtask

task automatic INSTRUCTION_Address_Valid (logic [IndexBits-1 :0] iIndex, logic [TagBits -1 :0]
iTag, output logic Hit , ref bit [IWayselectBits-1:0] Instruction_ways);
	Hit = 0;

	for (int j = 0;  j < IWays ; j++)
		if (L1InstructionCache[iIndex][j].MESI_bits != Invalid)
			if (L1InstructionCache[iIndex][j].TagBits == iTag)
			begin
				Instruction_ways = j;
				Hit = 1;
				return;
```

```
                end
endtask

//Check for Invalid states

task automatic If_Invalid_Data (logic [IndexBits-1:0] iIndex, output logic NotValid, ref bit
[DWayselectBits-1:0] Data_ways); // Find Invalid Cache line in DATA CACHE
        NotValid =  0;
        for (int i =0; i< DWays; i++ )
        begin
                if (L1DataCache[iIndex][i].MESI_bits == Invalid)
                begin
                        Data_ways = i;
                        NotValid = 1;
                        return;
                end
        end
endtask

task automatic If_Invalid_INSTRUCTION (logic [IndexBits - 1:0] iIndex, output logic NotValid,
ref bit [IWayselectBits-1:0] Instruction_ways); // Find Invalid Cache line in DATA CACHE
        NotValid =  0;
        for(int i =0; i< IWays; i++ )
                if (L1InstructionCache[iIndex][i].MESI_bits == Invalid)
                begin
                        Instruction_ways = i;
                        NotValid = 1;
                        return;
                end
endtask

//Send Invalidate Command From L2 Cache

task SendInvalidateCommandFromL2Cache ( logic [TagBits -1 :0] Tag, logic [IndexBits-1:0]
Index, logic x);

        Data_Address_Valid (Index, Tag, Hit, Data_ways);
        if (Hit == 1)
        begin
                if( x==1 && (L1DataCache[Index][Data_ways].MESI_bits == Modified))
                        $display("Write to L2 address       %d'h%h" ,AddrBits,Address);
                L1DataCache[Index][Data_ways].MESI_bits = Invalid;
        end
endtask
```

```
//Cache line Allocation

task automatic Data_Allocate_CacheLine (logic [IndexBits -1:0] iIndex, logic [TagBits -1 :0]
iTag, ref bit [DWayselectBits-1:0] Data_ways); // Allocacte Cache Line in DATA CACHE
        L1DataCache[iIndex][Data_ways].TagBits = iTag;
        UpdateLRUBits_data(iIndex, Data_ways);
endtask

task automatic INSTRUCTION_Allocate_Line (logic [IndexBits -1 :0] iIndex, logic [TagBits -1 :0]
iTag, ref bit [IWayselectBits-1:0] Instruction_ways); // Allocacte Cache Line in INSTRUCTION
CACHE
        L1InstructionCache[iIndex][Instruction_ways].TagBits = iTag;
        UpdateLRUBits_ins(iIndex, Instruction_ways);
endtask

//Eviction Line task

task automatic Eviction_DATA (logic [IndexBits -1:0] iIndex, ref bit [DWayselectBits-1:0]
Data_ways);
        for (int i =0; i< DWays; i++ )
                if( L1DataCache[iIndex][i].LRUbits ==  '0 )
                begin
                        if( x==1 && (L1DataCache[iIndex][i].MESI_bits == Modified) )
                                $display("Write to L2 address %d'h%h" ,AddrBits,Address);
                        Data_ways = i;
                end
endtask

task automatic Eviction_INSTRUCTION (logic [IndexBits - 1:0] iIndex, ref bit [IWayselectBits-
1:0] Instruction_ways);
        for (int i =0; i< IWays; i++ )
                if( L1InstructionCache[iIndex][i].LRUbits == '0 )
                begin
                        if( x==1 && (L1InstructionCache[iIndex][i].MESI_bits == Modified) )
                                        $display("Write to L2 address %d'h%h"
,AddrBits,Address);
                        Instruction_ways = i;
                end
endtask


//To Print Cache contents and MESI States
```

```
task DisplayCacheContentsMESIstates();
	$display("***DATA CACHE CONTENTS AND MESI states***");

	for(int i=0; i< Sets; i++)
	begin
		for(int j=0; j< DWays; j++)
			if(L1DataCache[i][j].MESI_bits != Invalid)
			begin
				if(!Flag)
				begin
					$display("Index = %d'h%h\n", IndexBits , i );
					Flag = 1;
				end
				$display(" Way = %d \n Tag = %d'h%h \n MESI = %s \n LRU =
%d'b%b", j,TagBits,L1DataCache[i][j].TagBits,
L1DataCache[i][j].MESI_bits,DWayselectBits,L1DataCache[i][j].LRUbits);
			end
		Flag = 0;
	end
	$display("----------END OF DATA CACHE----------\n\n");
	$display("***INSTRUCTION CACHE CONTENTS AND MESI states***");
	for(int i=0; i< Sets; i++)
	begin
		for(int j=0; j< IWays; j++)
			if(L1InstructionCache[i][j].MESI_bits != Invalid)
			begin
				if(!Flag)
				begin
					$display("Index = %d'h%h\n",IndexBits,i);
					Flag = 1;
				end
				$display(" Way = %d \n Tag = %d'h%h \n MESI = %s \n LRU =
%d'b%b", j,TagBits, L1InstructionCache[i][j].TagBits,
L1InstructionCache[i][j].MESI_bits,IWayselectBits,L1InstructionCache[i][j].LRUbits);
			end
		Flag = 0;
	end
	$display("-----END OF INSTRUCTION CACHE-----\n\n");
endtask

//Clear cache

task Cache_Clear();
DHitCounter    = 0;
```

```
DMissCounter = 0;
DReadCounter          = 0;
DWriteCounter   = 0;
IHitCounter     = 0;
IMissCounter   = 0;
IReadCounter  = 0;
fork
for(int i=0; i< Sets; i++)
                for(int j=0; j< DWays; j++)
                        L1DataCache[i][j].MESI_bits = Invalid;

        for(int i=0; i< Sets; i++)
                for(int j=0; j< IWays; j++)
                        L1InstructionCache[i][j].MESI_bits = Invalid;
join
endtask
endmodule
```

Outputs for Mode – 0:

```
# Errors: 0, Warnings: 0
# vsim -c work.SplitL1Cache "+ZERO_MODE"
# Start time: 23:08:53 on Mar 15,2024
# ** Note: (vsim-3813) Design is being optimized due to module recompilation...
# Loading sv_std.std
# Loading work.SplitL1Cache(fast)
# ***DATA CACHE CONTENTS AND MESI states***
# Index =          14'h000031d8
#
#  Way =           0
#  Tag =          12'h481
#  MESI = Modified
#  LRU =           3'b011
#  Way =           1
#  Tag =          12'h582
#  MESI = Exclusive
#  LRU =           3'b100
#  Way =           2
#  Tag =          12'h594
#  MESI = Exclusive
#  LRU =           3'b101
#  Way =           3
#  Tag =          12'h621
#  MESI = Exclusive
#  LRU =           3'b110
#  Way =           4
#  Tag =          12'h111
#  MESI = Exclusive
#  LRU =           3'b111
# ----------END OF DATA CACHE-----------
#
#
# ***INSTRUCTION CACHE CONTENTS AND MESI states***
# -----END OF INSTRUCTION CACHE-----
#
```

```
#
# ***INSTRUCTION CACHE CONTENTS AND MESI states***
# -----END OF INSTRUCTION CACHE-----
#
#
# ** Error (suppressible): (vsim-8604) L1SPLITCACHE-SUBMISSION.sv(96): NaN (not a number) resulted from a division operation.
# ---------Statistics for Data Cache-------------
# Data Cache Reads=           3
#  Data Cache Writes=           3
#  Data Cache Hits=           1
#  Data Cache Misses=           5
#  Data Cache Hit Ratio = 16.666667
#
# ---------Instruction Cache Statistics----------
# Instruction Cache Reads =           0
# Instruction Cache Misses =           0
# Instruction Cache Hits =           0
# Instruction Cache Hit Ratio =  -nan
#
# ** Note: $finish    : L1SPLITCACHE-SUBMISSION.sv(103)
#    Time: 0 ns  Iteration: 0  Instance: /SplitL1Cache
# 1
# Break at L1SPLITCACHE-SUBMISSION.sv line 103
```

Outputs for Mode – 1:

```
# Errors: 0, Warnings: 0
# vsim -c work.SplitL1Cache "+ZERO_MODE"
# Start time: 22:55:27 on Mar 15,2024
# ** Note: (vsim-3813) Design is being optimized due to module recompilation...
# Loading sv_std.std
# Loading work.SplitL1Cache(fast)
# Read for ownership from L2          32'h481c7631
# Write to L2 address          32'h481c7631
# Read from L2 address          32'h582c7632
# Read from L2 address          32'h594c7615
# Read from L2 address          32'h621c7600
# Read for ownership from L2          32'h111c762c
# Write to L2 address          32'h111c762c
# ***DATA CACHE CONTENTS AND MESI states***
# Index =          14'h000031d8
#
#  Way =           0
#  Tag =          12'h481
#  MESI = Modified
#  LRU =           3'b011
#  Way =           1
#  Tag =          12'h582
#  MESI = Exclusive
#  LRU =           3'b100
#  Way =           2
#  Tag =          12'h594
#  MESI = Exclusive
#  LRU =           3'b101
#  Way =           3
#  Tag =          12'h621
#  MESI = Exclusive
#  LRU =           3'b110
#  Way =           4
#  Tag =          12'h111
#  MESI = Exclusive
#  LRU =           3'b111
```

```
#  MESI = Exclusive
#  LRU =           3'b111
# ----------END OF DATA CACHE-----------
#
#
# ***INSTRUCTION CACHE CONTENTS AND MESI states***
# -----END OF INSTRUCTION CACHE-----
#
#
# ** Error (suppressible): (vsim-8604) L1SPLITCACHE-SUBMISSION.sv(96): NaN (not a number) resulted from a division operation.
# ---------Statistics for Data Cache-------------
# Data Cache Reads=           3
#  Data Cache Writes=           3
#  Data Cache Hits=           1
#  Data Cache Misses=           5
#  Data Cache Hit Ratio = 16.666667
#
# ---------Instruction Cache Statistics----------
# Instruction Cache Reads =           0
# Instruction Cache Misses =           0
# Instruction Cache Hits =           0
# Instruction Cache Hit Ratio =  -nan
#
# ** Note: $finish    : L1SPLITCACHE-SUBMISSION.sv(103)
#    Time: 0 ns  Iteration: 0  Instance: /SplitL1Cache
# 1
# Break at L1SPLITCACHE-SUBMISSION.sv line 103
```

Roles and Responsibilities:

1. Sai Rohith Reddy Yerram – Data Cache Coding Implementation and Documentation.
2. Bhoomika Kilari – Data Cache read and write Implementation and Documentation.
3. Vadnala Nikhitha – MESI Protocol code implementation and Documentation.
4. K Durga Manohar K – Instruction Cache code implementation and Documentation.

References:

Lectures and Slides shared by Professor Yuchen Huang.