ECE 485 Final Project: MESI PLRU Cache Simulator
Fall 2019

JP Grafeen, Jordan Bergmann, Kelly Makinster

Project Description:

Our team is responsible for the simulation of the last level cache (LLC) for a new processor that can be used with up to three other processors in a shared memory configuration. The cache has a total capacity of 16MB, uses 64-byte lines, and is 8-way set associative. It employs a write allocate policy and uses the MESI protocol to ensure cache coherence. The replacement policy is implemented with a pseudo-LRU scheme.

Not included in this simulation, the processor's next higher level cache uses 64-byte lines and is 4-way set associative. It employs a write-once policy: the first write to a line is write through; subsequent writes to the same line are write-back. Our system must maintain inclusivity.

This program models and simulates this cache's behavior (hits, misses, evictions, etc.) independent of the data. This includes communication between our LLC and the next higher level cache, bus operations that our LLC performs, snoop results that our LLC reports on the bus in response to snooping the simulated bus operations of other processors and their caches. We chose to do all of this in C++.

Assumptions:

- Write-hit action is not defined for the L2 cache. We have decided to use a Write-Back policy for the hit action.
- Write-miss action is not defined for the L1 cache. We have decided to use a Write Allocate policy for the miss action.
- Professoritorial Assumptions: The usage of an unsigned int in the required debug print statements, which is either 2 or 4 bytes depending on the compiler, indicates the address length is either 16b or 32b. As 16b is too small to fit the required offset, and index, 32b is assumed for the address length.
- Professoritorial Assumptions: The eviction policy should first verify that there are no open lines to copy into. Eviction should only occur if all lines are valid and a L1 Read or Write occurs.

Technical Details:

Cache: Cache is 16MiB, uses 64B lines, and is 8-way set associative using an assumed address size of 32b.

- 64B * 8 = 512B/index 16MiB / 512B = 32Ki indexes
- Log2(64) = 6'b
- log2(32K) = 15'b

∴ Addressing: [00-05] Byte Select, [06-20] Index, [21-32] Tag

Programming Container:

```
struct Tag_Array_Entry{
   char        MESI,          // 1 hot encoding of the MESI states using 8'b vector
   unsigned int Tag           // Data tag
}

Tag_Array_Entry   Tag_Array[Index][Associativity]
```

The common definitions used in this project are as follows:

```
#define CacheSize 16777216   //16MiB
#define CacheLine 64     //64Bytes
#define CacheAssc 8      //8-way set associativity
#define NofIndex  CacheSize / (CacheLine * CacheAssc)

#define BUS_READ        0x01  // Read request placed on Bus
#define BUS_WRITE       0x02  // Write request placed on Bus
#define BUS_INV         0x03  // Invalidate command placed on Bus
#define BUS_RWIM        0x04  // Read with intent to modify placed on Bus

#define SNP_NOHIT       0x00  // Miss reply to snooped request
#define SNP_HIT         0x01  // Hit reply to snooped request
#define SNP_HITM        0x02  // Hit Modified reply to snooped request

#define MSG_GETLINE   0x01  //L1 should send the latest version of a line to L2
#define MSG_SENDLINE  0x02  //L2 is sending a modified line to L1
#define MSG_INV_LINE  0x03  //L1 should invalidate the line
#define MSG_EVICTLINE 0x04  //L1 should evict the line
```

Traces:

The testbench will read events from a text file of the following format: *n address*
where *n* is the 'operation' (Communication, Snoop Result, or Debug Information)

- 0 read request from L1 data cache
- 1 write request from L1 data cache
- 2 read request from L1 instruction cache
- 3 snooped invalidate command
- 4 snooped read request
- 5 snooped write request
- 6 snooped read with intent to modify request
- 8 clear the cache and reset all state
- 9 print contents and state of each valid cache line (allow subsequent trace activity)

And where *address* is a hex value.

- The trace file needs to be passed to the program as a command line argument (argv).
  - This is done in mainline file.
  - main() will error check to see if the trace file exists and if so, open it and begin reading line by line until the end of the file.
  - For each line in the trace file the operation and address needs to be translated and the appropriate function called.
  - There should be a function for all 9 operations
    1. L1_Data_Read(unsigned int address)
    2. L1_Data_Write(unsigned int address)
    3. L1_Inst_Read(unsigned int address)
    4. SNP_Invalidate(unsigned int address)
    5. SNP_Read(unsigned int address)
    6. SNP_Write(unsigned int address)
    7. SNP_RWIM(unsigned int address)
    8. Clear_Cache()
    9. Print_Cache()

For each line in the trace file the operation needs to be reported if in the correct "mode"

- void BusOperation // Used to simulate a bus operation and to capture the snoop results of other LLC
- char GetSnoopResult // Used to simulate the reporting of snoop results by other caches
- void PutSnoopResult // Used to report the result of our snooping bus operations performed by other caches
- void MessageToCache // Used to simulate communication to our upper level cache. ex: L2 eviction "back invalidation"

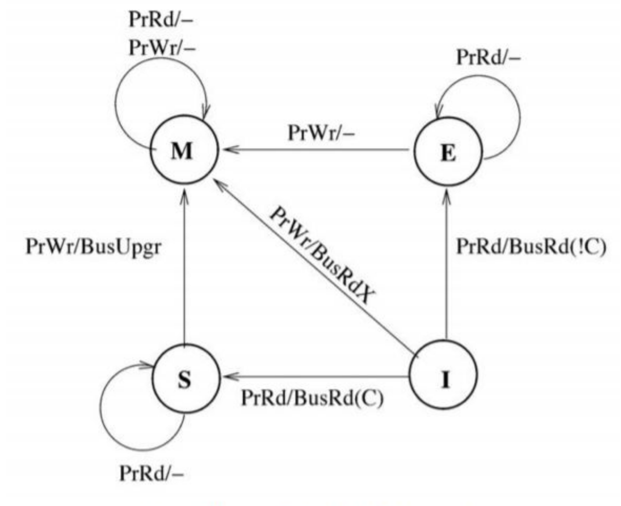
Each of these functions have an example stub in the project specification pdf.
The print statement styling should be preserved in the switch from C to C++.

Pseudo-PRLU:

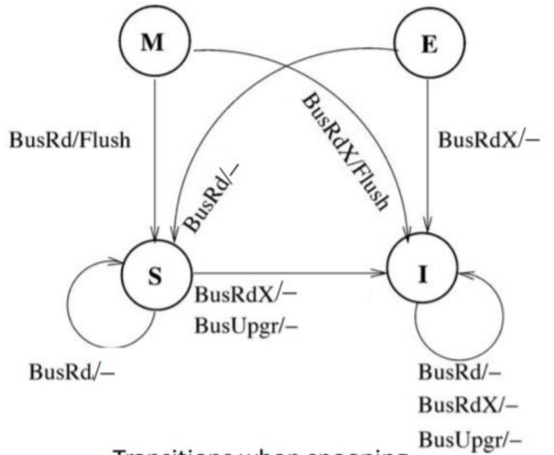
- This cache simulation uses Pseudo Least Recently Used (PLRU) for the eviction policy.
- If there are empty cache lines, the lowest valued way will be filled first.
- Given an index, on a miss, the Victim will be selected and overwritten with the new tag and control bits
- When a way is used, the pLRU array is updated such that if you go left for 0 and right for 1 you will arrive at the Most Recently Used way.
- The least recently used way is found by reversing that direction and going right for 0 and

MESI Protocol:

The MESI Protocol is used to maintain Cache coherence with other LLCs for different processors. The state changes for Snooped Operations and L1 to L2 Communication are detailed from class slides below.



Transitions when CPU Request



Transitions when snooping

Combined MESI and L1 to L2 Cache operations:

Combined Cache operations

- For Operation 0 'read request from L1 data cache'
    - on a hit,
        - increment m_CacheHit
        - increment m_CacheRead
        - stay in current MESI state
        - simulate SENDLINE
    - on a miss,
        - increment m_CacheMiss

```
            - increment m_CacheRead
            - if required, Evict cache line and send EVICTLINE to L1
                - if MESI state is 'Modified' for victim
                    - simulate GETLINE
                    - simulate a BusOperation for WRITE
            - simulate BusOperation for READ
            - set tag
            - if GetSnoopResult returns HITM or HIT update state to S
                - else update MESI state to 'Exclusive'
            - simulate SENDLINE

- For Operation 1 'write request from L1 data cache'
    - on a hit,
        - increment m_CacheHit
        - increment m_CacheWrite
        - state == M
            - stay in M
        - state == E
            - move to M
        - state == S
            - move to M
            - simulate BusOperation for INVALIDATE
        - state == I
            - Error State - can't have hit and Invalid
    - on a miss,
        - increment m_CacheMiss
        - increment m_CacheWrite
        - if required, Evict cache line and send EVICTLINE to L1
            - if MESI state is 'Modified' for victim
                - simulate GETLINE
                - simulate a BusOperation for WRITE
        - simulate BusOperation for RWIM
        - set tag
        - simulate SENDLINE
        - move to M (shoulda been in I)

- For Operation 2 'read request from L1 instruction cache'
    - on a hit,
        - increment m_CacheHit
        - increment m_CacheRead
        - stay in current MESI state
        - simulate SENDLINE
    - on a miss,
                - increment m_CacheMiss
                - increment m_CacheRead
                - if required, Evict cache line and send EVICTLINE to L1
                        - if MESI state is 'Modified' for victim
                                - simulate GETLINE
                                - simulate a BusOperation for WRITE
                - simulate BusOperation for READ
                - set tag
                - if GetSnoopResult returns HITM or HIT update state to S
                        - else update state to E
                - simulate SENDLINE

- For Operation 3 'snooped invalidate command'
        - on a hit,
                - state == M
                        - Error State - You should have been EXCLUSIVE
                - state == E
                        - Error State - You should have been EXCLUSIVE
                - state == S
                        - simulate PutSnoopResult for HIT
                        - move to I
                        - simulate INVALIDATELINE
                - state == I
                        - Error State - can't have hit and Invalid
        - on a miss,
                - simulate PutSnoopResult for NOHIT

- For Operation 4 'snooped read request'
        - on a hit,
                - simulate PutSnoopResult for HIT
                - state == M
                        - simulate a GETLINE
                        - simulate a BusOperation for WRITE
                        - move to S
                - state == E
                        - move to S
                - state == S
                        - stay in S
                - state == I
                        - Error State - can't have hit and Invalid
        - on a miss,
                - simulate PutSnoopResult for NOHIT
- For Operation 5 'snooped write request'
    - on a hit,
        - state == M
            - Error State - You should have been EXCLUSIVE
        - state == E
            - Error State - You should have been EXCLUSIVE
        - state == S
            - simulate PutSnoopResult for HIT
            - move to I
            - simulate INVALIDATELINE
        - state == I
            - Error State - can't have hit and Invalid
    - on a miss,
        - simulate PutSnoopResult for NOHIT

- For Operation 6 'snooped read with intent to modify request'
    - on a hit,
        - simulate PutSnoopResult for HIT
        - state == M
            - simulate a GETLINE
            - simulate a BusOperation for WRITE
            - move to I
            - simulate INVALIDATELINE
        - state == E
            - move to I
            - simulate INVALIDATELINE
        - state == S
            - move to I
            - simulate INVALIDATELINE
        - state == I
            - Error State - can't have a hit and Invalid
    - on a miss,
        - simulate PutSnoopResult for NOHIT
```

Test Cases:

JP Grafeen, Jordan Bergmann, Kelly Makinster						
Test Case Name:		**L1 to L2 Communication**	**Test ID:**		L1toL2_TEST	
Reference Doc.		**design_specification.md**				
Description:		Test to make sure that L1 data cache/instruction cache can communicate with L2 cache in to request reads and writes	**Type**		White Box	
Tester Information:						
Name of Tester		**Kelly Makinster**	**Date:**		**Dec 09 2019**	
Software Ver:		**Latest**	**Time:**		**1500**	
Setup:		L2 cache and both L1 caches are initially empty				
step	**Action**	**Expected Result**	**Pass**	**Fail**	**N/A**	**Comments**
1	n = 0 32 bit address	L2 cache miss. L2 reads from lower level and then messages L1 with SENDLINE	**X**			
2	n = 0 Same address	L2 cache hit. L2 messages L1 with SENDLINE of its cached data	**X**			
3	n = 2 32 bit address	L2 cache miss. L2 reads from lower level and then messages L1 with SENDLINE	**X**			
4	n = 2 Same address	L2 cache hit. L2 messages L1 with SENDLINE of its cached data	**X**			
5	n = 1 32 bit address	L2 cache miss. L2 reads from lower level and then messages L1 with SENDLINE. L2's copy of the data is marked as dirty and its state is changed to 'M'	**X**			
6	n = 1 Same address	L2 cache miss. L2 reads from lower level and then messages L1 with SENDLINE. L2's copy of the data is marked as dirty and its state is changed to 'M'	**X**			
Overall Test Results:			**X**			

JP Grafeen, Jordan Bergmann, Kelly Makinster						
Test Case Name:		**Offset test**	**Test ID:**		OFFSET-TEST	
Reference Doc.		**design_specification.md**				
Description:		Read in a cache line of data. Read all data within the 64 bit offset of said cache line.	**Type**		White Box	
Tester Information:						
Name of Tester		**Kelly Makinster**	**Date:**		**Dec 09 2019**	
Software Ver:		**latest**	**Time:**		**1500**	
Setup:		L2 cache and both L1 caches are initially empty, start in debug mode				
step	**Action**	**Expected Result**	**Pass**	**Fail**	**N/A**	**Comments**
1	n = 0 32 bit address	L2 cache miss. L2 goes to lower level to read data in	**X**			
2	n = 0 Address Offset+1	L2 cache hit. L2 messages L1 with SENDLINE of its cached data	**X**			
3	Repeat for all 64 offsets	L2 cache hit. L2 messages L1 with SENDLINE of its cached data	**X**			
Overall Test Results:			**X**			

JP Grafeen, Jordan Bergmann, Kelly Makinster			
Test Case Name:	**Index test**	**Test ID:**	INDEX-TEST
Reference Doc.	**design_specification.md**		

Description:		Read in a cache line of data. Reads every offset from that address.	**Type**		White Box	
Tester Information:						
Name of Tester		**Kelly Makinster**	**Date:**		**Dec 09 2019**	
Software Ver:		**2.65**	**Time:**		**1500**	
Setup:		L2 cache and both L1 caches are initially empty, debug mode is on				
step	**Action**	**Expected Result**	**Pass**	**Fail**	**N/A**	**Comments**
1	n = 0 32 bit address	L2 cache miss. L2 goes to lower level to read data in	**X**			
2	n = 0 Address index+1	L2 cache miss. L2 goes to lower level to read data in	**X**			
3	Repeat several times	L2 cache miss. L2 goes to lower level to read data in	**X**			
Overall Test Results:			**X**			

JP Grafeen, Jordan Bergmann, Kelly Makinster						
Test Case Name:		**SNOOP TEST**	**Test ID:**	SNOOP_TEST		
Reference Doc.		**design_specification.md**				
Description:		Test if snoop scheme is working as expected.	**Type**	Black Box		
Tester Information:						
Name of Tester		**Kelly Makinster**	**Date:**	**Dec 09 2019**		
Software Ver:		**Latest**	**Time:**	**1500**		
Setup:		L2 cache and both L1 caches are initially empty, debug mode is set on.				
step	**Action**	**Expected Result**	**Pass**	**Fail**	**N/A**	**Comments**
1	Read address ending in 0x00	Snooped a HIT	**X**			
2	Read address ending in 0x01	Snooped a HITM	**X**			
3	Read address ending in 0x02	Snooped a MISS	**X**			
4	Print Cache	Address ending in 0x00 is in state S Address ending in 0x01 is in state S Address ending in 0x02 is in state E	**X**			
Overall Test Results:			**X**			

JP Grafeen, Jordan Bergmann, Kelly Makinster						
Test Case Name:	**PLRU**		**Test ID:**	PLRU_TEST		
Reference Doc.	**design_specification.md**					
Description:	Test if PLRU scheme is working as expected.		**Type**	Black Box		
Tester Information:						
Name of Tester	**Jordan Bergmann**		**Date:**	**Dec 09 2019**		
Software Ver:	**Latest**		**Time:**	**1500**		
Setup:	L2 cache and both L1 caches are initially empty, debug mode is set off.					
step	**Action**	**Expected Result**	**Pass**	**Fail**	**N/A**	**Comments**
1	Run program with plru.din	Runs	**X**			
2	Check first 8 prints of cache report ways filled as expected.	All lines start invalid, so should validate and fill the lines in order.	**X**			
3	Check next 8 prints of cache against the PLRU scheme to check if ways are being replaced in the expected order.	Each print should show one line replaced, and should be in the order expected.	**X**			
Overall Test Results:			**X**			