ECE 585
Simulated Cache Test Plan
Team 4

Portland State University
Maseeh College of Engineering & Computer Science
Department of Electrical & Computer Engineering

Authors:
Kane-Pardy, Chris
Melli, Ameer
Monti, Caleb

December 10, 2024

ECE 585
Microprocessor System Design

Contents

1 Debug Mode (Verbose)

Verbose Output Objectives and Expected Results

When Debug Mode (Verbose) is enabled, the simulation must provide comprehensive logs detailing every internal operation, including cache actions, bus operations, snoop responses, state transitions, decision-making (e.g., evictions, misses, replacements), and all fprintf debug messages. Additionally, a final summary of statistics (reads, writes, hits, misses, and hit ratio) should be displayed after trace execution to ensure thorough tracking and analysis of the simulation's behavior.

Test Scenarios and Procedures

1. Ensure Full Verbose Output and Track Internal State Transitions

1.1 Run a Trace File in Debug Mode (Verbose)

Procedure:

1. Run a trace file that includes cache read and write operations in Debug Mode (Verbose).
2. Verify that the output includes:
    - Detailed logs of every cache read/write operation, including address, action, and cache state changes.
    - All bus operations (e.g., READ, WRITE, INVALIDATE) with relevant address and data (if applicable).
    - Snoop results (e.g., HIT, HITM) and how the system handles these snoops.
    - Intermediate state transitions, such as evictions, misses, and replacements during cache operations.
    - Any ‘fprintf‘ debug messages inserted in the code (e.g., debug prints showing variable states, function calls, or cache-related actions).
    - Information about the application of the replacement policy (e.g., pseudo-LRU), showing how cache access patterns influence eviction and replacement decisions.
    - A final summary of statistics after the trace execution, including:
        - Total cache reads
        - Total cache writes
        - Cache hits
        - Cache misses
        - Cache hit ratio

Expected Outcome:
The verbose output should show comprehensive logs for every operation, cache state change, bus interaction, internal decision-making point (e.g., eviction or miss decisions), and all debug messages from ‘fprintf‘ statements. At the end of the trace, the output should include the accurate final summary statistics (reads, writes, hits, misses, hit ratio).

2 Normal Mode Functionality

Objectives and Expected Results for Normal Mode

The simulation must ensure detailed real-time logs of cache operations, bus activities, snoop results, and state transitions in Normal Mode, along with a final summary of statistics (cache reads, writes, hits, misses, hit ratio). Additionally, it must support seamless toggling between Normal and Silent Modes via command-line arguments without requiring recompilation, where Normal Mode provides both detailed logs and summary statistics, and Silent Mode displays only the final summary.

Test Scenarios and Procedures

1. Validate Output in Normal Mode

1.1 Execute a Trace File in Normal Mode and Check Output

Procedure:

1. Run a trace file with a mix of cache read/write operations, snoop requests, and bus transactions, ensuring Normal Mode is enabled.
2. Observe the output during execution and verify it includes:
    - Per-operation logs for cache operations, such as hits, misses, writes, and reads.
    - Detailed bus operations (e.g., READ, WRITE, INVALIDATE).
    - Snoop results (e.g., HIT, HITM).
    - No information is omitted from the real-time logs during the trace.
3. After the trace completes, confirm that the following summary statistics are displayed:
    - Total cache reads
    - Total cache writes
    - Cache hits
    - Cache misses
    - Cache hit ratio
4. Verify that these statistics are accurate based on the trace operations.

Expected Outcome:
In Normal Mode, the simulation should display detailed logs during the trace, including cache operations, bus transactions, and snoop results, followed by an accurate final summary of statistics (reads, writes, hits, misses, hit ratio).

2. Ensure Mode Switching Works Correctly Without Recompilation

2.1 Toggle Between Normal and Silent Mode Using Command-Line Arguments

Procedure:

1. Run a trace file with Normal Mode enabled and verify that detailed logs and the final statistics are shown.
2. Without recompiling, switch to Silent Mode via the command-line arguments and re-run the same trace.
3. Confirm that Silent Mode only shows the summary statistics at the end, while Normal Mode shows both detailed logs during execution and the summary at the end.

Expected Outcome:
Normal and Silent Modes should be toggleable via command-line arguments without recompilation. In Normal Mode, the output should show detailed logs during the execution and the summary at the end. In Silent Mode, only the summary statistics should be shown.

3 Silent Mode Functionality

Objectives and Expected Results for Silent Mode

Silent Mode must suppress all intermediate logs, including cache operations, bus activities, and snoop results, displaying only a concise summary of cache usage statistics (reads, writes, hits, misses, and hit ratio) at the end of each trace. Mode toggling between Silent and Normal Modes must work seamlessly via command-line arguments without requiring recompilation, ensuring flexibility for testing and deployment.

Test Scenarios and Procedures

1. Verify Suppression of Detailed Logs and Display of Cache Usage Summary in Silent Mode

1.1 Run a Trace File in Silent Mode and Check Summary Statistics

Procedure:

1. Execute a trace file in Silent Mode that includes a variety of operations (reads, writes, snoop requests).
2. Observe the output during execution and confirm that:
    - No intermediate logs for individual cache operations, bus transactions, or snoop results are displayed.
    - At the end of the trace, only a concise summary with the following statistics is displayed:
        - Total cache reads
        - Total cache writes
        - Cache hits
        - Cache misses
        - Cache hit ratio

Expected Outcome:

Silent Mode suppresses all detailed logs and only displays the summary statistics at the end of the trace. The statistics accurately reflect the counts from the trace.

2. Confirm Mode Toggle Functionality Without Recompilation

2.1 Toggle Silent and Normal Mode with Command-Line Argument

Procedure:

1. Run a trace file first in Silent Mode and confirm that only the summary is displayed.
2. Without recompiling, run the same trace file in Normal Mode (by providing a different command-line argument) and confirm that detailed logs and summary statistics appear.

Expected Outcome:

Silent and Normal Modes can be toggled easily without requiring recompilation, demonstrating mode flexibility and allowing either mode to be selected as needed.

4 Associativity Verification

Objectives and Expected Results for 16-Way Associative Cache

The cache must support 16-way set associativity by allowing up to 16 unique lines per set without eviction, confirming full utilization in every way. Adding a 17th line to a full set should trigger an eviction following the cache's replacement policy, maintaining the set's capacity. Additionally, addresses spanning the full tag and index range should map correctly to their respective sets and distribute evenly across all ways, ensuring proper use of tag and index values for mapping and distribution.

Test Scenarios and Procedures

1. Verify 16-Way Associativity

1.1 Fill a Set with 16 Unique Lines

Procedure:

1. Select a single set index and read 16 unique addresses that should map to this set.
2. Load each address into the cache sequentially and confirm that all 16 lines fit into the set without eviction.
3. Run function $n = 9$ to print cache contents.

Expected Outcome:
The 16 reads should result in 16 misses, and printed cache contents should show the chosen set of the cache filled.

2. Check Cache Set Overflow

2.1 Exceed Capacity of a Set (Eviction on 17th Line)

Procedure:

1. Read 16 unique addresses into a single set.
2. Read a 17th unique address and confirm that it causes an eviction in the selected set, which already holds 16 lines. (This does not need to explicitly test pseudo-LRU, just that an "overflow" in our set causes an eviction, which would be FIFO in this case.) Print the cache contents before and after the 17th line is read to verify that one set is filled (16 ways only).

Expected Outcome:
The 17th line triggers an eviction in the selected set and should replace the line in the 0^{th} way, confirming that the cache correctly enforces the set's capacity limit of 16 lines. (17 misses w/ 1 eviction).

3. Mapping and Set Distribution

3.1 Verify Cross-Set and Inter-Set Mapping

Procedure:

1. Write a Python script to generate a trace for every line in the range:

   (**Tag:** 0xF-0x0)(**Index:** 0x3FFF-0x0)(**Byte Select:** 0x0)

   This should yield 256 Ki unique lines (16 ways for 16 Ki different sets).

Expected Outcome:
Addresses with different set indexes should map to different sets, ensuring that the cache uses both the index and tag to determine set placement correctly. (256k different addresses all result in misses).

5 Pseudo-LRU Replacement Policy Verification

Objectives and Expect Results for PLRU Replacement

Ensure the proper functionality of the pseudo-LRU (PLRU) policy within the cache system. Specifically, the simulation must verify that when a 17th line is added to a fully populated 16-way set, the cache correctly evicts the least recently used line according to the pseudo-LRU policy. Additionally, it should validate that each cache access (read or write) accurately updates the PLRU tracking bits to represent the most recently used cache line. These tests ensure that the cache maintains correct eviction behavior under overflow conditions while also properly tracking access patterns and maintaining accurate LRU representation through its pseudo-LRU mechanism.

Test Scenarios and Procedures

1. Validate Pseudo-LRU Tracking on Access

1.1 Update Pseudo-LRU Tracking on Access

Procedure:

1. Fill a set with 16 lines to establish the initial pseudo-LRU state.
2. Access the line stored in ways 1, 3, 5, and 14. Printing out the cache contents between each access to view the updated PLRU bits

Expected Outcome:
The PLRU for the selected way should be initialized to [111111111111111].
The access to data in Way 1 should result in the PLRU being [001011111111111].
The access to data in Way 3 should result in the PLRU being [001111111111111].
The access to data in Way 5 should result in the PLRU being [011101111111111].
The access to data in Way 14 should result in the PLRU being [111101111111110].

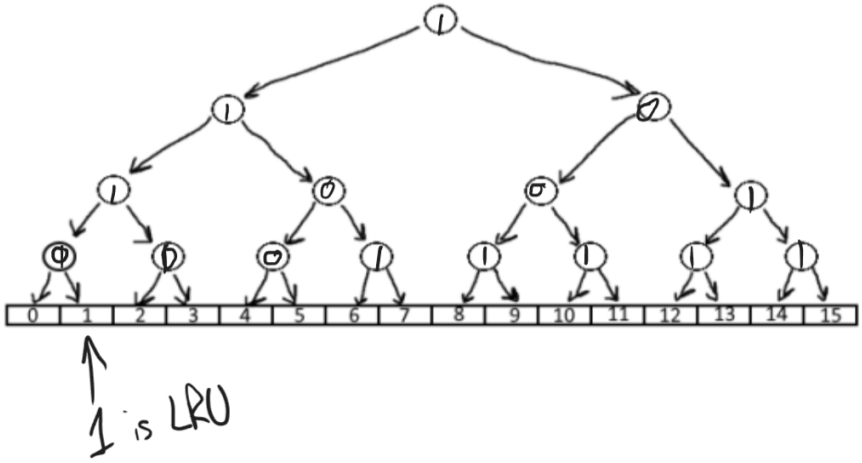
2. Verify Pseudo-LRU on Set Overflow

2.1 Insert 17 Unique Lines into a Set and Test Eviction

Procedure:

1. Fill a set with 16 unique lines.
2. Print cache contents.
3. Access the line stored in ways 0, 2, 4, and 9.
4. Read a 17th line into the cache.
5. Print cache contents.

Expected Outcome:
The cache evicts the pseudo least recently used line (which in this case should be "way 1", confirming that the pseudo-LRU policy works correctly on overflow and that the eviction respects the access history.



6 Write-Allocate Validation

Objective and Expected Outcomes of Write-Allocate

Verify that the cache correctly implements the Write-Allocate policy for write misses. Ensure that a write miss triggers the allocation of a new cache line, the entire cache line is loaded into the cache from the next level, and subsequent writes result in cache hits. Additionally, confirm that cache line state transitions follow the MESI protocol (e.g., transitioning to the Modified state) after a write-allocate and that cache statistics accurately track write misses and allocations triggered by this policy.

Test Scenarios and Procedures

1. Verify Write-Allocate Behavior and Cache Line Loading

1.1 Perform Write Miss and Ensure Cache Line is Allocated and Loaded

Procedure:

1. Use a trace file with a write miss to an address not currently in the cache.
2. Observe the cache behavior to confirm a write miss occurs and that a new cache line is allocated and fully loaded (e.g., 64 bytes) from the next level of memory into the cache during the write-allocate process.
3. Observe the MESI state transition of the allocated line to confirm it transitions to Modified.

Expected Outcome: A write miss is reported, the cache allocates a new cache line, and the full line is loaded into the cache. The cache line transitions to the MODIFIED state.

2. Validate Subsequent Writes After Write-Allocate

2.1 Perform Multiple Writes Following Write Miss

Procedure:

1. Use a trace file with a write miss followed by multiple writes to the same address.
2. Verify that these subsequent writes result in cache hits.

Expected Outcome: The first write causes a miss and triggers write-allocate. Subsequent writes to the same address should now hit the cache.

7 MESI State Accuracy

MESI States Overview

- **MODIFIED (M)**: The cache has a copy of the data that has been modified, and it is the only cache with this data. The cache is responsible for writing back the data when evicted.
- **EXCLUSIVE (E)**: The cache has the only copy of the data, but the data has not been modified. This means no other cache has the line, and it is clean in this cache.
- **SHARED (S)**: The cache line is present in multiple caches, and it is not modified (it is consistent with memory).
- **INVALID (I)**: The cache does not have a valid copy of the data.

Objectives and Expected Results for MESI State Tracking

The goal is to test the correct implementation of the MESI (MODIFIED, EXCLUSIVE, SHARED, INVALID) protocol in cache memory systems by verifying accurate state transitions during read, write, snoop, and eviction operations, ensuring proper cache coherence in multi-processor environments. After executing each test, the expected results include correct MESI state transitions during reads, writes, and snooping; proper management of SHARED, EXCLUSIVE, and MODIFIED states based on access patterns; accurate snooping responses to bus operations from other processors; and maintained cache coherence across multiple caches.

Test Scenarios and Procedures

1. SHARED State

1.1 PrRd to a SHARED Line

Procedure:

1. Simulate a PrRd to a line in the **SHARED** state.
2. Make sure the program is functioning in Normal Mode to view Bus Operations

Expected Outcome:
The state of the line should stay in SHARED.

1.2 PrWr to a SHARED Line

Procedure:

1. Simulate a PrWr to a line in the **SHARED** state.
2. Make sure the program is functioning in Normal Mode to view Bus Operations

Expected Outcome:
The state of the line should transition to MODIFIED and a BusUpgr should be issued.

1.3 BusRd to a SHARED Line

Procedure:

1. Simulate a BusRd being snooped by a line in the **SHARED** state.
2. Make sure the program is functioning in Normal Mode to view Bus Operations

Expected Outcome:
The state of the line should stay in SHARED.

1.4 BusUpgr to a SHARED Line

Procedure:

1. Simulate a BusUpgr being snooped by a line in the **SHARED** state.
2. Make sure the program is functioning in Normal Mode to view Bus Operations

Expected Outcome:
The state of the line should transition to INVALID.

2. EXCLUSIVE State

2.1 PrRd to an EXCLUSIVE Line

Procedure:

1. Simulate a PrRd to a line in the **EXCLUSIVE** state.
2. Make sure the program is functioning in Normal Mode to view Bus Operations

Expected Outcome:
The state of the line should stay EXCLUSIVE.

2.2 PrWr to an EXCLUSIVE Line

Procedure:

1. Simulate a PrWr to a line in the **EXCLUSIVE** state.
2. Make sure the program is functioning in Normal Mode to view Bus Operations

Expected Outcome:
The state of the line should transition to MODIFIED.

2.3 BusRd to an EXCLUSIVE Line

Procedure:

1. Simulate a BusRd being snooped by a line in the **EXCLUSIVE** state.
2. Make sure the program is functioning in Normal Mode to view Bus Operations

Expected Outcome:
The state of the line should transition to SHARED.

2.4 BusRdX to an EXCLUSIVE Line

Procedure:

1. Simulate a BusRdX being snooped by a line in the **EXCLUSIVE** state.
2. Make sure the program is functioning in Normal Mode to view Bus Operations

Expected Outcome:
The state of the line should transition to INVALID.

3. MODIFIED State

3.1 PrRd to a MODIFIED Line

Procedure:

1. Simulate a PrRd to a line in the **MODIFIED** state.
2. Make sure the program is functioning in Normal Mode to view Bus Operations

Expected Outcome:
The state of the line should stay MODIFIED.

3.2 PrWr to a MODIFIED Line

Procedure:

1. Simulate a PrWr to a line in the **MODIFIED** state.
2. Make sure the program is functioning in Normal Mode to view Bus Operations

Expected Outcome:
The state of the line should stay MODIFIED.

3.3 BusRd to a MODIFIED Line

Procedure:

1. Simulate a BusRd being snooped by a line in the **EXCLUSIVE** state.
2. Make sure the program is functioning in Normal Mode to view Bus Operations

Expected Outcome:
The state of the line should transition to SHARED, and a FlushWB should be issued.

3.4 BusRdX to a MODIFIED Line

Procedure:

1. Simulate a BusRdX being snooped by a line in the **MODIFIED** state.
2. Make sure the program is functioning in Normal Mode to view Bus Operations

Expected Outcome:
The state of the line should transition to INVALID, and a FlushWB should be issued.

4. INVALID State

4.1 PrRd to an INVALID Line

Procedure:

1. Simulate a PrRd to a line in the **INVALID** state.
2. Make sure the program is functioning in Normal Mode to view Bus Operations

Expected Outcome:
The state of the line should transition to either EXCLUSIVE or SHARED depending on the snoop result (HIT or HITM = SHARED, NOHIT = EXCLUSIVE).

4.2 PrWr to an INVALID Line

Procedure:

1. Simulate a PrWr to a line in the **INVALID** state.
2. Make sure the program is functioning in Normal Mode to view Bus Operations

Expected Outcome:
The state of the line should transition to MODIFIED.

4.3 BusRd/BusUpgr/BusRdX to an INVALID Line

Procedure:

1. Simulate a BusRd, BusUpgr, and BusRdX being snooped by a line in the **INVALID** state.
2. Make sure the program is functioning in Normal Mode to view Bus Operations

Expected Outcome:
The state of the line should stay INVALID.

8 Cache Coherence

Objectives and Expectations for Cache Coherence

These test scenarios aim to comprehensively validate the cache's coherence under various operations, ensuring compliance with MESI protocol and proper handling of bus operations. Specifically, scenarios test the cache's response to bus read and write commands, including triggering bus operations on cache misses, accurately transitioning between MESI states, and issuing appropriate bus commands such as BusRd, BusRdX, BusUpgr, and FlushWB. For snooped operations, the cache's responses are verified for lines in all states (SHARED, EXCLUSIVE, MODIFIED, and INVALID), in response to the commands and snoop results received (HIT, HITM, NOHIT), ensuring correct state transitions and output signals. Additionally, the scenarios confirm the cache's handling of clearing operations, where lines transition to INVALID or issue FlushWB when necessary. Expected outcomes include accurate state transitions, proper issuing of bus commands, and proper handling of shared data consistency and coherence across caches.

- **Bus Operations:** All BusRd, FlushWB, BusUpgr, and BusRdX operations execute correctly, maintaining consistency with memory and other caches.
- **MESI Protocol:** State transitions comply fully with MESI rules, ensuring correct handling of reads, writes, snooping, and evictions.
- **Snooping Behavior:** The LLC responds accurately to (and accurately issues) all snooping responses, including HIT, HITM, and NOHIT, to maintain coherence.

Test Scenarios and Procedures

1. Verify Execution of Bus Read Operations

1.1 Cache Miss Triggers Bus Read

Procedure:

1. Simulate a cache miss for a PrRd request.

Expected Outcome:
The cache sends a BusRd request, and the data is fetched.

1.2 Snoop Result HIT on BusRd

Procedure:

1. Simulate a bus read operation where another cache has the line in an unmodified valid state.

Expected Outcome:
The cache that already contains the line (in either E or S) sends a HIT signal along with its MESI state on the bus, causing the line to be read in as SHARED.

1.3 Snoop Result HITM on BusRd

Procedure:

1. Simulate a bus read operation where another cache has the line in the modified state.

Expected Outcome:
The cache that already contains the modified line sends a HITM signal, issuing a FlushWB at the current address, causing the line to be read in as SHARED.

1.4 Snoop Result NOHIT on BusRd

Procedure:

1. Simulate a bus read operation where no other cache has the line in a valid state.

Expected Outcome:
The cache receives the data in EXCLUSIVE.

2. PrWr Bus Operations

2.1 PrWr HIT in E

Procedure:

1. Simulate a PrWr Hit to a line in the EXCLUSIVE state.

Expected Outcome:
No bus operation should be observed, but the line should transition to M.

2.2 PrWr HIT in S

Procedure:

1. Simulate a PrWr Hit to a line in the SHARED state.

Expected Outcome:
A BusUpgr should be observed, and the line should transition to M.

2.3.1 PrWr MISS in I (NOHIT or HIT)

Procedure:

1. Simulate a PrWr Hit to a line in the INVALID state, where the snoop result will either be HIT or NOHIT.

Expected Outcome:
A BusRdX should be observed, and the line should be stored in MODIFIED.

2.3.2 PrWr MISS in I (HITM)

Procedure:

1. Simulate a PrWr Hit to a line in the INVALID state, where the snoop result will be HITM.

Expected Outcome:
A BusRdX and a FlushWB should be observed, and the line should be stored in MODIFIED.

3. Validate Responses to Snooped BusRd commands

3.1.1 Respond with HIT (S)

Procedure:

1. Simulate a snooped BusRd operation for a line in the SHARED state.

Expected Outcome:
Our LLC reports HIT.

3.1.2 Respond with HIT (E)

Procedure:

1. Simulate a snooped BusRd operation for a line in the EXCLUSIVE state.

Expected Outcome:
Our LLC reports HIT and transitions to SHARED.

3.2 Respond with HITM

Procedure:

1. Simulate a snooped read bus operation for a line in the MODIFIED state.

Expected Outcome:
Our LLC reports HITM and issues a FlushWB.

3.3 Respond with NOHIT

Procedure:

1. Simulate a snooped bus read operation for a line in the INVALID state in the LLC.

Expected Outcome:
The LLC replies with NOHIT.

4. Verify Result for a Snooped Bus Write

4.1 Snooped FlushWB

Procedure:

1. Perform a PrRd for a line that you'll write back from another cache.
2. Simulate the FlushWB from another cache's local eviction.

Expected Outcome:
The operation is observed by the LLC, no response is given.

5. Snooped RWIM/Invalidate

5.1.1 Snooped BusRdX for a SHARED line

Procedure:

1. Simulate a snooped BusRdX for a line that is SHARED in the LLC.

Expected Outcome:
The SHARED line reports HIT and moves to INVALID.

5.1.2 Snooped BusRdX for an EXCLUSIVE line

Procedure:

1. Simulate a snooped BusRdX for a line that is EXCLUSIVE in the LLC.

Expected Outcome:
The EXCLUSIVE line reports HIT and moves to INVALID.

5.1.3 Snooped BusRdX for a MODIFIED line

Procedure:

1. Simulate a snooped BusRdX for a line that is MODIFIED in the LLC.

Expected Outcome:
The MODIFIED line reports HITM, issues a FlushWB and moves to INVALID

5.1.4 Snooped BusRdX for an INVALID line

Procedure:

1. Simulate a snooped BusRdX for a line that is INVALID in the LLC.

Expected Outcome:
The line remains INVALID and reports NOHIT.

5.2.1 Snooped BusUpgr for a SHARED line

Procedure:

1. Simulate a snooped BusUpgr for a line that is SHARED in the LLC.

Expected Outcome:
The SHARED line reports HIT and moves to INVALID.

5.2.2 Snooped BusUpgr for an INVALID line

Procedure:

1. Simulate a snooped BusUpgr for a line that is INVALID in the LLC.

Expected Outcome:
The line remains INVALID and reports NOHIT.

6.Clear Cache

6.1 Clear Cache Command for an Unmodified Line

Procedure:

1. Run the clear cache command with lines either in the SHARED or EXCLUSIVE state.
2. Ensure to run in Normal Mode so that all operations are shown.

Expected Outcome:
The line moves to INVALID.

6.2 Clear Cache Command for a MODIFIED Line

Procedure:

1. Run the clear cache command with a line in the MODIFIED state.
2. Ensure to run in Normal Mode so that all operations are shown.

Expected Outcome:
The line moves to INVALID and issues a FlushWB.

9 Inclusivity Verification

Objectives and Expected Results for Inclusivity:

The test scenarios for SENDLINE, GETLINE, INVALIDATELINE, and EVICTLINE commands validate the interactions between the LLC and higher-level caches, ensuring inclusivity and coherence across the cache hierarchy. SENDLINE operations are verified for both PrRd and PrWr commands, where hits result in direct SENDLINE issuance, and misses trigger a BusRd followed by a SENDLINE. GETLINE commands are tested in response to BusRd commands, ensuring modified data is flushed to the higher-level cache before transitioning the line to SHARED. INVALIDATELINE scenarios confirm the correct handling of inclusivity by ensuring eviction notifications are sent to higher-level caches during collision misses or cache clear operations. EVICTLINE tests address scenarios involving MODIFIED lines, verifying that evictions or cache clears result in the write-back of modified data and proper invalidation in the higher-level cache. Expected outcomes across all tests include accurate command issuance, state transitions, and data handling to maintain cache inclusivity and coherence.

Test Scenarios and Procedures

1. SENDLINE

1.1 SENDLINE for PrRd commands

Procedure:

1. Perform a PrRd command that will result in a hit and a PrRd that will result in a miss.

Expected Outcome:
The PrRd resulting in a hit should issue a SENDLINE to the higher level cache, the PrRd resulting in a miss should issue a BusRd, then a SENDLINE.

1.2 SENDLINE for PrWr commands

Procedure:

1. Perform a PrRd command that will result in a hit and a PrRd that will result in a miss.

Expected Outcome:
The PrRd resulting in a hit should issue a SENDLINE to the higher level cache, the PrRd resulting in a miss should issue a BusRd, then a SENDLINE.

2. GETLINE

2.1 GETLINE in Response to BusRd commands

Procedure:

1. Simulate a snooped BusRd command to a MODIFIED line in the LLC.

Expected Outcome:
Since the line will be moving to SHARED, the LLC should issue a GETLINE command to the higher level cache, to FlushWB the updated data before transitioning.

3. INVALIDATELINE

3.1 INVALIDATELINE in Response to COLLISION MISSES

Procedure:

1. Fill all 16-Ways in 1 set of the cache.
2. Attempt to put a 17th line into the same set.

Expected Outcome:
The collision miss will result in a local eviction in the LLC. The higher level cache must be issued an INVALIDATELINE at the address of the line being locally evicted to ensure inclusivity.

3.2 INVALIDATELINE in Response to Cache Clear

Procedure:

1. Simulate a cache clear operation, with lines stored in SHARED or EXCLUSIVE.

Expected Outcome:
Since the lines will be getting removed from the LLC, they must also be removed from the higher-level cache.

4. EVICTLINE

4.1 EVICTLINE for Collision Misses on a modified line

Procedure:

1. PrWr to a line to put it into the MODIFIED state.
2. Fill the other 15 ways of the same set.
3. Read the 17th line into the same set.

Expected Outcome:
The collision miss will result in a local eviction in the LLC. The higher level cache must be issued an EVICTLINE at the address of the line so that the modified data can be written back, and the line will be moved to INVALID.

4.2 EVICTLINE in Response to Cache Clear

Procedure:

1. PrWr to a line to put it into the MODIFIED state.
2. Simulate a cache clear operation.

Expected Outcome:
Since the modified line will be getting removed from the LLC and needs to be written back, it must also be fetched and invalidated from the higher-level cache.

10 Write-Once Policy

Objectives and Expected Results for the Write-Once Policy

This test scenario focuses on the behavior of the LLC during the first write to a cache line and the subsequent eviction of a MODIFIED line. For PrWr operations, a simulated PrWr (equivalent to a PrWr miss in the L1) initiates the line's transition to MODIFIED as it is "written through" on the first modification to a line. A BusRd to the same cache line follows, to prompt a state change in the LLC. The expected outcome is that the MODIFIED line in the LLC issues a GETLINE command to fetch updated data from the higher-level cache, and finally transitions the line to SHARED. While we can't demonstrate a true write-once policy since the higher-level cache isn't modeled, this test will allow for an approximate simulation of expected write-once characteristics.

Test Scenarios and Procedures

1. First Write to a Cache Line, and Eviction of a MODIFIED LINE

1.1 PrWr's

Procedure:

1. Ensure the cache is operating in NormalMode
2. Simulate a PrWr, which is the same as a PrWr miss in the L1.
3. Send a BusRd to the same cache line.

Expected Outcome:

- The MODIFIED line in the LLC should first issue a GETLINE to obtain the updated data from the higher-level cache, FlushWB, and then finally transition to SHARED.

11 Cache Statistics Reporting

Objectives and Expected Results

The cache must correctly track the total number of read and write operations executed during the trace file's execution. Additionally, the cache should report accurate hit-and-miss counts based on the access patterns in the trace file while ensuring the hit ratio is calculated and reported correctly. It is crucial to validate that the cache's output behaves appropriately in both silent and normal modes. Silent mode should provide only a summary of statistics, while normal mode should log all detailed cache operations. These objectives ensure the correct tracking of read and write requests, accurate hit-and-miss detection, and proper reporting of cache performance metrics.

Test Scenarios and Procedures

1. Verify Tracking of Cache Reads and Writes

1.1 Process Read and Write Requests

Procedure:

1. Execute a trace file with a known mix of PrRd and PrWr requests to the cache.

Expected Outcome:
The counts for cache reads and writes should match the number of processor read and write requests in the trace file, confirming accurate operation tracking.

2. Confirm Cache Hit and Miss Count Accuracy

2.1 Track Hits and Misses During Cache Access

Procedure:

1. Execute a trace file containing multiple requests, with some resulting in cache hits and others in misses, the number of which should be known.

Expected Outcome:
The cache hit and miss counts should be accurately reported, matching the expected results based on the trace file.

3. Calculate and Report Cache Hit Ratio

3.1 Compute and Display Cache Hit Ratio

Procedure:

1. Run a trace file and record the total number of cache hits and misses.
2. Calculate the hit ratio as:

$$\text{hit ratio} = \frac{\text{number of hits}}{\text{number of hits} + \text{number of misses}}$$

Expected Outcome:
The displayed hit ratio should match the calculated value, ensuring that the system correctly tracks and reports cache performance.

4. Output Statistics in Silent and Normal Modes

4.1 Verify Output in Silent and Normal Mode

Procedure:

1. Run a trace file in Silent Mode.
2. Confirm that the only output displayed is the summary of cache statistics: total reads, writes, hits, misses, and hit ratio.
3. Swap to Normal Mode and ensure that along with bus operations now being visible, the cache results report matches the one from silent mode.

Expected Outcome:
Both cache results summaries should be identical.