Alexandria University
Faculty of Engineering
Computer Architecture
Spring 2025

Cache Coherence Report

Mohamed Atef Yousef 20011630

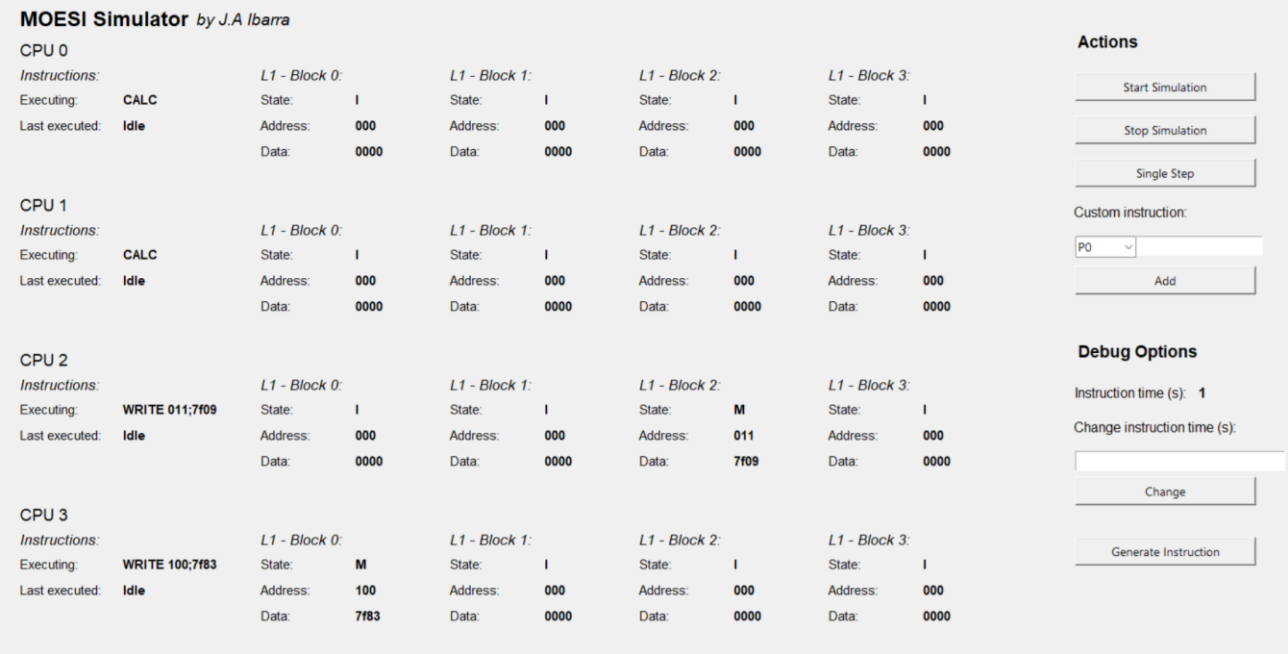
1. Introduction:

In this lab, we explored the operation of a multi-core cache system using the **MOESI** (Modified, Owned, Exclusive, Shared, Invalid) cache coherence protocol. The primary objective was to simulate and analyze how cache lines transition between states based on memory operations performed by four processors (P0–P3). We used a Python-based **MOESI** simulator (GitHub Repository) that randomly generates read and write instructions and visually shows the impact on each L1 cache.

The goal was to observe and explain how each instruction affects the cache line states across the processors, identifying the cause processor, transition type, and resulting coherence impact across the system. This hands-on experiment enhanced our understanding of distributed memory consistency and the detailed workings of MOESI transitions in shared-memory systems.

Example

Cycle 1

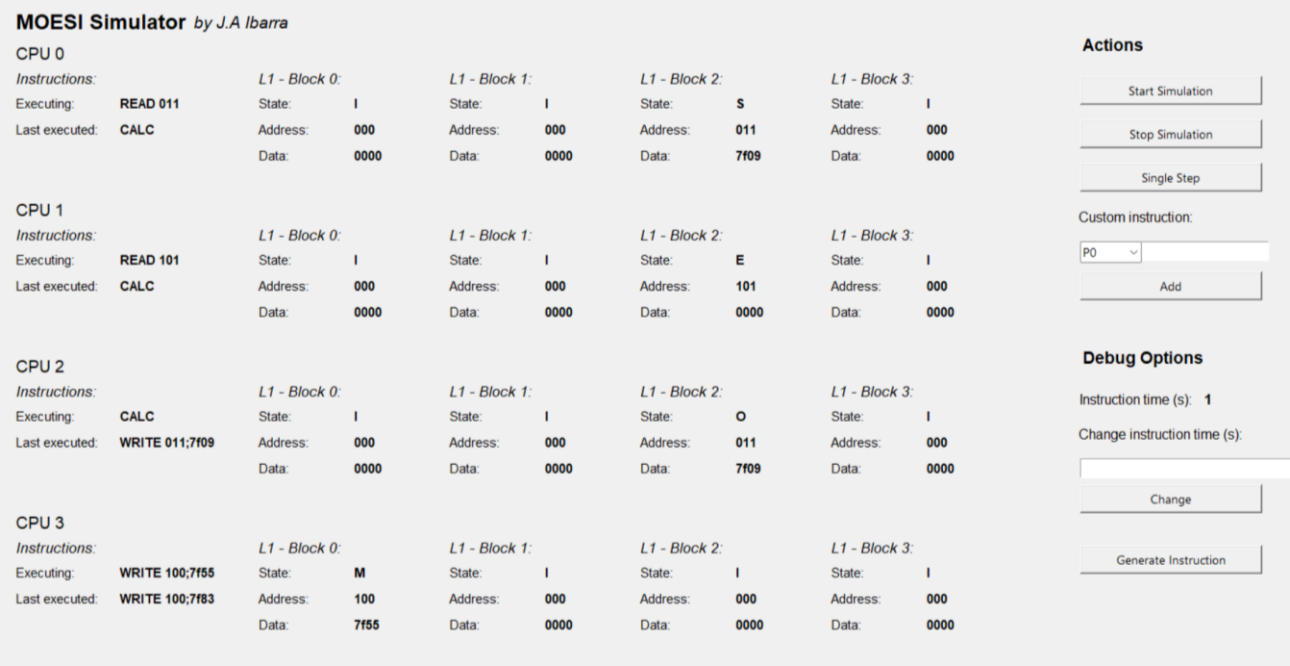



```
CPU 0: DECODED CALC

CPU 1: DECODED CALC

CPU 2: DECODED WRITE 3 32521

BUS: REQ FROM CPU 2: FOUND COPIES []

CPU 3: DECODED WRITE 4 32643

BUS: REQ FROM CPU 3: FOUND COPIES []
```

Instruction	Cause Processor	Transition	P0 Effect	P1 Effect	P2 Effect	P3 Effect
WRITE 011, 7F09	P2	I → M	-	-	-	-
WRITE 100, 7F83	P3	I → M	-	-	-	-

I → M (Invalid → Modified):
Happens when a processor writes to a memory address not currently in its cache (WRITE MISS), and no other caches have a copy.

Cycle 2




```
CPU 0: DECODED READ 3
CPU 0: READ MISS. PROBING...
BUS: REQ FROM CPU 0: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7F4C0>]
CPU 1: DECODED READ 5
CPU 0: READ HIT ON PROBE
CPU 1: READ MISS. PROBING...
BUS: REQ FROM CPU 0: UPDATING CPU: 2 ADDRESS: 3 TYPE: RHP
CPU 2: DECODED CALC
CPU 3: DECODED WRITE 4 32597
BUS: REQ FROM CPU 1: FOUND COPIES []
CPU 1: READ MISS ON PROBE. READ FROM MEMORY.
BUS: REQ FROM CPU 1: READING 5 FROM MEMORY
WRITE HIT
BUS: REQ FROM CPU 3: WRITING 32643 TO ADDRESS 4
```

Instruction	Cause Processor	Transition	P0 Effect	P1 Effect	P2 Effect	P3 Effect
READ 011	P0	I → S	-	-	I → O	-
READ 101	P1	I → E	-	-	-	-

I → S (Invalid → Shared) – on READ 011 by P0:

CPU 0 reads data it doesn't have (READ MISS), but a valid copy exists in CPU 2.

CPU 2 shares the data, changing its state to Owned (O).

CPU 0's cache line becomes Shared (S).

I → E (Invalid → Exclusive) – on READ 101 by P1:

CPU 1 reads data not present in any other cache (READ MISS, no copies).

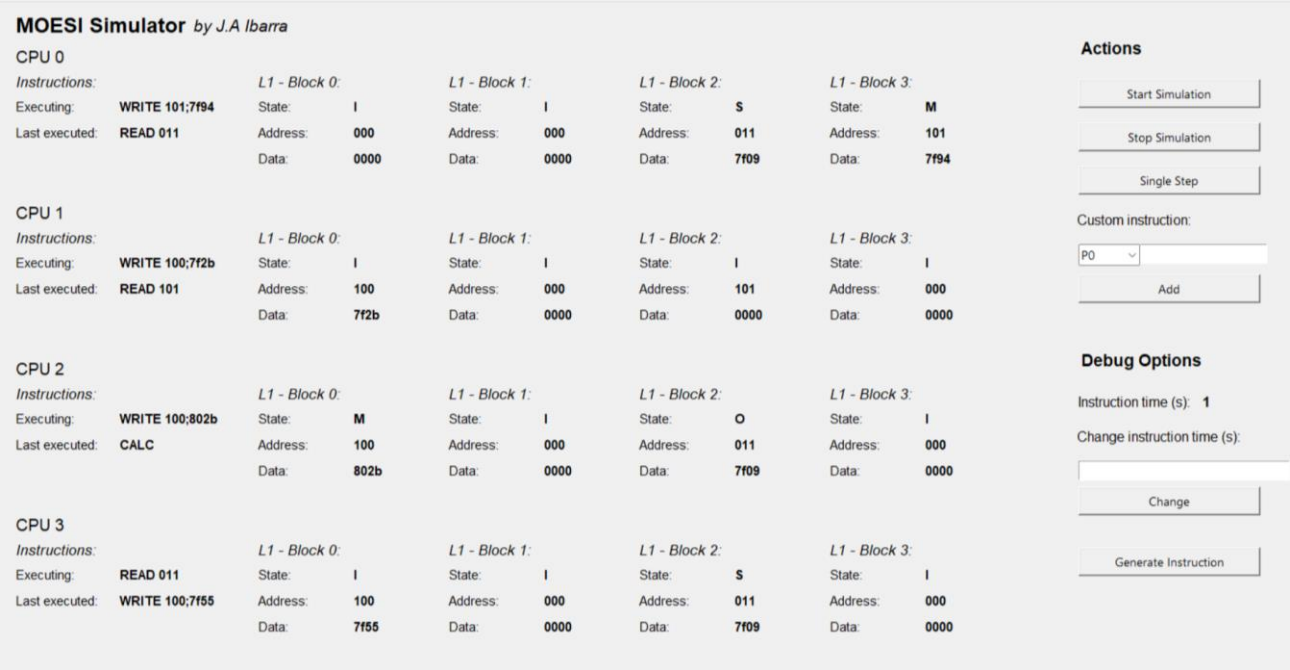
Since it’s the only copy, it gets the line in Exclusive (E) state.

WRITE 100, 7F55 by P3:

The transition is not shown, likely because it was a write hit (already had the line in M or E).

No state change across processors is noted.

Cycle 3




```
CPU 0: DECODED WRITE 5 32660

CPU 1: DECODED WRITE 4 32555

BUS: REQ FROM CPU 0: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7F010>]

CPU 2: DECODED WRITE 4 32811

BUS: REQ FROM CPU 0: UPDATING CPU: 1 ADDRESS: 5 TYPE: WM

CPU 3: DECODED READ 3

BUS: REQ FROM CPU 1: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7F730>]

CPU 3: READ MISS. PROBING...

BUS: REQ FROM CPU 1: UPDATING CPU: 3 ADDRESS: 4 TYPE: WM

BUS: REQ FROM CPU 2: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7F010>]

BUS: REQ FROM CPU 2: UPDATING CPU: 1 ADDRESS: 4 TYPE: WM

BUS: REQ FROM CPU 3: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7EF80>,
<CPU.CPU object at 0x0000021F84B7F4C0>]

CPU 3: READ HIT ON PROBE

BUS: REQ FROM CPU 3: UPDATING CPU: 0 ADDRESS: 3 TYPE: RHP

BUS: REQ FROM CPU 3: UPDATING CPU: 2 ADDRESS: 3 TYPE: RHP
```

Instruction	Cause Processor	Transition	P0 Effect	P1 Effect	P2 Effect	P3 Effect
WRITE 101, 7F94	P0	I → M	-	E → I	-	-
WRITE 100, 7F2B	P1	I → M	-	-	-	M → I
WRITE 100, 802B	P2	I → M	-	M → I	-	-
READ 011	P3	I → S	-	-	-	-

I → M (Invalid → Modified) – WRITE 101, 7F94 by P0:

CPU 0 writes to address 101; it doesn't have the line, so it moves from Invalid to Modified.

CPU 1 had the line in Exclusive (E), which is invalidated (E → I).

I → M – WRITE 100, 7F2B by P1:

CPU 1 writes a line it doesn’t have, so I → M.

CPU 3 had the line in Modified, which is now invalidated (M → I).

I → M – WRITE 100, 802B by P2:

CPU 2 writes a line it didn’t have; its state goes I → M.

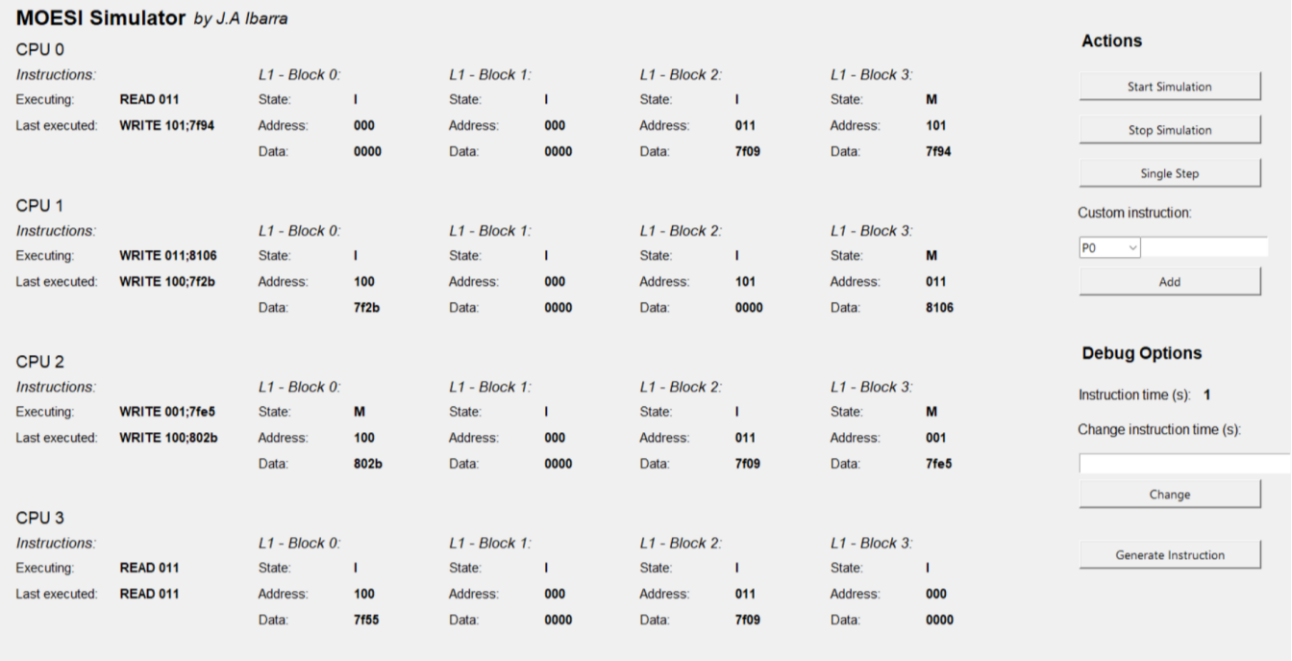
CPU 1 had the line in Modified, now invalidated (M → I).

I → S – READ 011 by P3:

CPU 3 reads a line it doesn’t have, so it becomes Shared (S).

Other processors may retain their states or adjust (e.g., from M/E → O or I).

Cycle 4




```
CPU 0: DECODED READ 3

CPU 0: READ HIT

CPU 1: DECODED WRITE 3 33030

CPU 2: DECODED WRITE 1 32741

BUS: REQ FROM CPU 1: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7EF80>,
<CPU.CPU object at 0x0000021F84B7F4C0>, <CPU.CPU object at
0x0000021F84B7F730>]

CPU 3: DECODED READ 3

BUS: REQ FROM CPU 1: UPDATING CPU: 0 ADDRESS: 3 TYPE: WM

CPU 3: READ HIT

BUS: REQ FROM CPU 1: UPDATING CPU: 2 ADDRESS: 3 TYPE: WM

BUS: REQ FROM CPU 1: UPDATING CPU: 3 ADDRESS: 3 TYPE: WM

BUS: REQ FROM CPU 2: FOUND COPIES []
```

Instruction	Cause Processor	Transition	P0 Effect	P1 Effect	P2 Effect	P3 Effect
READ 011	P0	-	-	-	-	-
WRITE 011, 8106	P1	I → M	S → I	-	O → I	S → I
WRITE 001, 7FE5	P2	I → M	-	-	-	-

READ 011 by P0:

No transition occurs (it’s a read hit), so the data is already in P0’s cache in a valid state like S or O.

WRITE 011, 8106 by P1:

CPU 1 performs a write, but others already have the line (P0, P2, P3).

P1: I → M – It didn’t have the line, now becomes the owner with updated data.

P0: S → I – Had a shared copy, now invalidated.

P2: O → I – Had an owned copy, now invalidated.

P3: S → I – Also invalidated.

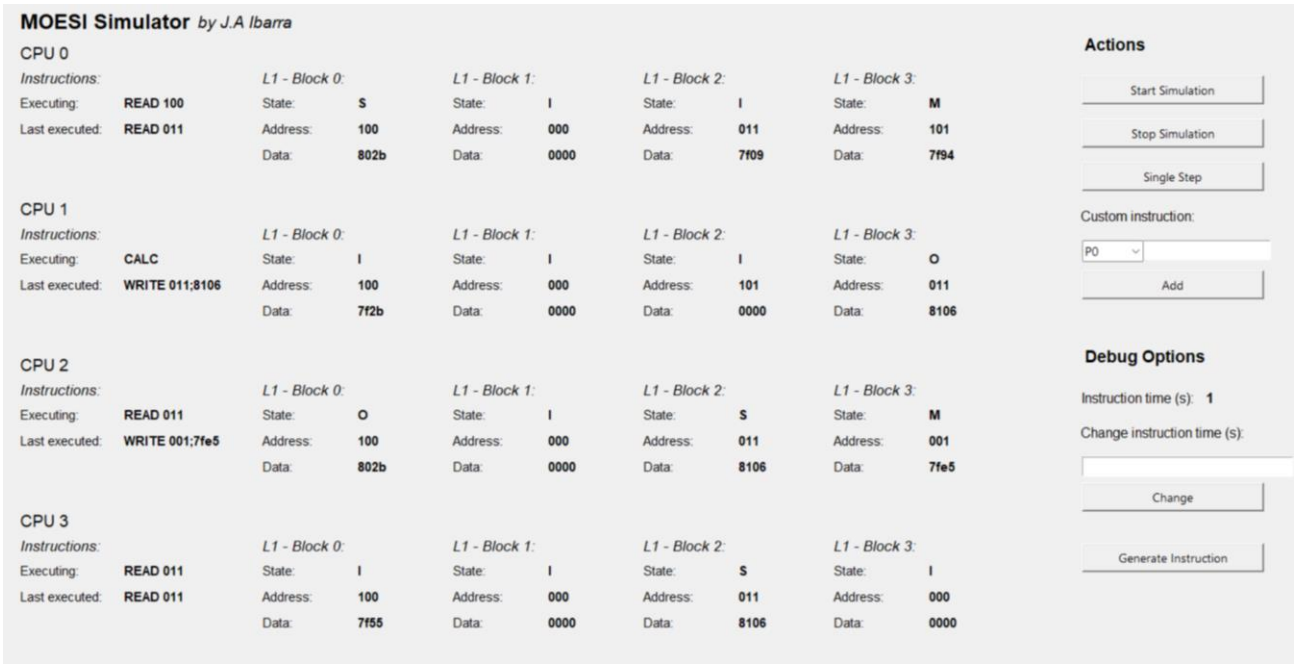
WRITE 001, 7FE5 by P2:

Simple case of a write miss with no existing copies.

P2: I → M – Becomes the only holder of the modified data.

Others remain unaffected.

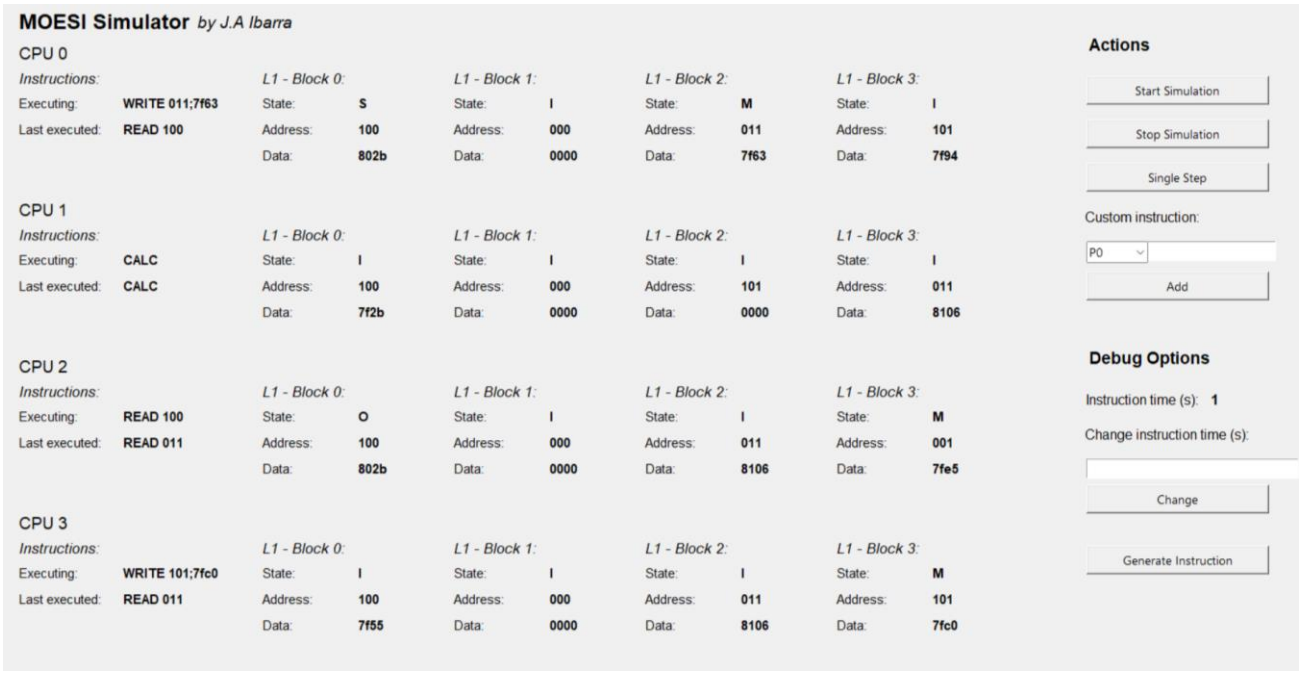
Cycle 5




```
CPU 0: DECODED READ 4

CPU 0: READ MISS. PROBING...

BUS: REQ FROM CPU 0: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7F4C0>]

CPU 0: READ HIT ON PROBE

CPU 1: DECODED CALC

CPU 2: DECODED READ 3

BUS: REQ FROM CPU 0: UPDATING CPU: 2 ADDRESS: 4 TYPE: RHP

CPU 2: READ MISS. PROBING...

CPU 3: DECODED READ 3

CPU 3: READ MISS. PROBING...

BUS: REQ FROM CPU 2: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7F010>]

CPU 2: READ HIT ON PROBE

BUS: REQ FROM CPU 2: UPDATING CPU: 1 ADDRESS: 3 TYPE: RHP

BUS: REQ FROM CPU 3: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7F010>,
<CPU.CPU object at 0x0000021F84B7F4C0>]

CPU 3: READ HIT ON PROBE

BUS: REQ FROM CPU 3: UPDATING CPU: 1 ADDRESS: 3 TYPE: RHP

BUS: REQ FROM CPU 3: UPDATING CPU: 2 ADDRESS: 3 TYPE: RHP
```

Instruction	Cause Processor	Transition	P0 Effect	P1 Effect	P2 Effect	P3 Effect
READ 100	P0	I → S	-	-	M → O	-
READ 011	P2	I → S	-	M → O	-	-
READ 011	P3	I → S	-	-	-	-

Cycle 6



CPU 0: DECODED WRITE 3 32611

BUS: REQ FROM CPU 0: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7F010>, <CPU.CPU object at 0x0000021F84B7F4C0>, <CPU.CPU object at 0x0000021F84B7F730>]

BUS: REQ FROM CPU 0: UPDATING CPU: 1 ADDRESS: 3 TYPE: WM

BUS: REQ FROM CPU 0: UPDATING CPU: 2 ADDRESS: 3 TYPE: WM

BUS: REQ FROM CPU 0: UPDATING CPU: 3 ADDRESS: 3 TYPE: WM

CPU 1: DECODED CALC

CPU 2: DECODED READ 4

CPU 2: READ HIT

CPU 3: DECODED WRITE 5 32704

BUS: REQ FROM CPU 3: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7EF80>]

BUS: REQ FROM CPU 3: UPDATING CPU: 0 ADDRESS: 5 TYPE: WM

Instruction	Cause Processor	Transition	P0 Effect	P1 Effect	P2 Effect	P3 Effect
WRITE 011, 7F63	P0	I → M	-	O → I	S → I	S → I
READ 100	P2	-	-	-	-	-
WRITE 101, 7FC0	P3	I → M	M → I	-	-	-

I → S – READ 100 by P0:

CPU 0 reads a line it doesn't have → becomes Shared (S).

CPU 2 had the line in Modified (M) → it downgrades to Owned (O), since it supplies the data.

I → S – READ 011 by P2:

CPU 2 reads a line it doesn't have → moves to Shared (S).

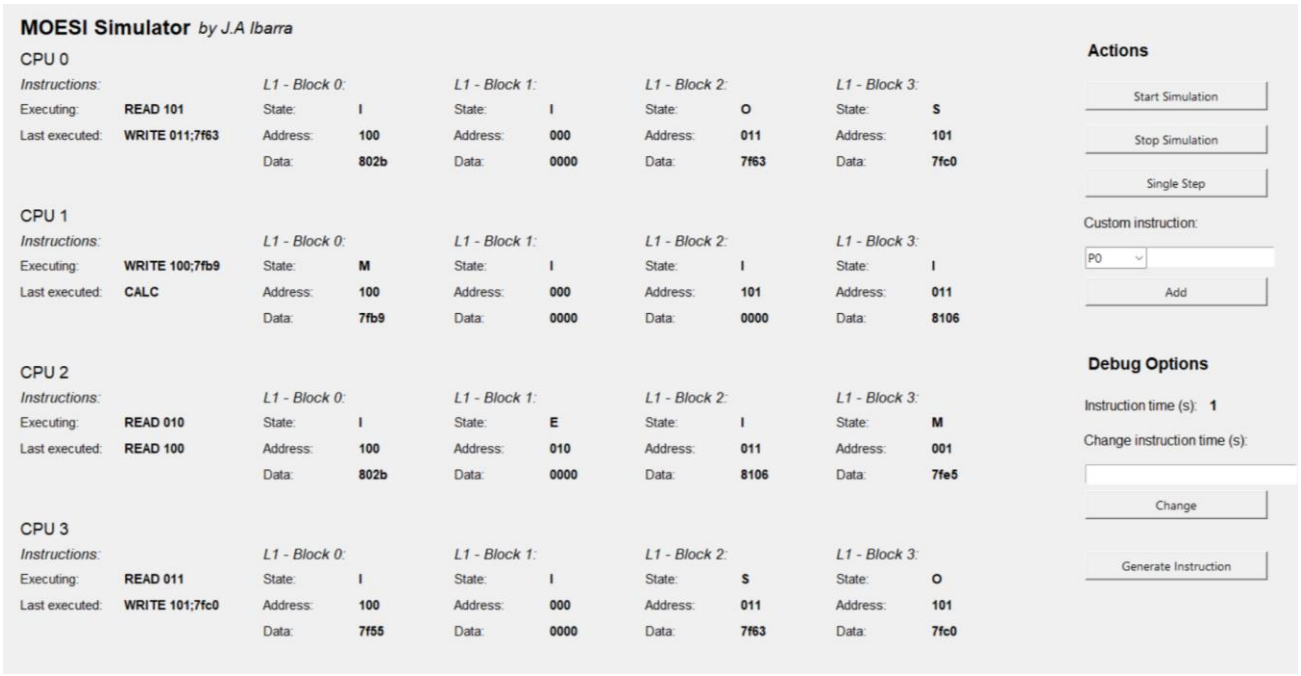
CPU 1 had the line in Modified (M) → downgrades to Owned (O).

I → S – READ 011 by P3:

CPU 3 also reads the same line and transitions to Shared.

No further state changes for others, as they already had the line in O or S.

Cycle 7




```
CPU 0: DECODED READ 5

CPU 0: READ MISS. PROBING...

BUS: REQ FROM CPU 0: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7F730>]

CPU 1: DECODED WRITE 4 32697

CPU 2: DECODED READ 2

CPU 0: READ HIT ON PROBE

CPU 2: READ MISS. PROBING...

BUS: REQ FROM CPU 0: UPDATING CPU: 3 ADDRESS: 5 TYPE: RHP

CPU 3: DECODED READ 3

BUS: REQ FROM CPU 1: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7EF80>,
<CPU.CPU object at 0x0000021F84B7F4C0>]

CPU 3: READ MISS. PROBING...

BUS: REQ FROM CPU 1: UPDATING CPU: 0 ADDRESS: 4 TYPE: WM

BUS: REQ FROM CPU 1: UPDATING CPU: 2 ADDRESS: 4 TYPE: WM

BUS: REQ FROM CPU 2: FOUND COPIES []

CPU 2: READ MISS ON PROBE. READ FROM MEMORY.

BUS: REQ FROM CPU 2: READING 2 FROM MEMORY

BUS: REQ FROM CPU 3: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7EF80>]

CPU 3: READ HIT ON PROBE

BUS: REQ FROM CPU 3: UPDATING CPU: 0 ADDRESS: 3 TYPE: RHP
```

Instruction	Cause Processor	Transition	P0 Effect	P1 Effect	P2 Effect	P3 Effect
READ 101	P0	I → S	-	-	-	M → O
WRITE 100, 7FB9	P1	I → M	-	-	O → I	-
READ 010	P2	I → E	-	-	-	-
READ 011	P3	I → S	M → O	-	-	-

I → S – READ 101 by P0:

CPU 0 reads a line it doesn’t have → transitions to Shared (S).

CPU 3 had the line in Modified (M) → downgrades to Owned (O) to supply the data.

I → M – WRITE 100, 7FB9 by P1:

CPU 1 writes a line it doesn’t have → transitions to Modified (M).

CPU 2 had it in Owned (O) → now invalidated (O → I).

I → E – READ 010 by P2:

CPU 2 reads a line with no other copies → moves to Exclusive (E).

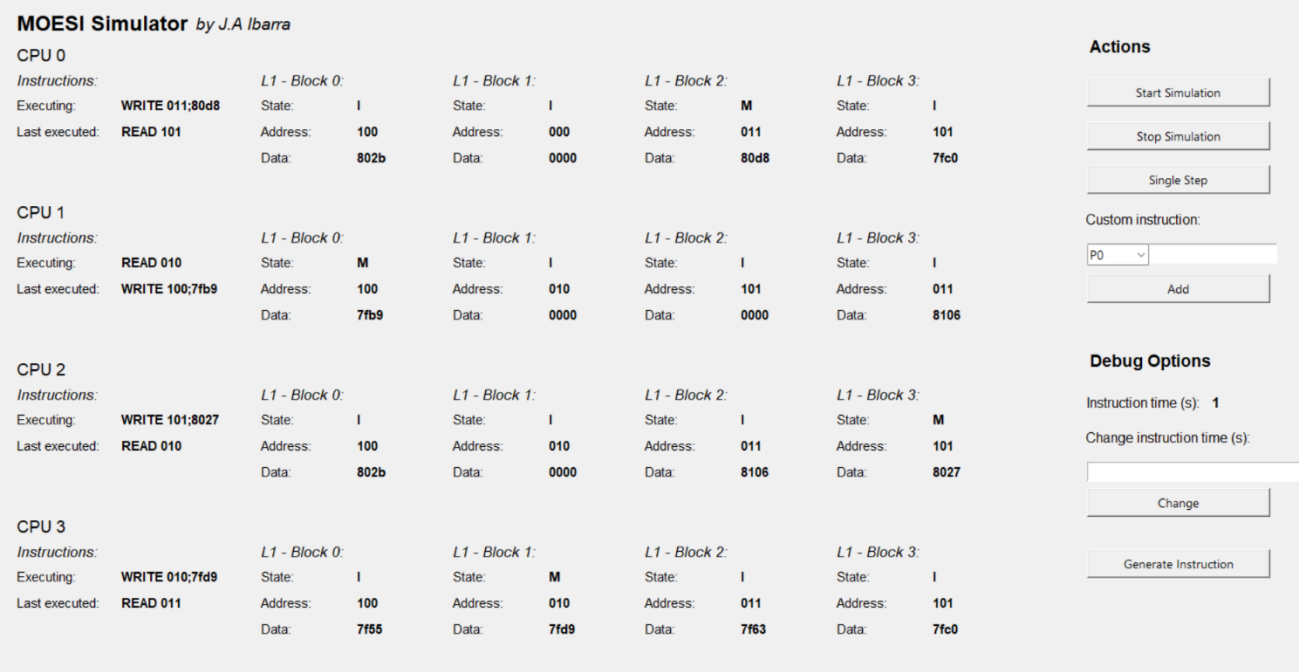
Since no one else has it, it can read and potentially write without notification.

I → S – READ 011 by P3:

CPU 3 reads a line it doesn’t have → moves to Shared (S).

CPU 0 had it in Modified (M) → becomes Owned (O) to provide the data.

Cycle 8




```
CPU 0: DECODED WRITE 3 32984

WRITE MISS

BUS: REQ FROM CPU 0: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7F730>]

CPU 1: DECODED READ 2

CPU 1: READ MISS. PROBING...

BUS: REQ FROM CPU 0: UPDATING CPU: 3 ADDRESS: 3 TYPE: WM

CPU 3: DECODED WRITE 2 32729

CPU 2: DECODED WRITE 5 32807

BUS: REQ FROM CPU 1: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7F4C0>]

CPU 1: READ HIT ON PROBE

BUS: REQ FROM CPU 1: UPDATING CPU: 2 ADDRESS: 2 TYPE: RHP

BUS: REQ FROM CPU 3: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7F010>,
<CPU.CPU object at 0x0000021F84B7F4C0>]

BUS: REQ FROM CPU 3: UPDATING CPU: 1 ADDRESS: 2 TYPE: WM

BUS: REQ FROM CPU 3: UPDATING CPU: 2 ADDRESS: 2 TYPE: WM

BUS: REQ FROM CPU 2: WRITING 32741 TO ADDRESS 1

BUS: REQ FROM CPU 2: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7EF80>,
<CPU.CPU object at 0x0000021F84B7F730>]

BUS: REQ FROM CPU 2: UPDATING CPU: 0 ADDRESS: 5 TYPE: WM

BUS: REQ FROM CPU 2: UPDATING CPU: 3 ADDRESS: 5 TYPE: WM
```

Instruction	Cause Processor	Transition	P0 Effect	P1 Effect	P2 Effect	P3 Effect
WRITE 011, 80D8	P0	O → M	-	-	-	S → I
READ 010	P1	I → S	-	-	E → O	-
WRITE 101, 8027	P2	I → M	S → I	-	-	O → I
WRITE 010, 7FD9	P3	I → M	-	S → I	O → I	-

O → M – WRITE 011 by P0:

CPU 0 has the line in Owned (O) and writes → transitions to Modified (M).

CPU 3 had a Shared (S) copy → must invalidate (S → I).

I → S – READ 010 by P1:

CPU 1 reads a line it doesn't have → moves to Shared (S).

CPU 2 had it in Exclusive (E) → downgrades to Owned (O) to share it.

I → M – WRITE 101 by P2:

CPU 2 writes a line it doesn't have → transitions to Modified (M).

CPU 0 had it in Shared (S) → invalidated.

CPU 3 had it in Owned (O) → invalidated.

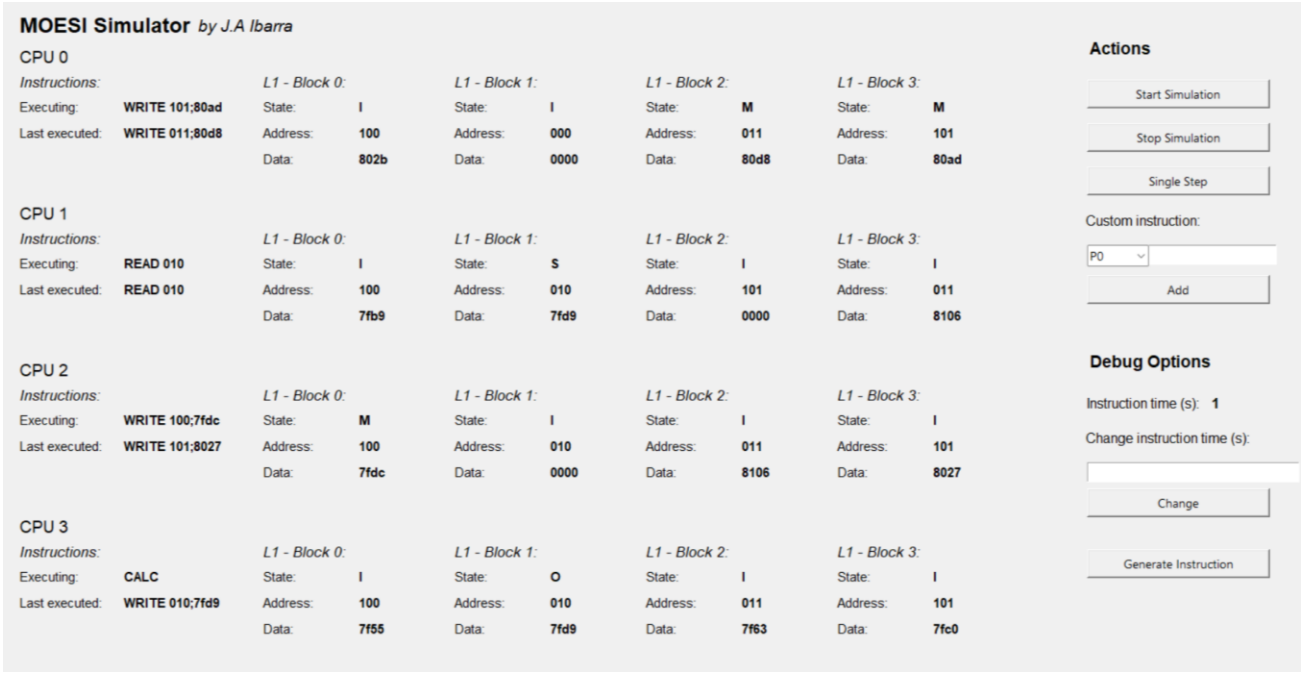
I → M – WRITE 010 by P3:

CPU 3 writes a line it doesn't have → transitions to Modified (M).

CPU 1 had it in Shared (S) → invalidated.

CPU 2 had it in Owned (O) → invalidated.

Cycle 9




```
CPU 0: DECODED WRITE 5 32941
BUS: REQ FROM CPU 0: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7F4C0>]
CPU 1: DECODED READ 2
BUS: REQ FROM CPU 0: UPDATING CPU: 2 ADDRESS: 5 TYPE: WM
CPU 1: READ MISS. PROBING...
BUS: REQ FROM CPU 1: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7F730>]
CPU 1: READ HIT ON PROBE
BUS: REQ FROM CPU 1: UPDATING CPU: 3 ADDRESS: 2 TYPE: RHP
CPU 2: DECODED WRITE 4 32732
BUS: REQ FROM CPU 2: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7F010>]
CPU 3: DECODED CALC
BUS: REQ FROM CPU 2: UPDATING CPU: 1 ADDRESS: 4 TYPE: WM
```

Instruction	Cause Processor	Transition	P0 Effect	P1 Effect	P2 Effect	P3 Effect
WRITE 101, 80AD	P0	I → M	-	-	M → I	-
READ 010	P1	I → S	-	-	-	M → O
WRITE 100, 7FDC	P2	I → M	-	M → I	-	-

WRITE 101, 80AD – by P0 (CPU 0):

CPU 0 writes to address 5.

Its local state was Invalid (I) → moves to Modified (M).

CPU 2 had the line in Modified (M) → must invalidate (M → I).

READ 010 – by P1 (CPU 1):

CPU 1 reads address 2, and didn’t have the line → I → Shared (S).

CPU 3 had it in Modified (M) → must downgrade to Owned (M → O) to allow sharing.

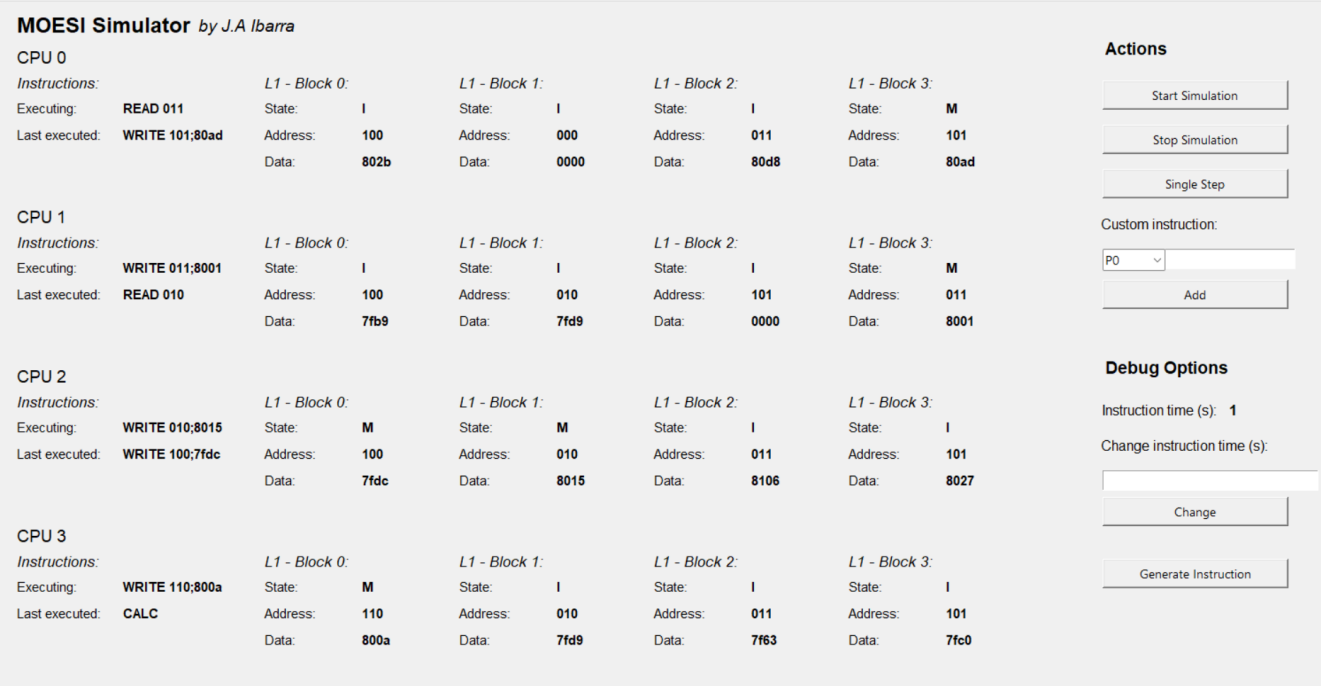
WRITE 100, 7FDC – by P2 (CPU 2):

CPU 2 writes to address 4.

It had the line as Invalid (I) → becomes Modified (M).

CPU 1 had it in Modified (M) → must invalidate (M → I).

Cycle 10




```
CPU 0: DECODED READ 3

CPU 0: READ HIT

CPU 1: DECODED WRITE 3 32769

BUS: REQ FROM CPU 1: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7EF80>]

BUS: REQ FROM CPU 1: UPDATING CPU: 0 ADDRESS: 3 TYPE: WM

CPU 2: DECODED WRITE 2 32789

BUS: REQ FROM CPU 2: FOUND COPIES [<CPU.CPU object at 0x0000021F84B7F010>,
<CPU.CPU object at 0x0000021F84B7F730>]

BUS: REQ FROM CPU 2: UPDATING CPU: 1 ADDRESS: 2 TYPE: WM

BUS: REQ FROM CPU 2: UPDATING CPU: 3 ADDRESS: 2 TYPE: WM

CPU 3: DECODED WRITE 6 32778

BUS: REQ FROM CPU 3: FOUND COPIES []
```

Instruction	Cause Processor	Transition	P0 Effect	P1 Effect	P2 Effect	P3 Effect
WRITE 011, 8001	P1	I → M	M → I	-	-	-
WRITE 010, 8015	P2	I → M	-	S → I	-	O → I
WRITE 110, 800A	P3	I → M	-	-	-	-

WRITE 011, 8001 – by P1 (CPU 1):

CPU 1 writes to address 3.

Its local cache line was Invalid (I) → becomes Modified (M).

CPU 0 had the line in Modified (M) → must invalidate (M → I) to ensure only one valid copy exists.

WRITE 010, 8015 – by P2 (CPU 2):

CPU 2 writes to address 2.

Its local state was Invalid (I) → moves to Modified (M).

CPU 1 had the line in Shared (S) → must invalidate (S → I).

CPU 3 had it in Owned (O) → must also invalidate (O → I).

WRITE 110, 800A – by P3 (CPU 3):

CPU 3 writes to address 6.

No other processors have this address cached.

So it transitions from Invalid (I) to Modified (M) with no other effects.

1. Simulation Results:

The simulation results demonstrate how cache coherence is maintained across different processors using the MOESI (Modified, Owned, Exclusive, Shared, Invalid) protocol. Each state plays a crucial role in ensuring data consistency:

M - Modified: The most updated version of the line is in this cache. No copies exist in any other cache. This data is different from the memory version.

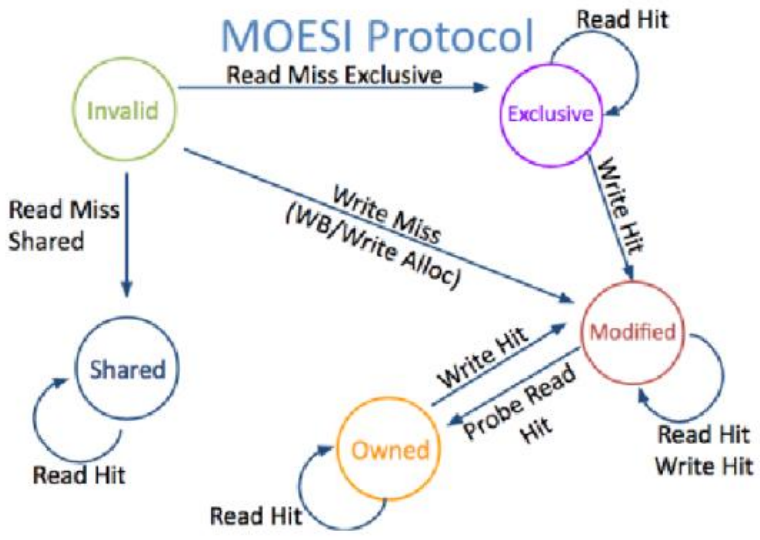
O - Owned: This cache line can exist in more than one cache. However, it contains the most updated version of the data. Similar to M, only the line in one core can be in this state; the others must be in state S.

E - Exclusive: This cache line has the same data as the main memory. No copies exist in any other cache. This occurs when it is read directly.

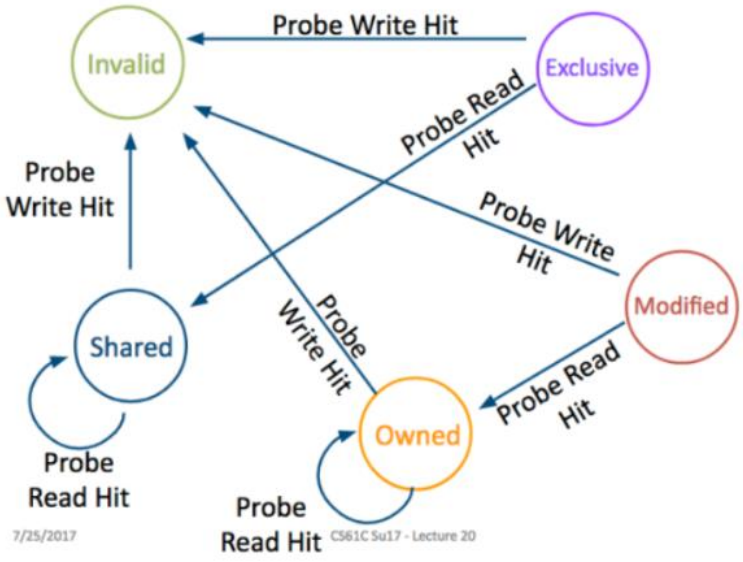
S - Shared: The line in this state is coherent with main memory. Copies may exist in other caches. The data is shared with other processors.

I - Invalid: The line is invalid. It follows several rules:

1. If a **write** is made and the line is not in state **M** or **E**, all copies must be invalidated, as they do not have the most updated data.
2. The system may discard an invalid line to make space for one that contains another address, depending on the replacement policy for **cache lines**.
3. If a line is in **M** state, reads from other caches obtain the updated data, and the one that shared it changes to **O** (Owned).
4. If a line is in **E** state, reads from other caches change to **S**, and the one that shared it changes to **S** (Shared).



State machine of a cache line for the MOESI protocol



Bus state machine for the MOESI protocol.

2. Tools and Environment:

MOESI simulator was used for simulation.

3. Conclusion:

Through this lab, we gained practical insight into how the MOESI cache coherence protocol ensures memory consistency across multiple processor cores. By simulating various memory operations across four CPUs, we observed how each instruction can cause a series of state transitions across the L1 caches, such as transitions from Invalid to Shared, Shared to Invalid, or Owned to Modified, depending on the access type and current cache states.

The step-by-step simulation highlighted the importance of bus-based communication in enforcing coherence and the impact of one processor's actions on the cache states of others. For instance, a write by one processor often resulted in invalidation of shared copies held by others, demonstrating the trade-offs between performance and consistency in cache coherence protocols.

Overall, this exercise solidified our theoretical understanding of the MOESI protocol by visualizing real-time state transitions and provided valuable experience in interpreting system-level cache interactions in multiprocessor systems.