Name: Gunavant Setty **Student ID: 200597215**

CSC/ECE – 506

I. MESI (Modified, Exclusive, Shared, Invalid) protocol

The MESI protocol introduces four distinct states for cache lines, allowing for more efficient handling of data. The **MESI (Modified, Exclusive, Shared, Invalid)** protocol improves upon the **MSI (Modified, Shared, Invalid)** protocol by introducing the **Exclusive (E)** state, which offers several key optimizations.

1. **Reduced Bus Transactions:** MESI significantly reduces unnecessary bus traffic, particularly for read-write sequences. In MSI, each read-write sequence requires two bus transactions: a read followed by an upgrade to modified state. MESI eliminates this overhead by allowing direct transition from Exclusive to Modified without additional bus traffic.
2. **Optimized Private Data Handling:** The Exclusive state is particularly beneficial for scenarios with little data sharing. When a processor reads data not present in other caches, it enters the Exclusive state. Subsequent writes to this data can occur without invalidation messages, reducing coherence traffic.
3. **Efficient Cache-to-Cache Transfers:** MESI enables more efficient data sharing by distinguishing between data stored in multiple caches and data exclusive to one cache. This allows for optimized data sharing and reduced memory accesses.

II. MOESI (Modified, Owned, Exclusive, Shared, Invalid) protocol:

The MOESI protocol introduces five distinct states for cache lines, allowing for more efficient handling of data when compared to MESI.

1. **Modified (M)**: Indicates that the cache line has been modified and is the only up-to-date copy in any cache. Data in this state is dirty and needs to be written back to main memory when evicted or requested by another processor.
2. **Owned (O)**: State is the key innovation of MOESI over MESI. In this state, the cache line is potentially shared but dirty. The cache with the Owned copy is responsible for writing back to memory and can supply data directly to other caches without accessing main memory.
3. **Exclusive (E):** State is the only copy in the cache hierarchy and is clean, matching the data in main memory. It can be modified without notifying other caches, allowing for efficient write operations.
4. **Shared (S):** In the Shared (S) state, multiple caches may have a copy of the data, which is clean and matches main memory. This state allows for read-only access without requiring further coherence actions, enabling efficient sharing of read-only data across multiple processors.
5. **Invalid (I):** An Invalid (I) cache line does not hold valid data. When a processor needs to access data in this state, it must fetch it from memory or another cache before use, ensuring that only up-to-date data is accessed.

III. DETAILS OF IMPLEMENTATION

MESI Implementation:

1. **Read/Write Operations:** Reads and writes are counted at the beginning of MESI_Processor_Access. The function handles both hit and miss scenarios for reads and writes.
2. **Cache Hit Handling:** Read hits increment the Readhits counter and add read_hit_latency to Total_execution_time. Write hits increment the Writehits counter and add write_hit_latency to Total_execution_time.
    - **Cache Miss Handling:** Read misses increment readMisses and trigger appropriate coherence actions. Write misses increment writeMisses and may cause invalidations in other caches.
    - **Memory Transactions:** mem_trans is incremented for read misses from memory and write misses that fetch data from memory. Memory accesses add memory_latency to Total_execution_time.
    - **Cache-to-Cache Transfers:** c_to_c_trans is incremented for write misses when data is copied from another cache. These transfers add flush_transfer latency to Total_execution_time.
    - **Coherence Operations:** Invalidations are tracked in MESI_Bus_Snoop when a cache line is invalidated. Flushes are counted when a cache line is flushed to maintain coherence.

MOESI Implementation:

The MOESI implementation follows a similar pattern to MESI, with additional handling for the Owner state:

1. **Read/Write Counters:** Incremented similarly to MESI**.**
2. **Hit/Miss Counters:** Handled in the same way as MESI.
3. **Memory Transactions:** Incremented on read and write misses from memory.
4. **Cache-to-Cache Transfers:** Incremented on write misses with data from another cache.
5. **Invalidations and Flushes:** Invalidations are incremented in MOESI_Bus_Snoop when a cache line is invalidated. Flushes are incremented in MOESI_Bus_Snoop when a cache line is flushed, including transitions to the Owner state.

Both implementations update the **Total_execution_time** based on the type of operation (read hit, write hit, memory access, cache-to-cache transfer) and their respective latencies.