DCACHE MAS

Based on your updated requirements:

1. **Cache Size:** 2 KB=2048 bytes
2. **Cache Line Size (Block Size):** 128 bits=16 bytes
3. **Associativity:** 4-way set-associative
4. **CPU Access:** Read/Write 32-bit words.

Let's re-derive the parameters and address breakdown:

- **Number of Cache Lines:** 16 bytes/line2048 bytes=128 lines
- **Number of Sets:** 4 ways128 lines=32 sets

Explanation of FSM States:

The Finite State Machine (FSM) controls the overall behavior of the cache, managing transitions between different operations like looking up data, fetching from memory, or writing back dirty lines.

`IDLE` State:

- **Purpose:** This is the default state where the cache is waiting for a request from the CPU.
- **Behavior:**
    - `cpu_ready` is asserted, indicating the cache is available.
    - If `cpu_req` is asserted by the CPU, the cache captures the `cpu_addr`, `cpu_we` (write enable), and `cpu_wdata` (if it's a write).
    - **Transition:** Moves to the `LOOKUP` state to process the request.

`LOOKUP` State:

- **Purpose:** In this state, the cache performs the core logic of checking for a cache hit or miss. It compares the tag of the requested address with the tags stored in the cache memory for the corresponding set. It also identifies if there's an invalid way or the Least Recently Used (LRU) way.
- **Behavior:**
    - The `cpu_addr` is decomposed into its Tag, Index, and Offset components.
    - For the given `req_idx` (set index), the tags of all 4 ways are compared with `req_tag`.
    - **If Hit (`is_hit` is true):**
        - `cpu_hit` is asserted, `cpu_miss` is de-asserted.

        - `cpu_valid` is asserted, and `cpu_rdata` is driven with the data from the hit cache line.
        - The LRU counter for the hit way is reset to 0 (most recently used), and other valid ways in the set have their LRU counters incremented.
        - If the original request was a write (`req_we_reg` is true), the `req_wdata_reg` is written to the cache line, and its `cache_dirty` bit is set.
        - **Transition:** Returns to `IDLE`.
    - **If Miss (`is_hit` is false):**
        - `cpu_hit` is de-asserted, `cpu_miss` is asserted.
        - The cache checks if there's an `invalid_way` available in the set.
        - If no `invalid_way` is found (all ways are valid), it identifies the `lru_way` (the way with the highest LRU counter).
        - **If `lru_way` is `dirty` (`is_lru_dirty` is true):** The cache needs to write this dirty line back to main memory before fetching the new line.
            - **Transition:** Moves to `WRITE_BACK`.
        - **If `lru_way` is `clean` (or if an `invalid_way` was found):** No write-back is needed. The cache can directly proceed to fetch the required line from main memory.
            - **Transition:** Moves to `FETCH_LINE`.

`WRITE_BACK` State:

- **Purpose:** This state is entered when a cache miss occurs and the Least Recently Used (LRU) cache line that needs to be evicted is `dirty` (meaning it has been modified in the cache but not yet written back to main memory).
- **Behavior:**
    - `mem_req` and `mem_we` are asserted to initiate a write operation to main memory.
    - `mem_addr` is set to the full address of the evicted line (Tag + Index + Offset bits all zero).
    - `mem_wdata` is set to the `evicted_data`.
    - **Transition:** Moves to `WRITE_BACK_WAIT`.

`WRITE_BACK_WAIT` State:

- **Purpose:** The cache waits here for the main memory to acknowledge that it has accepted the write-back data.
- **Behavior:**
    - Keeps `mem_req` and `mem_we` asserted.
    - **If `mem_ready` is asserted by main memory:** The write-back is complete.
        - `mem_req` is de-asserted.

        - **Transition:** Moves to `FETCH_LINE` to now fetch the line that caused the original miss.

`FETCH_LINE` State:

- **Purpose:** This state is entered after a cache miss (either directly if the evicted line was clean/invalid, or after a `WRITE_BACK_WAIT` if the evicted line was dirty). The cache requests the required data line from main memory.
- **Behavior:**
    - `mem_req` is asserted to initiate a read operation from main memory.
    - `mem_we` is de-asserted (for a read).
    - `mem_addr` is set to the full address of the requested line.
    - **Transition:** Moves to `FETCH_WAIT`.

`FETCH_WAIT` State:

- **Purpose:** The cache waits here for the main memory to provide the requested data line.
- **Behavior:**
    - Keeps `mem_req` asserted.
    - **If `mem_valid` is asserted by main memory:** The data is available on `mem_rdata`.
        - `mem_req` is de-asserted.
        - **Transition:** Moves to `REFILL`.

`REFILL` State:

- **Purpose:** This state is responsible for writing the newly fetched data line from main memory into the cache.
- **Behavior:**
    - The `mem_rdata` is written into the selected cache way (either the previously identified `invalid_way` or the `lru_way`).
    - The corresponding `cache_tag` is updated, and the `cache_valid` bit is set.
    - The LRU counter for this newly filled way is reset to 0, and other valid ways in the set have their LRU counters incremented.
    - **Write-Allocate Logic:** If the original CPU request was a write (`req_we_reg` was true), the `req_wdata_reg` is immediately applied to the newly refilled line, and its `cache_dirty` bit is set. If it was a read, the dirty bit is cleared.
    - `cpu_valid` is asserted, and `cpu_rdata` is driven with the fetched data (or the modified data if it was a write-allocate).
    - **Transition:** Returns to `IDLE`, ready for the next CPU request.