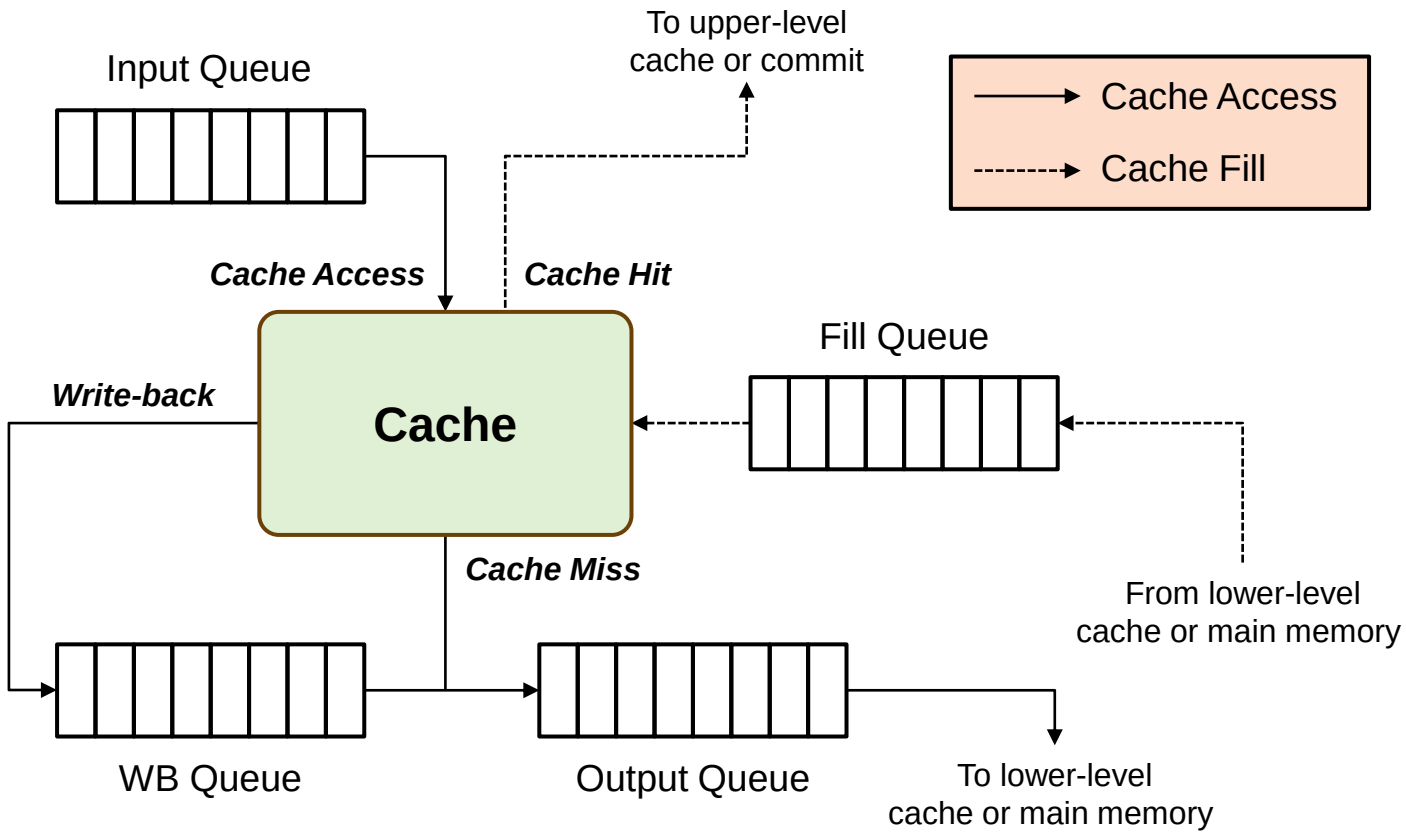
Computer Architecture

Lab 4: Memory System Simulation

Jaewoong Sim
Electrical and Computer Engineering
Seoul National University

Lab Overview



Cache Unit (cache.cc)

- The cache unit (`cache_c`) inherits from the `cache_base_c` class and implements the latency and interfaces.

- To do so, the cache unit has four memory queues to deal with different cache flows. There are basically two cache flows.

  1. **Cache Access**: This is from a processor or from the upper level cache (due to misses) to access the cache to see if it is a hit or a miss.
  2. **Cache Fill**: This is the flow where a cache line is filled into the cache; i.e., returned requested data from the lower-level cache or write-back data from upper-level cache.

- `cache_c::access()` and `cache_c::fill()` are the functions corresponding to each flow. These two will take effect after the intrinsic cache latency, and these are the **only functions** that need to add **m_latency (cache intrinsic latency)** to the request.

The Queues

- Input Queue
  - The input queue holds the memory requests to look up in the cache to determine a cache hit or a cache miss.
  - The request that comes from a processor or the upper-level cache (due to a miss) is inserted into this queue.

- Output Queue
  - The requests that `miss` in the cache are inserted into the output queue to be forwarded to the lower-level cache.
  - If no lower-level cache is available, the requests are forwarded to the memory controller (main memory).

The Queues

- Write-Back Queue
  - When a dirty cache line is evicted, it must be written back to the next level of memory hierarchy.
  - All write-back requests are initially inserted into this queue.

- Fill Queue
  - The requests in this queue are to be filled into the cache, and we have two such cases.
    1. The data returned from the lower-level cache or main memory is inserted into this queue.
    2. The write-back requests from the upper-level cache are also inserted into this queue.
  - Note: The write-back data **does not** change the LRU stack.

- Processing The Queues
  - The requests in each queue are processed by the `process_xxx_queue` functions.

Others

- In the memory hierarchy, there are also two data structures for in-flight (i.e., issued but not yet committed) and committed memory requests, `m_in_flight_reqs` and `m_done_queue`, respectively.

- The core should call `run_a_cycle` until all the issued requests are committed to the memory system.
- The write-back requests are created and issued from the caches (not from the core).
- Thus, even though all the requests that are issued from the core are done, there could be some in-flight write-back requests in the memory hierarchy.
- To handle this, there is `m_in_flight_wb_queue` for each cache or main memory to check if all the write-back requests are committed to the memory system.

The Flows

All possible flows from or to the cache are listed below.

[1] • **Upper-level cache => Input queue**: Forwarding of upper-level cache misses to the cache.

[2] • **Upper-level cache => Fill queue**: Forwarding of upper-level write-back requests to the cache.

[3] • **Lower-level cache or main memory => Fill queue**: Filling the cache with the data from the lower-level cache or main memory.

[4] • **Input queue => Cache**: Accessing the cache.

[5] • **Cache => Output queue**: Cache miss, access to the lower-level cache.

[6] • **Write-back queue => Output queue**: Write-back requests.

MISC

Memory Request (`mem_req_s`)

- In a write-back cache, if a STORE request misses in L1, it becomes a READ request for the lower-level cache access. Then, when we get the data from the lower-level memory, we fill the data into the L1 cache, and we update the dirty flag.

Terminology

- When the memory system has two-level caches (L1 and L2):
  - Upper-Level Cache : L1$
  - Lower-Level Cache : L2$