Caching

Prof. James L. Frankel
Harvard University

Version of 5:56 PM 19-Nov-2024

Memory Hierarchy

- Extremely limited number of registers in CPU

- Lots of medium speed, medium price main memory
- Terabytes of slow, cheap disk storage

Where Does the Cache Fit In?

- Extremely limited number of registers in CPU
- Small amount of fast, expensive memory – caches
- Lots of medium speed, medium price main memory
- Terabytes of slow, cheap disk storage

- The cache takes advantage of locality of reference to make the main memory appear to be faster

What is Locality of Reference?

- Future memory references (reads and writes) made by computer programs are more likely to be:
  - For locations previously referenced or
  - For locations near those previously referenced

Taking Advantage of Locality of Reference

- Add a relatively small amount (compared to the main memory) of fast and expensive memory that will be specially managed
- We will call this memory a **cache**
- Once memory has been referenced, the cache is loaded with a copy of the value in that memory location associated with its location in memory (the **tag** for that cache entry)
- Future references to that location are made to the value in the cache

Cache Access Terminology

- Cache **Miss**
  - An access to a location in memory that is *not* in the cache
- Cache **Hit**
  - An access to a location in memory that *is* in the cache
- The proportion of cache hits to total memory accesses in called the **hit rate**

Management of the Cache

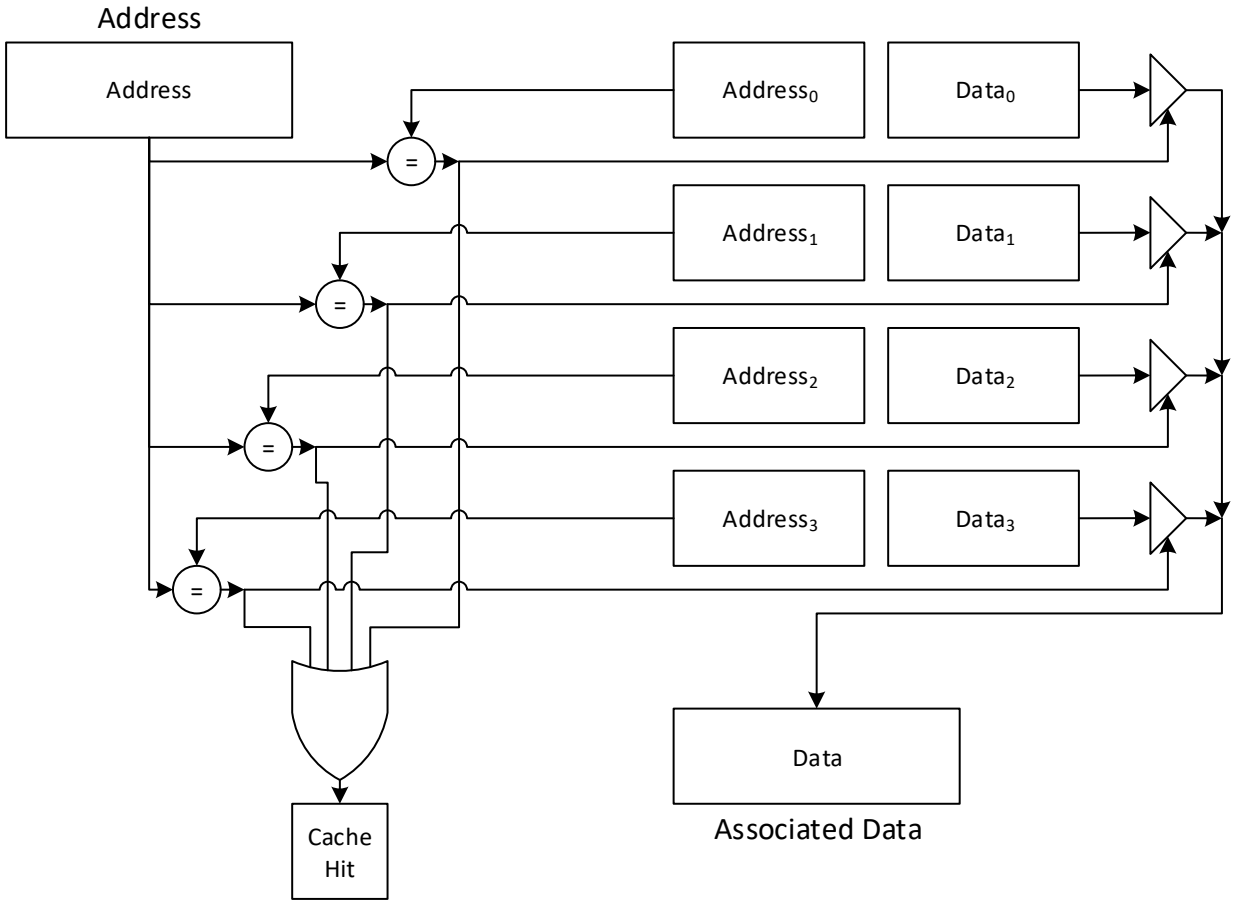
- The cache is limited in size
  - Not all memory locations can be kept in the cache
  - An algorithm needs to be used to determine which locations are kept in the cache and which are ejected once the cache becomes full
    - This is called the cache entry **replacement policy**
    - Often, **Least-Recently Used (LRU)** is used as the replacement policy

- On a cache miss, the access is added into the cache
  - Except if the location is **non-cacheable**
    - For example, memory-mapped I/O locations would be non-cacheable
    - Accesses to non-cacheable locations always cause a main memory access

Cache Implementation

- The cache is implemented using a CAM (Content-Addressable Memory) also known of as an Associative Memory
- Cache misses must be handled by hardware
  - Misses must be handled quickly

Simplified Fully-Associative Cache Schematic



Simplified Cache Observations

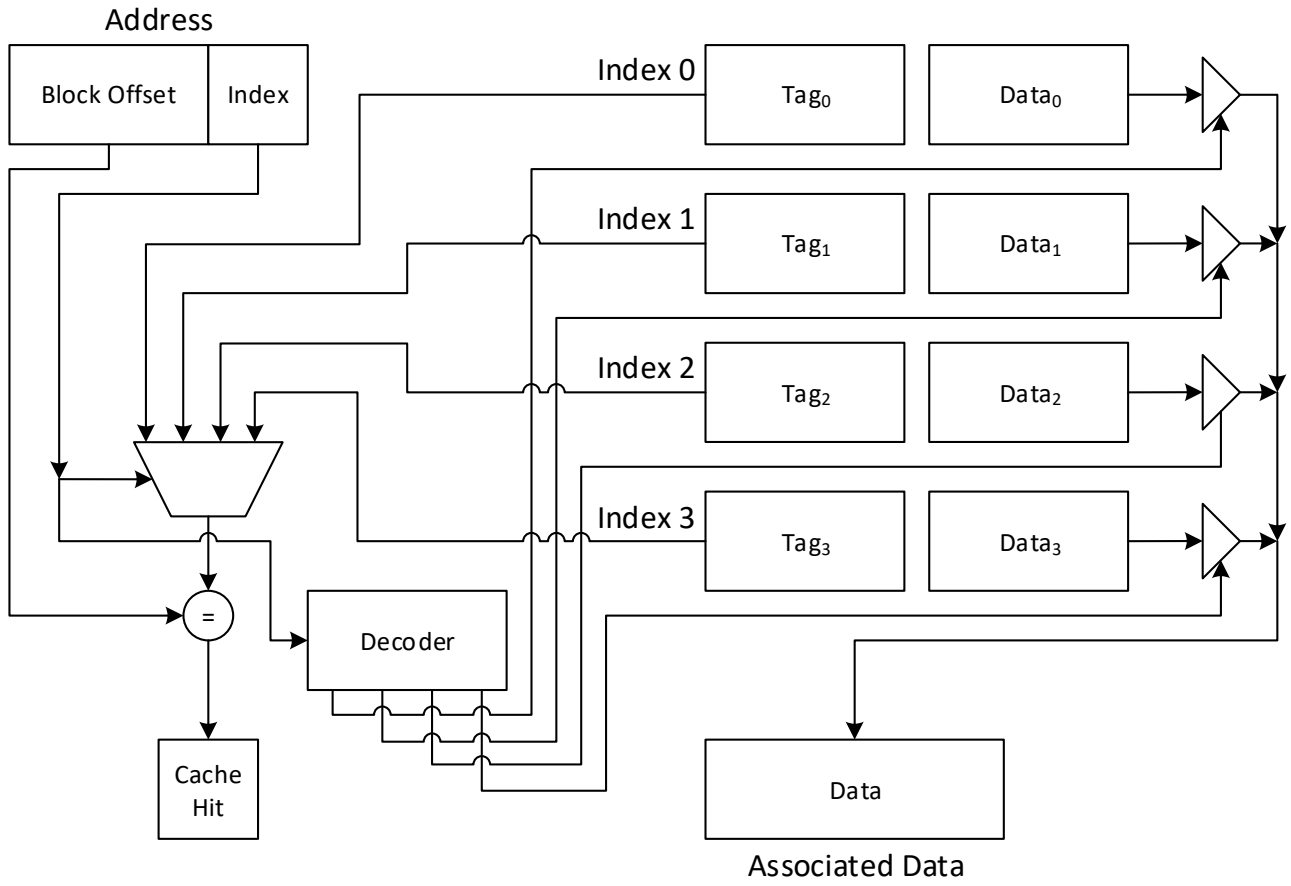
- The previous diagram doesn’t show how main memory is accessed on a cache miss
- The diagram doesn’t show how writes are handled
- The diagram doesn’t show how the replacement policy is implemented
- The diagram doesn’t show how the granularity of the address and the size of the cached data are handled (*i.e.,* all of the address may not be held in the cache Address entries)
- The diagram doesn’t show how cache line validity is handled

Fully-Associative vs. Direct-Mapped Cache

- A **fully-associative cache** allows an address and its data to be placed in *any* cache entry

- A **direct-mapped cache** allows an address and its data to be placed in *only one* cache entry as determined by the most-significant bits of the address

Simplified Direct-Mapped Cache Schematic



Cache Comparison

- A fully-associative cache is most flexible, but requires one comparator for each cache line

- A direct-mapped cache is least flexible and requires only one comparator for the whole cache
  - If there is a match on the MSBs of the main memory address – even though that cache line contains an entry from a different main memory address – that entry will be ejected
  - That is, two main memory addresses that match in their MSBs can never reside in the cache at the same time

Set Associative Caches

- As a compromise between fully-associative and direct-mapped caches, we can also build **set associative** caches

- A **two-way set associative cache** allows any main memory address to be placed in one of two different cache lines
- A **four-way set associative cache** allows any main memory address to be place in one of four different cache lines

Writes to Locations in the Cache

- When a write occurs to a location that is in the cache, in addition to updating the data in the cache, the location in main memory needs to be updated, too
  - If writes to the cache immediately write to the main memory, this mechanism is called a **write-through cache**
  - If writes to the cache do not write the altered data to main memory until that cache line is ejected, this mechanism is called a **write-back cache**
    - Altered data in cache lines that has not yet been written back causes that cache line to be labelled as **dirty**

Flag Bits Associated with each Cache Line

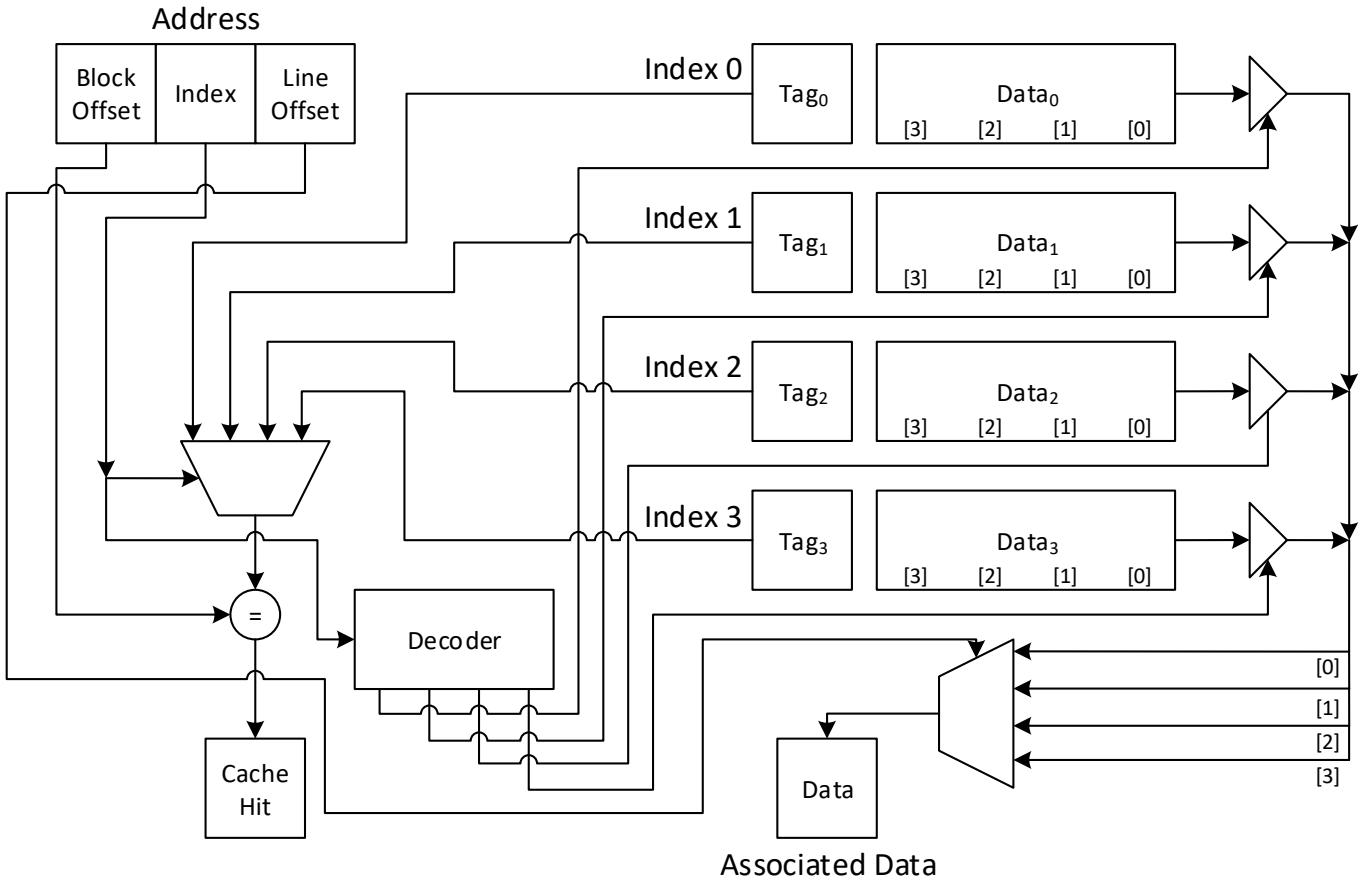
- Flag bits are associated with each cache line to retain information about that line
  - Valid vs. Invalid – **valid flag**
    - Is the cache line filled with valid address/data?
  - Clean vs. Dirty – **dirty flag**
    - Does the cache line contain data that needs to be written back to main memory?

Wider Cache Line

- Each entry in the cache is called a **cache line**
- The width of a cache line need not be the same as the word size
- A wider cache line will usually match the width of the memory bus
  - The memory bus width is the number of bits read on a memory read access

- When the cache line is wider than a word,
  - The tag is the address of the beginning of the memory block
  - Matching of the address is performed on the part of the address that is the same across the block

Simplified Direct-Mapped Cache with Wider Lines Schematic



Separate I and D Caches

- Separate caches for I (Instruction) and D (Data) accesses adjust better to access patterns
  - This way, data accesses do not eject instructions from the cache and vice versa
  - However, the I and D caches are committed by design to work for just instruction and data accesses, respectively
    - Less cache memory is available for just I or D
    - Because the ratio of how much cache is available for I and D is set at design time, flexibility is lost at run time
- Instruction cache probably doesn’t need to be able to deal with write updates – simplifies the design

Cache Levels

- Often there are multiple levels of caches
  - Facilitates a smooth trade-off between speed, cost, and size
- Level 1 (L1) Cache
  - In the processor core/chip
- Level 2 (L2) Cache
  - Often off chip
- Level 3 (L3) Cache
  - Often shared among cores/processors