Computer Architecture
Memory Cache

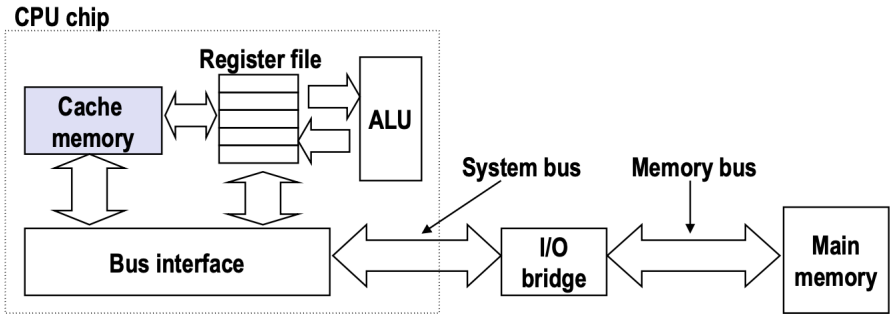
Cain Susko

Queen's University
School of Computing

March 30, 2022

Cache

The concept of a cache is to have a smaller 'cache' of used data from all data from memory in order to make the accessing of said important data faster. Recall that, faster memory is smaller and more expensive, hence only the important data is in the cache. A cache is conventionally based on SRAM, where the hardware manages the cache. When a CPU is looking for data in a computer, it looks in the cache first. An example of a cache structure is the following:



Organization

A cache is divided into sets S of lines E, in which each line can store a certain amount of bits B. Therefore, the Size of a cache is:

$$C = S \times E \times B$$

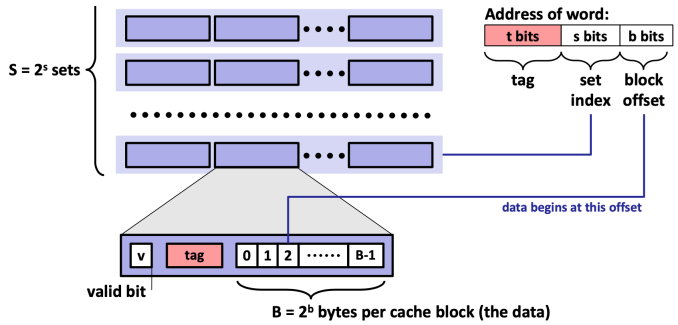
Within each word there is a 'valid' bit, a tag (the address for the word), and the data stored in B number of bits.

Read

The algorithm for reading from a cache is as follows:

- Locate set
- Check if any line in set has matching tag
- if yes and valid bit = true, hit !
- Locate data starting at offset

In more detail, the Tag and Addressing process is shown below:



Direct Addressing

In a direct address cache, each set S only has 1 line $E = 1$. Such that, when we are adding data to a direct address cache, there are two discrete outcomes:

$$\text{hit} \quad \text{miss}$$

if there is a data point at the tag x, and one is trying to add data at that data-point, it is a miss. If there was no previous data at x, then it is a hit.

Reading When a direct address cache reads, it must parse the address to find the correct place of the data. Given the address:

$$0[0000_2]$$

Assuming that: $M = 16, B = 2, S = 4, E = 1$, ($M$=mem) this input would be a miss and the desired data in the computer's memory would be: $M[0:1]$

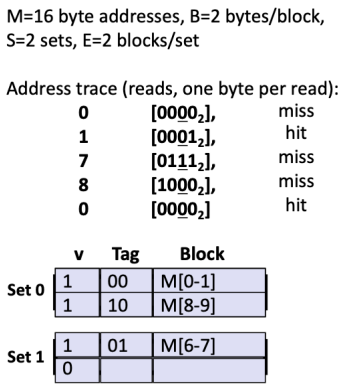
Address trace (reads, one byte per read):

0	**[0000_2],**	miss
1	**[0001_2],**	hit
7	**[0111_2],**	miss
8	**[1000_2],**	miss
0	**[0000_2]**	miss

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1			
Set 2			
Set 3	1	0	M[6-7]

Set Associative Cache

for a 2 way set associative cache there are 2 lines per set ($E = 2$). Each set is addressed by the tag of both words in the set, followed by the specification of which word in the set (it can be 0/1). And the block offset in memory.



While this style of cache uses less memory for addressing, it cannot store as much information and will evict data more often than a conventional cache.

Fully Associative Cache

A Fully associative cache is a cache which only has $S = 1$ set with all of the words in the cache ($E = C/B$). The problem with this configuration is that it makes it so that the tag must be quite large to be able to index the whole set. Additionally, when we search for a word, we are essentially doing linear search, which is not quite as fast as with the previous cache types

Writing

Data has multiple copies on the many memory units on a computer.

Hit When there is a hit, 2 things may happen:

- Write Through - write direct to memory
- Write Back - defer write to memory until eviction of line

Note: for write-back an extra (dirty) bit is needed to specify if a word is in memory yet.

Miss If there is a miss, 2 things can also happen:

- Write Allocate - load into cache and update in cache
- No Write Allocate - writes direct to memory, does not load into cache

Note: write-allocate is good for when more writes to the same location follow

Cache Performance

There are a few metrics we use in order to measure the performance of a cache:

Miss Rate This is equal to the fraction of memory which is not found in the cache. This can range from 3-10% for L1 and $< 1\%$ in L2.

Hit Time This equals the time to deliver a line from the cache to the processor. This is typically 4 cycles for L1 and 10 for L2.

Miss Penalty This equals the additional time required when a miss occurs in the cache. This is normally 50-200 cycles for main memory.