CSM 61C Fall 2019

Intro to Caching

Mentoring 8: November 4, 2019

1 Gotta Cache 'Em All

1.1 Please fill out the corresponding cells in the table below on caches.

	Compulsory	Conflict	Capacity
Cause			
Solution			

	Compulsory	Conflict	Capacity
Cause	First time entered in cache	Two addresses map to same block	Not enough space in cache
Solution	No solution. Darn.	Increase associativity	Increase size of cache

Meta: There is no "one size fits all" cache. Each miss (aside from compulsory) can be solved differently, but it's all a matter a trade-offs. We can increase associativity to fix conflict misses, but this adds hardware to make sure we're reading from the right set. Hardware is often costly due to size/spec restraints, and with the end of Moore's Law, we need to be strategic about which transistors are going where. We can increase the size of our cache to fix capacity misses, but as the cache gets larger, lookups become slower and our cache approaches the same limitations in speed as main memory. Optimizing the cache is usually a 2-level problem: 1) we want the cache to be big enough to hold as much useful data as possible, but small enough to maintain the speed of fast accesses, and 2) we want our code to be knowledgeable of the cache to reduce misses in general.

2 Break It Down Now Y'All

2.1 Calculate the number of bits used for the T/I/O for each cache specification below. Assume we are using a 16-bit system.

(a) 4 KiB cache, 256 B block size, direct mapped

$4 \text{ KiB} = 2^2 \times 2^{10} \text{ B} = 2^{12} \text{ B}$
$256 \text{ B} = 2^8 \text{ B} \rightarrow 8 \text{ bit offset}$
$\frac{2^{12}}{2^8} = 2^4 \rightarrow 4 \text{ bit index}$
$16 - 8 - 4 = 4 \text{ bit tag}$

(b) 4 KiB cache, 256 B block size, 4-way set associative

$4 \text{ KiB} = 2^2 x 2^{10} \text{ B} = 2^{12} \text{ B} \ / \ 2^2 = 2^{10} \text{ B}$
$256 \text{ B} = 2^8 \text{ B} \rightarrow 8 \text{ bit offset}$
$\frac{2^{10}}{2^8} = 2^2 \rightarrow 2 \text{ bit index}$
$16 - 8 - 2 = 6 \text{ bit tag}$

(c) 4 KiB cache, 256 B block size, fully associative.

$256 \text{ B} = 2^8 \text{ B} \rightarrow 8 \text{ bit offset}$
$16 - 8 = 8 \text{ bit tag}$
No need for index

2.2 For each sequence of memory accesses below, determine whether each memory access is a hit, compulsory miss, conflict miss, or capacity miss. Assume each cache has 4 KiB of memory split into 256 B blocks, and that the reads occurred in order.

(a) Cache Type: Direct-Mapped: Reads:
`0xABCD`
`0xABEF`

`0xABCD`: Compulsory Miss
`0xABEF`: Hit

(b) Cache Type: 4-way Set-Associative: Reads: `0xABCD`
`0xBBCD`

`0xABCD`: Compulsory Miss
`0xBBCD`: Compulsory Miss
META: Students might be confused why the second miss is not a conflict miss. Because the cache is 4-way set associative, loading an item which maps to the same set as 0xABCD doesn't conflict with the previous memory block.

(c) Cache Type: Fully Associative Reads:
`0xAABB`
`0xBBCC`

`0xAABB`: Compulsory Miss
`0xBBCC`: Compulsory Miss

3 Multilevel $

3.1 A machine has an 8 way set associative cache with 512 B of data. The size of each block is 16 B and there are 8 MiB of main memory. How large are the tag, index, and offset fields for an address on this machine using this cache?

Tag: 17 Index: 2 Offset: 4

Detailed solution: 8 MiB ($2^2$3 B) main memory $\rightarrow$ 23 bit memory address
cache size = 512 B = 2^9 B
block size = 16 B = 2^4 B
number of blocks = cache size / block size = $\frac{2^9}{2^4} = 2^5$
number of sets = number of blocks / associativity = $\frac{2^5}{8} = 2^2 = 4$
$I = \log_2(\text{num of sets}) = \log_2 4 = 2$
$O = \log_2(\text{block size}) = \log_2 2^4 = 4$
T = number of total address bits $- I - O = 23 - 2 - 4 = 17$

3.2 Now we have a different machine with two caches, an L1 and an L2 cache. Both caches are direct mapped caches. The L1 cache can hold 256 B of data and the L2 cache can hold 4 KiB of data. Assume the following code is run on this machine:

```
#define ARR_SIZE 2048

uint16_t sum (uint16_t *arr) {
    total = 0;

    for (int i = 0; i < ARR_SIZE; i++) {
        total += arr[i];
    }

    return total;
}
```

This produces a hit rate (HR) of $\frac{7}{8}$ for the L1 cache and $\frac{3}{4}$ for the L2 cache. Given that `arr` is a block aligned address and `sizeof (uint16_t) == 2`:

3.3 What is the blocksize of the L1 cache **in bytes** that produces its hit rate?

L1 blocksize: 16 B

A hit rate of $\frac{7}{8}$ for L1 means the miss/hit pattern is, for every 8 consecutive memory accesses (`arr[i]`), we miss on the first and hit on the next 7 accesses. This means on the first miss, L1 fetches a block from L2 into L1 that contains the element were currently accessing PLUS the next 7 elements in the array. Since each element is a `uint16_t`, which is 2 B, the block size for L1 is $8 \times 2 = 16$ B.

3.4 Use the variable Y to represent the answer to part (a). What is the blocksize of the L2 cache **in bytes** that produces its hit rate? Express your answer as a function of Y and NOT as a single number.

L2 blocksize: $4 \times Y = 64$ B

A hit rate of $\frac{3}{4}$ for L2 means the miss/hit pattern is, for every 4 misses at L1 level, L2 misses on the first L1 miss and hits on the next 3 L1 misses. So what happens is on the first L1 miss (and L2 also misses), L2 fetches a whole block from main memory, and that block is 4 times the size of a L1 block. This ensures that for the next 3 L1 misses, L2 already has the block that L1 is asking for.

Recall that the L1 HR is $\frac{7}{8}$ and the L2 HR is $\frac{3}{4}$. If the L1 Cache has a hit time of 2 cycles, the L2 cache has a hit time of 8 cycles, and main memory has a hit time of 96 cycles:

3.5 How many total cycles are spent accessing memory on this piece of code? Express your answer in the form $C \times 2^i$, where C is an integer not divisible by 2.

Total cycles: $2048 \times (2 + \frac{1}{8} \times (8 + \frac{1}{4} \times 96)) = 2^{11} \times (2 + \frac{8}{8} + \frac{96}{32}) = 2^{11} * 6 = 3 \times 2^{12} = 12,288$

3.6 If we change the L1 cache from being direct mapped to being fully associative with LRU, how does its hit rate change on the same code? Does it increase, decrease, not change at all, or is it impossible to tell?

No Change

The basic idea is that we never look back at a block once we've gone past it after its fetched from memory. So a cache of any associativity would not help.