CS 61C Fall 2019

Caches

Discussion 9: October 28, 2019

1 Understanding T/I/O

When working with caches, we have to be able to break down the memory addresses we work with to understand where they fit into our caches. There are three fields:

Tag - Used to distinguish different blocks that use the same index. Number of bits: (# of bits in memory address) - Index Bits - Offset Bits
Index - The set that this piece of memory will be placed in. Number of bits: $\log_2$(# of indices)
Offset - The location of the byte in the block. Number of bits: $\log_2$(size of block)

Given these definitions, the following is true:

$\log_2$(memory size) = address bit-width = # tag bits + # index bits + # offset bits

Another useful equality to remember is:

$$\text{cache size} = \text{block size} * \text{num blocks}$$

we have a capacity of 32B and a block size of 8B so we have $\frac{32B}{8B}$ = 4 blocks/indices

1.1 Assume we have a direct-mapped byte-addressed cache with capacity 32B and block size of 8B. Of the 32 bits in each address, which bits do we use to find the index of the cache to use?

$\text{Index} = \log_2\left(\frac{32}{8}\right) = \log_2(2^2) = 2$
$\text{offset} = \log_2(8) = \log_2(2^3) = 3$
$\text{Tag} = 32 - 3 - 2 = 27$

32 bit system = 32 bits

Tag	index	offset
27	2	3

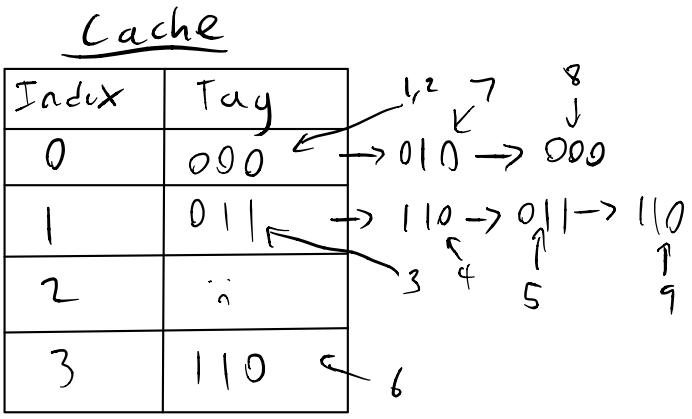
We use bits 3 and 4, where we denote the MSB as 31 and the LSB as 0.

1.2 Which bits are our tag bits? What about our offset?

The offset is 3 bits, and our tag is the remaining high-order bits.

1.3 Classify each of the following byte memory accesses as a cache hit (H), cache miss (M), or cache miss with replacement(R). It is probably best to try drawing out the cache before going through so that you can have an easier time seeing the replacements in the cache. The following white space is to do this:

	Address	T/I/O	T	I	O	Hit, Miss, Replace
1	0x00000004	000	0	0	100	M compulsory
2	0x00000005	000	0	0	101	H
3	0x00000068	011	0	1	000	M compulsory
4	0x000000C8	110	0	1	000	R compulsory
5	0x00000068	011	0	1	000	R conflict
6	0x000000DD	110	1	1	101	M compulsory
7	0x00000045	010	0	0	101	R compulsory
8	0x00000004	000	0	0	100	R capacity
9	0x000000C8	110	0	1	000	R capacity



- 0x00000004 Index 0, Tag 0: M
- 0x00000005 Index 0, Tag 0: H
- 0x00000068 Index 1, Tag 3: M
- 0x000000C8 Index 1, Tag 6: R
- 0x00000068 Index 1, Tag 3: R
- 0x000000DD Index 3, Tag 6: M
- 0x00000045 Index 0, Tag 2: R
- 0x00000004 Index 0, Tag 0: R
- 0x000000C8 Index 1, Tag 6: R

Note that the M and R distinction here is for student understanding, and that the cache doesn't behave differently for these cases.

2 The 3 C's of Misses

2.1 Classify each M and R above as one of the 3 types of misses described below:

1. Compulsory: First time you ask the cache for a certain block. A miss that must occur when you first bring in a block. Reduce compulsory misses by having longer cache lines (bigger blocks), which bring in the surrounding addresses along with our requested data. Can also pre-fetch blocks beforehand using a hardware prefetcher (a special circuit that tries to guess the next few blocks that you will want).
2. Conflict: Occurs if, hypothetically, you went through the ENTIRE string of accesses with a fully associative cache and wouldn't have missed for that specific access. Increasing the associativity or improving the replacement policy would remove the miss.
3. Capacity: Capacity misses are independent of the associativity of your cache. If you hypothetically ran the ENTIRE string of memory accesses with a fully associative cache of the same size as your cache, and it was a miss for that specific access, then this miss is a capacity miss. The only way to remove the miss is to increase the cache capacity.

Note: The test you can use to see if a miss is a conflict miss is the same as the test you can use to see if a miss is a capacity miss.

Note: There are many different ways of fixing misses. The name of the miss doesn't necessarily tell us the best way to reduce the number of misses.

- 0x00000004, Compulsory
- 0x00000005, N/A
- 0x00000068, Compulsory

if not seen block before
=> compulsory
else if from start fully associative:
if miss was avoided:
=> conflict
else:
=> capacity

- 0x000000C8, Compulsory
- 0x00000068, Conflict
- 0x000000DD, Compulsory
- 0x00000045, Compulsory
- 0x00000004, Capacity
- 0x000000C8, Capacity

Compulsory: never seen this block before
conflict: If I had a fully associative cache (I could put any block in any location) of the same size, if it could have fit, it is a conflict otherwise it is a Capacity miss

3 Code Analysis

Given the follow chunk of code, analyze the hit rate given that we have a byte-addressed computer with a total memory of **1 MiB**. It also features a **16 KiB** Direct-Mapped cache with **1 KiB** blocks. Assume that your cache begins cold.

$2^4 \cdot 2^{10} = 2^{14}$
$16 \cdot 1024$

```
#define NUM_INTS 8192   // 2^13
int A[NUM_INTS];        // A lives at 0x10000
int i, total = 0;
for (i = 0; i < NUM_INTS; i += 128) {
    A[i] = i;           // Line 1
}
for (i = 0; i < NUM_INTS; i += 128) {
    total += A[i];      // Line 2
}
```

3.1 How many bits make up a memory address on this computer?

Total mem size
We take $\log_2(1 \text{ MiB}) = \log_2(2^{20}) = 20$

3.2 What is the T:I:O breakdown?

size of block
Offset $= \log_2(1 \text{ KiB}) = \log_2(2^{10}) = 10$
size of Cache / size of a block
Index $= \log_2(\frac{16 \text{ KiB}}{1 \text{ KiB}}) = \log_2(16) = 4$
Tag $= 20 - 4 - 10 = 6$
index offset

3.3 Calculate the cache hit rate for the line marked Line 1:

jump by 128 where each item is an integer

The integer accesses are $4 * 128 = 512$ bytes apart, which means there are 2 accesses per block. The first accesses in each block is a compulsory cache miss, but the second is a hit because `A[i]` and `A[i+128]` are in the same cache block. Resulting in a hit rate of **50%**.

1024 bytes in a block.

3.4 Calculate the cache hit rate for the line marked Line 2:

The size of A is $8192 * 4 = 2^{15}$ bytes. This is exactly twice the size of our cache. At the end of Line 1, we have the second half of A inside our cache, but Line 2 starts with the first half of A. Thus, we cannot reuse any of the cache data brought in from Line 1 and must start from the beginning. Thus our hit rate is the same as Line 1 since we access memory in the same exact way as Line 1. We dont have to

consider cache hits for total, as the compiler will most likely store it in a register. Resulting in a hit rate of **50%**.

4 Cache Associativity

In the previous problems, we have a Direct-Mapped cache, in which blocks map to specifically one slot in our cache. This is good for quick replacement and finding out block, but not good for efficiency of space!

This is where we bring associativity into the matter. We define associativity as the number of slots a block can potentially map to in our cache. Thus, a Fully-Associative cache has the most associativity, meaning every block can go anywhere in the cache.

For an N-way associative cache, the following is true:

$$\text{N} * \#\text{ sets} = \#\text{ blocks}$$

Index = $\log_2\left(\frac{\text{cache size}}{\text{Block size} \cdot \text{associativity}}\right)$
$= \log_2\left(\frac{32}{8 \cdot 2}\right) = \log_2(2) = 1$
Offset = $\log_2(\text{block size})$
$= \log_2(8) = 3$
Tag = 8 − 1 − 3
= 4

4.1 Heres some practice involving a 2-way set associative cache. This time we have an 8-bit address space, 8 B blocks, and a cache size of 32 B. Classify each of the following accesses as a cache hit (H), cache miss (M) or cache miss with replacement (R). For any misses, list out which type of miss it is.

Address	T/I/O	T	I	O	Hit, Miss, Replace
0b0000 0100	0000		0	100	M compulsory
0b0000 0101	0000		0	101	H
0b0110 1000	0110		1	000	M compulsory
0b1100 1000	1100		1	000	M compulsory
0b0110 1000	0110		1	000	H
0b1101 1101	1101		1	101	R compulsory
0b0100 0101	0100		0	101	M compulsory
0b0000 0100	0000		0	100	H
0b1100 1000	1100		1	000	R capacity

For this solution, we assume that we have an LRU replacement policy. In general, this is not necessarily the case.

```
0b0000 0100     Tag 0000, Index 0, Offset 100 - M, Compulsory
0b0000 0101     Tag 0000, Index 0, Offset 101 - H
0b0110 1000     Tag 0110, Index 1, Offset 000 - M, Compulsory
0b1100 1000     Tag 1100, Index 1, Offset 000 - M, Compulsory
0b0110 1000     Tag 0110, Index 1, Offset 000 - H
0b1101 1101     Tag 1101, Index 1, Offset 101 - R, Compulsory
0b0100 0101     Tag 0100, Index 0, Offset 101 - M, Compulsory
0b0000 0100     Tag 0000, Index 0, Offset 100 - H
0b1100 1000     Tag 1100, Index 1, Offset 000 - R, Capacity
```

cache

Index	Tag
0	0000
0	0100
1	0110
1	1100

1, 2, 8
7
→ 1101 ← 6
3, 5
4, 9

4.2 What is the hit rate of our above accesses?

$\frac{\text{3 hits}}{\text{9 accesses}} = \frac{1}{3}$ hit rate