CS 61C Fall 2018

Caches II, Floating Point

Discussion 6: October 1, 2018

1 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.

```
#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
}
```

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

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

1.2 What is the T:I:O breakdown?

Offset $= \log_2(1 \text{ KiB} = \log_2(2^{10}) = 10$
Index $= \log_2(\frac{16 \text{ KiB}}{1 \text{ KiB}}) = \log_2(16) = 4$
Tag $= 20 - 4 - 10 = 6$

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

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%**.

1.4 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%**.

2 AMAT

Recall that AMAT stands for Average Memory Access Time. The main formula for it is:
AMAT = Hit Time + Miss Rate * Miss Penalty
We also have two types of miss rates, global and local. Global is calculated as: Fraction of ALL accesses that missed at that level over all accesses total. Whereas local is calculated: Fraction of ALL access that missed at that level over all access to that level total.

2.1 An L2$, out of 100 total accesses to the cache system, missed 20 times. What is the global miss rate of L2$?

$\frac{20}{100} = 20\%$

2.2 If L1$ had a miss rate of 50%, what is the local miss rate of L2$?

$\frac{20}{50\%*100} = \frac{20}{50} = 40\%$. We know that L2$ is accessed when L1$ misses, so if L1$ misses 50% of the time, that means we access L2$ 50 times.

Suppose your system consists of:

1. An L1$ that hits in 2 cycles and has a local miss rate of 20%
2. An L2$ that hits in 15 cycles and has a global miss rate of 5%
3. Main memory hits in 100 cycles

2.3 What is the local miss rate of L2$?

L2$ Local miss rate = $\frac{\text{Global Miss Rate}}{\text{L1\$ Miss Rate}} = \frac{5\%}{20\%} = 0.25 = 25\%$

2.4 What is the AMAT of the system?

AMAT = 2 + 20% x 15 + 5% x 100 = 10 cycles (using global miss rates)
Alternatively, AMAT = 2 + 20% x (15 + 25% x 100) = 10 cycles

2.5 Suppose we want to reduce the AMAT of the system to 8 cycles or lower by adding in a L3$. If the L3$ has a local miss rate of 30%, what is the largest hit time that the L3$ can have?

Let H = hit time of the cache. Using the AMAT equation, we can write:
$2 + 20\% * (15 + 25\% * (H + 30\% * 100)) \leq 8$
Solving for H, we find that $H \leq 30$. So the largest hit time is 30 cycles.

3 Floating Point

The IEEE 754 standard defines a binary representation for floating point values using three fields:

- The *sign* determines the sign of the number (0 for positive, 1 for negative)
- The *exponent* is in **biased notation** with a bias of 127
- The *significand or mantissa* is akin to unsigned, but used to store a fraction instead of an integer

The below table shows the bit breakdown for the single precision (32-bit) representation.

1	8	23
Sign	Exponent	Mantissa/Significand/Fraction

For normalized floats:
Value $= (-1)^{Sign} * 2^{Exp-Bias} * 1.\textbf{significand}_2$
For denormalized floats:
Value $= (-1)^{Sign} * 2^{Exp-Bias+1} * 0.\textbf{significand}_2$

Exponent	**Significand**	**Meaning**
0	Anything	Denorm
1-254	Anything	Normal
255	0	Infinity
255	Nonzero	NaN

3.1 How many zeroes can be represented using a float?

2

3.2 What is the largest finite positive value that can be stored using a single precision float?

`0x7F7FFFFF` $= (2 - 2^{-23}) * 2^{127}$

3.3 What is the smallest positive value that can be stored using a single precision float?

`0x00000001` $= 2^{-23} * 2^{-126}$

3.4 What is the smallest positive normalized value that can be stored using a single precision float?

`0x00800000` $= 2^{-126}$

3.5 Cover the following numbers from binary to decimal or from decimal to binary:

- 0x00000000

  0

- 8.25

  0x41040000

- 0x00000F00

  $(2^{-12} + 2^{-13} + 2^{-14} + 2^{-15}) * 2^{-126}$

- 39.5625

  0x421E4000

- 0xFF94BEEF

  NaN

- -∞

  0xFF800000

4 Extra Stuff on Caches!

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	Hit, Miss, Replace
`0b0000 0100`		
`0b0000 0101`		
`0b0110 1000`		
`0b1100 1000`		
`0b0110 1000`		
`0b1101 1101`		
`0b0100 0101`		
`0b0000 0100`		
`0b1100 1000`		

```
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
```

4.2 What is the hit rate of our above accesses?

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