CSE 351 Section 7 – Caches

Exercise 1: 2-way Set Associative Cache Accesses

Cache size: 64B, Block size: 4B, **2-way associative**, 12-bit addresses. All values shown in hex (Tags are padded).

Set	Valid	Tag	B0	B1	B2	B3
0	0	12	81	04	58	eb
0	0	07	B3	DF	D3	E4
1	0	4D	70	AE	CC	01
1	1	2F	01	20	40	03
2	1	03	4F	D4	A1	3B
2	1	0E	99	09	87	56
3	0	06	E9	09	ED	DD
3	0	6E	F2	AB	94	31

Set	Valid	Tag	B0	B1	B2	B3
4	0	06	CA	FE	F0	0D
4	0	2B	D3	3C	52	FB
5	1	21	DE	AD	BE	EF
5	0	20	AA	9B	DA	50
6	0	41	21	2D	BA	A0
6	1	37	22	B6	DB	AA
7	1	11	00	12	51	55
7	0	6C	12	7E	D2	00

Offset bits: _2_____

Index bits: _3_____

Tag bits: _7_____

	Hit or Miss?	Data returned
a) Read 1 byte at 0x435 = 0b 0100 0011 0101	Hit	0xAD
b) Read 1 byte at 0x388 = 0b 0011 1000 1000	Miss (no tag match)	n/a

Exercise 2: Fully Associative Cache Accesses

Cache size: 64B, Block size: 4B, **fully associative**, 12-bit addresses. All values shown in hex (Tags are padded).

Set	Valid	Tag	B0	B1	B2	B3
0	1	1F4	01	02	03	04
0	0	1F9	12	05	E2	93
0	0	266	20	BB	34	EA
0	1	0AB	02	30	44	67
0	1	100	F4	4D	EE	11
0	1	034	FD	EC	BA	23
0	1	077	12	23	34	45
0	0	341	73	16	2B	1F

Set	Valid	Tag	B0	B1	B2	B3
0	0	02F	AB	C5	43	FF
0	0	1C6	00	11	22	33
0	1	101	DA	14	EE	22
0	1	045	67	78	89	9A
0	0	208	26	8F	50	66
0	1	001	70	00	44	A6
0	1	016	90	32	AC	24
0	0	395	48	B3	20	53

Offset bits: _2_____

Index bits: _0_____

Tag bits: _10____

	Hit or Miss?	Data returned
a) Read 1 byte at 0x1DD = 0b 0001 1101 1101	Hit	0x23
b) Read 1 byte at 0x719 = 0b 0111 0001 1001	Miss (invalid)	n/a

Exercise 3: Code Analysis

This code accesses a two-dimensional array of size 64×64 `int`s, with variables `sum`, `i`, and `j` stored in registers. Assume we are using a **direct-mapped**, **1 KiB cache** with **16 B block size**, and that the cache starts cold/"empty."

```
int sum = 0;
for (int i = 0; i < 64; i++)
    for (int j = 0; j < 64; j++)
        sum += array[i][j];          // assume &array = 0x600000
```

a) What is the miss rate of the execution of the entire loop?

Every block can hold 4 ints (16B/4B per int), so we will need to pull a new block from memory every 4 accesses of the array. This means this miss rate is $\frac{4\ bytes\ per\ int}{16\ bytes\ per\ block} = \frac{1\ block}{4\ ints}$ = 0.25 = 25%

b) If we have an average memory access time (AMAT) of 60 ns and a hit time of 10 ns, what is the miss penalty?

With the 25% miss rate from part **(a)**, we have 10 + 0.25 * MP = 60. Solving for MP tells us the miss penalty is 200 ns.

c) What code modifications can change the miss rate? Brainstorm before trying to analyze.

Possible answers:

- switch the loops (*i.e.*, make `j` the outer loop and `i` the inner loop)
- switch `j` and `i` in the array access
- make `array` a different type (*e.g.*, `char[][]`, `long[][]`)
- make `array` an array of Linked Lists or a 2-level array, etc.

d) What cache parameter changes (size, associativity, block size) can change the miss rate?

Modifying cache size:
No, it doesn't matter how big the cache is in this case (if the block size doesn't change). We will still be pulling the same amount of data each miss, and we will still have to go to memory every time we exhaust that data.

Modifying associativity:
No, this is helpful if we want to reduce conflict misses, but since the data we're accessing is all in contiguous memory (thanks arrays!), booting old data to replace it with new data isn't an issue.

Modifying block size:
Yes, bigger blocks mean we pull bigger chunks of contiguous elements in the array every time we have a miss. Bigger chunks at a time means fewer misses down the line. Likewise, smaller blocks increase the frequency with which we need to go to memory (think back to the calculations we did in part **(a)** to see why this is the case).

NOTE: Remember that the results we got were for this specific example. There are some code examples in which changing the size or associativity of the cache will change the miss rate.