CSE 351 Section 7 – Caches

Hi there! Welcome back to section, we're happy that you're here ☺

Accessing a Cache (Hit or Miss?)

Assume the following caches all have block size $K = 4$ and are in the current state shown (you can ignore "—"). All values are shown in hex. Tag fields are NOT padded, while bytes of the cache blocks are shown in full. The word size for the machine with these caches is 12 bits (i.e. addresses are 12 bits long)

Direct-Mapped:

Set	Valid	Tag	B0	B1	B2	B3
0	1	15	63	B4	C1	A4
1	0	—	—	—	—	—
2	0	—	—	—	—	—
3	1	D	DE	AF	BA	DE
4	0	—	—	—	—	—
5	0	—	—	—	—	—
6	1	13	31	14	15	93
7	0	—	—	—	—	—

Set	Valid	Tag	B0	B1	B2	B3
8	0	—	—	—	—	—
9	1	0	01	12	23	34
A	1	1	98	89	CB	BC
B	0	1E	4B	33	10	54
C	0	—	—	—	—	—
D	1	11	C0	04	39	AA
E	0	—	—	—	—	—
F	1	F	FF	6F	30	0

Offset bits: _____

Index bits: _____

Tag bits: _____

	Hit or Miss?	Data returned
a) Read 1 byte at `0x7AC`		
b) Read 1 byte at `0x024`		
c) Read 1 byte at `0x99F`		

2-way Set Associative:

Set	Valid	Tag	B0	B1	B2	B3
0	0	—	—	—	—	—
1	0	—	—	—	—	—
2	1	3	4F	D4	A1	3B
3	0	—	—	—	—	—
4	0	6	CA	FE	F0	0D
5	1	21	DE	AD	BE	EF
6	0	—	—	—	—	—
7	1	11	00	12	51	55

Set	Valid	Tag	B0	B1	B2	B3
0	0	—	—	—	—	—
1	1	2F	01	20	40	03
2	1	0E	99	09	87	56
3	0	—	—	—	—	—
4	0	—	—	—	—	—
5	0	—	—	—	—	—
6	1	37	22	B6	DB	AA
7	0	—	—	—	—	—

Offset bits: _____

Index bits: _____

Tag bits: _____

	Hit or Miss?	Data returned
a) Read 1 byte at `0x435`		
b) Read 1 byte at `0x388`		
c) Read 1 byte at `0x0D3`		

Fully Associative:

Set	Valid	Tag	B0	B1	B2	B3
0	1	1F4	00	01	02	03
0	0	—	—	—	—	—
0	1	100	F4	4D	EE	11
0	1	77	12	23	34	45
0	0	—	—	—	—	—
0	1	101	DA	14	EE	22
0	0	—	—	—	—	—
0	1	16	90	32	AC	24

Set	Valid	Tag	B0	B1	B2	B3
0	0	—	—	—	—	—
0	1	AB	02	30	44	67
0	1	34	FD	EC	BA	23
0	0	—	—	—	—	—
0	1	1C6	00	11	22	33
0	1	45	67	78	89	9A
0	1	1	70	00	44	A6
0	0	—	—	—	—	—

Offset bits: _____

Index bits: _____

Tag bits: _____

	Hit or Miss?	Data returned
a) Read 1 byte at `0x1DD`		
b) Read 1 byte at `0x719`		
c) Read 1 byte at `0x2AA`		

Code Analysis

Consider the following code that accesses a <u>two-dimensional</u> array (of size 64×64 `int`s).
Assume we are using a direct-mapped, 1 KiB cache with 16 B block size.

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

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

b) What code modifications can <u>change</u> the miss rate? Brainstorm before trying to analyze.

c) What cache parameter changes (size, associativity, block size) can <u>change</u> the miss rate?

Cache ya later!

Suppose we have a 1 KiB direct-mapped cache with 256 B blocks.

a) The code snippet below loops through a character array. Give the value of LEAP that results in a hit rate of 15/16.

```
#define ARRAY_SIZE 8192
char string[ARRAY_SIZE];              // &string = 0x8000
for(i = 0; i < ARRAY_SIZE; i += LEAP) {
     string[i] |= 0x20;               // to lower
}
```

b) For the loop shown in part (a), let LEAP = 64. Which ONE of the following changes would **increase** the hit rate in the cache?

Increase Block Size Increase Cache Size Increase LEAP

c) For each of the following cache access parameters, calculate the AMAT. Please simplify and include units!

$ Hit Time	$ **Miss** Rate	MEM Hit Time
2 ns	40%	400 ns