CSE 351 Section 8 – More Caches, Processes & Concurrency

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

Practice Cache Exam Problem (11 pts)

We have a 64 KiB address space and two different caches. Both are 1 KiB, direct-mapped caches with random replacement and write-back policies. **Cache X** uses 64 B blocks and **Cache Y** uses 256 B blocks.

a) Calculate the TIO address breakdown for **Cache X**:

Tag	Index	Offset

b) During some part of a running program, **Cache Y**'s management bits are as shown below. Four options for the next two memory accesses are given (R = read, W = write). Circle the option that results in data from the cache being *written to memory*.

Line	Valid	Dirty	Tag
00	0	0	1000 01
01	1	1	0101 01
10	1	0	1110 00
11	0	0	0000 11

(1) `R 0x4C00, W 0x5C00`

(2) `W 0x5500, W 0x7A00`

(3) `W 0x2300, R 0x0F00`

(4) `R 0x3000, R 0x3000`

c) The code snippet below loops through a character array. Give the value of LEAP that results in a Hit Rate of 15/16 for **Cache Y**.

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

d) For the loop shown in part (c), let LEAP = 64. Circle ONE of the following changes that increases the hit rate of **Cache X**:

Increase Block Size Increase Cache Size Add a L2$ Increase LEAP

e) For the following cache access parameters, calculate the AMAT. Please simplify and include units.

L1$ Hit Time	L1$ **Miss** Rate	MEM Hit Time
2 ns	40%	400 ns

Benedict Cumbercache:

Given the following sequence of access results (addresses are given in decimal) on a cold/empty cache of size 16 bytes, what can we *deduce* about its properties? Assume an LRU replacement policy.

```
(0, Miss), (8, Miss), (0, Hit), (16, Miss), (8, Miss)
```

1) What can we say about the block size?

2) What is this cache's associativity?

3) How many sets could this cache have?

4) How many bits will the tag use given an n-bit address?

Fork and Concurrency:

Consider this code using Linux's `fork`:

```
int x = 7;
if( fork() ) {
    x++;
    printf(" %d ", x);
    fork();
    x++;
    printf(" %d ", x);
} else {
    printf(" %d ", x);
}
```

What are *all* the different possible outputs (i.e. order of things printed) for this code?
(Hint: there are four of them.)