# 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)

2^16 2^10 E = 1

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.

2^6 2^8

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

| Tag | Index | Offset |
|---|---|---|
| 6 | 4 | 6 |

s = cache bits - offset = 10-6
t = 16 - s - b = 16 - 4 - 6

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*.

cache Y has 8 offset bits (two hex digits) and two index bits (1/2 hex digit) so the third to last hex digit determines which set/line the address gets mapped to in cache (by looking at the last 2 bits of that hex digit)

to write to memory, we have to overwrite a dirty piece of memory. Thus we either have to overwrite line 01, or create some dirty piece through a write then evict it with a read (or another write)

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

this matches tag at line 01, so no eviction occurs

~~(1)~~ R 0x4C00, W 0x5C00 (2) W 0x5500, W 0x7A00

set: 00 00 01 10

(3) W 0x2300, R 0x0F00 (4) R 0x3000, R 0x3000

set: 11 11 00 00

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
}
```

this operation accesses the data at address i TWICE (it is both a read and write). So really we need to choose LEAP such that only 8 leaps (rather than 16) are in the same memory block.

256 / 8 = 32

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 |

AMAT = Hit time + miss rate * miss penality = 2 + 0.4 * 400 = 162

## 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?

   By first two, block size is <= 8.

2) What is this cache's associativity?

   The associativity < 3 because the second 8 call misses, so 8 must have been evicted from the cache (and there were only 2 other memory look ups). If E = 1

3) How many sets could this cache have?

   Since E = 2, cache size = 16, we know S = 16 / (2 * blocksize) = 8/ blocksize. Namely the number of sets <= 8

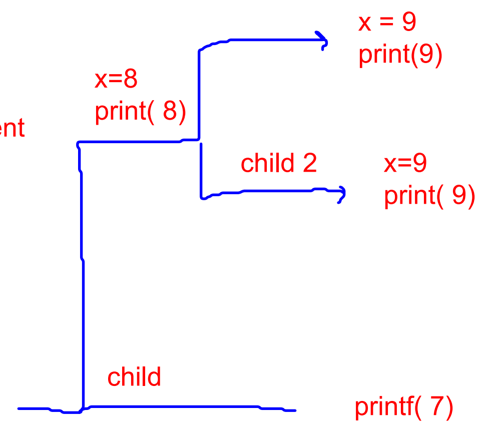
4) How many bits will the tag use given an *n*-bit address?

   We know blocksize * num sets = 8 as the associativity is 2 and cache size 16. Hence there are 3 bits allocated in total between the set index and block offset. Thus n-3 are tag bits

## 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.)

7 8 9 9
8 7 9 9
8 9 7 9
8 9 9 7

All we know is that the print(8) command has to come before the two print(9) commands, but the print(7) command may be interlaced between