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**:

$2^{10} \cdot 2^6 = 2^{16}$

addresses ⇒ 16 bit addresses

Tag	Index	Offset
16−10 = 6	$\frac{2^{10}}{2^6} = 2^4$ ⇒ 4	$2^6 = 64$ ⇒ 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: Tag: 6 bits; Index: $\frac{2^{10}}{2^8} = 2$ bits; Offset: 8 bits

we only care about 8 MSBs (tag + index)

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
dirty bit not set; 0b01001100... ; 0b01011100... ; dirty bit not set, so just update cache

(~~2~~) W 0x5500, W 0x7A00
0b01010101... ; 0b01111010 ; tags match, so don't write back ; dirty bit not set, so just update cache

[✓ (3)] W 0x2300, R 0x0F00
0b00100011.. ; 0b00001111... ; we wrote something to cache, then tried to read something different, so we need to write back to memory

(~~4~~) R 0x3000, R 0x3000
0b00110000 ; 0b00110000 ; (dirty bit not set on line 0)

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
}
```

We read then write each address. For a 15/16 hit rate, we must access 8 bytes per block: 1 miss for the first read, followed by 15 hits for subsequent R/Ws.

Since blocks are 256 B and chars are 1 byte: LEAP $= \frac{256}{8} = 32$

$\boxed{32}$

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] ↗ more hits/block

Increase Cache Size — N.C.

Add a L2$ — decreases miss penalty

Increase LEAP — N.C.

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 time)
(you always pay for hit time; you also pay for miss time, when you miss)

$2 + (0.4)(400) = 162$

$\boxed{162 \text{ ns}}$

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.

```
 ①           ②           ③           ④           ⑤
(0, Miss), (8, Miss), (0, Hit), (16, Miss), (8, Miss)
 00000       01000       00000       10000       01000
```

1) What can we say about the block size?

$\leq$ 8 byte block size

(access ② to address 8 misses)

2) What is this cache's associativity?

≤ 2 because ④ caused one of the prev entries to be evicted.

could it be direct mapped?

- block size = 1: 16 sets, 4 set bits ⇒ 16 should evict 0, but it doesn't
- block size = 2: 8 sets, 3 set bits, 1 offset bit ⇒ 16 should evict 0 but doesn't
- block size = 4: 4 sets, 2 set bits, 2 offset bits ⇒ 16 should evict 0 but doesn't
- block size = 8: 2 sets, 1 set bits, 3 offset bits ⇒ 16 should evict 0 but doesn't

} none match ⇒ must be 2-way

3) How many sets could this cache have?

We need to know block size for this. We know the cache is 2-way set associative and total size = 16.

$16 = 2 * S * K$

$K \in \{1, 2, 4, 8\}$ from Q1, so $S \in \{1, 2, 4, 8\}$

so we may have:
$S=8, K=1$
$S=4, K=2$
$S=2, K=4$
$S=1, K=8$

note that in each configuration, set bits + offset bits = 3. The last 3 bits of all of addresses are the same, so they'll all map to the same set regardless (which is what we want)
⇒ all set sizes work

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

offset bits $= \log_2 K$ $\qquad C = K \cdot E \cdot S \Rightarrow S = \frac{C}{KE}$

index bits $= \log_2\left(\frac{C}{KE}\right)$

tag bits $= n - \log_2\left(\frac{C}{KE}\right) - \log_2 K = n - \left(\log_2\left(\frac{C}{KE}\right) + \log_2 K\right) = n - \log_2\left(\frac{C}{KE} \cdot K\right) = n - \log_2(C/E)$

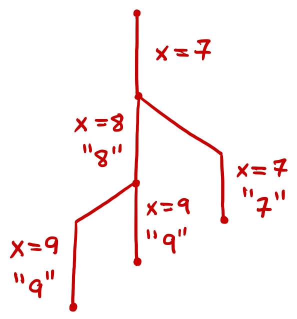
$= n - \log_2(16/2) = n-3$

Fork and Concurrency:

→ returns 0 to the child, child PID to parent

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
```

the only time we fork conditionally is the first fork. So the order of the 7 is undefined, but all others will always appear in the same relative order. (the order of the 9s may change, but that doesn't matter since they're both 9.)