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 \( \Rightarrow \) 16 bit addresses

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-10 = 6</td>
<td>( \frac{16}{2^4} = 2^4 \Rightarrow 4 )</td>
<td>( \frac{2^4}{2^2} = 2^2 \Rightarrow 0 )</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Line</th>
<th>Valid</th>
<th>Dirty</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>1000 01</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>1</td>
<td>0101 01</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1110 00</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0000 11</td>
</tr>
</tbody>
</table>

R 0x4C00, W 0x5C00 (2)
W 0x5500, W 0x7A00 (1)
W 0x2300, R 0x0F00 (3)
R 0x3000, R 0x3000 (4)

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**.

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

32

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

N.C.
N.C.

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

<table>
<thead>
<tr>
<th>L1$ Hit Time</th>
<th>L1$ Miss Rate</th>
<th>MEM Hit Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 ns</td>
<td>40%</td>
<td>400 ns</td>
</tr>
</tbody>
</table>

AMAT = (hit time) + (miss rate)(miss time)  
(you always pay for hit time, you also pay for miss time, when you miss)  
\[ 2 + (0.4)(400) = 162 \]
**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?

   - **4 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, 3 set bits | 16 should evict 0 but it doesn't
   - block size = 2: 8 sets, 5 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 * 8 + \( k \)
   - \( k \) ∈ \{1, 2, 4, 8\} from \( Q \), so \( S \in \{1, 2, 4, 8\} \)

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

   - offset bits = \( \log_2 \frac{k}{n} \)
   - index bits = \( \log_2 \left( \frac{c}{k} \right) \)
   - tag bits = \( n - \log_2 \left( \frac{k}{n} \right) - \log_2 k = n - \left( \log_2 \left( \frac{c}{k} \right) + \log_2 k \right) = n - \log_2 \left( \frac{c}{k} \cdot k \right) = n - \log_2 \left( \frac{c}{k} \right) = n - \log_2 \left( \frac{16}{2} \right) = n - 3 \)

---

**Fork and Concurrency:**

Consider this code using Linux's fork:

```c
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
7 8 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.)