CSE 140 - Computer Architecture
Project #2

Documentation

Christopher DeSoto & Aleksandr Brodskiy

April 5, 2018

1.)

```
# define ARRAY_SIZE 64

   int total = 0;
   char * array = malloc(ARRAY_SIZE * sizeof(char));      /* Line 1 */

// Note: chars are 1-byte wide

   for(int x = 0; x < ARRAY_SIZE; x++) array[x] = x;      /* Line 2 */
   for(int x = 0; x < ARRAY_SIZE; x++) total += array[x]; /* Line 3 */
   for(int x = 0; x < ARRAY_SIZE; x++) total += array[x]; /* Line 4
```

a.)
Assume the cache is direct-mapped and array is a pointer to the memory at offset 0b1001111000, what is the ratio of cache hits to cache misses while Line 4 is being executed?

The ratio of cache hits to cache misses is $\frac{48}{16}$
$\therefore$ *there are* 3 *hits for every miss*

b.)
Assume the cache is fully associative and array is a pointer to the memory at offset 0b1001111000, what is the ratio of cache hits to cache misses while Line 4 is being executed?

The ratio of cache hits to cache misses is $\frac{48}{16}$
$\therefore$ *there are* 3 *hits for every miss*

c.)
Assume the cache is direct-mapped and array is a pointer to the memory at offset 0b1001111101, what is the ratio of cache hits to cache misses while Line 4 is being executed?

$$\textit{The ratio of cache hits to cache misses is } \tfrac{47}{17}$$
$$\therefore \textit{ there are } 47 \textit{ hits for every } 17 \textit{ misses}$$

d.)
Assume the cache is fully associative and array is a pointer to the memory at offset 0b1001111101, what is the ratio of cache hits to cache misses while Line 4 is being executed?

$$\textit{The ratio of cache hits to cache misses is } \tfrac{47}{17}$$
$$\therefore \textit{ there are } 47 \textit{ hits for every } 17 \textit{ misses}$$

e.)
Do these results surprise you? Why or why not? Explain what's going on. What would happen if the cache were 2-way set associative? 4-way?

It would be anomalous to classify these results as surprising because the array is stored in contiguous memory. In this manner a fully associative cache cannot be utilized to its full potential. Therefore these results are not surprising.

2.)

Explain the differences between write−through and write−back caches, and when/why one might be preferred over the other.

The underlining difference between a write−through cache and a write−back cache is that the latter demarcate a block as dirty when the cache is modified while the memory is unchanged, while the former will immediately change new data from cache to memory.

In this manner a write−through cache is preferred over the write−back cache when reading operations are performed due to the fact that access to memory is not necessarily constantly obtained during these operations and, overhead would also be prominently decreased with the utilization of the write−through cache.

Therefore a write−back cache is preferable over write-through when writing operations are performed due to the fact that access to access memory is not necessarily obtained for each write operation. Essentially, the preference resides in the fact that, as previously stated, computation overhead is generally decreased.

3.)

Describe a situation in which a 2−way set associative cache would out−perform a fully associative cache. Vice−versa? You should describe these situations in relation to the attributes of the cache. (i.e. accesses are made to fill every block of the cache, hitting memory block−size apart each time...) Otherwise, the examples can be as contrived as you wish.

A 2−way set associative cache would theoretically outperform a fully−associate cache in situations in which the cache size is relatively small.

Essentially this is because there is lower a latency for each access made in regards to a 2−way cache.
As such, considering the contrary, a fully−associative cache will therefore be of more relevant pertinence than in the aforementioned situation.

4.)

Briefly describe the changes you would have to make to your implementations in this project to implement an L2 cache. Assume you're given a second cache struct. How would your L1 cache implementation change? How, if at all, would your L2 cache implementation need to be different from your original cache implementation? **YOU ARE NOT EXPECTED TO IMPLEMENT THIS**.

The implementation of the L1 cache would change in regards to the functionality of L1 Hits in which the L2 cache would be called in all such cases.
If there is a L1 cache miss however, a reciprocated check of the L2 for a hit would be performed; and if a hit were to be detected the L1 cache would be committed.

5.)

Suppose for a moment that virtual memory was included in this project. How would you expect the input to accessMemory() to change? (in other words, what type of information should the cache now receive in order to do its thing?) **YOU ARE NOT EXPECTED TO IMPLEMENT THIS**.

Although the input to accessMemory*() could effectively support more functionality in regards to accessing the virtual memory such as another table or similar data structure, the output of the cache contents would inevitably remain unchanged.*