Direct-mapped Caches

Recall that a *cache* is a *fast* memory device that contains a *small* subset of the data in main memory. Cache data is arranged in *blocks*. We will initially consider caches where the block size is 1 byte, but later we will consider larger block-sizes (to take advantage of *spatial locality*).

In a ***direct mapped*** cache, each memory address maps to a fixed location in the cache called the *index*. Suppose the memory is *byte addressable* and addresses are n bits long. Thus main memory can hold 2^n bytes of data. Suppose the cache can hold 2^m bytes of data, where $m \ll n$. The m least significant bits of the address form the *index*, and the remaining $n - m$ bits form the *tag*.

$n - m$ bits (tag)	m bits (index)

Problem 1: Your friend suggests that the following scheme would be equally appropriate for a direct-mapped cache: the m most significant bits form the index, the remaining $n - m$ bits form the tag. Do you agree? Justify your answer?

Problem 2 (a): A system has 1 GB of byte addressable memory and a 256 KB direct-mapped cache (one byte per block). Identify two distinct memory addresses that map to the *same* location in the cache.

Problem 2 (b): Consider an array `char a[400000]` accessed as follows:

```
for(int i = 0; i < 100; ++i)
  for(int j = 0; j < 400000; ++j)
    a[j]++;
```

What is the pattern of hits and misses you would expect to see, assuming that the cache is devoted *only* to the array `a`, and is initially empty?

Large cache blocks

As we saw in the preceding example, a cache with one byte per block cannot take advantage of *spatial locality*: when we access `a[0]`, the neighboring array elements `a[1]`, `a[2]`, etc. are *not* loaded into the cache automatically. (The only hits in the cache are because of *temporal locality*). To exploit spatial locality, we allow cache blocks to hold several bytes (usually a power of two). On a cache miss, we load *an entire block* of data into the cache.

Consider a ***direct mapped*** cache where the blocksize is 2^b bytes and the cache holds 2^m *blocks*. Once again, suppose memory is byte addressable with n bit addresses. These are interpreted as follows:

$n - m - b$ bits (tag)	m bits (index)	b bits (block offset)

The *index* field identifies the *block*, the *block offset* field identifies the *byte within the block*, and the *tag* field distinguishes between addresses that refer to different blocks but map to the same index.

Problem 3: Explain why the following scheme would *not* be appropriate for a direct-mapped cache: the m least significant bits form the index, the $n - m - b$ most significant bits form the tag, and the remaining bits form the block offset.

Problem 4: A system has 1 GB of byte addressable memory and a 256 KB direct-mapped cache with 8 bytes per block. Consider an array `int a[100000]` accessed as follows (`int` = 4 bytes):

```
for(int i = 0; i < 100; ++i)
  for(int j = 0; j < 100000; ++j)
    a[j]++;
```

What is the pattern of hits and misses you would expect to see, assuming that the cache is devoted *only* to the array `a`, and is initially empty?