Review Problem 40

- A 32-bit CPU has this $2^8$ line DM cache with $2^4$ byte blocks. What memory locations are now held?

<table>
<thead>
<tr>
<th>Valid Bit</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>00001000 F</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>00000002 F</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>00000000</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>00000001</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>00000000</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>00000001</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>00000010</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>00000001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$2^8-1$</td>
<td>0</td>
<td>0000000F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache Index</th>
<th>BS</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00</td>
<td></td>
</tr>
</tbody>
</table>
Block Size Tradeoff

In general, larger block size take advantage of spatial locality **BUT:**
Larger block size means larger miss penalty:
  Takes longer time to fill up the block
If block size is too big relative to cache size, miss rate will go up
  Too few cache blocks

\[
\text{Cache: } \text{Hit Time} + \text{Miss Penalty} \times \text{Miss Rate}
\]

- **Miss Penalty** vs **Block Size**
  - Exploits Spatial Locality
  - Fewer blocks: compromises temporal locality
- **Miss Rate** vs **Block Size**
  - Increased Miss Penalty & Miss Rate
- **Average Access Time** vs **Block Size**
  - Right balance
Direct Mapped Cache Problems

What if regularly used items happen to map to the same cache line? Ex. &\((\text{sum}) = 0\), &\((\text{l}) = 64\), cache is 64 bytes

```
int sum = 0;
...
for (int i=0; i!=N; i++) {
    sum += i;
}
```

Thrashing – Continually loading into cache but evicting it before reuse
Cache Miss Types

Several different types of misses (categorized based on problem/solution)

3 C's of cache design

Compulsory/Coldstart
First access to a block – basically unavoidable (though bigger blocks help)
For long-running programs this is a small fraction of misses

Capacity
The block needed was in the cache, but unloaded because too many other accesses intervened.
Solution is to increase cache size (but bigger is slower, more expensive)

Conflict
The block needed was in the cache, and there was enough room to hold it and all intervening accesses, but blocks mapped to the same location knocked it out.

Solutions
Cache size
Associativity

Invalidate
I/O or other processes invalidate the cache entry

Categorize:
1. Never asked for that block before -> cold
2. Since the last access regarded enough blocks to fill the cache? -> capacity
3. Otherwise conflict.
Fully Associative Cache

No cache index – blocks can be in any cache line

Cache Tag = 57

Tag

Valid Bit

Data

Byte Select = 01
Fully Associative vs. Direct Mapped

No conflict misses
only capacity + coldstart

Significant Hardware overhead - capacity + performance

Wait for hit detection before using the data
DM cache can access data + hit defect in parallel.
N-way Set Associative

N lines are assigned to each cache index
~ N direct mapped caches working in parallel

Direct mapped = 1-way set associative
Fully Associative = $2^N$-way set associative (where $2^N$ is # of cache lines)
2-Way Set Associative Cache

Cache index selects a "set", two tags compared in parallel

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
```

- **Cache Tag = 57**
- **Cache Index = 04**
- **Byte Select = 01**

Diagram:
- Valid Tag Block
- Block Tag Valid
- Addr Cache Tag
- Hit
- Cache Block
N-way vs. Other Caches

Four conflict misses the DM
FA has 0

Four compares than FA

Slower than DM, faster than FA