Review: Direct-Mapped Cache Terminology

- **Index**: specifies the cache index ("row")/block
- **Offset**: specifies which byte within the block we want
- **Tag**: distinguishes between all the memory addresses that map to the same location

Area (cache size, B) = Height (# of blocks) * Width (size of one block, B/block)

**How to Split Cache?**

- Small or large block sizes?
- Large
  - Better spatial locality
  - Too large, misses increase with large penalty

**Caching Terminology**

- When reading memory, 3 things can happen:
  - Cache hit: cache block is valid and contains proper address, so read desired word
  - Cache miss: nothing in cache in appropriate block, so fetch from memory
  - Cache miss, block replacement: wrong data in cache at appropriate block, so discard it and fetch desired data from memory

**Memory Access without Cache**

- Load word instruction: `lw $t0, 0($t1)`
- $t1 contains 1022\text{ten}, Memory[1022] = 99

1. Processor issues address 1022\text{ten} to Memory
2. Memory reads word at address 1022\text{ten} (99)
3. Memory sends 99 to Processor
4. Processor loads 99 into register $t1

**Cache Terms**

- Hit rate: fraction of access that hit in the cache [0.0–1.0]
- Miss rate: 1 – Hit rate
- Miss penalty: time to replace a block from lower level in memory hierarchy to cache
- Hit time: time to access cache memory (including tag comparison)

**Abbreviation**: "$ = cache
Accessing data in a direct mapped cache

- Ex: 16 KiB of data, direct-mapped, 4 word blocks
- Can you work out height, width, area?

Read 0x00000014

Accessing data in a direct mapped cache

- 4 Addresses:
  - 0x00000014, 0x0000001C, 0x00000034, 0x0000008014

4 Addresses divided (for convenience) into
Tag, Index, Byte Offset fields

1. Read 0x00000014

So we read block 1 (0000000001)

No valid data

So load that data into cache, setting tag, valid

Read from cache at offset, return b

2. Read 0x0000001C = 0...00 .001 1100

16 KiB Direct Mapped Cache, 16 blocks

Valid bit determines whether anything is stored in that row
(when computer initially turned on, all entries invalid)
Load that cache block, return word f

3. Read 0x000000034 = 0...0 0.011 0100

So read block 3

Load that cache block, return word f

4. Read 0x00008014 = 0...10 0.001 0100

So read Cache Block 1, Data is Valid

Index Valid, Tag Matches

Index Valid, Tag Matches, return d

No valid data
Cache Block 1 Tag does not match (0 is not 2)

<table>
<thead>
<tr>
<th>Index</th>
<th>Valid</th>
<th>Offset Field 3</th>
<th>Offset Field 2</th>
<th>Offset Field 1</th>
<th>Offset Field 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1022</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1023</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Miss, so replace block 1 with new data & tag

<table>
<thead>
<tr>
<th>Index</th>
<th>Valid</th>
<th>Offset Field 3</th>
<th>Offset Field 2</th>
<th>Offset Field 1</th>
<th>Offset Field 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1022</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1023</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

And return word J

Do an example yourself. What happens?

- Choose from Cache: Hit, Miss, Miss w. replace
- Values returned: a, b, c, d, e, ..., k, l
- Address (hex): 0000000100
- Index = 3, Tag mismatch, so replace from memory.
- Offset = 0, value = e

Answers

- 0x00000030 a hit
  - Index = 3, Tag matches, Offset = 0, value = e

- 0x00000010 a miss
  - Index = 1, Tag mismatch, so replace from memory.
  - Offsets = 0xc, value = d

- Since reads, values must = memory values whether or not cached:
  - 0x00000030 = e
  - 0x00000010 = d

Multiword-Block Direct-Mapped Cache

Peer Instruction

1) Memory hierarchies were invented before 1950. (UNIVAC I wasn’t delivered until 1951)
2) All caches take advantage of spatial locality.
3) All caches take advantage of temporal locality.

Peer Instruction

A. For a given cache size: a larger block size can cause a lower hit rate than a smaller one.
B. If you know your computer’s cache size, you can often make your code run faster.
C. Memory hierarchies take advantage of spatial locality by keeping the most recent data items closer to the processor.

And in Conclusion...

- Mechanism for transparent movement of data among levels of a storage hierarchy:
  - set of address/value bindings
  - address = index to set of candidates
  - compare desired address with tag
  - service hit or miss
  - load new block and binding on miss