Improving our Simple Cache

1. How to handle a write?
2. Efficient Bit Manipulation
3. How to eliminate even more conflicts?
4. Can hierarchy help?

Issue #1: What to do on a write?

<table>
<thead>
<tr>
<th>Memory</th>
<th>Cache (N = 5)</th>
<th>Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>7</td>
<td>1. Read 24</td>
</tr>
<tr>
<td>21</td>
<td>3</td>
<td>2. Write 24</td>
</tr>
<tr>
<td>22</td>
<td>27</td>
<td>3. Read 26</td>
</tr>
<tr>
<td>23</td>
<td>32</td>
<td>4. Write 25</td>
</tr>
<tr>
<td>24</td>
<td>101</td>
<td>5. Write 24</td>
</tr>
<tr>
<td>25</td>
<td>78</td>
<td>6. Write 29</td>
</tr>
</tbody>
</table>

Comparing Write Strategies

- Write-through:
- Write-back

- How to improve write-through?
Issue #2: Efficient Bit Manipulation

Given cache with 8 bytes per block, N = 16, what is index of address “153”?

OLD: Index = \frac{\text{Address}}{\text{BytesPerBlock}} \mod N

NEW: (assuming dealing with powers-of-2)
a. Express in binary. (153\text{_{10}} = 99\text{_{16}})
b. Grab the right bits!
   ByteOffset =
   Index =

Example #1: Bit Manipulation

1. Suppose cache has:
   - 8 byte blocks
   - 256 blocks
   Show how to break the following address into the tag, index, & byte offset.
   0000 1000 0101 1100 0001 0001 0111 1001

2. Same cache, but now 4-way associative. How does this change things?
   0000 1000 0101 1100 0001 0001 0111 1001

Real Cache with Efficient Bit Manipulation

Suppose a direct-mapped cache divides addresses as follows:

<table>
<thead>
<tr>
<th>21 bits</th>
<th>7 bits</th>
<th>4 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>index</td>
<td>byte offset</td>
</tr>
</tbody>
</table>

Example #2: Bit Manipulation

What is the block size?

The number of blocks?

Total size of the cache?
(usually refers to size of data only)
Key Rules

• How do the # sets and # blocks relate?

• Calculate # index bits from # sets

• One hex ‘digit’ = 4 bits
  – 0x1234 = 0001 0010 0011 0100

Issue #3: How to eliminate even more conflicts?

• Fully associative cache – cache block can go ____________ in cache

• Pros

• Cons

• Can view all caches as n-way associative:
  – Direct-mapped,  n =
  – 4-way associative,  n =
  – Fully associative,  n =

Issue #4: More hierarchy – L2 cache?

• Add a second level cache:
  – often primary cache is on the same chip as the processor
  – use SRAMs to add another cache above primary memory (DRAM)
  – miss penalty goes down if data is in 2nd level cache

• Performance smarts:
  – try and optimize the ____________ on the 1st level cache

  – try and optimize the ____________ on the 2nd level cache

Memory Hierarchy
Questions

• Will the miss rate of a L2 cache be higher or lower than for the L1 cache?

• Claim: “The register file is really the highest level cache”
  What are reasons in favor and against this statement?