Improving our Simple Cache

1. How to handle a write?
2. Efficient Bit Manipulation
3. How to handle a miss?
4. How to eliminate even more conflicts?
5. 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**

OLD: \[ \text{Index} = \left\lfloor \frac{\text{ByteAddress}}{\text{BytesPerBlock}} \right\rfloor \mod N \]

Example:
- BytesPerBlock = 8
- \( N = 16 \)

How t
New: \( \text{ByteOffset} = \)
- Index =

Example:
- Address \( 0000\ 1000\ 0101\ 1100\ 0001\ 0001\ 0111\ 1001 \)

---

**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**

Example #2: Bit Manipulation

Suppose a direct-mapped cache divides addresses as follows:

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

What is the block size?

The number of blocks?

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

- How the # sets and # blocks relate?
- Calculate # index bits from # sets
- One hex ‘digit’ = 4 bits
  - \(0x1234 = 0001\ 0010\ 0011\ 0100\)

Exercise #1

Suppose a cache divides addresses as follows:

<table>
<thead>
<tr>
<th>tag</th>
<th>index</th>
<th>byte offset</th>
</tr>
</thead>
</table>

Fill in the values for a direct-mapped or 4-way associative cache:

<table>
<thead>
<tr>
<th></th>
<th>Direct-mapped</th>
<th>4-way associative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of blocks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total size of cache</td>
<td>(e.g. 32 * 128 – don’t have to multiply out)</td>
<td></td>
</tr>
<tr>
<td>Tag size (# bits)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2

1. Suppose cache has:
   - 4 byte blocks
   - 128 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 8-way associative. How does this change things?
   \(0000\ 1000\ 0101\ 1100\ 0001\ 0001\ 0111\ 1001\)

Exercise #3

- Given a cache that is:
  - 4-way associative
  - 32 blocks
  - 16 byte block size
   What is the cache index and byte offset for the following address:
   \(0x3ab12395\)

<table>
<thead>
<tr>
<th>Cache index</th>
<th>Byte offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

And this one:

\(0x70ff1213\)

<table>
<thead>
<tr>
<th>Cache index</th>
<th>Byte offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Do these addresses conflict in the cache?
Exercise #4

- Cache parameters are often a power of two. Given what you know so far, explain why each of the following should be a power of two (or need not be).
  - Block size
  - Number of cache blocks
  - Number of cache sets
  - Associativity

Exercise #5

- What is the total number of bits needed to implement the storage for the direct mapped cache given in Exercise #2. What do you need these bits for besides the data?

Issue #3: How to handle a miss?

- Things we need to do:
  1. _____________ the CPU until miss completes
  2. _____________ old data from the cache
     Which data?
  3. _____________ the needed data from memory
     Pay the _____________
     How long does this take?
  4. _____________ the CPU

What about a write miss?

Decreasing the Miss Penalty

- Time to fetch data from memory (with sample times) = SendAddress (1 bus cycle) + Initiate DRAM Access (15 bus cycles per word read) + Bus transfer time (1 bus cycle per word)
- How can we decrease this?
Issue #4: 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 #5: 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

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?
More Questions

• How else might you might improve the performance of a cache. Think about either:
  – Improving hit time
  – Improving the hit rate
  – Decreasing miss penalty