Caches III
CSE 351 Winter 2020

Instructor:
Ruth Anderson

Teaching Assistants:
Jonathan Chen
Justin Johnson
Porter Jones
Josie Lee
Jeffery Tian
Callum Walker
Eddy (Tianyi) Zhou

https://what-if.xkcd.com/111/
Administrivia

- hw15 due Friday (2/21)
- Lab 3 due Monday (2/24)
- hw16 due Monday (2/24)
- hw17 due Wednesday (2/26)
Making memory accesses fast!

- Cache basics
- Principle of locality
- Memory hierarchies
- Cache organization
  - Direct-mapped (sets; index + tag)
  - Associativity (ways)
  - Replacement policy
  - Handling writes
- Program optimizations that consider caches
Review: Direct-Mapped Cache

Hash function: \((\text{block number}) \mod (\# \text{ of blocks in cache})\)
- Each memory address maps to \textit{exactly} one index in the cache
- Fast (and simpler) to find a block

Here \(K = 4 \text{ B}\) and \(C/K = 4\)
Direct-Mapped Cache Problem

What happens if we access the following addresses?
- 8, 24, 8, 24, 8, ...?
- Conflict in cache (misses!)
- Rest of cache goes unused

Solution?

Here $K = 4 \text{ B}$ and $C/K = 4$
Associativity

- What if we could store data in any place in the cache?
  - More complicated hardware = more power consumed, slower

- So we combine the two ideas:
  - Each address maps to exactly one set
  - Each set can store block in more than one way

- 1-way: 8 sets, 1 block each
- 2-way: 4 sets, 2 blocks each
- 4-way: 2 sets, 4 blocks each
- 8-way: 1 set, 8 blocks

<table>
<thead>
<tr>
<th>Set</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-way</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-way</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4-way</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8-way</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cache Organization (3)

- **Associativity** \( (E) \): # of ways for each set
  - Such a cache is called an “\( E \)-way set associative cache”
  - We now index into cache *sets*, of which there are \( S = C/K/E \)
  - Use lowest \( \log_2(C/K/E) = s \) bits of block address
    - Direct-mapped: \( E = 1 \), so \( s = \log_2(C/K) \) as we saw previously
    - Fully associative: \( E = C/K \), so \( s = 0 \) bits

Used for tag comparison  Selects the set  Selects the byte from block

Tag \( (t) \)  Index \( (s) \)  Offset \( (k) \)

Decreasing associativity  Increasing associativity
Direct mapped (only one way)  Fully associative (only one set)

**Note:** The textbook uses “b” for offset bits
Example Placement

- Where would data from address \(0x1833\) be placed?
  - Binary: \(0b\ 0001\ 1000\ 0011\ 0011\)

- \(m\)-bit address:
  - Tag \((t)\)
  - Index \((s)\)
  - Offset \((k)\)

\[
t = m - s - k \quad s = \log_2(C/K/E) \quad k = \log_2(K)
\]

- \(s = ?\)
  - Direct-mapped

- \(s = ?\)
  - 2-way set associative

- \(s = ?\)
  - 4-way set associative

- \(\text{block size:} \ 16\ B\)
- \(\text{capacity:} \ 8\ \text{blocks}\)
- \(\text{address:} \ 16\ \text{bits}\)
Block Replacement

- Any empty block in the correct set may be used to store block
- If there are no empty blocks, which one should we replace?
  - No choice for direct-mapped caches
  - Caches typically use something close to least recently used (LRU) (hardware usually implements “not most recently used”)

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Polling Question

- We have a cache of size 2 KiB with block size of 128 B. If our cache has 2 sets, what is its associativity?
  - Vote at [http://pollev.com/rea](http://pollev.com/rea)
  - A. 2
  - B. 4
  - C. 8
  - D. 16
  - E. We’re lost...

- If addresses are 16 bits wide, how wide is the Tag field?
General Cache Organization \((S, E, K)\)

- \(E\) = blocks (or lines) per set
- \(S\) sets = \(2^s\)
- Cache size: \(C = K \times E \times S\) data bytes (doesn’t include \(V\) or \(Tag\))

\(V\) = valid bit
\(K\) = bytes per block
Notation Review

- We just introduced a lot of new variable names!
  - Please be mindful of block size notation when you look at past exam questions or are watching videos

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Variable</th>
<th>Formulas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block size</td>
<td>$K$ ($B$ in book)</td>
<td>$M = 2^m \leftrightarrow m = \log_2 M$</td>
</tr>
<tr>
<td>Cache size</td>
<td>$C$</td>
<td></td>
</tr>
<tr>
<td>Associativity</td>
<td>$E$</td>
<td>$S = 2^s \leftrightarrow s = \log_2 S$</td>
</tr>
<tr>
<td>Number of Sets</td>
<td>$S$</td>
<td>$K = 2^k \leftrightarrow k = \log_2 K$</td>
</tr>
<tr>
<td>Address space</td>
<td>$M$</td>
<td>$C = K \times E \times S$</td>
</tr>
<tr>
<td>Address width</td>
<td>$m$</td>
<td>$s = \log_2(C/K/E)$</td>
</tr>
<tr>
<td>Tag field width</td>
<td>$t$</td>
<td>$m = t + s + k$</td>
</tr>
<tr>
<td>Index field width</td>
<td>$s$</td>
<td></td>
</tr>
<tr>
<td>Offset field width</td>
<td>$k$ ($b$ in book)</td>
<td></td>
</tr>
</tbody>
</table>
Example Cache Parameters Problem

- 4 KiB address space, 125 cycles to go to memory. Fill in the following table:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Size</td>
<td>256 B</td>
</tr>
<tr>
<td>Block Size</td>
<td>32 B</td>
</tr>
<tr>
<td>Associativity</td>
<td>2-way</td>
</tr>
<tr>
<td>Hit Time</td>
<td>3 cycles</td>
</tr>
<tr>
<td>Miss Rate</td>
<td>20%</td>
</tr>
<tr>
<td>Tag Bits</td>
<td></td>
</tr>
<tr>
<td>Index Bits</td>
<td></td>
</tr>
<tr>
<td>Offset Bits</td>
<td></td>
</tr>
<tr>
<td>AMAT</td>
<td></td>
</tr>
</tbody>
</table>
Cache Read

1) Locate set
2) Check if any line in set is valid and has matching tag: hit
3) Locate data starting at offset

Address of byte in memory:

\[ \text{Address} = \text{tag bits} \, \text{s bits} \, \text{k bits} \]

Data begins at this offset

\[ S = \text{# sets} = 2^s \]

\[ E = \text{blocks/lines per set} \]

\[ K = \text{bytes per block} \]
Example: Direct-Mapped Cache ($E = 1$)

Direct-mapped: One line per set
Block Size $K = 8$ B

$S = 2^s$ sets

Address of `int`:

$\begin{array}{c}
\text{Address of `int`:} \\
0...01 100
\end{array}$

Find set
Example: Direct-Mapped Cache ($E = 1$)

Direct-mapped: One line per set
Block Size $K = 8$ B

Address of int:

| t bits | 0...01 | 100 |

Block offset

Valid? + Match?: yes = hit
**Example: Direct-Mapped Cache ($E = 1$)**

Direct-mapped: One line per set  
Block Size $K = 8$ B

No match? Then old line gets evicted and replaced
Example: Set-Associative Cache ($E = 2$)

2-way: Two lines per set
Block Size $K = 8$ B

Address of `short int`:

| t bits | 0...01 | 100 |

find set
Example: Set-Associative Cache ($E = 2$)

2-way: Two lines per set
Block Size $K = 8$ B

Address of `short int`:

```
0...01 100
```

Compare both

Valid? +
Match: yes = hit

Block offset
### Example: Set-Associative Cache \(E = 2\)

2-way: Two lines per set
Block Size \(K = 8\) B

<table>
<thead>
<tr>
<th>V</th>
<th>Tag</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
</table>

Valid? + Match: yes = hit

Address of short int:

![bits]

\[0 \ldots 01\]
\[100\]

Block offset

\[\text{short int (2 B) is here}\]

No match?
- One line in set is selected for eviction and replacement
- Replacement policies: random, least recently used (LRU), ...
Types of Cache Misses: 3 C’s!

- **Compulsory** (cold) miss
  - Occurs on first access to a block

- **Conflict** miss
  - Conflict misses occur when the cache is large enough, but multiple data objects all map to the same slot
    - *e.g.* referencing blocks 0, 8, 0, 8, ... could miss every time
  - Direct-mapped caches have more conflict misses than $E$-way set-associative (where $E > 1$)

- **Capacity** miss
  - Occurs when the set of active cache blocks (the *working set*) is larger than the cache (just won’t fit, even if cache was *fully-associative*)
  - **Note:** *Fully-associative* only has Compulsory and Capacity misses
Example Code Analysis Problem

- Assuming the cache starts \textit{cold} (all blocks invalid) and \texttt{sum}, \texttt{i}, and \texttt{j} are stored in registers, calculate the \textbf{miss rate}:
  
  \begin{itemize}
  \item $m = 12$ bits, $C = 256$ B, $K = 32$ B, $E = 2$
  \end{itemize}

```c
#define SIZE 8
long ar[SIZE][SIZE], sum = 0; // &ar=0x800
for (int i = 0; i < SIZE; i++)
  for (int j = 0; j < SIZE; j++)
    sum += ar[i][j];
```