Memory & Caches III
CSE 351 Autumn 2022

Instructor: Justin Hsia

Teaching Assistants:
Angela Xu
Assaf Vayner
David Dai
James Froelich
Paul Stevans
Arjun Narendra
Carrie Hu
Dominick Ta
Jenny Peng
Renee Ruan
Armin Magness
Clare Edmonds
Effie Zheng
Kristina Lansang
Vincent Xiao

http://xkcd.com/908/
 Relevant Course Information

- Lab 3 due Friday (11/11)
- Lab 4 released Monday, due after Thanksgiving
  - Can do Part 1 after today; will need Lecture 19 to do Part 2
- hw17 due Wednesday (11/16)
  - Covers the major cache mechanics – BIG homework
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
Reading Review

❖ Terminology:
  - Associativity: sets, fully-associative cache
  - Replacement policies: least recently used (LRU)
  - Cache line: cache block + management bits (valid, tag)
  - Cache misses: compulsory, conflict, capacity

❖ Questions from the Reading?
Review: Direct-Mapped Cache

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

- Here \( K = 4 \text{ B} \) and \( C/K = 4 \)

<table>
<thead>
<tr>
<th>Memory</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Num</td>
<td>Block Data</td>
</tr>
<tr>
<td>00 00</td>
<td></td>
</tr>
<tr>
<td>00 01</td>
<td></td>
</tr>
<tr>
<td>00 10</td>
<td></td>
</tr>
<tr>
<td>00 11</td>
<td></td>
</tr>
<tr>
<td>01 00</td>
<td></td>
</tr>
<tr>
<td>01 01</td>
<td></td>
</tr>
<tr>
<td>01 10</td>
<td></td>
</tr>
<tr>
<td>01 11</td>
<td></td>
</tr>
<tr>
<td>10 00</td>
<td></td>
</tr>
<tr>
<td>10 01</td>
<td></td>
</tr>
<tr>
<td>10 10</td>
<td></td>
</tr>
<tr>
<td>10 11</td>
<td></td>
</tr>
<tr>
<td>11 00</td>
<td></td>
</tr>
<tr>
<td>11 01</td>
<td></td>
</tr>
<tr>
<td>11 10</td>
<td></td>
</tr>
<tr>
<td>11 11</td>
<td></td>
</tr>
</tbody>
</table>
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 $K = 4$ 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
Cache Organization (3)

- **Associativity** ($E$): # of ways for each set
  - Such a cache is called an “$E$-way set associative cache”
  - We index into cache sets, of which there are $S = (C/K)/E$
  - Use lowest $\log_2(S) = s$ 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

---

**Diagram: Tag, Index, Offset**

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

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

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

❖ Where would data from address \(0x1833\) be placed?

- **Binary**: \(0b\ 0001\ 1000\ 0011\ 0011\)

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

**Block Structure**

- **Block size:** 16 B
- **Capacity:** 8 blocks
- **Address:** 16 bits

\[
\text{Index} (s) \quad \text{Tag (t)} \quad \text{Offset} (k)
\]

<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>

**Direct-mapped**

\[s = ?\]

**2-way set associative**

\[s = ?\]

**4-way set associative**

\[s = ?\]
Block Placement and Replacement

- Any empty block in the correct set may be used to store block
  - **Valid bit** for each cache block indicates if valid (1) or mystery (0) data
- 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>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

❖ 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 in Ed Lessons

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 \((S, E, K)\)

\[E = \text{blocks (or lines) per set}\]

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

\[\text{Cache size: } C = K \times E \times S \text{ data bytes (doesn’t include } V \text{ or Tag)}\]

\[K = \text{bytes per block}\]
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>
</table>
| Block size         | $K$ ($B$ in book) | $M = 2^m \leftrightarrow m = \log_2 M$
| Cache size         | $C$      | $S = 2^s \leftrightarrow s = \log_2 S$
|                    |          | $K = 2^k \leftrightarrow k = \log_2 K$                                  |
| Associativity      | $E$      | $C = K \times E \times S$                                               |
| Number of Sets     | $S$      | $s = \log_2 (C/K/E)$                                                   |
| Address space      | $M$      | $m = t + s + k$                                                         |
| Address width      | $m$      |                                                                          |
| Tag field width    | $t$      |                                                                          |
| Index field width  | $s$      |                                                                          |
| Offset field width | $k$ ($b$ in book) |                                                                          |
Example Cache Parameters Problem

- 1 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>64 B</td>
</tr>
<tr>
<td>Block Size</td>
<td>8 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

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

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

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:

- \( t \) bits
- \( s \) bits
- \( k \) bits

- Tag
- Set index
- Block offset

\( K = \text{bytes per block} \)

\( V \) is the valid bit.
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
```

$S = 2^s$ sets
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

This is why we want alignment!
Example: Set-Associative Cache ($E = 2$)

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

Address of `short int`:

```
| addr | 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`:

\[ t \text{ bits} \downarrow \begin{array}{c} 0 \ldots 1 \uparrow \end{array} 100 \]

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

valid? + match: yes = hit

\( \text{Address of short int:} \)

\( t \) bits 0...01 100

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

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 **cold** (all blocks invalid) and sum, i, and j are stored in registers, calculate the **miss rate**:
  - \( m = 10 \) bits, \( C = 64 \) B, \( K = 8 \) B, \( E = 2 \)

```c
#define SIZE 8
short ar[SIZE][SIZE], sum = 0;  // &ar=0x200
for (int i = 0; i < SIZE; i++)
    for (int j = 0; j < SIZE; j++)
        sum += ar[j][i];
```