Recommended 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.

![Diagram showing memory and cache mapping](image)

<table>
<thead>
<tr>
<th>Block Num</th>
<th>Block Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>00</td>
<td>10</td>
</tr>
<tr>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
</tr>
<tr>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
</tr>
<tr>
<td>11</td>
<td>01</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

Here $K = 4$ 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 $K = 4$ 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

<table>
<thead>
<tr>
<th>Set</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-way:</td>
<td>1 set, 8 blocks</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-way:</td>
<td>2 sets, 4 blocks each</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-way:</td>
<td>4 sets, 2 blocks each</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-way:</td>
<td>8 sets, 1 block each</td>
</tr>
</tbody>
</table>

Diagram:
- Direct-mapped
- Fully associative
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

---

Note: The textbook uses “b” for offset bits

---

Diagram:

- **Tag** ($t$) — Used for tag comparison
- **Index** ($s$) — Selects the set
- **Offset** ($k$) — Selects the byte from block

**Decreasing associativity**
- Direct mapped (only one way)

**Increasing associativity**
- Fully associative (only one set)
Example Placement

❖ Where would data from address 0x1833 be placed?

- **Binary:** 0b 0001 1000 0011 0011

\[ t = m - s - k \]

\[ s = \log_2\left(\frac{C}{K/E}\right) \]

\[ k = \log_2(K) \]

\( m \)-bit address:

<table>
<thead>
<tr>
<th>Tag (t)</th>
<th>Index (s)</th>
<th>Offset (k)</th>
</tr>
</thead>
</table>

**Direct-mapped**

- **Set**
  - 0
  - 1
  - 2
  - 3
  - 4
  - 5
  - 6
  - 7

**Tag (t)**

- (000)
- (001)
- (010)
- (011)
- (100)
- (101)
- (110)
- (111)

**Data**

- ✓

**2-way set associative**

- **Set**
  - 0
  - 1

**Tag (t)**

- (000)
- (001)

**Data**

- ✓

**4-way set associative**

- **Set**
  - 0
  - 1
  - 2
  - 3

**Tag (t)**

- (000)
- (001)
- (010)
- (011)
- (100)
- (101)
- (110)
- (111)

**Data**

- ✓
  - ✓
  - ✓

**Set size:** 16 B
**Capacity:** 8 blocks
**Address:** 16 bits
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”)
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? $k = \log_2(K) = 7$ bits, $s = \log_2(S) = 1$ bit, $t = m - s - k = 8$ bits
General Cache Organization \( (S, E, K) \)

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

\[ V \text{ Tag} 0 1 2 \ldots K-1 \]

valid bit

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

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cache Size</strong></td>
<td>64 B</td>
</tr>
<tr>
<td><strong>Block Size</strong></td>
<td>8 B</td>
</tr>
<tr>
<td><strong>Associativity</strong></td>
<td>2-way</td>
</tr>
<tr>
<td><strong>Hit Time</strong></td>
<td>3 cycles</td>
</tr>
<tr>
<td><strong>Miss Rate</strong></td>
<td>20%</td>
</tr>
<tr>
<td><strong>Tag Bits</strong></td>
<td>5</td>
</tr>
<tr>
<td><strong>Index Bits</strong></td>
<td>2</td>
</tr>
<tr>
<td><strong>Offset Bits</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>AMAT</strong></td>
<td>28 clock cycles</td>
</tr>
</tbody>
</table>

\[ \begin{align*}
    \text{AMAT} &= \text{HT} + \text{MR} \times \text{MP} \\
    t &= m - s - k \\
    s &= \log_2(C/K/E) \\
    k &= \log_2(C) \\
    \text{MP} &= 2^{10} B \iff m = 10 \text{ bits}
\end{align*} \]
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

$E = \text{blocks/lines per set}$

$S = \# \text{sets} = 2^s$

$V$ Tag $0 1 2 \ldots \ldots K-1$

valid bit

$K = \text{bytes per block}$

Address of byte in memory:

<table>
<thead>
<tr>
<th>bits</th>
<th>bits</th>
<th>bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>set</td>
<td>block</td>
</tr>
</tbody>
</table>

data begins at this offset
Example: Direct-Mapped Cache \((E = 1)\)

Direct-mapped: One line per set
Block Size \(K = 8\) B

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

\[ \text{Address of } \text{int:} \]

```
<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>
<tbody>
<tr>
<td>set 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set (S-1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Block Size \(K = 8\) B

8B in block

\[ \text{find set} \]
Example: Direct-Mapped Cache ($E = 1$)

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

Address of `int`:

- Tag
- Block offset

valid? + match?: yes = hit

Block Size $K = 8$ B
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:

\[\begin{array}{c|c}
\text{bits} & 0...01 & 100 \\
\hline
\text{set 0} & \text{find set} & \\
\text{set 1} & & \\
\text{set 2} & &
\end{array}\]
Example: Set-Associative Cache \((E = 2)\)

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

Valid? +
Match: yes = hit

Address of `short int`:
- \(t\) bits
- 0…01
- 100

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

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

Address of `short int`:

\[
\begin{array}{l}
\text{bits} \\
0...01 \\
100
\end{array}
\]

valid? + match: yes = hit

No match?
- One line in set is selected for eviction and replacement
- Replacement policies: random, least recently used (LRU), ...

short int (2 B) is here
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];
```

### Diagram
- **Block**: 4 shorts
- **Cache**:
  - 4 elements of a row of the matrix
  - Jumping by a row skips two block numbers (and sets)
  - Cache block holds 4 elements of a row of the matrix

### Table
<table>
<thead>
<tr>
<th>Address</th>
<th>Offset</th>
<th>Index</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b 10 0000 0000</td>
<td>M</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>0b 10 0010 0000</td>
<td>M</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>0b 10 0111 0000</td>
<td>M</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>0b 10 0100 0000</td>
<td>M</td>
<td>00</td>
<td>01</td>
</tr>
</tbody>
</table>

- (1st way) (1st way)
- (2nd way) (2nd way)
- (replacement)