I'm sorry, we can't approve this permit. Your land isn't zoned for giant-money-bin construction.

Also, you're a duck.
Administrivia

- Questions doc: [https://tinyurl.com/CSE351-7-31](https://tinyurl.com/CSE351-7-31)

- hw16 due Wednesday (8/5) – 10:30am
  
  long, start early!

- Lab 3 due Tonight (7/31) – 11:59pm
  - You get to write some buffer overflow exploits!

- Lab 4 released later today
  - All about caches!

- Unit Summary 2 Due next Wednesday (8/5) – 11:59pm

  Course staff 1-on-1s – see Piazza
Making memory accesses fast!

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

- **Block size (K):** basic unit of transfer between memory and the cache, given in bytes (e.g. 64 B).
- **Cache size (C):** Total amount of data that can be stored in the cache, given in bytes (e.g. 32 KiB).
  - Must be multiple of block size
  - Number of blocks in cache is calculated by C/K
- **Associativity (E):** Number of ways blocks can be stored in a cache set, or how many blocks in each set
- **Number of sets (S):** Number of unique sets that blocks can be placed into in a cache (calculated as C/K/E).
Review: TIO address breakdown

- TIO address breakdown:

  \[ \text{Tag} \quad \text{Index} \quad \text{Offset} \]

  \[
  m \text{-bit address: } \begin{array}{c}
  \text{Tag } (t) \\
  \text{Index } (s) \\
  \text{Offset } (k)
  \end{array}
  \]

  Block Number

- **Index** \((s)\) field tells you where to look in cache
  - Number of bits is determined by number of sets \((\log_2 (C/K/E)) = \log_2 (s)\)
  - Need enough bits to reference every set in the cache

- **Tag** \((t)\) field lets you check that data is the block you want
  - Rest of the bits not used for index and offset \((m - s - k)\)

- **Offset** \((k)\) field selects specified start byte within block
  - Number of bits is determined by block size \((\log_2 (K))\)
  - Need enough bits to reference every byte in a block
Review: Cache Lookup Process

- CPU requests data at a given address
- Cache breaks down address into different bit fields
  - Determines offset, index, and tag bits
- Cache checks to see if block containing address is already in the cache
  - Uses index bits to find which set to look in
  - Uses tag bits to make sure the block in the set matches

- If block is in the cache, it’s a **cache hit**
  - Data is returned to CPU starting at byte offset
- If block is not in the cache, it’s a **cache miss**
  - Block is loaded from memory into the cache, evicting other blocks from the cache if necessary
  - Data is returned to CPU starting at byte offset
## Review: Direct-Mapped Cache

### Hash function:

\[(\text{block number}) \mod (\# \text{ of blocks in cache})\]

- Each memory address maps to exactly one index in the cache.
- Fast (and simpler) to find a block.

### Example

<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>00 00 00</td>
</tr>
<tr>
<td>00 01</td>
<td>00 01 11</td>
</tr>
<tr>
<td>00 10</td>
<td>00 10 01</td>
</tr>
<tr>
<td>00 11</td>
<td>00 11 01</td>
</tr>
<tr>
<td>01 00</td>
<td>01 00 11</td>
</tr>
<tr>
<td>01 01</td>
<td>01 01 01</td>
</tr>
<tr>
<td>01 10</td>
<td>01 10 10</td>
</tr>
<tr>
<td>01 11</td>
<td>01 11 10</td>
</tr>
<tr>
<td>10 00</td>
<td>10 00 11</td>
</tr>
<tr>
<td>10 01</td>
<td>10 01 01</td>
</tr>
<tr>
<td>10 10</td>
<td>10 10 01</td>
</tr>
<tr>
<td>10 11</td>
<td>10 11 10</td>
</tr>
<tr>
<td>11 00</td>
<td>11 00 10</td>
</tr>
<tr>
<td>11 01</td>
<td>11 01 11</td>
</tr>
<tr>
<td>11 10</td>
<td>11 10 01</td>
</tr>
<tr>
<td>11 11</td>
<td>11 11 01</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?
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

![Diagram showing different ways of mapping addresses to cache sets](image-url)
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

**Note:** The textbook uses "b" for offset bits

---

**Diagram:***
- Used for tag comparison
- Selects the set
- Selects the byte from block

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

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

  \[
  \begin{align*}
  t &= m - s - k \\
  s &= \log_2(C/K/E) \\
  k &= \log_2(K) = 4
  \end{align*}
  \]

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

  \[
  E=1,\ s=\frac{C}{K/E} = 8 \\
  s = \log_2(8) = 3
  \]
  - Direct-mapped

  \[
  E=2,\ s=\frac{C}{K/E} = 4 \\
  s = ?\log_2(4) = 2
  \]
  - 2-way set associative

  \[
  E=4,\ s=\frac{C}{K/E} = 2 \\
  s = ?\log_2(2) = 1
  \]
  - 4-way set associative

**Table: Direct-Mapped Cache**

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>3</td>
<td>✔</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

**Table: 2-Way Set Associative Cache**

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>✔</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>✔</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>✔</td>
</tr>
<tr>
<td>011</td>
<td>3</td>
<td>✔</td>
</tr>
</tbody>
</table>

**Table: 4-Way Set Associative Cache**

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>✔</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>✔</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>✔</td>
</tr>
<tr>
<td>011</td>
<td>3</td>
<td>✔</td>
</tr>
</tbody>
</table>
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>

Direct-mapped

<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>

2-way set associative

<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>
</tbody>
</table>

4-way set associative

<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>
</tbody>
</table>
Polling Question [Cache III]

- 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/pbjones](http://pollev.com/pbjones)
  - 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 \((S, E, K)\)

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

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

**Cache size:**

\[ C = K \times E \times S \text{ data bytes} \]

\((\text{doesn't include V or Tag})\)
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) | \[
M = 2^m \leftrightarrow m = \log_2 M \\
S = 2^s \leftrightarrow s = \log_2 S \\
K = 2^k \leftrightarrow k = \log_2 K
\] |
| Cache size            | \( C \)          |                                           |
| Associativity         | \( E \)          |                                           |
| Number of Sets        | \( S \)          |                                           |
| Address space         | \( M \)          |                                           |
| Address width         | \( m \)          |                                           |
| Tag field width       | \( t \)          |                                           |
| Index field width     | \( s \)          |                                           |
| Offset field width    | \( k \) (\( b \) in book) | \[
C = K \times E \times S \\
s = \log_2 (C/K/E) \\
m = t + s + k
\] |
Example Cache Parameters Problem

- 4 KiB address space, 125 cycles to go to memory.

Fill in the following table:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Problem</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>$12 - 2 - 5 = 5$ bits</td>
</tr>
<tr>
<td>Index Bits</td>
<td>$\log_2(4) = 2$ bits</td>
</tr>
<tr>
<td>Offset Bits</td>
<td>$\log_2(32) = 5$ bits</td>
</tr>
<tr>
<td>AMAT</td>
<td>$3 + 2(125) = 28$ cycles</td>
</tr>
</tbody>
</table>

\[ \begin{align*}
C &= 2^{10} \\
K &= 2^{4} \\
E &= 2^{2} \\
M &= \log_2(M) \\
m &= \log_2(C/K/E) \\
k &= \log_2(K) \\
C/K/E &= 8/2 = 4 \\
2^8 &= 256 \\
2^5 &= 32 \\
2^2 &= 4 \\
HT &= 2 \\
MR &= 2 \\
AMAT &= 3 + 2(125) = 28 \text{ cycles}
\end{align*} \]
Cache Read

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

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

Address of byte in memory:
- \( t \) bits
- \( s \) bits
- \( k \) bits

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

valid bit

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

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

Address of \texttt{int}:

\[\begin{array}{c|c}
\text{bits} & 00100 \\
\hline
\end{array}\]

\(S = 2^s\) sets

\[\begin{array}{c|c|c|c|c|c|c|c}
\text{V} & \text{Tag} & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
\text{set 0} & & & & & & & \\
\hline
\text{set 1} & & & & & & & \\
\hline
\text{set 2} & & & & & & & \\
\hline
\text{...} & & & & & & & \\
\hline
\text{set} S-1 & & & & & & & \\
\end{array}\]
Example: Direct-Mapped Cache ($E = 1$)

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

Valid? + match?: yes = hit

Address of `int`: 0...01 100

Block offset
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`:

| 0...01 | 100 |

2 blocks per set

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

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

Address of short int:

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

Address of short int: 0...01 100

Valid? + match: yes = hit

Compare both

Block offset

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 \( \text{sum} \), \( i \), and \( j \) are stored in registers, calculate the **miss rate**:
  - \( m = 12 \) bits, \( C = 256 \) B, \( K = 32 \) B, \( E = 2 \)
  - \( S = \frac{C}{K/E} = 4 \)

```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];
```

![Cache Diagram](cache_diagram.png)
Notes Diagrams

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

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

$V$ (valid bit)

$K = \text{bytes per block}$

$\text{Set} \quad \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}$

$\text{Set} \quad \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}$

$\text{Set} \quad \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}$

$\text{Set} \quad \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}$

$\text{direct-mapped}$

$\text{2-way: 4 sets, 2 blocks each}$

$\text{4-way: 2 sets, 4 blocks each}$

$\text{8-way: 1 set, 8 blocks}$

$\text{fully associative}$