Caches III
CSE 351 Autumn 2017

Instructor: Justin Hsia
Teaching Assistants: Lucas Wotton, Michael Zhang, Parker DeWilde, Ryan Wong, Sam Gehman, Sam Wolfson, Savanna Yee, Vinny Palaniappan

Administrivia
- Midterm regrade requests due end of tonight
- Lab 3 due Friday
- HW 4 is released, due next Friday (11/17)
- No lecture on Friday – Veteran’s Day!

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

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

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 \( C/K/E \)
  - 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

Example Placement
- Where would data from address 0x1833 be placed?

Diagram notes:
- Use lowest \( \log_2(C/K/E) = s \) bits of block address
- \( m \)-bit address: Tag (\( t \)), Index (\( i \)), Offset (\( k \))
**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”)

**General Cache Organization (S, E, K)**
- \( E = \text{blocks/lines per set} \)
- \( S = \# \text{sets} = 2^s \)
- \( K = \text{bytes per block} \)
- Cache size: \( C = K \times E \times S \) data bytes (doesn’t include V or Tag)
- Valid bit

**Peer Instruction 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 [PollEv.com/justinh]
  - A. 2
  - B. 4
  - C. 8
  - D. 16
  - E. We’re lost...

**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>Variable</th>
<th>This Quarter</th>
<th>Formulas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block size</td>
<td>( E ) (in book)</td>
<td></td>
</tr>
<tr>
<td>Cache size</td>
<td>( C )</td>
<td></td>
</tr>
<tr>
<td>Associativity</td>
<td>( E )</td>
<td></td>
</tr>
<tr>
<td>Number of Sets</td>
<td>( S )</td>
<td></td>
</tr>
<tr>
<td>Address space</td>
<td>( M )</td>
<td></td>
</tr>
<tr>
<td>Address width</td>
<td>( m )</td>
<td></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>( b ) (in book)</td>
<td></td>
</tr>
</tbody>
</table>

**Cache Read**
- \( E = \text{blocks/lines per set} \)
- \( S = \# \text{sets} = 2^s \)
- \( K = \text{bytes per block} \)

**Example: Direct-Mapped Cache (\( E = 1 \))**
- Direct-mapped: One line per set
- Block Size \( K = 8 \) B

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

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

Address of $\text{int}$:
valid? + match?: yes = hit
block offset

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

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

Address of $\text{int}$:
valid? + match?: yes = hit
block offset

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 $\text{short int}$:
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 $\text{short int}$:
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 $\text{short int}$:
valid? + match?: yes = hit
block offset

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
What about writes?

- Multiple copies of data exist:
  - L1, L2, possibly L3, main memory
- What to do on a write-hit?
  - Write-through: write immediately to next level
  - Write-back: defer write to next level until line is evicted (replaced)
    - Must track which cache lines have been modified ("dirty bit")
- What to do on a write-miss?
  - Write-allocate: ("fetch on write") load into cache, update line in cache
    - Good if more writes or reads to the location follow
  - No-write-allocate: ("write around") just write immediately to memory
- Typical caches:
  - Write-back + Write-allocate, usually
  - Write-through + No-write-allocate, occasionally

Write-back, write-allocate example

```plaintext
mov 0xFACE, F
```

Contents of memory stored at address G

Memory

```
0xCAFE
0xBEEF
```

Tag (there is only one set in this tiny cache, so the tag is the entire block address!)

In this example we are sort of ignoring block offsets. Here a block holds 2 bytes (16 bits, 4 hex digits).

Normally a block would be much bigger and thus there would be multiple items per block. While only one item in that block would be written at a time, the entire line would be brought into cache.

Step 1: Bring F into cache

```
0xCAFE
0xBEEF
```

Step 2: Write 0xFEED to cache only and set dirty bit

```
0xCAFE
0xFEED
```

Write hit!

Write 0xFEED to cache only
---

**Write-back, write-allocate example**

```
mov 0xAFACE, F
mov 0x0FEED, F
mov G, %rax
```

---

**Example Cache Parameters Problem**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Size</td>
<td>4 KiB</td>
</tr>
<tr>
<td>Block Size</td>
<td>16 B</td>
</tr>
<tr>
<td>Associativity</td>
<td>4-way</td>
</tr>
<tr>
<td>Hit Time</td>
<td>3 cycles</td>
</tr>
<tr>
<td>Miss Rate</td>
<td>20%</td>
</tr>
<tr>
<td>Write Policy</td>
<td>Write-through</td>
</tr>
<tr>
<td>Replacement Policy</td>
<td>LRU</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>

---

**Peer Instruction Question**

Which of the following cache statements is FALSE?

  - A. We can reduce compulsory misses by decreasing our block size
  - B. We can reduce conflict misses by increasing associativity
  - C. A write-back cache will save time for code with good temporal locality on writes
  - D. A write-through cache will always match data with the memory hierarchy level below it
  - E. We’re lost...

---

**Example Code Analysis Problem**

Assuming the cache starts cold (all blocks invalid), calculate the **miss rate** for the following loop:

```
#define AR_SIZE 2048

int int_ar[AR_SIZE], sum=0;  // &int_ar=0x80000

for (int i=0; i<AR_SIZE; i++)
    sum += int_ar[i];

for (int j=AR_SIZE-1; j>=0; j--)
    sum += int_ar[j];
```