Lecture overview

• **Topics**
  - Basic cache structures
    - Direct mapped cache
    - Set associative cache
    - Fully associative cache
  
  • Suggested reading
    - H&P Chapter 5.3 5.4

Recall: Overview of memory system hierarchy

• registers ↔ memory
  - by compiler/programmer

• cache ↔ memory
  - by hardware

• memory ↔ disks
  - by hardware and operating system (virtual memory)
Cache

- A hardware component in the processor system
  - a small, fast memory
- To achieve a high cache hit, data blocks need to be dynamically transferred between cache and main memory
  - Based on the principle of locality
- A data block may contain multiple bytes/words
  - Further discussion will follow

Cache (cont.)

- For control of the cache operation, four issues should be addressed
  - Where to put a memory block in cache?
    - Block placement strategy
  - How to find a memory block in cache?
    - Block identification
  - If there are no free spaces in cache, which block can be replaced by a new memory block?
    - Block replacement
  - When memory data is updated, how is cache involved in write?
    - Write strategy
- They are closely related to the cache structure

Cache (cont.)

- There are three typical cache structures
  - Direct mapped cache
  - Set associative cache
  - Fully associative cache
Direct mapped cache

- A memory block can map to one and only one cache location
  - For a cache of $2^m$ blocks, a memory block with (block) address $X$ maps to the cache location $X \mod 2^m$
- See an example in the next slide

Direct mapped cache - example

Cache basic fields

- A cache consists of multiple entries
- A cache entry has at least the following fields
  - Valid
    - Indicating whether the cache entry holds valid data
  - Data
    - Storing memory data block
      - A data block (cache line) is the minimum amount of memory data that can be transferred between cache and main memory. The data block is also called cache block when stored in the cache.
  - Tag
    - Identifying the memory data block cached
Example

• Assume the following memory locations are sequentially accessed. How is the cache below updated?
  • 10110, 11010, 10000, 00010

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example (cont.)

• Assume the following memory locations are sequentially accessed. How is the cache below updated?
  • 10110, 11010, 10000, 00010
    • After handling a miss on reference to address 10110

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Y</td>
<td>10</td>
<td>mem[10110]</td>
</tr>
<tr>
<td>111</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example (cont.)

• Assume the following memory locations are sequentially accessed. How is the cache below updated?
  • 10110, 11010, 10000, 00010
    • After handling a miss on reference to address 11010

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>Y</td>
<td>11</td>
<td>mem[11010]</td>
</tr>
<tr>
<td>011</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Y</td>
<td>10</td>
<td>mem[10110]</td>
</tr>
<tr>
<td>111</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example (cont.)

• Assume the following memory locations are sequentially accessed. How is the cache below updated?
  • 10110, 11010, 10000, 00010
    • After handling a miss on reference to address 10000

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Y</td>
<td>10</td>
<td>mem[10000]</td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>Y</td>
<td>11</td>
<td>mem[11010]</td>
</tr>
<tr>
<td>011</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Y</td>
<td>10</td>
<td>mem[10110]</td>
</tr>
<tr>
<td>111</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example (cont.)

• Assume the following memory locations are sequentially accessed. How is the cache below updated?
  • 10110, 11010, 10000, 00010
    • After handling a miss on reference to address 00010

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Y</td>
<td>10</td>
<td>mem[10000]</td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>Y</td>
<td>00</td>
<td>mem[00010]</td>
</tr>
<tr>
<td>011</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Y</td>
<td>10</td>
<td>mem[10110]</td>
</tr>
<tr>
<td>111</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cache block

• A cache block, aka a cache line, is the smallest section of data that can be transferred between cache and memory
  • Identified by a single tag
  • Often contains multiple bytes/words
  • The size is often a power-of-two addressable units

• An example is given in the next slide
  • 32-byte block
1 KB direct mapped cache with 32-byte blocks

- Give a 32-bit byte memory address
  - The uppermost 22 bits of the address make the cache tag
  - The lowest 5 bits are the byte select

### Block size

- Larger block size takes advantage of spatial locality
- But larger block size means larger miss penalties
  - It takes longer time to copy a block
  - If the block size is too big, there are too few blocks in cache and the miss rate will go up
    - One extreme case is discussed in the next slide
- Trade-off should be played based on the average access time

### One extreme case: 1-block cache

- Only one entry in the cache
  - cache size = 1 cache block
- Potential problem?
  - Ping Pong effect/thrashing
    - The locality says if a memory data item is accessed, it will likely be accessed again soon, But it is unlikely that it will be accessed again immediately!!!
    - Causes large conflict misses
      - Will be discussed later
    - An example is given in the next slide
Ping-Pong effect - example

• 1-block cache for instruction memory

<table>
<thead>
<tr>
<th>Valid Bit</th>
<th>Cache Tag</th>
<th>Cache Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>word 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>word 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>word 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>word 0</td>
</tr>
</tbody>
</table>

• Instruction execution flow
  • instruction size: 1 word

• Cache data field contents

<table>
<thead>
<tr>
<th>Cache Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>word 3</td>
</tr>
<tr>
<td>word 2</td>
</tr>
<tr>
<td>word 1</td>
</tr>
<tr>
<td>word 0</td>
</tr>
</tbody>
</table>

How to reduce the Ping-Pong effect?

• Make cache size bigger

How to reduce the Ping-Pong effect?

• Use multiple entries for a memory block
  • Fully associative cache
  • Set associative cache
Fully associative cache

• A memory block can be mapped to any location in the cache.
• The full cache needs to be searched for a memory block
  • Assume cache size is $M$ blocks. We need $M$ comparators for a fully associative cache -- costly

<table>
<thead>
<tr>
<th>Cache Tag (27 bits long)</th>
<th>Byte Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g.: 0x01</td>
<td></td>
</tr>
</tbody>
</table>

![Diagram of Fully Associative Cache]

n-way set associative cache

• Cache is divided into sets
• Each set has $n$ blocks
  • A memory block can be mapped to a set and can be stored in any location in the set
• $n$ comparators are required to search a block in a set
• An example is given in the next slide

2-way set associative cache

<table>
<thead>
<tr>
<th>Cache Tag</th>
<th>Cache Set Index</th>
<th>Byte Select</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Diagram of 2-way Set Associative Cache]
In-class exercise

Give an 8-block cache, how is the memory block 12 (i.e. 12 is the block address) is placed in the cache?
Assume the cache is
1) Fully associative
2) Direct mapped
3) 2-way set associative

Recall: Cache (cont.)

• For control of the cache operation, four issues should be addressed
  • Where to put a memory block in cache?
    • Block placement strategy
  • How to find a memory block in cache?
    • Block identification
  • If there are no free spaces in cache, which block can be replaced by a new memory block?
    • Block replacement
  • When memory data is updated, how is cache involved in write?
    • Write strategy

• They are closely related to the cache structure

Block placement

• How to determine the cache location for a memory block?
• Based on memory block address and cache structures:
  • Direct mapped
    • A memory block can be placed in one and only one location in the cache
  • Fully associative
    • A memory block can be placed in any location in the cache
  • Set associative
    • A memory block can be placed in any location in a set (one and only one) of the cache
Example (1)

• For a memory block with the block address 28, where can it be placed in a **direct mapped** cache of 8 blocks?

Example (2)

• For a memory block with the block address 28, where can it be placed in a **fully associative** cache of 8 blocks?

Example (3)

• For a memory block with the block address 28, where can it be placed in a **2-way set associative** cache of 8 blocks?
Block identification

• How to identify whether a memory block is cached? And where?
  • Based on the memory block address
    • Tag field
    • Set-index field
      • Specifying the possible cache locations
      • Field size
        • 0 bit → fully associative cache
        • n bit → 2^n sets
  • If the entry with the matched tag is found and valid, the block is cached.

<table>
<thead>
<tr>
<th>Block Address</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>Set Index</td>
</tr>
</tbody>
</table>

Block replacement

• When cache is full, which cache block is replaced by the new memory block?
  • For direct mapped
    • Only one option
      • Fixed location
      • The related cache block is always replaced
  • For set associative or fully associative
    • Multiple options
    • Two typical approaches
      • Random
        • Randomly select one of the multiple optional locations for replacement
      • LRU (Least Recently Used)
        • Select the block that is least recently used

Comparison of two replacement policies

Experiment results: cache miss rate of different cache configurations

<table>
<thead>
<tr>
<th>associ.</th>
<th>2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LRU Random</td>
<td>LRU Random</td>
<td>LRU Random</td>
</tr>
<tr>
<td>cache size</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 KB</td>
<td>5.2%  5.7%</td>
<td>4.7%  5.3%</td>
<td>4.4%  5.0%</td>
</tr>
<tr>
<td>64 KB</td>
<td>1.9%  2.0%</td>
<td>1.5%  1.7%</td>
<td>1.4%  1.5%</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15% 1.17%</td>
<td>1.13% 1.13%</td>
<td>1.12% 1.12%</td>
</tr>
</tbody>
</table>

• What conclusion can you draw from the experiment data?
Where are we?

- **Topics**
  - Cache operations and control
    - Block placement
    - Block identification
    - Block replacement
    - Write strategy

- **Suggested reading**
  - H&P Chapter 5.4

Write strategy

- When memory data is updated, how is the cache involved in write? (How is the data saved?)

- **Four typical write policies**
  - When the write has a cache hit (write hit)
    - Write though
    - Write back
  - When the write has a cache miss (write miss)
    - Write allocate
    - Write not allocate

Policies on write hit

- **Write through (WT)**
  - Data is written to both cache and memory.

- **Write back (WB)**
  - Data is written only to cache.
  - The modified cache block is written to memory only when it is replaced.
  - Additional dirty bit is required.
    - A block is dirty if data in the block has been modified; otherwise, the block is clean and no need to be written back when it is replaced.
Policies on write hit (cont.)

• **Pros and Cons of WT and WB**
  - **WT**
    + read misses don’t result in writes
    - potentially high memory traffic
      • performance degradation
  - **WB**
    + less memory accesses
    • no memory writes needed for repeated processor writes
    - cache coherence issue
      • to be discussed in the multiprocessor design

Improve design with write buffer

• To reduce the impact of slow memory access on the performance, a write buffer can be used
• **For a write-to-memory operation,**
  • Processor: writes data to the write buffer
  • Memory controller: writes data in the buffer to memory
• **Write buffer is just a FIFO (First In First Out)**
  • Small, typical number of entries: 4
  • Works fine if processor write frequency << 1 / DRAM write cycle.

Improve design with write buffer (cont.)

• If processor write frequency > 1 / DRAM write cycle,
  • Write buffer will overflow (aka write buffer saturation)
• **Solution for write buffer saturation**
  • Add a second level (L2) cache
  • Faster than DRAM
Types of cache misses

- Cache performance is closely related to cache misses.
- There are three types of cache misses
  - **Compulsory miss (aka cold start miss)**
    - First access to a memory data and the data has never been cached.
  - **Conflict miss (aka collision miss)**
    - Due to competition for an entry in a set
    - In a non-fully associative cache
  - **Capacity miss**
    - Due to limited cache size
    - Related to fully associative cache

Note: the three types of misses are also forms a so-called 3Cs model for cache performance estimation.

Recall: Example

- Assume the following memory locations are sequentially accessed. How is the cache below updated?
  - 10110, 11010, 10000, 00010

After power on

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Recall: Ping-Pong effect - example

- 1-block cache for instruction memory
- Instruction execution flow
  - instruction size: 1 word
- Cache data field contents
Recall: Comparison of two replacement policies

Experiment results: cache miss rate of different cache configurations

<table>
<thead>
<tr>
<th>Cache size</th>
<th>2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LRU Random</td>
<td>LRU Random</td>
<td>LRU Random</td>
</tr>
<tr>
<td>16 KB</td>
<td>5.2%</td>
<td>4.7%</td>
<td>4.4%</td>
</tr>
<tr>
<td></td>
<td>5.7%</td>
<td>5.3%</td>
<td>5.0%</td>
</tr>
<tr>
<td>64 KB</td>
<td>1.9%</td>
<td>1.5%</td>
<td>1.4%</td>
</tr>
<tr>
<td></td>
<td>2.0%</td>
<td>1.7%</td>
<td>1.5%</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15%</td>
<td>1.13%</td>
<td>1.12%</td>
</tr>
<tr>
<td></td>
<td>1.17%</td>
<td>1.13%</td>
<td>1.12%</td>
</tr>
</tbody>
</table>

• What conclusion can you draw from the experiment data?

Trade-offs in the cache design

• Impact of cache size, associativity, and block size on cache performance

<table>
<thead>
<tr>
<th>Design change</th>
<th>Effect on miss rate</th>
<th>Negative performance effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increasing cache size</td>
<td>Decrease capacity misses</td>
<td>May increase hit time</td>
</tr>
<tr>
<td>Increasing associativity</td>
<td>Decrease conflict misses</td>
<td>May increase hit time</td>
</tr>
<tr>
<td>Increasing block size</td>
<td>Decreases compulsory misses</td>
<td>Increase miss penalty May increase conflict misses</td>
</tr>
</tbody>
</table>

• Trade-off should be played in the cache design