Announcements

- **Final Review Discussion Section**
  - Wed. Nov. 30, 6:30 - 7:50, Center 216

- **Final Exam**
  
  **When:** Thursday., December 8, 7 - 9:59 PM  
  **Where:** Center 216
How is a Block found in the Cache?

- 4 Kbyte Cache direct mapped cache with 1 word (4-byte) blocks
Total number of bits in the Cache

- 4 Kbyte Cache direct mapped cache with 1 word (4-byte) blocks
Handling a Cache Read Miss

A mis-match on tag and/or Valid bit indicates a miss.

- Stall CPU
- Make read request to memory (via I/O controller)
- When memory returns the data, write it into the cache
- Return the data to the CPU
Handling a Cache Write Miss

A mis-match on tag and/or Valid bit indicates a miss.

- Write tag, valid bit and data into the cache
  Works only if block size = word size
- Should the data be written to memory also?
Dealing with Stores

- Stores must be handled differently than loads, because...
  - They don’t necessarily require the CPU to stall
    - stores don’t produce register values used by other instructions
  - They change the content of cache/memory (creating memory consistency issues)

- Policy decisions for stores
  - write-through => all writes go to both cache and main memory
  - write-back => writes go only to cache. Modified cache lines are written back to memory when the line is replaced.
  - write-allocate => on a store miss, bring written line into the cache
  - write-around => on a store miss, write to main memory, & ignore cache
Handling a Cache Write Miss

- **Write-through Cache**
  - Write data to cache as well as memory
  - Don’t need to consider whether the write hits or misses the cache
  - Disadvantage: Every write causes the data to be written to the main memory
    - Use write buffer so CPU can proceed with the following instructions

- **Write-back Cache**
  - When write occurs, write the new value only to the block in the cache
  - Write cache data to memory when it is about to be overwritten for another address
  - Improves performance over write-through cache
  - More complex to implement
Summary for Stores

- On a store hit, write the new data to cache. In a *write-through* cache, write the data immediately to memory. In a *write-back* cache, mark the line as dirty.
- On a store miss, initiate a cache block load from memory for a write-allocate cache. Write directly to memory for a no write-allocate cache.
- On any kind of cache miss in a write-back cache, if the line to be replaced in the cache is dirty, write it back to memory.
Taking advantage of Spatial Locality

- Consider following address trace:
  - 0,1,2,3,17,8,9,10,11,17,4,5,6,7…

- Notice that addresses lie in the vicinity of each other

- Instructions show high degree of spatial locality
  - Typically accessed sequentially
  - Generally, code consists of a lot of loops

- Data also shows spatial locality
  - e.g. different elements of a structure may be accessed

- Why not bring multiple words on a cache miss?
  - Instead of bringing a single?
Spatial Locality: Larger Cache Blocks

- Large cache blocks take advantage of *spatial locality*.
- Too large of a block size can waste cache space.
- Larger cache blocks require less tag space.
A 64 KB Cache using 16-byte Blocks

Address (bit positions)
31...16 15...4 32 10

Hit
Tag
Index
2 Byte offset

V Tag
16 bits

Data
128 bits

Block offset

4K entries

Mux

16 32 32 32 32
A 64 KB Cache using 16-byte Blocks

Number of blocks = Cache size / Block size = (64 KB / 16 B) = 4K
Block index = \( \log_2(\text{Number of blocks}) = \log_2(4K) = 12 \)
Complication with Larger Blocks

- Write-through cache: Can’t write to the cache while performing a tag comparison
  - Ok if there is a hit in the cache
  - Not ok if there is a cache miss:
    - The block has to be fetched from memory and placed in the cache
    - Rewrite the word that caused the miss into the cache
Impact of Block Size on Miss Rate

- In general, larger block decreases miss rate, however,
  - Larger block size means larger miss penalty:
    - Takes longer time to fill up the block
  - Miss rates go up if block size is too big
    - Since there are too few cache blocks
Flexible Placement of Blocks

- **Direct mapped cache**
  - A block can go in exactly one place in the cache
  - Leads to collision among blocks

- **Fully associative cache**
  - A block can go in any place in the cache
  - All addresses have to be compared simultaneously
  - Slow and expensive

- **N-way set-associative cache**
  - Consists of a number of sets
  - Each set consists of N blocks
  - Each block in memory maps to a unique set
    - A block can be placed in any element of the set

**Set containing a memory block**

= \((\text{block number}) \mod (\text{number of sets in the cache})\)

**Number of sets in the cache**

= \(\frac{\text{Cache size}}{((\text{block size}) \times (\text{associativity}))}\)
Locating a Block in a Cache

Block address = $12_{10}$

**Direct Mapped**
- Direct mapped
- Block # 0 1 2 3 4 5 6 7
- Tag
- Search

**Set-associative**
- Set associative
- Set # 0 1 2 3
- Tag
- Search

**Fully associative**
- Fully associative
- Tag
- Search
Cache Configurations

- An eight-block cache with various configurations

<table>
<thead>
<tr>
<th>One-way Associative (direct mapped)</th>
<th>Tag Data</th>
<th>Tag Data</th>
<th>Tag Data</th>
<th>Tag Data</th>
<th>Tag Data</th>
<th>Tag Data</th>
<th>Tag Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Two-way set associative</th>
<th>Tag Data</th>
<th>Tag Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Four-way set associative</th>
<th>Tag Data</th>
<th>Tag Data</th>
<th>Tag Data</th>
<th>Tag Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Eight-way set associative (fully associative)</th>
<th>Tag Data</th>
<th>Tag Data</th>
<th>Tag Data</th>
<th>Tag Data</th>
<th>Tag Data</th>
<th>Tag Data</th>
<th>Tag Data</th>
<th>Tag Data</th>
<th>Tag Data</th>
<th>Tag Data</th>
<th>Tag Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Accessing a 4-way Set-associative cache

- 4 K-byte 4-way set-associative cache, with a block size of 4 bytes
Accessing a Direct Mapped Cache

- 64 KB cache, direct-mapped, 32-byte cache block size
Accessing a Set-associative Cache

- 32 KB cache, 2-way set-associative, 16-byte block size
Cache Associativity and Miss Rate

Data cache miss rate for SPEC2000 benchmark

- Benefit of going from direct mapped to two-way set associative cache is significant.
- Benefits of further increase in associativity are smaller.
Associative Caches: Higher hit rates, but...

- Longer access time (longer to determine hit/miss, more muxing of outputs)
- More space (longer tags)
  - 16 KB, 16-byte blocks, direct mapped, tag = ?
  - 16 KB, 16-byte blocks, 4-way, tag = ?
A Fully-associative cache

<table>
<thead>
<tr>
<th>Word address string:</th>
<th>00010100</th>
<th>00000101</th>
<th>00010100</th>
<th>00000100</th>
<th>00001001</th>
<th>00001111</th>
<th>00001000</th>
<th>00001010</th>
<th>00001100</th>
<th>00001001</th>
<th>00001111</th>
<th>00001000</th>
<th>00001001</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The tag identifies the address of the cached data

Valid bit indicates that entry is valid

<table>
<thead>
<tr>
<th>tag</th>
<th>v</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4 entries, each block holds one word, any block can hold any word.

- A cache that can put a block of data anywhere is called **fully associative**
- To access the cache, address must be compared with all the entries in the cache
Cache Organization

- A typical cache has three dimensions

```
tag   data
>tag   data
>tag   data
>tag   data
>tag   data
>tag   data
>tag   data
>tag   data
>tag   data
>tag   data
>tag   data
>tag   data
<tag   data
```
The Three Cs

- **Compulsory misses:** First time access
  - Caused by the first access to a block that has never been in the cache
  - Also called cold-start misses
  - Can be reduced by increasing the block size

- **Capacity misses:** Cache is not large enough
  - Caused when cache cannot contain all the blocks needed
  - Occur because of blocks being replaced and later retrieved
  - Can be reduced by enlarging the cache

- **Conflict misses:** Multiple addresses map to the same cache line
  - Occur in direct mapped and set-associative caches
  - Multiple blocks compete for the same set
  - Can be eliminated by using fully associative cache
Which Block Should be Replaced on a Miss?

- Direct Mapped is Easy
- Set associative or fully associative:
  - Random (large associativities)
  - LRU (smaller associativities)
- Miss rates for the two schemes:

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>2-Way Set Associative</th>
<th>4-Way Set Associative</th>
<th>8-Way Set Associative</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LRU</td>
<td>Random</td>
<td>LRU</td>
</tr>
<tr>
<td>16 KB</td>
<td>5.18</td>
<td>5.69</td>
<td>4.67</td>
</tr>
<tr>
<td>64 KB</td>
<td>1.88</td>
<td>2.01</td>
<td>1.54</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15</td>
<td>1.17</td>
<td>1.13</td>
</tr>
</tbody>
</table>

LRU is preferred scheme for a small size cache
Cache Performance

- 64 KB each instruction cache and data cache (direct mapped)

<table>
<thead>
<tr>
<th>Program</th>
<th>Block Size in words</th>
<th>Instruction miss rate</th>
<th>Data miss rate</th>
<th>Effective Combined miss rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>1</td>
<td>6.1%</td>
<td>2.1%</td>
<td>5.4%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2.0%</td>
<td>1.7%</td>
<td>1.9%</td>
</tr>
<tr>
<td>spice</td>
<td>1</td>
<td>1.2%</td>
<td>1.3%</td>
<td>1.2%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.3%</td>
<td>0.6%</td>
<td>0.4%</td>
</tr>
</tbody>
</table>

- In general, Average Access Time:
  \[ \text{Average Access Time} = \text{Hit Time} \times (1 - \text{Miss Rate}) + \text{Miss Penalty} \times \text{Miss Rate} \]
Measuring Cache Performance

CPU time = (CPU execution clock cycles + Memory stall cycles) * Clock cycle time

Memory stall cycles = Read-stall cycles + Write-stall cycles

Read-stall cycles = (Reads per program) * Read miss rate * Read miss penalty

Write-stall cycles = (Writes per program) * Write miss rate * Write miss penalty
Cache Performance

- Ideal CPI for a processor (i.e. without memory stalls) is 1.4.
- 35% of the instructions are loads and stores.
- Miss penalty is 10 cycles
- How much do we improve the performance if the miss rate is reduced from 10% to 2%?
Summary

• The Principle of Locality:
  – Program likely to access a relatively small portion of the address space at any instant of time.
    • Temporal Locality: Locality in Time
    • Spatial Locality: Locality in Space

• Three Major Categories of Cache Misses:
  – Compulsory Misses: sad facts of life. Example: cold start misses.
  – Conflict Misses: increase cache size and/or associativity.
    Nightmare Scenario: ping pong effect!
  – Capacity Misses: increase cache size

• Cache Design Space
  – total size, block size, associativity
  – replacement policy
  – write-hit policy (write-through, write-back)

• Caches give an illusion of a large, cheap memory with the access time of a fast, expensive memory