 Hits vs. Misses

- Read hits:
  - this is what we want!
- Read misses:
  - stall the CPU, fetch block from memory, deliver to cache, restart
    - Put data read from memory into the cache Data field
    - Write upper bits of address into the cache Tag field
    - Turn the Valid Bit ON
- How to partition memory address:
  - # of bits for offset: specify the correct byte within a block
  - # of bits for index: specify the correct cache block
  - # of bits for tag: remaining bits
  - lw $s1, 0000 1001 1100 0110 1010 1111 0101 1000

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>offset</th>
</tr>
</thead>
</table>

 Block Write Strategies

1) Write-Through:
   - Write the data to (a) cache and (b) block in main memory
   **Advantage:** Easier implementation, only need write buffer
   **Disadvantage:** lower speed

2) Write-Back:
   - Write the data only to cache block. Write to memory only when block is replaced
   **Advantage:** Writes are limited only by cache write rate, higher speed, Multi-word writes supported, since only one (efficient) write to main memory is made per block
   **Disadvantage:** more difficult to implement
Write-Through

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update memory
- But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = 1 + 0.1\times100 = 11
- Solution: write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

Write-Back

- Alternative: On data-write hit, just update the block in cache
  - Keep track of whether each block is dirty
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first
Write Allocation

- What should happen on a write miss?

- Alternatives for write-through
  - Allocate on miss: fetch the block
  - Non-Allocate: don’t fetch the block
    - Since programs often write a whole block before reading it (e.g., initialization)

- For write-back
  - Usually fetch the block

Memory System support caches

- Assume
  - 1 clock cycle to send the address
  - 15 cc for each DRAM access initiated
  - 1 cc to send a word of data

- Make reading multiple words easier by using banks of memory

1+15+4x1=20

1+4x15+4x1=65

1+15+1=17
5.3 Measuring and Improving Cache Performance

• Simplified model:

\[
\text{execution time} = (\text{execution cycles} + \text{memory stall cycles}) \times \text{cycle time}
\]

\[
\text{stall cycles} = \# \text{ of instructions} \times \text{miss ratio} \times \text{miss penalty}
\]

• \( \text{AMAT} = \text{hit time} + \text{miss rate} \times \text{miss penalty} \)

• Two ways of improving performance:
  – decreasing the miss rate
    • Larger block size, larger cache, higher associativity
    • use split caches
  – decreasing the miss penalty
    Multi-level cache

Cost of a Cache Miss

**Example: Instruction Cache Miss** (couldn’t find instr. in cache)

• **Observe**: Bigger blocks take more time to fetch

• **Oops**: Bigger blocks exploit *spatial locality* property

• Which one? **Solution**: Make cache as big as possible
  This decreases effect of blocksize
Cache Design Trade-offs

<table>
<thead>
<tr>
<th>Design change</th>
<th>Effect on miss rate</th>
<th>Negative performance effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increase cache size</td>
<td>Decrease capacity misses</td>
<td>May increase access time</td>
</tr>
<tr>
<td>Increase associativity</td>
<td>Decrease conflict misses</td>
<td>May increase access time</td>
</tr>
<tr>
<td>Increase block size</td>
<td>Decrease compulsory misses</td>
<td>Increases miss penalty. For very large block size, may increase miss rate due to pollution.</td>
</tr>
</tbody>
</table>

Cache Performance Example

- **Given**
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- **Miss cycles per instruction**
  - I-cache: $0.02 \times 100 = 2$
  - D-cache: $0.36 \times 0.04 \times 100 = 1.44$
- **Actual CPI** = $2 + 2 + 1.44 = 5.44$
  - Ideal CPU is $5.44/2 = 2.72$ times faster
Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2ns
    - 2 cycles per instruction

Decreasing miss penalty with multilevel caches

- Add a second level cache:
  - often primary cache is on the same chip as the processor
  - use SRAMs to add another cache above primary memory (DRAM)
  - miss penalty goes down if data is in 2nd level cache
- Example:
  - CPI of 1.0 on a 500Mhz machine with a 5% miss rate, 200ns DRAM access
  - Adding 2nd level cache with 20ns access time decreases miss rate to 2%
- Using multilevel caches:
  - try and optimize the hit time on the 1st level cache
  - try and optimize the miss rate on the 2nd level cache
4 Questions for Memory Hierarchy

• Q1: Where can a block be placed?

<table>
<thead>
<tr>
<th>Scheme</th>
<th># of sets</th>
<th>Blocks per set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct mapped</td>
<td># of blocks</td>
<td>1</td>
</tr>
<tr>
<td>Set Associative</td>
<td># of blocks/Associativity</td>
<td>2-8</td>
</tr>
<tr>
<td>Fully Associative</td>
<td>1</td>
<td># of blocks</td>
</tr>
</tbody>
</table>

• Q2: How is a block found?

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Location method</th>
<th>Comparisons required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct mapped</td>
<td>index</td>
<td>1</td>
</tr>
<tr>
<td>Set Associative</td>
<td>Index set, search among elements</td>
<td>Degree of associativity</td>
</tr>
<tr>
<td>Fully Associative</td>
<td>Search all cache entries</td>
<td>Size of the cache</td>
</tr>
<tr>
<td></td>
<td>Separate lookup table</td>
<td>0</td>
</tr>
</tbody>
</table>

• Q3: Which block should be replaced on a cache miss

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Which block to replace</th>
<th>replace scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct mapped</td>
<td>Only one</td>
<td>none</td>
</tr>
<tr>
<td>Set Associative</td>
<td># of blocks in a set</td>
<td>LRU, random</td>
</tr>
<tr>
<td>Fully Associative</td>
<td>All blocks</td>
<td>LRU, random</td>
</tr>
</tbody>
</table>

• Q4: What happens on a write?
  - Write back
    - Allocate
  - Write through
    - Allocate, nonallocate
Sources of Misses (The 3 Cs)

- Compulsory misses (aka cold start misses)
  - First access to a block
- Capacity misses
  - Due to finite cache size
  - A replaced block is later accessed again
- Conflict misses (aka collision misses)
  - In a non-fully associative cache
  - Due to competition for entries in a set
  - Would not occur in a fully associative cache of the same total size

Hits vs. Misses

- Read hits:
  - this is what we want!
- Read misses:
  - stall the CPU, fetch block from memory, deliver to cache, restart
    - Put data read from memory into the cache Data field
    - Write upper bits of address into the cache Tag field
    - Turn the Valid Bit ON
- Write hits:
  - can replace data in cache and memory (write-through)
  - write the data only into the cache (write-back the memory later)
- Write misses:
  - read the entire block into the cache, then write the word
The Memory Hierarchy: Summary

The BIG Picture

- Common principles apply at all levels of the memory hierarchy
  - Based on notions of caching
- At each level in the hierarchy
  - Block placement
  - Finding a block
  - Replacement on a miss
  - Write policy

Write Policy

- Write-through
  - Update both upper and lower levels
  - Simplifies replacement, but may require write buffer
- Write-back
  - Update upper level only
  - Update lower level when block is replaced
  - Need to keep more state
- Virtual memory
  - Only write-back is feasible, given disk write latency
Review

- **Examples** of direct mapped cache and memory address partition
- Direct-mapped cache with larger block size
- Memory organization supports larger block size
  - With interleaved memory bank
- Cache Performance
  - execution time = (execution cycles + memory stall cycles) × cycle time, where
    stall cycles = # of instructions × miss ratio × miss penalty
  - AMAT = hit time + miss rate × miss penalty
- How to improve performance?
  - Larger block size, larger cache, higher associativity, use split caches, Multi-level cache