Dealing with Stores

• Stores must be handled differently than loads, because...
  – they don’t necessarily require the CPU to stall.
  – they change the content of cache/memory (creating memory consistency issues)

• Q: Can you think of a situation when you might need to load from memory before you can execute a store?
  – Can you think of another one?
Policy decisions for stores

• Keep memory and cache identical?
  - Write-through \[\Rightarrow\text{ all writes go to both cache and main memory}\]
  - Write-back \[\Rightarrow\text{ writes go only to cache. Modified cache lines are written back to memory when the line is replaced.}\]

• Make room in cache for store miss?
  - write-allocate \[\Rightarrow\text{ on a store miss, bring written line into the cache}\]
  - write-around \[\Rightarrow\text{ on a store miss, ignore cache}\]
Dealing with 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 write-around 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.
Example -- DEC Alpha 21164 Caches

- ICache and DCache -- 8 KB, DM, 32-byte lines
  - D$ is also dual-read-ported, single-write ported, write-through, read-allocate...
- L2 cache -- 96 KB, 2-way SA, 32-byte lines
- L3 cache -- 1 MB, DM, 32-byte lines
Cache Performance

- CPI = BCPI + MCPI
  - BCPI = base CPI, which means the CPI assuming perfect memory
  - MCPI = the memory CPI, the number of cycles (per instruction) the processor is stalled waiting for memory.
Cache Performance

CPI = BCPI + MCPI

- BCPI = base CPI, which means the CPI assuming perfect memory
- MCPI = the memory CPI, the number of cycles (per instruction) the processor is stalled waiting for memory.

MCPI = accesses/instruction * miss rate * miss penalty

- this assumes we stall the pipeline on both read and write misses, that the miss penalty is the same for both, that cache hits require no stalls.
- If the miss penalty or miss rate is different for Inst cache and data cache (common case), then

\[
MCPI = I^\$ \text{ accesses/inst} \times I^\$MR \times I^\$MP + D^\$ \text{ acc/inst} \times D^\$MR \times D^\$MP
\]

\[1.0 \times 10^{-1} \times 10^{-6} \times 9 \times 5 \times 10 \times 1.0 + 2 + 2.5\]
In fact...

- Can generalize this formula further for other stalls:

\[
\text{CPI} = \text{BCPI} + \text{DH_SPI} + \text{BH_SPI} + \text{MCPI}
\]

- \text{DH_SPI} = \text{data hazard stalls per instruction}
- \text{BH_SPI} = \text{branch hazard stalls per instruction.}
Cache Performance

Instruction cache miss rate of 4%
Data cache miss rate of 10%
BCPI = 1.0 (no data or control hazards)
20% of instructions are loads and stores
Miss penalty = 12 cycles

\[
\text{CPI} = 1.8 + \frac{0.04 \times 12}{2} + \frac{0.2 \times 12}{1.0} + 0.48 + 0.24 = 1.72
\]

\[
\begin{array}{|c|c|}
\hline
\text{Selection} & \text{CPI (rounded if necessary)} \\
\hline
A & 1.24 \\
B & 1.34 \\
C & 1.48 \\
D & 1.72 \\
E & \text{None of the above} \\
\hline
\end{array}
\]
Cache Performance

• Unified cache
• 25% of instructions are loads and stores
• BCPI = 1.2, miss penalty of 10 cycles

• If we improve the miss rate from 10% to 4% (e.g. with a larger cache), how much do we improve performance?
Cache Performance

• BCPI = 1
• Miss rate of 8% overall, 20% loads, miss penalty 20 cycles, never stalls on stores.

• What is the speedup from doubling the CPU clock rate?
Three types of cache misses

- Compulsory (or cold-start) misses
  - first access to the data.
- Capacity misses
  - we missed only because the cache isn’t big enough.
- Conflict misses
  - we missed because the data maps to the same line as other data that forced it out of the cache.

DM cache

<table>
<thead>
<tr>
<th>tag</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

address_string:

<table>
<thead>
<tr>
<th>address</th>
<th>Comp</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000100</td>
<td>M Comp</td>
</tr>
<tr>
<td>00001000</td>
<td>M Comp</td>
</tr>
<tr>
<td>00001100</td>
<td>M Comp</td>
</tr>
<tr>
<td>00010100</td>
<td>M Conflict</td>
</tr>
<tr>
<td>00000100</td>
<td>M Comp</td>
</tr>
<tr>
<td>00001000</td>
<td>M Comp</td>
</tr>
<tr>
<td>00011000</td>
<td>M Comp</td>
</tr>
<tr>
<td>00001100</td>
<td>M Conflict</td>
</tr>
<tr>
<td>00000100</td>
<td>M Comp</td>
</tr>
<tr>
<td>00001000</td>
<td>M Comp</td>
</tr>
</tbody>
</table>

- Would also have a FA, i.e., a fault.
Q: Categorizing Misses

• Suppose you experience a cache miss on a block (let's call it block A).
• You have accessed block A in the past. There have been precisely $10^7$ different blocks accessed between your last access to block A and your current miss.
• Your block size is 32-bytes and you have a 64KB cache. What kind of miss was this?

\[
\frac{2^{16}}{2^5} = 2^8 \\text{ \& \, 2K \ entries}
\]

<table>
<thead>
<tr>
<th>Selection</th>
<th>Cache Miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Compulsory</td>
</tr>
<tr>
<td>B</td>
<td>Capacity</td>
</tr>
<tr>
<td>C</td>
<td>Conflict</td>
</tr>
<tr>
<td>D</td>
<td>Both Capacity and Conflict</td>
</tr>
<tr>
<td>E</td>
<td>None of the above</td>
</tr>
</tbody>
</table>
So, then, how do we decrease...

- Compulsory misses?
- Capacity misses?
- Conflict misses?
Cache Associativity

![Graph showing cache associativity with varying sizes and associativity levels.](image-url)
LRU replacement algorithms

- only needed for associative caches
- requires one bit for 2-way set-associative, 8 bits (per set, 2/line) for 4-way, 24 bits for 8-way...
- can be emulated with log n bits (NMRU)
- can be emulated with use bits for highly associative caches (like page tables)
- However, for most caches (eg, associativity <= 8), LRU is calculated exactly.
Caches in Current Processors

• Not long ago, they were DM at lowest level (closest to CPU), associative further away. Today they are less associative near the processor (2-4+), and more associative farther away (4-16).

• split I and D close (L1) to the processor (for throughput rather than miss rate), unified further away (L2 and beyond).

• write-through and write-back both common, but never write-through all the way to memory.

• 64-byte cache lines common (but getting larger)

• Non-blocking
  – processor doesn’t stall on a miss, but only on the use of a miss (if even then)
  – this means the cache must be able to keep track of multiple outstanding accesses, even multiple outstanding misses.
# Intel Nehalem (i7)

- **Instruction Cache**: 32 KB, 4-way, 64-byte line
- **Data Cache**: 32 KB, 8-way, 64-byte line, write-back, write-allocate
- **Unified L2 Cache**: 256 KB, 8-way, 64-byte line, write-back, write-allocate
- **Shared, unified L3 Cache**: 8 MB, 16-way, 64-byte line, write-back, write-allocate
Key Points

- Caches give illusion of a large, cheap memory with the access time of a fast, expensive memory.
- Caches take advantage of memory locality, specifically temporal locality and spatial locality.
- Cache design presents many options (block size, cache size, associativity, write policy) that an architect must combine to minimize miss rate and access time to maximize performance.
ADVANCED CACHE ARCHITECTURES