Outline of Today's Lecture

- The Memory Hierarchy
- Direct Mapped Cache.
- Two-Way Set Associative Cache
- Fully Associative cache
- Replacement Policies
- Write Strategies
The Big Picture: Where are We Now?

- The Five Classic Components of a Computer

- Today's Topic: Memory System

The Motivation for Caches

- Motivation:
  - Large memories (DRAM) are slow
  - Small memories (SRAM) are fast
  - Make the average access time small by:
    - Servicing most accesses from a small, fast memory.
  - Reduce the bandwidth required of the large memory
Levels of the Memory Hierarchy

- **CPU Registers**
  - Capacity: 100s Bytes
  - Access Time: <1s
  - Cost: ~$0.0005/bit

- **Cache**
  - Capacity: K Bytes
  - Access Time: 1-30 ns
  - Cost: ~$0.0005/bit

- **Main Memory**
  - Capacity: M Bytes
  - Access Time: 100ns-1us
  - Cost: ~$0.000001/bit

- **Disk**
  - Capacity: 1-30 G Bytes
  - Access Time: 3-15 ms
  - Cost: 10-5 cents

- **Tape**
  - Capacity: Infinite
  - Access Time: sec-min
  - Cost: ~$0.000001/bit

---

The Principle of Locality

- **The Principle of Locality:**
  - Program access a relatively small portion of the address space at any instant of time.
  - Example: 90% of time in 10% of the code

- **Two Different Types of Locality:**
  - **Temporal Locality** (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
  - **Spatial Locality** (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.
Memory Hierarchy: Principles of Operation

- At any given time, data is copied between only 2 adjacent levels:
  - Upper Level (Cache): the one closer to the processor
    - Smaller, faster, and uses more expensive technology
  - Lower Level (Memory): the one further away from the processor
    - Bigger, slower, and uses less expensive technology

- Block:
  - The minimum unit of information that can either be present or not present in the two level hierarchy

Memory Hierarchy: Terminology

- **Hit**: data appears in some block in the upper level (example: Block X)
  - **Hit Rate**: the fraction of memory access found in the upper level
  - **Hit Time**: Time to access the upper level which consists of RAM access time + Time to determine hit/miss

- **Miss**: data needs to be retrieve from a block in the lower level (Block Y)
  - **Miss Rate** = 1 - (Hit Rate)
  - **Miss Penalty** = Time to replace a block in the upper level + Time to deliver the block the processor

- Hit Time $\ll$ Miss Penalty
Four Questions for Memory Hierarchy Designers

- **Q1:** Where can a block be placed in the upper level? *(Block placement)*
- **Q2:** How is a block found if it is in the upper level? *(Block identification)*
- **Q3:** Which block should be replaced on a miss? *(Block replacement)*
- **Q4:** What happens on a write? *(Write strategy)*

Direct Mapped Cache

- Direct Mapped cache is an array of fixed size blocks.
- Each block holds consecutive bytes of main memory data.
- The Tag Array holds the Block Memory Address.
- A valid bit associated with each cache block tells if the data is valid.

- **Cache Index:** The location of a block (and it’s tag) in the cache.
- **Block Offset:** The byte location in the cache block.

Cache-Index = \(<\text{Address}> \mod (\text{Cache\_Size})\)/ \(\text{Block\_Size}\)

Block-Offset = \(<\text{Address}> \mod (\text{Block\_Size})\)

Tag = \(<\text{Address}> / (\text{Cache\_Size})\)
The Simplest Cache: Direct Mapped Cache

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

4 Byte Direct Mapped Cache

- Location 0 can be occupied by data from:
  - Memory location 0, 4, 8, ... etc.
  - In general: any memory location whose 2 LSBs of the address are 0s
  - Address<1:0> => cache index

Which one should we place in the cache?

How can we tell which one is in the cache?

Direct Mapped Cache (Cont.)

For a Cache of $2^M$ bytes with block size of $2^L$ bytes

- There are $2^{M-L}$ cache blocks,
- Lowest L bits of the address are Block-Offset bits
- Next (M - L) bits are the Cache-Index.
- The last (32 - M) bits are the Tag bits.

\[
\begin{array}{cccc}
& 32-M & M-L & L \\
\hline
\text{Tag} & \text{Cache Index} & \text{block offset} \\
\end{array}
\]

Data Address
Example: 1-KB Cache with 32B blocks:

Cache Index = (<Address> Mod (1024))/ 32

Block-Offset = <Address> Mod (32)

Tag = <Address> / (1024)

Direct Mapped Cache Data

Example: 1KB Direct Mapped Cache with 32B Blocks

- For a 1024 (2^10) byte cache with 32-byte blocks:
  - The uppermost 22 = (32 - 10) address bits are the Cache Tag
  - The lowest 5 address bits are the Byte Select (Block Size = 2^5)
  - The next 5 address bits (bit5 - bit9) are the Cache Index
Example: 1K Direct Mapped Cache

Cache Tag: 0x0002fe

Valid Bit: 0

Cache Data:
- Byte 31: **
- Byte 0: 0
- Byte 63: **
- Byte 33: Byte 32

Cache Index: 0x00

Byte Select: 0x00

Cache Miss

Example: 1K Direct Mapped Cache

Cache Tag: 0x0002fe

Valid Bit: 1

Cache Data:
- Byte 31: **
- Byte 63: **
- Byte 33: Byte 32

Cache Index: 0x00

Byte Select: 0x00

New Block of data
Example: 1K Direct Mapped Cache

- **Cache Tag**: 0x000050
- **Cache Index**: 0x01
- **Byte Select**: 0x08

- **Valid Bit**: 1
- **Cache Tag**: 0x0002fe
- **Cache Data**: Byte 31: ***, Byte 1: Byte 0
- **Byte Select**: Byte 63: ***, Byte 33: Byte 32

Cache Hit

Example: 1K Direct Mapped Cache

- **Cache Tag**: 0x000050
- **Cache Index**: 0x02
- **Byte Select**: 0x04

- **Valid Bit**: 1
- **Cache Tag**: 0x002450
- **Cache Data**: Byte 31: ***, Byte 1: Byte 0
- **Byte Select**: Byte 63: ***, Byte 33: Byte 32

Cache Miss
Example: 1K Direct Mapped Cache

<table>
<thead>
<tr>
<th>Cache Tag</th>
<th>Cache Index</th>
<th>Byte Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x002450</td>
<td>0x02</td>
<td>0x04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Valid Bit</th>
<th>Cache Tag</th>
<th>Cache Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x0002fe</td>
<td>** Byte 31</td>
</tr>
<tr>
<td>1</td>
<td>0x000050</td>
<td>** Byte 63</td>
</tr>
<tr>
<td>1</td>
<td>0x002450</td>
<td>** Byte 33</td>
</tr>
<tr>
<td></td>
<td></td>
<td>** Byte 0</td>
</tr>
</tbody>
</table>

Block Size Tradeoff

- In general, larger block size takes advantage of spatial locality **BUT**:  
  - Larger block size means larger miss penalty:  
    - Takes longer time to fill up the block  
    - If block size is too big relative to cache size, miss rate will go up  
    - Too few cache blocks  

- **Average Access Time**:
  - Hit Time x (1 - Miss Rate) + Miss Penalty x Miss Rate
A N-way Set Associative Cache

- **N-way set associative**: N entries for each Cache Index
  - N direct mapped caches operating in parallel

- **Example: Two-way set associative cache**
  - Cache Index selects a “set” from the cache
  - The two tags in the set are compared in parallel
  - Data is selected based on the tag result

Advantages of Set associative cache

- **Higher Hit rate** for the same cache size.

- **Fewer Conflict Misses.**

- **Can have a larger cache but keep the index smaller** *(same size as virtual page index)*
Disadvantage of Set Associative Cache

- N-way Set Associative Cache versus Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss decision and set selection

- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue, recover later if it is a miss.

And yet Another Extreme Example:
Fully Associative cache

- Fully Associative Cache -- push the set associative idea to its limit!
  - Forget about the Cache Index
  - Compare the Cache Tags of all cache entries in parallel
  - Example: Block Size = 32B blocks, we need $N$ 27-bit comparators

- By definition: Conflict Miss = 0 for a fully associative cache
Sources of Cache Misses

- **Compulsory** (cold start or process migration, first reference): first access to a block
  - Miss in infinite cache
  - “Cold” fact of life: not a whole lot you can do about it
  - Prefetch, larger block gives implicit prefetch

- **Capacity:**
  - Miss in Fully Associative cache with LRU replacement
  - Cache cannot contain all blocks access by the program
  - Solution: increase cache size

- **Conflict** (collision):
  - Hit in Fully Associative, miss in Set-Associative
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity

- **Invalidation**: other process (e.g., I/O) updates memory

<table>
<thead>
<tr>
<th></th>
<th>Direct Mapped</th>
<th>N-way Set Associative</th>
<th>Fully Associative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Size</td>
<td>Big</td>
<td>Medium</td>
<td>Small</td>
</tr>
<tr>
<td>Compulsory Miss</td>
<td>Same</td>
<td>Same</td>
<td>Same</td>
</tr>
<tr>
<td>Conflict Miss</td>
<td>High</td>
<td>Medium</td>
<td>Zero</td>
</tr>
<tr>
<td>Capacity Miss</td>
<td>Low(er)</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Invalidation Miss</td>
<td>Same</td>
<td>Same/lower</td>
<td>Same/lower</td>
</tr>
</tbody>
</table>

Note:
If you are going to run “billions” of instruction, Compulsory Misses are insignificant.
The Need to Make a Decision!

- **Direct Mapped Cache:**
  - Each memory location can map to only 1 cache location (frame)
  - No need to make any decision :-)  
    - Current item replaced the previous item in that cache location

- **N-way Set Associative Cache:**
  - Each memory location have a choice of N cache frames

- **Fully Associative Cache:**
  - Each memory location can be placed in ANY cache frame

- Cache miss in a N-way Set Associative or Fully Associative Cache:
  - Bring in new block from memory
  - Throw out a cache block to make room for the new block
  - We need to make a decision on which block to throw out!

---

Cache Block Replacement Policy

- **Random Replacement:**
  - Hardware randomly selects a cache item and throw it out

- **Least Recently Used:**
  - Hardware keeps track of the access history
  - Replace the entry that has not been used for the longest time.
  - For two way set associative cache only needs one bit for LRU replacement.

- **Example of a Simple “Pseudo” Least Recently Used Implementation:**
  - Assume 64 Fully Associative Entries
  - Hardware replacement pointer points to one cache entry
  - Whenever an access is made to the entry the pointer points to:
    - Move the pointer to the next entry
  - Otherwise: do not move the pointer

---