Why Cache?

- Memory accesses are slow
- ~10 ns to access memory - but processors run at 2.4 GHz = 0.417 ns cycle time
Types of Caching Schemes

- Direct Mapped: 1-to-1
- Fully Associative: 1-to-all
- Set Associative: 1-to-n
## Direct Mapped Cache

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

- Each memory block can only go to one location in cache
- Good for: large caches (fast lookup)
- # index bits? # tag bits?
Direct Mapped Cache

- Each memory block can only go to one location in cache
- Good for: large caches (fast lookup)
- # index bits? # tag bits?
  \[ \log_2(\text{cache\_size}), \text{mem\_bits} - \text{index} - \text{offset} \]
Fully Associative Cache

- Every memory block can go in any location
- Good for: caches with large miss penalties
- # index bits?
Fully Associative Cache

- Every memory block can go in any location
- Good for: caches with large miss penalties
- \# index bits?
  0 - there is only one set!
Set-Associative Cache

- Each memory block can only go to \( n \) locations in cache
- Index: specifies which chunk of \( n \) locations the block can be stored
- Trade-off between direct mapped & fully associative
- \# index bits?
Set-Associative Cache

- Each memory block can only go to n locations in cache
- Index: specifies which chunk of n locations the block can be stored
- Trade-off between direct mapped & fully associative
- # index bits?
  \[ \log_2\left(\frac{\text{cache\_size}}{n}\right) \]
Set-Associative Cache

- Given: 16 bits addressable memory, 4 word blocks, 1024-entry cache, 4 way associativity
- # Tag bits? # Index bits? # offset bits?
Set-Associative Cache

- Given: 16 bits addressable memory, 4 word blocks, 256-block cache, 4 way associativity
- # Tag bits? # Index bits? # offset bits?

  - Offset bits: \( \log_2(4) = 2 \)
  - Index bits: \( \log_2(256) - \log_2(4) = 6 \)
  - Tag bits: \( 14 - 6 = 8 \)
Cache Size

• Large caches hold a lot of data, but are slow

• Small caches are fast, but will miss often

• Cache sizing is a non-trivial problem - no easy answer to cache size
Virtual Memory

- Used in multitasking kernels
- Idea: every program sees a full memory
- OS shows a “virtual memory” to each program & manages the memory for the programs
Virtual Memory

• Segmented or Paged?

• Segmented: divide the memory into variable sizes

• Paged: everything is one uniform size

• Typically, use paging - easier to manage
Virtual Memory

- Page table: a big lookup table that translates between virtual address spaces and physical address spaces
- Pages are typically 4 kB in size
Virtual Memory

• What does this have to do with architecture?
• Need to quickly map between virtual and physical addresses
• Enter the Translation Lookaside Buffer (TLB)
Virtual Memory

- TLB: small cache to translate between virtual and physical memory addresses
- Basically a cache for the page table
- Similar issues apply for TLBs as regular caches:
  - Eviction policy
  - Size tradeoffs