Introduction

- Memory Hierarchy
  - A structure that uses multiple levels of memories; as the distance from the processor increases, the size of the memories and the access time both increase.
  - Fact: Large memories are slow and fast memories are small.
  - How do we create a memory that gives the illusion of being large and fast?
    - With hierarchy.
    - With parallelism.
Chapter 5 — Large and Fast: Exploiting Memory Hierarchy

Major Components of a Computer

Processor
- Control
- Datapath

Memory

Devices
- Input
- Output

Cache

Main Memory

Secondary Memory (Disk)

Processor - Memory Performance Gap

“Moore’s Law”

μProc 55%/year (2X/1.5yr)

Processor-Memory Performance Gap (grows 50%/year)

DRAM 7%/year (2X/10yrs)
Cache Hierarchies

- Internet browsers cache web pages – Why?
- Data and instructions are stored in Main Memory composed of dynamic memory (DRAM). DRAM is a technology that has high bit density, but relatively poor latency – an access to data in memory can take as many as 300 cycles today.
- Hence, some data is stored on the processor in a structure called the cache – caches employ Static Ram (SRAM) technology, which is faster, but has lower bit density.

A Typical Memory Hierarchy

- Takes advantage of the principle of locality to present the user with as large a memory as possible in the cheapest technology at the fastest speed.
The Memory Hierarchy: Why Does it Work?

- **Temporal Locality** - locality in time
  - If a memory location is referenced then it tends to be referenced again soon:
    - Keep *most recently accessed* data items closer to the processor.
- **Spatial Locality** - locality in space
  - If a memory location is referenced, the locations with nearby addresses tend to be referenced also:
    - Move blocks consisting of *contiguous words* closer to the processor.

Taking Advantage of Locality

- Store *everything* on disk (level furthest from the CPU).
- Copy recently accessed (and nearby) items from disk to smaller DRAM (main) memory.
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM (cache) memory.
- Ideal memory:
  - Access time of SRAM.
  - Capacity and cost/GB of disk.
Memory Hierarchy Technologies

- Caches use Static RAM for speed and technology compatibility:
  - Fast (typical access times of 0.5 to 2.5 nsec).
  - Low density (6 transistor cells), higher power, expensive.
  - Content lasts as long as power is on.
- Main memory uses Dynamic RAM for size (density):
  - Slower (typical access times of 50 to 70 nsec).
  - High density (1 transistor cells), lower power, cheaper.
  - Needs to be “refreshed” regularly (~ every 8 msec):
    - Consumes 1% to 2% of the active cycles of the DRAM.

Memory Costs as of 2016

<table>
<thead>
<tr>
<th>Memory Technology</th>
<th>Typical Access Time</th>
<th>$ per GiB in 2016</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static RAM</td>
<td>0.5 – 2.5 ns</td>
<td>$500 - $1000</td>
</tr>
<tr>
<td>Dynamic RAM</td>
<td>50 - 70 ns</td>
<td>$10 - $20</td>
</tr>
<tr>
<td>Flash memory</td>
<td>5000 – 50,000 ns</td>
<td>$0.75 - $1.00</td>
</tr>
<tr>
<td>Magnetic Disk</td>
<td>5,000,000 – 20,000,000 ns</td>
<td>$0.05 - $0.10</td>
</tr>
</tbody>
</table>
The Memory Hierarchy: Terminology

- **Block** or **Line** - the minimum unit of information that is accessed in a cache.
- **Hit Rate** - the fraction of memory accesses found in a given level of the memory hierarchy:
  - **Hit Time** – the time to access that level which consists of:
    - Time to access the block + Time to determine hit/miss.
- **Miss Rate** - the fraction of memory accesses not found in a level of the memory hierarchy $\Rightarrow 1 - (\text{Hit Rate})$
- **Miss Penalty** – the time to replace a block in that level with the corresponding block from a lower level which consists of:
  - Time to access the block in the lower level + Time to transmit that block to the level that experienced the miss + Time to insert the block in that level + Time to pass the block to the requester.

Hit Time $\ll$ Miss Penalty

Characteristics of the Memory Hierarchy

Increasing distance from the processor in access time.

- Processor
- L1$: 4-8 bytes (word)
- L2$: 8-32 bytes (block)
- Main Memory: 1 to 4 blocks
- Secondary Memory: 1,024+ bytes (disk sector)

Inclusive—what is in L1$ is a subset of what is in L2$ is a subset of what is in MM is a subset of what is in SM.

(Relative) size of the memory at each level
### DRAM Generations

<table>
<thead>
<tr>
<th>Year Introduced</th>
<th>Chip size</th>
<th>$ per GIB</th>
<th>Total access time to a new row/column</th>
<th>Average column access time to existing row</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td>64 Kbit</td>
<td>$1,500,000</td>
<td>250 ns</td>
<td>150 ns</td>
</tr>
<tr>
<td>1983</td>
<td>256 Kbit</td>
<td>$500,000</td>
<td>185 ns</td>
<td>100 ns</td>
</tr>
<tr>
<td>1985</td>
<td>1 Megabit</td>
<td>$200,000</td>
<td>135 ns</td>
<td>40 ns</td>
</tr>
<tr>
<td>1989</td>
<td>4 Megabit</td>
<td>$50,000</td>
<td>110 ns</td>
<td>40 ns</td>
</tr>
<tr>
<td>1992</td>
<td>16 Megabit</td>
<td>$15,000</td>
<td>90 ns</td>
<td>30 ns</td>
</tr>
<tr>
<td>1996</td>
<td>64 Megabit</td>
<td>$10,000</td>
<td>60 ns</td>
<td>12 ns</td>
</tr>
<tr>
<td>1998</td>
<td>128 Megabit</td>
<td>$4,000</td>
<td>60 ns</td>
<td>10 ns</td>
</tr>
<tr>
<td>2000</td>
<td>256 Megabit</td>
<td>$1,000</td>
<td>55 ns</td>
<td>7 ns</td>
</tr>
<tr>
<td>2004</td>
<td>512 Megabit</td>
<td>$250</td>
<td>50 ns</td>
<td>5 ns</td>
</tr>
<tr>
<td>2007</td>
<td>1 Gbit</td>
<td>$50</td>
<td>45 ns</td>
<td>1.25 ns</td>
</tr>
<tr>
<td>2010</td>
<td>2 Gbit</td>
<td>$30</td>
<td>40 ns</td>
<td>1 ns</td>
</tr>
<tr>
<td>2012</td>
<td>4 Gbit</td>
<td>$1</td>
<td>35 ns</td>
<td>0.8 ns</td>
</tr>
</tbody>
</table>

### Cache Basics

- Two questions to answer in hardware:
  - Q1: How do we know if a data item is in the cache?
  - Q2: If it is in the cache, how do we find it?
- Direct mapped method
  - Each memory block is mapped to exactly one block in the cache
    - Lots of lower level blocks must *share* blocks in the cache.
  - How do we know which particular block is stored in a cache location?
    - Store a portion of the block address as well as the data.
    - The higher-order address bits stored with the data are called the *tag*.
  - What if there is no data in a location?
    - Valid bit: 1 = present, 0 = not present
    - Initially set to 0.
Concluding Remarks

- Fast memories are small, large memories are slow:
  - We really want fast, large memories 😊
  - Caching gives this illusion 😊
- Principle of locality:
  - Programs use a small part of their memory space frequently.
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk