The Memory Hierarchy

CMPU 224 – Computer Organization
Jason Waterman
Random-Access Memory (RAM)

• Key features
  • Can access any address in any order
    • Think of RAM as a big array
  • Lose information if powered off
  • RAM is traditionally packaged as a chip
  • Basic storage unit is normally a cell (one bit per cell)
  • Multiple RAM chips form a memory

• RAM comes in two varieties:
  • DRAM (Dynamic RAM)
    • Main memory in a computer system
  • SRAM (Static RAM)
    • Used in computer caches
Memory Arrays: DRAM
Memory Arrays: SRAM
Relative Memory Sizes of SRAM vs. DRAM

1 Memory cell in 0.5μm processes
a) Gate Array SRAM
b) Embedded SRAM
c) Standard SRAM (6T cell with local interconnect)
d) ASIC DRAM
e) Standard DRAM (stacked cell)

[From Foss, R.C. “Implementing Application-Specific Memory”, ISSCC 1996]
## SRAM vs DRAM Summary

<table>
<thead>
<tr>
<th></th>
<th>Trans. per bit</th>
<th>Access time</th>
<th>Needs refresh?</th>
<th>Sensitivity?</th>
<th>Cost</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM</td>
<td>6</td>
<td>1X</td>
<td>No</td>
<td>No</td>
<td>1000x</td>
<td>Cache memories</td>
</tr>
<tr>
<td>DRAM</td>
<td>1</td>
<td>10X</td>
<td>Yes</td>
<td>Yes</td>
<td>1X</td>
<td>Main memories, frame buffers</td>
</tr>
</tbody>
</table>

### Diagram
- **Dynamic RAM cell (DRAM)**
  - Word Line (WL)
  - Bit Line (BL)
  - Data
  - Cs

---

4/20/22 :: CMPU 224 -- Computer Organization :: 6
Nonvolatile Memories

• Nonvolatile memories retain value even if powered off
  • Read-only memory (ROM): programmed during production
  • Programmable ROM (PROM): can be programmed once
  • Erasable PROM (EPROM): can be bulk erased (UV, X-Ray)
  • Electrically erasable PROM (EEPROM): electronic erase capability
  • Flash memory: EEPROMs with partial (block-level) erase capability
    • Wears out after about 100,000 erasings

• Uses for Nonvolatile Memories
  • Firmware programs stored in a ROM (BIOS, controllers for disks, network cards, graphics accelerators, security subsystems,...)
  • Solid state disks (laptops, thumb drives, smart phones, tablets)
  • Disk caches
A bus is a collection of parallel wires that carry address, data, and control signals.

Buses are typically shared by multiple devices.
Memory Read Transaction

• Let’s go through the steps in reading a memory value

Load operation: `movq (%rdi), %rax`
Memory Read Transaction (1)

- CPU places address A on the memory bus

Load operation: \texttt{movq} (\%rdi), \%rax
Memory Read Transaction (2)

- Main memory reads address A from the memory bus, retrieves word x, and places it on the bus

Load operation: `movq (%rdi), %rax`
Memory Read Transaction (3)

- CPU read word x from the bus and copies it into register %rax

Load operation: movq (%rdi), %rax
Memory Write Transaction

- Let’s go through the steps in writing a value to memory

Store operation: `movq %rax, (%rdi)`
Memory Write Transaction (1)

- CPU places address A on bus
- Main memory reads it and waits for the corresponding data word to arrive

Store operation: \texttt{movq %rax, (%rdi)}
Memory Write Transaction (2)

- CPU places data word \( y \) on the bus

Store operation: `movq %rax, (%rdi)`
Memory Write Transaction (3)

- Main memory reads data word y from the bus and stores it at address A

Store operation: `movq %rax, (%rdi)`
The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds

![Graph showing the gap widens between DRAM, disk, and CPU speeds over time.](Image)
The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds

Log Scale

The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds

Log Scale
The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds

The diagram shows the gap between DRAM, disk, and CPU speeds over time, with DRAM being 10x slower than CPU. The Y-axis represents time in nanoseconds (ns), with a log scale, and the X-axis represents years from 1985 to 2015.
The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds

**The gap widens between DRAM, disk, and CPU speeds**

- **Disk seek time**
- **SSD access time**
- **DRAM access time**
- **SRAM access time**
- **CPU cycle time**
- **Effective CPU cycle time**
How Big is a Nanosecond

• See an explanation from Vassar’s very own Grace Hopper
  • https://www.youtube.com/watch?v=9eyFDBPk4Yw
Memory Systems

• Registers
  • Item on our desk ~1 foot

• Main Memory
  • Item in our building ~100 feet

• Hard Drive
  • Item is in another state ~190 miles
Cache Memories

- **Cache memories** are small, fast SRAM-based memories managed automatically in hardware
  - Hold frequently accessed blocks of main memory
- **CPU** looks first for data in cache, then in main memory
- **Typical system structure:**

![Diagram of computer organization]

- CPU chip
- Register file
- ALU
- Cache memory
- Bus interface
- System bus
- Memory bus
- I/O bridge
- Main memory

4/20/22
CMPU 224 -- Computer Organization
Cache Memories

• **Cache memories** are small, fast SRAM-based memories managed automatically in hardware
  - Hold frequently accessed blocks of main memory
• CPU looks first for data in cache, then in main memory
• Typical system structure:
Memory Systems

- Registers
  - Item on our desk ~1 foot

- Cache Memory
  - Items in the same room ~10 feet

- Main Memory
  - Item in our building ~100 feet

- Hard Drive
  - Item is in another state ~180 miles
Memory Hierarchies

• Some fundamental and enduring properties of hardware and software:
  • Fast storage technologies cost more per byte, have less capacity, and require more power (heat!)
  • The gap between CPU and main memory speed is widening

• These fundamental properties complement each other beautifully

• They suggest an approach for organizing memory and storage systems known as a memory hierarchy
Memory Hierarchies

• Some fundamental and enduring properties of hardware and software:
  • Fast storage technologies cost more per byte, have less capacity, and require more power (heat!)
  • The gap between CPU and main memory speed is widening

• These fundamental properties complement each other beautifully

• They suggest an approach for organizing memory and storage systems known as a memory hierarchy
But the cache is small, why does it help?

• Locality to the rescue!
  • The key to bridging this CPU-Memory gap is a fundamental property of computer programs known as **locality**

• **Principle of Locality:** Programs tend to use data and instructions with addresses near or equal to those they have used recently

• **Temporal locality:**
  • Recently referenced items are likely to be referenced again in the near future

• **Spatial locality:**
  • Items with nearby addresses tend to be referenced close together in time
Locality Example

sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;

• Data references
  • Reference array elements in succession (stride-1 reference pattern)  Spatial locality
  • Reference variable sum each iteration  Temporal locality

• Instruction references
  • Reference instructions in sequence  Spatial locality
  • Cycle through loop repeatedly  Temporal locality
Qualitative Estimates of Locality

• **Claim:** Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer

• **Question:** Does this function have good locality with respect to array `a`?

```c
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];

    return sum;
}
```
Locality Example

• **Question:** Does this function have good locality with respect to array `a`?

```c
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```
Locality Example

- **Question**: Can you permute the loops so that the function scans the 3D array `a` with a stride-1 reference pattern (and thus has good spatial locality)?

```c
int sum_array_3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < N; k++)
                sum += a[k][i][j];

    return sum;
}
```
Caches

• **Cache**: A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device
  - Keep a copy of popular data in the smaller, faster memory

• Fundamental idea of a memory hierarchy:
  - For each $k$, the faster, smaller device at level $k$ serves as a cache for the larger, slower device at level $k+1$

• Why do memory hierarchies work?
  - Because of locality, programs tend to access the data at level $k$ more often than they access the data at level $k+1$
  - Thus, the storage at level $k+1$ can be slower, and thus larger and cheaper per bit

• **Big Idea**: The memory hierarchy creates a large pool of storage that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top
Example Memory Hierarchy

- **L0:** CPU registers hold words retrieved from the L1 cache
- **L1:** L1 cache holds cache lines retrieved from the L2 cache
- **L2:** L2 cache holds cache lines retrieved from L3 cache
- **L3:** L3 cache holds cache lines retrieved from main memory
- **L4:** Main memory holds disk blocks retrieved from local disks
- **L5:** Local disks hold files retrieved from disks on remote servers
- **L6:** Remote secondary storage (e.g., Web servers)

Smaller, faster, and costlier (per byte) storage devices

Larger, slower, and cheaper (per byte) storage devices
Summary

• The speed gap between CPU, memory and mass storage continues to widen

• Well-written programs exhibit a property called locality

• Memory hierarchies based on caching close this gap by exploiting locality
  • Keep a subset of popular memory items in a smaller, faster memory

• Next time:
  • How caches are used and organized
  • How to find data in the cache (cache lookups)