The Memory Hierarchy

Reminders:
Lab 5 due on Thursday
Quiz 2 – Tuesday, Nov 10th
Memory Technologies

- Technologies have vastly different tradeoffs between capacity, latency, bandwidth, energy, and cost
  - ... and thus different applications

<table>
<thead>
<tr>
<th>Memory Technology</th>
<th>Capacity</th>
<th>Latency</th>
<th>Cost/GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>100s of bits</td>
<td>20 ps</td>
<td>$$$$$</td>
</tr>
<tr>
<td>SRAM</td>
<td>~10 KB-10 MB</td>
<td>1-10 ns</td>
<td>~$1000</td>
</tr>
<tr>
<td>DRAM</td>
<td>~10 GB</td>
<td>80 ns</td>
<td>~$10</td>
</tr>
<tr>
<td>Flash*</td>
<td>~100 GB</td>
<td>100 us</td>
<td>~$1</td>
</tr>
<tr>
<td>Hard disk*</td>
<td>~1 TB</td>
<td>10 ms</td>
<td>~$0.1</td>
</tr>
</tbody>
</table>

* non-volatile (retains contents when powered off)
Memory Technologies:
SRAM, DRAM, Flash, Hard Disk

NOTE: Demystification, will not be on the quiz
Static RAM (SRAM)

Drivers

SRAM cell

Wordlines (horizontal)

Bitlines (vertical)

Sense amplifiers

8x6 SRAM array

Address decoder

Data in

6

Address

3

Data out

6

October 27, 2020

MIT 6.004 Fall 2020
SRAM Cell

6-transistor (6T) cell:
- Two CMOS inverters (4 FETs) forming a bistable element
- Two access transistors

Bistable element (two stable states) stores a single bit

- Vdd
- GND
- Vdd
- GND

October 27, 2020
SRAM Read

1. Drivers precharge all bitlines to Vdd (1), and leave them floating.
2. Address decoder activates one wordline.
3. Each cell in the activated word slowly pulls down one of the bitlines to GND (0).
4. Sense amplifiers sense change in bitline voltages, produce output data.
1. Drivers set and hold bitlines to desired values (Vdd and GND for 1, GND and Vdd for 0)

2. Address decoder activates one wordline

3. Each cell in word is overpowered by the drivers, stores value

Cell transistors are carefully sized so that bitline GND overpowers cell Vdd, but bitline Vdd does not overpower cell GND
Multiported SRAMs

- SRAM so far can do either one read or one write/cycle
- We can do multiple reads and writes with multiple ports by adding one set of wordlines and bitlines per port

- **Cost/bit for N ports?**
  - Wordlines? \( N \)
  - Bitlines? \( 2 \times N \)
  - Access FETs? \( 2 \times N \)

- Wires dominate area \( \rightarrow O(N^2) \) area!
Summary: SRAMs

- Array of $k \times b$ cells (k words, b cells per word)
- Cell is a bistable element + access transistors
  - Analog circuit with carefully sized transistors
- Read: Precharge bitlines, activate wordline, sense
- Write: Drive bitlines, activate wordline, overpower cells

- 6 FETs/cell... can we do better?
1T Dynamic RAM (DRAM) Cell

Storage capacitor

1T DRAM Cell

wordline

access FET

bitline

✓ ~20x smaller area than SRAM cell → Denser and cheaper!

✗ Problem: Capacitor leaks charge, must be refreshed periodically (~milliseconds)

Trench capacitors take little area
DRAM Writes and Reads

- **Writes**: Drive bitline to Vdd or GND, activate wordline, charge or discharge capacitor

- **Reads**:
  1. Precharge bitline to Vdd/2
  2. Activate wordline
  3. Capacitor and bitline share charge
     - If capacitor was discharged, bitline voltage decreases slightly
     - If capacitor was charged, bitline voltage increases slightly
  4. Sense bitline to determine if 0 or 1

- **Issue**: **Reads are destructive!** (charge is gone!)
  - Data must be rewritten to cells at end of read
Non-Volatile Storage: Flash

Flash Memory: Use “floating gate” transistors to store charge (floating gate is a well insulated conductor)

- **Very dense**: Multiple bits/transistor, read and written in blocks
- **Slow** (especially on writes), 10-100 us
- **Limited number of writes**: charging/discharging the floating gate (writes) requires large voltages that damage transistor

Electrons here diminish strength of field from control gate ⇒ no inversion ⇒ NFET stays off even when word line is high.

Cyferz (CC BY 2.5)
Non-Volatile Storage: Hard Disk

Hard Disk: Rotating magnetic platters + read/write head
- Extremely slow (~10ms): Mechanically move head to position, wait for data to pass underneath head
- ~100MB/s for sequential read/writes
- ~100KB/s for random read/writes
- Cheap
The Memory Hierarchy
Different technologies have vastly different tradeoffs
Size is a fundamental limit, even setting cost aside:
  - Small + low latency, high bandwidth, low energy, or
  - Large + high-latency, low bandwidth, high energy
Can we get best of both worlds? (large, fast, cheap)
The Memory Hierarchy

Want large, fast, and cheap memory, but...
Large memories are slow (even if built with fast components)
Fast memories are expensive

Solution: Use a hierarchy of memories with different tradeoffs to fake a large, fast, cheap memory.
Memory Hierarchy Interface

Approach 1: Expose Hierarchy
- Registers, SRAM, DRAM, Flash, Hard Disk each available as storage alternatives
- Tell programmers: “Use them cleverly”

Approach 2: Hide Hierarchy
- Programming model: Single memory, single address space
- Machine transparently stores data in fast or slow memory, depending on usage patterns
Typical Memory Access Patterns

- array accesses
- local variable accesses
- procedure calls
- loop

address

data

stack

code

time
Common Predictable Patterns

- Two predictable properties of memory accesses:
  - Temporal locality: If a location has been accessed recently, it is likely to be accessed (reused) soon
  - Spatial locality: If a location has been accessed recently, it is likely that nearby locations will be accessed soon
Caches

- Cache: A small, interim storage component that transparently retains (caches) data from recently accessed locations

- Processor sends accesses to cache. Two options:
  - **Cache hit**: Data for this address in cache, returned quickly
  - **Cache miss**: Data not in cache
    - Fetch data from memory, send it back to processor
    - Retain this data in the cache (replacing some other data)
  - Processor must deal with variable memory access time
A Typical Memory Hierarchy

Computers use many levels of caches:

<table>
<thead>
<tr>
<th>Access time</th>
<th>Capacity</th>
<th>Managed By</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 cycle</td>
<td>1 KB</td>
<td>Software/Compiler</td>
</tr>
<tr>
<td>2-4 cycles</td>
<td>32 KB</td>
<td>Hardware</td>
</tr>
<tr>
<td>10 cycles</td>
<td>256 KB</td>
<td>Hardware</td>
</tr>
<tr>
<td>40 cycles</td>
<td>10 MB</td>
<td>Hardware</td>
</tr>
<tr>
<td>200 cycles</td>
<td>10 GB</td>
<td>Software/OS</td>
</tr>
<tr>
<td>10-100us</td>
<td>100 GB</td>
<td>Software/OS</td>
</tr>
<tr>
<td>10ms</td>
<td>1 TB</td>
<td>Software/OS</td>
</tr>
</tbody>
</table>
Cache Metrics

- **Hit Ratio**: 
  \[ HR = \frac{\text{hits}}{\text{hits} + \text{misses}} = 1 - MR \]

- **Miss Ratio**: 
  \[ MR = \frac{\text{misses}}{\text{hits} + \text{misses}} = 1 - HR \]

- **Average Memory Access Time (AMAT)**: 
  \[ AMAT = \text{HitTime} + \text{MissRatio} \times \text{MissPenalty} \]

- Goal of caching is to improve AMAT
- Formula can be applied recursively in multi-level hierarchies:

  \[ AMAT = \text{HitTime}_{L_1} + \text{MissRatio}_{L_1} \times AMAT_{L_2} = \]
  \[ AMAT = \text{HitTime}_{L_1} + \text{MissRatio}_{L_1} \times (\text{HitTime}_{L_2} + \text{MissRatio}_{L_2} \times AMAT_{L_3}) = \ldots \]
Example: How High of a Hit Ratio?

What hit ratio do we need to break even? (Main memory only: AMAT = 100)

\[ 100 = 4 + (1 - HR) \times 100 \Rightarrow HR = 4\% \]

What hit ratio do we need to achieve AMAT = 5 cycles?

\[ 5 = 4 + (1 - HR) \times 100 \Rightarrow HR = 99\% \]
Basic Cache Algorithm (Reads)

On reference to Mem[X], look for X among cache tags

HIT: X = Tag(i) for some cache line i
Return Data(i)

MISS: X not found in Tag of any cache line
Read Mem[X]
Return Mem[X]
Select a line k to hold Mem[X]
Write Tag(k) = X, Data(k) = Mem[X]

Q: How do we “search” the cache?
Thank you!

Next lecture: Cache Tradeoffs