Memory Hierarchy

Why are you dressed like that? It's not Halloween

It makes me look faster, don't you think?

• Memory Flavors
• Principle of Locality
• Program Traces
• Memory Hierarchies
• Associativity

(Study Chapter 7)

Quiz #2 Information

Logistics:
   Date:      Tuesday, 5/4
   Format:    Open book, Open notes, No Computers

Coverage:
   Until Lecture 17 (pipelined CPUs)

Under your desk you'll find some sand, boron, phosphorous and aluminum… build a 1 GHz pipelined CPU. Extra credit: booting Linux…
What We Want in a Memory

<table>
<thead>
<tr>
<th>Capacity</th>
<th>Latency</th>
<th>Cost/Gb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>1000's of bits</td>
<td>20 ps</td>
</tr>
<tr>
<td>SRAM</td>
<td>100's Kbytes</td>
<td>1 ns</td>
</tr>
<tr>
<td>DRAM</td>
<td>100's Mbytes</td>
<td>40 ns</td>
</tr>
<tr>
<td>Hard disk*</td>
<td>100's Gbytes</td>
<td>10 ms</td>
</tr>
<tr>
<td>Want</td>
<td>1 Gbyte</td>
<td>1 ns</td>
</tr>
</tbody>
</table>

* non-volatile

SRAM Memory Cell

There are two bit-lines per column, one supplies the bit the other it’s complement.

On a Read Cycle -
A single word line is activated (driven to “1”), and the access transistors enable the selected cells, and their complements, onto the bit lines.

Writes are similar to reads, except the bit-lines are driven with the desired value of the cell.

The writing has to “overpower” the original contents of the memory cell.
Tricks to Make SRAMs Fast

Forget that it is a digital circuit
1) Precharge the bit lines prior to the read (for instance – while the address is being decoded) because the access FETs are good pull-downs and poor pull-ups
2) Use a differential amplifier to “sense” the difference in the two bit-lines long before they reach a valid logic levels.

Multiport SRAMs (a.k.a. Register Files)

One can increase the number of SRAM ports by adding access transistors. By carefully sizing the inverter pair, so that one is strong and the other weak, we can assure that our WRITE bus will only fight with the weaker one, and the READs are driven by the stronger one. Thus minimizing both access and write times.

What is the cost per cell of adding a new read or write port?
1-T Dynamic Ram

Six transistors/cell may not sound like much, but they can add up quickly. What is the fewest number of transistors that can be used to store a bit?

\[ C = \varepsilon \frac{A}{d} \]

C in storage capacitor determined by:
- better dielectric
- more area
- thinner film

Six transistors/cell may not sound like much, but they can add up quickly. What is the fewest number of transistors that can be used to store a bit?

Side-by-Side Comparison

One 6-T SRAM Cell
- \( V_{DD} \)
- \( V_{SS} \)
- inverter pullup
- inverter pulldown
- strapped word line
- access FET

Two 1-T DRAM Cells
- poly word line
- metal bit line
Tricks for Increasing Throughput

The first thing that should pop into your mind when asked to speed up a digital design...

PIPELINING

Synchronous DRAM (SDRAM)

Column Multiplexer/Shifter

Row Address Decoder

Row 1
Row 2
Row 2^N

Col. 1
Col. 2
Col. 3
Col. 2^N

Multiplexed Address

word lines

bit lines

memory cell (one bit)

D

Pipe

Data out

Clock

Hard Disk Drives

Typical high-end drive:
• Average latency = 4 ms
• Average seek time = 9 ms
• Transfer rate = 40 M bytes/sec
• Capacity = 250 G byte
• Cost = $250

Figure from www.pctechguide.com
Quantity vs Quality...

Your memory system can be
• BIG and SLOW... or
• SMALL and FAST.

We’ve explored a range of
 circuit-design trade-offs.

Is there an
 ARCHITECTURAL solution
to this DELIMA?

Best of Both Worlds

What we WANT:  A BIG, FAST memory!

We’d like to have a memory system that
• PERFORMS like 2 GBytes of SRAM; but
• COSTS like 512 MBytes of slow memory.

SURPRISE: We can (nearly) get our wish!

KEY: Use a hierarchy of memory technologies:
Key IDEA

- Keep the most often-used data in a small, fast SRAM (often local to CPU chip)
- Refer to Main Memory only rarely, for remaining data.

The reason this strategy works: LOCALITY

Locality of Reference:
Reference to location X at time t implies that reference to location X+ΔX at time t+Δt becomes more probable as ΔX and Δt approach zero.

Typical Memory Reference Patterns

MEMORY TRACE – A temporal sequence of memory references (addresses) from a real program.

TEMPORAL LOCALITY – If an item is referenced, it will tend to be referenced again soon.

SPATIAL LOCALITY – If an item is referenced, nearby items will tend to be referenced soon.
**Working Set**

- $S$ is the set of locations accessed during $\Delta t$.
- Working set: a set $S$ which changes slowly w.r.t. access time.
- Working set size, $|S|$.

**Exploiting the Memory Hierarchy**

**Approach 1 (Cray, others): Expose Hierarchy**
- Registers, Main Memory, Disk each available as storage alternatives;
- Tell programmers: "Use them cleverly".

**Approach 2: Hide Hierarchy**
- Programming model: SINGLE kind of memory, single address space.
- Machine AUTOMATICALLY assigns locations to fast or slow memory, depending on usage patterns.
Why We Care

CPU performance is dominated by memory performance.
More significant than:
ISA, circuit optimization, pipelining, super-scalar, etc

TRICK #1: How to make slow MAIN MEMORY appear faster than it is.

Technique: CACHEING – Next 2 Lectures

TRICK #2: How to make a small MAIN MEMORY appear bigger than it is.

Technique: VIRTUAL MEMORY – Lecture after that

The Cache Idea:
Program-Transparent Memory Hierarchy

GOALS:
1) Improve the average access time

- \( \alpha \) HIT RATIO: Fraction of refs found in CACHE.
- \( (1-\alpha) \) MISS RATIO: Remaining references.

\[ t_{ave} = \alpha t_c + (1-\alpha)(t_c + t_m) = t_c + (1-\alpha)t_m \]

2) Transparency (compatibility, programming ease)

Challenge: To make the hit ratio as high as possible.

Cache contains TEMPORARY COPIES of selected main memory locations... eg. Mem[100] = 37
How High of a Hit Ratio?

Suppose we can easily build an on-chip static memory with a 0.8 nS access time, but the fastest dynamic memories that we can buy for main memory have an average access time of 10 nS. How high of a hit rate do we need to sustain an average access time of 1 nS?

\[
\alpha = 1 - \frac{t_{acc} - t_c}{t_m} = 1 - \frac{1 - 0.8}{10} = 98\%
\]

WOW, a cache really needs to be good?

The Cache Principle

Find “Hart, Lee”

5-Minute Access Time:

5-Second Access Time:

ALGORITHM: Look on your desk for the requested information first, if it's not there check secondary storage
Basic Cache Algorithm

ON REFERENCE TO Mem[X]: Look for X among cache tags...

HIT: \( X = TAG(i) \), for some cache line \( i \)

READ: return DATA(i)  
WRITE: change DATA(i);  
      Start Write to Mem(X)

MISS: \( X \) not found in TAG of any cache line

REPLACEMENT SELECTION:
      Select some LINE \( k \) to hold Mem[X] (Allocation)

READ: Read Mem[X]  
      Set TAG(k)=X, DATA(K)=Mem[X]

WRITE: Start Write to Mem(X)  
        Set TAG(k)=X, DATA(K)= new Mem[X]

---

Associativity: Parallel Lookup

Find “Hart, Lee”

Nope, “Smith”
Nope, “Jones”

HERE IT IS!

Nope, “LeVile”
Fully-Associative Cache

The extreme in associatively: All comparisons made in parallel
Any data item could be located in any cache location

Direct-Mapped Cache

(NO Parallelism:
Look in JUST ONE place, determined by parameters of incoming request (address bits)
... can use ordinary RAM as table)
Direct Mapped Cache

Low-cost extreme:
Single comparator
Use ordinary (fast) static RAM for cache tags & data:

![Diagram of Direct Mapped Cache]

**Incoming Address**

- T: Upper-address bits
- K: K-bit Cache Index

**K x (T + D)-bit static RAM**

**Tag**

**Data**

**DISADVANTAGE:** COLLISIONS

**QUESTION:** Why not use HIGH-order bits as the Cache Index?

The Problem with Collisions

**PROBLEM:**
Contention among H's....
- CAN'T cache both "Hart" & "Heel"

... Suppose H's tend to come at once?

==> BETTER IDEA:
File by LAST letter!
Cache Questions = Cash Questions

What lies between Fully Associate and Direct-Mapped?
When I put something new into the cache, what data gets thrown out?
How many processor words should there be per tag?
When I write to cache, should I also write to memory?
What do I do when a write misses cache, should space in cache be allocated for the written address.
What if I have INPUT/OUTPUT devices located at certain memory addresses, do we cache them?

Answers: Stay Tuned