The Big Picture: Where are we now?

So far: Focus on processor datapath and control

Next: Focus on the memory system

Today's Lecture - Caches

- Memory hierarchy
- Locality
- Static memory design
- Cache design

Since 1980, CPU has outpaced DRAM ...

Q. How do architects address this gap?
A. Put smaller, faster "cache" memories between CPU and PRAM. Create a "memory hierarchy".

1977: DRAM faster than microprocessors

Apple ][ (1977)

CPU: 1000 ns
DRAM: 400 ns

Basic Idea: Variable-latency memory port

Data in upper memory returned with lower latency.
Data in lower level returned with higher latency.

Q. How do architects address this gap?
A. Put smaller, faster "cache" memories between CPU and PRAM. Create a "memory hierarchy".
Discuss cache behavior:

- Temporal locality: Keep most recently accessed data closer to processor.
- Spatial locality: Move contiguous blocks in the address space to upper levels.

**2004 Memory Hierarchy: Apple iMac G5**

- Managed by compiler
- Managed by hardware
- Managed by OS, hardware, application

<table>
<thead>
<tr>
<th>Reg</th>
<th>L1 Inst</th>
<th>L1 Data</th>
<th>L2</th>
<th>DRAM</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (cycles)</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>11</td>
<td>88</td>
</tr>
</tbody>
</table>

**Goal: Illusion of large, fast, cheap memory**

Let programs address a memory space that scales to the disk size, at a speed that is usually as fast as register access.

**Homework 3**
- Due: October 20 (Wednesday)
- Final demo: October 22 (Friday)
- Report due: October 25 (Monday, 11:59 PM)

**iMac's PowerPC 970: All caches on-chip**

- L1 (64K Instruction)
- L1 (32K Data)
- L2
- 512K
- L2

**Latency: A closer look**

**Read latency:** Time to return first byte of a random access

<table>
<thead>
<tr>
<th>Reg</th>
<th>L1 Inst</th>
<th>L1 Data</th>
<th>L2</th>
<th>DRAM</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (cycles)</td>
<td>0.6n</td>
<td>1.9n</td>
<td>1.9n</td>
<td>6.9n</td>
<td>55n</td>
</tr>
<tr>
<td>Latency (sec)</td>
<td>1.6G</td>
<td>533M</td>
<td>533M</td>
<td>145M</td>
<td>18M</td>
</tr>
</tbody>
</table>

**Architect's latency toolkit:**

1. Parallelism. Request data from N 1-bit-wide memories at the same time. Overlaps latency cost for all N bits. Provides N times the bandwidth.
2. Pipeline memory. If memory has N cycles of latency, issue a request each cycle, receive it N cycles later.
Caching terminology

**Hit:** Data appears in upper level block (ex: Blk X)

**Hit Rate:** The fraction of memory accesses found in upper level.

**Hit Time:** Time to access upper level. Includes hit/miss check.

**Miss:** Data retrieval from lower level needed (Ex: Blk Y)

**Miss Rate:** 1 - Hit Rate

**Miss Penalty:** Time to replace block in upper level + deliver to CPU

Static Memory Design

Review: Two inverters store a bit

The other elements in a memory circuit control reading and writing

Example: Flip-Flop

16-transistor circuit.

Most transistors implement read/write semantics

Putting it all together: an SRAM array

Architects specify number of rows and columns. Word and bit lines slow down as array grows larger!

Cache Design Example

For use in arrays: Static RAM (SRAM) cell

Writing a bit

Drive bit lines with new data and activate word line

Reading a bit

Activate word line let cell drive bit lines.
CPU address space: An array of “blocks”

32-bit Memory Address  Block #  32-byte blocks

<table>
<thead>
<tr>
<th>Which block?</th>
<th>Byte #</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
</tbody>
</table>

27 bits  5 bits

The job of a cache is to hold a “popular” subset of blocks.

One approach: Fully Associative Cache

Cache Tag (27 bits)  Byte Select

Block # ("Tags")

Ex: 0x01

Cache Data Holds 4 blocks

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>...</td>
<td>1</td>
</tr>
</tbody>
</table>

Conclusions

- Program locality is why building a memory hierarchy makes sense
- Latency toolkit: hierarchy design, bit-wise parallelism, pipelining.
- In practice: how many rows, how many columns, how many arrays.
- Cache operation: compare tags, detect hits, select bytes.