Review: Major Components of a Computer

- Processor
  - Control
  - Datapath
- Memory
- Devices
  - Input
  - Output
- Cache
- Main Memory
- Secondary Memory (Disk)

The “Memory Wall”
- The Processor vs DRAM speed disparity continues to grow

- Good memory hierarchy (cache) design is increasingly important to overall performance
The Memory Hierarchy

- Take advantage of the principle of locality to present the user with as much memory as is available in the cheapest technology at the speed offered by the fastest technology.

![Memory Hierarchy Diagram]

The Memory Hierarchy: Why Does it Work?

- **Temporal Locality** (Locality in Time):
  - Keep most recently accessed data items closer to the processor.

- **Spatial Locality** (Locality in Space):
  - Move blocks consisting of contiguous words to the upper levels.

![Memory Hierarchy Diagram with Temporal and Spatial Locality]
The Memory Hierarchy: Terminology

- **Hit**: data is in some block in the upper level (Blk X)
  - **Hit Rate**: fraction of memory accesses found in upper level
  - **Hit Time**: Time to access the upper level which consists of
    - RAM access time + Time to determine hit/miss

- **Miss**: data is not in the upper level so needs to be retrieved from a block in the lower level (Blk Y)
  - **Miss Rate** = 1 - (Hit Rate)
  - **Miss Penalty**: Time to bring in a block from the lower level and replace a block in the upper level with it
    + Time to deliver the block to the processor
  - **Hit Time << Miss Penalty**

How is the Hierarchy Managed?

- **registers ↔ memory**
  - by compiler (programmer?)

- **cache ↔ main memory**
  - by the cache controller hardware

- **main memory ↔ disks**
  - by the operating system (virtual memory)
  - virtual to physical address mapping assisted by the hardware (TLB)
  - by the programmer (files)
The Cache

- Two questions to answer (in hardware):
  - Q1: How do we know if a data item is in the cache?
  - Q2: If it is, how do we find it?

Direct mapped

- For each item of data at the lower level, there is exactly one location in the cache where it might be - so lots of items at the lower level must share locations in the upper level

- Address mapping:
  - (block address) modulo (# of blocks in the cache)

- First consider block sizes of one word

Caching: A Simple First Example

Q1: Is it there?

Compare the cache tag to the high order 2 memory address bits to tell if the memory block is in the cache

Q2: How do we find it?

Use next 2 low order memory address bits – the index – to determine which cache block (i.e., modulo the number of blocks in the cache)
Direct Mapped Cache

Consider the main memory word reference string
Start with an empty cache - all blocks initially marked as not valid

0 miss

1 miss

2 miss

3 miss

4 miss

3 hit

4 hit

15 miss

8 requests, 6 misses

MIPS Direct Mapped Cache Example

One word/block, cache size = 1K words

What kind of locality are we taking advantage of?
Handling Cache Hits

- Read hits (I$ and D$)
  - this is what we want!

- Write hits (D$ only)
  - allow cache and memory to be inconsistent
    - write the data only into the cache (then write-back the cache contents to the memory when that cache block is "evicted")
    - need a dirty bit for each cache block to tell if it needs to be written back to memory when it is evicted
  - require the cache and memory to be consistent
    - always write the data into both the cache and the memory (write-through)
    - don’t need a dirty bit
    - writes run at the speed of the main memory - slow! – or can use a write buffer, so only have to stall if the write buffer is full

Another Reference String Mapping

- Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid

```
0  4  0  4  0  4  0  4
```

- 8 requests, 8 misses
- Ping pong effect due to conflict misses - two memory locations that map into the same cache block
Sources of Cache Misses

- **Compulsory** (cold start or process migration, first reference):
  - First access to a block, “cold” fact of life, not a whole lot you can do about it
  - If you are going to run “millions” of instruction, compulsory misses are insignificant

- **Conflict** (collision)
  - Multiple memory locations mapped to the same cache location
    - Solution 1: increase cache size
    - Solution 2: increase associativity

- **Capacity**
  - Cache cannot contain all blocks accessed by the program
    - Solution: increase cache size

Taking Advantage of Spatial Locality

- Let cache block hold more than one word

  Start with an empty cache - all blocks initially marked as not valid

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Mem(1)</td>
<td>Mem(0)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Mem(3)</td>
<td>Mem(2)</td>
<td>Mem(3)</td>
<td>Mem(2)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Mem(1)</td>
<td>Mem(0)</td>
<td>Mem(3)</td>
<td>Mem(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Mem(5)</td>
<td>Mem(4)</td>
<td>Mem(3)</td>
<td>Mem(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Mem(5)</td>
<td>Mem(4)</td>
<td>Mem(3)</td>
<td>Mem(2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 8 requests, 4 misses
Multiword Block Direct Mapped Cache

- Four words/block, cache size = 1K words

Miss Rate vs Block Size vs Cache Size

- Miss rate goes up if the block size becomes a significant fraction of the cache size because the number of blocks that can be held in the same size cache is smaller (increasing capacity misses)
**Block Size Tradeoff**

- Larger block sizes take advantage of spatial locality **but**
  - If the block size is too big relative to the cache size, the miss rate will go up
  - Larger block size means larger miss penalty
    - Latency to first word in block + transfer time for remaining words

- In general, **Average Memory Access Time**
  \[ \text{Average Memory Access Time} = \text{Hit Time} \times \text{Hit Rate} + \text{Miss Penalty} \times \text{Miss Rate} \]

**Other Ways to Reduce Cache Miss Rates**

1. **Allow more flexible block placement**
   - In a **direct mapped cache** a memory block maps to exactly one cache block
   - At the other extreme, could allow a memory block to be mapped to any cache block – **fully associative cache**
   - A compromise is to divide the cache into **sets** each of which consists of **n “ways” (n-way set associative)**

2. **Use multiple levels of caches**
   - Add a second level of caches on chip – normally a **unified L2 cache** (i.e., it holds both instructions and data)
     - L1 caches focuses on **minimizing hit time** in support of a shorter clock cycle (smaller with smaller block sizes)
     - L2 cache focuses on **reducing miss rate** to reduce the penalty of long main memory access times (larger with larger block sizes)
Improving Cache Performance

Reduce the hit time
- smaller cache
- direct mapped cache
- smaller blocks
- for writes
  - no write allocate – just write to write buffer
  - write allocate – write to a delayed write buffer that then writes to the cache

Reduce the miss penalty
- smaller blocks
  - for large blocks fetch critical word first
- use a write buffer
  - check write buffer (and/or victim cache) on read miss – may get lucky
- use multiple cache levels – L2 cache not tied to CPU clock rate
- faster backing store/improved memory bandwidth
  - wider buses
  - SDRAMs

Reduce the miss rate
- bigger cache
- associative cache
- larger blocks (16 to 64 bytes)
- use a victim cache – a small buffer that holds the most recently discarded blocks

Cache Summary

- The Principle of Locality:
  - Program likely to access a relatively small portion of the address space at any instant of time
    - Temporal Locality: Locality in Time
    - Spatial Locality: Locality in Space

- Three major categories of cache misses:
  - Compulsory misses: sad facts of life, e.g., cold start misses
  - Conflict misses: increase cache size and/or associativity
    Nightmare Scenario: ping pong effect!
  - Capacity misses: increase cache size

- Cache design space
  - total size, block size, associativity (replacement policy)
  - write-hit policy (write-through, write-back)
  - write-miss policy (write allocate, write buffers)