The Motivation for Caches

- Motivation:
  - Large memories (DRAM) are slow
  - Small memories (SRAM) are fast

- Make the *average access time* small by:
  - Servicing most accesses from a small, fast memory.

- Reduce the *bandwidth* required of the large memory
An Expanded View of the Memory System

- **Processor**
  - Control
- **Datapath**
- **Memory**

### Speed:
- Fastest
- Slowest

### Size:
- Smallest
- Biggest

### Cost:
- Highest
- Lowest

Levels of the Memory Hierarchy

**Capacity**

<table>
<thead>
<tr>
<th>Level</th>
<th>Capacity</th>
<th>Access Time</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Registers</td>
<td>100s Bytes</td>
<td>&lt;10s ns</td>
<td>$.01-.001/bit</td>
</tr>
<tr>
<td>Cache</td>
<td>K Bytes</td>
<td>10-100 ns</td>
<td>$.01-.001/byte</td>
</tr>
<tr>
<td>Main Memory</td>
<td>M Bytes</td>
<td>100ns-1us</td>
<td>$.01-.001</td>
</tr>
<tr>
<td>Disk</td>
<td>G Bytes</td>
<td>ms $10^{-3}$-$10^{-4}$</td>
<td>10-6 cents</td>
</tr>
<tr>
<td>Tape</td>
<td>infinite</td>
<td>sec-min $10^{-15}$</td>
<td></td>
</tr>
</tbody>
</table>

**Staging Xfer Unit**

- prog./compiler
  - 1-8 bytes
- cache ctrl
  - 8-128 bytes
- OS
  - 512-4K bytes
- user/operator
  - Mbytes
The Principle of Locality

Program access a relatively small portion of the address space at any instant of time.

Example: 90% of time in 10% of the code

Two Different Types of Locality:
- Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
- Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.

Memory Hierarchy: Principles of Operation

At any given time, data is copied between only 2 adjacent levels:
- Upper Level (Cache): the one closer to the processor
  - Smaller, faster, and uses more expensive technology
- Lower Level (Memory): the one further away from the processor
  - Bigger, slower, and uses less expensive technology

Block:
- The minimum unit of information that can either be present or not present in the two level hierarchy
Memory Hierarchy: Terminology

° Hit: data appears in some block in the upper level (example: Block X)
   • Hit Rate: the fraction of memory access found in the upper level
   • Hit Time: Time to access the upper level which consists of
     RAM access time + Time to determine hit/miss

° Miss: data needs to be retrieved from a block in the lower level (Block Y)
   • Miss Rate = 1 - (Hit Rate)
   • Miss Penalty = Time to replace a block in the upper level +
     Time to deliver the block the processor

° Hit Time << Miss Penalty

Basic Terminology: Typical Values

<table>
<thead>
<tr>
<th>Typical Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block (line) size</td>
</tr>
<tr>
<td>Hit time</td>
</tr>
<tr>
<td>Miss penalty (access time)</td>
</tr>
<tr>
<td>Miss penalty (transfer time)</td>
</tr>
<tr>
<td>Miss rate</td>
</tr>
<tr>
<td>Cache Size</td>
</tr>
</tbody>
</table>
How Does Cache Work?

- **Temporal Locality (Locality in Time):** If an item is referenced, it will tend to be referenced again soon.
  - Keep more recently accessed data items closer to the processor

- **Spatial Locality (Locality in Space):** If an item is referenced, items whose addresses are close by tend to be referenced soon.
  - Move blocks consists of contiguous words to the cache

---

**The Simplest Cache:** Direct Mapped Cache

**Location 0 can be occupied by data from:**
- Memory location 0, 4, 8, ... etc.
- In general: any memory location whose 2 LSBs of the address are 0s
- Address<1:0> => cache index
- (Block Addr) module (#cache blocks)

- How can we tell which one is in the cache?
- Which one should we place in the cache?
Example:

Consider an eight-word direct mapped cache. Show the contents of the cache as it responds to a series of requests (decimal addresses):

22, 26, 22, 26, 16, 3, 16, 18

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Hit</td>
</tr>
<tr>
<td>26</td>
<td>Hit</td>
</tr>
<tr>
<td>22</td>
<td>Hit</td>
</tr>
<tr>
<td>26</td>
<td>Hit</td>
</tr>
<tr>
<td>16</td>
<td>Miss</td>
</tr>
<tr>
<td>3</td>
<td>Miss</td>
</tr>
<tr>
<td>16</td>
<td>Hit</td>
</tr>
<tr>
<td>18</td>
<td>Miss</td>
</tr>
</tbody>
</table>

Cache Organization: Cache Tag and Cache Index

- Assume a 32-bit memory (byte) address:
  - A $2^N$ bytes direct mapped cache:
    - Cache Index: The lower $N$ bits of the memory address
    - Cache Tag: The upper (32 - $N$) bits of the memory address
**Cache Access Example**

**Start Up**

Access 000 01 (miss)

Miss Handling:
- Write Tag & Set V
- Load Data

Access 010 10 (miss)

Load Data
- Write Tag & Set V

Access 000 01 (HIT)

Access 010 10 (HIT)

**Sad Fact of Life:**
- A lot of misses at start up:
  - Compulsory Misses
    - (Cold start misses)

**Definition of a Cache Block**

- Cache Block: the cache data that has its own cache tag

- Our previous “extreme” example (see slide 10):
  - 4-byte Direct Mapped cache: Block Size = 1 Byte
  - Take advantage of Temporal Locality: If a byte is referenced, it will tend to be referenced soon.
  - Did not take advantage of Spatial Locality: If a byte is referenced, its adjacent bytes will be referenced soon.

- In order to take advantage of Spatial Locality: increase the block size

- Question: Can we say “the larger, the better” for the block size?
**Example: 1 KB Direct Mapped Cache with 32 B Blocks**

- For a $2^N$ byte cache:
  - The uppermost $(32 - N)$ bits are always the Cache Tag
  - The lowest $M$ bits are the Byte Select (Block Size = $2^M$)

<table>
<thead>
<tr>
<th>Cache Tag</th>
<th>Example: 0x50</th>
<th>Cache Index</th>
<th>Byte Select</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Ex: 0x01</td>
<td>Ex: 0x00</td>
</tr>
</tbody>
</table>

Valid Bit  | Cache Tag    | Cache Data  |
------------|--------------|-------------|
            | 0x50         | Byte 31 **  | Byte 1 Byte 0 0 |
            |              | Byte 63 **  | Byte 33 Byte 32 1 |
            |              |              | 2 3 |
            |              |              | Byte 1023 ** Byte 992 31 |

**Mapping an Address to a Multiword Cache Block**

Consider a cache with 64 Blocks and a block size of 16 bytes. What block number does byte address 1200 map to?

\[ \frac{1200}{16} \mod 64 = 11 \]
Bits in a cache

How many total bits are required for a directed mapped cache with 64 KB of data and one-word blocks, assuming a 32-bit address?

\[ 2^{14} (32 + (32-14-2)+1) = 2^{14} \times 49 = 784 \text{ Kbits} \]

Block Size Tradeoff

- In general, larger block size take advantage of spatial locality BUT:
  - Larger block size means larger miss penalty:
    - Takes longer time to fill up the block
    - If block size is too big relative to cache size, miss rate will go up

- Average Access Time:
  - \[ \text{Average Access Time} = \text{Hit Time} \times (1 - \text{Miss Rate}) + \text{Miss Penalty} \times \text{Miss Rate} \]
Another Extreme Example

Cache Size = 4 bytes  Block Size = 4 bytes

• Only ONE entry in the cache
• True: If an item is accessed, likely that it will be accessed again soon
  • But it is unlikely that it will be accessed again immediately!!!
  • The next access will likely to be a miss again
    - Continually loading data into the cache but discard (force out) them before they are used again
    - Worst nightmare of a cache designer: Ping Pong Effect

• Conflict Misses are misses caused by:
  • Different memory locations mapped to the same cache index
    - Solution 1: make the cache size bigger
    - Solution 2: Multiple entries for the same Cache Index

How Do you Design a Cache?

• Set of Operations that must be supported
  • read: data <= Mem[Physical Address]
  • write: Mem[Physical Address] <= Data

• Determine the internal register transfers

• Design the Datapath

• Design the Cache Controller
Impact on Cycle Time

**Cache Hit Time:**
directly tied to clock rate
increases with cache size
increases with associativity

**Average Memory Access time =**
Hit Time + Miss Rate x Miss Penalty

Time = IC x CT x (ideal CPI + memory stalls)

Improving Cache Performance: 3 general options

1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.

Memory stall cycles = Read stall cycles + write-stall cycles

Read stall cycles = #reads * read miss rate * read miss penalty
Write stall cycles = #writes * write miss rate * write miss penalty

Memory stall cycles = #access * miss rate * miss penalty
### Examples:

**Q1:** Assume an instruction cache miss rate for gcc of 2% and a data cache miss rate of 4%. If a machine has a CPI of 2 without any memory stalls and the miss penalty is 40 cycles for all misses, determine how much faster a machine would run with a perfect cache that never missed. (It is known that the frequency of all loads and stores in gcc is 36%, unused)

**Answer:**

- Instruction miss cycles = $I \times 2\% \times 40 = 0.80I$
- Data miss cycles = $I \times 4\% \times 40 = 0.56I$
- The CPI with memory stalls is $2 + 1.36 = 3.36$

The performance with the perfect cache is better by 1.68

**Q2:** Suppose we speed up the machine by reducing CPI from 2 to 1.

**Q3:** If we double clock rate,

---

### And yet Another Extreme Example: Fully Associative

- **Fully Associative Cache**
  - Forget about the Cache Index
  - Compare the Cache Tags of all cache entries in parallel
  - Example: Block Size = 32 B blocks, we need N 27-bit comparators

- By definition: Conflict Miss = 0 for a fully associative cache

<table>
<thead>
<tr>
<th>Ex: 0x01</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Tag (27 bits long)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache Tag</th>
<th>Valid Bit</th>
<th>Cache Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>Byte 31</td>
<td>**</td>
</tr>
<tr>
<td>**</td>
<td>Byte 63</td>
<td>**</td>
</tr>
<tr>
<td>:</td>
<td>Byte 33</td>
<td>**</td>
</tr>
<tr>
<td>:</td>
<td>Byte 32</td>
<td>:</td>
</tr>
</tbody>
</table>
A Two-way Set Associative Cache

- N-way set associative: N entries for each Cache Index
  - N direct mapped caches operates in parallel

- Example: Two-way set associative cache
  - Cache Index selects a “set” from the cache
  - The two tags in the set are compared in parallel
  - Data is selected based on the tag result

Disadvantage of Set Associative Cache

- N-way Set Associative Cache versus Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss

- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.
Example:

There are three small caches, each consisting of four one-word blocks. One cache is fully associative, a second is two-way set associative, and the third is direct mapped. Find the number of misses for each cache organization given the following sequence of block addresses: 0, 8, 0, 6, 8

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>8</th>
<th>0</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0000</td>
<td>1000</td>
<td>0000</td>
<td>0110</td>
<td>1000</td>
</tr>
<tr>
<td>F. A.</td>
<td>M</td>
<td>M</td>
<td>H</td>
<td>M</td>
<td>H</td>
</tr>
<tr>
<td>2-Way</td>
<td>M</td>
<td>M</td>
<td>H</td>
<td>M</td>
<td>H</td>
</tr>
<tr>
<td>D. M.</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
</tbody>
</table>

A Summary on Sources of Cache Misses

- **Compulsory (cold start, first reference):** first access to a block
  - “Cold” fact of life: not a whole lot you can do about it

- **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 access by the program
  - Solution: increase cache size

- **Invalidation:** other process (e.g., I/O, processors) updates memory
### Sources of Cache Misses Quiz

<table>
<thead>
<tr>
<th></th>
<th>Direct Mapped</th>
<th>N-way Set Associative</th>
<th>Fully Associative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Size</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compulsory Miss</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflict Miss</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capacity Miss</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Invalidation Miss</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Sources of Cache Misses Answer

<table>
<thead>
<tr>
<th></th>
<th>Direct Mapped</th>
<th>N-way Set Associative</th>
<th>Fully Associative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Size</td>
<td>Big</td>
<td>Medium</td>
<td>Small</td>
</tr>
<tr>
<td>Compulsory Miss</td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>(but who cares!)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflict Miss</td>
<td>High</td>
<td>Medium</td>
<td>Zero</td>
</tr>
<tr>
<td>Capacity Miss</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Invalidation Miss</td>
<td>Same</td>
<td>Same</td>
<td>Same</td>
</tr>
</tbody>
</table>

**Note:**
If you are going to run “billions” of instruction, Compulsory Misses are insignificant.
The Need to Make a Decision!

- Direct Mapped Cache:
  - Each memory location can only mapped to 1 cache location
  - No need to make any decision :-)  
    - Current item replaced the previous item in that cache location

- N-way Set Associative Cache:
  - Each memory location have a choice of N cache locations

- Fully Associative Cache:
  - Each memory location can be placed in ANY cache location

- Cache miss in a N-way Set Associative or Fully Associative Cache:
  - Bring in new block from memory
  - Throw out a cache block to make room for the new block
  - We need to make a decision on which block to throw out!

Cache Block Replacement Policy

- Random Replacement:
  - Hardware randomly selects a cache item and throw it out

- Least Recently Used:
  - Hardware keeps track of the access history
  - Replace the entry that has not been used for the longest time

- Example of a Simple “Pseudo” Least Recently Used Implementation:
  - Assume 64 Fully Associative Entries
  - Hardware replacement pointer points to one cache entry
  - Whenever an access is made to the entry the pointer points to:
    - Move the pointer to the next entry
  - Otherwise: do not move the pointer
Cache Write Policy: Write Through versus Write Back

- Cache read is much easier to handle than cache write:
  - Instruction cache is much easier to design than data cache

- Cache write:
  - How do we keep data in the cache and memory consistent?

- Two options (decision time again :-)
  - Write Back: write to cache only. Write the cache block to memory when that cache block is being replaced on a cache miss.
    - Need a “dirty” bit for each cache block
    - Greatly reduce the memory bandwidth requirement
    - Control can be complex
  - Write Through: write to cache and memory at the same time.
    - What!!! How can this be? Isn’t memory too slow for this?

Write Buffer for Write Through

- A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory

- Write buffer is just a FIFO:
  - Typical number of entries: 4
  - Works fine if: Store frequency (w.r.t. time) \(<\ 1 / DRAM \text{ write cycle}\)

- Memory system designer’s nightmare:
  - Store frequency (w.r.t. time) \(>\ 1 / DRAM \text{ write cycle}\)
  - Write buffer saturation
**Write Buffer Saturation**

- **Store frequency (w.r.t. time) > 1 / DRAM write cycle**
  - If this condition exists for a long period of time (CPU cycle time too quick and/or too many store instructions in a row):
    - Store buffer will overflow no matter how big you make it
    - The CPU Cycle Time <= DRAM Write Cycle Time

- **Solution for write buffer saturation:**
  - Use a write back cache
  - Install a second level (L2) cache:

**Write Allocate versus Not Allocate**

- Assume: a 16-bit write to memory location 0x0 and causes a miss
  - Do we read in the rest of the block (Byte 2, 3, ... 31)?
    - Yes: Write Allocate
    - No: Write Not Allocate

---

[Diagram of Write Buffer Saturation]

[Diagram of Write Allocate versus Not Allocate]
Recap: Improving Cache Performance

1. Reduce the miss rate,
2. **Reduce the miss penalty**, or
3. Reduce the time to hit in the cache.

Memory stall cycles = Read stall cycles + write-stall cycles

Read stall cycles = \#reads * read miss rate * read miss penalty
Write stall cycles = \#writes * write miss rate * write miss penalty

Memory stall cycles = \#access * miss rate * miss penalty

Reducing Memory Transfer Time

**Solution 1**
High BW DRAM

**Solution 2**
Wide Path Between Memory & Cache

**Solution 3**
Memory Interleaving

Examples:
- Page Mode DRAM
- SDRAM
- CDRAM
- RAMbus

Cost
**Cycle Time versus Access Time**

- DRAM (Read/Write) Cycle Time >> DRAM (Read/Write) Access Time
  - 2:1; why?
- DRAM (Read/Write) Cycle Time:
  - How frequent can you initiate an access?
  - Analogy: A little kid can only ask his father for money on Saturday
- DRAM (Read/Write) Access Time:
  - How quickly will you get what you want once you initiate an access?
  - Analogy: As soon as he asks, his father will give him the money
- DRAM Bandwidth Limitation analogy:
  - What happens if he runs out of money on Wednesday?

**Increasing Bandwidth - Interleaving**

Access Pattern without Interleaving:

Access Pattern with 4-way Interleaving:

We can Access Bank 0 again
Main Memory Performance

- Timing model
  - 1 to send address,
  - 6 access time, 1 to send data
  - Cache Block is 4 words

- Simple M.P. = $4 \times (1+6+1) = 32$
- Wide M.P. = $1 + 6 + 1 = 8$
- Interleaved M.P. = $1 + 6 + 4 \times 1 = 11$

Independent Memory Banks

- How many banks?
  - Number banks  Number clocks to access word in bank
  - For sequential accesses, otherwise will return to original bank before it has next word ready

- Increasing DRAM => fewer chips => harder to have banks
  - Growth bits/chip DRAM : 50%-60%/yr
  - Nathan Myrvold M/S: mature software growth (33%/yr for NT) - growth MB/$ of DRAM (25%-30%/yr)
**SPARCstation 20’s Memory System**

- Memory Controller
- Memory Bus (SIMM Bus) 128-bit wide datapath
- Memory Module 0
- Memory Module 1
- Memory Module 2
- Memory Module 3
- Memory Module 4
- Memory Module 5
- Memory Module 6
- Memory Module 7

**SPARCstation 20’s External Cache**

- SuperSPARC Processor
  - Instruction Cache
  - Data Cache
  - Register File

- Processor Module (Mbus Module)

- External Cache
  - 1 MB
  - Direct Mapped
  - Write Back
  - Write Allocate

- *SPARCstation 20’s External Cache:*
  - Size and organization: 1 MB, direct mapped
  - Block size: 128 B
  - Sub-block size: 32 B
  - Write Policy: Write back, write allocate
SPARCstation 20’s Internal Instruction Cache

- SPARCstation 20’s Internal Instruction Cache:
  - Size and organization: 20 KB, 5-way Set Associative
  - Block size: 64 B
  - Sub-block size: 32 B
  - Write Policy: Does not apply

- Note: Sub-block size the same as the External (L2) Cache

SPARCstation 20’s Internal Data Cache

- SPARCstation 20’s Internal Data Cache:
  - Size and organization: 16 KB, 4-way Set Associative
  - Block size: 64 B
  - Sub-block size: 32 B
  - Write Policy: Write through, write not allocate

- Sub-block size the same as the External (L2) Cache
Two Interesting Questions?

- Why did they use N-way set associative cache internally?
  - Answer: A N-way set associative cache is like having N direct mapped caches in parallel. They want each of those N direct mapped cache to be 4 KB. Same as the "virtual page size."

- How many levels of cache does SPARCstation 20 has?
  - Answer: Three levels.
    - (1) Internal I & D caches, (2) External cache and (3) ...

SPARCstation 20's Memory Module

- Supports a wide range of sizes:
  - Smallest 4 MB: 16 2Mb DRAM chips, 8 KB of Page Mode SRAM
  - Biggest: 64 MB: 32 16Mb chips, 16 KB of Page Mode SRAM
DRAM Performance

- A 60 ns (t_{RAC}) DRAM can
  - perform a row access only every 110 ns (t_{RC})
  - perform column access (t_{CAC}) in 15 ns, but time between column accesses is at least 35 ns (t_{PC}).
    - In practice, external address delays and turning around buses make it 40 to 50 ns
- These times do not include the time to drive the addresses off the microprocessor nor the memory controller overhead.
  - Drive parallel DRAMs, external memory controller, bus to turn around, SIMM module, pins...
  - 180 ns to 250 ns latency from processor to memory is good for a “60 ns” (t_{RAC}) DRAM

Summary:

- The Principle of Locality: Temporal Locality vs Spatial Locality
- Four Questions For Any Cache
  - Where to place in the cache
  - How to locate a block in the cache
  - Replacement
  - Write policy: Write through vs Write back
    - Write miss:
- Three Major Categories of Cache Misses:
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Conflict Misses: increase cache size and/or associativity.
    - Nightmare Scenario: ping pong effect!
  - Capacity Misses: increase cache size