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
Outline of Today’s Lecture

° Recap of Memory Hierarchy & Introduction to Cache
° A In-depth Look at the Operation of Cache
° Cache Write and Replacement Policy
° Summary

An Expanded View of the Memory System
The Principle of Locality

What are the principles of Locality?
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 retrieve 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>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Block (line) size</td>
<td>4 - 128 bytes</td>
</tr>
<tr>
<td>Hit time</td>
<td>1 - 4 cycles</td>
</tr>
<tr>
<td>Miss penalty</td>
<td>8 - 32 cycles (and increasing)</td>
</tr>
<tr>
<td></td>
<td>(access time) (6-10 cycles)</td>
</tr>
<tr>
<td></td>
<td>(transfer time) (2 - 22 cycles)</td>
</tr>
<tr>
<td>Miss rate</td>
<td>1% - 20%</td>
</tr>
<tr>
<td>Cache Size</td>
<td>1 KB - 256 KB</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

---

![Cache Diagram](cache.11)

The Simplest Cache: Direct Mapped Cache

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Memory</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>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

- **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

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

```
<table>
<thead>
<tr>
<th></th>
<th>Cache Tag Example: 0x50</th>
<th>Cache Index Ex: 0x01</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid Bit</td>
<td>Stored as part of the cache “state”</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x50</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2^N Bytes Direct Mapped Cache</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Byte 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Byte 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Byte 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Byte 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Byte $2^N - 1$</td>
<td></td>
</tr>
</tbody>
</table>
```

Cache Access Example

Start Up
Access 000 01
Start Up
Access 010 10

Miss Handling:
- Load Data
- Write Tag & Set V
- Access 000 01 (HIT)
- 000
- $M[00001]$
- 010
- $M[01010]$

Access 010 10 (HIT)
- 000
- $M[00001]$
- 010
- $M[01010]$

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 in its own cache tag

° Our previous “extreme” example:
  • 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

### 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 = $2^M$)
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

\begin{tabular}{|c|c|c|c|c|}
\hline
Valid Bit & Cache Tag & Cache Data \\
\hline
\hline
& & Byte 3 & Byte 2 & Byte 1 & Byte 0 & 0 \\
\hline
\end{tabular}

° Cache 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
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.
And yet Another Extreme Example: Fully Associative

- Fully Associative Cache -- push the set associative idea to its limit!
  - Forget about the Cache Index
  - Compare the Cache Tags of all cache entries in parallel
  - Example: Block Size = 2 B blocks, we need N 27-bit comparators

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

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) updates memory
### Source 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>

Categorize as high, medium, low, zero

### 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.
Summary:

° The Principle of Locality:
  • Program 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. Example: cold start misses.
  • Conflict Misses: increase cache size and/or associativity.
    Nightmare Scenario: ping pong effect!
  • Capacity Misses: increase cache size

What about ..

° Replacement

° Writing

NEXT CLASS
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
  - Damn! 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 write cycle

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

- Store frequency (w.r.t. time) -> 1 / DRAM write cycle
  - If this condition exist 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
What is a Sub-block?

- **Sub-block:**
  - A unit within a block that has its own valid bit
  - Example: 1 KB Direct Mapped Cache, 32-B Block, 8-B Sub-block
    - Each cache entry will have: 32/8 = 4 valid bits

- **Write miss:** only the bytes in that sub-block is brought in.

![Cache Diagram]

---

**SPARCstation 20’s Memory System**

- **Memory Controller**
- **Memory Bus (SIMM Bus):** 128-bit wide datapath
- **Memory Modules:** 0 to 7
- **Processor Module (Mbus Module):**
  - **External Cache**
  - **SuperSPARC Processor**
    - Instruction Cache
    - Data Cache
    - Register File
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

- 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:
- 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.”
  - Virtual Page Size: cover in next week’s virtual memory lecture
- 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

**Summary:**

- **The Principle of Locality:**
  - Program 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. Example: cold start misses.
  - Conflict Misses: increase cache size and/or associativity.
    - Nightmare Scenario: ping pong effect!
  - Capacity Misses: increase cache size

- **Write Policy:**
  - Write Through: need a write buffer. Nightmare: WB saturation
  - Write Back: control can be complex