Memory Hierarchy

1977: DRAM faster than microprocessors

Apple ][ (1977)
CPU: 1000 ns
DRAM: 400 ns

Memory Hierarchy: Apple iMac G5

 Managed by compiler
 Reg

 Managed by hardware
 L1 Inst
 L1 Data
 L2
 DRAM
 Disk

Latency, Cycles, Time

Size
1K, 0.6 ns
64K, 3.1 ns
32K, 3.1 ns
19 ns
1.9 ns
32K, 1.9 ns
512K, 6.9 ns
L2, 55 ns
1024K, 12 ms

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

Since 1980, CPU has outpaced DRAM ...
Q. How do architects address this gap?
A. Put smaller, faster “cache” memories between CPU and DRAM.
Create a “memory hierarchy”.

Levels of the Memory Hierarchy

Capacity Access Time Cost

Register

Cache

Main Memory

Blocks

Cache

Blocks

Main Memory

Blocks

Cache

Disks

Files

Upper Level

faster

Registered

Memory

Cache

Main Memory

Blocks

Main Memory

Blocks

Cache

Disks

Files

Tape

Company

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

CPU

Latency

Performance
CPU

60% per yr
2X in 1.5 yrs

DRAM
9% per yr
2X in 10 yrs

Gap grew 50% per year

Year

CS252 S05
The Principle of Locality

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time. (This is kind of like in real life, we all have a lot of friends. But at any given time most of us can only keep in touch with a small group of them.)

- Two Different Types of Locality:
  - **Temporal Locality** (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - **Spatial Locality** (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)

- Last 15 years, HW relied on locality for speed
  - It is a property of programs which is exploited in machine design.

---

Memory Hierarchy: Terminology

- Hit: data appears in some block in the upper level (example: Block X)
  - **Hit Rate**: 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**

- Hit rate: fraction found in that level
  - So high that usually talk about Miss rate
- Miss rate fallacy: as MIPS to CPU performance, miss rate to average memory access time in memory
- Average memory-access time = Hit time + Miss rate x Miss penalty (ns or clocks)
- Miss penalty: time to replace a block from lower level, including time to replace in CPU
  - access time: time to lower level
  - transfer time: time to transfer block

Performance – Cache Memory System

- **T_c**: Effective memory access time in cache memory system
- **T_c**: Cache access time
- **T_m**: Main memory access time
  \[ T_e = T_c + (1 - h) \times T_m \]
  - Example: \( T_c = 0.4\text{ns}, T_m = 1.2\text{ns}, h = 0.85\% \)
  - \( T_e = 0.4 + (1 - 0.85) \times 1.2 = 0.58\text{ns} \)

Cache Measures

- **Hit rate**: fraction found in that level
  - So high that usually talk about Miss rate
- Miss rate fallacy: as MIPS to CPU performance, miss rate to average memory access time in memory
- Average memory-access time = Hit time + Miss rate x Miss penalty (ns or clocks)
- Miss penalty: time to replace a block from lower level, including time to replace in CPU
  - access time: time to lower level
  - transfer time: time to transfer block

4 Questions for Memory Hierarchy

- **Q1**: Where can a block be placed in the upper level? *(Block placement)*
- **Q2**: How is a block found if it is in the upper level? *(Block identification)*
- **Q3**: Which block should be replaced on a miss? *(Block replacement)*
- **Q4**: What happens on a write? *(Write strategy)*
Q1: Where can a block be placed in the upper level?

- Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets

![Cache Diagram]

Q2: How is a block found if it is in the upper level?

- Tag on each block
  - No need to check index or block offset
  - Increasing associativity shrinks index, expands tag

<table>
<thead>
<tr>
<th>Block Address</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Mapped</td>
<td>0</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>Direct Mapped</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2-Way Assoc</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Q3: Which block should be replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)
  - FIFO, MRU, LFU (frequently), MFU

<table>
<thead>
<tr>
<th>Assoc: 2-way</th>
<th>4-way</th>
<th>8-way</th>
<th>Size</th>
<th>LRU</th>
<th>Ran</th>
<th>LRU</th>
<th>Ran</th>
<th>LRU</th>
<th>Ran</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 KB</td>
<td>5.2%</td>
<td>5.7%</td>
<td>4.7%</td>
<td>5.3%</td>
<td>4.4%</td>
<td>5.0%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64 KB</td>
<td>1.9%</td>
<td>2.0%</td>
<td>1.5%</td>
<td>1.7%</td>
<td>1.4%</td>
<td>1.5%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15%</td>
<td>1.17%</td>
<td>1.13%</td>
<td>1.13%</td>
<td>1.12%</td>
<td>1.12%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q4: What happens on a write?

<table>
<thead>
<tr>
<th>Policy</th>
<th>Write-Through</th>
<th>Write-Back</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data written to cache block also written to lower level memory</td>
<td>Write data only to the cache Update lower level when a block falls out of the cache</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Debug</th>
<th>Easy</th>
<th>Hard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do read misses produce writes?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Do repeated writes make it to lower level?</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Additional option -- let writes to an un-cached address allocate a new cache line (“write allocate”).

Q3: After a cache read miss, if there are no empty cache blocks, which block should be removed from the cache?

- The Least Recently Used (LRU) block? Appealing, but hard to implement for high associativity
- A randomly chosen block? Easy to implement, how well does it work?

<table>
<thead>
<tr>
<th>Miss Rate for 2-way Set Associative Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>16 KB</td>
</tr>
<tr>
<td>64 KB</td>
</tr>
<tr>
<td>256 KB</td>
</tr>
</tbody>
</table>

Write Buffers for Write-Through Caches

- Holds data awaiting write-through to lower level memory

Q: Why a write buffer?
- A. So CPU doesn’t stall
Q: Why a buffer, why not just one register?
- A. Bursts of writes are common.
Q: Are Read After Write (RAW) hazards an issue for write buffer?
- A. Yes! Drain buffer before next read, or send read 1st after check write buffers.
5 Basic Cache Optimizations

- Reducing Miss Rate
  1. Larger Block size (compulsory misses)
  2. Larger Cache size (capacity misses)
  3. Higher Associativity (conflict misses)

- Reducing Miss Penalty
  4. Multilevel Caches

- Reducing hit time
  5. Giving Reads Priority over Writes
    - E.g., Read complete before earlier writes in write buffer

Solution: Add a Layer of Indirection

User programs run in a standardized virtual address space

Address Translation hardware managed by the operating system (OS) maps virtual address to physical memory

Hardware supports “modern” OS features:
- Protection, Translation, Sharing

Page tables encode virtual address spaces

A virtual address space is divided into blocks of memory called pages

A machine usually supports pages of a few sizes (MIPS R4000):
- 4 Kbytes
- 16 Kbytes
- 64 Kbytes
- 256 Kbytes
- 1 Mbyte
- 4 Mbytes
- 16 Mbytes

A page table is indexed by a virtual address

A valid page table entry codes physical memory “frame” address for the page

The Limits of Physical Addressing

- All programs share one address space:
  - The physical address space

  Machine language programs must be aware of the machine organization

  No way to prevent a program from accessing any machine resource

3 Advantages of Virtual Memory

- Translation:
  - Program can be given consistent view of memory, even though physical memory is scrambled
  - Makes multithreading reasonable (now used a lot)
  - Only the most important part of program (“Working Set”) must be in physical memory.
  - Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later.

- Protection:
  - Different threads (or processes) protected from each other.
  - Different pages can be given special behavior
  - (Read Only, Invisible to user programs, etc).
  - Kernel data protected from User programs
  - Very important for protection from malicious programs

- Sharing:
  - Can map same physical page to multiple users
    (“Shared memory”)

Details of Page Table

- Page table maps virtual page numbers to physical frames (“PTE = Page Table Entry”)

- Virtual memory = treat memory = cache for disk

OS manages the page table for each ASID
be written to disk.

Schedule pages is still clear.

Place pages on free frame.

Adjunct's role:

Look (TLB)

Virtual Addresses

A table for 4KB pages for a 32-bit address space has 1M entries

Each process needs its own address space!

Two-level Page Tables

32 bit virtual address

8 entries.

Top-level table wired in main memory

Subset of 1024 second-level pages in main memory; rest are on disk or unallocated

TLB Design Concepts

The TLB caches page table entries

Physical and virtual pages must be the same size!

Physical Address

V=0 pages either reside on disk or have not yet been allocated.

OS handles V=0

"Page fault"

MIPS Address Translation: How does it work?

"Virtual Addresses"

"Physical Addresses"

Translation Look-Aside Buffer (TLB)

A small fully-associative cache of mappings from virtual to physical addresses.

TLB also contains protection bits for virtual address

Fast common case: Virtual address is in TLB, process has permission to read or write it.

VM and Disk: Page replacement policy

Set of all pages in memory

Page Table

Dirty bit: page written.

Used bit: set to 1 on any reference

Head pointer

Place pages on free list if used bit is still clear. Schedule pages with dirty bit set to be written to disk.

Tail pointer:

Architect's role: support setting dirty and used bits

Free list

Free Pages

Page address 03275 is where in memory?

Memory broken into page frames

Page Table

Page offset

Program address 03275

Program address 01000

Program address 01000

Program address 01000

Page Table

0

0

1

0

0

1

0

1

1

1

0

0

...
### Problems With Overlapped TLB Access

Overlapped access only works as long as the address bits used to index into the cache do not change as the result of VA translation. Example: suppose everything the same except that the cache is increased to 8 K bytes instead of 4 K:

This bit is changed by VA translation, but is needed for cache lookup.

- Solutions:
  - go to 8K byte page sizes;
  - go to 2 way set associative cache; or

### Summary #1/3: The Cache Design Space

- Several interacting dimensions:
  - Cache Size
  - Block Size
  - Associativity
  - Replacement Policy
  - Write-through vs Write-back
  - Write allocation
- The optimal choice is a compromise:
  - depends on access characteristics
    - workload
    - use (I-cache, D-cache, TLB)
  - depends on technology / cost
- Simplicity often wins

### Summary #2/3: Caches

- 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.
  - Capacity Misses: increase cache size and/or associativity.
  - Conflict Misses: increase cache size and/or associativity. Nightmare Scenario: ping pong effect!
- Write Policy: Write Through vs Write Back
- Today CPU time is a function of (ops, cache misses) vs. just (ops): affects Compilers, Data structures, and Algorithms

### Summary #3/3: TLB, Virtual Memory

- Page tables map virtual address to physical address
- TLBs are important for fast translation
- TLB misses are significant in processor performance
  - funny times, as most systems can't access all of 2nd level cache without TLB misses!
- Caches, TLBs, Virtual Memory all understood by examining how they deal with 4 questions:
  1) Where can block be placed?
  2) How is block found?
  3) What block is replaced on miss?
  4) How are writes handled?
- Today VM protection is more important than memory hierarchy benefits, but computers insecure