CS 61C: Great Ideas in Computer Architecture

Direct-Mapped Caches

Instructor: Justin Hsia
Review of Last Lecture

• Floating point (single and double precision) approximates real numbers
  – Exponent field uses biased notation
  – Special cases: 0, ±∞, NaN, denorm
  – MIPS has special instructions and registers

• Performance measured in latency or bandwidth

• Latency measurement:
  – CPU Time = Instructions × CPI × Clock Cycle Time
  – Affected by different components of the computer
Great Idea #3: Principle of Locality/Memory Hierarchy
Agenda

• Memory Hierarchy Overview
• Administrivia
• Direct-Mapped Caches
• Direct-Mapped Cache Example
• Cache Reads and Writes
Storage in a Computer

• Processor
  – Holds data in register files (~ 100 bytes)
  – Registers accessed on sub-nanosecond timescale

• Memory ("main memory")
  – More capacity than registers (~ Gbytes)
  – Access time ~ 50-100 ns

• Hundreds of clock cycles per memory access?!
Processor-Memory Gap

1989 first Intel CPU with cache on chip
1998 Pentium III has two cache levels on chip

“Moore’s Law”

μProc
55%/year (2X/1.5yr)

Processor-Memory Performance Gap (grows 50%/year)

DRAM
7%/year (2X/10yrs)
Library Analogy

• Writing a report on a specific topic
  – e.g. the works of J.D. Salinger

• While at library, check out books and keep them on desk

• If need more, check them out and bring to desk
  – Don’t return earlier books since might still need them
  – Limited space on desk; which books do we keep?

• You hope this collection of ~10 books on desk enough to write report, despite 10 being only 0.00001% of the books in UC Berkeley libraries
Principle of Locality (1/3)

- **Principle of Locality:** Programs access only a small portion of the full address space at any instant of time
  - **Recall:** Address space holds both code and data
  - Loops and sequential instruction execution mean generally localized code access
  - Stack and Heap try to keep your data together
  - Arrays and structs naturally group data you would access together
Principle of Locality (2/3)

• **Temporal Locality** (locality in time)
  – Go back to the same book on desk multiple times
  – If a memory location is referenced then it will tend to be referenced again soon

• **Spatial Locality** (locality in space)
  – When go to book shelf, grab many books on J.D. Salinger since library stores related books together
  – If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon
Principle of Locality (3/3)

• We exploit the principle of locality in hardware via a memory hierarchy where:
  – Levels closer to processor are faster (and more expensive per bit so smaller)
  – Levels farther from processor are larger (and less expensive per bit so slower)

• **Goal:** Create the illusion of memory being almost as fast as fastest memory and almost as large as biggest memory of the hierarchy
Memory Hierarchy Schematic

Processor

Level 1

Level 2

Level 3

... 

Level n

Smaller, Faster, More expensive

Bigger, Slower, Cheaper
Cache Concept

• Introduce intermediate hierarchy level: memory cache, which holds a copy of a subset of main memory
  – As a pun, often use $ ("cash") to abbreviate cache (e.g. D$ = Data Cache, L1$ = Level 1 Cache)
• Modern processors have separate caches for instructions and data, as well as several levels of caches implemented in different sizes
• Implemented with same IC processing technology as CPU and integrated on-chip – faster but more expensive than main memory
Memory Hierarchy Technologies

• Caches use static RAM (SRAM)
  + Fast (typical access times of 0.5 to 2.5 ns)
  – Low density (6 transistor cells), higher power, expensive
    ($2000 to $4000 per GB in 2011)
• Static: content will last as long as power is on

• Main memory uses dynamic RAM (DRAM)
  + High density (1 transistor cells), lower power, cheaper
    ($20 to $40 per GB in 2011)
  – Slower (typical access times of 50 to 70 ns)
• Dynamic: needs to be “refreshed” regularly (~ every 8 ms)
Memory Transfer in the Hierarchy

Inclusive: data in L1$ ⊂ data in L2$ ⊂ data in MM ⊂ data in SM

Block: Unit of transfer between memory and cache
Managing the Hierarchy

• registers ↔ memory
  – By compiler (or assembly level programmer)

• cache ↔ main memory
  – By the cache controller hardware

• main memory ↔ disks (secondary storage)
  – By the OS (virtual memory, which is a later topic)
  – Virtual to physical address mapping assisted by the hardware (TLB)
  – By the programmer (files)
Typical Memory Hierarchy

- **On-Chip Components**
  - Control
  - Datapath
  - RegFile
  - Instr Cache
  - Data Cache
  - Second Level Cache (SRAM)

- **Secondary Memory**
  - Main Memory (DRAM)
  - Secondary Memory (Disk or Flash)

### Speed (cycles)
- ½’s
- 1’s
- 10’s
- 100’s
- 1,000,000’s

### Size (bytes)
- 100’s
- 10K’s
- M’s
- G’s
- T’s

### Cost/bit
- highest
- lowest

Cost/bit: highest ← 1/2’s, 1’s, 10’s, 100’s, 1,000,000’s → lowest

Cost/bit: highest ← 100’s, 10K’s, M’s, G’s, T’s → lowest
Review So Far

• **Goal:** present the programmer with \( \approx \) as much memory as the *largest* memory at \( \approx \) the speed of the *fastest* memory

• **Approach:** Memory Hierarchy
  
  — Successively higher levels contain “most used” data from lower levels
  
  — Exploits *temporal and spatial locality*
  
  — We will start by studying caches
Agenda

• Memory Hierarchy Overview
• Administrivia
• Direct-Mapped Caches
• Direct-Mapped Cache Example
• Cache Reads and Writes
Administrivia

• Midterm
  – Friday 7/13, 9am-12pm, 245 Li Ka Shing
  – How to study:
    • Studying in groups can help
    • Take old exams for practice
    • Look at lectures, section notes, projects, hw, labs, etc.
  – Will cover material through caches

• Project 1 due Sunday
  – Brandon extra OHs Sat, 3-5pm in lab
Agenda

• Memory Hierarchy Overview
• Administrivia

• Direct-Mapped Caches
• Direct-Mapped Cache Example
• Cache Reads and Writes
Cache Management

• Library analogy: organization is necessary!

• What is the overall organization of blocks we impose on our cache?
  – Where do we put a block of data from memory?
  – How do we know if a block is already in cache?
  – How do we quickly find a block when we need it?
  – When do we replace something in the cache?
General Notes on Caches

- **Recall**: Memory is *byte-addressed*
- We haven’t specified the size of our “blocks,” but will be multiple of word size (32-bits)
  - How do we access individual words or bytes within a block?
- Cache is smaller than memory
  - Can’t fit all blocks at once, so multiple blocks in memory map to the same cache slot (*row*)
  - Need some way of identifying which memory block is currently in the row
Direct-Mapped Caches (1/3)

- Each memory block is mapped to exactly one row in the cache (direct-mapped)
  - Use simple hash function
- Effect of block size:
  - Spatial locality dictates our blocks consist of adjacent bytes, which differ in address by 1
  - Offset field: Lowest bits of memory address can be used to index to specific bytes within a block
  - Block size needs to be a power of two (in bytes)
Direct-Mapped Caches (2/3)

• Effect of cache size: (total stored data)
  – Determines number of blocks cache holds
  – If could hold all of memory, would use remaining bits (minus offset bits) to select appropriate row of cache
    – **Index field:** Apply hash function to remaining bits to determine *which row* the block goes in
      • *(block address) modulo (# of blocks in the cache)*
    – **Tag field:** Leftover upper bits of memory address determine *which portion of memory* the block came from (identifier)
TIO Address Breakdown

• Memory address fields:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T bits</td>
<td>I bits</td>
<td>O bits</td>
</tr>
</tbody>
</table>

• Meaning of the field sizes:
  - O bits $\leftrightarrow 2^O$ bytes/block = $2^{O-2}$ words/block
  - I bits $\leftrightarrow 2^I$ rows in cache = cache size / block size
  - T bits = $A - I - O$, where $A =$ # of address bits
  (A = 32 here)
Direct-Mapped Caches (3/3)

• What’s actually in the cache?
  – Each row contains the actual data block to store
    (B bits = $8 \times 2^O$ bits)
  – In addition, must save Tag field of address as
    identifier ($T$ bits)
  – **Valid bit**: Indicates whether the block in that row
    is valid or not

• Total bits in cache = \# rows $\times (B + T + 1)$
  = $2^l \times (8 \times 2^O + T + 1)$ bits
Cache Example (1/2)

• Cache parameters:
  – Address space of 64B, block size of 1 word, cache size of 4 words

• TIO Breakdown:
  – 1 word = 4 bytes, so \( O = \log_2(4) = 2 \)
  – Cache size / block size = 4, so \( I = \log_2(4) = 2 \)
  – \( A = \log_2(64) = 6 \) bits, so \( T = 6 - 2 - 2 = 2 \)

• Bits in cache = \( 2^2 \times (8 \times 2^2 + 2 + 1) = 140 \) bits
Which blocks map to each row of the cache? (see colors)

On a memory request: (let’s say 001011\textsubscript{two})

1) Take Index field (10)

2) Check if Valid bit is true in that row of cache

3) If valid, then check if Tag matches
Direct-Mapped Cache Internals

- Four words/block, cache size = 1 Ki words
Caching Terminology (1/2)

• When reading memory, 3 things can happen:
  – Cache hit:
    Cache block is valid and contains the proper address, so read the desired word
  – Cache miss:
    Nothing in that row of the cache (not valid), so fetch from memory
  – Cache miss with block replacement:
    Wrong block is in the row, so discard it and fetch desired data from memory
Caching Terminology (2/2)

• How effective is your cache?
  – Want to max cache hits and min cache misses
  – Hit rate (HR): Percentage of memory accesses in a program or set of instructions that result in a cache hit
  – Miss rate (MR): Like hit rate, but for cache misses
    \[ MR = 1 - HR \]

• How fast is your cache?
  – Hit time (HT): Time to access cache (including Tag comparison)
  – Miss penalty (MP): Time to replace a block in the cache from a lower level in the memory hierarchy
Sources of Cache Misses: The 3Cs

• **Compulsory**: (cold start or process migration, 1\textsuperscript{st} reference)
  – First access to block impossible to avoid; Effect is small for long running programs

• **Capacity**:
  – Cache cannot contain all blocks accessed by the program

• **Conflict**: (collision)
  – Multiple memory locations mapped to the same cache location
Get To Know Your Instructor
Agenda

• Memory Hierarchy Overview
• Administrivia
• Direct-Mapped Caches
  • Direct-Mapped Cache Example
• Cache Reads and Writes
**Direct-Mapped Cache Example**

- Consider the sequence of memory address accesses

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

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>3</th>
<th>4</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>Mem(0)</td>
<td>00</td>
<td>Mem(0)</td>
<td>00</td>
<td>Mem(0)</td>
<td>00</td>
<td>Mem(0)</td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>Mem(1)</td>
<td>00</td>
<td>Mem(1)</td>
<td>00</td>
<td>Mem(1)</td>
<td>00</td>
<td>Mem(1)</td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>Mem(2)</td>
<td>00</td>
<td>Mem(2)</td>
<td>00</td>
<td>Mem(2)</td>
<td>00</td>
<td>Mem(2)</td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>Mem(3)</td>
<td>00</td>
<td>Mem(3)</td>
<td>00</td>
<td>Mem(3)</td>
<td>00</td>
<td>Mem(3)</td>
</tr>
</tbody>
</table>

- 8 requests, 6 misses (HR = 0.25, MR = 0.75)
Taking Advantage of Spatial Locality

• Let cache block hold more than one byte

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

0 miss

<table>
<thead>
<tr>
<th>00</th>
<th>Mem(1)</th>
<th>Mem(0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Mem(3)</td>
<td>Mem(2)</td>
</tr>
</tbody>
</table>

1 hit

<table>
<thead>
<tr>
<th>00</th>
<th>Mem(1)</th>
<th>Mem(0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Mem(3)</td>
<td>Mem(2)</td>
</tr>
</tbody>
</table>

2 miss

<table>
<thead>
<tr>
<th>00</th>
<th>Mem(1)</th>
<th>Mem(0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Mem(3)</td>
<td>Mem(2)</td>
</tr>
</tbody>
</table>

3 hit

<table>
<thead>
<tr>
<th>01</th>
<th>Mem(1)</th>
<th>Mem(0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Mem(3)</td>
<td>Mem(2)</td>
</tr>
</tbody>
</table>

4 miss

<table>
<thead>
<tr>
<th>00</th>
<th>Mem(1)</th>
<th>Mem(0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Mem(3)</td>
<td>Mem(2)</td>
</tr>
</tbody>
</table>

3 hit

<table>
<thead>
<tr>
<th>01</th>
<th>Mem(5)</th>
<th>Mem(4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Mem(3)</td>
<td>Mem(2)</td>
</tr>
</tbody>
</table>

4 hit

<table>
<thead>
<tr>
<th>01</th>
<th>Mem(5)</th>
<th>Mem(4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Mem(3)</td>
<td>Mem(2)</td>
</tr>
</tbody>
</table>

15 miss

<table>
<thead>
<tr>
<th>01</th>
<th>Mem(5)</th>
<th>Mem(4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Mem(3)</td>
<td>Mem(2)</td>
</tr>
</tbody>
</table>

• 8 requests, 4 misses \(\text{HR} = 0.5, \text{MR} = 0.5\)
Effect of Block and Cache Sizes on Miss Rate

- 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)
Agenda

• Memory Hierarchy Overview
• Administrivia
• Direct-Mapped Caches
• Direct-Mapped Cache Example
• Cache Reads and Writes
Cache Reads and Writes

• Want to handle reads and writes quickly while maintaining consistency between cache and memory (i.e. both know about all updates)
  – Policies for cache hits and misses are independent

• Here we assume the use of separate instruction and data caches (I$ and D$)
  – Read from both
  – Write only to D$ (assume no self-modifying code)
Handling Cache Hits

• Read hits (I$ and D$)
  – Fastest possible scenario, so want more of these

• Write hits (D$)
  1) Write-Through Policy: Always write data to cache and to memory (through cache)
     • Forces cache and memory to always be consistent
     • Slow! (every memory access is long)
     • Include a Write Buffer that updates memory in parallel with processor

Assume present in all schemes when writing to memory
Handling Cache Hits

• Read hits (I$ and D$)
  – Fastest possible scenario, so want more of these

• Write hits (D$)
  2) Write-Back Policy: Write data only to cache, then update memory when block is removed
     • Allows cache and memory to be inconsistent
     • Multiple writes collected in cache; single write to memory per block
     • Dirty bit: Extra bit per cache row that is set if block was written to (is “dirty”) and needs to be written back
Handling Cache Misses

• Miss penalty grows as block size does

• Read misses (I$ and D$)
  – Stall execution, fetch block from memory, put in cache, send requested word to processor, resume

• Write misses (D$)
  1) Write allocate: Fetch block from memory, put in cache, execute a write hit
     • Works with either write-through or write-back
     • Ensures cache is up-to-date after write miss
Handling Cache Misses

• Miss penalty grows as block size does
• Read misses (I$ and D$)
  – Stall execution, fetch block from memory, put in cache, send requested word to processor, resume
• Write misses (D$)
  2) No-write allocate: Skip cache altogether and write directly to memory
    • Cache is never up-to-date after write miss
    • Ensures memory is always up-to-date
Improvements

• **Nonblocking cache:** allows for multiple loads, so the processor can continue to access cache while cache is handling an earlier miss

• **Early restart:** Processor resumes as soon as requested portion of block is available

• **Critical word first:** Always load requested portion of block first
  
  – Early restart and critical word first can be used together
Summary

• Memory hierarchy exploits principle of locality to deliver lots of memory at fast speeds
• Direct-Mapped Cache: Each block in memory maps to exactly one row in the cache
  – Index to determine which row
  – Offset to determine which byte within block
  – Tag to identify if it’s the block you want

• Cache read and write policies:
  – *Write-back* and *write-through* for hits
  – *Write allocate* and *no-write allocate* for misses