Lecture Topics

- Today: The Memory Hierarchy
  (P&H 5.1-5.11)
- Next: continued

Announcements

- Milestone #7 (due 3/23)
- Milestone #8 (due 4/6)
The Memory Hierarchy

- Processes exhibit locality of reference (temporal and spatial)
- Provide a hierarchy of memory devices (fastest at the top, slowest at the bottom)
- Exploit locality of reference to provide the illusion of fast access to large address space
Four Questions

1) Where can a block be placed in the upper level? (block placement)

2) How is a block found if it is in the upper level? (block identification)

3) Which block should be replaced on a miss? (block replacement)

4) What happens on a write? (write strategy)

Cache

- Most processors have at least one level of cache (often two or three levels)

- Level 1 cache split: instruction cache and data cache (Harvard architecture)
  - allows simultaneous access for instruction fetch and load/store stages in pipelined implementations
  - different usage patterns (can be optimized)
Cache: Block Placement

- Direct mapped – exactly one location where a given block can be placed in the upper level
- Fully associative – block can be placed anywhere in the upper level
- Set associative – block can only be placed in a restricted set of locations (block mapped onto a set, and then the block is placed anywhere in that set)
- Increased associativity decreases miss rate, but with diminishing returns

- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3% miss rate
  - 2-way: 8.6% miss rate
  - 4-way: 8.3% miss rate
  - 8-way: 8.1% miss rate
Example

- 4-way set associative
  - 256 cache slots
  - each slot has 4 separate elements
  - each element has V bit, tag and data block

- Added cost (compared to direct mapped):
  - 4 comparators
  - 4-to-1 MUX
Cache: Block Identification

- Valid bit used to indicate whether an entry contains valid data or not
- Tag bits from address compared to tag bits in entry
- All possible locations searched at once:
  - direct mapped – only one location possible
  - fully associative – all locations possible
  - set associative – subset of locations possible

Separate tag array and data array
- tag array is smaller (and thus faster)
- contents of data array not needed until after hit/miss determined

Direct mapped simplest to implement (but usually has the worst performance)

Fully associative requires comparator and related logic for each cache slot – only practical for small sizes
Cache: Block Replacement

- Hardware must choose victim (block to be evicted)
- Direct mapped: only one choice
- Full associative and set associative: choose one of multiple entries
  - Random
  - LRU (least recently used)
  - FIFO (first in, first out)

FIFO easiest to implement (but usually has worst performance)
Random has reasonable performance
LRU is closest approximation to optimal (look at recent past, instead of looking into future)
Pseudo LRU is good approximation of LRU ("least recently used", "not used recently", "not most recently used")
- LRU easy for 2-way set associative: single bit marks most recently used item
- Pseudo LRU requires N-1 bits to track state

![Tree diagram](image)

**Cache: Write Strategy**

- Store operations modify data objects (about 10% of instructions in a typical program)
- Write back – block in lower level updated when block is evicted from upper level
  - block may be updated multiple times
  - modified bit used to flag modified blocks
Write through – block in upper level and in lower level updated together

When using "write through", there are two alternatives for write misses

- write allocate: the block is copied from RAM into the cache, updated, then copied back to RAM
- write around: the block is just updated in RAM (it is not copied from RAM to cache)

A write buffer can be used to reduce the miss penalty when a modified block is replaced

- modified block copied to write buffer while replacement block is copied from RAM
- contents of write buffer copied to RAM later
Case Study

- Freescale Semiconductor’s MPC7450 processor (PowerPC implementation)
- Split level 1 instruction and data caches
- Cache characteristics
  - 8-way set associative
  - 128 sets
  - 8 words (32 bytes) per block
- Pseudo LRU replacement
- 7 bits per set

<table>
<thead>
<tr>
<th>B0</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>L0</th>
<th>B4</th>
<th>B5</th>
<th>B6</th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
<th>L4</th>
<th>L5</th>
<th>L6</th>
<th>L7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

![Tree Diagram](image)
Cache Control

- Example cache characteristics
  - Direct-mapped, write-back, write allocate
  - Block size: 4 words (16 bytes)
  - Cache size: 16 KB (1024 blocks)
  - 32-bit byte addresses
  - Valid bit and modified bit per block
  - Blocking cache (CPU waits until access is complete)
**Interface Signals**

- **CPU**
  - Read/Write
  - Valid
  - Address \(^{32}\)
  - Write Data \(^{32}\)
  - Read Data \(^{32}\)
  - Ready

- **Cache**
  - Read/Write
  - Valid
  - Address \(^{32}\)
  - Write Data \(^{128}\)
  - Read Data \(^{28}\)
  - Ready

- **RAM**

---

**Finite State Machines**

Set of states, transition on each clock edge

- State values are binary encoded
- Current state stored in a register
- Next state = \(f_n\) (current state, current inputs)
Multilevel Caches

- Level 1 cache attached to CPU (small, fast)
- Level 2 cache services Level 1 misses (larger than Level 1, but still faster than RAM)
- RAM services L 2 cache misses
- Recent systems have Level 3 cache
Multilevel Caches

- **L-1 cache**
  - Focus on minimal hit time

- **L-2 cache**
  - Focus on low miss rate to avoid RAM access
  - Hit time has less overall impact

- **Results**
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size
Classifying Misses: the 3 C’s

- **Compulsory:** the first access to a block is not in the cache, so the block must be brought into the cache
- Also called *cold start misses* or *first reference misses*
- Misses even if the cache was infinite in size

Classifying Misses: the 3 C’s

- **Capacity:** if the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved
- Misses in fully associative cache
Classifying Misses: the 3 C’s

- **Conflict:** if block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory & capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set

- Also called *collision misses* or *interference misses*

- Misses in N-way set associative cache