Big Idea: Memory Hierarchy

- **Principle of Locality**: Programs access small portion of address space at any instant of time.
- What program structures lead to locality in instruction accesses?

Library Analogy

- Writing a report based on books on reserve
  - E.g., works of J.D. Salinger
- Go to library to get reserved book and place on desk in library
- If need more, check them out and keep on desk
  - But don’t return earlier books since might need them
- You hope this collection of ~10 books on desk enough to write report, despite 10 being only 0.00001% of books in UC Berkeley libraries

Cache Philosophy

- Programmer-invisible hardware mechanism to give illusion of speed of fastest memory with size of largest memory
  - Works fine even if programmer has no idea what a cache is
  - However, performance-oriented programmers today sometimes “reverse engineer” cache design to design data structures to match cache
  - We’ll do that in Project 3
**Memory Access without Cache**

- Load word instruction: `lw $t0, 0($t1)`
- `$t1` contains `1022_{ten}`, `Memory[1022] = 99`

1. Processor issues address `1022_{ten}` to Memory
2. Memory reads word at address `1022_{ten}(99)`
3. Memory sends 99 to Processor
4. Processor loads 99 into register `$t1$

**Memory Access with Cache**

- Load word instruction: `lw $t0, 0($t1)`
- `$t1` contains `1022_{ten}`, `Memory[1022] = 99`
- With cache (similar to a hash)
  1. Processor issues address `1022_{ten}` to Cache
  2. Cache checks to see if has copy of data at address `1022_{ten}`
     2a. If finds a match (Hit): cache reads 99, sends to processor
     2b. No match (Miss): cache sends address `1022_{ten}` to Memory
        i. Memory reads 99 at address `1022_{ten}`
        ii. Memory sends 99 to Cache
        iii. Cache replaces word with new 99
        iv. Cache sends 99 to processor
  3. Processor loads 99 into register `$t1$

**Cache “Tags”**

- Need way to tell if have copy of location in memory so that can decide on hit or miss
- On cache miss, put memory address of block in “tag address” of cache block
  – `1022` placed in tag next to data from memory (99)

**Anatomy of a 16 Byte Cache, 4 Byte Block**

- Operations:
  1. Cache Hit
  2. Cache Miss
  3. Refill cache from memory
- Cache needs Address Tags to decide if Processor Address is a Cache Hit or Cache Miss
  – Compares all 4 tags

**Cache Requirements**

- Suppose processor now requests location 511, which contains 11?
- Doesn’t match any cache block, so must “evict” one resident block to make room
  – Which block to evict?
- Replace “victim” with new memory block at address 511

**Block Must be Aligned in Memory**

- Word blocks are aligned, so binary address of all words in cache always ends in `0_{two}`
- How to take advantage of this to save hardware and energy?
- Don’t need to compare last 2 bits of 32-bit byte address (comparator can be narrower)
  => Don’t need to store last 2 bits of 32-bit byte address in Cache Tag (Tag can be narrower)
Anatomy of a 32B Cache, 8B Block

- Blocks must be aligned in pairs, otherwise could get same word twice in cache
- Tags only have even-numbered words
- Last 3 bits of address always 000 <sub>two</sub>
- Tags, comparators can be narrower
- Can get hit for either word in block

Big Idea: Locality

- **Temporal Locality** (locality in time)
  - Go back to same book on desktop 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, pick up multiple 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

- **Principle of Locality**: Programs access small portion of address space at any instant of time
- What program structures lead to temporal and spatial locality in instruction accesses?
- In data accesses?

Common Cache Optimizations

- Reduce tag overhead by having larger blocks
  - E.g., 2 words, 4 words, 8 words
- Separate caches for instructions and data
  - Double bandwidth, don’t interfere with each other
- Bigger caches (but access time could get bigger than one clock cycle if too big)
- Divide cache into multiple sets, only search inside one set => saves comparators, energy
  - If as many sets as blocks, then only 1 comparator (aka Direct-Mapped Cache)
  - But may increase Miss Rate

Hardware Cost of Cache

- Need to compare every tag to the Processor address
- Comparators are expensive
- Optimization: 2 sets => ½ comparators
- 1 Address bit selects which set

Processor Address Fields used by Cache Controller

- **Block Offset**: Byte address within block
- **Set Index**: Selects which set
- **Tag**: Remaining portion of processor address

<table>
<thead>
<tr>
<th>Processor Address (32-bits total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
</tr>
</tbody>
</table>

- Size of Index = \log_2 (number of sets)
- Size of Tag = Address size – Size of Index – \log_2 (number of bytes/block)
What is limit to number of sets?

- Can save more comparators if have more than 2 sets
- Limit: As Many Sets as Cache Blocks – only needs one comparator!
- Called “Direct-Mapped” Design

One More Detail: Valid Bit

- When start a new program, cache does not have valid information for this program
- Need an indicator whether this tag entry is valid for this program
- Add a “valid bit” to the cache tag entry
  - 0 => cache miss, even if by chance, address = tag
  - 1 => cache hit, if processor address = tag

Direct-Mapped Cache Example

- One word blocks, cache size = 1K words (or 4KB)

Adminstrivia

- Lab #5: MIPS Assembly
- HW #4 (of six), due Sunday
- Project 2a: MIPS Emulator, due Sunday
- Midterm, two weeks from yesterday

Agenda

- Memory Hierarchy Analogy
- Memory Hierarchy Overview
- Adminstrivia
- Caches
  - Fully Associative, N-Way Set Associative, Direct Mapped Caches
  - Cache Performance
  - Multilevel Caches

Cache Terms

- Hit rate: fraction of access that hit in the cache
- Miss rate: 1 – Hit rate
- Miss penalty: time to replace a block from lower level in memory hierarchy to cache
- Hit time: time to access cache memory (including tag comparison)
- Abbreviation: “S” = cache (A Berkeley innovation!)
### Mapping a 6-bit Memory Address

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Block Within 8 Blocks</th>
<th>Mem Block Within 8 Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

- In example, block size is 4 bytes/1 word (it could be multi-word)
- Memory and cache blocks are the same size, unit of transfer between memory and cache
- # Memory blocks >> # Cache blocks
  - 16 Memory blocks/16 words/8 bytes/6 bits to address all bytes
  - 4 Cache blocks, 4 bytes (1 word) per block
  - 4 Memory blocks map to each cache block
- Byte within block: low order two bits, ignore (nothing smaller than a block)
- Memory block to cache block, aka index: middle two bits
- Which memory block is in a given cache block, aka tag: top two bits

### Caching: A Simple First Example

#### Main Memory

- One word blocks. Two low-order bits define the byte in the block (32b words).
- Q: Where in the cache is the memory block?
- Use next 2 low-order memory address bits — the index — to determine which cache block (i.e., modulo the number of blocks in the cache)

#### Cache

- Q: Is the memory block in cache?
- Compare the cache tag to the high order 2 memory address bits to tell if the memory block is in the cache (provided valid bit is set)

### Cache Names for Each Organization

- **“Fully Associative”**: Block can go anywhere
  - First design in lecture
  - Note: No Index field, but 1 comparator/block
- **“Direct Mapped”**: Block goes one place
  - Note: Only 1 comparator
  - Number of sets = number blocks
- **“N-way Set Associative”**: N places for a block
  - Number of sets = number of blocks / N
  - Fully Associative: N = number of blocks
  - Direct Mapped: N = 1

### Range of Set-Associative Caches

- For a fixed-size cache, each increase by a factor of 2 in associativity doubles the number of blocks per set (i.e., the number of “ways”) and halves the number of sets—
  - decreases the size of the index by 1 bit and increases the size of the tag by 1 bit

Note: IBM persists in calling sets “ways” and ways “sets”. They’re wrong.
Review so far

- Principle of Locality for Libraries /Computer Memory
- Hierarchy of Memories (speed/size/cost per bit) to Exploit Locality
- Cache – copy of data lower level in memory hierarchy
- Direct Mapped to find block in cache using Tag field and Valid bit for Hit
- Larger caches reduce Miss rate via Temporal and Spatial Locality, but can increase Hit time
- Multilevel caches help Miss penalty
- AMAT helps balance Hit time, Miss rate, Miss penalty