Set Associative Mapping

- Generalizes all Cache Mapping Schemes
  - Assume cache contains \( N \) blocks
  - 1-way SA cache: Direct Mapping
  - \( M \)-way SA cache: if \( M = N \), then fully assoc.

- Advantage
  - Decreases miss rate (more places to find B)

- Disadvantage
  - Increases hit time (more places to look for B)
  - More complicated hardware
Three mapping schemes

- **Direct-mapped**
- **Fully associative**
- **Set associative**

**Set-Associative Cache**

- **mapping**: (memory address) modulo (# of sets in cache)
- **Example**: two-way set associative cache with 4 1-word block

**Direct-mapped**

<table>
<thead>
<tr>
<th>address</th>
<th>block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

2-way SA:

<table>
<thead>
<tr>
<th>address</th>
<th>Set</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0 or 1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0 or 1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2 or 3</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0 or 1</td>
</tr>
</tbody>
</table>

Fully SA:

<table>
<thead>
<tr>
<th>address</th>
<th>Set</th>
<th>block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0-3</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0-3</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0-3</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0-3</td>
</tr>
</tbody>
</table>

- Which block to replace for set-associative cache?
  - **Random**
  - The least recently used (LRU) block within a set.
  - First-in-First-Out (FIFO)
Replacement

• Choice of entry to replace on a miss
  – Least recently used (LRU)
    • Complex and costly hardware for high associativity
  – Random
    • Close to LRU, easier to implement
• Virtual memory
  – LRU approximation with hardware support

Example

• cache with 4 1-word block, Reference: 0 8 0 6 8

• Direct-mapped:

<table>
<thead>
<tr>
<th>address</th>
<th>block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

Ref H/M?

<table>
<thead>
<tr>
<th>0</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>M</td>
</tr>
<tr>
<td>0</td>
<td>M</td>
</tr>
<tr>
<td>6</td>
<td>M</td>
</tr>
<tr>
<td>8</td>
<td>M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>M[8]</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>M[6]</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

2-way SA:

<table>
<thead>
<tr>
<th>address</th>
<th>Set</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0 or 1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0 or 1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0 or 1</td>
</tr>
</tbody>
</table>

Ref H/M?

<table>
<thead>
<tr>
<th>0</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>M</td>
</tr>
<tr>
<td>0</td>
<td>M</td>
</tr>
<tr>
<td>6</td>
<td>M</td>
</tr>
<tr>
<td>8</td>
<td>M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set</th>
<th>block</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>M[0]</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>M[8]</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Example (continued)

• Reference: 0 8 0 6 8

• Fully associative

<table>
<thead>
<tr>
<th>address</th>
<th>Set</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0–3</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0–3</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0–3</td>
</tr>
</tbody>
</table>

2-way SA:

<table>
<thead>
<tr>
<th>address</th>
<th>Set</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0 or 1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0 or 1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0 or 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ref</th>
<th>H/M?</th>
<th>Block</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>M</td>
<td>0</td>
<td>M[0]</td>
</tr>
<tr>
<td>8</td>
<td>M</td>
<td>1</td>
<td>M[8]</td>
</tr>
<tr>
<td>0</td>
<td>H</td>
<td>2</td>
<td>M[6]</td>
</tr>
<tr>
<td>6</td>
<td>M</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ref</th>
<th>H/M?</th>
<th>Set</th>
<th>block</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>M</td>
<td>0</td>
<td>M[0]</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>M</td>
<td>1</td>
<td>M[8]</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>H</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>M</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Locating a block in SA cache

Cache Address Components:

• Index: i Selects the set S_i

• Tag: Used to find the memory block you’re looking for, by comparing with the n blocks in the selected set S

• Block Offset: Address of desired data within the block
2-way Cache Hardware

Cache Address

N-way cache:
N muxes, and gates, comparators

4-Way Set Associative Cache Organization
How Much Associativity

- 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%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%
### Finding a Block

<table>
<thead>
<tr>
<th>Associativity</th>
<th>Location method</th>
<th>Tag comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct mapped</td>
<td>Index</td>
<td>1</td>
</tr>
<tr>
<td>n-way set associative</td>
<td>Set index, then search entries within the set</td>
<td>n</td>
</tr>
<tr>
<td>Fully associative</td>
<td>Search all entries</td>
<td>#entries</td>
</tr>
<tr>
<td></td>
<td>Full lookup table</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Hardware caches**
  - Reduce comparisons to reduce cost
- **Virtual memory**
  - Full table lookup makes full associativity feasible
  - Benefit in reduced miss rate

### Partition of memory address in SA cache

- Suppose we have a 32KB 2-way SA cache with 2 word blocks.
- Determine the size of the tag, index and offset fields if we’re using a 32-bit architecture.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Set</th>
<th>Index</th>
<th>offset</th>
</tr>
</thead>
</table>

- # of bits for offset: specify the correct byte within a block, depends on the size of block
- # of bits for set index: specify the correct cache set, depends on the number of sets
- # of bits for tag: remaining bits
2-Way SA Cache Example (2/4)

• Set Index
  – need to specify correct row in cache
  – cache contains 32 KB = $2^{15}$ bytes
  – each block contains $2^3$ bytes (2 words)
  – # rows/cache = $\frac{2^{15} \text{ bytes/cach}}{2^3 \text{ bytes/block} \times 2 \text{ blocks/row}} = 2^{11}$ rows/cache
  – need 11 bits to specify this many rows

2-Way SACache Example (3/4)

• Block offset: 3 bits for 2-word block
• Tag
  – used remaining bits as tag
  – tag length = mem addr length
    - offset
    - index
    = 32 - 3 - 11 bits
    = 18 bits
  – so tag is leftmost 18 bits of memory address
2-way SA Cache Example (4/4)

- **Total size**
  
  \[ \text{Total size} = \text{number of rows} \times \text{size of each row} \]
  
  \[ = \text{number of rows} \times 2 \times (64 \text{ bits} + 18 \text{ bits} + 1 \text{ bits}) \]
  
  \[ = 2^{11} \text{ rows} \times 2 \times 83 \text{ bits} = 339968 \text{ bits} = 42,496 \text{B} \]
  
  > 32 KB

- Data (2 word)
- Tag
- Valid bit

**Hits vs. Misses**

- **Read hits:**
  - this is what we want!
- **Read misses:**
  - stall the CPU, fetch block from memory, deliver to cache, restart
  - Put data read from memory into the cache Data field
  - Write upper bits of address into the cache Tag field
  - Turn the Valid Bit ON
- **How to partition memory address:**
  - # of bits for offset: specify the correct byte within a block
  - # of bits for index: specify the correct cache block
  - # of bits for tag: remaining bits
  - \text{lw} $s1, 0000\ 1001\ 1100\ 0110\ 1010\ 1111\ 0101\ 1000$

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
</table>

54

55
Block Write Strategies

1) Write-Through:
   - Write the data to (a) cache and (b) block in main memory
   - **Advantage:** Easier implementation, only need write buffer
   - **Disadvantage:** lower speed

2) Write-Back:
   - Write the data only to cache block. Write to memory only when block is replaced
   - **Advantage:** Writes are limited only by cache write rate, higher speed, Multi-word writes supported, since only one (efficient) write to main memory is made per block
   - **Disadvantage:** more difficult to implement

---

Write-Through

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update memory
- But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = 1 + 0.1×100 = 11
- **Solution:** write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - **Only stalls on write if write buffer is already full**
**Write-Back**

- Alternative: On data-write hit, just update the block in cache
  - Keep track of whether each block is dirty
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first

**Write Allocation**

- What should happen on a write miss?
- Alternatives for write-through
  - Allocate on miss: fetch the block
  - Non-Allocate: don’t fetch the block
    - Since programs often write a whole block before reading it (e.g., initialization)
- For write-back
  - Usually fetch the block
Memory System support caches

- Assume 1 clock cycle to send the address
  - 15 cc for each DRAM access initiated
  - 1 cc to send a word of data
- Make reading multiple words easier by using banks of memory

5.3 Measuring and Improving Cache Performance

- Simplified model:
  \[
  \text{execution time} = (\text{execution cycles} + \text{memory stall cycles}) \times \text{cycle time}
  \]
  \[
  \text{stall cycles} = \# \text{ of instructions} \times \text{miss ratio} \times \text{miss penalty}
  \]
- AMAT = hit time + miss rate x miss penalty
- Two ways of improving performance:
  - decreasing the miss rate
    - Larger block size, larger cache, higher associativity
    - use split caches
  - decreasing the miss penalty
    Multi-level cache
Cost of a Cache Miss

Example: Instruction Cache Miss (couldn’t find instr. in cache)

- Observe: Bigger blocks take more time to fetch
- Oops: Bigger blocks exploit spatial locality property
- Which one? Solution: Make cache as big as possible
  This decreases effect of blocksize

---

Cache Design Trade-offs

<table>
<thead>
<tr>
<th>Design change</th>
<th>Effect on miss rate</th>
<th>Negative performance effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increase cache size</td>
<td>Decrease capacity misses</td>
<td>May increase access time</td>
</tr>
<tr>
<td>Increase associativity</td>
<td>Decrease conflict misses</td>
<td>May increase access time</td>
</tr>
<tr>
<td>Increase block size</td>
<td>Decrease compulsory misses</td>
<td>Increases miss penalty. For very large block size, may increase miss rate due to pollution.</td>
</tr>
</tbody>
</table>
Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- Miss cycles per instruction
  - I-cache: 0.02 × 100 = 2
  - D-cache: 0.36 × 0.04 × 100 = 1.44
- Actual CPI = 2 + 2 + 1.44 = 5.44
  - Ideal CPU is 5.44/2 = 2.72 times faster

Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2ns
  - 2 cycles per instruction
Decreasing miss penalty with multilevel caches

• Add a second level cache:
  – often primary cache is on the same chip as the processor
  – use SRAMs to add another cache above primary memory (DRAM)
  – miss penalty goes down if data is in 2nd level cache

• Example:
  – CPI of 1.0 on a 500Mhz machine with a 5% miss rate, 200ns DRAM access
  – Adding 2nd level cache with 20ns access time decreases miss rate to 2%

• Using multilevel caches:
  – try and optimize the hit time on the 1st level cache
  – try and optimize the miss rate on the 2nd level cache

4 Questions for Memory Hierarchy

• Q1: Where can a block be placed?

<table>
<thead>
<tr>
<th>Scheme</th>
<th># of sets</th>
<th>Blocks per set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct mapped</td>
<td># of blocks</td>
<td>1</td>
</tr>
<tr>
<td>Set Associative</td>
<td># of blocks/Associativity</td>
<td>2-8</td>
</tr>
<tr>
<td>Fully Associative</td>
<td>1</td>
<td># of blocks</td>
</tr>
</tbody>
</table>

• Q2: How is a block is found?

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Location method</th>
<th>Comparisons required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct mapped</td>
<td>index</td>
<td>1</td>
</tr>
<tr>
<td>Set Associative</td>
<td>Index set, search among elements</td>
<td>Degree of associativity</td>
</tr>
<tr>
<td>Fully Associative</td>
<td>Search all cache entries</td>
<td>Size of the cache</td>
</tr>
<tr>
<td></td>
<td>Separate lookup table</td>
<td>0</td>
</tr>
</tbody>
</table>
4 Questions for Memory Hierarchy

- Q3: Which block should be replaced on a cache miss
  - Scheme | Which block to replace | replace scheme
  - Direct mapped | Only one | none
  - Set Associative | # of blocks in a set | LRU, random
  - Fully Associative | All blocks | LRU, random

- Q4: What happens on a write?
  - Write back
    - Allocate
  - Write through
    - Allocate, nonallocate

Sources of Misses (The 3 Cs)

- Compulsory misses (aka cold start misses)
  - First access to a block
- Capacity misses
  - Due to finite cache size
  - A replaced block is later accessed again
- Conflict misses (aka collision misses)
  - In a non-fully associative cache
  - Due to competition for entries in a set
  - Would not occur in a fully associative cache of the same total size
**Hits vs. Misses**

- **Read hits:**
  - this is what we want!
- **Read misses:**
  - stall the CPU, fetch block from memory, deliver to cache, restart
    - Put data read from memory into the cache Data field
    - Write upper bits of address into the cache Tag field
    - Turn the **Valid Bit ON**
- **Write hits:**
  - can replace data in cache and memory (write-through)
  - write the data only into the cache (write-back the memory later)
- **Write misses:**
  - read the entire block into the cache, then write the word

---

**The Memory Hierarchy: Summary**

**The BIG Picture**

- Common principles apply at all levels of the memory hierarchy
  - Based on notions of caching
- At each level in the hierarchy
  - Block placement
  - Finding a block
  - Replacement on a miss
  - Write policy
### Write Policy

- **Write-through**
  - Update both upper and lower levels
  - Simplifies replacement, but may require write buffer

- **Write-back**
  - Update upper level only
  - Update lower level when block is replaced
  - Need to keep more state

- **Virtual memory**
  - Only write-back is feasible, given disk write latency