CS2100 Computer Organisation

Cache II
Road Map: Part II

- Performance
- Assembly Language
- Processor: Datapath
- Processor: Control
- Pipelining
- Cache

Cache Part II
- Type of Cache Misses
- Direct-Mapped Cache
- Set Associative Cache
- Block Replacement Policy
- Cache Framework
Cache Misses: Classifications

**Compulsory / Cold Miss**

- First time a **memory block** is accessed
- Cold fact of life: Not much can be done
- **Solution**: Increase cache block size

**Conflict Miss**

- Two or more distinct memory blocks map to the same cache block
- Big problem in direct-mapped caches
- **Solution 1**: Increase cache size
  - Inherent restriction on cache size due to SRAM technology
- **Solution 2**: **Set-Associative caches** (coming next ..)

**Capacity Miss**

- Due to limited cache size
Block Size Tradeoff (1/2)

**Average Access Time**

\[ = \text{Hit rate} \times \text{Hit Time} + (1 - \text{Hit rate}) \times \text{Miss penalty} \]

- Larger block size:
  - Takes advantage of spatial locality
  - Larger miss penalty: Takes longer time to fill up the block
  - If block size is too big relative to cache size
    - Too few cache block
    - Miss rate will go up

![Graphs showing the relationship between block size, miss penalty, miss rate, and average access time.](image)

- Exploits spatial locality
- Fewer blocks: compromises temporal locality
- Average access time decreases as block size increases
Block Size Tradeoff (2/2)
Another way to organize the cache blocks

SET ASSOCIATIVE CACHE
Set-Associative (SA) Cache Analogy

Many book titles start with "T"

⇒ Too many conflicts!

Hmm… how about we give more slots per alphabet, 2 books start with "A", 2 books start with "B", …. etc?
Set Associative (SA) Cache

- **N-way Set Associative Cache**
  - A memory block can be placed in a fixed number of locations ($N > 1$) in the cache

- **Key Idea:**
  - Cache consists of a number of sets:
    - Each set contains $N$ cache blocks
  - Each memory block maps to a unique cache set
  - Within the set, a memory block can be placed in any element of the set
An example of 2-way set associative cache
- Each set has two cache blocks
- A memory block maps to an unique set
  - In the set, the memory block can be placed in either of the cache blocks
  - Need to search both to look for the memory block
Set-Associative Cache: Mapping

Memory Address

Cache Block size = $2^N$ bytes

Cache Set Index

$\text{Cache Set Index} = (\text{BlockNumber}) \mod (\text{NumberOfCacheSets})$

Cache Block size = $2^N$ bytes

Number of cache sets = $2^M$

Offset = $N$ bits

Set Index = $M$ bits

Tag = $32 - (N + M)$ bits

Observation:
It is essentially unchanged from the direct-mapping formula
SA Cache Mapping: Example

Memory 4GB

1 Block = 4 bytes

Cache 4 KB

Memory Address

31 \[\rightarrow\] Block Number \[\rightarrow\] Offset

N N-1 0

Offset, N = 2 bits

Block Number = 32 – 2 = 30 bits

Check: Number of Blocks = 2^{30}

31 \[\rightarrow\] Tag \[\rightarrow\] Set Index \[\rightarrow\] Offset

N+M-1 N N-1 0

Number of Cache Blocks

= 4KB / 4bytes = 1024 = 2^{10}

4-way associative, number of sets

= 1024 / 4 = 256 = 2^8

Set Index, M = 8 bits

Cache Tag = 32 – 8 – 2 = 22 bits
Set Associative Cache Circuitry

4-way 4-KB cache: 1-word (4-byte) blocks

Note the simultaneous "search" on all elements of a set
Example:
Given this memory access sequence: 0 4 0 4 0 4 0 4

Result:
Cold Miss = 2 (First two accesses)
Conflict Miss = 6 (The rest of the accesses)
Example:
Given this memory access sequence: 0 4 0 4 0 4 0 4

Result:
Cold Miss = 2 (First two accesses)
Conflict Miss = None
Rule of Thumb:
A direct-mapped cache of size $N$ has about the same miss rate as a 2-way set associative cache of size $N / 2$
SA Cache Example: Setup

Given:

- Memory access sequence: 4, 0, 8, 36, 0
- 2-way set-associative cache with a total of four 8-byte blocks → total of 2 sets
- Indicate hit/miss for each access

<table>
<thead>
<tr>
<th>Offset, $N = 3$ bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Number $= 32 - 3 = 29$ bits</td>
</tr>
</tbody>
</table>

- 2-way associative, number of sets $= 2 = 2^1$
- Set Index, $M = 1$ bits

Cache Tag $= 32 - 3 - 1 = 28$ bits
Example: **LOAD #1 - 1**

- Load from 4 ➔ 0000000000000000000000000000000000000000000000000000000000000000

**Check:** Both blocks in **Set 0** are invalid [**Cold Miss**]
Example: LOAD #1 - 2

Load from 4 ➔ 000000000000000000000000000100

Result: Load from memory and place in **Set 0-Block 0**

<table>
<thead>
<tr>
<th>Set Idx</th>
<th>Valid</th>
<th>Tag</th>
<th>W0</th>
<th>W1</th>
<th>Valid</th>
<th>Tag</th>
<th>W0</th>
<th>W1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>M[0]</td>
<td>M[4]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: **LOAD #2**

- Load from 0 ➔ 0000000000000000000000000000000000000000000

**Result:**

[Valid and Tag match] in **Set 0-Block 0** [**Spatial Locality**]

<table>
<thead>
<tr>
<th>Set Idx</th>
<th>Block 0</th>
<th>Block 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Valid</td>
<td>Tag</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Example: **LOAD #3 - 1**

Load from 8 ➔

<table>
<thead>
<tr>
<th>Set Idx</th>
<th>Block 0</th>
<th>Block 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Valid</td>
<td>Tag</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Check:** Both blocks in **Set 1** are invalid [Cold Miss]
**Example: LOAD #3 - 2**

- Load from 8 ➔

```
00000000000000000000000000000000000000000000000000000000000000000
1 000
```

**Result:** Load from memory and place in **Set 1-Block 0**

<table>
<thead>
<tr>
<th>Set Idx</th>
<th>Block 0</th>
<th>Block 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Valid</td>
<td>Tag</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Example: **LOAD #4 - 1**

- Load from **36**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000000000000000000000000010</td>
<td>0</td>
<td>100</td>
</tr>
</tbody>
</table>

**Check:** [Valid but tag mismatch] **Set 0-Block 0**  
[Invalid] **Set 0-Block1**  
[**Cold Miss**]

<table>
<thead>
<tr>
<th>Set Idx</th>
<th>Block 0</th>
<th></th>
<th>Block 1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Valid</td>
<td>Tag</td>
<td>W0</td>
<td>W1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>M[0]</td>
<td>M[4]</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>M[8]</td>
<td>M[12]</td>
</tr>
</tbody>
</table>
Example: **LOAD #4 - 2**

- **Load from 36** ➔ **0000000000000000000000000010**

**Result:** Load from memory and place in **Set 0-Block 1**

<table>
<thead>
<tr>
<th>Set Idx</th>
<th>Block 0</th>
<th>Block 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Valid</td>
<td>Tag</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Example: **LOAD #5**

- **Load from 0 ➡️**

<table>
<thead>
<tr>
<th>Set Idx</th>
<th>Valid</th>
<th>Tag</th>
<th>W0</th>
<th>W1</th>
<th>Valid</th>
<th>Tag</th>
<th>W0</th>
<th>W1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>M[8]</td>
<td>M[12]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Check:** [Valid and tag match] Set 0-Block 0  
[Valid but tag mismatch] Set 0-Block1  

[Temporal Locality]
Who should I kick out next…..?

BLOCK REPLACEMENT POLICY
Block Replacement Policy (1/3)

- Set Associative or Fully Associative Cache:
  - Can choose where to place a memory block
  - Potentially replacing another cache block if full
  - Need block replacement policy

- Least Recently Used (LRU)
  - **How:** For cache hit, record the cache block that was accessed
    - When replacing a block, choose one which has not been accessed for the longest time
  - **Why:** Temporal locality
Block Replacement Policy (2/3)

- Least Recently Used policy in action:
  - 4-way Cache
  - Memory accesses: 0 4 8 12 4 16 12 0 4

![Diagram showing LRU cache replacements]
Block Replacement Policy (3/3)

- **Drawback for LRU**
  - Hard to keep track if there are many choices

- **Other replacement policies:**
  - First in first out (FIFO)
    - Second chance variant
  - Random replacement (RR)
  - Least frequently used (LFU)
Cache Organizations: Summary

One-way set associative (direct mapped)

Two-way set associative

Four-way set associative

Eight-way set associative (fully associative)
Cache Framework (1/2)

**Block Placement:** Where can a block be placed in cache?

- **Direct Mapped:** Only one block defined by index
- **N-way Set-Associative:** Any one of the N blocks within the set defined by index
- **Fully Associative:** Any cache block

**Block Identification:** How is a block found if it is in the cache?

- **Direct Mapped:** Tag match with only one block
- **N-way Set Associative:** Tag match for all the blocks within the set
- **Fully Associative:** Tag match for all the blocks within the cache
**Cache Framework (2/2)**

**Block Replacement:** Which block should be replaced on a cache miss?

- **Direct Mapped:** No Choice
- **n-way Set-Associative:** Based on replacement policy
- **Fully Associative:** Based on replacement policy

**Write Strategy:** What happens on a write?
- **Write Policy:** Write-through vs. write-back
- **Write Miss Policy:** Write allocate vs. write no allocate
Reading Assignment

- Large and Fast: Exploiting Memory Hierarchy
  - Chapter 7 sections 7.3 (3rd edition)
  - Chapter 5 sections 5.3 (4th edition)