CS354: Machine Organization and Programming

Lecture 21
Wednesday the October 21\textsuperscript{th} 2015

Section 2
Instructor: Leo Arulraj

© 2015 Karen Smoler Miller
© Some examples, diagrams from the CSAPP text by Bryant and O’Hallaron
1. Programming Assignment 2 was due by 9 AM today. You can submit it upto 48 hours after the deadline with penalties.

2. Email me if you will have conflicts with the CS354 Midterm Exam 2:
   Nov 10th Tues 5:30 PM to 7:00 PM at Van Vleck Room B130(Section 2)
   (Come to the Location about 15 mins earlier)
Lecture Overview

1. Types of Cache misses

2. Looking up the cache contents in Set Associative Caches

3. Tracing through an example Set Associative Cache
On a miss

- Send the memory request to main memory.
- Memory returns the entire block containing the needed byte/word.
- Place the block into the frame.
  - Set the tag bits
  - Mark the frame valid.

And, while doing this, extract the byte/word and return it to the processor, completing the memory access.
Why middle order bits are used as Set Index?

Higher order bits lead to consecutive addresses mapping to same set.
Types of misses:

1) compulsory
2) conflict
3) capacity
Types of Misses

• **Compulsory or cold misses:** Cache is empty to start with and will miss.

• **Conflict misses:** Cache has space but because objects map to the same cache block they keep missing.

• **Capacity misses:** Cache does not have space because size of the working set exceeds the size of the cache.
Conflict misses are common

Consider:

```c
float dotprod(float x[8], float y[8])
{
    float sum = 0.0; register int i;
    for(i=0; i<8; i++)
        sum += x[i] * y[i];
    return sum;
}
```

Analyze for \((S,E,B,m) = (2,1,16,6)\)
Conflict misses are common

• It causes thrashing: repeatedly loading and evicting same cache blocks

• Thrashing is easy to avoid once you know it is going on: Use padded arrays so that the accessed elements are mapped to different cache sets
To reduce conflict misses, increase set associativity.

2-way set associative

2 blocks per set (line)

4-way set associative
Set Associative Cache Organization

Set 0:
- Valid
- Valid

Set 1:
- Valid
- Valid

Set S - 1:
- Valid
- Valid

\[ E=2 \text{ lines per set} \]
Larger set size tends to lead to higher hit ratio (due to fewer conflict misses)

amount of circuitry goes up, leading to increase in $T_c$
Looking up a memory address in Set Associative Cache

- Valid Tag Cache block
- Valid Tag Cache block
- Valid Tag Cache block

Selected set

\[ \begin{array}{c}
\text{Valid} \\
\text{Tag} \\
\text{Cache block}
\end{array} \]

\[ \begin{array}{c}
\text{Valid} \\
\text{Tag} \\
\text{Cache block}
\end{array} \]

\[ \begin{array}{c}
\text{Valid} \\
\text{Tag} \\
\text{Cache block}
\end{array} \]

\[ \begin{array}{c}
\text{Valid} \\
\text{Tag} \\
\text{Cache block}
\end{array} \]

\[ \begin{array}{c}
\text{Valid} \\
\text{Tag} \\
\text{Cache block}
\end{array} \]

\[ \vdots \]

\[ \begin{array}{c}
\text{Valid} \\
\text{Tag} \\
\text{Cache block}
\end{array} \]

\[ \begin{array}{c}
\text{Valid} \\
\text{Tag} \\
\text{Cache block}
\end{array} \]

\[ \begin{array}{c}
\text{Valid} \\
\text{Tag} \\
\text{Cache block}
\end{array} \]

\[ \begin{array}{c}
\text{Valid} \\
\text{Tag} \\
\text{Cache block}
\end{array} \]

\[ \begin{array}{c}
\text{Valid} \\
\text{Tag} \\
\text{Cache block}
\end{array} \]

- $t$ bits
- $s$ bits
- $b$ bits

- Tag
- Set index
- Block offset

- Selected set

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block

- Valid Tag Cache block
Looking up a memory address in Set Associative Cache

(1) The valid bit must be set

(2) The tag bits in one of the cache lines must match the tag bits in the address

(3) If (1) and (2), then cache hit, and block offset selects starting byte
Tracing through a sample Set Associative Cache from CSAPP textbook practice problem 6.13
Set Associative Cache Practice
Proble 6.13-6.16

- Consider a cache with: \((S,E,B,m) = (8,2,4,13)\)
- Analyze memory references to:
  - 0x0E34
  - 0x0DD5
  - 0x1FE4

The memory layout is shown in the next slide.
### 2-way set associative cache

<table>
<thead>
<tr>
<th>Set index</th>
<th>Tag</th>
<th>Valid</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>09</td>
<td>1</td>
<td>86</td>
<td>30</td>
<td>3F</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>45</td>
<td>1</td>
<td>60</td>
<td>4F</td>
<td>E0</td>
<td>23</td>
</tr>
<tr>
<td>2</td>
<td>EB</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>3</td>
<td>06</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>4</td>
<td>C7</td>
<td>1</td>
<td>06</td>
<td>78</td>
<td>07</td>
<td>C5</td>
</tr>
<tr>
<td>5</td>
<td>71</td>
<td>1</td>
<td>0B</td>
<td>DE</td>
<td>18</td>
<td>4B</td>
</tr>
<tr>
<td>6</td>
<td>91</td>
<td>1</td>
<td>A0</td>
<td>B7</td>
<td>26</td>
<td>2D</td>
</tr>
<tr>
<td>7</td>
<td>46</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set index</th>
<th>Tag</th>
<th>Valid</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>38</td>
<td>1</td>
<td>00</td>
<td>BC</td>
<td>0B</td>
<td>37</td>
</tr>
<tr>
<td>2</td>
<td>0B</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>3</td>
<td>32</td>
<td>1</td>
<td>12</td>
<td>08</td>
<td>7B</td>
<td>AD</td>
</tr>
<tr>
<td>4</td>
<td>05</td>
<td>1</td>
<td>40</td>
<td>67</td>
<td>C2</td>
<td>3B</td>
</tr>
<tr>
<td>5</td>
<td>6E</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>6</td>
<td>F0</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>7</td>
<td>DE</td>
<td>1</td>
<td>12</td>
<td>C0</td>
<td>88</td>
<td>37</td>
</tr>
</tbody>
</table>

The following figure shows the format of an address (one bit per box). Indicate (by labeling the diagram) the fields that would be used to determine the following:

- **CO**: The cache block offset
- **CI**: The cache set index
- **CT**: The cache tag

```
01234567890
```

© CSAPP textbook by Bryant and O’Hallaron
Cache Replacement Policies

• Which block to replace or evict to make space for new blocks?
  • **Random Replacement Policy:** chooses a random victim block.
  • **Least Recently Used (LRU) Policy:** chooses the block that was last accessed furthest in the past.
  • **Least Frequently Used (LFU) Policy:** chooses the block that was least frequently accessed in the past.