CS 354: Intro to Computer Systems (Spring 2018)
Lecture 13 - Set Associative Caches

The following problem concerns basic cache lookups.

- The memory is byte addressable.
- Memory accesses are to 1-byte words (not 4-byte words).
- Physical addresses are 12 bits wide.
- The cache is 4-way set associative, with a 2-byte block size and 32 total lines.

In the following tables, all numbers are given in hexadecimal. The contents of the cache are as follows:

<table>
<thead>
<tr>
<th>Index</th>
<th>Tag</th>
<th>Valid</th>
<th>Byte 0</th>
<th>Byte 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>29</td>
<td>0</td>
<td>34</td>
<td>29</td>
</tr>
<tr>
<td>1</td>
<td>F3</td>
<td>1</td>
<td>0D</td>
<td>8F</td>
</tr>
<tr>
<td>2</td>
<td>A7</td>
<td>1</td>
<td>E2</td>
<td>04</td>
</tr>
<tr>
<td>3</td>
<td>3B</td>
<td>0</td>
<td>AC</td>
<td>1F</td>
</tr>
<tr>
<td>4</td>
<td>80</td>
<td>1</td>
<td>60</td>
<td>35</td>
</tr>
<tr>
<td>5</td>
<td>EA</td>
<td>1</td>
<td>B4</td>
<td>17</td>
</tr>
<tr>
<td>6</td>
<td>1C</td>
<td>0</td>
<td>3F</td>
<td>A4</td>
</tr>
<tr>
<td>7</td>
<td>0F</td>
<td>0</td>
<td>00</td>
<td>FF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>Tag</th>
<th>Valid</th>
<th>Byte 0</th>
<th>Byte 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>39</td>
<td>AE</td>
<td>7D</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0C</td>
<td>3A</td>
<td>4A</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>D2</td>
<td>04</td>
<td>E3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>E0</td>
<td>0</td>
<td>B5</td>
<td>70</td>
</tr>
<tr>
<td>4</td>
<td>19</td>
<td>57</td>
<td>49</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>67</td>
<td>DB</td>
<td>8A</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>01</td>
<td>0</td>
<td>3A</td>
<td>CI</td>
</tr>
<tr>
<td>7</td>
<td>AF</td>
<td>1</td>
<td>B1</td>
<td>5F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>Tag</th>
<th>Valid</th>
<th>Byte 0</th>
<th>Byte 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>68</td>
<td>F2</td>
<td>8B</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>D9</td>
<td>1</td>
<td>A5</td>
<td>3C</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
<td>0</td>
<td>EE</td>
<td>05</td>
</tr>
<tr>
<td>3</td>
<td>37</td>
<td>1</td>
<td>49</td>
<td>F3</td>
</tr>
<tr>
<td>4</td>
<td>00</td>
<td>0</td>
<td>70</td>
<td>AB</td>
</tr>
<tr>
<td>5</td>
<td>18</td>
<td>1</td>
<td>2C</td>
<td>D3</td>
</tr>
<tr>
<td>6</td>
<td>7F</td>
<td>1</td>
<td>DF</td>
<td>05</td>
</tr>
<tr>
<td>7</td>
<td>99</td>
<td>0</td>
<td>AC</td>
<td>96</td>
</tr>
</tbody>
</table>

Part 1

The box below shows the format of a physical address. Indicate (by labeling the diagram) the fields that would be used to determine the following:

\[ CO \] The block offset within the cache line
\[ CI \] The cache index
\[ CT \] The cache tag

```
11 10  9  8  7  6  5  4  3  2  1  0
```

```
Part 2

For the given physical address, indicate the cache entry accessed and the cache byte value returned in hex. Indicate whether a cache miss occurs.

If there is a cache miss, enter "n" for “Cache Byte returned”.

Physical address: 0x3B6

Physical address format (one bit per box)

<table>
<thead>
<tr>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

Physical memory reference

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Offset (CO)</td>
<td>0x</td>
</tr>
<tr>
<td>Cache Index (CI)</td>
<td>0x</td>
</tr>
<tr>
<td>Cache Tag (CT)</td>
<td>0x</td>
</tr>
<tr>
<td>Cache Hit? (Y/N)</td>
<td></td>
</tr>
<tr>
<td>Cache Byte returned</td>
<td>0x</td>
</tr>
</tbody>
</table>

Part 3

In the 4-way Set Associative Cache given above, list all the hex memory addresses that will hit in Set 7.
Cache Miss Rate Analysis

You are evaluating the cache performance of the following code on a machine with a 64-byte direct-mapped data cache (C = 64) with block size of 16-bytes (B = 16).

You are given the following definitions:

```c
struct point {
    int x;
    int y;
};
struct point grid[4][4];
int total_x = 0, total_y = 0;
int i, j;
```

You should also assume the following:

- `sizeof(int) == 4.`
- `grid` begins at memory address 0.
- The cache is initially empty.
- The only memory accesses are to the entries of the array `grid`.
- Variables `i`, `j`, `total_x`, and `total_y` are stored in registers.

A. Determine the cache performance for the following code snippet 1:

```c
Code snippet 1:
for (i = 0; i < 4; i++) {
    for (j = 0; j < 4; j++) {
        total_x += grid[i][j].x;
    }
}
for (i = 0; i < 4; i++) {
    for (j = 0; j < 4; j++) {
        total_y += grid[i][j].y;
    }
}
```

1. What is the total number of reads that miss in the cache? 

2. What is the miss rate? 

B. Determine the cache performance for the following code snippet 2:

   **Code snippet 2:**
   ```c
   for (i = 0; i < 4; i++) {
       for (j = 0; j < 4; j++) {
           total_x += grid[i][j].x;
           total_y += grid[i][j].y;
       }
   }
   ```

   1. What is the total number of reads that miss in the cache? ________________

   2. What is the miss rate? ________________

C. Which of these 2 code snippets is better with respect to cache performance?

   a. Code snippet #1

   b. Code snippet #2

   Why? (just a single line explanation is sufficient)
Linking

Consider the three files `sum.h`, `sum.c` and `main.c` as shown below and answer the questions that follow.

`sum.h`

```c
#ifndef SUM_H
#define SUM_H
extern int sum(int, int);
#endif
```

`sum.c`

```c
#include "sum.h"
extern int num_ops;
static int global_sum = 0;
int sum(int x, int y)
{
    global_sum += (x+y);
    num_ops++;
    return x+y;
}
```

`main.c`

```c
#include "sum.h"
#define SUCCESS 0
int num_ops = 0;
int main()
{
    int a = 10;
    int b = 3;
    int result = sum(a,b);
    return SUCCESS;
}
```
The final executable **a.out** is produced by running the following command:

```
% gcc sum.c main.c -m32
```

**Questions:**

A. The variable **num_ops** is **ONLY declared but not defined** in the file __________

B. The variable **num_ops** is **defined** in the file __________

C. Do the following variables / functions need **relocation** when the executable (a.out) is formed? (Just answer: Yes or No).
   a. Variable **global_sum** in **sum.c** - __________
   b. Variable **result** in **main.c** - __________
   c. Variable **num_ops** in **main.c** - __________
   d. Function **sum()** in **sum.c** - __________
   e. Function **main()** in **main.c** - __________

D. Can the variable **global_sum** be accessed in the file **main.c** without changing the type of the variable **global_sum** in **sum.c**? Yes OR No.

E. What type of object file is **sum.o**?
   **sum.o** is generated using the command: gcc -c sum.c -m32
   i. **Relocatable Object File**
   ii. **Executable Object File**

F. Which part of **program memory** (code, data, stack, heap) are the following variables / functions stored?
   a. Variable **global_sum** in **sum.c** - ________________
   b. Variable **result** in **main.c** - ________________
   c. Variable **num_ops** in **main.c** - ________________
   d. Function **sum()** in **sum.c** - ________________
   e. Function **main()** in **main.c** - ________________