CS61C Summer 2016 Discussion 8 – Caches

In the following diagram, each blank box in the CPU Cache represents 8 bits (1 byte) of data. Our memory is **byte-addressed**, meaning that there is one address for each byte. Compare this to **word-addressed**, which means that there is one address for each word.

Index bits= log$_2$(Number of index rows)  Offset bits= log$_2$(Number of offsets columns)

<table>
<thead>
<tr>
<th>CPU Cache</th>
<th>Offset</th>
<th>Tag bits</th>
<th>Index bits</th>
<th>Offset bits</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3 2 1 0</td>
<td>30</td>
<td>0</td>
<td>2</td>
<td>32</td>
</tr>
</tbody>
</table>

What type of cache is this? **fully associative**

<table>
<thead>
<tr>
<th>CPU Cache</th>
<th>Index Number</th>
<th>Offset</th>
<th>Tag bits</th>
<th>Index bits</th>
<th>Offset bits</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>3 2 1 0</td>
<td>29</td>
<td>1</td>
<td>2</td>
<td>32</td>
</tr>
</tbody>
</table>

What type of cache is this? **2-way set associative**

<table>
<thead>
<tr>
<th>CPU Cache</th>
<th>Index Number</th>
<th>Offset</th>
<th>Tag bits</th>
<th>Index bits</th>
<th>Offset bits</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>3 2 1 0</td>
<td>28</td>
<td>2</td>
<td>2</td>
<td>32</td>
</tr>
</tbody>
</table>

What type of cache is this? **direct mapped**

1. **Direct mapped caches**  
   1. How many bytes of data can our cache hold? **16 bytes**  
      How many words? **4 words**
   2. Fill in the “Tag bits, Index bits, Offset bits” with the correct T:I:O breakdown according to the diagram.
   3. Let’s say we have a 8192KiB cache with an 128B block size, what is the tag, index, and offset of 0xFEEDF00D?

<table>
<thead>
<tr>
<th>Address size (bits)</th>
<th>Cache size</th>
<th>Block size</th>
<th>Tag bits</th>
<th>Index bits</th>
<th>Offset bits</th>
<th>Bits per row</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>32</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
### 2. Cache hits and misses

Assume we have the following cache. Of the 32 bits in each address, which bits do we use to find the row of the cache to use? **We use the 4th and 5th least significant bit since the offset is 3 bits**

Classify each of the following byte memory accesses as a cache hit (H), cache miss (M), or cache miss with replacement (R).

| CPU Cache | Index Number | Offset | | | | 1.0x00000004 | Index 0, Tag 0: M  
2.0x00000005 | Index 0, Tag 0: H  
3.0x00000068 | Index 1, Tag 3: M  
4.0x000000C8 | Index 1, Tag 6: R  
5.0x00000068 | Index 1, Tag 3: R  
6.0x000000DD | Index 3, Tag 6: M  
7.0x00000045 | Index 0, Tag 2: R  
8.0x00000004 | Index 0, Tag 0: R  
9.0x000000C8 | Index 1, Tag 6: R

#### 3. Analyzing C Code

```c
#define NUM_INTS 8192
int A[NUM_INTS]; /* A lives at 0x10000 */
int i, total = 0;
for (i = 0; i < NUM_INTS; i += 128) { A[i] = i; } /* Line 1 */
for (i = 0; i < NUM_INTS; i += 128) { total += A[i]; } /* Line 2 */
```

Let’s say you have a byte-addressed computer with a total memory of 1MiB. It features a 16KiB CPU cache with 1KiB blocks.

1. How many bits make up a memory address on this computer? **20**
2. What is the T:I:O breakdown? tag bits: **6** index bits: **4** offset bits: **10**
3. Calculate the cache hit rate for the line marked Line 1: **50%**
   
   The integer accesses are $4\times128=512$ bytes apart, which means there are 2 accesses per block. The first accesses in each block is a cache miss, but the second is a hit because $A[i]$ and $A[i+128]$ are in the same cache block.

4. Calculate the cache hit rate for the line marked Line 2: **50%**
   
   The size of $A$ is $8192\times4=2^{15}$ bytes. This is exactly twice the size of our cache. At the end of line 1, we have the second half of $A$ inside the cache, while in line 2 we start accesses from
the beginning of the array. Thus we cannot reuse any of the content of A and we get the same hit rate as before. Note that we do not have to consider cache hits for `total`, since the compiler will probably leave it in a register.

4. N-Way Set Associative Caches

1. Assuming 32 bits of physical memory, for an 8-way set associative 4KiB cache with 16B blocks, how big are the T, I, and O fields?
   \( T = 23, I = 5, O = 4 \)

2. How many total bits of storage are required for the cache if it uses write back and LRU replacement?
   \[ 2^8 \text{ rows, each row has } \text{Tag, Data, Valid, Dirty, } \log_2(\text{Associativity}) \text{ (for LRU)} = 23 + 128 + 1 + 1 + 3 = 156 \text{ bits. } 156 \times 2^8 = 39936 \text{ bits.} \]

3. Associativity usually improves the miss ratio, but not always. Give a short series of address references for which a two-way set-associative cache with LRU replacement would experience more misses than a direct-mapped cache of the same size.
   The key to this problem is to note that two addresses that map to the same set in a set-associative cache may map to a different block in a direct-mapped cache. A simple example is with a four-word cache and 1 word blocks. The example sequence is addresses 0, 2, 4, 0, 2. In the direct-mapped cache, 0 and 4 map to block 0, while 2 maps to block 2. Accesses to 0 and 4 miss because they conflict in block 0, but the second access to 2 hits. The hit rate is 1/5.

   With a 2-way set-associative cache, all three address map to the first set. Thus after the first two misses, 4 kicks out 0, 0 kicks out 2, and 2 kicks out 4. The hit rate is 0/5.