Name: Aaron Wang, Ethan Little
CSE 20221 Logic Design
HW10: Caches

Preamble

This assignment is a bit different from our recent assignments – you will not use Logisim-evolution or work with albaCore at all. Instead, you will answer questions about cache organization and simulation (by hand).

Note 1: we will go over multiple examples of how to work with direct-mapped caches on 12/5 and 12/10.

Note 2: for consistency with Computer Architecture, we will use the Ki/Mi/Gi/Ti/etc. notation to denote power-of-2 numbers. For example, 5 KiB is 5 * 2^{10} bytes, whereas 5 KB is 5 * 10^{10} bytes.
Ki = 2^{10}
Mi = 2^{20}
Gi = 2^{30}
Ti = 2^{40}

Problem 1 - Cache Organization

Tips: 1) pay attention to how the fields change between Questions 1, 2, 3, and 4 – Question 5 will ask you to explain the trends between each pair of questions; 2) in some cases you can copy/paste work that is unchanged from the previous question (if you make a mistake in an earlier question, you will not be penalized again).

Question 1

Consider a 16-bit, word-addressable machine like albaCore. Assume the memory hierarchy uses 4-word blocks and the machine has a 2 KiB direct-mapped cache. Determine the number of bits required for block offset, index, and tag *and* fill in the table noting the bit range for each field (see Problem 2 as an example for how to fill this table in). Show work for all "number of bits" calculations.

Number of bits for block offset: $log_2(4) = 2$
Number of bits for index: $\frac{2 \times 2^{10}\ bytes}{2\ bytes\ per\ word} \times \frac{1\ block}{4\ words} = 256\ blocks,\ log_2(256) = 8$
Number of bits for tag: $16 - (2 + 8) = 6$

Bit range:	15	10	9	2	1	0
Field:	Tag		Index		Block Offset	

Question 2

Note the **bolded** text indicating what is different here from Question 1.

Consider a 16-bit, word-addressable machine like albaCore. Assume the memory hierarchy uses **8-word blocks** and the machine has a 2 KiB direct-mapped cache. Determine the number of bits required for block offset, index, and tag *and* fill in the table noting the bit range for each field (see Problem 2 as an example for how to fill this table in). Show work for all "number of bits" calculations.

Number of bits for block offset: $log_2(8) = 3$
Number of bits for index: $\frac{2\times2^{10}\ bytes}{2\ bytes\ per\ word} \times \frac{1\ block}{8\ words} = 128\ blocks,\ log_2(128) = 7$
Number of bits for tag: $16 - (3 + 7) = 6$

Bit range:	15	10	9	3	2	0
Field:	Tag		Index		Block Offset	

Question 3

Note the **bolded** text indicating what is different here from Question 2.

Consider a 16-bit, word-addressable machine like albaCore. Assume the memory hierarchy uses 8-word blocks and the machine has an **8 KiB** direct-mapped cache. Determine the number of bits required for block offset, index, and tag *and* fill in the table noting the bit range for each field (see Problem 2 as an example for how to fill this table in). Show work for all "number of bits" calculations.

Number of bits for block offset: $log_2(8) = 3$
Number of bits for index: $\frac{8\times2^{10}\ bytes}{2\ bytes\ per\ word} \times \frac{1\ block}{8\ words} = 512\ blocks,\ log_2(512) = 9$
Number of bits for tag: $16 - (3 + 9) = 4$

Bit range:	15	12	11	3	2	0
Field:	Tag		Index		Block Offset	

Question 4

Note the **bolded** text indicating what is different here from Question 3.

Consider a **64-bit**, word-addressable machine. Assume the memory hierarchy uses 8-word blocks and the machine has an 8 KiB direct-mapped cache. Determine the number of bits required for block offset, index, and tag *and* fill in the table noting the bit range for each field (see Problem 2 as an example for how to fill this table in). Show work for all "number of bits" calculations.

Number of bits for block offset: $log_2(8) = 3$
Number of bits for index: $\frac{8 \times 2^{10}\ bytes}{8\ bytes\ per\ word} \times \frac{1\ block}{8\ words} = 128\ blocks,\ log_2(128) = 7$
Number of bits for tag: $64 - (3 + 7) = 54$

Bit range:	63	10	9	3	2	0
Field:	Tag		Index		Block Offset	

Question 5
Answer each of the following by explaining the trends demonstrated in Questions 1-4. If you are unsure of the trend because of the limited data points in this problem, refer to examples from class and/or do the math for a few additional cache organizations. In some cases there may be *no change*, but make sure to indicate that. Note: these refer to changes in terms of powers of 2, because that is common and makes the trends clearer, but it is possible to have a cache size, etc. that is not a power of 2.

a. Between Q1 to Q2, the **block size** changes. As the **block size** increases by a power of 2 (e.g., 1 ⟷ 2, 2 ⟷ 4, 4 ⟷ 8, etc.) how does this affect the number of bits for block offset, index, and tag?

Increasing block size by a power of two increases the bits required for the offset by 1, decreases the bits required for the index by 1, and does not affect the bits for the tag.

b. Between Q2 to Q3, the overall **cache size** changes. As the overall **cache size** increases by a power of 2 (1 KiB ⟷ 2 KiB, 2 KiB ⟷ 4 KiB, 4 KiB ⟷ 8 KiB, etc.) how does this affect the number of bits for block offset, index, and tag?

Increasing cache size by a power of two increases the bits required for the index by 1, decreases the bits for the tag by 1, and does not affect the bits needed for the offset.

c. Between Q3 to Q4, the overall **word size** changes. As the overall **word size** increases by a power of 2 (e.g., 16-bit ⟷ 32-bit, 32-bit ⟷ 64-bit, etc.) how does this affect the number of bits for block offset, index, and tag?

Increasing word size by a power of two decreases the bits required for the index by 1, increases the bits for the tag by the new power of two plus the additional bit gained from the index, and does not affect the bits needed for the offset.

Problem 2 - Cache Simulation
Consider a 32-bit, word-addressable machine with the following number of bits required for block offset, index, and tag as well as the table noting the bit range for each field.

Number of bits for block offset: 1
Number of bits for index: 8
Number of bits for tag: 23

Bit range:	31	9	8	1	0	0
Field:	Tag		Index		Block offset	

Given the access pattern in the table below, and the initial cache state, indicate whether each access is a hit or miss. For both hits and misses, explain why the access results in a hit or miss. For misses, explain what will be brought into the cache and where specifically it will be placed. If data is evicted, note that, but you do not need to determine if the data is sent to memory (write policy is not specified). You are **not** required to update the cache itself for this problem (sidenote: filling in the tag and valid bits and showing the complete history of cache state *is* something we typically do in Computer Architecture).

Only the first 10 indices of the cache are shown (out of a total 2^8), and all accesses will map to these indices (if they don't, check your work). The actual data stored is not important, so an X (don't care) is included in each word of each cache block that currently has valid data.

Cache

	Data		Tag (23 bits, hexadecimal)	Valid (1 bit)
Index (8 bits, hexadecimal)	Word 1	Word 0		
00				
01				
02	X	X	0x710A31	1
03				
04				
05				
06				
07				
08				
09	X	X	0x055F00	1
…	…	…	…	…

Accesses (as hexadecimal memory addresses)	**Summary (hit or miss and explanation)**
E2146200	Miss; there’s no data at index: 00 710a31 00 0
0ABE0012	Hit; At index: 09 the tag is 0x055F00 so it matches. 055F00 09 0
E2146201	Hit; put in there from Instruction #1 b/c it put’s in whole block not just that one word 710a31 00 1
32638207	Miss; There’s no data in the index: 03 1931c1 03 1
94113804	Miss; There is different data at that spot in the cache. At index: 02 the tag is 710A31 while we want 0x4a089c 4a089c 02 0

Deliverable Checklist

Problem 1 (25 points)

	Deliverable	Points
1.	5 points for each of 5 questions	25

Problem 2 (25 points)

	Deliverable	Points
1.	5 points for each of 5 accesses	25