Cache Simulator Demo

Let's get some practice with the cache simulator! First, go to:

`https://courses.cs.washington.edu/courses/cse351/cachesim/`

At the top you'll see 4 boxed regions:

- System Parameters † — This lets you play around with the structure/format of the cache
- Manual Memory Access † — This is where you actually make reads and writes to memory
- History — An interactive log of executed accesses. You can type/paste accesses here, too!
- Simulation Messages — Describes the most recent actions made by the simulator.

† These include "Explain" toggles that walk you through execution step-by-step.

---

a) Set the following System Parameters (but *don't* generate the system yet):

Address Width → **6**, Cache Size → **16**, Block Size → **4**, Associativity → **2**, leave the rest at default values.

Based on just the system parameter numbers above shown, predict the following:

i) Highest memory address: 0b **0011 1111** ii) Number of sets in cache: **2**

[*Click "Generate System" to verify your responses*]

b) We are about to **READ** the byte at the address **0x2A**. Predict the following:

i) This block will be placed in set #: **0** ii) The stored tag bits will be: 0b **101**

iii) The 4 bytes of *data* in this block are (in order): 0x**e9**, 0x**36**, 0x**ae**, 0x**32**

[*Enter "2a" into the Read Addr and click "Read" to verify your responses*]

c) We are about to **WRITE** the byte **0xB1** to the address **0x1B**. Predict the following:

i) This block will be placed in set #: **0** ii) The stored tag bits will be: 0b **011**

[*Enter "1b" into the Write Addr and "b1" into the Write Byte and then click "Write" to verify your responses*]

iii) Notice that the value of the byte at address 0x1B is different in the cache and memory.

What indicates this disparity in the cache? **The dirty bit**

What would have happened if our write miss policy were "**No Write-Allocate**" instead?

We would write directly to memory and not cache the block starting at 0x18

d) We are about to **READ** the byte at address **0x01**. Predict the following:

i) This block will be placed in set #: **0** ii) The stored tag bits will be: 0b **000**

iii) Will this access cause a conflict/replacement? (circle one) (Yes) No

iv) If yes, which block will be evicted? (circle one) (Read from (b)) Write from (c)

[*Enter "01" into the Read Addr and click "Read" to verify your responses*]

e) We are about to **WRITE** the byte **0xE9** to the address **0x1C**. Predict the following:

i) This block will be placed in set #: **1** ii) The stored tag bits will be: 0b **011**

iii) Will this access cause a conflict/replacement? (circle one) Yes (No)

iv) If yes, which block will be evicted? Read from (b) Write from (c) Read from (d)

[*Enter "1c" into the Write Addr and "e9" into the Write Byte and then click "Write" to verify your responses*]

f) At this point, your **History** should show:

```
  R(0x2a) = M
  W(0x1b, 0xb1) = M
  R(0x01) = M
  W(0x1c, 0xe9) = M
>
```

Append the bolded text below so that your History looks like:

```
  R(0x2a) = M
  W(0x1b, 0xb1) = M
  R(0x01) = M
  W(0x1c, 0xe9) = M
> W(0x03, 0xff)
  R(0x27)
  R(0x10)
  W(0x1d, 0x00)
```

[*Click "Load." You'll notice that " = ?" is appended to each of these new memory accesses*]

Predict if '?' will resolve to Hit (H) or Miss (M) for each of the new accesses:

i) `W(0x03, 0xff) =` **H**

ii) `R(0x27) =` **M**

iii) `R(0x10) =` **M**

iv) `W(0x1d, 0x00) =` **H**

[*Click the down arrow (↓) to verify your responses for each access*]

g) The cache, after the 8 executions detailed above, should look like this:

	V	D	T	Cache Data	
Set 0	1	1	0	20 f6 ef ff	2
	1	0	2	b8 bd 1a ca	1
Set 1	1	1	3	e9 00 f6 e5	1
	1	0	4	1a 6f 7e 63	2

The small numbers on the right (outside of the sets) indicate how recently used each line is within the set, with smaller numbers being *more recently* used).

i) An **LRU** replacement policy will evict which block on the next conflict in set 0? (Line 1) Line 2

ii) What is one benefit of using **LRU** over **Random**?

Favors temporal locality, as local variables usually are reused frequently.

iii) What is one benefit of using **Random** over **LRU**?

Cheaper and faster to use as there is no need to maintain record of the most recently used block.

h) If we were to flush the cache right now (don't actually) how many bytes in memory would change? **3**

How many bytes would change if we were using **Write Through** instead of **Write Back**? **0**

Can you explain why these numbers are the same/different? (if not, try changing the write hit policy and re-running using the history above).

Write Back won't write the new value to memory directly but will instead cache it and mark its block as dirty. When any dirty block is removed from the cache the memory corresponding to that block will be updated.

Write Through will write any new value to memory directly, thus meaning that no block in the cache will be dirty and no values in memory need to be updated when flushing the cache.