Computer Architecture Laboratory
Assignment 6

Add caches to the simulated memory system.

- Add one cache between the IF stage and the main memory. Let us call this the "level 1 instruction cache" or the `L1i-cache`.
- Add one cache between the MA stage and the main memory. Let us call this the "level 1 data cache" or the `L1d-cache`.

The configurations of the caches are as follows:

Cache size	16B	128B	512B	1kB
Latency	1 cycle	2 cycles	3 cycles	4 cycles
Line Size	4B			
Associativity	2			
Write Policy	Write Through			

Perform the following analyses:

1. First fix the size of the `L1d-cache` at 1kB. Vary the size of the `L1i-cache` from 16B to 1kB (remember to change latency accordingly), and study the performance (instructions per cycle). Plot your results for the different benchmarks.
2. Now fix the size of the `L1i-cache` at 1 kB. Vary the size of the `L1d-cache` from 16B to 1kB, and plot your results for the different benchmarks.
3. In your report, correlate the nature of the benchmark to the observed plot.
4. Create a toy-benchmark that shows significant performance improvement when the L1i-cache is increased from 16B to 128B. Assume favorable L1d-cache size.
5. Create a toy-benchmark that shows significant performance improvement when the L1d-cache is increased from 16B to 128B. Assume favorable L1i-cache size.

Note: Implement a single "`Cache`" class in the `processor/memorysystem` package. The `L1i-cache` and the `L1d-cache` must be objects of this class.

Tips

- Implement a separate "`CacheLine`" class. It will have, among other fields, an integer array field named `data`, and an integer field named `tag`.
- In the `Cache` class, instantiate an array of `CacheLine`'s of size equal to the number of lines required.
- In the `Cache` class, have a function: `cacheRead(int address)`. This function is called when the pipeline makes a read (load) request. This function takes an address as an argument. If the line corresponding to this address is present in the cache, it returns the corresponding 4B value to the pipeline. If the line corresponding to this address is not present, it calls the `handleCacheMiss()` function.
- In the `Cache` class, have a function: `cacheWrite(int address, int value)`. This function is called when the pipeline makes a write (store) request. This function takes an address and a value as arguments. If the line corresponding to this address is present in the cache, the value is written to the line, the line written to the main memory, and (in parallel) the pipeline is informed that the store has completed. If the line corresponding to this address is not present, it calls the `handleCacheMiss()` function.
- In the `Cache` class, have a function: `handleCacheMiss(int address)`. This function requests the main memory for the given line.
- In the `Cache` class, have a function: `handleResponse()`. This function handles the response from the lower level in the memory system (in our case, the main memory). The response is a cache line. This line has to be placed in the appropriate position in the cache. If there is a read request waiting for this line, the pipeline must be given the value. If there is a write request waiting, the value is written to the line, the line written to the main memory, and (in parallel) the pipeline is informed that the store has completed.
- Note: the function descriptions given above are merely to get you started. You will have to modify the function prototypes as you see fit.
- Don't forget: the hashes still have to match!