1 Problem statement

Simulate L1 cache in C++ for quad core processors, with cache coherence support.

1.1 Simulation details

1. Memory address is 32-bit. If any address is less than 32 bit, assume remaining MSB to be 0. E.g. the address 0x817b08 is actually 0x00817b08.
2. Each memory reference accesses 32-bit (4-bytes) of data. That is word size is 4- bytes.
3. We are only interested in the data cache and will not model the instruction cache.
4. Each processor has its own L1 data cache. The L1 data caches are backed up by main memory — there is no L2 data cache.
5. L1 data cache uses write-back, write-allocate policy and LRU replacement policy.
6. L1 data caches are kept coherent using MESI cache coherence protocol.
7. Initially all the caches are empty.
8. Ties are broken arbitrarily on the bus, when multiple cores attempt bus transactions simultaneously.
9. L1 cache hit is 1 cycle. Fetching a block from memory to cache takes additional 100 cycles. Sending a word from one cache to another (e.g., BusUpdate) takes only 2 cycles. However, sending a cache block with N words (each word is 4 bytes) to another cache takes 2N cycles. Assume that evicting a dirty cache block to memory when it gets replaced is 100 cycles.
10. Assume that the caches are blocking. That is, if there is a cache miss, the cache cannot process further requests from the processor core and the core is completely halted. However, the snooping transactions from the bus still need to be processed in the cache.
11. In each cycle, each core can execute at most one memory reference instruction.

You may need to make additional assumptions. Clearly state those assumptions in your report.

1.2 Input details and command line parameters

Tracefiles will be input to your simulator. Each parallel application is run using 4 threads on 4 processor cores, and 4 separate trace files with memory reference instructions have been generated.

E.g. `app1_proc0.trace, app1_proc1.trace, app1_proc2.trace` and `app1_proc3.trace` are 4 trace files for the first application app1 running with 4 threads on 4 processor cores. Each of the four files contain lines like the following, where the first column denotes Read(R) vs. Write(W), and the second column denotes the memory address for the read/write operation. Similar four files will be given for other applications.

```
W 0x7e1afe78
R 0x7e1ac04c
```

Traces for two parallel applications are available at http://www.cse.iitd.ac.in/~rijurekha/col216_2025/assignment3_traces.zip. `$make` should create an executable L1simulate, which should take in the following input parameters from command line.

```
$./L1simulate -h
-t <tracefile>:  name of parallel application (e.g.  app1) whose 4 traces are to be used
in simulation
-s <s>:  number of set index bits (number of sets in the cache = S = 2^s)
```

```
-E <E>:  associativity (number of cache lines per set)
-b <b>:  number of block bits (block size = B = 2^b)
-o:  <outfilename> logs output in file for plotting etc.
-h:  prints this help
```

1.3 Simulation output

Your program should generate the following output in each run. You can generate additional output as well.

1. Number of read/write instructions per core.
2. Total execution cycles per core to complete its instruction trace.
3. Number of idle cycles per core (these are cycles where the core is waiting for the request to the cache to be completed i.e. it is not a 1-cycle L1 cache hit).
4. Data cache miss rate for each core (miss rate = #misses/#accesses).
5. Number of cache evictions per core.
6. Number of writebacks per core.
7. Number of invalidations on the bus.
8. Amount of data traffic (in bytes) on the bus.

1.4 Experiments and report

Write a PDF report in latex with three sections on the following:

1. Your implementation (e.g., main classes, data structures used, flow chart of important functions etc.).
2. Experimental section with graphs of the above simulation outputs with default parameters as 4KB 2-way set associative L1 cache per processor, with 32-byte block size. Run your simulator 10 times for the same application with these same default parameters, and report distributions of the outputs over the 10 runs. Remark on which outputs remain the same vs. which outputs change with each simulator run, and why.
3. Add code in the simulator to compute maximum execution time for any core for an application. Vary the cache parameters (cache size, associativity, block size) from default to 3 other values, in powers of 2, one parameter at a time, keeping the others constant. Plot maximum execution time vs. each cache parameter, and explain your observations.

1.5 What to submit and grading criteria

Use git and commit regularly. Submit the repo (without the memory traces) on Moodle as `entrynum1_entrynum2_assign3.zip`, including (a) source code and Makefile, (b) readme on how to compile and run your code, and (c) PDF report as described above.

1. Code compiles and runs correctly with given traces [6 marks]
2. Code runs correctly on unseen traces [3 marks]
3. Report [6 marks]
4. Q/A on code and observations [5 marks]
5. Bonus [2 marks] Generate interesting traces (e.g. to demonstrate false sharing). These traces can be small, and even hand-generated.