For this assignment you must extend your Assignment 5 cache simulator to handle associativity and write policies. The cache configuration will be determined by command-line arguments, as will the input file name.

For this assignment, memory accesses will be both reads and writes. The input file will have lines of the format of "0xhhhhhhhh A", where A is either "R" or "W", for read or write. Again, it does not matter what the data value is, all we need to know are the sequences of addresses and the operation. You can also assume that each address is a **word-aligned** address, meaning its lowest two bits are 0. The write policy for the cache will be either write-through or write-back, as given by a command-line argument.

Your program will need to simulate the appropriate cache (from the configuration arguments) over the memory access trace, and at the end print out the following statistics:

* The total number of data bytes that the cache stores (i.e. the cache size)
* The total number of overhead bytes in the cache (round to integer, assume 2 bits per block for LRU)
* The total number of memory accesses (i.e., the size of input)
* The number of memory reads (how many input lines are reads)
* The number of memory writes (how many input lines are writes)
* The number of cache hits on reads
* The number of cache hits on writes
* The number of writes to memory (== memory writes if write-through, or number of evicted dirty blocks if write-back)
* The overall cache hit rate (as a fraction between 0 and 1.0).

For set-associativity greater than 1, you must implement a Least-Recently-Used replacement policy within each set. An easy way to do this is: keep an LRU counter for each block; each time a block is accessed, reset its counter to 0, and increment all other block counters in the set; when a new block is entering the set, replace the block with the highest counter. You do not need to limit your simulation counter to 2 bits. This of course assumes that all blocks are full; if the cache set isn't full yet, then just fill an empty block slot.

Your program's command line syntax is:

**progname [options] inputfilename**

Your program must take the following command line options (any order):

* "**-b #**" will set the block size in 32-bit words (legal values are positive powers of 2, default is 1).
* "**-a #**" will set the associativity (legal values are positive powers of 2, default is 1 (direct-mapped)).
* "**-n #**" will set the total number of data words (not blocks) in the cache (legal values are positive powers of 2, default is 1024).
* "**-w T|B**" will set the write policy, T sets it to write through, B to write back; default is write-back.

For write-back operation, part of the block overhead must be a dirty bit, which is a flag indicating whether or not a write has occurred in that block. Blocks that have not had a write are "clean" and do not need written back to memory when they are evicted, since they are unchanged.

In testing your program, you should probably construct some small traces by hand, and run your program with them on a small cache configuration, and verify by hand that your program is working correctly.

For larger tests, you can take your previous programming assignments and just print the address of each array element as it is being accessed (use the %p format). This will give you large input files to test, but you probably will not be able to decide whether your program is calculating the correct hit rate (since the input will be too big to hand-calculate).

Submit the source code for the program. Good programming practices (good commenting, good program structuring (i.e., creating and using functions), good indentation, good function and variable naming) will contribute to your grade (as will poor programming practices, too!).

### Submission

**Turned In!**

Apr 23 at 11:37pm

[Submission Details](https://nmsu.instructure.com/courses/1049933/assignments/4984079/submissions/3367007)

[Download CacheSet-1.h](https://nmsu.instructure.com/courses/1049933/assignments/4984079/submissions/3367007?download=107954887)

[Download CacheSet-1.cpp](https://nmsu.instructure.com/courses/1049933/assignments/4984079/submissions/3367007?download=107954888)

[Download Set-1.h](https://nmsu.instructure.com/courses/1049933/assignments/4984079/submissions/3367007?download=107954889)

[Download Set-1.cpp](https://nmsu.instructure.com/courses/1049933/assignments/4984079/submissions/3367007?download=107954891)

[Download test-2.cpp](https://nmsu.instructure.com/courses/1049933/assignments/4984079/submissions/3367007?download=107954892)

[Download makefile-2](https://nmsu.instructure.com/courses/1049933/assignments/4984079/submissions/3367007?download=107954893)

[Download test-2.txt](https://nmsu.instructure.com/courses/1049933/assignments/4984079/submissions/3367007?download=107954894)

[Download test1.txt](https://nmsu.instructure.com/courses/1049933/assignments/4984079/submissions/3367007?download=107954895)

[Download test2.txt](https://nmsu.instructure.com/courses/1049933/assignments/4984079/submissions/3367007?download=107954896)

[Download README-2](https://nmsu.instructure.com/courses/1049933/assignments/4984079/submissions/3367007?download=107954897)

Grade: 15 (20 pts possible)

Graded Anonymously: no

#### **Comments:**

Update: Incorrect display and number of writes to memory. -1 Incorrect number of total accesses. -1 Incorrect total number of reads/writes and cache size must be in bytes. -1 Incorrect number of total hits. -1 Incorrect overhead. -1

Jesus Gardea Rueda, May 4 at 10:47pm