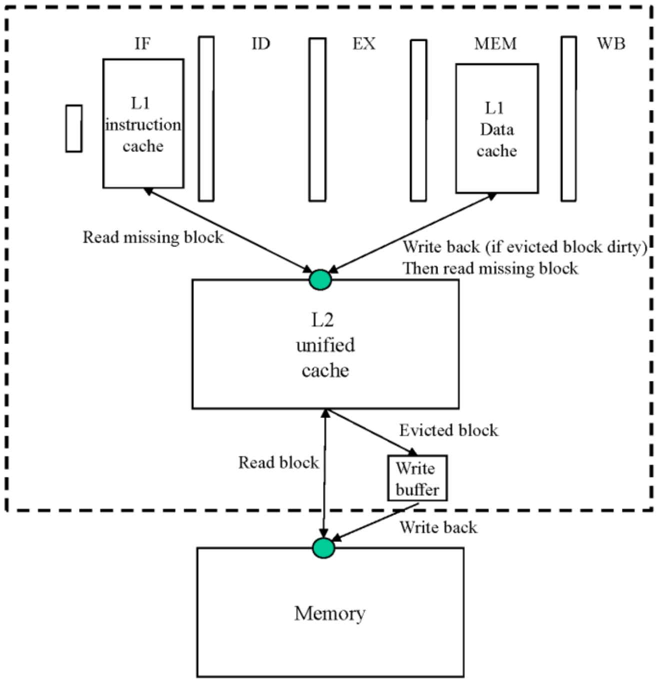
Project #2 (Due November 7 – by midnight)

NOTE: This project should be done by the same teams as Project 1 (unless changes are discussed with instructor).

The goal of this assignment is to study the effect of caches on CPU performance by adding L1 and L2 caches to the *five_stage* pipeline. You will ignore all other hazards in the pipeline and only study the effects of stalling due to cache misses. Specifically, the pipeline should stall if, while fetching an instruction in the IF stage, the instruction is not found in the L1 instruction cache. Similarly, the pipeline should stall if while loading or storing data in the MEM stage, a miss occurs in the L1 data cache. The L2 cache is a unified cache shared between the two L1 caches. All caches use the write back policy.



Your implementation should include a one-entry write buffer between the L2 and the memory to reduce the L2 miss penalty. Specifically, if an access to a block, B1, causes an L2 miss, and a dirty block, B2, is to be evicted to make room for B1, then B2 is put in the write buffer while B1 is being fetched from memory. Then, B2 is written back to memory after B1 is fetched and the pipeline resumes execution.

The implemented L2 should be single-ported (it can service only one access at a time). That is, the L2 will not be available to reply to requests from one of the two L1 caches until it finishes processing previously issued requests (possibly from the other L1 cache). Similarly, in your implementation, the memory should also be single ported. That is, if the memory is busy writing back the block from the write buffer, then it cannot service a new request from L2 until it finishes the write back.

The program *five_stage+.c* is a modification of *five_stage.c* provided for Project 1. It simulates the L1 caches and includes the file *cache.h* that contains the data structures and functions that implement caches. Your task is to add the L2 cache and the write buffer. You should also add the flexibility of reading the following memory hierarchy configuration parameters from a configuration file "cache_config.txt" (see this example configuration file):

1. the size of the L1 instruction cache in Kilo Bytes (0 means a perfect cache with miss penalty = 0)
2. the associativity of the L1 instruction cache
3. the size of the L1 data cache in Kilo Bytes
4. the associativity of the L1 data cache
5. the size of the L2 cache in Kilo Bytes
6. the associativity of the L2 cache
7. the cache block size in bytes (the same for all three caches)
8. the L2 access time (in cycles) – this latency is to access (read or write) a block.
9. the memory access time (in cycles) – this latency is to access (read or write) a block.
10. the number of "warm up" instructions (see below).

All 10 parameters are integers and each of the first seven parameters should be a power of 2 (to simplify the simulation). The access time for the L1 cache is assumed to be zero (access is completed within the execution cycle). The “warm up” period is a period in which the simulation in running without collecting results. For example, if the “warm up” period is 1000 instructions, then collecting statistics in the simulation (execution cycles, memory references, miss rates) should start with the fetching of the 1001st instruction from the trace. The purpose of the warm up is to skip the transient period by filtering out compulsory misses from the statistics since they are independent of the cache organization.

Your implementation should make the following assumptions:

- All the blocks in the caches are initially invalid.
- LRU block replacement (implemented using an LRU stack for each cache set).
- The L2 is “inclusive” of the L1 data cache in the sense that a block is not evicted from L2 if it resides in the L1 data cache.

In addition to the total number of execution cycles, the simulator should report the total number of memory accesses and the miss rates for L1 and L2 (all calculated after the warm up period). As in project 1, you should write your own simple test programs to verify the correctness of your simulation (it makes sense to set the warm up period to 1 when you are running a few instructions to test the simulator).

After completing and testing your implementation, you should conduct the following experiments using the two long traces that you used in project 1 assuming a memory access time of 60 cycles and an L2 access time of 6 cycles. For each experiment, you should have a “warm up” period of 100000 instructions.

Experiment 1: You will examine the effect of the block size and associativity assuming identical L1 instruction (I) and data (D) caches and an L2 which is 8 times larger than each L1. You will do this by experimenting with different L1 cache sizes (1KB, 4KB and 16KB), two block sizes (4B and 16B) and different associativity (direct mapped and 4-way associative). For each of the two long traces, build two tables similar to the ones shown below, one to report the miss rates and the other to report the total number of cycles needed to complete the execution of the trace (after warm up).

Miss rate (after warm up)																		
	Direct map									4-way associative								
	1KB L1 (8KB L2)			4KB LI (32KB L2)			16KB LI (128KB L2)			1KB L1 (8KB L2)			4KB LI (32KB L2)			16KB LI (128KB L2)		
	I	D	L2	I	D	L2	I	D	L2	I	D	L2	I	D	L2	I	D	L2
4B																		
16B																		

Execution cycles (after warm up)																		
	Direct map									4-way associative								
	1KB L1 (8KB L2)			4KB LI (32KB L2)			16KB LI (128KB L2)			1KB L1 (8KB L2)			4KB LI (32KB L2)			16KB LI (128KB L2)		
	I	D	L2	I	D	L2	I	D	L2	I	D	L2	I	D	L2	I	D	L2
4B																		
16B																		

After completing the tables, comment on what you learned from the experiment about the effect of the size, associativity and block size on each of the caches.

Experiment 2: Assume that the block size for the architecture is 16B and the L2 size is 128KB. You are assigned the task of using your simulator to select the configurations of the L1 caches which is most efficient (strikes the right tradeoff between performance and cost) for the two given benchmark traces. This includes the sizes and associativities of the L1 caches (I and D caches may be of different sizes and associativity). Assume that the cost of a cache is S * (1 + 0.1 * A), where S is the cache size and A is the associativity. You should define your own criteria for balancing the tradeoff between performance and cost.

What to submit:

1) Your *five_stage+.c* and *cache.h* (which includes all functions and declarations).
2) A .pdf file with
    a. The tables containing the results of running your simulation with *trace_view_on* = 0 for each long trace file. For experiment 1, the values of the parameters are specified. For experiment 2, you should select the parameters iteratively until you find the best design.
    b. Your remarks and explanation of the results and the conclusions that you draw from the experiments.

Grading Rubric:

Points (total 60)	*Criteria*
15	A basic simulator that compiles and run
15	Correctness of the simulator (based on your own verification efforts)
30	Correctness of the simulator (based on the TA tests)
20	Results and analysis for experiment 1
20	Results and analysis for experiment 2

NOTES:

- Make sure your code compiles and runs on linux.cs.pitt.edu.
- Make sure that the results submitted are collected with the final submitted version of the code
- The cache configuration file should be always named “cache_config.txt” and be in the directory where the program is executing. It should not be an input parameter.
- Your output format should look as follows:

    [cycle number] instruction that exited the pipeline in this cycle

- At the end of the simulation, print the cache statistics in the following format:
    - I-cache: *** accesses, *** misses, miss rate = ***
    - D-cache: *** accesses, *** misses, miss rate = ***
    - L2-cache: *** accesses, *** misses, miss rate = ***
- All submissions of written content should be in a single pdf file.