1. Cache Documentation

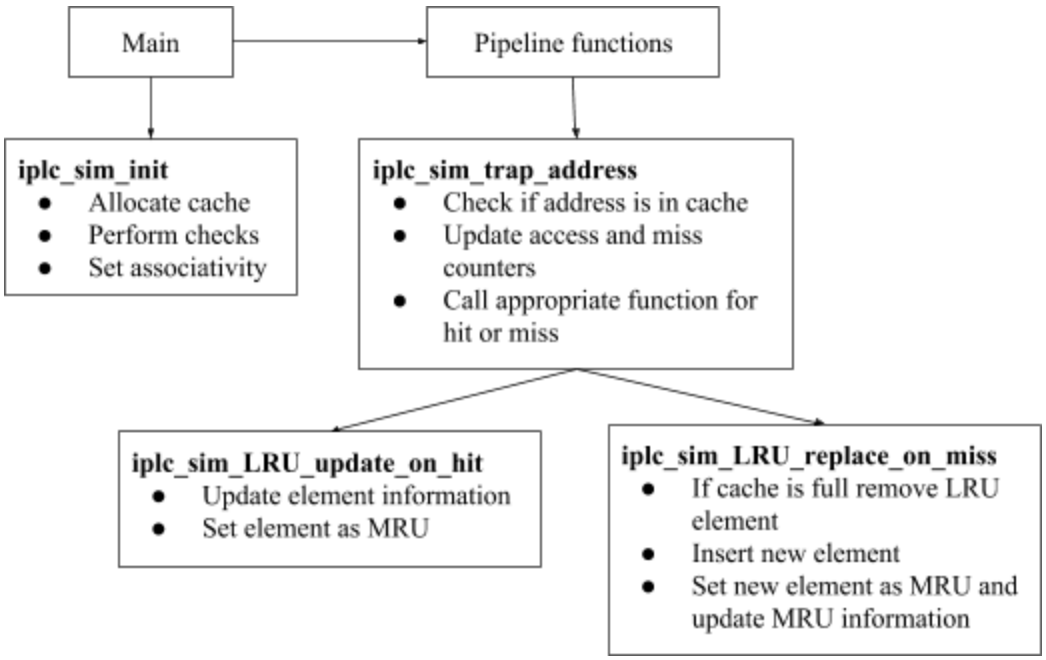
void iplc_sim_init(int index, int blocksize, int assoc)
This function takes an index, the block size, and the associativity of the cache as parameters. Before allocating the cache it will perform checks to make sure the cache meets pre-established specifications such as ensuring the cache does not go over the max cache size. It uses this information to allocate memory and dynamically create the cache with the proper associativity.

int iplc_sim_trap_address(unsigned int address)
This function deals with checking if a given address is in the cache. It is called by various pipeline functions and considers the given associativity in addition to searches through the cache data structure. It will update the counter for hits, misses, and cache accesses. After looking through the appropriate entries for the address it will call the appropriate function to deal with a hit or a miss.

void destroy_cache()
Goes through each element of the cache and deallocates the memory.

void iplc_sim_LRU_replace_on_miss(int index, int tag)
When it is determined that an element is not in the cache, this function is called. It takes a cache index and a tag as parameters, both of which are determined by the caller. It puts the element in the cache and declares it as the most recently used entry. It then updates the entry that was previously the most recently used entry. If the cache is full, the LRU item is then replaced.

void iplc_sim_LRU_update_on_hit(int index, int assoc_entry)
This function takes in a cache index and a given associative entry. It is called if an element has already been determined to be in the cache and it will update the element's information. It will set it as the MRU and update the previous MRU information on the cache.



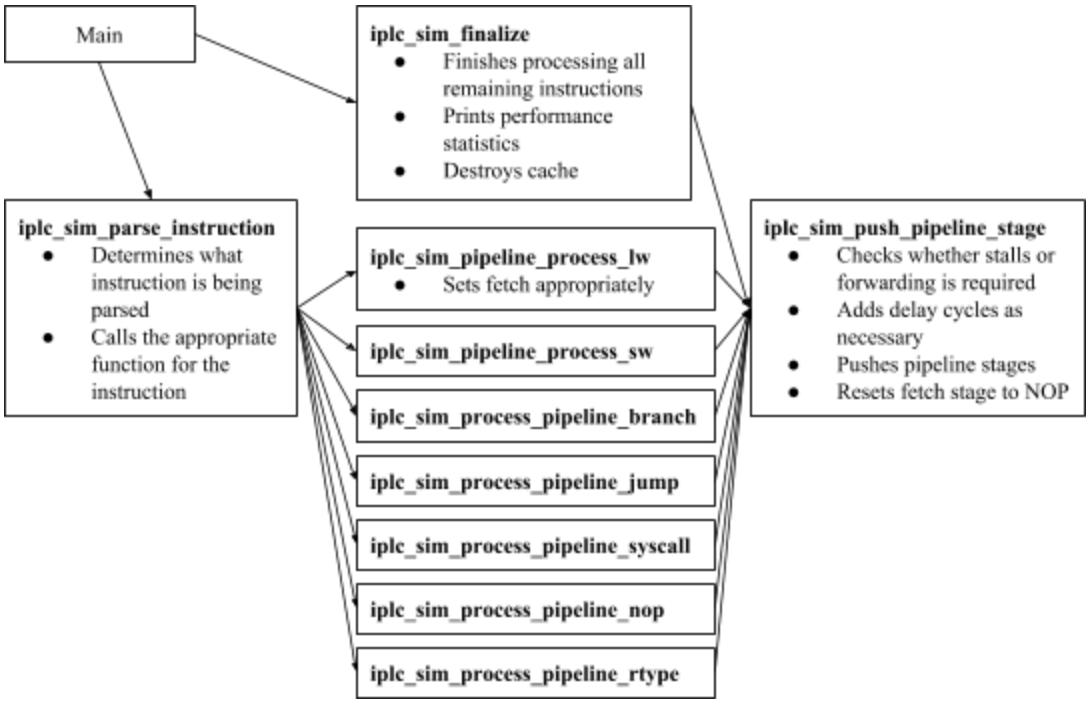
2. Pipeline Documentation

void iplc_sim_dump_pipeline()
This function dumps the current contents of the pipeline, utilizing switch case architecture.

void iplc_sim_push_pipeline_stage()
This function performs a variety of tasks related to the pipeline, such as checking whether various stages of the pipeline require stalls or forwarding, checking writeback stage, checking for branch and branch prediction, checking for LW delays and data hits or misses (and adding delay cycles as necessary), checking for SW memory access and data misses (and adding delay cycles as necessary), incrementing the number of pipeline cycles by 1, pushing pipeline stages, and resetting fetch stage to NOP.

void iplc_sim_process_pipeline_rtype(char *instruction, int dest_reg, int reg1, int reg2_or_constant)
Example implementation of an rtype pipeline process.

void iplc_sim_process_pipeline_lw(int dest_reg, int base_reg, unsigned int data_address),
void iplc_sim_process_pipeline_sw(int src_reg, int base_reg, unsigned int data_address),
void iplc_sim_process_pipeline_branch(int reg1, int reg2),
void iplc_sim_process_pipeline_jump(char *instruction),
void iplc_sim_process_pipeline_syscall(),
void iplc_sim_process_pipeline_nop()
Assigns the variables in the fetch to the provided arguments.



3. Performance Evaluation

Index Size (bits)	Block Size (words)	Associativity	Cache Size	Branch Prediction	Cache Miss Rate	CPI
7	1	1	7168	NOT TAKEN	0.006943	1.279393
6	2	1	5632	NOT TAKEN	0.004238	1.254102
6	4	1	9664	NOT TAKEN	0.001422	1.227070
6	1	2	7296	NOT TAKEN	0.005577	1.265978
5	2	2	5696	NOT TAKEN	0.002872	1.240689
5	4	2	9728	NOT TAKEN	0.001422	1.227070
5	1	4	7424	NOT TAKEN	0.005577	1.265978
4	2	4	5760	NOT TAKEN	0.002872	1.240689
4	4	4	9792	NOT TAKEN	0.001422	1.227070
7	1	1	7168	TAKEN	0.006943	1.076080
6	2	1	5632	TAKEN	0.004238	1.051017
6	4	1	9664	TAKEN	0.001422	1.026171
6	1	2	7296	TAKEN	0.005577	1.062671
5	2	2	5696	TAKEN	0.002872	1.037608
5	4	2	9728	TAKEN	0.001422	1.024171
5	1	4	7424	TAKEN	0.005577	1.062671
4	2	4	5760	TAKEN	0.002872	1.037608
4	4	4	9792	TRUE	0.001422	1.024171

There are a few key trends that are pertinent to mention. Because of the nature of the system, as the block size increases, the overall cache size increases as well. This is important to note because one can notice the trend between a larger cache size and a lower cache miss rate and CPI. Although a larger cache size is a downside in terms of memory usage, significant performance increases can be had as a result. When looking directly at the cache size, cache miss rate, and CPI, an inverse correlation can be observed. As the cache size increases, both the cache miss rate and CPI decrease. This is significant because it suggests that if one uses more memory for the cache, one can increase the CPI of the system by a considerable amount.

Another important aspect to consider is whether branch prediction is taken or not. The trends described above apply both when branch prediction is taken and when it is not, but when branch prediction is taken, significant performance increases are exhibited. With the same size cache, branch prediction lowers the CPI even more so. Although this has a tradeoff of having to delay the system in the event of incorrect prediction, the overall benefit to the CPI is sufficient enough to incur this penalty.

The "best" cases, in terms of both cache miss rate and CPI, were with index size 5, block size 4, associativity 2, TAKEN, and with index size 4, block size 4, associativity 4, TAKEN. These cases have the same cache miss rate and CPI. The "worst" case in terms of cache miss rate and CPI was with index size 7 block size 1, associativity 1, NOT TAKEN.