Assignment 2

Name: Deepak Nolastname

ASU ID: 1211301777

1) Cache Simulation in SimpleScalar is implemented as follows:

a) **Structure** (cache.h): Each cache block/line is represented as linked list node with two main reference pointers (next, previous). A cache set (which is linked list) consists of group of different cache blocks (nodes of link list) linked together. The group of different sets forms a cache structure. Different cache sets are referenced using the index field and each block in the set is referenced by binding index for the block.

A cache structure has following main parts: name, number of sets, block size, cache associativity, replacement policy, cache status variables (cache hits, misses, etc.), base cache address, array of cache sets and derived data for internal cache maintenance. There is an enumeration for cache replacement policy which decides how the data is replaced/updated when cache miss/hit occurs. Also, the deceleration of different functions are provided here so that those functions can be accessed elsewhere by including this header file.

b) **Functions** (cache.c): There are two main functions used for simulation that are present in cache.c.

i) cache_create: this is used to create a cache structure with the configurations defined by the user. The cache structure is defined in cache.h and on a call to this function a cache structure is created and all the configurations are set as per the user input. Firstly, all the inputs are validated for correctness and then parameters are set. The derived parameters are then calculated and the status registers are initialized. Based on the size of cache block, associativity and the number of sets, memory is dynamically allocated and linked together. The data is inserted at the head of the linked list. All the parameters required to maintain the cache are initialized in this function.

ii) cache_access: this function is used to access the data inside a cache. The cache update (on a cache miss/hit) happens inside this function. It takes in parameters including cache name, access type (read/write), number of bytes to access, pointer to access data buffer, etc. Firstly the pointer to the address to access (block and set) is calculated. These pointers are then validated for permissions and accessibility. Then if found valid, based on the associativity of the cache (high/low), present address to be accessed and block status, the cache hit/miss is determined. If there is a cache miss, based on the replacement policy, the pointer in the linked list are updated. Accordingly the linked list node which is to be removed (based on replacement policy) is removed and the new data is then loaded. In case of a cache hit, only the status of the cache block/set (again based on replacement policy) is updated for the future accesses. Subsequently, the different cache status values are updated.

iii) There are multiple other utility functions which are available in the cache.c. These are used to update the linked list(cache sets), print the cache status/configuration, check validity of a cache block, flush the cache/block and perform hashing for highly associative cache. The functions such as the block update, updates (removes/adds) the position of a cache block inside the cache set. There are different functions to add update the cache at different positions inside the set.

2). a) Following is the miss rate table for different cache configuration for **Instruction cache:**

NOTE: Due to large difference in scales, the data for Anagram has been plotted in a different diagram.

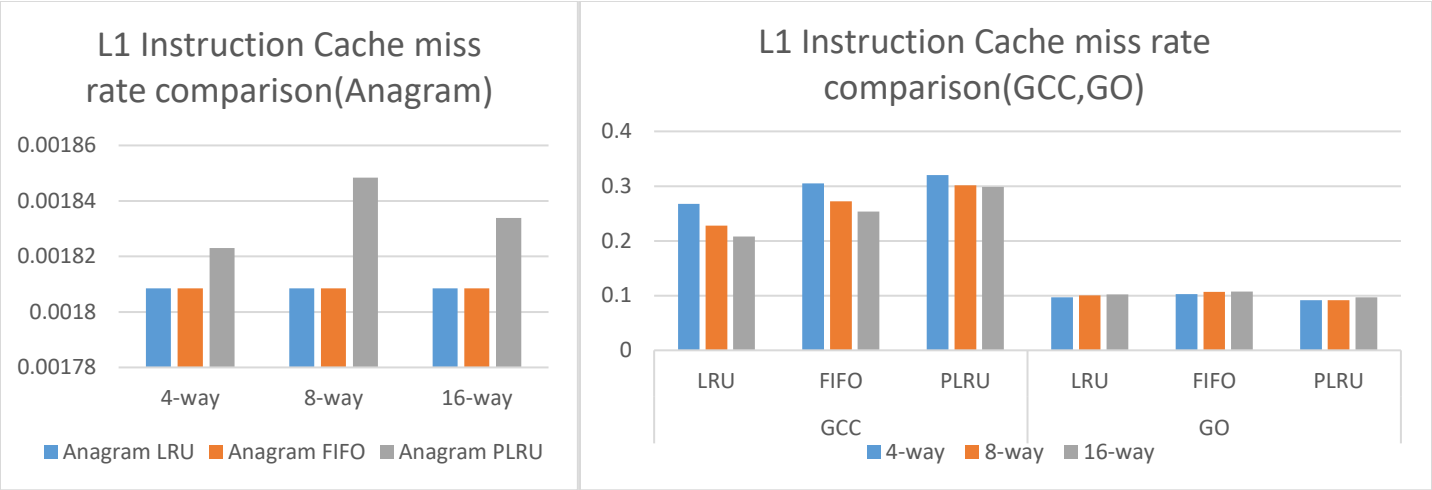

Figure 1. Instruction Cache miss rate (in %) with respect to different configurations

	Anagram			GCC			GO		
	LRU (%)	FIFO (%)	PLRU (%)	LRU (%)	FIFO (%)	PLRU (%)	LRU (%)	FIFO (%)	PLRU (%)
4-way	0.001808448	0.001808445	0.001822975	0.267572	0.305166	0.320317	0.096864	0.102647	0.091396
8-way	0.001808449	0.001808445	0.001848395	0.227552	0.271963	0.301592	0.100536	0.106912	0.091366
16-way	0.001808449	0.001808445	0.001833871	0.208133	0.25347	0.298326	0.102142	0.107591	0.097103

b) Following is the miss rate table for different cache configuration for **Data cache:**

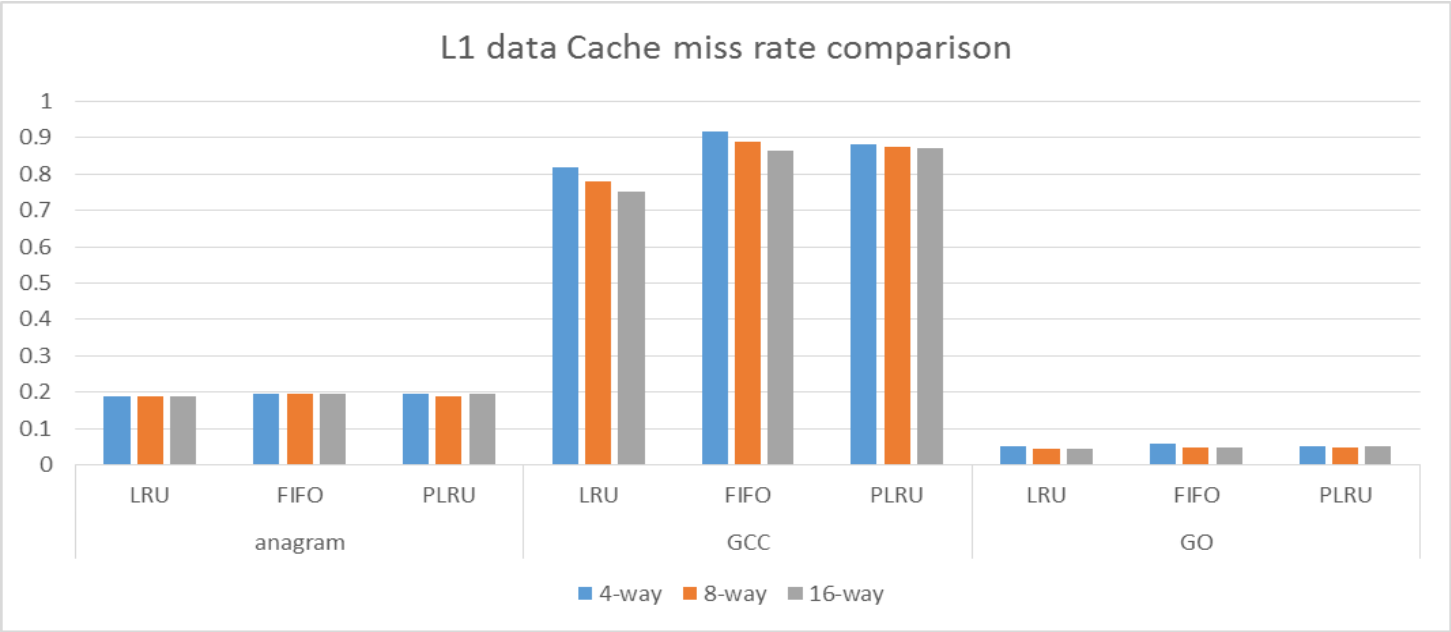

Figure 2. Data Cache miss rate (in %) with respect to different configurations

	Anagram			GCC			GO		
	LRU (%)	FIFO (%)	PLRU (%)	LRU (%)	FIFO (%)	PLRU (%)	LRU (%)	FIFO (%)	PLRU (%)
4-way	0.18915	0.19708	0.196362844	0.8178	0.91656	0.88234	0.05224	0.05954	0.05262
8-way	0.18872	0.19636	0.188315404	0.78035	0.89003	0.87349	0.0457	0.049	0.04905
16-way	0.18876	0.19643	0.193993062	0.7532	0.86556	0.87163	0.04461	0.04711	0.05064

3).a) Following is the miss rate table for different cache configuration for **Instruction cache:**

NOTE: Due to large difference in scales, the data for Anagram has been plotted in a different diagram.

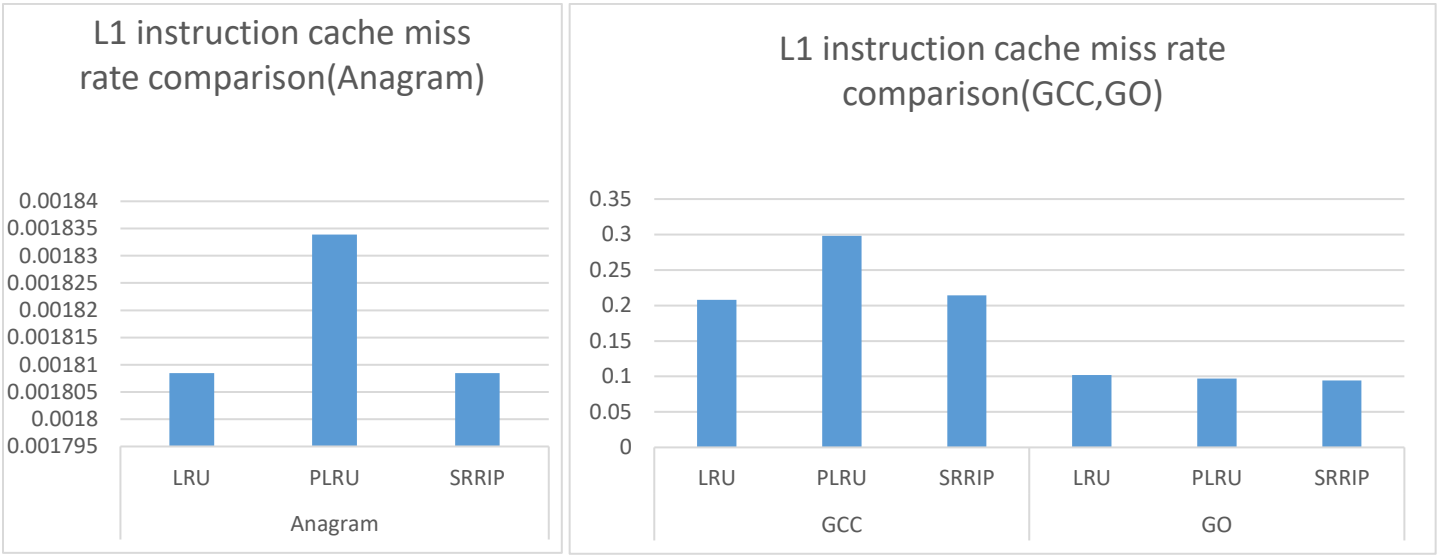

Figure 3. Instruction Cache miss rate (in %) with respect to different configurations

	Anagram			GCC			GO		
	LRU (%)	PLRU (%)	SRRIP (%)	LRU (%)	PLRU (%)	SRRIP (%)	LRU (%)	PLRU (%)	SRRIP (%)
16-way	0.18876	0.19399	0.18952	0.7532	0.87163	0.76183	0.04461	0.05064	0.04337

b) Following is the miss rate table for different cache configuration for **Data cache:**

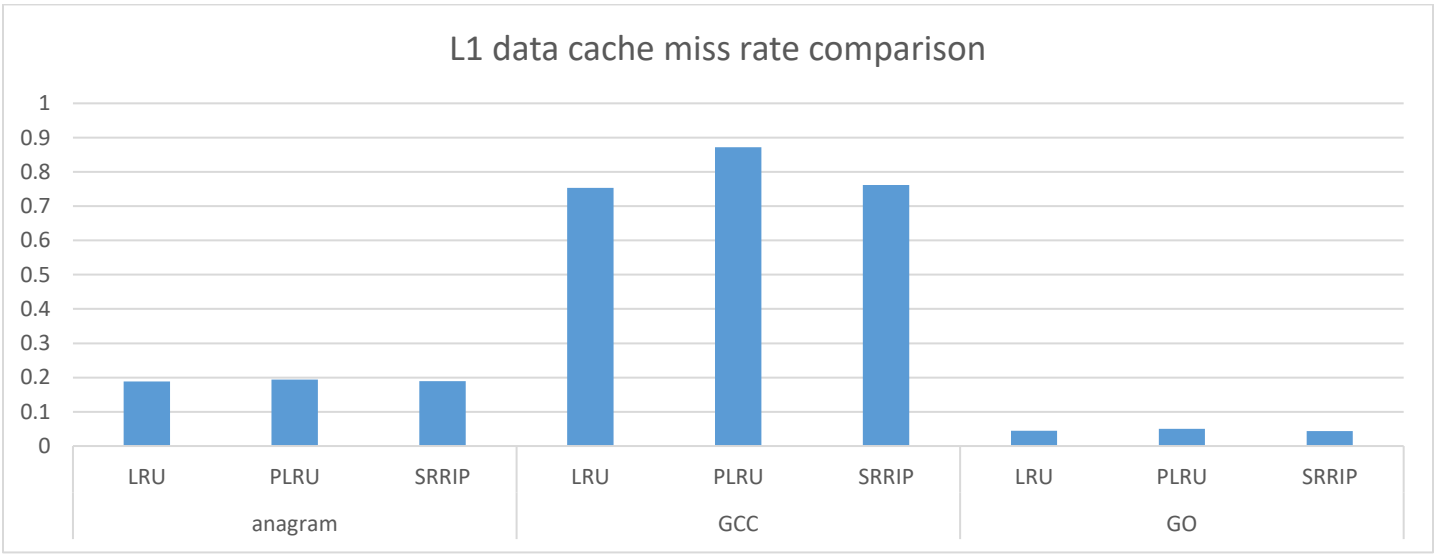

Figure 4. Data Cache miss rate (in %) with respect to different configurations

	Anagram			GCC			GO		
	LRU (%)	PLRU (%)	SRRIP (%)	LRU (%)	PLRU (%)	SRRIP (%)	LRU (%)	PLRU (%)	SRRIP (%)
16-way	0.18876	0.19399	0.18952	0.7532	0.87163	0.76183	0.04461	0.05064	0.04337