Main Data Structures Used:

cache_block: Holds one Cache Block

```
typedef struct  cache_block {
        unsigned long  tag;
        //char*          pData; //2^len_offset bytes of data.
        int            valid;   //Is theBlock valid
        //int            dirty;   //For write_back. Since we only use write through, not needed.
} cache_block;
```

cache_set: Holds one cache set. Including array of cache blocks and the index for First In First Out (FIFO) Replacement Algorithm.

```
typedef struct cache_set {
        cache_block*  pCacheBlock;  //array of CacheBlock
        int       index_to_be_reused; //For Replacement algorithm: First In First Out (FIFO)
} cache_set;
```

counter: Holds the counters for cache performance.

```
typedef struct counter {
        unsigned int            hit;
        unsigned int            miss;
        unsigned int            read;
        unsigned int            write;
} counter;
```

Cache_Spec: Holds the specifications of the cache

```
typedef struct Cache_Spec {
        int         cache_size;         //how many bytes in total of all blocks
        int         cache_setsize;      //how many blocks in one set
        int         block_size;         //how many bytes in one block
        int         len_address;        //48 bits in this situation
        int         number_of_blocks;   //# of blocks
        int         number_of_sets;     //# of sets
        int         len_offset;         //from block_size
        int         len_index;          //from cache_setsize
        int         len_tag;            //len_address - len_offset - len_index
        int         start_offset;       //always 0
        int         start_index;        //len_offset
        int         start_tag;          //len_address - len_offset - len_index
} Cache_Spec;
```

Comparison of Type A and Type B cache:

In case of associative, A and B are identical as there is no index.

In all other cases, Type A has better performance than Type B.

In Type A, the index for the cache set is in the least significant bits after the offset. This means that consequent blocks will go to different cache sets. So, the caching is more evenly spread throughout all the cache sets. All cache sets are more heavily utilized.

In Type B, the index for the cache set are in the most significant bits. As a result, the memory is divided into number of sets ($48bits$: $2^{16} * 4G/number\_of\_set = 262{,}144G/number\_of\_set$) and each is assigned to one cache set. For all practical purposes, that means only one cache set is utilized. All other cache sets are idle.