# Cache Simulator

Name:Gaurav Chaudhari
Rollno :2021MCS2132

Cache simulator:
In this project we tried to implement cache simulator such that from memory traces we can Conclude how much hit rate miss rate and number of cpu cycles will take place if given cache replacement policy is employed.

We implemented for 3 major cache replacement policies:

1. LRU : A Least Recently Used (LRU) Cache organizes items in order of use, allowing you to quickly identify which item hasn't been used for the longest amount of time.
2. FIFO : The way it works is once the cache is full it simply replaces the first page that was placed in the cache with the desired page,
3. LFU: Least Frequently Used (LFU) is a caching algorithm in which the least frequently used cache block is removed whenever the cache is overflowed.

We divide cache policy in following structures

```
typedef struct {
 int setcount;     number of sets
 int countblock;   number of blocks
 int block_size;   size of blocks in bytes
 int evict_mode;  type of policy implemented
} parameters;

typedef struct {
 uint32_t tag;
 bool valid;
 unsigned int load_ts;  load timestamp for block
 unsigned int access_ts;   access timestamp for the block
 unsigned int count_access; number of accesses of particular block
} Block;

typedef struct {
 Block *countblock;       set associativity count of blocks in each set
} Set;

typedef struct {
 Set *setcount;   array of sets in cache
 int num_sets;   number of sets total
 int blocks_in_set;    number of blocks in each es
 int loads;             number of load instructions
 int stores;            number of store instructions
```

```
 int load_hit;
 int load_miss;
 int store_hit;
 int store_miss;
 int cycles;
 int ts;
} Cache;
```

These are the functions in program

```
Cache *create_init(int num_sets, int blocks_in_set);
Initializing cache
void free_cache(Cache *c);
Freeing cache values
void summary(Cache *c,char* outputfile);
Printing result in output file

void load_block(Block *b, uint32_t t, unsigned int ts);
Loading the block in memory

bool check2power (int num);
 Checking value power of 2

int find_pow(int num);

uint32_t bitmask(uint32_t source, int length, int lower);

Block *is_hit(Cache *c, Set *s, int slots, uint32_t t);

Block *replace_block(int lru, Set *s, int slots);
Replace block in
Block *handle_write_back(parameters *p, Cache *c,
      Set *s, uint32_t t);

void handle_load(parameters *p, Cache *c,
     Set *s, uint32_t t, bool hit);
handle load instruction

void handle_store(parameters *p, Cache *c,
     Set *s, uint32_t t, bool hit);
Handling store instructions
#endif
```

Driver code for replacing based on replacement policy:

```
 for (int i = 0; i < slots; i++) {
   b = s->countblock + i;

   if (!(b->valid)) {
     return b;
   }
   else {
     if (MODE==0) {
//lru


 if (lru_index == -1 || b->access_ts < lru_timestamp) {
   lru_timestamp = b->access_ts;
   lru_index = i;
 }
```

Least recently used policy check access timestamp of of all the blocks in the set and return block to the lowest timestamp value

```
     }
     else if (MODE==1){
//fifo
 if (fifo_index == -1 || b->load_ts < fifo_timestamp) {
   fifo_timestamp = b->load_ts;
   fifo_index = i;
 }
```

FIFO policy check load timestamp of of all the blocks in the set and return block to the lowest timestamp value

```
     }
   //lfu
     else if (MODE==2)
     {
       if (lfu_index==-1 || b->count_access < minfreq){
         minfreq = b->count_access;
         lfu_index = i;
       }
     }
```

Least Frequently Used policy check access count of of all the blocks in the set and return block to the lowest access counts

```
   }
 }
```