```cpp
#include <iostream>
#include <ctype.h>
#include <string.h>
#include <math.h>
#include <cstdlib>
#include <ctime>
#include <sstream>
#include <fstream>
#include <vector>
#include <sys/time.h>

#define TOTAL_ADDRESS_BITS 64
#define KB 1024

using namespace std;
typedef struct
{
    bool is_available;
    unsigned long long int tag;
} cache_entries;

unsigned long int miss = 0;
unsigned long int r_miss = 0;
unsigned long int w_miss = 0;
unsigned long int total_access = 0;
unsigned long int total_read = 0;
unsigned long int total_write = 0;
unsigned long int total_hit = 0;

int decimalBinaryBits(int n)
{
    int num_bits = 0;

    while (1)
    {
        if ((1 << num_bits) & n)
            break;
        else
            num_bits++;
    }

    return num_bits;
}

unsigned long int create_mask(unsigned a, unsigned b)
{
    unsigned long int r = 0;
    for (unsigned i = a; i <= b; i++)
```

```
        r |= 1 << i;

    return r;
}

void check_cache_entry(vector<vector<cache_entries> > &cache, int set_id, int assoc,
unsigned long int addr, char mode, char rep_policy, unsigned long int tag)
{
    bool found = false;
    int j = 0, i = 0;
    srand(time(0));
    struct timespec t;

    total_access++;
    if (mode == 'r')
        total_read++;
    else
        total_write++;

    for (int i = 0; i < assoc; i++)
    {
        if ((cache[set_id][i].is_available == true) && (cache[set_id][i].tag == tag))
        {

            if (rep_policy == 'l')
            {
                cache_entries c1;
                c1.tag = tag;
                c1.is_available = true;
                cache[set_id].erase(cache[set_id].begin() + i);
                cache[set_id].insert(cache[set_id].begin(), c1);
                cache[set_id].resize(assoc);
            }

            found = true;
            total_hit++;
            break;
        }
    }

    if (!found)
    {
        miss++;
        if (mode == 'r')
            r_miss++;
        else
            w_miss++;
        if (rep_policy == 'l')
        {
            cache_entries c1;
            c1.tag = tag;
            c1.is_available = true;
            cache[set_id].pop_back();
            cache[set_id].insert(cache[set_id].begin(), c1);
            cache[set_id].resize(assoc);
        }
        else if (rep_policy == 'r')
        {
            for (i = 0; i < assoc; i++)
            {
                if (cache[set_id][i].is_available == false)
                {
                    j = i;
                    break;
                }
                else
                {
                    j = rand() % assoc;
                }
            }

            cache[set_id][j].is_available = true;
            cache[set_id][j].tag = tag;
        }
    }
}

int main(int argc, char *argv[])
{

    int nk = 0, assoc = 0, blocksize = 0;
    char rep_policy;
    unsigned long int nb = 0, ns = 0;
    int bo = 0, si = 0, tag = 0;
    unsigned long int tag_num = 0;
    char mode;
    unsigned long int r = 0;
    unsigned long int result = 0;
    unsigned long int addr = 0x1FFF;
    stringstream ss;
    int block_id = 0;
    int set_id = 0;
    int offset = 0;
    string tmpLine;
    nk = atoi(argv[1]);
    assoc = atoi(argv[2]);
    blocksize = atoi(argv[3]);
    rep_policy = argv[4][0];

    nb = (nk * KB) / blocksize;
    ns = nb / assoc;
    bo = decimalBinaryBits(blocksize);
    si = decimalBinaryBits(ns);
    tag = TOTAL_ADDRESS_BITS - si - bo;

    r = create_mask(bo + si, 63);

    vector<vector<cache_entries> > cache(ns, vector<cache_entries>(assoc));
    for (int i = 0; i < ns; i++)
        for (int j = 0; j < assoc; j++)
            cache[i][j].is_available = false;

    while (getline(cin, tmpLine))
    {
        stringstream ss1(tmpLine);
        ss1 >> mode;
        ss1 >> hex >> addr;

        block_id = addr / blocksize;
        set_id = block_id % ns;
        offset = addr % blocksize;
        result = r & addr;
        tag_num = result >> (bo + si);

        check_cache_entry(cache, set_id, assoc, addr, mode, rep_policy, tag_num);
        ss.clear();
    }

    cout << miss << " " << 100 * (double)miss / (double)total_access << "% ";
    cout << r_miss << " " << 100 * (double)r_miss / (double)total_read << "% ";
    cout << w_miss << " " << 100 * (double)w_miss / (double)total_write << "% " <<
endl;

    return 0;
}
```