```c
//*********************************************************************
//                                                                    *
//                 University Of North Carolina Charlotte             *
//                                                                    *
//Program: Cache Simulator                                            *
//Description: This program is used to read trace.din file including  *
//             memory access operations (data rd/wr, instr. read)     *
//             and simulate cache behavior for different cache para.  *
//             then output the total number of misses.                *
//                                                                    *
//File Name: main.c                                                   *
//File Version: 1.0                                                   *
//Baseline: Homework_1_Delivery                                       *
//                                                                    *
//Course: ECGR5181                                                    *
//                                                                    *
//Prepared by: Karim H. ERIAN - kerian@uncc.edu - 801020354           *
//                                                                    *
//Under Suppervision of: Dr. Hamed Tabkhi                             *
//                                                                    *
//*********************************************************************
#include <stdio.h>
#include <math.h>
#include <string.h>


//functions declarations:
int argument_to_int(char *arg[]);
long long get_tag(char addr[]);
int get_index(char addr[]);
long long address_to_long(char addr[]);
int get_LRU(int indx, int lru[]);

//Global variables declarations:
int blockSize;
int cacheSize;
int maxNumberOfBlocks; //the real number of blocks in cache
int tagAddressLength;
int indx; //index
int indexLen; //index length
int offsetLen; //Offset Length



//*********************************************************************
// Function Name: main()                                              *
// Description: -Cache initialization and simulation                 *
//              -Call functions to:                                   *
//                 *translate arguments                               *
//                 *translate address                                 *
//                 *get tag and get index from address                *
// Input: strings: Cache type (separated/combined)-size-block size -  *
//                 set associativity (1 = DM or 4)                    *
// Return: int                                                        *
//*********************************************************************
int main(int argc, char *argv[])
{
//info about trace.din:
//file size is 832477 entries
//2: instrunction fetch
//0: data read
//1: data write

//Data initialization:
//preparing i/o files
   FILE *pfin;
```

```c
    char *mode = "r";
    FILE *pfout;
    long int i = 0; //counter to know the number of operations

    //init hits and misses counters
    int hit = 0;
    int miss = 0;

    //arrays for LRU flags
    int lru[4096][4]; //can take vaue from 0 to 3, 3 is the lru.
    int lruInstr[4096][4]; //same but for instructions.

    // address in decimal value (long long for the address size)
    long long addrInLong = 0;

    int op;//from file
    char address[8];//from file

    blockSize = argument_to_int(&argv[3]);
    cacheSize = argument_to_int(&argv[2]) * 1024;
    char cacheCombinedSeparated = *argv[1]; //c for combined and s for separated.
    char assoc = *argv[4]; //1  means direct map, 4 means 4 set associativity

    tagAddressLength = 0;
    maxNumberOfBlocks = (int)(cacheSize / blockSize);

    long long cacheBlockTag[4096][4];//to be used in comparison - our max @32K-8B
    long long cacheTagInstr[4096][4];//same for instr
    long long requiredTag;
    int limit = 1; // used as associativity number

    //loops counters
    int sc = 0;
    int lc = 0;
    int mc = 0;

    int flag = 0;//match address flag

    char hitORmiss = 'm'; //for debugging
    int lru_index = 0;

    int addressLen = 0;

    for (lc = 0; lc < 4096; lc++)
    {
       for (mc = 0; mc < 4; mc++)
       {
          //for (sc = 0; sc < 8; sc++)
          //{
             cacheBlockTag[lc][mc] = 0xffffffff;
             cacheTagInstr[lc][mc] = 0xffffffff;
             lru[lc][mc] = 0; //means empty
          //}
       }
    }
    if (assoc == '4')
    {
       limit = 4;
    } else {
       limit = 1;
    }

    offsetLen = (int)((float)log(blockSize)/log(2));
    if (limit == 1)
    {
       indexLen = (int)((float)log(maxNumberOfBlocks)/log(2));//direct map
```

```c
      }else {
         indexLen = (int)((float)log(maxNumberOfBlocks)/log(2)) - 2;
      }
      tagAddressLength = 32 - (indexLen + offsetLen);
      //opening file for reading
      pfin = fopen("trace.din",mode);
      if (pfin == NULL) {
        printf("Can't open input file\n");
        return(0);
      }
      //opening file for writing - used for debugging
      pfout = fopen("out.txt","w");
      //loop on file till end of file and read data inside
      while (fscanf(pfin, "%d %s",&op,address) != EOF)
      {
         //prepare required data
         addressLen = sizeof(address);
         indx = get_index(address);
         requiredTag = get_tag(address);
         addrInLong = address_to_long (address);

         ////////////////////
         //for combined or data cache:
         if (cacheCombinedSeparated == 'c' || ((cacheCombinedSeparated == 's') &&(op != 2)))
         {
            //search all set of cache
            for (lc = 0; lc < limit; lc++)
            {
               //init flag for tag found (0 = false)
               flag = 0;
               //for (sc = 0; sc < tagAddressLength; sc++)
               //{
                  if (requiredTag == cacheBlockTag[indx][lc])
                  {
                     //if tag is found, set the flag, increase hit counter, write
                     //h in the output file in front of address (for debugging)
                     flag = 1;
                     hit++; //we found a hit
                     hitORmiss = 'h';
                     lc = limit; //exit this entry to check the next entry
                  }else {
                     flag = 0;
                  }


               //}
            }
            if (flag == 0)//not found in any set
            {
               miss++;
               hitORmiss = 'm';
               //for (sc = 0; sc < tagAddressLength; sc++)
               //{
               if (limit == 1)
               {
                  //replacement policy
                  cacheBlockTag[indx][0] = requiredTag; //only 1 place in DM
               } else {
                  //using LRU policy for replacement
                  lru_index = get_LRU(indx,lru[indx]);
                  cacheBlockTag[indx][lru_index] = requiredTag;
               }
               //}
            }
         }else { //instructions cache
            for (lc = 0; lc < limit; lc++)
            {
               flag = 0;
               //for (sc = 0; sc < tagAddressLength; sc++)
               //{
                  if (requiredTag == cacheTagInstr[indx][lc])
                  {
                     flag = 1;
                     hit++; //we found a hit
                     hitORmiss = 'h';
                     lc = limit; //exit this entry to check the next entry
                  }else {
                     flag = 0;
                  }


               //}
            }
            if (flag == 0)//not found in any set
            {
               miss++;
               hitORmiss = 'm';
               //for (sc = 0; sc < tagAddressLength; sc++)
               //{
               if (limit == 1)
               {
                  cacheTagInstr[indx][0] = requiredTag;//use LRU to get it change 0
               } else {
                  lru_index = get_LRU(indx,lruInstr[indx]);
                  cacheTagInstr[indx][lru_index] = requiredTag;
               }
               //}
            }
         }
         ////////////////////
         //count number of entries
         i++;
         //o/p data in file for debugging
         fprintf(pfout, "%d  %s   %d    %lld     %lld
         %c\n",op,address,indx,requiredTag,addrInLong,hitORmiss);
      }
      printf("number of requests: %ld \n",i);
      fclose(pfout);
      fclose(pfin);
      float percent = hit/i * 100;
      //-------------------------
      printf("Tag length %d\nindex length  %d\noffset length
      %d\n",tagAddressLength,indexLen,offsetLen);
      printf("number of miss = %d and hits = %d \n\n",miss,hit);
      //-------------------------
      return 0;
   }

   //**********************************************************************
   // Function Name: argument_to_int                                      *
   // Description: transform passed argument into integer                 *
   // Input: 2D array                                                     *
   // Return: integer                                                     *
   //**********************************************************************
   int argument_to_int(char *arg[])
   {

      return result;
   }

   //**********************************************************************
   // Function Name: address_to_long                                      *
   // Description: transform passed address into ldecimal value           *
   // Input: 1D array                                                      *
   // Return: long long int                                                *
   //***********************************************************************
   long long address_to_long(char addr[])
   {

      return result;
   }

   //***********************************************************************
   // Function Name: get_tag                                               *
   // Description: get the tag from address into decimal value             *
   // Input: 1D array                                                      *
   // Return: long long int                                                *
   //***********************************************************************
   long long get_tag(char addr[])
   {

      return addressDec;
   }
   #if 0
   long long get_tag(char addr[], int addressLen)
   {

      return result;
   }
   #endif
   //***********************************************************************
   // Function Name: get_index                                             *
   // Description: get the index from address into decimal value           *
   // Input: 1D array                                                      *
   // Return: int                                                          *
   //***********************************************************************
   int get_index(char addr[])
   {

      return addressDec;

   }

   #if 0
   int get_index(char addr[], int addressLen)//needs adjustments
   {

      return result;
   }
   #endif

   //***********************************************************************
   // Function Name: get_LRU                                               *
   // Description: get the LRU block                                       *
   // Input: integer and 1D array                                          *
   // Return: int                                                          *
   //***********************************************************************
   int get_LRU(int indx,int lru[])
   {

      return result;
   }//end of get_LRU
   //end of file :)
```