

```

1  //*****
2  //
3  //          University Of North Carolina Charlotte
4  //
5  //Program: Cache Simulator
6  //Description: This program is used to read trace.din file including
7  //              memory access operations (data rd/wr, instr. read)
8  //              and simulate cache behavior for different cache para.
9  //              then output the total number of misses.
10 //
11 //File Name: main.c
12 //File Version: 1.0
13 //Baseline: Homework_1_Delivery
14 //
15 //Course: ECGR5181
16 //
17 //Prepared by: Karim H. ERIAN - kerian@uncc.edu - 801020354
18 //
19 //Under Supervision of: Dr. Hamed Tabkhi
20 //
21 //*****
22 #include <stdio.h>
23 #include <math.h>
24 #include <string.h>
25
26
27 //functions declarations:
28 int argument_to_int(char *arg[]);
29 long long get_tag(char addr[]);
30 int get_index(char addr[]);
31 long long address_to_long(char addr[]);
32 int get_LRU(int indx, int lru[]);
33
34 //Global variables declarations:
35 int blockSize;
36 int cacheSize;
37 int maxNumberOfBlocks; //the real number of blocks in cache
38 int tagAddressLength;
39 int indx; //index
40 int indexLen; //index length
41 int offsetLen; //Offset Length
42
43
44
45 //*****
46 // Function Name: main()
47 // Description: -Cache initialization and simulation
48 //              -Call functions to:
49 //                  *translate arguments
50 //                  *translate address
51 //                  *get tag and get index from address
52 // Input: strings: Cache type (separated/combined)-size-block size -
53 //              set associativity (1 = DM or 4)
54 // Return: int
55 //*****
56 int main(int argc, char *argv[])
57 {
58 //info about trace.din:
59 //file size is 832477 entries
60 //2: instruction fetch
61 //0: data read
62 //1: data write
63
64 //Data initialization:
65 //preparing i/o files
66 FILE *pfin;

```

```

67     char *mode = "r";
68     FILE *pfout;
69     long int i = 0; //counter to know the number of operations
70
71     //init hits and misses counters
72     int hit = 0;
73     int miss = 0;
74
75     //arrays for LRU flags
76     int lru[4096][4]; //can take vaue from 0 to 3, 3 is the lru.
77     int lruInstr[4096][4]; //same but for instructions.
78
79     // address in decimal value (long long for the address size)
80     long long addrInLong = 0;
81
82     int op;//from file
83     char address[8];//from file
84
85     blockSize = argument_to_int(&argv[3]);
86     cacheSize = argument_to_int(&argv[2]) * 1024;
87     char cacheCombinedSeparated = *argv[1]; //c for combined and s for separated.
88     char assoc = *argv[4]; //1 means direct map, 4 means 4 set associativity
89
90     tagAddressLength = 0;
91     maxNumberOfBlocks = (int)(cacheSize / blockSize);
92
93     long long cacheBlockTag[4096][4]; //to be used in comparison - our max @32K-8B
94     long long cacheTagInstr[4096][4]; //same for instr
95     long long requiredTag;
96     int limit = 1; // used as associativity number
97
98     //loops counters
99     int sc = 0;
100    int lc = 0;
101    int mc = 0;
102
103    int flag = 0; //match address flag
104
105    char hitORmiss = 'm'; //for debugging
106    int lru_index = 0;
107
108    int addressLen = 0;
109
110    for (lc = 0; lc < 4096; lc++)
111    {
112        for (mc = 0; mc < 4; mc++)
113        {
114            //for (sc = 0; sc < 8; sc++)
115            //{
116                cacheBlockTag[lc][mc] = 0xffffffff;
117                cacheTagInstr[lc][mc] = 0xffffffff;
118                lru[lc][mc] = 0; //means empty
119            //}
120        }
121    }
122    if (assoc == '4')
123    {
124        limit = 4;
125    } else {
126        limit = 1;
127    }
128
129    offsetLen = (int)((float)log(blockSize)/log(2));
130    if (limit == 1)
131    {
132        indexLen = (int)((float)log(maxNumberOfBlocks)/log(2)); //direct map

```

```

133 }else {
134     indexLen = (int)((float)log(maxNumberOfBlocks)/log(2)) - 2;
135 }
136 tagAddressLength = 32 - (indexLen + offsetLen);
137 //opening file for reading
138 pfin = fopen("trace.din",mode);
139 if (pfin == NULL) {
140     printf("Can't open input file\n");
141     return(0);
142 }
143 //opening file for writing - used for debugging
144 pfout = fopen("out.txt","w");
145 //loop on file till end of file and read data inside
146 while (fscanf(pfin, "%d %s",&op,address) != EOF)
147 {
148     //prepare required data
149     addressLen = sizeof(address);
150     indx = get_index(address);
151     requiredTag = get_tag(address);
152     addrInLong = address_to_long (address);
153
154     ///////////////////////////////////
155     //for combined or data cache:
156     if (cacheCombinedSeparated == 'c' || ((cacheCombinedSeparated == 's') &&(op != 2)))
157     {
158         //search all set of cache
159         for (lc = 0; lc < limit; lc++)
160         {
161             //init flag for tag found (0 = false)
162             flag = 0;
163             //for (sc = 0; sc < tagAddressLength; sc++)
164             //{
165                 if (requiredTag == cacheBlockTag[indx][lc])
166                 {
167                     //if tag is found, set the flag, increase hit counter, write
168                     //h in the output file in front of address (for debugging)
169                     flag = 1;
170                     hit++; //we found a hit
171                     hitORmiss = 'h';
172                     lc = limit; //exit this entry to check the next entry
173                 }else {
174                     flag = 0;
175                 }
176
177             //}
178         }
179         if (flag == 0)//not found in any set
180         {
181             miss++;
182             hitORmiss = 'm';
183             //for (sc = 0; sc < tagAddressLength; sc++)
184             //{
185                 if (limit == 1)
186                 {
187                     //replacement policy
188                     cacheBlockTag[indx][0] = requiredTag; //only 1 place in DM
189                 } else {
190                     //using LRU policy for replacement
191                     lru_index = get_LRU(indx,lru[indx]);
192                     cacheBlockTag[indx][lru_index] = requiredTag;
193                 }
194             //}
195         }
196     }
197     }else { //instructions cache
198         for (lc = 0; lc < limit; lc++)

```

```

199     {
200         flag = 0;
201         //for (sc = 0; sc < tagAddressLength; sc++)
202         //{
203             if (requiredTag == cacheTagInstr[indx][lc])
204             {
205                 flag = 1;
206                 hit++; //we found a hit
207                 hitORmiss = 'h';
208                 lc = limit; //exit this entry to check the next entry
209             }else {
210                 flag = 0;
211             }
212
213
214         //}
215     }
216     if (flag == 0)//not found in any set
217     {
218         miss++;
219         hitORmiss = 'm';
220         //for (sc = 0; sc < tagAddressLength; sc++)
221         //{
222             if (limit == 1)
223             {
224                 cacheTagInstr[indx][0] = requiredTag;//use LRU to get it change 0
225             } else {
226                 lru_index = get_LRU(indx,lruInstr[indx]);
227                 cacheTagInstr[indx][lru_index] = requiredTag;
228             }
229             //}
230         }
231     }
232     ///////////////////////////////////
233     //count number of entries
234     i++;
235     //o/p data in file for debugging
236     fprintf(pfout, "%d %s %d %lld %lld\n",op,address,indx,requiredTag,addrInLong,hitORmiss);
237 }
238 printf("number of requests: %ld \n",i);
239 fclose(pfout);
240 fclose(pfin);
241 float percent = hit/i * 100;
242 //-----
243 printf("Tag length %d\nindex length %d\noffset length\n",tagAddressLength,indexLen,offsetLen);
244 printf("number of miss = %d and hits = %d \n\n",miss,hit);
245 //-----
246 return 0;
247 }
248
249 //*****
250 // Function Name: argument_to_int *
251 // Description: transform passed argument into integer *
252 // Input: 2D array *
253 // Return: integer *
254 //*****
255 int argument_to_int(char *arg[])
256 {
257
258     return result;
259 }
260
261 //*****
262 // Function Name: address_to_long *
```

```

263 // Description: transform passed address into ldecimal value      *
264 // Input: 1D array                                                *
265 // Return: long long int                                          *
266 //*****
267 long long address_to_long(char addr[])
268 {
269
270     return result;
271 }
272
273 //*****
274 // Function Name: get_tag                                          *
275 // Description: get the tag from address into decimal value      *
276 // Input: 1D array                                                *
277 // Return: long long int                                          *
278 //*****
279 long long get_tag(char addr[])
280 {
281
282     return addressDec;
283 }
284 #if 0
285 long long get_tag(char addr[], int addressLen)
286 {
287
288     return result;
289 }
290 #endif
291 //*****
292 // Function Name: get_index                                        *
293 // Description: get the index from address into decimal value    *
294 // Input: 1D array                                                *
295 // Return: int                                                    *
296 //*****
297 int get_index(char addr[])
298 {
299
300     return addressDec;
301 }
302
303
304 #if 0
305 int get_index(char addr[], int addressLen)//needs adjustments
306 {
307
308     return result;
309 }
310 #endif
311
312 //*****
313 // Function Name: get_LRU                                          *
314 // Description: get the LRU block                                  *
315 // Input: integer and 1D array                                    *
316 // Return: int                                                    *
317 //*****
318 int get_LRU(int indx,int lru[])
319 {
320
321     return result;
322 }//end of get_LRU
323 //end of file :)
324

```