/home/diana/Documents/Comp Org/Project/main.c

```c
#include <stdio.h>
#include <getopt.h>
#include <math.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <string.h>
#include "cache.h"

//#define DEBUG

// Global Variables
// Cache structures
struct Cache *iCache, *dCache, *l2Cache; // L1 I and D caches, L2 Unified cache

// Main memory parameters, taken from project description
int mem_sendaddr = 10;
int mem_ready = 30;
int mem_chunktime = 15;
int mem_chunksize = 8;

// Variables for pulling out values from address fields
unsigned long long address, cacheStartAddress, byteStartAddress, byteEndAddress, cacheEndAddress; // Read in new addresses
unsigned long long l1Tag, l2Tag, l2WritebackTag, l2WritebackIndex; // Address Tag
unsigned long long l1IndexField, l2IndexField; // Address Index
unsigned long long l1Byte, l2Byte; // Address Byte Field

// Calculate cost of the system
double l1Cost = 0, l2Cost = 0, mainMemoryCost = 0;

unsigned long long traceCounter = 0; // Track total number of traces received
unsigned long long readTime = 0, writeTime = 0, instructionTime = 0; // Track total number of traces received
unsigned long long int executionTime = 0; // Track total execution time
unsigned long long int misalignments = 0; // Track total misalignments
unsigned long long flushTime = 0, flushes = 0; // Track time to flush
int flushMax = 380000; // Flush cache every time this value is reached

// Multiple counters for troubleshooting
unsigned long long readCounter = 0; // Track number of read traces received
unsigned long long writeCounter = 0; // Track number of write traces received
unsigned long long instructionCounter = 0; // Track number of instruction traces received
unsigned long long l1HitCounter = 0;
unsigned long long l1MissCounter = 0;
unsigned long long l2HitCounter = 0;
unsigned long long l2MissCounter = 0;

// Helper Function Declarations
void setDebugStatus(int status);
void findFields();
void findNextFields();
double calculateCost(int level, int size, int associativity);
int scanCache(struct Cache* cache, unsigned long long targetTag, unsigned long long targetIndex, char op);
unsigned long long moveBlock(struct Cache* cache, unsigned long long targetTag, unsigned long long targetIndex, int isDirty);
struct Cache * initialize(int newCacheSize, int newBlockSize, int newAssociativity, int newMissTime, int newHitTime);
void printHelpStatement();
void flushCaches(struct Cache * iCache, struct Cache * dCache, struct Cache * l2Cache, int mainMemoryTime, int transferTime, int busWidth);
void printCacheStatus(struct Cache * cache, int * cacheTracker);
void printReport();
void freeCache(struct Cache * cache);
void writeBack(unsigned long long writeBackTag);

int main(int argc, char* argv[]) {
    //*************************  Local Variables
    // Variables to read in traces
    char opCode; // Track type of trace: R [Read], W[Write], or I[Instruction]
    int byteSize = 0; // Track number of bytes referenced by trace
    FILE * configFile; // File to read in configuration and trace files

    // Debugging flag
    int debug = 0;

    // Variables to calculate cost of accessing memory
    unsigned int l1_hit_time = 1, l1_miss_time = 1;
    unsigned int l2_hit_time = 5, l2_miss_time = 7;
    unsigned int l2_transfer_time = 5, l2_bus_width = 16; // L1 -> L2 access penalties

    // Default cache values
    // Use the fact that L1 Data and Instruction caches are always the same size
```

```
79    int l1size = 8192, l2size = 32768, l1assoc = 1, l2assoc = 1;
80
81 #ifndef DEBUG
82    // Check input arguments
83    if (argc < 2 || argc > 4) {
84       // Put in too many arguments, print help statement
85       printHelpStatement();
86       return -1;
87    }
88
89    if (argc >= 2) { // Configuration file was added
90       printf("Configuration: %s\n\n", argv[1]);
91
92       // Read in configFile
93       configFile = fopen(argv[1], "r");
94
95       int value = 0;
96       char parameter[16];
97       while (fscanf(configFile, "%s %d\n", parameter, &value) != EOF) {
98          if (!strcmp(parameter, "l1size")) {
99             l1size = value;
100         }
101         if (!strcmp(parameter, "l2size")) {
102            l2size = value;
103         }
104         if (!strcmp(parameter, "l1assoc")) {
105            l1assoc = value;
106         }
107         if (!strcmp(parameter, "l2assoc")) {
108            l2assoc = value;
109         }
110         if (!strcmp(parameter, "memchunksize")) {
111            mem_chunksize = value;
112         }
113      }
114   }
115
116   // Finished reading file --> Close it
117   fclose(configFile);
118
119   if (argc > 2) { // Flags were added
120      int i = 2;
121      for (i = 2; i < argc; i++) {
122         if (strcmp(argv[i], "-h") == 0) // Print help statement
123            printHelpStatement();
124         else if (strcmp(argv[i], "-d") == 0) // Start debugging statements
125            debug = 1;
126      }
127   }
128
129
130 #endif
131
132 #ifdef DEBUG
133    debug = 1;
134    configFile = fopen("default_8", "r");
135    FILE * inputFile = fopen("traces-short/tr6", "r");
136
137    int value = 0;
138    char parameter[16];
139    while (fscanf(configFile, "%s %d\n", parameter, &value) != EOF) {
140       if (!strcmp(parameter, "l1size")) {
141          l1size = value;
142       }
143       if (!strcmp(parameter, "l2size")) {
144          l2size = value;
145       }
146       if (!strcmp(parameter, "l1assoc")) {
147          l1assoc = value;
148       }
149       if (!strcmp(parameter, "l2assoc")) {
150          l2assoc = value;
151       }
152       if (!strcmp(parameter, "memchunksize")) {
153          mem_chunksize = value;
154       }
155    }
156
157    // Finished reading file --> Close it
158    fclose(configFile);
159 #endif
```

```
160
161   // Initiate caches based on configFile values
162   iCache = initialize(l1size, 32, l1assoc, l1_miss_time, l1_hit_time);
163   dCache = initialize(l1size, 32, l1assoc, l1_miss_time, l1_hit_time);
164   l2Cache = initialize(l2size, 64, l2assoc, l2_miss_time, l2_hit_time);
165
166   // Calculate cost of the system based on configuration sizes
167   l1Cost = calculateCost(1, l1size, l1assoc);
168   l2Cost = calculateCost(2, l2size, l2assoc);
169   if (mem_chunksize <= 16) {
170      mainMemoryCost = 50 + 25;
171   } else if (mem_chunksize == 32) {
172      mainMemoryCost = 50 + 125;
173   } else {
174      mainMemoryCost = 50 + 225;
175   }
176
177   // Main memory access penalty
178   unsigned int mainMemoryTime = mem_sendaddr + mem_ready + (mem_chunktime * 64 / mem_chunksize);
179
180   // Set debug status in cache.c
181   setDebugStatus(debug);
182
183   // ***************************** Start Reading in the Traces
184   unsigned long long writeOverTag; // returned value if a LRU was dirty
185   int hit; // Flag to tell if there was a hit in a cache
186
187   int tracker = 0;
188   // iCache tracker
189   int iCacheIndexTracker[iCache->lengthOfWay];
190   for (tracker = 0; tracker < iCache->lengthOfWay; tracker++) {
191      iCacheIndexTracker[tracker] = 0;
192   }
193
194   // dCache tracker
195   int dCacheIndexTracker[dCache->lengthOfWay];
196   for (tracker = 0; tracker < dCache->lengthOfWay; tracker++) {
197      dCacheIndexTracker[tracker] = 0;
198   }
199
200   // l2Cache tracker
201   int l2CacheIndexTracker[l2Cache->lengthOfWay];
202   for (tracker = 0; tracker < l2Cache->lengthOfWay; tracker++) {
203      l2CacheIndexTracker[tracker] = 0;
204   }
205
206 #ifndef DEBUG
207      while (scanf("%c %Lx %d\n", &opCode, &address, &byteSize) == 3) {
208 #endif
209
210         if (debug) {
211            printf("---------- Inputting Traces Now ----------\n");
212         }
213 #ifdef DEBUG
214         while (fscanf(inputFile, "%c %Lx %d\n", &opCode, &address, &byteSize) != EOF) {
215 #endif
216            traceCounter++; // Increment trace counter
217            int counted = 0; // Only count each trace once
218
219            byteStartAddress = address & (~3); // Find start address for desired word
220            byteEndAddress = byteStartAddress + 4; // Find the end address for desired word
221
222            findFields(); // Have pulled out Index, Byte, and Tag fields for L1 cache
223
224            int bytesLeft = byteSize;
225            if (debug) printf("\nTrace #%lld --- Address  = %llx, Byte Size = %d, Type = %c \n", traceCounter, address, byteSize, opCode);
226            unsigned long long traceTime = 0; // track the time this trace took
227
228            while (bytesLeft > 0) { // Go through and repeat for as many bytes as necessary
229
230               if (debug) {
231                  printf("\t Requested Cache Block Start/End address = %llx, \t%llx\n", cacheStartAddress, cacheEndAddress);
232                  printf("\t *** Byte Start/End Address = %llx, \t %llx *** \n", byteStartAddress, byteEndAddress);
233                  printf("\t *** Desired Byte Address = %llx *** \n", address);
234                  printf("\t *** L1: Index Field = %llx, Tag Field = %llx *** \n", l1IndexField, l1Tag);
235               }
236
237               if (opCode == 'I') {
238                  if (!counted) {
239                     instructionCounter++; // Increment instruction counter
240                     counted = 1;
```

```
241             }
242
243             // Track which indexes have been changed
244             iCacheIndexTracker[l1IndexField] = 1;
245             l2CacheIndexTracker[l2IndexField] = 1;
246
247             hit = scanCache(iCache, l1Tag, l1IndexField, opCode); // Check L1I cache
248             if (hit == 1) { // Found in L1I cache
249                 if (debug) printf("\t HIT: Found in L1, adding L1 hit time (+%d)\n", l1_hit_time);
250                 l1HitCounter++; //Increment hit counter
251                 traceTime += iCache->hitTime;
252             } else { // Not in L1 Cache, all tags were valid, LRU is dirty
253                 if (debug) {
254                     printf("\t MISS: Not found in L1, adding L1 miss time (+%d)\n", l1_miss_time);
255                 }
256
257                 l1MissCounter++;
258                 traceTime += iCache->missTime;
259
260                 // Find LRU to write over in L1
261                 writeOverTag = moveBlock(iCache, l1Tag, l1IndexField, 0); // Tag is now in L1, do accounting details
262
263
264                 if (writeOverTag) {
265                     if (debug) printf("\t Need to write over a dirty block\n");
266                     writeBack(writeOverTag); // Reconstruct fields for L2 write-back
267
268                     if (debug) printf("\t L2 Writeback Index = %llx, Tag = %llx\n", l2WritebackIndex, l2WritebackTag);
269
270                     if (scanCache(l2Cache, l2WritebackTag, l2WritebackIndex, 'W')) { // Be seen as a write
271                         if (debug) {
272                             printf("\t\t HIT: Found in L2, adding L2 hit time\n");
273                             printf("Adding L1->L2 transfer time (+%d)\n", l2_transfer_time * (iCache->blockSize / l2_bus_width));
274                         }
275                         traceTime += l2Cache->hitTime;
276                         traceTime += l2_transfer_time * (iCache->blockSize / l2_bus_width); // Move desired block from L1 -> L2
277                         iCache->instructionTime += l2_transfer_time * (iCache->blockSize / l2_bus_width); // Add to instruction time counter
278
279                     } else {
280                         printf("\t\t MISS: Not Found in L2, adding L2 miss time\n");
281                         traceTime += l2Cache->missTime;
282
283                         // Find LRU to write over in L2
284                         writeOverTag = moveBlock(l2Cache, l2WritebackTag, l2WritebackIndex, 1); // Mark as dirty
285
286                         if (writeOverTag) {
287                             printf("\t\t Need to write over a dirty block\n");
288                             printf("\t\t Adding L2-> memory time (+%d)\n", mainMemoryTime);
289                             traceTime += mainMemoryTime;
290                         }
291
292                         if (debug) printf("Bringing block into L2, adding memory -> L2 time (+%d)\n", mainMemoryTime);
293                         traceTime += mainMemoryTime;
294
295                         if (debug) printf("Adding L2 replay time (+%d)\n", l2Cache->hitTime);
296                         traceTime += l2Cache->hitTime;
297
298                         if (debug) printf("Adding L1->L2 transfer time (+%d)\n", l2_transfer_time * (iCache->blockSize / l2_bus_width));
299                         traceTime += l2_transfer_time * (iCache->blockSize / l2_bus_width); // Move desired block from L1 -> L2
300                         iCache->instructionTime += l2_transfer_time * (iCache->blockSize / l2_bus_width); // Add to instruction time counter
301                     }
302                 }
303
304                 // Scan L2
305                 if (debug) printf("Now scanning L2 for target tag\n"
306                         "\t *** L2: Index Field = %llx, Tag Field = %llx *** \n", l2IndexField, l2Tag);
307                 if (scanCache(l2Cache, l2Tag, l2IndexField, opCode)) { // Hit in L2
308                     if (debug)
309                         printf("\t HIT: Found in L2, adding L2 hit time (+%d)\n", l2_hit_time);
310                     l2HitCounter++; // Increment hit counter
311                     traceTime += l2Cache->hitTime;
312
313                 } else { // Miss in L2 Cache, read from memory
314                     if (debug) {
315                         printf("\t MISS: Not found in L2, adding L2 miss time (+%d)\n", l2_miss_time);
316                     }
317
318                     traceTime += l2Cache->missTime;
319
320                     // Find LRU to write over in L2
321                     writeOverTag = moveBlock(l2Cache, l2Tag, l2IndexField, 0); // Tag is now in L2
```

```
322
323                 if (writeOverTag) { // LRU in L2 was dirty
324                     if (debug) {
325                         printf("\t Kicking out dirty block in L2: Tag = %llx \n", writeOverTag);
326                         printf("\t Rewriting dirty block L2 -> main memory (+%d)\n", mainMemoryTime);
327                     }
328                     traceTime += mainMemoryTime; // Move dirty block from L2 -> memory
329                 }
330
331                 // Bring block into L2
332                 if (debug) {
333                     printf("\t Bringing block back to L2, adding main memory -> L2 time (+%d)\n", mainMemoryTime);
334                     printf("\t Adding L2 replay time (+%d)\n", l2Cache->hitTime);
335                 }
336                 traceTime += mainMemoryTime; // Move dirty block from L2 -> memory
337                 traceTime += l2Cache->hitTime;
338             }
339
340             // Bring into L1 cache
341             if (debug) {
342                 printf("\t Writing block back to L1...");
343                 printf("Adding L2->L1 transfer time (+%d)\n", l2_transfer_time * (iCache->blockSize / l2_bus_width));
344                 printf("\t Adding L1 replay time (+%d)\n", l1_hit_time);
345             }
346
347             // Time penalty
348             traceTime += l2_transfer_time * (iCache->blockSize / l2_bus_width); // Move desired block from L1 -> L2
349             iCache->instructionTime += l2_transfer_time * (iCache->blockSize / l2_bus_width); // Add to instruction time counter
350             traceTime += iCache->hitTime;
351         }
352     } else if (opCode == 'R') {
353         if (!counted) {
354             readCounter++; // increment read counter
355             counted = 1;
356         }
357
358         // Track which indexes have been changed
359         dCacheIndexTracker[l1IndexField] = 1;
360         l2CacheIndexTracker[l2IndexField] = 1;
361
362         hit = scanCache(dCache, l1Tag, l1IndexField, opCode); // Check L1I cache
363         if (hit == 1) { // Found in L1I cache
364             if (debug)
365                 printf("\t HIT: Found in L1, adding L1 hit time (+%d)\n", l1_hit_time);
366             l1HitCounter++; //Increment hit counter
367             traceTime += dCache->hitTime;
368             //
369         } else { // Not in L1 Cache, all tags were valid, LRU is dirty
370             if (debug) {
371                 printf("\t MISS: Not found in L1, adding L1 miss time (+%d)\n", l1_miss_time);
372             }
373
374             l1MissCounter++;
375             traceTime += dCache->missTime;
376
377             writeOverTag = moveBlock(dCache, l1Tag, l1IndexField, 0); // Tag is now in L1, do accounting details
378
379             if (writeOverTag) {
380                 if (debug) printf("\t Need to write over a dirty block\n");
381                 writeBack(writeOverTag); // Reconstruct fields for L2 write-back
382
383                 if (debug) printf("\t L2 Writeback Index = %llx, Tag = %llx\n", l2WritebackIndex, l2WritebackTag);
384
385                 if (scanCache(l2Cache, l2WritebackTag, l2WritebackIndex, 'W')) { // Be seen as a write
386                     if (debug) {
387                         printf("\t\t HIT: Found in L2, adding L2 hit time\n");
388                         printf("Adding L1->L2 transfer time (+%d)\n", l2_transfer_time * (dCache->blockSize / l2_bus_width));
389                     }
390                     traceTime += l2Cache->hitTime;
391                     traceTime += l2_transfer_time * (dCache->blockSize / l2_bus_width); // Move desired block from L1 -> L2
392                 } else {
393                     if (debug) printf("\t\t MISS: Not Found in L2, adding L2 miss time\n");
394                     traceTime += l2Cache->missTime;
395
396                     // Find LRU to write over in L2
397                     writeOverTag = moveBlock(l2Cache, l2WritebackTag, l2WritebackIndex, 1); // Mark as dirty
398
399                     if (writeOverTag) {
400                         if (debug) {
401                             printf("\t\t Need to write over a dirty block\n");
402                             printf("\t\t Adding L2-> memory time (+%d)\n", mainMemoryTime);
```

```
403                         }
404                         traceTime += mainMemoryTime;
405                     }
406 
407                     if (debug) printf("Bringing block into L2, adding memory -> L2 time (+%d)\n", mainMemoryTime);
408                     traceTime += mainMemoryTime;
409 
410                     if (debug) printf("Adding L2 replay time (+%d)\n", l2Cache->hitTime);
411                     traceTime += l2Cache->hitTime;
412 
413                     if (debug) printf("Adding L1->L2 transfer time (+%d)\n", l2_transfer_time * (iCache->blockSize / l2_bus_width));
414                     traceTime += l2_transfer_time * (iCache->blockSize / l2_bus_width); // Move desired block from L1 -> L2
415                 }
416             }
417 
418             // Scan L2
419             if (debug) printf("Now scanning L2 for target tag\n"
420                     "\t *** L2: Index Field = %llx, Tag Field = %llx *** \n", l2IndexField, l2Tag);
421             if (scanCache(l2Cache, l2Tag, l2IndexField, opCode)) { // Hit in L2
422                 if (debug)
423                     printf("\t HIT: Found in L2, adding L2 hit time (+%d)\n", l2_hit_time);
424                 l2HitCounter++; // Increment hit counter
425                 traceTime += l2Cache->hitTime;
426 
427             } else { // Miss in L2 Cache, read from memory
428                 if (debug) {
429                     printf("\t MISS: Not found in L2, adding L2 miss time (+%d)\n", l2_miss_time);
430                 }
431 
432                 traceTime += l2Cache->missTime;
433 
434                 // Find LRU to write over in L2
435                 writeOverTag = moveBlock(l2Cache, l2Tag, l2IndexField, 0); // Tag is now in L2
436 
437                 if (writeOverTag) { // LRU in L2 was dirty
438                     if (debug) {
439                         printf("\t Kicking out dirty block in L2: Tag = %llx \n", writeOverTag);
440                         printf("\t Rewriting dirty block L2 -> main memory (+%d)\n", mainMemoryTime);
441                     }
442                     traceTime += mainMemoryTime; // Move dirty block from L2 -> memory
443                 }
444 
445                 // Bring block into L2
446                 if (debug) {
447                     printf("\t Bringing block back to L2, adding main memory -> L2 time (+%d)\n", mainMemoryTime);
448                     printf("\t Adding L2 replay time (+%d)\n", l2Cache->hitTime);
449                 }
450                 traceTime += mainMemoryTime; // Move dirty block from L2 -> memory
451                 traceTime += l2Cache->hitTime;
452             }
453 
454             // Bring into L1 cache
455             if (debug) {
456                 printf("\t Writing block back to L1...");
457                 printf("Adding L2->L1 transfer time (+%d)\n", l2_transfer_time * (dCache->blockSize / l2_bus_width));
458                 printf("\t Adding L1 replay time (+%d)\n", l1_hit_time);
459             }
460 
461             // Time penalty
462             traceTime += l2_transfer_time * (dCache->blockSize / l2_bus_width); // Move desired block from L1 -> L2
463             traceTime += dCache->hitTime;
464         }
465     } else if (opCode == 'W') {
466         if (!counted) {
467             writeCounter++; // Increment write counter
468             counted = 1;
469         }
470 
471         // Track which indexes have been accessed
472         dCacheIndexTracker[l1IndexField] = 1;
473         l2CacheIndexTracker[l2IndexField] = 1;
474 
475         hit = scanCache(dCache, l1Tag, l1IndexField, opCode); // Check L1I cache
476         if (hit == 1) { // Found in L1I cache
477             if (debug)
478                 printf("\t HIT: Found in L1, adding L1 hit time (+%d)\n", l1_hit_time);
479             l1HitCounter++; //Increment hit counter
480             traceTime += dCache->hitTime;
481             //
482         } else { // Not in L1 Cache, all tags were valid, LRU is dirty
483             if (debug) {
```

```
484                 printf("\t MISS: Not found in L1, adding L1 miss time (+%d)\n", l1_miss_time);
485             }
486
487             l1MissCounter++;
488             traceTime += dCache->missTime;
489
490             // Find LRU to write over in L1
491             writeOverTag = moveBlock(dCache, l1Tag, l1IndexField, 1); // Tag is now in L1, do accounting details
492
493             if (writeOverTag) {
494                 if (debug) printf("\t Need to write over a dirty block\n");
495                 writeBack(writeOverTag); // Reconstruct fields for L2 write-back
496
497                 if (debug) printf("\t L2 Writeback Index = %llx, Tag = %llx\n", l2WritebackIndex, l2WritebackTag);
498
499                 if (scanCache(l2Cache, l2WritebackTag, l2WritebackIndex, 'W')) { // Be seen as a write
500                     if (debug) {
501                         printf("\t\t HIT: Found in L2, adding L2 hit time\n");
502                         printf("Adding L1->L2 transfer time (+%d)\n", l2_transfer_time * (iCache->blockSize / l2_bus_width));
503                     }
504                     traceTime += l2Cache->hitTime;
505                     traceTime += l2_transfer_time * (dCache->blockSize / l2_bus_width); // Move desired block from L1 -> L2
506                 } else {
507                     if (debug) printf("\t\t MISS: Not Found in L2, adding L2 miss time\n");
508                     traceTime += l2Cache->missTime;
509
510                     // Find LRU to write over in L2
511                     writeOverTag = moveBlock(l2Cache, l2WritebackTag, l2WritebackIndex, 1); // Mark as dirty
512
513                     if (writeOverTag) {
514                         if (debug) {
515                             printf("\t\t Need to write over a dirty block\n");
516                             printf("\t\t Adding L2-> memory time (+%d)\n", mainMemoryTime);
517                         }
518                         traceTime += mainMemoryTime;
519                     }
520
521                     if (debug) printf("Bringing block into L2, adding memory -> L2 time (+%d)\n", mainMemoryTime);
522                     traceTime += mainMemoryTime;
523
524                     if (debug) printf("Adding L2 replay time (+%d)\n", l2Cache->hitTime);
525                     traceTime += l2Cache->hitTime;
526
527                     if (debug) printf("Adding L1->L2 transfer time (+%d)\n", l2_transfer_time * (dCache->blockSize / l2_bus_width));
528                     traceTime += l2_transfer_time * (iCache->blockSize / l2_bus_width); // Move desired block from L1 -> L2
529                 }
530             }
531
532             // Scan L2
533             if (debug) printf("Now scanning L2 for target tag\n"
534                     "\t *** L2: Index Field = %llx, Tag Field = %llx *** \n", l2IndexField, l2Tag);
535             if (scanCache(l2Cache, l2Tag, l2IndexField, 'R')) { // Hit in L2, not dirty there
536                 if (debug)
537                     printf("\t HIT: Found in L2, adding L2 hit time (+%d)\n", l2_hit_time);
538                 l2HitCounter++; // Increment hit counter
539                 traceTime += l2Cache->hitTime;
540
541             } else { // Miss in L2 Cache, read from memory
542                 if (debug) {
543                     printf("\t MISS: Not found in L2, adding L2 miss time (+%d)\n", l2_miss_time);
544                 }
545
546                 traceTime += l2Cache->missTime;
547
548                 // Find LRU to write over in L2
549                 writeOverTag = moveBlock(l2Cache, l2Tag, l2IndexField, 0); // Tag is now in L2
550
551                 if (writeOverTag) { // LRU in L2 was dirty
552                     if (debug) {
553                         printf("\t Kicking out dirty block in L2: Tag = %llx \n", writeOverTag);
554                         printf("\t Rewriting dirty block L2 -> main memory (+%d)\n", mainMemoryTime);
555                     }
556                     traceTime += mainMemoryTime; // Move dirty block from L2 -> memory
557                 }
558
559                 // Bring block into L2
560                 if (debug) {
561                     printf("\t Bringing block back to L2, adding main memory -> L2 time (+%d)\n", mainMemoryTime);
562                     printf("\t Adding L2 replay time (+%d)\n", l2Cache->hitTime);
563                 }
564                 traceTime += mainMemoryTime; // Move dirty block from L2 -> memory
```

```
565                     traceTime += l2Cache->hitTime;
566                 }
567 
568                 // Bring into L1 cache
569                 if (debug) {
570                     printf("\t Writing block back to L1...");
571                     printf("Adding L2->L1 transfer time (+%d)\n", l2_transfer_time * (dCache->blockSize / l2_bus_width));
572                     printf("\t Adding L1 replay time (+%d)\n", l1_hit_time);
573                 }
574 
575                 // Time penalty
576                 traceTime += l2_transfer_time * (dCache->blockSize / l2_bus_width); // Move desired block from L1 -> L2
577                 traceTime += dCache->hitTime;
578             }
579         } // Finished reading trace
580 
581         // Sent 4 bytes to the processor
582 
583         int transferred = 0;
584         //              moved = 0;
585         // Find the next fields if you need to move on to the next block
586         while ((byteStartAddress < byteEndAddress) && (bytesLeft > 0)) {
587             if (byteStartAddress >= address) { // Moved desired bytes
588                 //                  byteAddress++;
589                 transferred++;
590                 bytesLeft--;
591             }
592             byteStartAddress++;
593         }
594 
595         if (debug) printf("%d bytes transferred.\n", transferred);
596 
597         if (bytesLeft > 0) { // Reached end of cache block
598             misalignments++; // Have to do multiple memory access
599             if (byteEndAddress != cacheEndAddress) { // We can go on to another word in the same block
600                 //                  byteStartAddress++;
601                 byteEndAddress = byteStartAddress + 4;
602                 if (debug) {
603                     printf("%d bytes left. Still in the same block.\n\n", bytesLeft);
604                     printf("\t *** Byte Start/End Address = %llx, \t%llx *** \n", byteStartAddress, byteEndAddress);
605                 }
606             } else {
607                 // Move on to the next block
608                 //                  byteStartAddress; // Potential overflow error HERE
609                 byteEndAddress = byteStartAddress + 4;
610                 findFields();
611 
612                 if (debug) {
613                     printf("\t Need to access the next block! \n");
614                     printf("\t Requested Cache Block Start/End address = %llx, \t%llx\n", cacheStartAddress, cacheEndAddress);
615                     printf("\t *** Byte Start/End Address = %llx, \t%llx *** \n", byteStartAddress, byteEndAddress);
616                     printf("\t *** L1: Index Field = %llx, Tag Field = %llx *** \n\n", l1IndexField, l1Tag);
617                 }
618             }
619         } else {
620             if (debug) printf("%d bytes left.\n", bytesLeft);
621         }
622     }
623 
624     // Finished reading a trace, be ready to read in the next trace
625     if (traceCounter % flushMax == 0) { // Reached the point where we need to flush the caches
626         if (debug) {
627             printf("Time to flush the caches! \n");
628         }
629         flushes++; // increment flush counter
630         flushCaches(iCache, dCache, l2Cache, mainMemoryTime, l2_transfer_time, l2_bus_width);
631 
632         // Reset trackers
633         for (tracker = 0; tracker < iCache->lengthOfWay; tracker++)
634             iCacheIndexTracker[tracker] = 0;
635 
636         // dCache tracker
637         for (tracker = 0; tracker < dCache->lengthOfWay; tracker++)
638             dCacheIndexTracker[tracker] = 0;
639 
640         // l2Cache tracker
641         for (tracker = 0; tracker < l2Cache->lengthOfWay; tracker++)
642             l2CacheIndexTracker[tracker] = 0;
643     }
644 
645     executionTime += traceTime; // Add trace time to the total execution time
```

```
646         if (debug) {
647             printf("\t ---> Total trace time: %llu\n", traceTime);
648             printf("\t ---> Total execution time: %llu\n", executionTime);
649             printf("-------------------- End of Trace --------------------\n");
650         }
651 
652         switch (opCode) {
653             case 'R':
654                 readTime += traceTime;
655                 if (debug) printf("ReadTime= %llu\n", readTime);
656                 break;
657             case 'W':
658                 writeTime += traceTime;
659                 if (debug) printf("WriteTime= %llu\n", writeTime);
660                 break;
661             case 'I':
662                 instructionTime += traceTime;
663                 if (debug) printf("InstTime= %llu\n", instructionTime);
664                 break;
665         }
666 
667 #ifdef DEBUG
668         }
669         fclose(inputFile);
670         flushCaches(iCache, dCache, l2Cache, mainMemoryTime, l2_transfer_time, l2_bus_width);
671         flushes++;
672 #endif
673 
674 #ifndef DEBUG
675     }
676 #endif
677 
678     printReport();
679 
680     if (debug) {
681         printf("-----------------------------------------------------\n");
682         printf("Cache Final Contents: \n\n"
683                "Memory Level: L1 Instruction Cache\n");
684         printCacheStatus(iCache, iCacheIndexTracker);
685 
686         printf("\nMemory Level: L1 Data Cache\n");
687         printCacheStatus(dCache, dCacheIndexTracker);
688 
689         printf("\nMemory Level: L2 Unified Cache \n");
690         printCacheStatus(l2Cache, l2CacheIndexTracker);
691     }
692 
693     // Free all allocated memory
694     freeCache(iCache);
695     freeCache(dCache);
696     freeCache(l2Cache);
697 
698     return 0;
699 }
700 
701 double calculateCost(int level, int size, int associativity) {
702     double result;
703     int i = 0;
704     if (level == 1) { // L1 Cache
705         result = 100 * (size / 4096); // L1 cache memory costs $100 for each 4KB
706         // Add an additional $100 for each doubling in associativity beyond direct-mapped
707         for (i = 0; i < size / 4096; i++) {
708             result += 100 * log(associativity) / log(2);
709         }
710     } else if (level == 2) { // L2 Cache
711         if (size < 65536) // Check if L2 memory is < 64Kb (still need to pay for full cost)
712             result = 50;
713         else
714             result = 100; // L2 cache memory costs $50 per 64KB
715         // Add an additional $50 for each doubling in associativity
716         if (associativity > 1)
717             result += 50 * log(associativity) / log(2);
718     }
719     return result;
720 }
721 
722 void findFields() {
723     // Pull out the byte field from the address
724     // Fill space with 1's, then shift over by byeOffset
725     l1Byte = (~0) << iCache->byteOffsetSize;
726     l2Byte = (~0) << l2Cache->byteOffsetSize;
```

```
727
728     cacheStartAddress = l1Byte;
729     cacheStartAddress = cacheStartAddress & byteStartAddress; // Track start of the block
730     cacheEndAddress = cacheStartAddress + iCache->blockSize; // Track end of the block
731
732     l1Byte = ~l1Byte; // Switch all 1's to 0's
733     l1Byte = l1Byte & byteStartAddress; // Mask out all fields but the incoming byte field
734
735     l2Byte = ~l2Byte; // Switch all 1's to 0's
736     l2Byte = l2Byte & byteStartAddress; // Mask out all fields but the incoming byte field
737
738     // Pull out the index field from the address
739     // Fill space with 1's, then shift over by byeOffset
740     l1IndexField = (~0) << (iCache->indexFieldSize + iCache->byteOffsetSize);
741     l2IndexField = (~0) << (l2Cache->indexFieldSize + l2Cache->byteOffsetSize);
742     l1Tag = l1IndexField; // Save correct placing of 1's for finding the tag field next
743     l1IndexField = ~l1IndexField; // Switch all 1's to 0's
744     l1IndexField = l1IndexField & byteStartAddress; // Mask out all fields but the incoming byte/index field
745
746     l2Tag = l2IndexField; // Save correct placing of 1's for finding the tag field next
747     l2IndexField = ~l2IndexField; // Switch all 1's to 0's
748     l2IndexField = l2IndexField & byteStartAddress; // Mask out all fields but the incoming byte/index field
749
750     // Shift out the byte field
751     l1IndexField = l1IndexField >> iCache->byteOffsetSize;
752     l2IndexField = l2IndexField >> l2Cache->byteOffsetSize;
753
754     // Pull out the tag field
755     l1Tag = l1Tag & byteStartAddress;
756     l2Tag = l2Tag & byteStartAddress;
757
758     // Shift out other fields
759     l1Tag = l1Tag >> (iCache->indexFieldSize + iCache->byteOffsetSize);
760     l2Tag = l2Tag >> (l2Cache->indexFieldSize + l2Cache->byteOffsetSize);
761
762     return;
763 }
764
765 void printHelpStatement() {
766     printf("Memory Simulation Project, ECEN 4593 \n"
767             "Usage: cat <traces> | ./main <config_file> -FLAGS\n"
768             "Configuration File: Used to set up caches. \n\n"
769             "Flag Options:\n"
770             "\t -h: Print up usage statement\n"
771             "\t -d: Turn on debugging statements");
772 }
773
774 void printReport() {
775     printf("-----------------------------------------------------\n");
776     printf("\t\t\t\t Simulation Results\n");
777     printf("-----------------------------------------------------\n");
778     printf("Memory System:\n");
779     printf("\t D Cache: Size = %d, \tAssociativity = %d, \tBlock size = %d\n", dCache->cacheSize, dCache->associativity, dCache->blockSize);
780     printf("\t I Cache: Size = %d, \tAssociativity = %d, \tBlock size = %d\n", iCache->cacheSize, iCache->associativity, iCache->blockSize);
781     printf("\t L2 Unified Cache: Size = %d, \tAssociativity = %d, \tBlock size = %d\n", l2Cache->cacheSize, l2Cache->associativity, l2Cache->blockSize);
782     printf("\t Main Memory: Ready time = %d, \tChunksize = %d, \tChunktime = %d,\n\n", mem_ready, mem_chunksize, mem_chunktime);
783
784     printf("Execution Time = %llu, \tTotal Traces Read = %llu\n", executionTime, traceCounter);
785
786     flushTime = iCache->flushTime + dCache->flushTime + l2Cache->flushTime;
787     printf("Flush time = %llu\n", flushTime);
788     printf("Total time (Execution Time + Flush Time) = %llu\n\n", flushTime + executionTime);
789
790     printf("Number of Reference Types: \t[Percentage] \n"
791             "\tReads \t= \t%llu\t\t\t[%.1f%%]\n"
792             "\tWrites \t= \t%llu\t\t\t[%.1f%%]\n"
793             "\tInst. \t= \t%llu\t\t\t[%.1f%%]\n"
794             "\tTotal \t= \t%llu\n\n",
795             readCounter, (double) readCounter * 100 / traceCounter,
796             writeCounter, (double) writeCounter * 100 / traceCounter,
797             instructionCounter, (double) instructionCounter * 100 / traceCounter,
798             readCounter + writeCounter + instructionCounter);
799
800     printf("Total Cycles for Activities: \t[Percentage] \n"
801             "\tReads \t= \t%llu\t\t\t[%.1f%%]\n"
802             "\tWrites \t= \t%llu\t\t\t[%.1f%%]\n"
803             "\tInst. \t= \t%llu\t\t\t[%.1f%%]\n"
804             "\tTotal \t= \t%llu\n\n",
805             readTime, (double) readTime * 100 / executionTime,
806             writeTime, (double) writeTime * 100 / executionTime,
807             instructionTime, (double) instructionTime * 100 / executionTime,
```

```
808         readTime + writeTime + instructionTime);
809
810     double readCycles = 0, writeCycles = 0, insCycles = 0;
811     if (readCounter)
812         readCycles = ((double) readTime) / ((double) readCounter);
813     if (writeCounter)
814         writeCycles = ((double) writeTime) / ((double) writeCounter);
815     if (instructionCounter)
816         insCycles = ((double) (readTime + writeTime + instructionTime)) / ((double) instructionCounter);
817
818     printf("Average Cycles for Activities: \n"
819         "\tReads = %.1f, \tWrites = %.1f, \tInstructions = %.1f\n\n",
820         readCycles, writeCycles, insCycles);
821
822     unsigned long long idealTime = readCounter + writeCounter + (2 * instructionCounter);
823
824     printf("Ideal: Execution Time = %llu; \tCPI = %.1f\n", idealTime, ((double) idealTime) / ((double) instructionCounter));
825     printf("Ideal Misaligned Execution Time = %llu; \tCPI = %.1f\n\n", idealTime + misalignments, ((double) idealTime + (double) misalignments) / ((double) instruct
826
827     unsigned long long totalRequests = iCache->hits + iCache->misses;
828     printf("Memory Level: L1 Instruction Cache\n"
829         "\tHit Count = %llu, \t Miss Count = %llu\n", iCache->hits, iCache->misses);
830     printf("\tTotal Requests = %llu\n", totalRequests);
831     printf("\tHit Rate: %.1f%%, \t Miss Rate = %.1f%%\n", (double) iCache->hits * 100 / totalRequests, (double) iCache->misses * 100 / totalRequests);
832     printf("\tKickouts = %llu, \t Dirty-kickouts = %llu, \t Transfers = %llu\n",
833         iCache->kickouts, iCache->dirtyKickouts, iCache->transfers);
834     printf("\tFlush Kickouts = %llu\n\n", iCache->flushKickouts);
835
836     totalRequests = dCache->hits + dCache->misses;
837     printf("Memory Level: L1 Data Cache\n"
838         "\tHit Count = %llu, \t Miss Count = %llu\n", dCache->hits, dCache->misses);
839     printf("\tTotal Requests = %llu\n", totalRequests);
840     printf("\tHit Rate: %.1f%%, \t Miss Rate = %.1f%%\n", (double) dCache->hits * 100 / totalRequests, (double) dCache->misses * 100 / totalRequests);
841     printf("\tKickouts = %llu, \t Dirty-kickouts = %llu, \t Transfers = %llu\n",
842         dCache->kickouts, dCache->dirtyKickouts, dCache->transfers);
843     printf("\tFlush Kickouts = %llu\n\n", dCache->flushKickouts);
844
845     totalRequests = l2Cache->hits + l2Cache->misses;
846     printf("Memory Level: L2 Cache\n"
847         "\tHit Count = %llu, \t Miss Count = %llu\n", l2Cache->hits, l2Cache->misses);
848     printf("\tTotal Requests = %llu\n", totalRequests);
849     printf("\tHit Rate: %.1f%%, \t Miss Rate = %.1f%%\n", (double) l2Cache->hits * 100 / totalRequests, (double) l2Cache->misses * 100 / totalRequests);
850     printf("\tKickouts = %llu, \t Dirty-kickouts = %llu, \t Transfers = %llu\n",
851         l2Cache->kickouts, l2Cache->dirtyKickouts, l2Cache->transfers);
852     printf("\tFlush Kickouts = %llu\n\n", l2Cache->flushKickouts);
853
854     printf("\nSystem Costs:\n");
855     printf("\t L1 Cache Costs (Instruction Cache $%d + Data Cache $%d) = $%d\n", (int) l1Cost, (int) l1Cost, (int) l1Cost * 2);
856     printf("\t L2 Cache Costs: $%d \n", (int) l2Cost);
857     printf("\t Main Memory Costs: $%d \n", (int) mainMemoryCost);
858     printf("\t Total Memory Cost: $%d \n\n", (int) (mainMemoryCost + l1Cost * 2 + l2Cost));
859
860     printf("Flushes = %llu, Invalidates = %llu\n\n", flushes, iCache->invalidates + dCache->invalidates + l2Cache->invalidates);
861
862     printf("-------------------------------------------------------\n");
863
864 }
865
866 void writeBack(unsigned long long writeBackTag) {
867     unsigned long long addressTemp = 0; // Fill with 1s
868     addressTemp += (l1IndexField << iCache->byteOffsetSize); // Put index field back in the right spot
869     addressTemp += (writeBackTag << (iCache->byteOffsetSize + iCache->indexFieldSize)); // Put tag field back in the right spot
870
871     // Pull out L2 Writeback index/tag
872     l2WritebackIndex = ~0; // Fill with ones
873     l2WritebackIndex = l2WritebackIndex << (l2Cache->byteOffsetSize + l2Cache->indexFieldSize);
874
875     l2WritebackTag = l2WritebackIndex; // 1's only in the tag field
876
877     l2WritebackIndex = ~l2WritebackIndex; // 1's only in the byte/index field
878
879     l2WritebackIndex = l2WritebackIndex & addressTemp; // Pulled out index
880     l2WritebackTag = l2WritebackTag & addressTemp; // Pulled out index
881
882     l2WritebackIndex = l2WritebackIndex >> l2Cache->byteOffsetSize; // shift over
883     l2WritebackTag = l2WritebackTag >> (l2Cache->byteOffsetSize + l2Cache->indexFieldSize);
884
885     return;
886 }
887
```