Phil Nevins
Daniel Anishchenko
Cody Reid
Kenneth Sutter
3/15/24

Group 1 485/585 Project Report Winter 2024

Internal Design Documentation

High level algorithm:

1. Prompt user for trace file and read in trace file
   fopen("example.txt", "r");
2. Convert trace file data to binary
   a. Going one line at a time through a loop (for or while loop)
   b. Isolate N from binary address
   c. We need to back/front fill 0s to match a 32 bit operation
   d. Store N in array, store address in array
3. Look at the N (i.e., read, write, instruction, etc) from the first
   a.
4. Jump to the appropriate function defined by N (switch case?)
   a. 0 read data request to L1 data cache
   b. 1 write data request to L1 data cache
   c. 2 instruction fetch (a read request to L1 instruction cache)
   d. 3 invalidate command from L2
   e. 4 data request from L2 (in response to snoop)
   f. 8 clear the cache and reset all state (and statistics)
   g. 9 print contents and state of the cache (allow subsequent trace activity)
5. Define the functions from step 4
6. Loop until trace file is EOF
7. Exit condition to pause

Assumptions

32-bit processor so we have 32 address bits

L1 instruction cache is four-way set associative and consists of 16K sets and 64-byte lines.

L1 data cache is eight-way set associative and consists of 16K sets of 64-byte lines.

The L1 data cache is write-back and uses write allocate functionality. It will write-back except for on the first write to a specific line which is then write-through.

Both caches employ LRU replacement policy and are backed by a shared L2 cache.

The cache hierarchy employs inclusivity.

64-byte lines = 26 = 6 address bits for Byte Select bits (bit [5:0])
16K sets cache = 2^{14} = 14 address bits for index (bit[19:6])
Tag bits = 32 – 6 – 14 = 12 bits (bit[31:20])

Four-Way: Each set in the cache is associatively mapped with four cache lines.

Eight-Way: Each set in the cache is associatively mapped with eight cache lines.

Each set can store data from up to four or eight different memory locations

Mode 0: executes program and displays a summary of the usage statistics and has a response to 9s in the trace file and nothing else.

Mode 1: does everything mode 0 does and will display the communication between cache L1 and L2 caches.

Policies
MESI
LRU replacement

Cache Commands
0 read data request to L1 data cache
1 write data request to L1 data cache
2 instruction fetch (a read request to L1 instruction cache)
3 invalidate command from L2
4 data request from L2 (in response to snoop)
8 clear the cache and reset all state (and statistics)
9 print contents and state of the cache (allow subsequent trace activity)

memory references do not span cache line boundaries

Cache hit ratio = (cache hits) / [(cache hits) + (cache misses)]

Outermost cache will do the snooping.

Design Decisions
We decided to create this cache simulation in the C programming language. This is due to our team being comfortable with the C programming language.

C Programming libraries to use
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <stdbool.h>

Started with a one-set associative cache, and then expanded to four and eight way.

Used struts for each cache. Using arrays and variables for the members of the struct to hold each bit needed for the data and instructions.

Also used structs to hold the performance statistics of each cache. Each member of the struct holds a different statistic.

Use case statements to convert hex to binary

Separate functions for each command.

Utilize invalidate function for snoop request

Use counters for each statistic that we can update for each command executed.

Test plan:
Our test plan is to break the project into multiple smaller parts. We would like to test each part as we go to ensure we have less debugging time overall. We plan to write test trace files to load in for each command. These test trace files will allow us to test each command and function individually. This makes it easier to isolate bugs in each part of the code. We will also need to test input trace file edge cases. In the modules section of this report we have included our test trace files.

Trace files to test:
Read print test
Write print test
Fetch print test
All commands test
Snoop print test
Read clear print test
Invalidate test

Source Code

```
/**
 * @file cache_simulator.c
 * @brief Cache Simulator for L1 Instruction and Data Cache
 *
 * This program simulates an L1 instruction cache and an L1 data cache as
part of a two-level cache hierarchy,
 * including a shared L2 cache. The L1 instruction cache is four-way set
associative with 16K sets and 64-byte lines.
 * The L1 data cache is eight-way set associative with 16K sets of 64-byte
lines. The data cache utilizes a write-back
 * policy with write allocate and performs write-back for all but the
first write to a line, which is write-through.
 * Both caches implement an LRU (Least Recently Used) replacement policy
and maintain inclusivity with the shared L2 cache.
 *
 * The simulator processes a trace file containing memory access
instructions to model the behavior of the caches
 * under various access patterns. It tracks and reports cache hits,
misses, and the resultant state transitions
 * within the MESI (Modified, Exclusive, Shared, Invalid) protocol. This
detailed simulation helps in understanding
 * the performance and efficiency of the cache hierarchy in a computing
system.
 *
 * @authors
 * - Phil N
 * - Cody R
```

```
 * - Ken S
 * - Danial A
 *
 * @date March 10, 2024
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <stdbool.h>

#define MAX_LINES 100
#define MAX_LENGTH 100
#define INDEX 14
#define TAG 12
#define OFFSET 6

int SETS = 16384;
#define WAYS_PER_SET 8
#define WAYS_DATA_CACHE 8
#define WAYS_INSTRUCTION_CACHE 4

struct CacheData {
    char MESI;
    char offset[OFFSET + 1];
    char index[INDEX + 1];
    char tag[TAG + 1];
    int LRU;
    int Full;
};

struct InstructionData {
    char MESI;
    char offset[OFFSET + 1];
    char index[INDEX + 1];
    char tag[TAG + 1];
    int LRU;
    int Full;
};

struct stats {
    double hits;
    double misses;
    double reads;
    double writes;
};

struct InstructionStats {
    unsigned int hits;
    double misses;
    double reads;
    double writes;
};
```

```
struct temp {
    char offset[OFFSET + 1];
    char index[INDEX + 1];
    char tag[TAG + 1];
};

struct CacheData*** DataCache;
struct stats* stats;
struct InstructionStats* InstructionStats;
struct InstructionData*** InstructionCache;
struct temp* temp;
int CurrentMode;
int writeSets[16384];
int writeWays[8];
int InstructionWriteSets[16384];
int InstructionWriteWays[4];
int writeCounter;
int InstructionWriteCounter;
char bin[33];
char N_Value;
int ModeInput;

/**
 * @brief Converts a hexadecimal string from a line into its binary
representation.
 *
 * @param Lines A 2D array containing lines read from the trace file.
 * @param LineCount Total number of lines read from the trace file.
 * @param count The current line number being processed.
 * @param bin A string to store the converted binary representation.
 */
void hex2bin(char Lines[MAX_LINES][MAX_LENGTH], int LineCount, int count,
char bin[33]);

/**
 * @brief Parses a binary string into separate fields for tag, index, and
offset.
 *
 * @param bin A binary string representing an address.
 * @param temp A pointer to a struct temp where the parsed bits will be
stored.
 */
void parseBitString(char bin[33], struct temp* temp);

/**
 * @brief Searches the cache for a given tag, index, and offset
combination.
 *
 * @param target A pointer to a struct temp containing the tag, index, and
offset to search for.
 * @return int The way in which the address was found, or -1 if it was not
found.
 */
```

```
int searchCache(struct temp* target);

/**
 * @brief Writes a block into the cache, updating its state based on MESI
protocol.
 *
 * @param source A pointer to a struct temp containing the tag, index, and
offset to write.
 */
void writeCache(struct temp* source);

/**
 * @brief Finds the index of the cache line with the lowest LRU value.
 *
 * @param size The size of the cache set.
 * @return int The index of the cache line with the lowest LRU value.
 */
int findLowestLRUIndex(int size);

/**
 * @brief Updates the LRU values for cache lines in a set after a cache
line is accessed.
 *
 * @param accessedSet The set number of the accessed cache line.
 * @param accessedIndex The index within the set of the accessed cache
line.
 */
void updateLRU(int accessedSet, int accessedIndex);

/**
 * @brief Prints statistics of cache operations including hits, misses,
reads, and writes.
 */
void printStats(void);

/**
 * @brief Prints the current state of the data and instruction caches.
 */
void printCurrentCacheState(void);

/**
 * @brief Clears all entries in the cache and resets statistics.
 */
void clearCacheAndStats(void);

/**
 * @brief Converts an integer to a binary string representing a cache
index.
 *
 * @param n The integer to convert.
 * @param binaryString A string to store the binary representation.
 */
void intToBinaryString(int n, char* binaryString);
```

```
/**
 * @brief Main function of the cache simulation program.
 *
 * Initializes memory, reads trace file, and executes cache operations
based on the trace. Cleans up resources upon completion or error.
 *
 * @return int Program exit status.
 */
int main() {
    temp = malloc(sizeof(struct temp));
    if (temp == NULL) {
        fprintf(stderr, "Error: Memory allocation for temp failed.\n");
        exit(EXIT_FAILURE);
    }

    stats = malloc(sizeof(struct stats));
    if (stats == NULL) {
        fprintf(stderr, "Error: Memory allocation for stats failed.\n");
        exit(EXIT_FAILURE);
    }

    InstructionStats = malloc(sizeof(struct InstructionStats));
    if (InstructionStats == NULL) {
        fprintf(stderr, "Error: Memory allocation for InstructionStats
failed.\n");
        exit(EXIT_FAILURE);
    }

    stats->hits = 0;
    stats->misses = 0;
    stats->reads = 0;
    stats->writes = 0;

    InstructionStats->hits = 0;
    InstructionStats->misses = 0;
    InstructionStats->reads = 0;
    InstructionStats->writes = 0;

    DataCache = malloc(SETS * sizeof(struct CacheData**));
    for (int i = 0; i < SETS; i++) {
        DataCache[i] = malloc(WAYS_DATA_CACHE * sizeof(struct
CacheData*));
        for (int j = 0; j < WAYS_DATA_CACHE; j++) {
            DataCache[i][j] = malloc(sizeof(struct CacheData));
            DataCache[i][j]->MESI = 'I';
            DataCache[i][j]->Full = 0;
            strcpy(DataCache[i][j]->offset, "000000");
            strcpy(DataCache[i][j]->tag, "0000000000000");
            intToBinaryString(i, DataCache[i][j]->index);
            DataCache[i][j]->LRU = 0;
        }
    }
```

```
    InstructionCache = malloc(SETS * sizeof(struct InstructionData**));
    for (int i = 0; i < SETS; i++) {
        InstructionCache[i] = malloc(WAYS_INSTRUCTION_CACHE *
sizeof(struct InstructionData*));
        for (int j = 0; j < WAYS_INSTRUCTION_CACHE; j++) {
            InstructionCache[i][j] = malloc(sizeof(struct
InstructionData));
            InstructionCache[i][j]->MESI = 'I';
            InstructionCache[i][j]->Full = 0;
            strcpy(InstructionCache[i][j]->offset, "000000");
            strcpy(InstructionCache[i][j]->tag, "0000000000000");
            intToBinaryString(i, InstructionCache[i][j]->index);
            InstructionCache[i][j]->LRU = 0;
        }
    }

        /**< Start of Simulation */
    printf("0 - Mode 0\n");
    printf("1 - Mode 1\n");
    printf("Enter the mode: ");
    scanf("%d", &ModeInput);
    if (ModeInput == 0 || ModeInput == 1) {
        switch (ModeInput) {
            case 0:
                CurrentMode = 0;
                break;
            case 1:
                CurrentMode = 1;
                break;
            default:
                printf("Invalid ModeInput selected.\n");
                break;
        }
    }

    FILE* TraceFile;
    char Lines[MAX_LINES][MAX_LENGTH];
    int LineCount = 0;
    char fileName[MAX_LENGTH];
    printf("\nEnter the name of the tracefile: ");
    scanf("%s", fileName);
    TraceFile = fopen(fileName, "r");
    if (TraceFile == NULL) {
        printf("Error... Issue opening file.\n");
        return 1;
    }
    while (fgets(Lines[LineCount], MAX_LENGTH, TraceFile) != NULL) {
        LineCount++;
        if (LineCount >= MAX_LINES) {
            printf("Error ... Maximum number of lines reached.\n");
            break;
        }
    }
```

```
fclose(TraceFile);

/**< Pull N Value from the tracefile */
for (int i = 0; i < LineCount; i++) {
    N_Value = Lines[i][0];

    switch (N_Value) {
        /**< Read */
        case '0':
            stats->reads++;
            hex2bin(Lines, LineCount, i, bin);
            parseBitString(bin, temp);
            if (searchCache(temp) > -1) {
                stats->hits++;
            } else {
                stats->misses++;
                if (ModeInput == 1)
                printf("Read from L2 < %s >\n", bin);
            }
            break;

        /**< Write */
        case '1':
            stats->writes++;

            if (ModeInput == 1)
            printf("Write to L2 < %s >\n", bin);

            hex2bin(Lines, LineCount, i, bin);
            parseBitString(bin, temp);
            if (searchCache(temp) < 0) {
                stats->misses++;
                if (ModeInput == 1) {
                printf("Read from L2 < %s >\n", bin);
                printf("Read For Ownership from L2 < %s >\n", bin);
                }
            } else {
                stats->hits++;
            }
            break;

        /**< Fetch */
    case '2':
        hex2bin(Lines, LineCount, i, bin);
        parseBitString(bin, temp);
        char *endptr2;
        long int set2 = strtol(temp->index, &endptr2, 2);
        int way2 = searchCache(temp); // way hit

            if (way2 > -1 && N_Value == '2') {

            InstructionStats->hits++;
            updateLRU(set2, way2);
            }
```

```
        if(way2 == -1 && N_Value == '2'){
            InstructionStats->misses++;
        }
break;

    /**< Invalidate */
case '3':
    hex2bin(Lines, LineCount, i, bin);
    parseBitString(bin, temp);
    char *endptr3;
    long int set3 = strtol(temp->index, &endptr3, 2);
    int way3 = searchCache(temp);
    if(way3 == -1){
        //printf("failure due to not finding temp in DataArrray");
        break;
    }
    DataCache[set3][way3]->MESI = 'I';
    break;

    /**< Snoop Request */
case '4':
    hex2bin(Lines, LineCount, i, bin);
    parseBitString(bin, temp);
    char *endptr4;
    long int set4 = strtol(temp->index, &endptr4, 2);
    int way4 = searchCache(temp);

    if(way4 == -1){
        //printf("failure due to not finding temp in DataArrray");
        break;
    }

    if(ModeInput == 1){
    printf("Return data to L2 < %s >\n", bin);
    }

    DataCache[set4][way4]->MESI = 'I';
    break;

    /**< Clear Cache and Stats */
    case '8':
        hex2bin(Lines, LineCount, i, bin);
        parseBitString(bin, temp);
        clearCacheAndStats();
        break;

    /**< Print Cache */
    case '9':
        hex2bin(Lines, LineCount, i, bin);
        parseBitString(bin, temp);
        printCurrentCacheState();
        break;
```

```
            default:
                printf("Unknown command, Exiting program. \n");
                exit(1);
                break;
        }
    }

    /**< Free Everything */
    for (int i = 0; i < SETS; i++) {
        for (int j = 0; j < WAYS_DATA_CACHE; j++) {
            free(DataCache[i][j]);
        }
        free(DataCache[i]);
    }
    free(DataCache);
    for (int i = 0; i < SETS; i++) {
        for (int j = 0; j < WAYS_INSTRUCTION_CACHE; j++) {
            free(InstructionCache[i][j]);
        }
        free(InstructionCache[i]);
    }
    free(InstructionCache);
    free(temp);
    free(stats);
    free(InstructionStats);

    /**< Exit Condition To Pause */
    int choice;
    do {
        printf("Enter 1 to exit: \n");
        scanf("%d", &choice);
        if (choice == 1) {
            printf("Exiting...\n");
            return 0;
        } else {
            printf("Invalid choice. Please enter 1.\n");
        }
    } while (choice != 1);
    return 0;
}

/******************************** Helper Functions
********************************/

void hex2bin(char Lines[MAX_LINES][MAX_LENGTH], int LineCount, int count,
char bin[33]){
    for(int i = 0; i < 33; i++){
        bin[i] = '\0';
    }

    int hexStartIndex = 2; // Start index for reading hexadecimal values

    // Loop through each hexadecimal value in the line
    for (int i = 0; i < 8; i++) {
```

```
        char hexValue = Lines[count][hexStartIndex + i]; // Get the
hexadecimal character
        char binary[5] = {'\0'}; // Buffer to hold binary representation
of the current hexadecimal character

        // Convert hexadecimal character to binary
        switch (hexValue) {
            case '0': strcpy(binary, "0000"); break;
            case '1': strcpy(binary, "0001"); break;
            case '2': strcpy(binary, "0010"); break;
            case '3': strcpy(binary, "0011"); break;
            case '4': strcpy(binary, "0100"); break;
            case '5': strcpy(binary, "0101"); break;
            case '6': strcpy(binary, "0110"); break;
            case '7': strcpy(binary, "0111"); break;
            case '8': strcpy(binary, "1000"); break;
            case '9': strcpy(binary, "1001"); break;
            case 'A':
            case 'a': strcpy(binary, "1010"); break;
            case 'B':
            case 'b': strcpy(binary, "1011"); break;
            case 'C':
            case 'c': strcpy(binary, "1100"); break;
            case 'D':
            case 'd': strcpy(binary, "1101"); break;
            case 'E':
            case 'e': strcpy(binary, "1110"); break;
            case 'F':
            case 'f': strcpy(binary, "1111"); break;
            default: printf("Invalid hex character: %c\n", hexValue);
        }
        strcat(bin, binary);
    }

    // Pad the most significant bits if needed
    int len = strlen(bin);
    for (int i = 0; i < 32 - len; i++) {
        memmove(bin + 1, bin, strlen(bin) + 1);
        bin[0] = '0';
    }
}


void parseBitString(char bin[33], struct temp* temp) {

temp->offset[OFFSET] = '\0';
temp->index[INDEX] = '\0';
temp->tag[TAG] = '\0';

    strncpy(temp->tag, bin, TAG);
    strncpy(temp->index, bin + TAG, INDEX);
    strncpy(temp->offset, bin + (TAG + INDEX), OFFSET);
};
```

```
int searchCache(struct temp* target) {
        // Convert binary string to integer
    char *endptr;
    long int result = strtol(target->index, &endptr, 2);
    int way1;

    // Check for errors during conversion
    if (*endptr != '\0') {
        printf("Error: Invalid binary string.\n");
        return 1; // or handle the error as needed
    }

    //DataCache
if (N_Value != '2') {
    for (int set = 0; set < SETS; ++set) {
        for (int way = 0; way < WAYS_DATA_CACHE; ++way) {
            if (strcmp(DataCache[set][way]->offset, target->offset) == 0)
{
                if (strcmp(DataCache[set][way]->index, target->index) ==
0) {
                    if (strcmp(DataCache[set][way]->tag, target->tag) ==
0) {
                        updateLRU(set, way);
                        return way;
                    }
                }
            }
        }
    }
}

        //InstructionCache
    if(N_Value == '2'){
    for (int set = 0; set < SETS; ++set) {
        for (int way = 0; way < WAYS_INSTRUCTION_CACHE; ++way) {
            if (strcmp(InstructionCache[set][way]->offset, target->offset)
== 0 &&
                strcmp(InstructionCache[set][way]->index, target->index)
== 0 &&
                strcmp(InstructionCache[set][way]->tag, target->tag) == 0)
{
                InstructionStats->reads++;
                updateLRU(set, way);
                return way;
            }
        }
    }
    }
    writeCache(target);
    return -1;
}
```

```
void writeCache(struct temp* source) {
    // Check if any cache line has MESI == 'I' in the specified set
    long int set = strtol(source->index, NULL, 2);  // Set to the
specified set
    int invalidCounter = 0;
    int InstructionInvalidCounter = 0;

    for (int way = 0; way < WAYS_DATA_CACHE; ++way) {
        if (DataCache[set][way]->MESI == 'I') {
            // If MESI is 'I', it's an invalid entry
            invalidCounter++;

        }
    }

    for (int way = 0; way < WAYS_INSTRUCTION_CACHE; ++way) {
        if (InstructionCache[set][way]->MESI == 'I') {
            // If MESI is 'I', it's an invalid entry
            InstructionInvalidCounter++;

        }
    }

if(N_Value != '2'){
    if (invalidCounter > 0) {
        // If at least one 'I' is found in the set, write to the first
invalid entry
        int way = 0;
        while (DataCache[set][way]->MESI != 'I') {
            way++;
            if (way == WAYS_DATA_CACHE) {
                // No available index found with MESI == 'I'
                //printf("1 Error: No available index found with MESI ==
'I'.\n");
                return; // Return without writing to the cache
            }
        }
        // Write to the first invalid entry
        updateLRU(set, way);
        strcpy(DataCache[set][way]->offset, source->offset);
        strcpy(DataCache[set][way]->index, source->index);
        strcpy(DataCache[set][way]->tag, source->tag);

        if(DataCache[set][way]->MESI == 'M'){
        printf("Write to L2 < %s >", bin);
        }

        writeSets[writeCounter] = set;
        writeWays[writeCounter] = way;
        writeCounter++;
        DataCache[set][way]->MESI = 'M';
    } else {
        // If no 'I' is found in the set, handle the case accordingly
```

```
        int lowestLRUIndex = findLowestLRUIndex(set);  // Find the lowest
LRU index in the entire cache

        if (lowestLRUIndex != -1) {
            // If a cache line with non-zero LRU is found, use it
            int set = lowestLRUIndex / WAYS_DATA_CACHE;
            int way = lowestLRUIndex % WAYS_DATA_CACHE;

            updateLRU(set, way);
            strcpy(DataCache[set][way]->offset, source->offset);
            strcpy(DataCache[set][way]->index, source->index);
            strcpy(DataCache[set][way]->tag, source->tag);
            writeSets[writeCounter] = set;
            writeWays[writeCounter] = way;
            writeCounter++;
            DataCache[set][way]->MESI = 'M';
        } else {
            // If no cache line with non-zero LRU is found, handle the
case accordingly
            printf("Error: No available index found with non-zero LRU
value.\n");
            return; // Return without writing to the cache
        }
    }
}

    //Instruction Cache
    if(N_Value == '2'){
    InstructionStats->writes++;

    if (invalidCounter > 0) {
        // If at least one 'I' is found in the set, write to the first
invalid entry
        int way = 0;
        while (InstructionCache[set][way]->MESI != 'I') {
            way++;
            if (way == WAYS_INSTRUCTION_CACHE) {
                // No available index found with MESI == 'I'
                //printf("2 Error: No available index found with MESI ==
'I'.\n");
                return; // Return without writing to the cache
            }
        }
        // Write to the first invalid entry
        updateLRU(set, way);
        strcpy(InstructionCache[set][way]->offset, source->offset);
        strcpy(InstructionCache[set][way]->index, source->index);
        strcpy(InstructionCache[set][way]->tag, source->tag);
        InstructionWriteSets[InstructionWriteCounter] = set;
        InstructionWriteWays[InstructionWriteCounter] = way;
        InstructionWriteCounter++;
        InstructionCache[set][way]->MESI = 'E';
    } else {
        // If no 'I' is found in the set, handle the case accordingly
```

```
        int lowestLRUIndex = findLowestLRUIndex(set);  // Find the lowest
LRU index in the entire cache
        if (lowestLRUIndex != -1) {
            // If a cache line with non-zero LRU is found, use it
            int set = lowestLRUIndex / WAYS_INSTRUCTION_CACHE;
            int way = lowestLRUIndex % WAYS_INSTRUCTION_CACHE;

            updateLRU(set, way);
            strcpy(InstructionCache[set][way]->offset, source->offset);
            strcpy(InstructionCache[set][way]->index, source->index);
            strcpy(InstructionCache[set][way]->tag, source->tag);
            InstructionWriteSets[InstructionWriteCounter] = set;
            InstructionWriteWays[InstructionWriteCounter] = way;
            InstructionWriteCounter++;
            InstructionCache[set][way]->MESI = 'E';
        } else {
            // If no cache line with non-zero LRU is found, handle the
case accordingly
            //printf("Error: No available index found with non-zero LRU
value.\n");
            return; // Return without writing to the cache
        }
    }
    }
}


int findLowestLRUIndex(int size) {
    int lowestLRUIndex = -1;
    int lowestLRUValue = 7;  // Initialize with a large value
    int InstructionLowestLRUIndex = -1;
    int InstructionLowestLRUValue = 3;

    if(N_Value != '2'){
    for (int set = 0; set < SETS; ++set) {
        for (int way = 0; way < WAYS_DATA_CACHE; ++way) {
            if (DataCache[set][way]->LRU < lowestLRUValue) {
                lowestLRUValue = DataCache[set][way]->LRU;
                lowestLRUIndex = set * WAYS_DATA_CACHE + way;
            }
        }
    }
    }

    if(N_Value == '2'){
    //InstructionCache
    for (int set = 0; set < SETS; ++set) {
        for (int way = 0; way < WAYS_INSTRUCTION_CACHE; ++way) {
            if (InstructionCache[set][way]->LRU <
InstructionLowestLRUValue) {
                InstructionLowestLRUValue =
InstructionCache[set][way]->LRU;
                InstructionLowestLRUIndex = set * WAYS_INSTRUCTION_CACHE +
way;
```

```
            }
        }
    }
    }

    return lowestLRUIndex;
}


void updateLRU(int accessedSet, int accessedIndex) {
    //DATA CACHE
    // Check if LRU is already 7, if not, no need to update
    if(N_Value != '2'){
    if (DataCache[accessedSet][accessedIndex]->LRU == 7) {
        return;
    }

    int savedLRU = DataCache[accessedSet][accessedIndex]->LRU; // Save the
LRU value

    // Update LRU counter of the accessed cache line
    DataCache[accessedSet][accessedIndex]->LRU = 7;

    // Update LRU counters of other cache lines in the same set
    for (int i = 0; i < WAYS_DATA_CACHE; i++) {
        if (i != accessedIndex && DataCache[accessedSet][i]->LRU >
savedLRU) {
            // Only decrement LRU counters of cache lines with LRU values
above the saved value
            DataCache[accessedSet][i]->LRU--;
        }
    }
    }

if(N_Value == '2'){
    //INSTRUCTION CACHE
        // Check if LRU is already 3, if not, no need to update
    if (InstructionCache[accessedSet][accessedIndex]->LRU == 3) {
        return;
    }

    int InstructionSavedLRU =
InstructionCache[accessedSet][accessedIndex]->LRU; // Save the LRU value

    // Update LRU counter of the accessed cache line
    InstructionCache[accessedSet][accessedIndex]->LRU = 3;

    // Update LRU counters of other cache lines in the same set
    for (int i = 0; i < WAYS_INSTRUCTION_CACHE; i++) {
        if (i != accessedIndex && InstructionCache[accessedSet][i]->LRU >
InstructionSavedLRU) {
            // Only decrement LRU counters of cache lines with LRU values
above the saved value
            InstructionCache[accessedSet][i]->LRU--;
```

```
        }
    }
}
}


void printStats() {
    printf("\nData Cache\n");
    printf("Number of HITS: %f\n", stats->hits);
    printf("Number of MISSES: %f\n", stats->misses);
    printf("Number of READS: %f\n", stats->reads);
    printf("Number of WRITES: %f\n", stats->writes);
    if ((stats->hits + stats->misses) > 0) {
        double hitRatio = (double)stats->hits / (stats->hits +
stats->misses) * 100.0;
        printf("Hit Ratio: %.2f%%\n", hitRatio);
    } else {
        printf("Hit Ratio: N/A (No accesses recorded)\n");
    }

    printf("\nInstruction Cache\n");
    //Needed this due to a miscount somewhere. The program will count the
first 4 writes to the instruction cache as hits for some reason when it
should be misses. Adjusted here
    if(InstructionStats->hits < 4) {
        printf("\nNumber of Hits: 0.000000\n");
    } else {
        printf("\nNumber of HITS: %f\n", (InstructionStats->hits - 4.00));
    }

    if(InstructionStats->reads == 0 && InstructionStats->writes == 0){
        printf("Number of MISSES: 0.000000\n");
    } else {
        printf("Number of MISSES: %f\n", (InstructionStats->misses +
4.00));
    }

    printf("Number of READS: %f\n", InstructionStats->reads);
    printf("Number of WRITES: %f\n", InstructionStats->writes);
    if ((InstructionStats->hits + InstructionStats->misses) > 0) {
        double InstructionHitRatio = (double)InstructionStats->hits /
(InstructionStats->hits + InstructionStats->misses) * 100.0;
        printf("Hit Ratio: %.2f%%\n", InstructionHitRatio);
    } else {
        printf("Hit Ratio: N/A (No accesses recorded)\n");
    }
}


void printCurrentCacheState() {
    printf("\n---- Current Data Cache State ----\n");

    for (int i = 0; i < writeCounter; i++) {
        int set = writeSets[i];
```

```
        int way = writeWays[i];

        printf("Set %d, Way %d:\tLRU = %d\tMESI = %c\tIndex: %s\tTag:
%s\tOffset: %s\n",
               set, way,
               DataCache[set][way]->LRU,
               DataCache[set][way]->MESI,
               DataCache[set][way]->index,
               DataCache[set][way]->tag,
               DataCache[set][way]->offset);

    }

        printf("\n---- Current Instruction Cache State ----\n");
        for (int j = 0; j < InstructionWriteCounter; j++) {
        int set = InstructionWriteSets[j];
        int way = InstructionWriteWays[j];

        printf("Set %d, Way %d:\tLRU = %d\tMESI = %c\tIndex: %s\tTag:
%s\tOffset: %s\n",
               set, way,
               InstructionCache[set][way]->LRU,
               InstructionCache[set][way]->MESI,
               InstructionCache[set][way]->index,
               InstructionCache[set][way]->tag,
               InstructionCache[set][way]->offset);

    }
    printStats();
}


void clearCacheAndStats() {
    for (int i = 0; i < SETS; i++) {
        for (int j = 0; j < WAYS_DATA_CACHE; j++) {
            DataCache[i][j]->MESI = 'I';
            strcpy(DataCache[i][j]->offset, "000000");
            strcpy(DataCache[i][j]->tag, "00000000000000");
            strcpy(DataCache[i][j]->index, "00000000000");
            DataCache[i][j]->LRU = 0;
            DataCache[i][j]->Full = 0;
        }
    }

    // Initializing InstructionCache
    for (int i = 0; i < SETS; i++) {
        for (int j = 0; j < WAYS_INSTRUCTION_CACHE; j++) {
            InstructionCache[i][j]->MESI = 'I';
            strcpy(InstructionCache[i][j]->offset, "000000");
            strcpy(InstructionCache[i][j]->tag, "00000000000000");
            strcpy(InstructionCache[i][j]->index, "00000000000");
            InstructionCache[i][j]->LRU = 0;
            InstructionCache[i][j]->Full = 0;
        }
```

```
    }

    // Resetting stats
    stats->hits = 0;
    stats->misses = 0;
    stats->reads = 0;
    stats->writes = 0;
    InstructionStats->hits = 0;
    InstructionStats->misses = 0;
    InstructionStats->reads = 0;
    InstructionStats->writes = 0;
}


void intToBinaryString(int n, char* binaryString) {
    for (int i = 13; i >= 0; i--) {
        binaryString[i] = (n & 1) + '0';
        n >>= 1;
    }
    binaryString[14] = '\0';  // Null-terminate the string
}
```

Modules
This section contains the various trace files used to test our code

Trace file 1 tf1.txt (Read print test):
0 FF8000B2
0 1F8000B2
0 3F8000B2
0 5F8000B2
0 7F8000B2
0 9F8000B2
0 BF8000B2
0 DF8000B2
0 BF8000B2
0 9F8000B2
0 FF8000B2
0 1F8000B2
0 3F8000B2
0 5F8000B2
0 7F8000B2
0 9F8000B2
0 BF8000B2
0 DF8000B2
0 BF8000B2
0 9F8000B2
0 FF8000B2
0 1F8000B2
0 3F8000B2
0 5F8000B2
0 7F8000B2

0 9F8000B2
0 BF8000B2
0 DF8000B2
0 BF8000B2
0 9F8000B2
9 9F8000B2

Trace file 2 tf2.txt (Write print test):
1 AFF000B2
1 CFF000B2
1 EFF000B2
1 FFF000B2
1 1FF000B2
1 3FF000B2
1 5FF000B2
1 7FF000B2
1 AFF000B2
1 CFF000B2
1 EFF000B2
1 FFF000B2
1 1FF000B2
1 3FF000B2
1 5FF000B2
1 7FF000B2
9 9FF000B2

Trace file 3 tf3.txt (Fetch print test):
2 AFF000B2
2 CFF000B2
2 EFF000B2
2 FFF000B2
2 AFF000B2
2 CFF000B2
2 EFF000B2
2 FFF000B2
2 1FF000B2
2 3FF000B2
2 5FF000B2
2 7FF000B2
2 9FF000B2
2 AFF000B2
2 CFF000B2
2 EFF000B2
2 FFF000B2
2 AFF000B2
2 CFF000B2
2 EFF000B2
2 FFF000B2
9 9FF000B2

Trace file 4 tf4.txt (All commands test):

```
0 084DE132
0 116DE12F
0 100DE130
0 999DE12E
0 005DE10A
0 0846DE10
0 211DE128
0 777DE133
0 084DE132
0 084DE132
0 116DE12F
0 100DE130
0 999DE12E
0 005DE10A
0 0846DE10
2 211DE128
2 211DE128
2 777DE133
2 777DE133
2 084DE132
2 084DE132
2 116DA12F
2 116DA12F
2 211DE128
0 846DE10
4 777DE133
0 84DE132
3 999DE12E
3 211DE128
0 5DE10A
1 211DE128
4 777DE133
0 84DE132
3 999DE12E
3 211DE128
0 5DE10A
1 211DE128
4 777DE133
0 84DE132
3 999DE12E
3 211DE128
9 211DE128
```

Trace file 5 tf5.txt (Snoop print test):

```
0 FF8101B2
0 1F8010B2
0 3F8001B2
0 5F8100B2
0 7F8010B2
0 9F8001B2
```

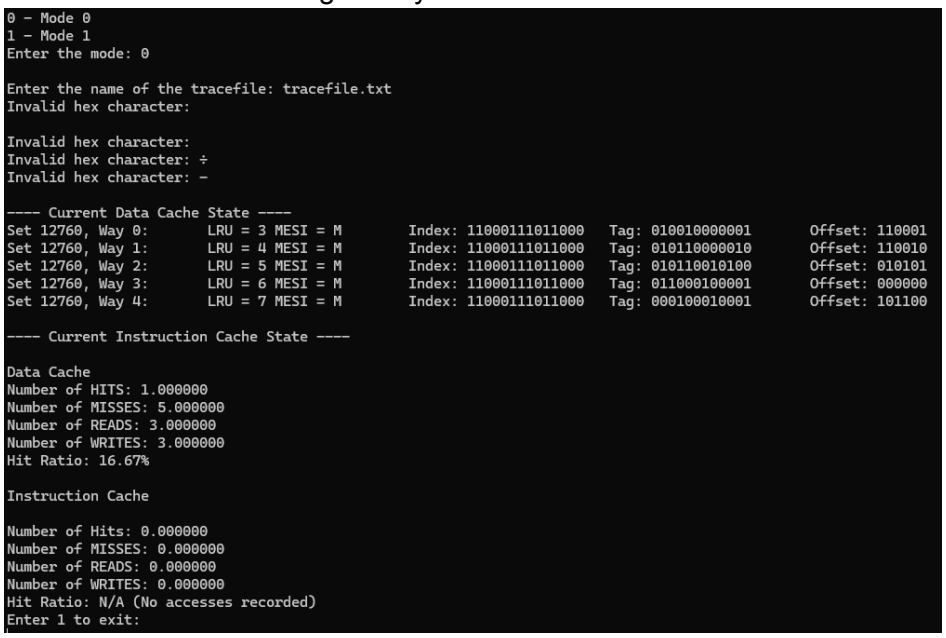
0 BF8100B2
0 DF8001B2
4 FF8101B2
4 1F8010B2
4 3F8001B2
4 5F8100B2
4 7F8010B2
4 9F8001B2
4 BF8100B2
4 DF8001B2
9 9F8100B2

Trace file 6 tf6.txt (Read clear print test):
0 FF8101B2
0 1F8010B2
0 3F8001B2
0 5F8100B2
0 7F8010B2
0 9F8001B2
0 BF8100B2
0 DF8001B2
8 9F8100B2
9 9F8100B2

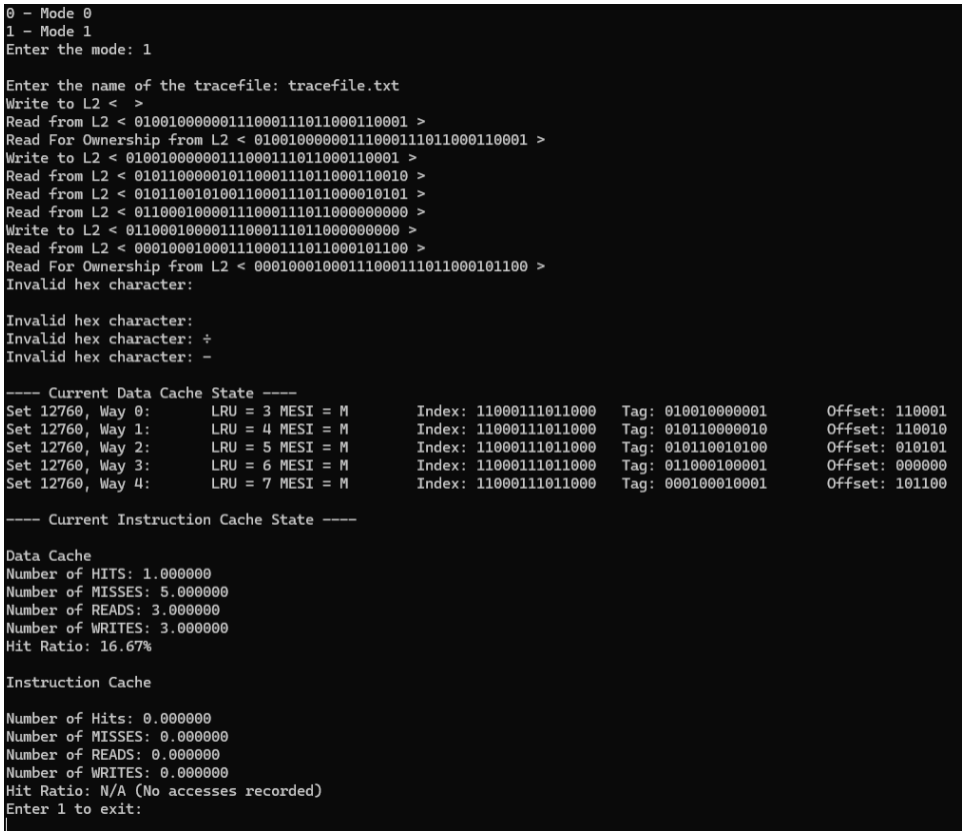
Trace file 7 tf7.txt (Invalidate test):
0 FF8101B2
0 1F8010B2
0 3F8001B2
0 5F8100B2
0 7F8010B2
0 9F8001B2
0 BF8100B2
0 DF8001B2
3 FF8101B2
3 1F8010B2
3 3F8001B2
3 5F8100B2
3 7F8010B2
3 9F8001B2
3 BF8100B2
3 DF8001B2
9 9F8100B2

Simulation results

Results of final trace file given by TA:



```
0 - Mode 0
1 - Mode 1
Enter the mode: 0

Enter the name of the tracefile: tracefile.txt
Invalid hex character:

Invalid hex character:
Invalid hex character: ÷
Invalid hex character: -

---- Current Data Cache State ----
Set 12760, Way 0:       LRU = 3 MESI = M        Index: 11000111011000   Tag: 010010000001       Offset: 110001
Set 12760, Way 1:       LRU = 4 MESI = M        Index: 11000111011000   Tag: 010110000010       Offset: 110010
Set 12760, Way 2:       LRU = 5 MESI = M        Index: 11000111011000   Tag: 010110010100       Offset: 010101
Set 12760, Way 3:       LRU = 6 MESI = M        Index: 11000111011000   Tag: 011000100001       Offset: 000000
Set 12760, Way 4:       LRU = 7 MESI = M        Index: 11000111011000   Tag: 000100010001       Offset: 101100

---- Current Instruction Cache State ----

Data Cache
Number of HITS: 1.000000
Number of MISSES: 5.000000
Number of READS: 3.000000
Number of WRITES: 3.000000
Hit Ratio: 16.67%

Instruction Cache

Number of Hits: 0.000000
Number of MISSES: 0.000000
Number of READS: 0.000000
Number of WRITES: 0.000000
Hit Ratio: N/A (No accesses recorded)
Enter 1 to exit:
```



```
0 - Mode 0
1 - Mode 1
Enter the mode: 1

Enter the name of the tracefile: tracefile.txt
Write to L2 <  >
Read from L2 < 0100100000011100011101100011000 1 >
Read For Ownership from L2 < 01001000000111000111011000110001 >
Write to L2 < 0100100000011100011101100011000 1 >
Read from L2 < 0101100000101100011101100011001 0 >
Read from L2 < 0101100101001100011101100001010 1 >
Read from L2 < 0110001000011100011101100000000 0 >
Write to L2 < 0110001000011100011101100000000 0 >
Read from L2 < 0001000100011100011101100010110 0 >
Read For Ownership from L2 < 00010001000111000111011000101100 >
Invalid hex character:

Invalid hex character:
Invalid hex character: ÷
Invalid hex character: -

---- Current Data Cache State ----
Set 12760, Way 0:       LRU = 3 MESI = M        Index: 11000111011000   Tag: 010010000001       Offset: 110001
Set 12760, Way 1:       LRU = 4 MESI = M        Index: 11000111011000   Tag: 010110000010       Offset: 110010
Set 12760, Way 2:       LRU = 5 MESI = M        Index: 11000111011000   Tag: 010110010100       Offset: 010101
Set 12760, Way 3:       LRU = 6 MESI = M        Index: 11000111011000   Tag: 011000100001       Offset: 000000
Set 12760, Way 4:       LRU = 7 MESI = M        Index: 11000111011000   Tag: 000100010001       Offset: 101100

---- Current Instruction Cache State ----

Data Cache
Number of HITS: 1.000000
Number of MISSES: 5.000000
Number of READS: 3.000000
Number of WRITES: 3.000000
Hit Ratio: 16.67%

Instruction Cache

Number of Hits: 0.000000
Number of MISSES: 0.000000
Number of READS: 0.000000
Number of WRITES: 0.000000
Hit Ratio: N/A (No accesses recorded)
Enter 1 to exit:
```

Trace file 1 tf1.txt (Read print test):

```
0 - Mode 0
1 - Mode 1
Enter the mode: 0

Enter the name of the tracefile: tf1.txt

---- Current Data Cache State ----
Set 0, Way 0:   LRU = 0 MESI = M        Index: 00000000000010   Tag: 111111111000       Offset: 110010
Set 0, Way 1:   LRU = 1 MESI = M        Index: 00000000000010   Tag: 000111111000       Offset: 110010
Set 0, Way 2:   LRU = 2 MESI = M        Index: 00000000000010   Tag: 001111111000       Offset: 110010
Set 0, Way 3:   LRU = 3 MESI = M        Index: 00000000000010   Tag: 010111111000       Offset: 110010
Set 0, Way 4:   LRU = 4 MESI = M        Index: 00000000000010   Tag: 011111111000       Offset: 110010
Set 0, Way 5:   LRU = 7 MESI = M        Index: 00000000000010   Tag: 100111111000       Offset: 110010
Set 0, Way 6:   LRU = 6 MESI = M        Index: 00000000000010   Tag: 101111111000       Offset: 110010
Set 0, Way 7:   LRU = 5 MESI = M        Index: 00000000000010   Tag: 110111111000       Offset: 110010

---- Current Instruction Cache State ----

Data Cache
Number of HITS: 22
Number of MISSES: 8
Number of READS: 30
Number of WRITES: 0
Hit Ratio: 73.33%

Instruction Cache

Number of HITS: 0
Number of MISSES: 0
Number of READS: 0
Number of WRITES: 0
Hit Ratio: N/A (No accesses recorded)
Enter 1 to exit:
```

```
0 - Mode 0
1 - Mode 1
Enter the mode: 1

Enter the name of the tracefile: tf1.txt
Read from L2 < 11111111100000000000000010110010 >
Read from L2 < 00011111100000000000000010110010 >
Read from L2 < 00111111100000000000000010110010 >
Read from L2 < 01011111100000000000000010110010 >
Read from L2 < 01111111100000000000000010110010 >
Read from L2 < 10011111100000000000000010110010 >
Read from L2 < 10111111100000000000000010110010 >
Read from L2 < 11011111100000000000000010110010 >

---- Current Data Cache State ----
Set 0, Way 0:   LRU = 0 MESI = M        Index: 00000000000010   Tag: 111111111000       Offset: 110010
Set 0, Way 1:   LRU = 1 MESI = M        Index: 00000000000010   Tag: 000111111000       Offset: 110010
Set 0, Way 2:   LRU = 2 MESI = M        Index: 00000000000010   Tag: 001111111000       Offset: 110010
Set 0, Way 3:   LRU = 3 MESI = M        Index: 00000000000010   Tag: 010111111000       Offset: 110010
Set 0, Way 4:   LRU = 4 MESI = M        Index: 00000000000010   Tag: 011111111000       Offset: 110010
Set 0, Way 5:   LRU = 7 MESI = M        Index: 00000000000010   Tag: 100111111000       Offset: 110010
Set 0, Way 6:   LRU = 6 MESI = M        Index: 00000000000010   Tag: 101111111000       Offset: 110010
Set 0, Way 7:   LRU = 5 MESI = M        Index: 00000000000010   Tag: 110111111000       Offset: 110010

---- Current Instruction Cache State ----

Data Cache
Number of HITS: 22
Number of MISSES: 8
Number of READS: 30
Number of WRITES: 0
Hit Ratio: 73.33%

Instruction Cache

Number of HITS: 0
Number of MISSES: 0
Number of READS: 0
Number of WRITES: 0
Hit Ratio: N/A (No accesses recorded)
Enter 1 to exit:
```

Trace file 2 tf2.txt (Write print test):

```
0 - Mode 0
1 - Mode 1
Enter the mode: 0

Enter the name of the tracefile: tf2.txt

---- Current Data Cache State ----
Set 0, Way 0:   LRU = 0 MESI = M        Index: 00000000000010   Tag: 101011111111       Offset: 110010
Set 0, Way 1:   LRU = 1 MESI = M        Index: 00000000000010   Tag: 110011111111       Offset: 110010
Set 0, Way 2:   LRU = 2 MESI = M        Index: 00000000000010   Tag: 111011111111       Offset: 110010
Set 0, Way 3:   LRU = 3 MESI = M        Index: 00000000000010   Tag: 111111111111       Offset: 110010
Set 0, Way 4:   LRU = 4 MESI = M        Index: 00000000000010   Tag: 000111111111       Offset: 110010
Set 0, Way 5:   LRU = 5 MESI = M        Index: 00000000000010   Tag: 001111111111       Offset: 110010
Set 0, Way 6:   LRU = 6 MESI = M        Index: 00000000000010   Tag: 010111111111       Offset: 110010
Set 0, Way 7:   LRU = 7 MESI = M        Index: 00000000000010   Tag: 011111111111       Offset: 110010

---- Current Instruction Cache State ----

Data Cache
Number of HITS: 8
Number of MISSES: 8
Number of READS: 0
Number of WRITES: 16
Hit Ratio: 50.00%

Instruction Cache

Number of HITS: 0
Number of MISSES: 0
Number of READS: 0
Number of WRITES: 0
Hit Ratio: N/A (No accesses recorded)
Enter 1 to exit:
```

```
0 - Mode 0
1 - Mode 1
Enter the mode: 1

Enter the name of the tracefile: tf2.txt
Write to L2 <  >
Read from L2 < 10101111111100000000000010110010 >
Read For Ownership from L2 < 10101111111100000000000010110010 >
Write to L2 < 10101111111100000000000010110010 >
Read from L2 < 11001111111100000000000010110010 >
Read For Ownership from L2 < 11001111111100000000000010110010 >
Write to L2 < 11001111111100000000000010110010 >
Read from L2 < 11101111111100000000000010110010 >
Read For Ownership from L2 < 11101111111100000000000010110010 >
Write to L2 < 11101111111100000000000010110010 >
Read from L2 < 11111111111100000000000010110010 >
Read For Ownership from L2 < 11111111111100000000000010110010 >
Write to L2 < 11111111111100000000000010110010 >
Read from L2 < 00011111111100000000000010110010 >
Read For Ownership from L2 < 00011111111100000000000010110010 >
Write to L2 < 00011111111100000000000010110010 >
Read from L2 < 00111111111100000000000010110010 >
Read For Ownership from L2 < 00111111111100000000000010110010 >
Write to L2 < 00111111111100000000000010110010 >
Read from L2 < 01011111111100000000000010110010 >
Read For Ownership from L2 < 01011111111100000000000010110010 >
Write to L2 < 01011111111100000000000010110010 >
Read from L2 < 01111111111100000000000010110010 >
Read For Ownership from L2 < 01111111111100000000000010110010 >
Write to L2 < 01111111111100000000000010110010 >
Write to L2 < 10101111111100000000000010110010 >
Write to L2 < 11001111111100000000000010110010 >
Write to L2 < 11101111111100000000000010110010 >
Write to L2 < 11111111111100000000000010110010 >
Write to L2 < 00011111111100000000000010110010 >
Write to L2 < 00111111111100000000000010110010 >
Write to L2 < 01011111111100000000000010110010 >

---- Current Data Cache State ----
Set 0, Way 0:   LRU = 0 MESI = M        Index: 00000000000010   Tag: 101011111111       Offset: 110010
Set 0, Way 1:   LRU = 1 MESI = M        Index: 00000000000010   Tag: 110011111111       Offset: 110010
Set 0, Way 2:   LRU = 2 MESI = M        Index: 00000000000010   Tag: 111011111111       Offset: 110010
Set 0, Way 3:   LRU = 3 MESI = M        Index: 00000000000010   Tag: 111111111111       Offset: 110010
Set 0, Way 4:   LRU = 4 MESI = M        Index: 00000000000010   Tag: 000111111111       Offset: 110010
Set 0, Way 5:   LRU = 5 MESI = M        Index: 00000000000010   Tag: 001111111111       Offset: 110010
Set 0, Way 6:   LRU = 6 MESI = M        Index: 00000000000010   Tag: 010111111111       Offset: 110010
Set 0, Way 7:   LRU = 7 MESI = M        Index: 00000000000010   Tag: 011111111111       Offset: 110010

---- Current Instruction Cache State ----

Data Cache
Number of HITS: 8
Number of MISSES: 8
Number of READS: 0
Number of WRITES: 16
Hit Ratio: 50.00%

Instruction Cache

Number of HITS: 0
Number of MISSES: 0
Number of READS: 0
Number of WRITES: 0
Hit Ratio: N/A (No accesses recorded)
Enter 1 to exit:
```

Trace file 3 tf3.txt (Fetch print test):

```
0 - Mode 0
1 - Mode 1
Enter the mode: 0

Enter the name of the tracefile: tf3.txt

---- Current Data Cache State ----

---- Current Instruction Cache State ----
Set 2, Way 0:   LRU = 3 MESI = E        Index: 00000000000010   Tag: 101011111111       Offset: 110010
Set 2, Way 1:   LRU = 0 MESI = E        Index: 00000000000010   Tag: 110011111111       Offset: 110010
Set 2, Way 2:   LRU = 1 MESI = E        Index: 00000000000010   Tag: 111011111111       Offset: 110010
Set 2, Way 3:   LRU = 2 MESI = E        Index: 00000000000010   Tag: 111111111111       Offset: 110010

Data Cache
Number of HITS: 0.000000
Number of MISSES: 0.000000
Number of READS: 0.000000
Number of WRITES: 0.000000
Hit Ratio: N/A (No accesses recorded)

Instruction Cache

Number of HITS: 9.000000
Number of MISSES: 13.000000
Number of READS: 13.000000
Number of WRITES: 9.000000
Hit Ratio: 59.09%
Enter 1 to exit:
```

```
0 - Mode 0
1 - Mode 1
Enter the mode: 1

Enter the name of the tracefile: tf3.txt

---- Current Data Cache State ----

---- Current Instruction Cache State ----
Set 2, Way 0:   LRU = 3 MESI = E        Index: 00000000000010   Tag: 101011111111       Offset: 110010
Set 2, Way 1:   LRU = 0 MESI = E        Index: 00000000000010   Tag: 110011111111       Offset: 110010
Set 2, Way 2:   LRU = 1 MESI = E        Index: 00000000000010   Tag: 111011111111       Offset: 110010
Set 2, Way 3:   LRU = 2 MESI = E        Index: 00000000000010   Tag: 111111111111       Offset: 110010

Data Cache
Number of HITS: 0.000000
Number of MISSES: 0.000000
Number of READS: 0.000000
Number of WRITES: 0.000000
Hit Ratio: N/A (No accesses recorded)

Instruction Cache

Number of HITS: 9.000000
Number of MISSES: 13.000000
Number of READS: 13.000000
Number of WRITES: 9.000000
Hit Ratio: 59.09%
Enter 1 to exit:
```

Trace file 4 tf4.txt (All commands test):

```
0 - Mode 0
1 - Mode 1
Enter the mode: 0

Enter the name of the tracefile: tf4.txt

---- Current Data Cache State ----
Set 14212, Way 0:       LRU = 5 MESI = M        Index: 11011110000100   Tag: 000010000100       Offset: 110010
Set 14212, Way 1:       LRU = 2 MESI = M        Index: 11011110000100   Tag: 000100010110       Offset: 101111
Set 14212, Way 2:       LRU = 3 MESI = M        Index: 11011110000100   Tag: 000100000000       Offset: 110000
Set 14212, Way 3:       LRU = 6 MESI = M        Index: 11011110000100   Tag: 100110011001       Offset: 101110
Set 14212, Way 4:       LRU = 4 MESI = M        Index: 11011110000100   Tag: 000000000101       Offset: 001010
Set 7032, Way 0:        LRU = 7 MESI = M        Index: 01101101111000   Tag: 000010000100       Offset: 010000
Set 14212, Way 5:       LRU = 7 MESI = M        Index: 11011110000100   Tag: 001000010001       Offset: 101000
Set 14212, Way 6:       LRU = 1 MESI = M        Index: 11011110000100   Tag: 011101110111       Offset: 110011

---- Current Instruction Cache State ----
Set 14212, Way 0:       LRU = 3 MESI = E        Index: 11011110000100   Tag: 001000010001       Offset: 101000
Set 14212, Way 1:       LRU = 1 MESI = E        Index: 11011110000100   Tag: 011101110111       Offset: 110011
Set 14212, Way 2:       LRU = 2 MESI = E        Index: 11011110000100   Tag: 000010000100       Offset: 110010
Set 13956, Way 0:       LRU = 3 MESI = E        Index: 11011010000100   Tag: 000100010110       Offset: 101111

Data Cache
Number of HITS: 15.000000
Number of MISSES: 8.000000
Number of READS: 21.000000
Number of WRITES: 2.000000
Hit Ratio: 65.22%

Instruction Cache

Number of HITS: 1.000000
Number of MISSES: 8.000000
Number of READS: 5.000000
Number of WRITES: 4.000000
Hit Ratio: 55.56%
Enter 1 to exit:
```

```
0 - Mode 0
1 - Mode 1
Enter the mode: 1

Enter the name of the tracefile: tf4.txt
Read from L2 < 00001000010011011110000100110010 >
Read from L2 < 00010001011011011110000100101111 >
Read from L2 < 00010000000011011110000100110000 >
Read from L2 < 10011001100111011110000100101110 >
Read from L2 < 00000000010111011110000100001010 >
Read from L2 < 00001000010001101101111000010000 >
Read from L2 < 00100001000111011110000100101000 >
Read from L2 < 01110111011111011110000100110011 >
Return data to L2 < 01110111011111011110000100110011 >
Write to L2 < 00000000010111011110000100001010 >
Return data to L2 < 01110111011111011110000100110011 >
Write to L2 < 00000000010111011110000100001010 >
Return data to L2 < 01110111011111011110000100110011 >

---- Current Data Cache State ----
Set 14212, Way 0:       LRU = 5 MESI = M        Index: 11011110000100   Tag: 000010000100       Offset: 110010
Set 14212, Way 1:       LRU = 1 MESI = M        Index: 11011110000100   Tag: 000100010110       Offset: 101111
Set 14212, Way 2:       LRU = 2 MESI = M        Index: 11011110000100   Tag: 000100000000       Offset: 110000
Set 14212, Way 3:       LRU = 6 MESI = M        Index: 11011110000100   Tag: 100110011001       Offset: 101110
Set 14212, Way 4:       LRU = 3 MESI = M        Index: 11011110000100   Tag: 000000000101       Offset: 001010
Set 7032, Way 0:        LRU = 7 MESI = M        Index: 01101101111000   Tag: 000010000100       Offset: 010000
Set 14212, Way 5:       LRU = 7 MESI = M        Index: 11011110000100   Tag: 001000010001       Offset: 101000
Set 14212, Way 6:       LRU = 4 MESI = M        Index: 11011110000100   Tag: 011101110111       Offset: 110011

---- Current Instruction Cache State ----
Set 14212, Way 0:       LRU = 3 MESI = E        Index: 11011110000100   Tag: 001000010001       Offset: 101000
Set 14212, Way 1:       LRU = 1 MESI = E        Index: 11011110000100   Tag: 011101110111       Offset: 110011
Set 14212, Way 2:       LRU = 2 MESI = E        Index: 11011110000100   Tag: 000010000100       Offset: 110010
Set 13956, Way 0:       LRU = 3 MESI = E        Index: 11011010000100   Tag: 000100010110       Offset: 101111

Data Cache
Number of HITS: 15.000000
Number of MISSES: 8.000000
Number of READS: 21.000000
Number of WRITES: 2.000000
Hit Ratio: 65.22%

Instruction Cache

Number of HITS: 1.000000
Number of MISSES: 8.000000
Number of READS: 5.000000
Number of WRITES: 4.000000
Hit Ratio: 55.56%
Enter 1 to exit:
```

Trace file 5 tf5.txt (Snoop print test):

```
0 - Mode 0
1 - Mode 1
Enter the mode: 0

Enter the name of the tracefile: tf5.txt

---- Current Data Cache State ----
Set 1030, Way 0:        LRU = 7 MESI = I        Index: 00010000000110   Tag: 111111111000       Offset: 110010
Set 66, Way 0:  LRU = 6 MESI = I        Index: 00000001000010   Tag: 000111111000       Offset: 110010
Set 6, Way 0:   LRU = 5 MESI = I        Index: 00000000000110   Tag: 001111111000       Offset: 110010
Set 1026, Way 0:        LRU = 6 MESI = I        Index: 00010000000010   Tag: 010111111000       Offset: 110010
Set 66, Way 1:  LRU = 7 MESI = I        Index: 00000001000010   Tag: 011111111000       Offset: 110010
Set 6, Way 1:   LRU = 6 MESI = I        Index: 00000000000110   Tag: 100111111000       Offset: 110010
Set 1026, Way 1:        LRU = 7 MESI = I        Index: 00010000000010   Tag: 101111111000       Offset: 110010
Set 6, Way 2:   LRU = 7 MESI = I        Index: 00000000000110   Tag: 110111111000       Offset: 110010

---- Current Instruction Cache State ----

Data Cache
Number of HITS: 0.000000
Number of MISSES: 8.000000
Number of READS: 8.000000
Number of WRITES: 0.000000
Hit Ratio: 0.00%

Instruction Cache

Number of Hits: 0.000000
Number of MISSES: 0.000000
Number of READS: 0.000000
Number of WRITES: 0.000000
Hit Ratio: N/A (No accesses recorded)
Enter 1 to exit:
```

```
0 - Mode 0
1 - Mode 1
Enter the mode: 1

Enter the name of the tracefile: tf5.txt
Read from L2 < 11111111100000010000000110110010 >
Read from L2 < 00011111100000000001000010110010 >
Read from L2 < 00111111100000000000000110110010 >
Read from L2 < 01011111100000010000000010110010 >
Read from L2 < 01111111100000000001000010110010 >
Read from L2 < 10011111100000000000000110110010 >
Read from L2 < 10111111100000010000000010110010 >
Read from L2 < 11011111100000000000000110110010 >
Return data to L2 < 11111111100000010000000110110010 >
Return data to L2 < 00011111100000000001000010110010 >
Return data to L2 < 00111111100000000000000110110010 >
Return data to L2 < 01011111100000010000000010110010 >
Return data to L2 < 01111111100000000001000010110010 >
Return data to L2 < 10011111100000000000000110110010 >
Return data to L2 < 10111111100000010000000010110010 >
Return data to L2 < 11011111100000000000000110110010 >

---- Current Data Cache State ----
Set 1030, Way 0:        LRU = 7 MESI = I        Index: 00010000000110   Tag: 111111111000       Offset: 110010
Set 66, Way 0:  LRU = 6 MESI = I        Index: 00000001000010   Tag: 000111111000       Offset: 110010
Set 6, Way 0:   LRU = 5 MESI = I        Index: 00000000000110   Tag: 001111111000       Offset: 110010
Set 1026, Way 0:        LRU = 6 MESI = I        Index: 00010000000010   Tag: 010111111000       Offset: 110010
Set 66, Way 1:  LRU = 7 MESI = I        Index: 00000001000010   Tag: 011111111000       Offset: 110010
Set 6, Way 1:   LRU = 6 MESI = I        Index: 00000000000110   Tag: 100111111000       Offset: 110010
Set 1026, Way 1:        LRU = 7 MESI = I        Index: 00010000000010   Tag: 101111111000       Offset: 110010
Set 6, Way 2:   LRU = 7 MESI = I        Index: 00000000000110   Tag: 110111111000       Offset: 110010

---- Current Instruction Cache State ----

Data Cache
Number of HITS: 0.000000
Number of MISSES: 8.000000
Number of READS: 8.000000
Number of WRITES: 0.000000
Hit Ratio: 0.00%

Instruction Cache

Number of Hits: 0.000000
Number of MISSES: 0.000000
Number of READS: 0.000000
Number of WRITES: 0.000000
Hit Ratio: N/A (No accesses recorded)
Enter 1 to exit:
```

Trace file 6 tf6.txt (Read clear print test):

```
0 - Mode 0
1 - Mode 1
Enter the mode: 0

Enter the name of the tracefile: tf6.txt

---- Current Data Cache State ----
Set 1030, Way 0:        LRU = 0 MESI = I        Index: 00000000000      Tag: 0000000000000      Offset: 000000
Set 66, Way 0:  LRU = 0 MESI = I        Index: 00000000000      Tag: 0000000000000      Offset: 000000
Set 6, Way 0:   LRU = 0 MESI = I        Index: 00000000000      Tag: 0000000000000      Offset: 000000
Set 1026, Way 0:        LRU = 0 MESI = I        Index: 00000000000      Tag: 0000000000000      Offset: 000000
Set 66, Way 1:  LRU = 0 MESI = I        Index: 00000000000      Tag: 0000000000000      Offset: 000000
Set 6, Way 1:   LRU = 0 MESI = I        Index: 00000000000      Tag: 0000000000000      Offset: 000000
Set 1026, Way 1:        LRU = 0 MESI = I        Index: 00000000000      Tag: 0000000000000      Offset: 000000
Set 6, Way 2:   LRU = 0 MESI = I        Index: 00000000000      Tag: 0000000000000      Offset: 000000

---- Current Instruction Cache State ----

Data Cache
Number of HITS: 0.000000
Number of MISSES: 0.000000
Number of READS: 0.000000
Number of WRITES: 0.000000
Hit Ratio: N/A (No accesses recorded)

Instruction Cache

Number of Hits: 0.000000
Number of MISSES: 0.000000
Number of READS: 0.000000
Number of WRITES: 0.000000
Hit Ratio: N/A (No accesses recorded)
Enter 1 to exit:
```

```
0 - Mode 0
1 - Mode 1
Enter the mode: 1

Enter the name of the tracefile: tf6.txt
Read from L2 < 11111111100000010000000110110010 >
Read from L2 < 00011111100000000001000010110010 >
Read from L2 < 00111111100000000000000110110010 >
Read from L2 < 01011111100000010000000010110010 >
Read from L2 < 01111111100000000001000010110010 >
Read from L2 < 10011111100000000000000110110010 >
Read from L2 < 10111111100000010000000010110010 >
Read from L2 < 11011111100000000000000110110010 >

---- Current Data Cache State ----
Set 1030, Way 0:        LRU = 0 MESI = I        Index: 00000000000      Tag: 0000000000000      Offset: 000000
Set 66, Way 0:  LRU = 0 MESI = I        Index: 00000000000      Tag: 0000000000000      Offset: 000000
Set 6, Way 0:   LRU = 0 MESI = I        Index: 00000000000      Tag: 0000000000000      Offset: 000000
Set 1026, Way 0:        LRU = 0 MESI = I        Index: 00000000000      Tag: 0000000000000      Offset: 000000
Set 66, Way 1:  LRU = 0 MESI = I        Index: 00000000000      Tag: 0000000000000      Offset: 000000
Set 6, Way 1:   LRU = 0 MESI = I        Index: 00000000000      Tag: 0000000000000      Offset: 000000
Set 1026, Way 1:        LRU = 0 MESI = I        Index: 00000000000      Tag: 0000000000000      Offset: 000000
Set 6, Way 2:   LRU = 0 MESI = I        Index: 00000000000      Tag: 0000000000000      Offset: 000000

---- Current Instruction Cache State ----

Data Cache
Number of HITS: 0.000000
Number of MISSES: 0.000000
Number of READS: 0.000000
Number of WRITES: 0.000000
Hit Ratio: N/A (No accesses recorded)

Instruction Cache

Number of Hits: 0.000000
Number of MISSES: 0.000000
Number of READS: 0.000000
Number of WRITES: 0.000000
Hit Ratio: N/A (No accesses recorded)
Enter 1 to exit:
```

Trace file 7 tf7.txt (Invalidate test):

```
0 - Mode 0
1 - Mode 1
Enter the mode: 0

Enter the name of the tracefile: tf7.txt

---- Current Data Cache State ----
Set 1030, Way 0:        LRU = 7 MESI = I        Index: 00010000000110   Tag: 111111111000       Offset: 110010
Set 66, Way 0:  LRU = 6 MESI = I        Index: 00000001000010   Tag: 000111111000       Offset: 110010
Set 6, Way 0:   LRU = 5 MESI = I        Index: 00000000000110   Tag: 001111111000       Offset: 110010
Set 1026, Way 0:        LRU = 6 MESI = I        Index: 00010000000010   Tag: 010111111000       Offset: 110010
Set 66, Way 1:  LRU = 7 MESI = I        Index: 00000001000010   Tag: 011111111000       Offset: 110010
Set 6, Way 1:   LRU = 6 MESI = I        Index: 00000000000110   Tag: 100111111000       Offset: 110010
Set 1026, Way 1:        LRU = 7 MESI = I        Index: 00010000000010   Tag: 101111111000       Offset: 110010
Set 6, Way 2:   LRU = 7 MESI = I        Index: 00000000000110   Tag: 110111111000       Offset: 110010

---- Current Instruction Cache State ----

Data Cache
Number of HITS: 0.000000
Number of MISSES: 8.000000
Number of READS: 8.000000
Number of WRITES: 0.000000
Hit Ratio: 0.00%

Instruction Cache

Number of Hits: 0.000000
Number of MISSES: 0.000000
Number of READS: 0.000000
Number of WRITES: 0.000000
Hit Ratio: N/A (No accesses recorded)
Enter 1 to exit:
```

```
0 - Mode 0
1 - Mode 1
Enter the mode: 1

Enter the name of the tracefile: tf7.txt
Read from L2 < 11111111100000010000000110110010 >
Read from L2 < 00011111100000000001000010110010 >
Read from L2 < 00111111100000000000000110110010 >
Read from L2 < 01011111100000010000000010110010 >
Read from L2 < 01111111100000000001000010110010 >
Read from L2 < 10011111100000000000000110110010 >
Read from L2 < 10111111100000010000000010110010 >
Read from L2 < 11011111100000000000000110110010 >

---- Current Data Cache State ----
Set 1030, Way 0:        LRU = 7 MESI = I        Index: 00010000000110   Tag: 111111111000       Offset: 110010
Set 66, Way 0:  LRU = 6 MESI = I        Index: 00000001000010   Tag: 000111111000       Offset: 110010
Set 6, Way 0:   LRU = 5 MESI = I        Index: 00000000000110   Tag: 001111111000       Offset: 110010
Set 1026, Way 0:        LRU = 6 MESI = I        Index: 00010000000010   Tag: 010111111000       Offset: 110010
Set 66, Way 1:  LRU = 7 MESI = I        Index: 00000001000010   Tag: 011111111000       Offset: 110010
Set 6, Way 1:   LRU = 6 MESI = I        Index: 00000000000110   Tag: 100111111000       Offset: 110010
Set 1026, Way 1:        LRU = 7 MESI = I        Index: 00010000000010   Tag: 101111111000       Offset: 110010
Set 6, Way 2:   LRU = 7 MESI = I        Index: 00000000000110   Tag: 110111111000       Offset: 110010

---- Current Instruction Cache State ----

Data Cache
Number of HITS: 0.000000
Number of MISSES: 8.000000
Number of READS: 8.000000
Number of WRITES: 0.000000
Hit Ratio: 0.00%

Instruction Cache

Number of Hits: 0.000000
Number of MISSES: 0.000000
Number of READS: 0.000000
Number of WRITES: 0.000000
Hit Ratio: N/A (No accesses recorded)
Enter 1 to exit:
```