2021학년도 2학기
Computer Architecture

Term Project: Cache Implementation

과 목 명	ComputerArchitecture
담당교수	HyungJune Lee
학 과	컴퓨터공학과
학 번	2071035 / 2076235
이 름	LeeSomin / AnHeejae
제 출 일	2021.12.04

이화여자대학교
EWHA WOMANS UNIVERSITY

2071035 이소민(Lee Somin)
2076235 안희재(An Heejae)

Table of Contents

2071035 이소민(Lee Somin)
2076235 안희재(An Heejae)

1. Code: cache_impl.h

```
/*
 * cache_impl.h
 *
 * 20493-01 Computer Architecture
 * Term Project on Implentation of Cache Mechanism
 *
 * Skeleton Code Prepared by Prof. HyungJune Lee
 * Nov 15, 2021
 *
 */

/* DO NOT CHANGE THE FOLLOWING DEFINITIONS EXCEPT 'DEFAULT_CACHE_ASSOC */

#ifndef _CACHE_IMPL_H_
#define _CACHE_IMPL_H_

#define WORD_SIZE_BYTE                   4
#define DEFAULT_CACHE_SIZE_BYTE          32
#define DEFAULT_CACHE_BLOCK_SIZE_BYTE    8
#define DEFAULT_CACHE_ASSOC              2   /* This can be changed to 1(for direct mapped
cache) or 4(for fully assoc cache) */
#define DEFAULT_MEMORY_SIZE_WORD         128
#define CACHE_ACCESS_CYCLE               1
#define MEMORY_ACCESS_CYCLE              100
#define CACHE_SET_SIZE
((DEFAULT_CACHE_SIZE_BYTE)/(DEFAULT_CACHE_BLOCK_SIZE_BYTE*DEFAULT_CACHE_ASSOC))

/* Function Prototypes */
void init_memory_content();
void init_cache_content();
void print_cache_entries();
int check_cache_data_hit(void* addr, char type);
int access_memory(void *addr, char type);


/* Cache Entry Structure */
typedef struct cache_entry {
    int valid;
    int tag;
    int timestamp;
    char data[DEFAULT_CACHE_BLOCK_SIZE_BYTE];
} cache_entry_t;


#endif
```

2. Code: cache.c

```
/*
 * cache.c
 *
 * 20493-01 Computer Architecture
 * Term Project on Implentation of Cache Mechanism
 *
 * Skeleton Code Prepared by Prof. HyungJune Lee
 * Nov 15, 2021
 *
 */


#include <stdio.h>
#include <string.h>
#include "cache_impl.h"

extern int num_cache_hits;
extern int num_cache_misses;

extern int num_bytes;
extern int num_access_cycles;

extern int global_timestamp;

cache_entry_t cache_array[CACHE_SET_SIZE][DEFAULT_CACHE_ASSOC];
int memory_array[DEFAULT_MEMORY_SIZE_WORD];


/* DO NOT CHANGE THE FOLLOWING FUNCTION */
void init_memory_content() {
    unsigned char sample_upward[16] = {0x001, 0x012, 0x023, 0x034, 0x045, 0x056, 0x067, 0x078,
0x089, 0x09a, 0x0ab, 0x0bc, 0x0cd, 0x0de, 0x0ef};
    unsigned char sample_downward[16] = {0x0fe, 0x0ed, 0x0dc, 0x0cb, 0x0ba, 0x0a9, 0x098, 0x087,
0x076, 0x065, 0x054, 0x043, 0x032, 0x021, 0x010};
    int index, i=0, j=1, gap = 1;

    for (index=0; index < DEFAULT_MEMORY_SIZE_WORD; index++) {
        memory_array[index] = (sample_upward[i] << 24) | (sample_upward[j] << 16) |
(sample_downward[i] << 8) | (sample_downward[j]);
        if (++i >= 16)
            i = 0;
        if (++j >= 16)
            j = 0;

        if (i == 0 && j == i+gap)
```

```
            j = i + (++gap);

        printf("mem[%d] = %#x\n", index, memory_array[index]);
    }
}

/* DO NOT CHANGE THE FOLLOWING FUNCTION */
void init_cache_content() {
    int i, j;

    for (i=0; i<CACHE_SET_SIZE; i++) {
        for (j=0; j < DEFAULT_CACHE_ASSOC; j++) {
            cache_entry_t *pEntry = &cache_array[i][j];
            pEntry->valid = 0;
            pEntry->tag = -1;
            pEntry->timestamp = 0;
        }
    }
}

/* DO NOT CHANGE THE FOLLOWING FUNCTION */
/* This function is a utility function to print all the cache entries. It will be useful for your
debugging */
void print_cache_entries() {
    int i, j, k;

    for (i=0; i<CACHE_SET_SIZE; i++) {
        printf("[Set %d] ", i);
        for (j=0; j <DEFAULT_CACHE_ASSOC; j++) {
            cache_entry_t *pEntry = &cache_array[i][j];
            printf("V: %d Tag: %#x Time: %d Data: ", pEntry->valid, pEntry->tag,
pEntry->timestamp);
            for (k=0; k<DEFAULT_CACHE_BLOCK_SIZE_BYTE; k++) {
                printf("%#x(%d) ", pEntry->data[k], k);
            }
            printf("\t");
        }
        printf("\n");
    }
}

int check_cache_data_hit(void* addr, char type) {

    int access_addr = (int)(addr);

    //add cache access cycle time to the total access cycle
```

```
    num_access_cycles += CACHE_ACCESS_CYCLE;

    //calculate the block address for calculating the tag and set
    int block_address = access_addr / DEFAULT_CACHE_BLOCK_SIZE_BYTE;

    //calculate the tag and set bit using block address and number of sets
    int tag = block_address / CACHE_SET_SIZE;
    int set = block_address % CACHE_SET_SIZE;

    //check if the data is in the cache
    for (int entry = 0; entry < DEFAULT_CACHE_ASSOC; entry++) {
        //when the valid bit is 1 and the tag of the cache entry matches the calculated tag,
        if (cache_array[set][entry].valid && cache_array[set][entry].tag == tag) {

            //update the time stamp of the cache entry
            cache_array[set][entry].timestamp = global_timestamp;

            //calculate the byte offset
            int byte_offset = access_addr % DEFAULT_CACHE_BLOCK_SIZE_BYTE;

            //set the byte size of the data to fetch using switch
            int byte_size = 0;
            switch (type)
            {
            case 'w'://word
                byte_size = 4;
                break;

            case 'h'://half word
                byte_size = 2;
                break;

            case 'b'://byte
                byte_size = 1;
                break;

            default:
                byte_size = -1;
                fprintf(stderr,"error: Association type not defined");
                break;
            }

            //add byte size of accessed data to num_bytes
            num_bytes += byte_size;

            //copy the data from the cache entry
```

```
            int value_returned = 0;
            for (int j = byte_offset + byte_size - 1; j >= byte_offset; j--) {
                //shift 2 digits in hexadecimal and append the following value using or operation
                int digit = cache_array[set][entry].data[j] & 0xff; //prevent copying sign extension
                value_returned = (value_returned << 8) | digit;

            }
            //return data fetched from cache
            return value_returned;
        }
    }
    /* Return the data */
    //this return statement is executed when there is no matching data in the cache
    return -1;
}

/* this function is to find the entry index in set for copying to cache */
int find_entry_index_in_set(int set)
{
    int entry_index;

    /* check if there exists any empty cache space by cheking 'valid' */
    for (int i = 0; i < DEFAULT_CACHE_ASSOC; i++)
    {
        if (!cache_array[set][i].valid)
        {
            return i;
        }
    }
    /* if the set has only 1 entry, return index 0 */
    if (DEFAULT_CACHE_ASSOC == 1)     // case of 1-way set associative cache (direct mapped
cache)
    {
        return 0;
    }
    /* otherwise, search over all entries to find the least recently used entry bu cheking
'timestamp' */
    else                         // case of 2/4-way set associative cache
    {
        entry_index = 0;
        for (int i = 0; i < DEFAULT_CACHE_ASSOC; i++)       // find minimum timestamp(the one
unused for the longest time)
        {
            if (cache_array[set][i].timestamp < cache_array[set][entry_index].timestamp)
            {
                entry_index = i;
```

```
            }
        }
    }
    /* return the cache index for copying from memory */
    return entry_index;
}

int access_memory(void *addr, char type) {

    int access_addr = (int)(addr);

    //add memory access cycle time to the total access cycle
    num_access_cycles += MEMORY_ACCESS_CYCLE;

    //calculate the block address for calculating the tag and set
    int block_address = access_addr / DEFAULT_CACHE_BLOCK_SIZE_BYTE;

    //calculate the tag and set bit using block address and number of sets
    int tag = block_address / CACHE_SET_SIZE;
    int set = block_address % CACHE_SET_SIZE;

    //get entry index to copy the data and paste to
    int entry_index = find_entry_index_in_set(set);

    //get the word address to use in the memory access
    int word_address = block_address * DEFAULT_CACHE_BLOCK_SIZE_BYTE / WORD_SIZE_BYTE;

    //exception: when the word address goes over the size of the memory
    if (DEFAULT_MEMORY_SIZE_WORD < word_address) {
        printf("error: Memory address out of range");
        return -1;
    }

    //set the cache entry for data copy from memory
    cache_array[set][entry_index].valid = 1;
    cache_array[set][entry_index].tag = tag;
    cache_array[set][entry_index].timestamp = global_timestamp;

    //copy a block(2words) from memory
    int mem_data = memory_array[word_address];
    for (int i = 0; i < DEFAULT_CACHE_BLOCK_SIZE_BYTE; i++) {
        if (i == DEFAULT_CACHE_BLOCK_SIZE_BYTE / 2) {
            mem_data = memory_array[word_address + 1];
        }

        //cut last two digits in hexadecimal form and save to the cache data location
```

```
        cache_array[set][entry_index].data[i] = mem_data % 256;
        mem_data >>= 8;
    }

    //calculate the byte offset
    int byte_offset = access_addr % DEFAULT_CACHE_BLOCK_SIZE_BYTE;

    //set the byte size of the data to fetch using switch
    int byte_size = 0;
    switch (type)
    {
    case 'w'://word
        byte_size = 4;
        break;

    case 'h'://half word
        byte_size = 2;
        break;

    case 'b'://byte
        byte_size = 1;
        break;

    default:
        byte_size = -1;
        fprintf(stderr,"error: Association type not defined");
        break;
    }

    num_bytes += byte_size;

    //copy the data from the cache entry
    int value_returned = 0;
    for (int j = byte_offset + byte_size - 1; j >= byte_offset; j--) {
        //shift 2 digits in hexadecimal and append the following value using or operation
        int digit = cache_array[set][entry_index].data[j] & 0xff; //prevent copying sign extension
        value_returned = (value_returned << 8) | digit;
    }

    //return fetched data from memory
    return value_returned;
}
```

3. Code: main.c

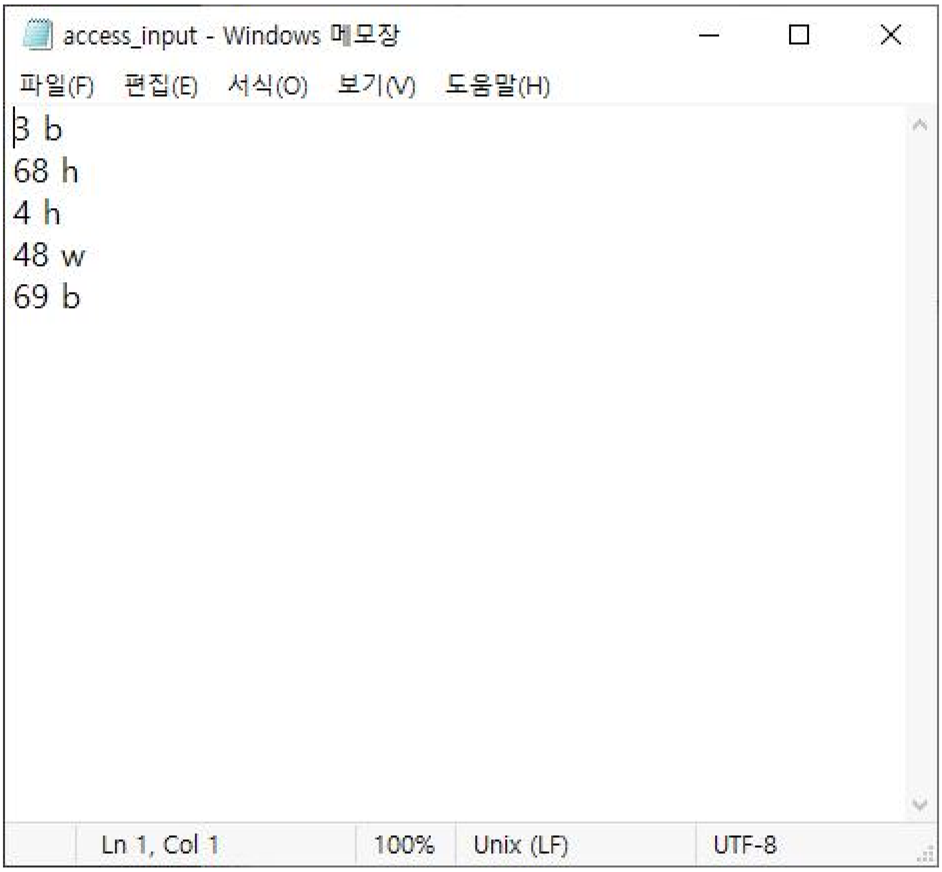
```
/*
 * main.c
 *
 * 20493-01 Computer Architecture
 * Term Project on Implentation of Cache Mechanism
 *
 * Skeleton Code Prepared by Prof. HyungJune Lee
 * Nov 15, 2021
 *
 */

#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "cache_impl.h"

#define MAX_LINE_SIZE 10

int num_cache_hits = 0;
int num_cache_misses = 0;

int num_bytes = 0;
int num_access_cycles = 0;

int global_timestamp = 0;

int retrieve_data(void *addr, char data_type) {
    int value_returned = -1; /* accessed data */

    /* Invoke check_cache_data_hit() */
    value_returned = check_cache_data_hit(addr, data_type);

    if (value_returned == -1) { //miss event
        num_cache_misses++;
        //access memory when miss event
        value_returned = access_memory(addr, data_type);
    }
    else //hit event
        num_cache_hits++;

    return value_returned;
}
```

```
int main(void) {
    FILE *ifp = NULL, *ofp = NULL;
    unsigned long int access_addr; /* byte address (located at 1st column) in "access_input.txt" */
    char access_type; /* 'b'(byte), 'h'(halfword), or 'w'(word) (located at 2nd column) in
"access_input.txt" */
    int accessed_data; /* This is the data that you want to retrieve first from cache, and then
from memory */

    init_memory_content();
    init_cache_content();

    ifp = fopen("access_input.txt", "r");
    if (ifp == NULL) {
        printf("Can't open input file\n");
        return -1;
    }
    ofp = fopen("access_output.txt", "w");
    if (ofp == NULL) {
        printf("Can't open output file\n");
        fclose(ifp);
        return -1;
    }

    fputs("[Accessed Data]\n", ofp);

        /* read each line and get the data in given (address, type) by invoking retrieve_data() */
        while (1)
    {
        char *tmp;// temporary storage for line read from file.
        char line[MAX_LINE_SIZE]; // buffer space to copy the line.
                int i = 1;

                //get a line from the file
        tmp = fgets(line, MAX_LINE_SIZE, ifp);

        //when there is no line to read, break.
        if (tmp == NULL)
            break;

        //get the string before first " " and change it to int
                access_addr = atoi(strtok(tmp," "));

        //initialize access_type and find for data type using while loop
        access_type = tmp[0];
        while (!(access_type == 'w' || access_type == 'h' || access_type == 'b'))
            access_type = tmp[i++];
```

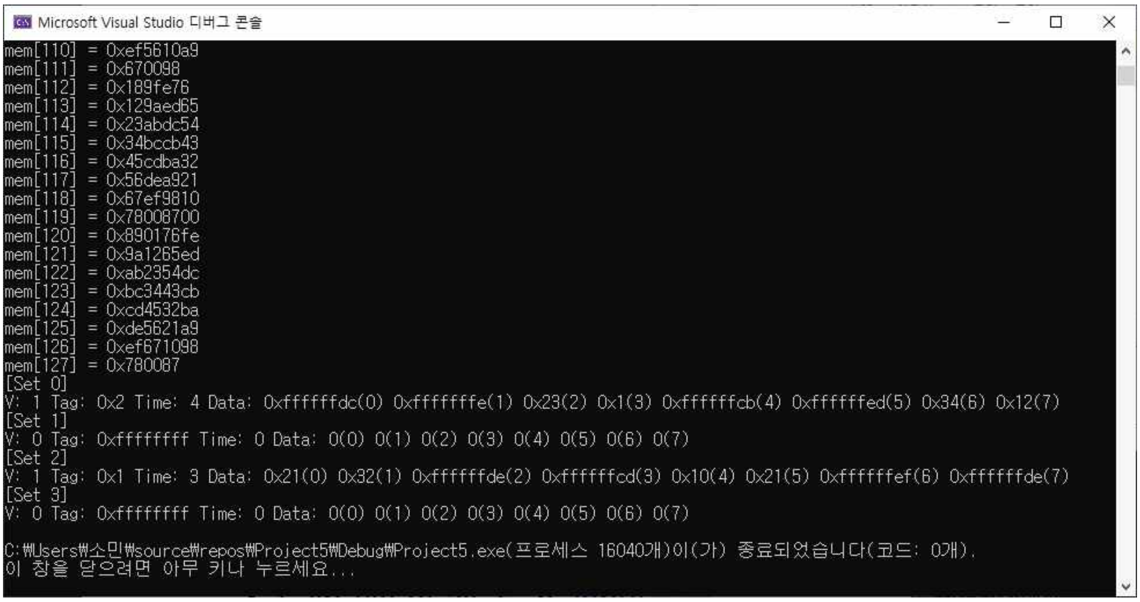
```
        //fetch data from cache or memory
                accessed_data = retrieve_data(access_addr, access_type);

        //write fetched information in file
                fprintf(ofp,"%d    %c       %#x\n", access_addr, access_type, accessed_data);
        global_timestamp++;

        }

        /* print hit ratio and bandwidth for each cache mecahanism as ragards to cache
association size */
        fputs("-----------------------------------------\n", ofp);

        switch (DEFAULT_CACHE_ASSOC)       // select the output statement according to the
assocativity.
        {
        case 1:
                fputs("[Direct mapped cache performance]\n", ofp);
                break;
        case 2:
                fputs("[2-way set associative cache performance]\n", ofp);
                break;
        case 4:
                fputs("[Fully associative cache performance]\n", ofp);
                break;
        default:
                fputs("error\n", ofp);
        }

    //calculate hit ratio and bandwidth
    float hit_ratio = (float)num_cache_hits / (float)(num_cache_hits + num_cache_misses);
    float bandwidth = (float)num_bytes / (float)num_access_cycles;

    //file write
        fprintf(ofp, "Hit ratio = %.2f (%d/%d)\n", hit_ratio, num_cache_hits, num_cache_hits +
num_cache_misses);
        fprintf(ofp, "Bandwidth = %.2f (%d/%d)", bandwidth, num_bytes, num_access_cycles);

        /* close files */
    fclose(ifp);
    fclose(ofp);

    print_cache_entries();
    return 0;
}
```

4. Results

input file 1:



direct mapped:



(consol output)

2071035 이소민(Lee Somin)
2076235 안희재(An Heejae)

```
access_output - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
[Accessed Data]
3       b       0x1
68      h       0xedcb
4       h       0xeddc
48      w       0xcdde3221
69      b       0xed
-----------------------------------------
[Direct mapped cache performance]
Hit ratio = 0.00 (0/5)
Bandwidth = 0.02 (10/505)

Ln 1, Col  100%  Windows (CRLF)  UTF-8
```

(file output)

2-way set associative:

```
Microsoft Visual Studio 디버그 콘솔
mem[108] = 0xcd3432cb
mem[109] = 0xde4521ba
mem[110] = 0xef5610a9
mem[111] = 0x670098
mem[112] = 0x189fe76
mem[113] = 0x129aed65
mem[114] = 0x23abdc54
mem[115] = 0x34bccb43
mem[116] = 0x45cdba32
mem[117] = 0x56dea921
mem[118] = 0x67ef9810
mem[119] = 0x78008700
mem[120] = 0x890176fe
mem[121] = 0x9a1265ed
mem[122] = 0xab2354dc
mem[123] = 0xbc3443cb
mem[124] = 0xcd4532ba
mem[125] = 0xde5621a9
mem[126] = 0xef671098
mem[127] = 0x780087
[Set 0]
V: 1 Tag: 0x4 Time: 4 Data: 0xffffffdc(0) 0xfffffffe(1) 0x23(2) 0x1(3) 0xffffffcb(4) 0xffffffed(5) 0x34(6) 0x12(7)
V: 1 Tag: 0x3 Time: 3 Data: 0x21(0) 0x32(1) 0xffffffde(2) 0xffffffcd(3) 0x10(4) 0x21(5) 0xffffffef(6) 0xffffffde(7)
[Set 1]
V: 0 Tag: 0xffffffff Time: 0 Data: 0(0) 0(1) 0(2) 0(3) 0(4) 0(5) 0(6) 0(7)
V: 0 Tag: 0xffffffff Time: 0 Data: 0(0) 0(1) 0(2) 0(3) 0(4) 0(5) 0(6) 0(7)

C:₩Users₩소민₩source₩repos₩Project5₩Debug₩Project5.exe(프로세스 6776개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

(consol output)

2071035 이소민(Lee Somin)
2076235 안희재(An Heejae)

access_output - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
[Accessed Data]
3       b       0x1
68      h       0xedcb
4       h       0xeddc
48      w       0xcdde3221
69      b       0xed
----------------------------------------
[2-way set associative cache performance]
Hit ratio = 0.20 (1/5)
Bandwidth = 0.02 (10/405)
```

Ln 1, Col 1 100% Windows (CRLF) UTF-8

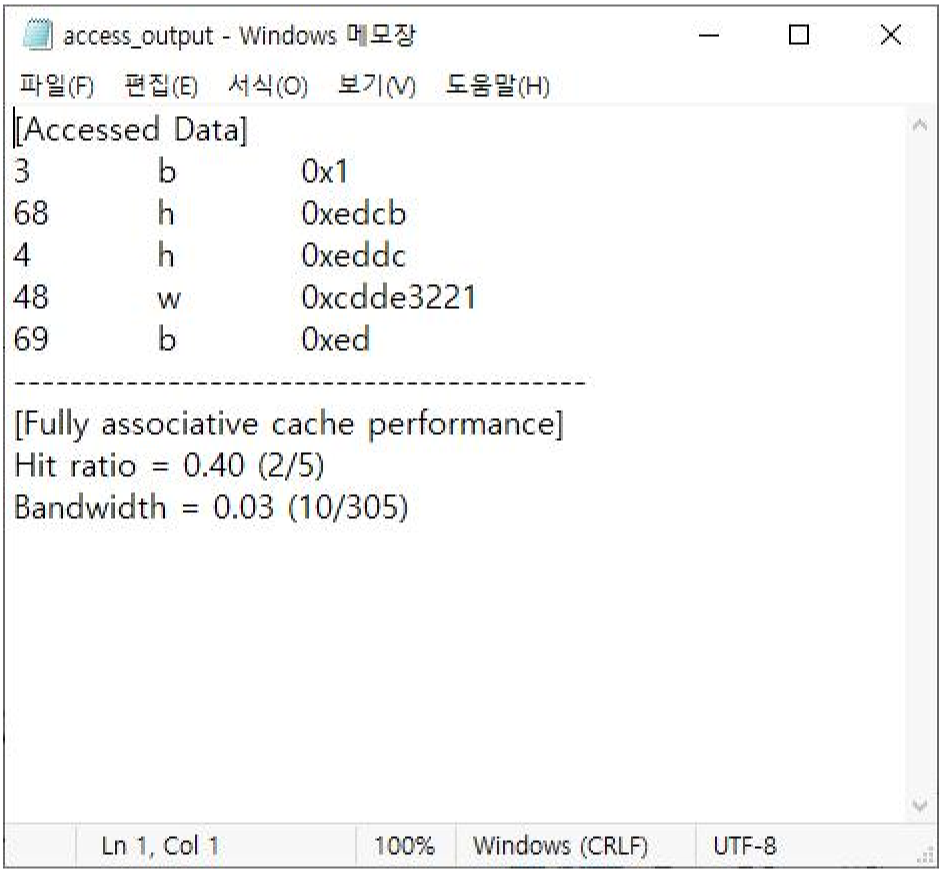
(file output)

Fully associative:

Microsoft Visual Studio 디버그 콘솔

```
mem[107] = 0xbc2343dc
mem[108] = 0xcd3432cb
mem[109] = 0xde4521ba
mem[110] = 0xef5610a9
mem[111] = 0x670098
mem[112] = 0x189fe76
mem[113] = 0x129aed65
mem[114] = 0x23abdc54
mem[115] = 0x34bccb43
mem[116] = 0x45cdba32
mem[117] = 0x56dea921
mem[118] = 0x67ef9810
mem[119] = 0x78008700
mem[120] = 0x890176fe
mem[121] = 0x9a1265ed
mem[122] = 0xab2354dc
mem[123] = 0xbc3443cb
mem[124] = 0xcd4532ba
mem[125] = 0xde5621a9
mem[126] = 0xef671098
mem[127] = 0x780087
[Set 0]
V: 1 Tag: 0 Time: 2 Data: 0xffffffed(0) 0xfffffffe(1) 0x12(2) 0x1(3) 0xffffffdc(4) 0xffffffed(5) 0x23(6) 0x12(7)
V: 1 Tag: 0x8 Time: 4 Data: 0xffffffdc(0) 0xfffffffe(1) 0x23(2) 0x1(3) 0xffffffcb(4) 0xffffffed(5) 0x34(6) 0x12(7)
V: 1 Tag: 0x6 Time: 3 Data: 0x21(0) 0x32(1) 0xffffffde(2) 0xffffffcd(3) 0x10(4) 0x21(5) 0xffffffef(6) 0xffffffde(7)
V: 0 Tag: 0xffffffff Time: 0 Data: 0(0) 0(1) 0(2) 0(3) 0(4) 0(5) 0(6) 0(7)

C:₩Users₩소민₩source₩repos₩Project5₩Debug₩Project5.exe(프로세스 12080개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

(consol output)


access_output - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
[Accessed Data]
3	b	0x1
68	h	0xedcb
4	h	0xeddc
48	w	0xcdde3221
69	b	0xed
----------------------------------------
[Fully associative cache performance]
Hit ratio = 0.40 (2/5)
Bandwidth = 0.03 (10/305)
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

(file output)

2071035 이소민(Lee Somin)
2076235 안희재(An Heejae)

input file 2:

```
access_input - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
334	b
490	h
329	w
489	h
338	h
279	b
148	w
339	b
Ln 1, Col 1	100%	Unix (LF)	UTF-8
```

direct mapped:

```
Microsoft Visual Studio 디버그 콘솔
mem[110] = 0xef5610a9
mem[111] = 0x670098
mem[112] = 0x189fe76
mem[113] = 0x129aed65
mem[114] = 0x23abdc54
mem[115] = 0x34bccb43
mem[116] = 0x45cdba32
mem[117] = 0x56dea921
mem[118] = 0x67ef9810
mem[119] = 0x78008700
mem[120] = 0x890176fe
mem[121] = 0x9a1265ed
mem[122] = 0xab2354dc
mem[123] = 0xbc3443cb
mem[124] = 0xcd4532ba
mem[125] = 0xde5621a9
mem[126] = 0xef671098
mem[127] = 0x780087
[Set 0]
V: 0 Tag: 0xffffffff Time: 0 Data: 0(0) 0(1) 0(2) 0(3) 0(4) 0(5) 0(6) 0(7)
[Set 1]
V: 1 Tag: 0xf Time: 3 Data: 0xffffffdc(0) 0x54(1) 0x23(2) 0xffffffab(3) 0xffffffcb(4) 0x43(5) 0x34(6) 0xffffffbc(7)
[Set 2]
V: 1 Tag: 0xa Time: 7 Data: 0x54(0) 0xffffffba(1) 0xffffffab(2) 0x45(3) 0x43(4) 0xffffffa9(5) 0xffffffbc(6) 0x56(7)
[Set 3]
V: 0 Tag: 0xffffffff Time: 0 Data: 0(0) 0(1) 0(2) 0(3) 0(4) 0(5) 0(6) 0(7)

C:₩Users₩소민₩source₩repos₩Project5₩Debug₩Project5.exe(프로세스 14480개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

(consol output)

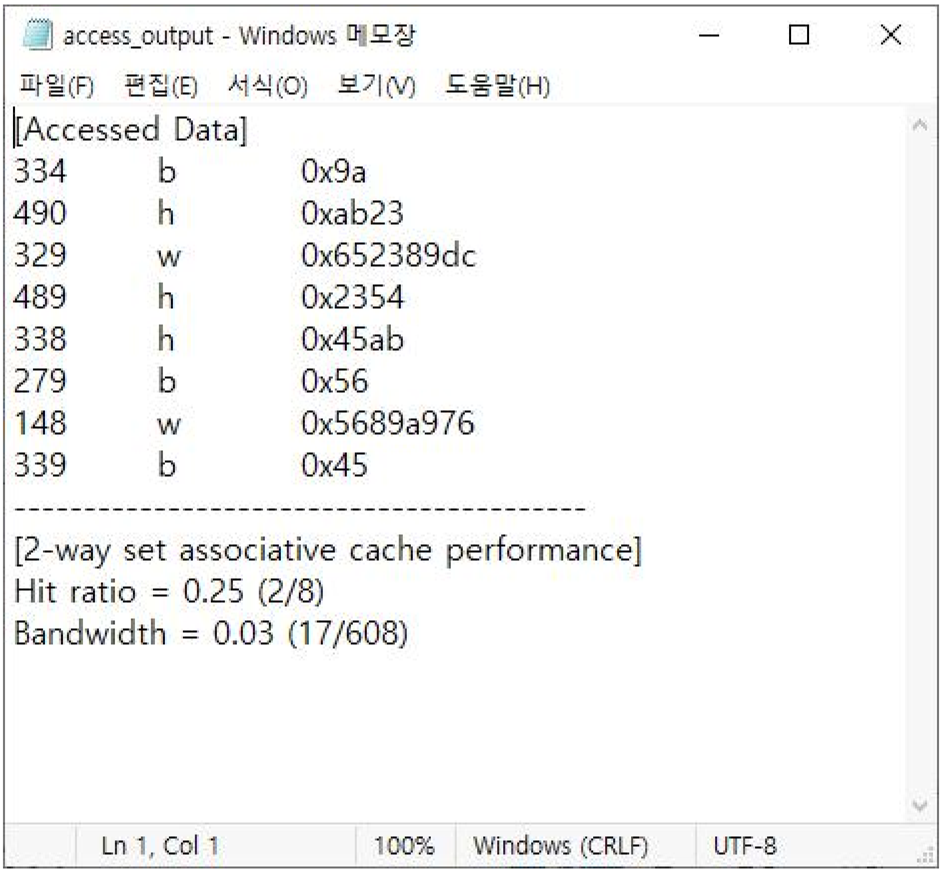
```
access_output - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
[Accessed Data]
334	b	0x9a
490	h	0xab23
329	w	0x652389dc
489	h	0x2354
338	h	0x45ab
279	b	0x56
148	w	0x5689a976
339	b	0x45
----------------------------------------
[Direct mapped cache performance]
Hit ratio = 0.00 (0/8)
Bandwidth = 0.02 (17/808)

Ln 1, Col 1	100%	Windows (CRLF)	UTF-8
```

(file output)

2-way set associative:

```
Microsoft Visual Studio 디버그 콘솔
mem[108] = 0xcd3432cb
mem[109] = 0xde4521ba
mem[110] = 0xef5610a9
mem[111] = 0x670098
mem[112] = 0x189fe76
mem[113] = 0x129aed65
mem[114] = 0x23abdc54
mem[115] = 0x34bccb43
mem[116] = 0x45cdba32
mem[117] = 0x56dea921
mem[118] = 0x67ef9810
mem[119] = 0x78008700
mem[120] = 0x890176fe
mem[121] = 0x9a1265ed
mem[122] = 0xab2354dc
mem[123] = 0xbc3443cb
mem[124] = 0xcd4532ba
mem[125] = 0xde5621a9
mem[126] = 0xef671098
mem[127] = 0x780087
[Set 0]
V: 1 Tag: 0x9 Time: 6 Data: 0xffffff87(0) 0xffffffba(1) 0x78(2) 0x45(3) 0x76(4) 0xffffffa9(5) 0xffffff89(6) 0x56(7)
V: 1 Tag: 0x15 Time: 7 Data: 0x54(0) 0xffffffba(1) 0xffffffab(2) 0x45(3) 0x43(4) 0xffffffa9(5) 0xffffffbc(6) 0x56(7)
[Set 1]
V: 1 Tag: 0x14 Time: 2 Data: 0x76(0) 0xffffffdc(1) 0xffffff89(2) 0x23(3) 0x65(4) 0xffffffcb(5) 0xffffff9a(6) 0x34(7)
V: 1 Tag: 0x1e Time: 3 Data: 0xffffffdc(0) 0x54(1) 0x23(2) 0xffffffab(3) 0xffffffcb(4) 0x43(5) 0x34(6) 0xffffffbc(7)

C:\Users\소민\source\repos\Project5\Debug\Project5.exe(프로세스 11500개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

(consol output)

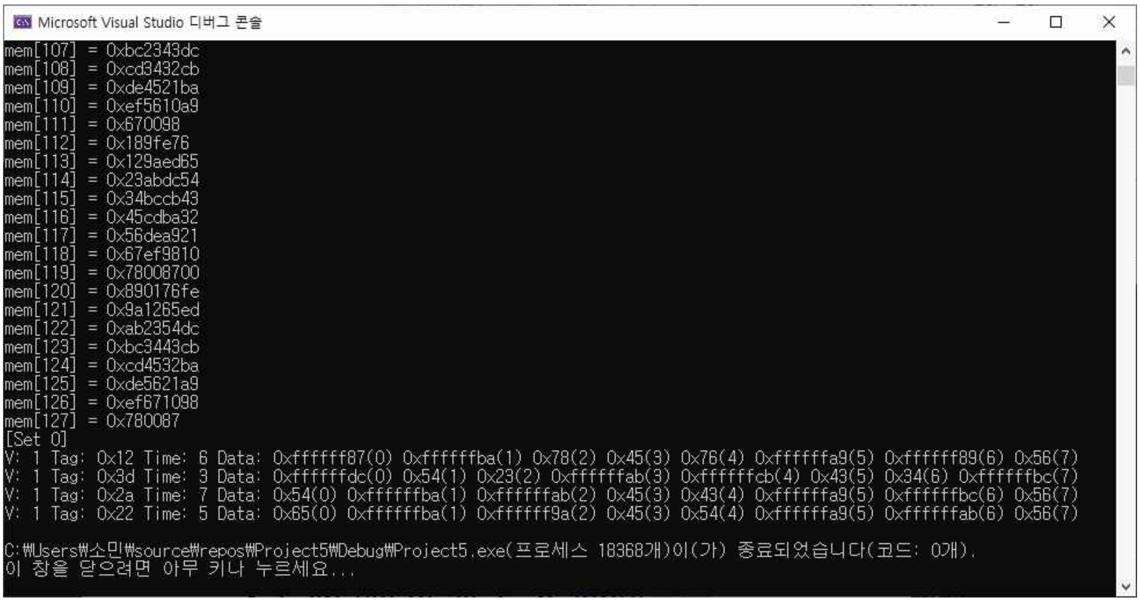

access_output - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
[Accessed Data]
334	b	0x9a
490	h	0xab23
329	w	0x652389dc
489	h	0x2354
338	h	0x45ab
279	b	0x56
148	w	0x5689a976
339	b	0x45
------------------------------------------
[2-way set associative cache performance]
Hit ratio = 0.25 (2/8)
Bandwidth = 0.03 (17/608)
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

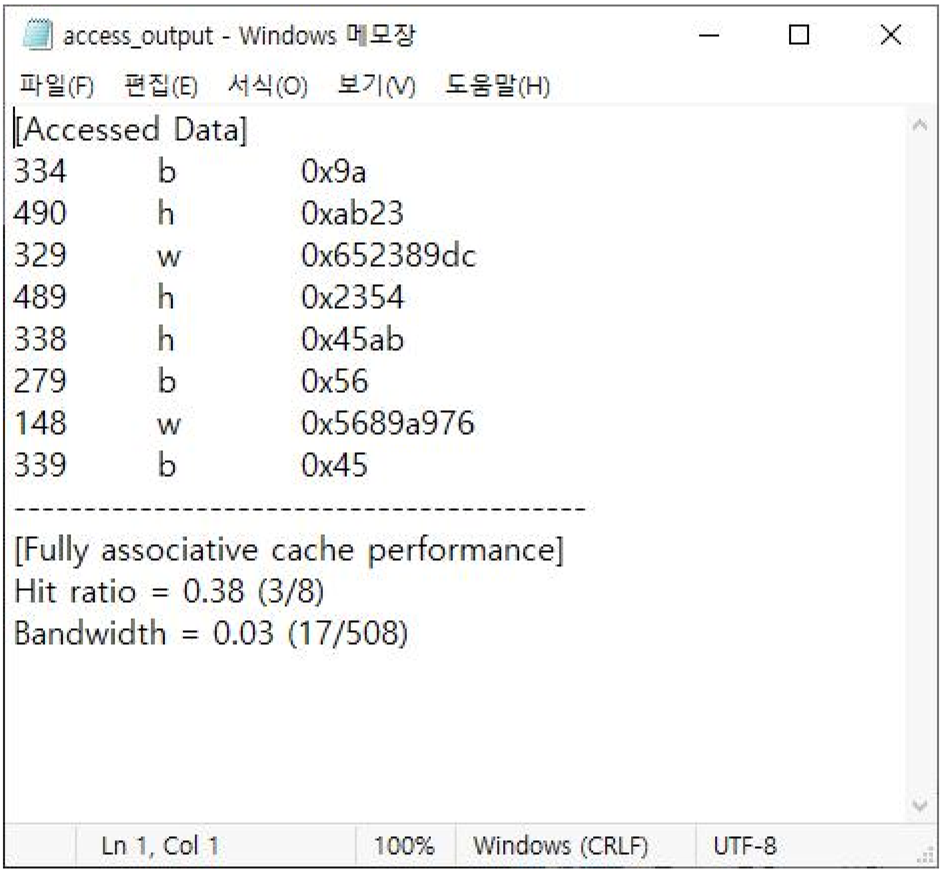
(file output)

Fully associative:


Microsoft Visual Studio 디버그 콘솔

```
mem[107] = 0xbc2343dc
mem[108] = 0xcd3432cb
mem[109] = 0xde4521ba
mem[110] = 0xef5610a9
mem[111] = 0x670098
mem[112] = 0x189fe76
mem[113] = 0x129aed65
mem[114] = 0x23abdc54
mem[115] = 0x34bccb43
mem[116] = 0x45cdba32
mem[117] = 0x56dea921
mem[118] = 0x67ef9810
mem[119] = 0x78008700
mem[120] = 0x890176fe
mem[121] = 0x9a1265ed
mem[122] = 0xab2354dc
mem[123] = 0xbc3443cb
mem[124] = 0xcd4532ba
mem[125] = 0xde5621a9
mem[126] = 0xef671098
mem[127] = 0x780087
[Set 0]
V: 1 Tag: 0x12 Time: 6 Data: 0xffffff87(0) 0xffffffba(1) 0x78(2) 0x45(3) 0x76(4) 0xffffffa9(5) 0xffffff89(6) 0x56(7)
V: 1 Tag: 0x3d Time: 3 Data: 0xffffffdc(0) 0x54(1) 0x23(2) 0xffffffab(3) 0xffffffcb(4) 0x43(5) 0x34(6) 0xffffffbc(7)
V: 1 Tag: 0x2a Time: 7 Data: 0x54(0) 0xffffffba(1) 0xffffffab(2) 0x45(3) 0x43(4) 0xffffffa9(5) 0xffffffbc(6) 0x56(7)
V: 1 Tag: 0x22 Time: 5 Data: 0x65(0) 0xffffffba(1) 0xffffff9a(2) 0x45(3) 0x54(4) 0xffffffa9(5) 0xffffffab(6) 0x56(7)

C:₩Users₩소민₩source₩repos₩Project5₩Debug₩Project5.exe(프로세스 18368개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

(consol output)




```
[Accessed Data]
334	b	0x9a
490	h	0xab23
329	w	0x652389dc
489	h	0x2354
338	h	0x45ab
279	b	0x56
148	w	0x5689a976
339	b	0x45
----------------------------------------
[Fully associative cache performance]
Hit ratio = 0.38 (3/8)
Bandwidth = 0.03 (17/508)
```

(file output)

5. Code Analysis

<cahe.c>

int check_cache_data_hit(void* addr, char type)

input: void* addr , char type

return: -1 , int value_returned

This function gets byte address (addr) and data type(type) for the input and returns -1 when there is no matching data in the cache_array or data fetched from cache(value_returned) when there is matching data in the cache.

When it is called it first casts the addr in to int form. Then, it adds cycle time to the global access cycle time(num_access_address). It calculates the block address by dividing the byte address with block size. Based on the block address the function calculates tag and set of the address. Then, it starts the 'for' loop for every entry of the set. If the cache entry's valid bit is 1 and the tag of the entry matches the tag calculated based on the block address, the time stamp of the entry is updated and byte offset is calculated with access_addr remainder when divided with the block size. Then, based on the data type, byte size is fixed inside the 'switch' statement. The global variable num_bytes are increased with the accessed byte size. Then, by inner 'for' loop, the data is copied to value_returned. In each cycle, the sign extension is erased with the 'and' operation with 0xff, and the value is appended to the result by shifting two digits in hexadecimal(8bit in binary) and executing 'or' operation with the digit to be added. Finally, value_returned is returned. If there is no such data in cache, -1 is returned.

int find_entry_index_in_set(int set)

input: int set

return: int i, 0, int entry_index

This function gets the set number(set) and returns the entry index(entry_index) to save the data to at next.

The function returns the index of not used entry of the set by first 'for' loop. If there is no empty entry in the set, it returns the index number of the least recently updated entry. However if the cache associative is direct mapped, the function returns the very first entry, 0.

int access_memory(void *addr, char type)

input: void* addr , char type

return: int -1 , int value_returned

This function gets byte address (addr) and data type(type) for the input and returns -1 when the address goes over the range of memory index or data fetched from memory(value_returned) when the address is appropriate.

When it is called it first casts the addr in to int form. Then, it adds cycle time to the global access cycle time(num_access_address). It calculates the block address by dividing the byte address with block size. Based on the block address the function calculates tag and set of the address. Then, the function calls 'find_entry_index_in_set()' to get the entry index(entry_index) to save the fetched data to. Then, it calculates the word address(word_address). If the word address goes over the size of the memory array, the function prints error message and returns -1. Otherwise, the function sets the cache entry's data preparing the data copy. It copies 2 digits of data each from memory array to each index of cache data array using remainder operation and shift right operation. When 'for' loop iterates for the half of the cache block size,

2071035 이소민(Lee Somin)
2076235 안희재(An Heejae)

mem_data is updated to the next element of the memory array to copy, since the block contains two words and a memory array element contains one word. The byte offset is calculated with access_addr remainder when divided with the block size. Then, based on the data type, byte size is fixed inside the 'switch' statement. The global variable num_bytes are increased with the accessed byte size. Then, by 'for' loop, the data is copied to value_returned. In each cycle, the sign extension is erased with the 'and' operation with 0xff, and the value is appended to the result by shifting two digits in hexadecimal(8bit in binary) and executing 'or' operation with the digit to be added. Finally, value_returned is returned.

<main.c>

int retrieve_data(void *addr, char data_type)

input: void* addr , char type

return: int value_returned

This function gets byte address (addr) and data type(type) for the input and returns -1 when there is no matching data in the cache nor memory, or data fetched from cache or memory (value_returned) when there is matching data.

When it is called it first initializes the return variable(value_returned) to -1. Then the function invokes the 'check_cache_data_hit()' to see if there is matching data in cache. If there is(if check_cache_data_hit() not returns -1), it increases the num_cache_hit and returns the return value. Otherwise, it is a miss event, so the function increases num_cache_misses and call 'access_memory()' to see if there is matching data in memory. If there is, the function returns the data value and if there isn't, the function returns -1.

int main(void)

In main function, input and output file is opened as ifp and ofp. While loops until there is no line to read. Inside while loop, a line is read to tmp. Then, the access address is set to the string tokenized until " ", casted to integer. Then, the inner while loops until it finds the character representing the three data type. Then, it fetches the data from cache or memory by calling retrieve_data(). When the data is fetched, the information are put to output file. When the while loop ends, association type is put in to the output file using switch statement. Finally, the program calculates the performance of the data access and print it to the output file and closes two file.