Worralop Srichainont (6632200221)

Assignment 02: Cache Design

Preparations

เตรียมไฟล์ทั้งหมด 5 ไฟล์ ดังนี้

CacheSim.h

```h
#ifndef CACHESIM_H_INCLUDED
#define CACHESIM_H_INCLUDED

#define MAX_BLOCK_SIZE 128

typedef unsigned char Byte;
typedef enum { LRU, RR } Replacement;

typedef struct {
    int valid;
    unsigned long tag;
    int age;       // For LRU
    int rr_ptr;    // For RR (used at set level)
    Byte data[MAX_BLOCK_SIZE];
} CacheLine;

typedef struct {
    CacheLine* lines;
} CacheSet;

extern int CACHE_SIZE;
extern int BLOCK_SIZE;
extern int ASSOCIATIVITY;
extern int NUM_SETS;
extern int OFFSETLEN;
extern int SETLEN;
extern Replacement REPLACEMENT_POLICY;

extern CacheSet* cache;
extern long HIT;
extern long MISS;

void initCache(int cacheKB, int blockSize, int associativity, Replacement policy);
void accessCache(unsigned long addr);
int log2int(int x);

#endif
```

CacheSim.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "CacheSim.h"

int CACHE_SIZE;
int BLOCK_SIZE;
int ASSOCIATIVITY;
int NUM_SETS;
int OFFSETLEN;
int SETLEN;
Replacement REPLACEMENT_POLICY;

CacheSet* cache;
long HIT = 0;
long MISS = 0;

int log2int(int x) {
    int r = 0;
    while (x >>= 1) r++;
    return r;
}

void initCache(int cacheKB, int blockSize, int associativity, Replacement policy) {
    CACHE_SIZE = cacheKB * 1024;
    BLOCK_SIZE = blockSize;
    ASSOCIATIVITY = associativity;
    REPLACEMENT_POLICY = policy;

    NUM_SETS = CACHE_SIZE / (BLOCK_SIZE * ASSOCIATIVITY);
    OFFSETLEN = log2int(BLOCK_SIZE);
    SETLEN = log2int(NUM_SETS);

    HIT = MISS = 0;

    cache = (CacheSet*)malloc(sizeof(CacheSet) * NUM_SETS);
    for (int i = 0; i < NUM_SETS; i++) {
        cache[i].lines = (CacheLine*)malloc(sizeof(CacheLine) * ASSOCIATIVITY);
        for (int j = 0; j < ASSOCIATIVITY; j++) {
            cache[i].lines[j].valid = 0;
            cache[i].lines[j].tag = 0;
            cache[i].lines[j].age = 0;
        }
        cache[i].lines[0].rr_ptr = 0;
    }
}
```

```c
void calAddr(unsigned long addr, unsigned long* tag, unsigned long* setIdx) {
    *tag = addr >> (OFFSETLEN + SETLEN);
    *setIdx = (addr >> OFFSETLEN) & ((1 << SETLEN) - 1);
}

void accessCache(unsigned long addr) {
    unsigned long tag, setIdx;
    calAddr(addr, &tag, &setIdx);

    CacheSet* set = &cache[setIdx];
    int hit = 0;

    for (int i = 0; i < ASSOCIATIVITY; i++) {
        if (set->lines[i].valid && set->lines[i].tag == tag) {
            HIT++;
            hit = 1;
            if (REPLACEMENT_POLICY == LRU)
                set->lines[i].age = 0;
        } else if (REPLACEMENT_POLICY == LRU) {
            set->lines[i].age++;
        }
    }

    if (hit) return;

    MISS++;
    int repl_index = 0;

    if (REPLACEMENT_POLICY == RR) {
        repl_index = set->lines[0].rr_ptr;
        set->lines[0].rr_ptr = (set->lines[0].rr_ptr + 1) % ASSOCIATIVITY;
    } else if (REPLACEMENT_POLICY == LRU) {
        int max_age = -1;
        for (int i = 0; i < ASSOCIATIVITY; i++) {
            if (!set->lines[i].valid) {
                repl_index = i;
                break;
            }
            if (set->lines[i].age > max_age) {
                max_age = set->lines[i].age;
                repl_index = i;
            }
        }
    }

    set->lines[repl_index].valid = 1;
    set->lines[repl_index].tag = tag;
    set->lines[repl_index].age = 0;
}
```

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "CacheSim.h"

int main(int argc, char* argv[]) {
    if (argc < 6) {
        fprintf(stderr, "Usage: %s <trace_file> <cache_kb> <block_size> <ways> <policy:
LRU|RR>\n", argv[0]);
        return 1;
    }

    const char* trace_file = argv[1];
    int cache_kb = atoi(argv[2]);
    int block_size = atoi(argv[3]);
    int associativity = atoi(argv[4]);
    Replacement policy = (strcmp(argv[5], "LRU") == 0) ? LRU : RR;

    initCache(cache_kb, block_size, associativity, policy);

    FILE* input = fopen(trace_file, "r");
    if (!input) {
        perror("Error opening trace file");
        return 2;
    }

    char line[1024];
    unsigned long addr;
    while (fgets(line, sizeof(line), input)) {
        if (strncmp(line, "0x", 2) != 0) continue;  // Skip headers
        if (sscanf(line, "0x%lx", &addr) == 1) {
            accessCache(addr);
        }
    }
    fclose(input);

    printf("=== Simulation Results ===\n");
    printf("Cache Size  : %d KB\n", cache_kb);
    printf("Block Size  : %d Bytes\n", block_size);
    printf("Associativity: %d-way\n", associativity);
    printf("Policy      : %s\n", (policy == LRU ? "LRU" : "RR"));
    printf("HIT         : %ld\n", HIT);
    printf("MISS        : %ld\n", MISS);
    printf("Hit Rate    : %.2f%%\n", 100.0 * HIT / (HIT + MISS));
    return 0;
}
```

ส่วนอีก 2 ไฟล์ที่ต้องใช้คือ

- gcc_ld_trace.txt
- go_ld_trace.txt

สามารถ download ได้จากลิงค์ดังกล่าว

https://www.cp.eng.chula.ac.th/~krerk/books/Computer%20Architecture/CacheSim/

หลังจากนั้นใช้คำสั่งดังกล่าวบน Terminal เพื่อ compile

```
gcc CacheSim.c main.c -o CacheSim -lm
```

จากนั้นใช้คำสั่งในนี้ในการ run

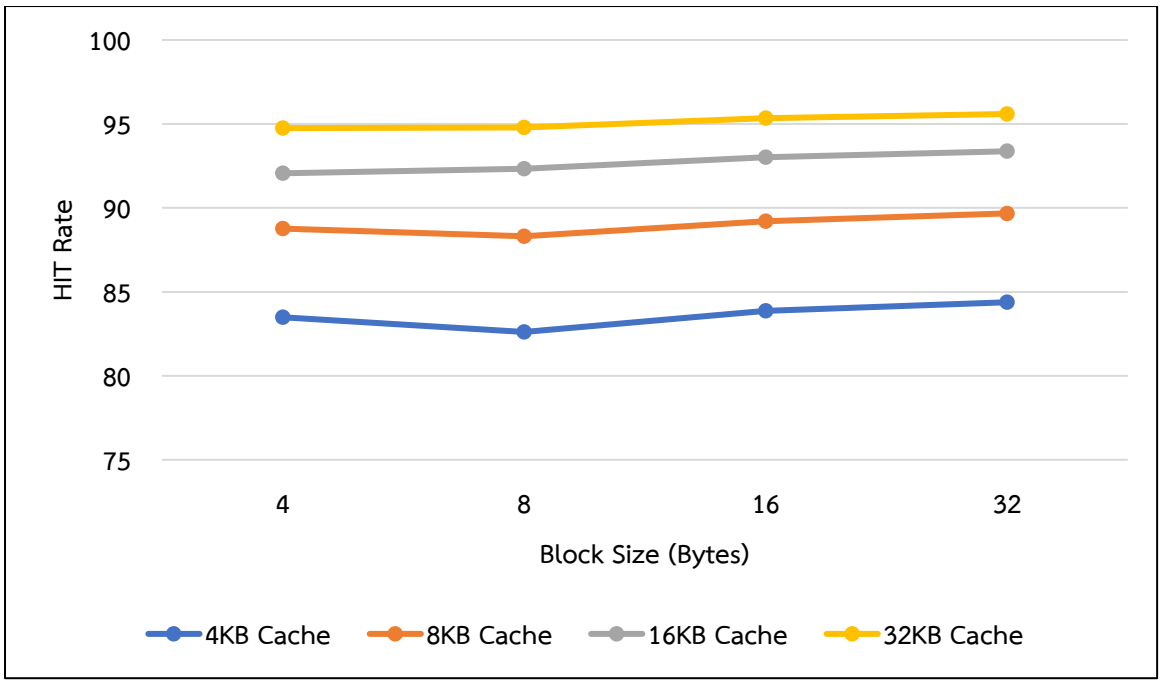
```
./CacheSim <trace_file> <cache> <associativity> <ways> <policy>
```

- **trace_file** คือ gcc_ld_trace.txt หรือ go_ld_trace.txt
- **cache** คือขนาด cache file ในหน่วย KB
- **block_size** คือ block size ในหน่วย B
- **associativity** ประกอบด้วย 1 (Direct mapped), 2 (Two-ways) และ 4 (Four-ways)
- **policy** ประกอบด้วย **LRU** (Least Recently Used) และ **RR** (Round Robin)

Part A: Block Size Tradeoff on direct mapped cache

Block Size (B)	Cache Size (KB)			
	4	8	16	32
4	HIT: 1,669,560 MISS: 330,440 HIT Rate: 83.48%	HIT: 1,775,198 MISS: 224,802 HIT Rate: 88.76%	HIT: 1,849,934 MISS: 150,066 HIT Rate: 92.06%	HIT: 1,895,088 MISS: 104,912 HIT Rate: 94.75%
8	HIT: 1,652,147 MISS: 347,853 HIT Rate: 82.61%	HIT: 1,766,215 MISS: 233,785 HIT Rate: 88.31%	HIT: 1,846,558 MISS: 153,442 HIT Rate: 92.33%	HIT: 1,895,885 MISS: 104,115 HIT Rate: 94.79%
16	HIT: 1,677,463 MISS: 322,537 HIT Rate: 83.87%	HIT: 1,784,151 MISS: 215,849 HIT Rate: 89.21%	HIT: 1,860,161 MISS: 139,839 HIT Rate: 93.01%	HIT: 1,906,747 MISS: 93,253 HIT Rate: 95.34%
32	HIT: 1,687,566 MISS: 312,434 HIT Rate: 84.38%	HIT: 1,793,431 MISS: 206,569 HIT Rate: 89.67%	HIT: 1,867,498 MISS: 132,502 HIT Rate: 93.37%	HIT: 1,911,848 MISS: 88,152 HIT Rate: 95.59%

Analysis



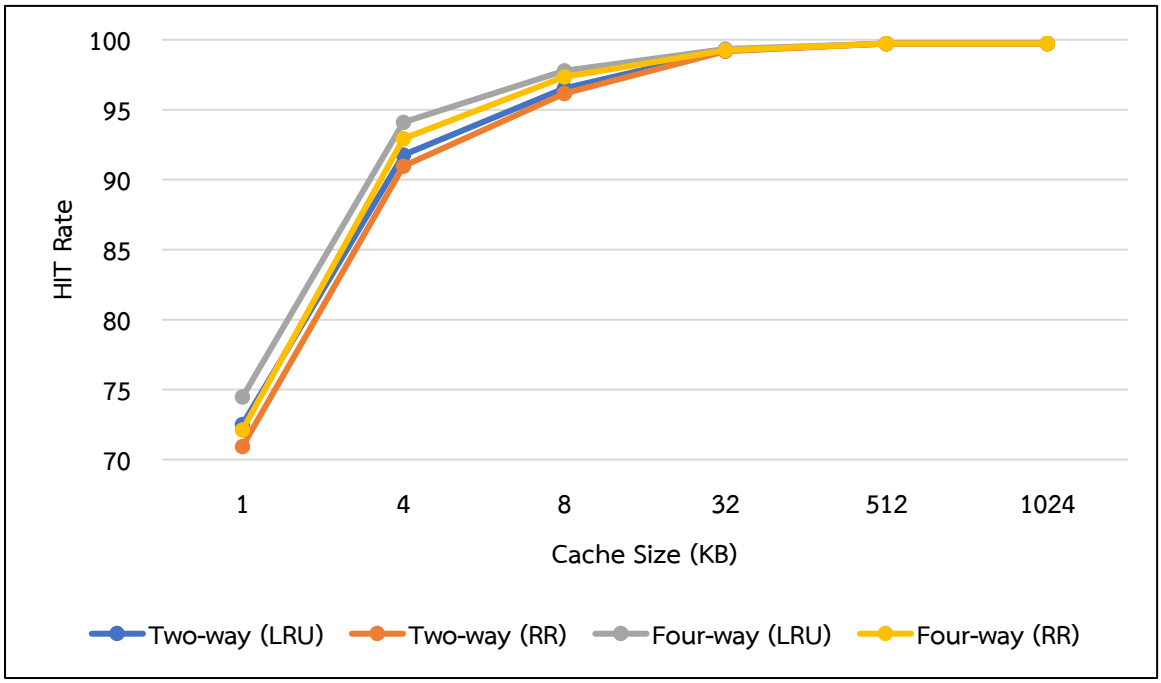
จากผลการทดลองดังกราฟข้างบน จะพบว่าค่า block size และ cache size ที่มากขึ้น ส่งผลให้ HIT Rate มากขึ้นตามไปด้วย เนื่องจากว่า cache size ที่มากขึ้น ส่งผลให้ storage capacity มากขึ้น ซึ่งทำให้ HIT Rate สูงขึ้นอย่างเห็นได้ชัด และ block size ที่มากขึ้น ส่งผลให้ HIT Rate สูงขึ้นเล็กน้อย เพราะว่าได้ใช้ประโยชน์จาก Spatial Locality

Part B: N-way associative cache with replacement algorithms: Least recently used (LRU), and Round Robin (RR).

การทดลองนี้ใช้ **go_ld_trace.txt** เป็น trace file และใช้ Block Size เท่ากับ **16 Bytes**

Cache Size (KB)	Two-way (LRU)	Two-way (RR)	Four-way (LRU)	Four-way (RR)
1	**HIT:** 1,087,611 **MISS:** 412,389 **HIT Rate:** 72.51%	**HIT:** 1,064,125 **MISS:** 435,875 **HIT Rate:** 70.94%	**HIT:** 1,117,107 **MISS:** 382,893 **HIT Rate:** 74.47%	**HIT:** 1,081,807 **MISS:** 418,193 **HIT Rate:** 92.94%
4	**HIT:** 1,376,563 **MISS:** 123,437 **HIT Rate:** 91.77%	**HIT:** 1,364,648 **MISS:** 135,352 **HIT Rate:** 90.98%	**HIT:** 1,411,710 **MISS:** 88,290 **HIT Rate:** 94.11%	**HIT:** 1,394,122 **MISS:** 105,878 **HIT Rate:** 92.94%
8	**HIT:** 1,447,733 **MISS:** 52,267 **HIT Rate:** 96.52%	**HIT:** 1,442,396 **MISS:** 57,604 **HIT Rate:** 96.16%	**HIT:** 1,466,878 **MISS:** 33,122 **HIT Rate:** 97.79%	**HIT:** 1,460,146 **MISS:** 39,854 **HIT Rate:** 97.34%
32	**HIT:** 1,488,461 **MISS:** 11,539 **HIT Rate:** 99.23%	**HIT:** 1,487,550 **MISS:** 12,450 **HIT Rate:** 99.17%	**HIT:** 1,489,895 **MISS:** 10,105 **HIT Rate:** 99.33%	**HIT:** 1,488,888 **MISS:** 11,112 **HIT Rate:** 99.26%
512	**HIT:** 1,495,944 **MISS:** 4,056 **HIT Rate:** 99.73%	**HIT:** 1,495,942 **MISS:** 4,058 **HIT Rate:** 99.73%	**HIT:** 1,495,947 **MISS:** 4,053 **HIT Rate:** 99.73%	**HIT:** 1,495,947 **MISS:** 4,053 **HIT Rate:** 99.73%
1024	**HIT:** 1,495,947 **MISS:** 4,053 **HIT Rate:** 99.73%	**HIT:** 1,495,947 **MISS:** 4,053 **HIT Rate:** 99.73%	**HIT:** 1,495,947 **MISS:** 4,053 **HIT Rate:** 99.73%	**HIT:** 1,495,947 **MISS:** 4,053 **HIT Rate:** 99.73%

Analysis



จากผลการทดลองดังกราฟข้างต้น จะพบว่าปัจจัย cache size, associativity และ replacement algorithm มีผลต่อ HIT Rate โดยที่ cache size มีผลต่อ HIT Rate อย่างชัดเจน โดยเฉพาะในจุดที่เพิ่ม cache size จาก 1KB เป็น 4KB แต่เมื่อเพิ่มขึ้นไปเรื่อยๆจะเกิด diminishing return สังเกตได้ว่าเมื่อเพิ่ม cache size จาก 512KB เป็น 1024KB ค่า HIT Rate แทบจะไม่เปลี่ยนแปลงเลย นอกจากนี้ associativity ก็มีผลเช่นกัน สังเกตได้ว่าประสิทธิภาพของ associativity แบบ four-way จะดีกว่า two-way เล็กน้อย เนื่องจากสามารถลด conflict misses ได้ ส่วน replacement algorithm ในการทดลองนี้พบว่า replacement algorithm แบบ Least Recently Used (LRU) ดีกว่า Round Robin (RR) เล็กน้อย เนื่องจากว่า Least Recently Used (LRU) จะพิจารณาความบ่อยในการใช้งาน แต่ Round Robin (RR) จะไม่สนใจในส่วนนี้ ซึ่งอาจมีบางกรณีที่ LRU สามารถทำงานได้ดีกว่า RR เช่นในผลการทดลองนี้