EEE3050 Theory on Computer Architectures (Spring 2017)
Prof. Jinkyu Jeong

HW3

2017.06.07

TA 이규선(GYUSUN LEE) / 안민우(MINWOO AHN)

What to do?

- Implement 2-way & 4-way set associative cache in C language
  - Write your code on main.c code in skeleton directory
  - Direct mapped cache is provided. Free to refer this code when implement your own code
  - You have to implement “Dirty Bit” in your cache
  - Use FIFO(First-in-First-out) mechanism to replacement policy
  - You should fill up your own read_op(), write_op(), flush() functions
- Environment
  - Recommend to implement your code on Ubuntu

Given Files

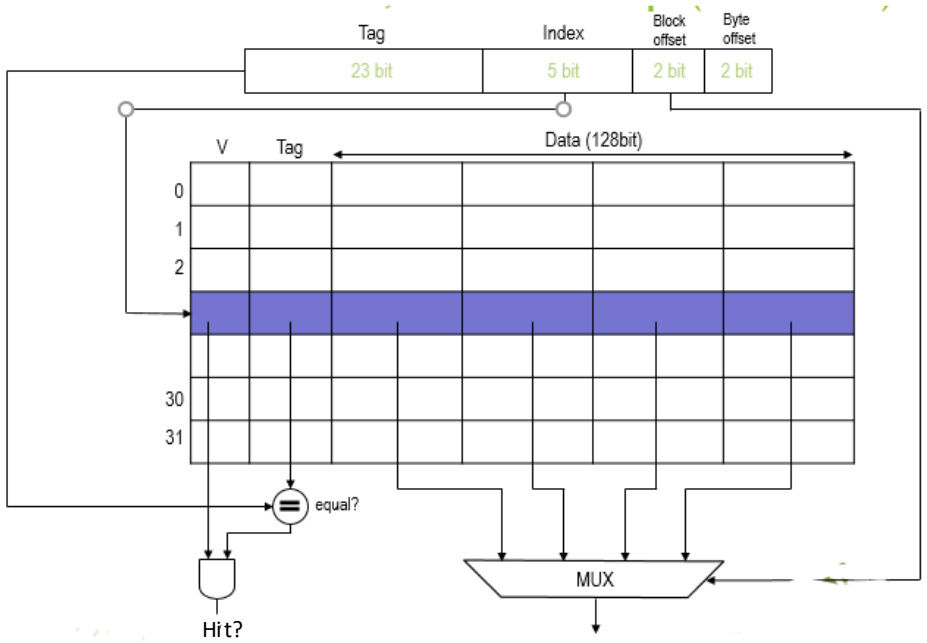
- direct_map
  - main.c : direct mapped cache is implemented in this code. Look read_op(), and write_op() function.
  - mem.h : declaration of functions, structures
- output
  - golden_cache_trace*.txt : golden result obtained by input file named *
- skelaton
  - You have to implement your 2-way & 4-way set associative cache in this directory

Input Files

- Input files are set of [read | write] operation with 30bit physical address
  - Read example, “R 1000101111110010101011011111111”
  - Write example, “W 1000101111110010101011011111111 !@#$”
  - !@#$ is 4 bytes string value which have to write
  - Every read and write operation requests 4 bytes size
  - 30bit physical address will be changed to integer in your skelaton

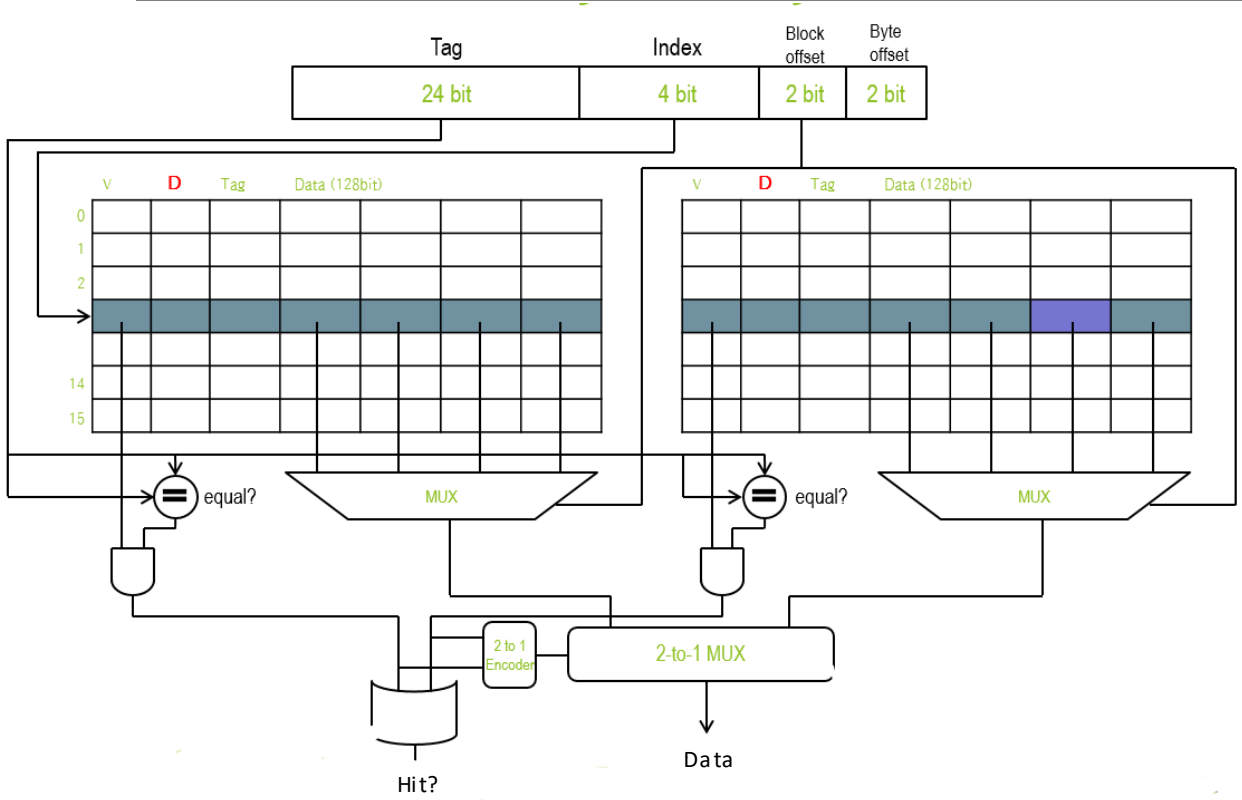
```
W 1000101111011111000111111111000 IJKL
R 1000101111011111000111000011000
R 1000101111011111000100100011000
W 1000101111011111000101010011000 efgh
W 1000101111011111000101100011000 CDEF
W 1000101111011111000100101010000 TUVW
R 1000101111011111000111111111000
R 1000101111011111000111010011000
W 1000101111011111000110110011000 WXYZ
W 1000101111011111000110011111000 QRST
R 1000101111011111000111010001000
R 1000101111011111000100011111000
R 1000101111011111000111111101000
W 1000101111011111000100100001000 RSTU
R 1000101111011111000111101010000
W 1000101111011111000110101110000 UVWX
W 1000101111011111000100000111000 MNOP
W 1000101111011111000110101010000 TUVW
W 1000101111011111000110111111000 YZAB
W 1000101111011111000100110010000 VWXY
```

The Big Picture – Direct Map



- Input file use 30bits(not 32bits) physical address.
- In direct map,
  - Address layout :
    - First 21 bits from MSB => tag
    - Next 5 bits => index in cache
    - Next 2 bits => block offset
    - Next 2 bits => byte offset (Not used)
  - Cache entry layout:
    - 1 bit for valid bit
    - 21 bits for tag
    - 32 * 4 bits for data block.

The Big Picture - 2-way(4-way)



- Input file use 30bits(not 32bits) physical address.
- # of ways, # of entries, # of blocks per entry, etc. are given as input.
- You should calculate properly with these variables

Replacement Policy : FIFO(First-in-First-out)

Direct Map Code

- Do not modify main function
- mem.h
  - cache_entry
  - cache
  - disk
  - read_op()
  - write_op()
  - flush()
  - Miscellaneous : Will not provided in your skelaton

```
1  #ifndef __mem__
2  #define __mem__
3
4  struct cache_entry {
5          char valid;
6          char dirty;     // You have to use this.
7          int tag;
8          char data[4][4];        //data[block_offset][byte_offset]
9  };
10
11 struct cache_entry cache[2][16];
12
13 char memory[1024 * 1024 * 256][4];
14
15 void read_op(char *);
16 void write_op(char *, char *);
17
18 /* Flush data in cache to disk*/
19 void flush(void);
20 void print_cache(FILE *);
21 int hw3_check(void);
22
23 /* Initialization before start simulation. */
24 void initialize(void);
25
26 /* Miscellaneous */
27 char *cut_a_to_b(char *, int, int);
28 int binary_to_dec(char *, int);
29 void n_copy(char *, char *, int);
30 int n_cmp(char *, char *, int);
31
32 #endif
```

mem.h

Direct Map Code

```
void read_op(char *addr){
        /* Direct Map Cache */
        int block_offset;
        int index;
        int tag;
        int address;
        struct cache_entry *entry_of_addr;

        block_offset = binary_to_dec(cut_a_to_b(addr, 26, 27), 2);
        index = binary_to_dec(cut_a_to_b(addr, 21, 25), 5);
        tag = binary_to_dec(cut_a_to_b(addr, 0, 20), 21);
        address = binary_to_dec(addr, 30);
        entry_of_addr = &cache[index / 16][index % 16];

        if(entry_of_addr -> valid == 0 || entry_of_addr -> tag != tag){ //cache miss
                miss_count++;

                if(entry_of_addr -> valid && entry_of_addr -> tag != tag){      //wrong tag => write data to memory from cache
                        n_copy(memory[(entry_of_addr -> tag * 512 + index * 16 + 0) / 4], entry_of_addr -> data[0], 4);
                        n_copy(memory[(entry_of_addr -> tag * 512 + index * 16 + 4) / 4], entry_of_addr -> data[1], 4);
                        n_copy(memory[(entry_of_addr -> tag * 512 + index * 16 + 8) / 4], entry_of_addr -> data[2], 4);
                        n_copy(memory[(entry_of_addr -> tag * 512 + index * 16 + 12) / 4], entry_of_addr -> data[3], 4);
                }
                //load data to cache from memory
                n_copy(entry_of_addr -> data[0], memory[address / 16 * 4 + 0], 4);
                n_copy(entry_of_addr -> data[1], memory[address / 16 * 4 + 1], 4);
                n_copy(entry_of_addr -> data[2], memory[address / 16 * 4 + 2], 4);
                n_copy(entry_of_addr -> data[3], memory[address / 16 * 4 + 3], 4);
                entry_of_addr -> valid = 1;
                entry_of_addr -> tag = tag;
        }
        else if(entry_of_addr -> valid == 1 && entry_of_addr -> tag == tag){    //cache hit
                hit_count++;

        }

}
```

Write data on cache entry to appropriate address of disk

Load data from disk

Direct Map Code

```
void write_op(char *addr, char *write){
        /* Direct Map Cache */
        int block_offset;
        int index;
        int tag;
        int address;
        struct cache_entry *entry_of_addr;

        block_offset = binary_to_dec(cut_a_to_b(addr, 26, 27), 2);
        index = binary_to_dec(cut_a_to_b(addr, 21, 25), 5);
        tag = binary_to_dec(cut_a_to_b(addr, 0, 20), 21);
        address = binary_to_dec(addr, 30);
        entry_of_addr = &cache[index / 16][index % 16];

        if(entry_of_addr -> valid == 0 || entry_of_addr -> tag != tag){ //cache miss
                miss_count++;

                if(entry_of_addr -> valid && entry_of_addr -> tag != tag){
                        n_copy(memory[(entry_of_addr -> tag * 512 + index * 16 + 0) / 4], entry_of_addr -> data[0], 4);
                        n_copy(memory[(entry_of_addr -> tag * 512 + index * 16 + 4) / 4], entry_of_addr -> data[1], 4);
                        n_copy(memory[(entry_of_addr -> tag * 512 + index * 16 + 8) / 4], entry_of_addr -> data[2], 4);
                        n_copy(memory[(entry_of_addr -> tag * 512 + index * 16 + 12) / 4], entry_of_addr -> data[3], 4);
                }
                n_copy(entry_of_addr -> data[0], memory[address / 16 * 4 + 0], 4);
                n_copy(entry_of_addr -> data[1], memory[address / 16 * 4 + 1], 4);
                n_copy(entry_of_addr -> data[2], memory[address / 16 * 4 + 2], 4);
                n_copy(entry_of_addr -> data[3], memory[address / 16 * 4 + 3], 4);
                entry_of_addr -> valid = 1;
                entry_of_addr -> tag = tag;
                n_copy(entry_of_addr -> data[block_offset], write, 4);
        }
        else if(entry_of_addr -> valid == 1 && entry_of_addr -> tag == tag){    //cache hit
                hit_count++;

                n_copy(entry_of_addr -> data[block_offset], write, 4);
        }

}
```

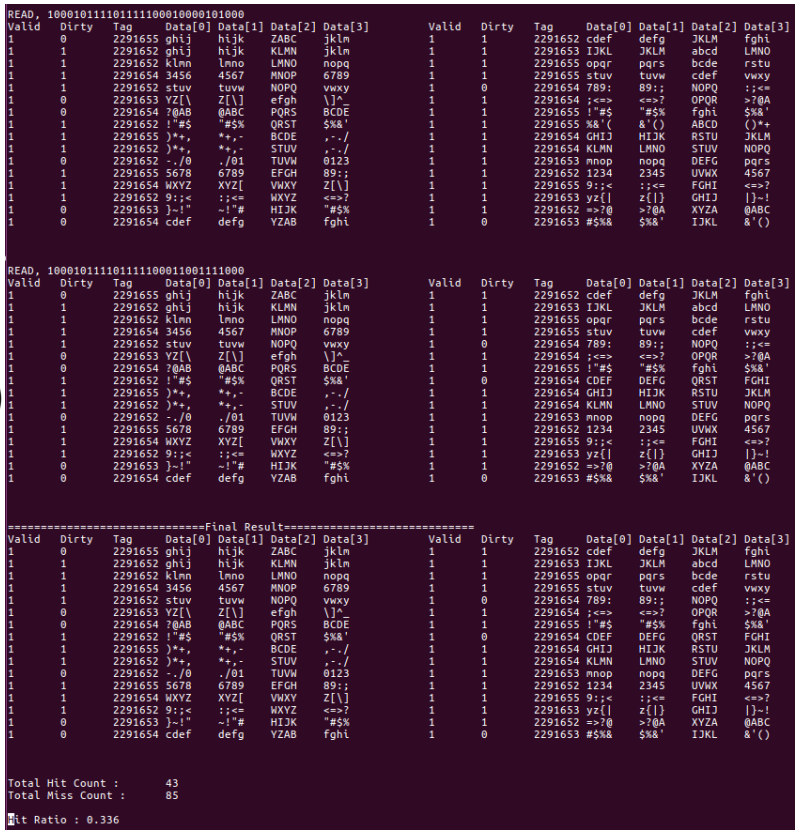
Direct Map Code

```
/* Write data in cache to memory */
void flush(){
        int i;
        struct cache_entry *c1, *c2;

        for(i = 0; i < 16; i++){
                c1 = &cache[0][i];
                c2 = &cache[1][i];
                n_copy(memory[(c1 -> tag * 512 + i * 16 + 0) / 4], c1 -> data[0], 4);
                n_copy(memory[(c1 -> tag * 512 + i * 16 + 4) / 4], c1 -> data[1], 4);
                n_copy(memory[(c1 -> tag * 512 + i * 16 + 8) / 4], c1 -> data[2], 4);
                n_copy(memory[(c1 -> tag * 512 + i * 16 + 12) / 4], c1 -> data[3], 4);
                n_copy(memory[(c2 -> tag * 512 + (i + 16) * 16 + 0) / 4], c2 -> data[0], 4);
                n_copy(memory[(c2 -> tag * 512 + (i + 16) * 16 + 4) / 4], c2 -> data[1], 4);
                n_copy(memory[(c2 -> tag * 512 + (i + 16) * 16 + 8) / 4], c2 -> data[2], 4);
                n_copy(memory[(c2 -> tag * 512 + (i + 16) * 16 + 12) / 4], c2 -> data[3], 4);
        }
}
```

Execute Code

- (Linux)
  - To run direct mapped cache, type $make command under direct_map directory
  - “direct_map” executable file will be made
  - Run and enter input file name
  - Check your result from output file name “cache_trace_*.txt” (* is input file name)
  - Same as execute your skeleton code



Submission

- Compress your main.c and mem.h codes as “YourStudentID.zip”(Don’t change file name)
  - Without subdirectories
  - YourStudentID.zip
  - TA will not grade your assignment if YOU NOT FOLLOW THIS FORMAT.
- Upload your zip file to I-Campus Assignments bulletin
- PLEASE DO NOT COPY. YOU WILL GET -100 POINTS IF YOU COPIED.
- Due date: June 19th, 24:00

Follow this format!!

Grade Policy

Grade Policy :

1. “[Error!!]” is printed => -20 points

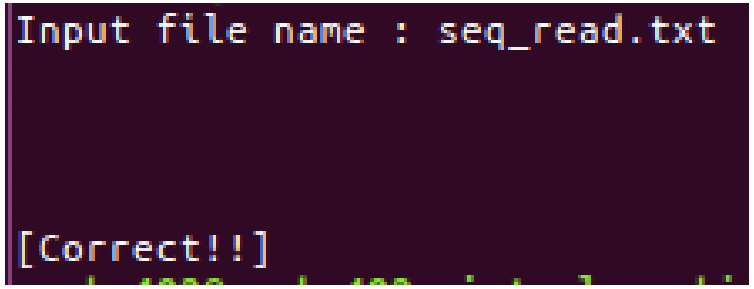
2. Output file is differ from golden result
    2-1. seq_read.txt => -40 points
    2-2. seq_write.txt => -40 points
    2-3. others => -20 points each

3. No comments of each functions(read_op(), write_op(), flush()) : -5 points each

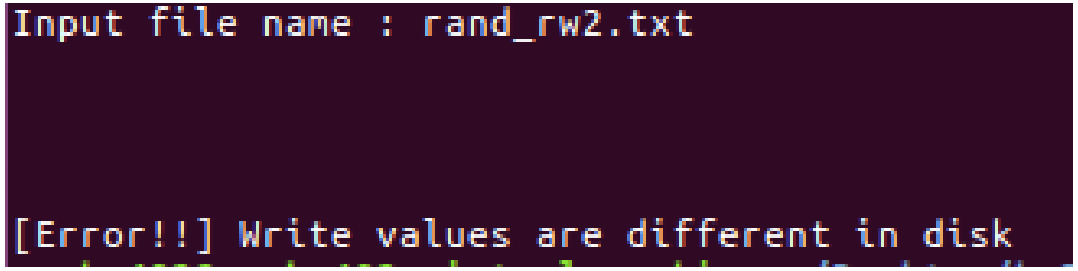
- Penalty for late submission : -10% per every 24 hours

TIPS

- How to know my code “work correctly”?
  - Your simulation outputs should be same with “golden results”
  - And, after executed your code, “[correct]” should be printed
- Lecture notes will help your assignment



Working correctly



Working wrong

Tutorial

- There will be analyze of direct map code session on June 7th 20:00 in semiconductor building(#400126)

- Attendance is not mandatory, it will be videotaped

Questions

- If you have “any” questions, please upload it to i-Campus Q&A bulletin.
  - This is for sharing your questions with others.
- You can also visit Semiconductor Building #400509(Please e-mail TA before visiting)