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

HW3

2017.06.07

TA 이규선(GYUSUN LEE) / 안민우(MINWOO AHN)

mem.h

```c
#ifndef __mem__
#define __mem__

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

struct cache_entry {
        char valid;
        int tag;
        int dirty;
        char **data;
        /* You can add variables for your purpose  */

};

struct cache {
        struct cache_entry **way_cache;
        int num_entry;
        int num_ways;
        int bytes_per_block;    // # of bytes / block
        int block_per_entry;    // # of block / entry
};

struct cache cache;
char memory[1024 * 1024 * 256][4];

char *read_op(int);
void write_op(int, char *);

/* Flush data in cache to disk*/
void flush(void);
void print_cache(FILE *);
int hw3_check(void);

/* Initialization before start simulation. */
void initialize(void);

#endif
```

- struct cache_entry
  - valid bit
  - dirty bit
  - tag
  - data[block_offset][byte_offset]
  - **You can add new variables for your own purpose!**
- struct cache
  - num_entry, num_ways, block_per_entry => inputs
  - bytes_per_block => 4 (constant)
  - way_cache[way_offset][entry_offset]

main.c – main()

```
/* Get Input */
printf("====================Welcome to Cache Simulation!====================\n");
printf("Enter # of Ways : ");
scanf("%d", &cache.num_ways);
printf("Enter # of Entries : ");
scanf("%d", &cache.num_entry);
printf("Enter # of blocks per entry : ");
scanf("%d", &cache.block_per_entry);

cache.bytes_per_block = 4;
```

- Input : num_ways, num_entry, block_per_entry
- Constant : bytes_per_block = 4
- Possible input set [num_ways, num_entry, block_per_entry]
  - [1, 32, 4], [1, 32, 8], [2, 16, 4], [2, 16, 8], [4, 4, 4], [4, 4, 8]

main.c – initialize()

```c
void initialize(){
        int i, j, k, l;

        printf("Initialize Your Cache...\n");
        /* cache[num_ways][num_entry]*/
        cache.way_cache = (struct cache_entry **)malloc(sizeof(struct cache_entry *) * cache.num_ways);

        for(i = 0; i < cache.num_ways; i++)     cache.way_cache[i] = (struct cache_entry *)malloc(sizeof(struct cache_entry) * cache.num_entry);

        for(i = 0; i < cache.num_ways; i++){
                for(j = 0; j < cache.num_entry; j++){
                        cache.way_cache[i][j].data = (char **)malloc(sizeof(char *) * cache.block_per_entry);

                        for(k = 0; k < cache.block_per_entry; k++)      cache.way_cache[i][j].data[k] = (char *)malloc(sizeof(char) * cache.bytes_per_block);

                        /* Initialize other variables in your cache_entry structure! */
                        /* e.g. valid, dirty, tag, etc.                              */
                        cache.way_cache[i][j].valid = 0;
                        cache.way_cache[i][j].tag = -1;
                        cache.way_cache[i][j].dirty = 0;
                        ////////////////////////////////////////////////////////////
                }
        }

        /* Disk Data Initialization */
        for(i = 0; i < 1024 * 1024 * 256; i++){
                memory[i][0] = i % 94 + 33;
                memory[i][1] = (i + 1) % 94 + 33;
                memory[i][2] = (i + 2) % 94 + 33;
                memory[i][3] = (i + 3) % 94 + 33;
        }

        printf("Finished to Initialize Your Cache!\n");
}
```

You have to initialize variables which you added in struct cache_entry!

main.c – read_op(), write_op, flush()

```
char *read_op(int addr){
        /* You have to implement your own read_op function here! */


        ////////////////////////////////////////////////////////////////////////////
        return "NONE";
}

void write_op(int addr, char *write){
        /* You have to implement your own write_op function here! */


        ////////////////////////////////////////////////////////////////////////////
}

/* Write data in cache to disk */
void flush(){
        /* You have to implement your own flush function using dirty bits here! */


        ////////////////////////////////////////////////////////////////////////////
}
```

inputs

- seq_read.txt, seq_write.txt, seq_rw.txt, rand_rw1.txt, rand_rw2.txt, rand_rw3.txt
- Format : 2000 operations per file, there is H(HALT instruction) at the end of file

```
W 586664100 nVNH
W 586667112 2xpj
R 586663696
R 586665144
R 586665924
W 586663560 ~f^X
W 586664216 >&|v
W 586664408 `H@:
R 586666788
W 586663676 N6.(
R 586663996
W 586665576 <$zt
W 586666836 6|tn
R 586663764
R 586664092
R 586666316
R 586666232
W 586666608 ZB:4
W 586666192 ~f^X
R 586663784
R 586665372
R 586664412
R 586665596
R 586663488
R 586663944
R 586667276
R 586666344
R 586664680
R 586664896
W 586664280 *phb
H
```