CS2100 Computer Organisation

Lab 06: Direct Mapped Cache Simulator (Week 12th October)

Instruction

Short and clean
We have separated the lab information into i) **instruction** and ii) **report**. Whenever there is a question in the instruction (easily identified as they have **[X pts]** tagged to the end), write / type your answer in the corresponding location in the **report** document. Please take note of the submission specification at the end of this document.

Objective

In this lab, you will complete a simple Cache Simulator written in C. There are 3 objectives for this lab:

1. A quick revision of C programming.
2. By completing the key logic, you get to understand the Direct Mapped Cache better.
3. With the completed cache simulator, you can use it to verify cache behaviors.

Please note that this lab is challenging, especially if you have forgotten most of C 😊.
Complete as much as you can before attending the lab session.

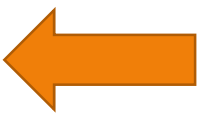
Task 1: Basic Understanding [7 marks]

We will use the cache access example from "Lecture #13 – Cache – DM", the parameters of the cache are:

- Direct Mapped
- 16Kbytes capacity with 16 Bytes cache block (i.e. 1024 cache blocks)

Given the parameters, compute the number of bits for the following fields given a 32-bit memory address **[3 pts]**:

Tag	Cache Index	Offset
18 bits	10 bits	4 bits



Suppose we store the memory address as an **unsigned integer** in C, e.g.

unsigned int *address* = 0x12345678

(Note: Unsigned integer stores non-negative integer)

Give two different C programming statement to calculate the various fields from ***address*:**

a. **Arithmetic version:** Using addition/subtraction/division / modulo / multiplication

b. **Bit-wise operation version:** Using "**&**" (AND), "**|**"(OR), "**<<**" (Shift left), "**>>**" (Shift right), etc (take a look at lecture #7 Slide "Logical Operation", under "C Language"). Note: For simplicity you can pre-calculate some intermediate values using arithmetic and predefined math functions to help.

You can assume the following values are given:

- **blockSize** = Cache block size in bytes
- **nCacheBlock** = Number of cache block

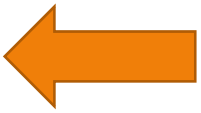
The "offset" field has already been completed as example:

- Offset

a.	`offset = address % blockSize;`
b.	`offset = address & (blockSize – 1);`

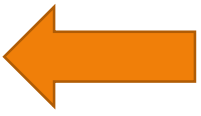
- Cache Index **[2 pts]**

a.	`cacheIdx =`
b.	`cacheIdx =`



- Tag **[2 pts]**

a.	`tag =`
b.	`tag =`



Task 2: Let's complete the simulator! (`DMCache_Simulator.c`) [8 marks]

Take a look at the given C program `DMCache_Simulator.c`. The main flow of the program is:

a. Initialize cache from command line argument.
b. Read memory address MA from user.
    i. Access cache with MA, check for cache hit / miss
    ii. Update cache information accordingly
c. Report the overall cache statistics (Number of accesses, number of hit, percentage of hit)

Nearly all core logics have been written, except part (b). All key logic for that part has been encapsulated in the function ***accessCache***(). You can locate the portions you need to work on by looking for "//TODO" comments in that function.

Your task is to complete the ***accessCache*()** function. In particular:

a. Calculate the various fields needed to access the cache. Use Task 1's answer to help.
b. Specify the "cache hit" condition in the if-statement.
c. Update the cache information after a cache miss.

Note that you only need ~6-7 lines of C code **in total, including the partial statements given for part (a).**

Below is a sample user session (user input in **bold;** comment in ***red italics***):

```
Prompt> DMCache_Simulator.exe 16 16   //Command line argument: Total Size (in KB), Block Size (in Byte)
Direct Mapped Cache [16384 Byte(s) total capacity | 16 Byte(s) block size]
14                          //User input: this is read as an hexadecimal value, i.e. 0x00000014
Address[0x00000014] = [Tag:0x0|Idx:0x1|Offset:0x4] => Miss
0x1c                        //Hexadecimal input can have the prefix "0x" too
Address[0x0000001c] = [Tag:0x0|Idx:0x1|Offset:0xc] => Hit!
34
Address[0x00000034] = [Tag:0x0|Idx:0x3|Offset:0x4] => Miss
8018
Address[0x00008018] = [Tag:0x2|Idx:0x1|Offset:0x8] => Miss
10
Address[0x00000010] = [Tag:0x0|Idx:0x1|Offset:0x0] => Miss
```

```
^D                          //Terminate input by [Ctrl-D] then enter
Total Access = 5
Cache Hit = 1, 20.0000%
```

Note that this is the same set of cache accesses illustrated in lecture #12, slide 32-53.

Redirection To facilitate testing

Since it is quite cumbersome (and slow) to key the memory address by hand, you can **redirect** the input from a file instead:

```
Prompt> DMCache_Simulator.exe 16 16  < DM_LectureSample.txt  //note the "<", the "DM_LectureSample.txt" contains the memory accesses
Address[0x00000014] = [Tag:0x0|Idx:0x1|Offset:0x4] => Miss
Address[0x0000001c] = [Tag:0x0|Idx:0x1|Offset:0xc] => Hit!
Address[0x00000034] = [Tag:0x0|Idx:0x3|Offset:0x4] => Miss
Address[0x00008018] = [Tag:0x2|Idx:0x1|Offset:0x8] => Miss
Address[0x00000010] = [Tag:0x0|Idx:0x1|Offset:0x0] => Miss
Total Access = 5
Cache Hit = 1, 20.0000%
```

Using the redirection "<" symbol, the simulator will read directly from a file instead of keyboard.

Demonstrate the finished program to your lab TA. Your lab TA will test your program using the "`SPECINT-401-1K-Access.dat`" as input **[8 pts]**

You do not need to submit any program.
Please do not send your programs to your labTA after the lab; they will not be accepted.

Ok, that was fun, what else? You can see that the direct mapped cache is really quite simple. You can explore how to write a simulator for set-associative cache. Other than that, use the DMCache simulator to study the effect of cache parameters, e.g. if we have bigger block, what's the impact on cache hit percentage? etc.

Marking Scheme: Report – 7 marks; Demonstration – 8 marks; Total: 15 marks.