CS2100 Computer Organisation

Lab 07: 2-Way SA Cache with "Not Me" policy (Week 17th 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, we will explore 2-way set associative cache with an interesting replacement policy. The objectives are similar to lab 06:

1. Another chance for you to exercise C programming.
2. By completing the key logic and observing the code, you get to understand the Set-Associative Cache better.
3. You get to see a block replacement policy in action.

Please note that this lab is challenging, especially if you still hates C after lab 06 😉.
Complete the coding before your lab!

Task 1: Basic Understanding [9 marks]

Suppose we have the following cache setup:

- 2-Way Set Associative Cache
- 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	Set Index	Offset
19 bits	9 bits	4 bits

As discussed in lecture, when all valid cache locations are occupied, SA and FA cache uses **block replacement policy** to select a **victim block** for eviction. Let us study an example of 2-way SA cache:

SetIdx	**Location 0 ("left")**			**Location 1 ("right")**		
	Valid	**Tag**	**Data**	**Valid**	**Tag**	**Data**
……….	**…………….**			**…………….**		
1	**1**	**AAAA**	**A**	**1**	**BBBB**	**B**
……….	**…………….**			**…………….**		

For ease of reference, the two locations in a cache set is call the "***left*** location" (or the "location 0") and "***right*** location" (or the "location 1"). In the illustration above, the left location is occupied by block with tag "AAAA", the right location has block "BBBB".

Now, if block "CCCC" come along and mapped to the same set, we need to select either the left or right location to evict, this is known as block replacement policy.

Suppose we use the following policy (a variant of "**Not Me!**" policy):

- Each ***<u>Set</u>*** has a "`NotLeft`" flag. This flag indicate whether the left location should be chosen as the victim block. If "`NotLeft`" is **true**, then the victim is the **right block**; if "`NotLeft`" is **false**, then the victim is the **left block**.
- "`NotLeft`" is initialized to **false** at the start.
- If a block is loaded into the left location the "`NotLeft`" will be set to **true**.
- If a block is loaded into the right location the "`NotLeft`" will be set to **false**.
- If the block on the left location is **accessed** (ie. cache hit), the "`NotLeft`" will be set to **true**.
- If the block on the right location is **accessed** (ie. cache hit), the "`NotLeft`" will be set to **false**.

Check your understanding by answering the following. We start with **empty cache set S** with "`NotLeft`" is initialized to **false** at the start. The memory blocks given are all mapped to the same **cache set S**. The first two accesses are shown as example. **[6 pts]**

Block Accessed (in sequence)	Cache Miss?	Block replaced (if applicable)
i. AAAA	~~Hit~~ / **Miss**	**Left** / ~~Right / Not applicable~~
ii. BBBB	~~Hit~~ / **Miss**	~~Left~~ / **Right** / ~~Not applicable~~
iii. CCCC	Hit / [✓ Miss]	[✓ Left] / Right / Not applicable
iv. BBBB	[✓ Hit] / Miss	Left / Right / [✓ Not applicable]
v. DDDD	Hit / [✓ Miss]	[✓ Left] / Right / Not applicable

NotLeft
F
T
F
T
F

What do you think is the formal name of this policy? (Remove incorrect answers) **[1 pt]**

Least Recently Used

Least Frequently Used

Random Replacement

First in first out

[✓ First in first out with Second Chance]

Can we use the sample setup (i.e. one flag) for 4-way or more SA cache? **[1 pt]**

Why (this is not marked, but make sure you can explain)?

Yes / [✓ No]

Task 2: Let's complete the simulator! (`NotMe_Cache_Simulator.c`) [4 marks]

Take a look at the given C program `NotMe_Cache_Simulator.c`. The main flow of the program is the same as the DM Cache simulator from lab 06:

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)

As the cache is slightly more complicated than DM Cache, most of the logic has been coded. ☺ Take a moment to read and learn. Your task is to **update the "NotLeft" flag** in ***accessCache***(). You can locate the portions you need to work on by looking for "//TODO" comments in that function.

Below is a sample user session using the **redirection technique** used in lab 06:

```
Prompt> NotMe_Cache_Simulator.exe 16 16  < NotMe_Sample.txt
2-Way SA Cache [16384 Byte(s) total capacity | 16 Byte(s) block size | 512 Set(s)]
Address[0x00000014] = [Tag:0x0|SetIdx:0x1|Offset:0x4] => Miss [Replaced 0]
Address[0x0000001c] = [Tag:0x0|SetIdx:0x1|Offset:0xc] => Hit [LEFT] !
Address[0x00ab001e] = [Tag:0x558|SetIdx:0x1|Offset:0xe] => Miss [Replaced 1]
Address[0x00ab0010] = [Tag:0x558|SetIdx:0x1|Offset:0x0] => Hit [RIGHT] !
Address[0x00de0014] = [Tag:0x6f0|SetIdx:0x1|Offset:0x4] => Miss [Replaced 0]
Address[0x00000014] = [Tag:0x0|SetIdx:0x1|Offset:0x4] => Miss [Replaced 1]
Address[0x00ab0010] = [Tag:0x558|SetIdx:0x1|Offset:0x0] => Miss [Replaced 0]
Address[0x00ca0018] = [Tag:0x650|SetIdx:0x1|Offset:0x8] => Miss [Replaced 1]
Address[0x00000014] = [Tag:0x0|SetIdx:0x1|Offset:0x4] => Miss [Replaced 0]
Address[0x00ca0010] = [Tag:0x650|SetIdx:0x1|Offset:0x0] => Hit [RIGHT] !
Total Access = 10
Cache Hit = 3, 30.0000%
```

Note: You should try to work out the answer by hand tracing to better understand the policy.

Note 2: The above assumed you compiled your program into an executable with name "NotMe_Cache_Simulator.exe". Replace it with the executable name if needed. If you are using ***gcc***, you can use the "-o NotMe_Cache_Simulator.exe" flag to generate the executable with the correct name, e.g. "gcc –o ***NotMe_Cache_Simulator.exe*** NotMe_Cache_Simulator.c"

Demonstrate the finished program to your lab TA. Your lab TA will test your program using the "`NotMe_Sample.dat`" as input **[4 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.
Marking Scheme: Report – 11 marks; Demonstration – 4 marks; Total: 15 marks.