Sample Solution for Homework Assignment 4

CS/ECE 6810: Computer Architecture
Nov 28, 2018
Name, UID

Memory Systems

Due Date: December 11, 2018.
(120 points)

Important Notes:

- Solutions turned in must be your own. Please, mention references (if any) at the end of each question. Please refrain from cheating.
- All solutions must be accompanied by the equations used/logic/intermediate steps and assumptions. Writing only the final answer will receive **zero** credit.
- All units must be mentioned wherever required.
- Late submissions **(after 11:59 pm on 12/11/2018)** will not be accepted
- All submissions must be in PDF only. Scanned copies of handwritten solutions will not be accepted as a valid submission.

1. **Virtually Indexed Cache.** Referring to the lecture slide on virtual address translation and TLB, explain the challenges if the number of bits in the page offset does not equal the number of bits in the sum of "Index" and "Byte" (Please see `http://www.cs.utah.edu/~bojnordi/classes/6810/f18/slides/18-tlb.pdf`). You are asked to identify the issues and their corresponding solutions from the literature. **(20 points)**

   Special Instructions :

   i Strictly NO collaboration or brainstorming on this question is accepted.

   ii Mention ALL references used to answer this question. If no references are mentioned, 0 points will be awarded.

   Answer: we'll evaluate your answers individually based on your explanations, references and assumptions.

2. **Virtual Memory and TLB.** Consider an operating system (OS) using 1KB pages for mapping virtual to physical addresses. Initially, the TLB is empty and all of the required pages for the user application as well as the page table are stored in the main memory. (There is no need to transfer data between main memory and the storage unit.) Every access to the TLB and main memory takes respectively 1 and 200 processor cycles. The TLB can store up 16 entries.

i Assuming that the main memory size is 1MB and the page table has 2048 entries, show the number of required bits for the physical and virtual address fields. **(5 points)**

Answer: Every page is 1KB; therefore, 10 bits are required for page offsets (1 points). The page table stores 2048 entries; therefore, 11 bits are required for virtual page numbers (2 points). The memory size is 1MB (=2^{20}B) that results in a total of 20 bits per physical addresses (2 points). In all, 10 bits for page offsets, 11 bits for virtual page numbers, and 10 bits for physical frame numbers.

ii Assume that the user application only generates five memory requests to the virtual addresses `0000`, `0004`, `0008`, `0800`, and `0804` (all in hexadecimal); the first request is ready at time 0; and the processor generates every next request immediately after serving the current one. Please find the execution time with and without using TLB in the proposed system and compute the attainable speedup due to using TLB. **(15 points)**

Answer: In the case of serving the requests *without TLB*, every request needs two accesses to the main memory taking 2×200 cycles. Therefore, the total execution time is $5 \times 400 = 2000$ cycles (4 points). In the case of execution *with TLB*, the processors needs to access TLB on every request that takes 1 cycle. On a TLB hit, the second access is to the main memory that adds a 200 cycle delay; otherwise, two accesses to the main memory are needed for accessing the page table and serving the request that add 400 cycles. After every TLB miss, the mapping will be added to the TLB for future accesses. First, we compute the page numbers of all memory requests: `00`, `00`, `00`, `02`, and `02`. Then, we identify the TLB hist and misses: `Miss`, `Hit`, `Hit`, `Miss`, and `Hit` (6 points). To compute the total execution time, we consider 201 cycles for each hit and 401 cycles for each miss; therefore, the execution time is 401+201+201+401+201=1405 (4 points). Speedup = $\frac{2000}{1405} = 1.42$ (1 points).

Request Address (hex)	Page Number	Page Offset	TLB Lookup
`0000`	00000000000	0000000000	miss
`0004`	00000000000	0000000100	hit
`0008`	00000000000	0000001000	hit
`0800`	00000000010	0000000000	miss
`0804`	00000000010	0000000100	hit

3. **DRAM Address Mapping.** Consider a simple in-order DRAM command scheduler. Initially, all DRAM banks are precharged and the scheduling queue contains seven read requests to the following physical addresses: `00040108`, `01040101`, `FF042864`, `A5181234`, `A5184321`, `00161804`, and `01040104` (all in hexadecimal). Using the following address mapping scheme, show all the required commands issued by the controller to serve the memory requests. **(20 points)**

row (10)	bank (4)	rank (2)	channel (0)	column (16)

Answer: First, we convert the request addresses to channel, rank, bank, row, and column IDs. Then, track the state of each row buffer (per bank) to determine a command sequence for each memory request. On a row buffer conflict, we use PRE, ACT, RD. If the row buffer is empty, an ACT followed by a RD is necessary. On a row buffer hit, we only use a RD command.

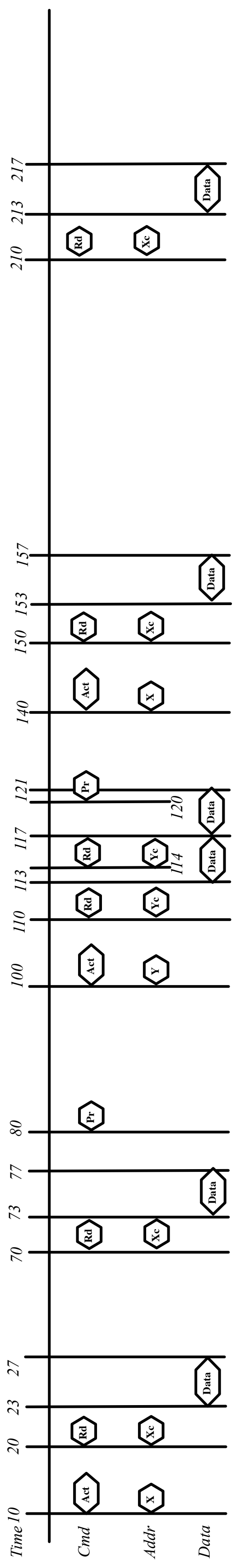
Req. Address	row (10)	bank (4)	rank (2)	column (16)	commands
`00040108`	0000,0000,00	00,01	00	,0000,0001,0000,1000	ACT - RD
`01040101`	0000,0001,00	00,01	00	,0000,0001,0000,0001	PRE -ACT -RD
`FF042864`	0000,1111,00	00,01	00	,0010,1000,0110,0100	PRE -ACT -RD
`A5181234`	1010,0101,00	0110	00	,0001,0010,0011,0100	ACT -RD
`A5184321`	1010,0101,00	01,10	00	,0100,0011,0010,0001	RD
`00161804`	0000,0000,00	01,01	10	,0001,1000,0000,0100	ACT -RD
`01040104`	0000,0001,00	00,01	00	,0000,0001,0000,0100	PRE -ACT -RD

First one has 2 points. Others, each one has 3 points

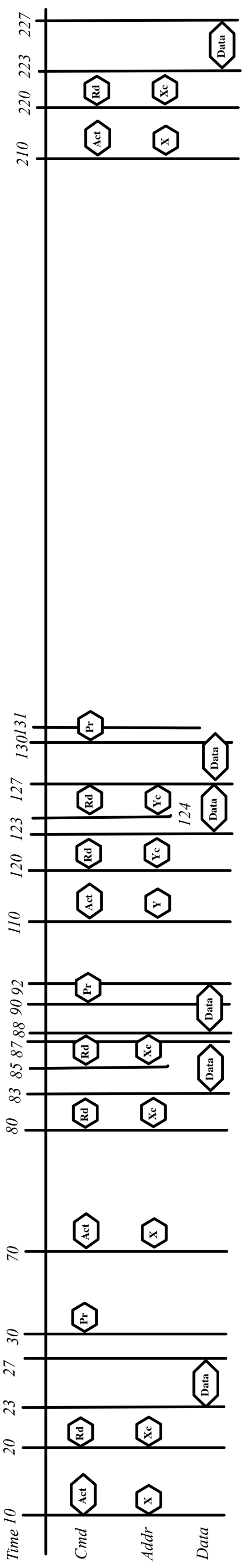
4. **DRAM Row (Page) Management.** A computer system includes DRAM and CPU. The DRAM subsystem comprises a single channel/rank/bank. CPU generates a sequence of memory requests to rows `X` and `Y`. Table below shows the accessed rows by the memory requests and their arrival times at the DRAM controller. Assume that the DRAM bank is precharged initially; the DRAM scheduling queue has infinite size; and the memory interface must enforce the following timing constraints (all in ns): tCAS = 3, tRAS = 20, tRP = 20, tRCD = 10, and tBURST = 4.

Accessed Row	**Arrival time (ns)**
`X`	10
`X`	70
`Y`	80
`X`	85
`Y`	110
`X`	210

The goal is to evaluate two row management policies, namely *open-page* and *closed-page*. Show all the necessary transactions on the command, address, and data buses for each policies. Discuss which policy performs better for the given example. (Note: you are allowed to reorder requests already waiting in the memory controller.) **(20 points)**
Answer: In the open-page policy, an open row is only closed using a precharge command to resolve a row buffer conflict. As command reordering is allowed for this question, we serve all of the row hits before closing a row. To implement a closed-page policy with command reordering, we issue a precharge command to an open row when there is no pending requests ready to schedule at the memory controller. In both policies, we can issue a command only if it is ready to issue without violating any timing constraints. The following diagram shows all of the necessary transactions on the command, address, and data buses for the two row management policies. Open-page policy performs better for these requests.



Open Page Policy



Closed-page Policy

Open-page has 8 points. Close-page has 12 points.

5. **Data TLB.** Consider the following pseudo code, in which **X** and **Y** are allocated as contiguous integer arrays in memory and are aligned to the 4KB boundaries. Assume that **k** and **l** are allocated in the register file with no need for memory accesses. We execute the code on a machine with virtual memory where the integer type (**int**) is 4 bytes wide. Assume that the system include a direct-mapped D-TLB with 1024 entries for serving all data accesses only. Initially, the D-TLB is empty. Find the hit rate of the D-TLB when running the code. **(20 points)**

```
#define M 1024
int X[M*M];
int Y[M*M];
int k, l;
  for ($k = 0; k < M; k++$) do
    for ($l = 0; l < M; l++$) do
      Y[l*M+k] = X[k*M+l];
    end for
  end for
```

Answer: Consider **X** and **Y** as two matrices, each with 1024 rows and 1024 columns. Each row is a 4KB page; therefore, all of the accesses within a row need the same TLB translation. Switching from one row to another requires a different TLB translation. According to the pseudo code, two memory accesses are performed per each loop iterations: (1) a read to row k and column l of **X** followed by (2) a write to row l and column k of **Y**. All of the **X** accesses for *l=0* and **Y** accesses for *k=0* result in a TLB miss. Moreover, D-TLB is a direct-mapped buffer that results in conflicts between **X** and **Y**. On every iteration of the outer loop, k is fixed while l varies from 0 to 1023, which results in spreading the **Y** accesses across all of the TLB entries and using only one TLB entry for all of the **X** accesses. Therefore, **Y** conflicts with **X** once per every outer-loop iteration, thereby evicting the **X** mapping from the TLB. This conflict results in (1) one additional miss for **Y** when $k = l$ ($k > 0$) and another TLB miss for **X** on $l + 1$ ($l < 1023$). Also, another conflict miss occurs for **Y** on accessing a page k previously accessed for **X**, while $k > 0$. In all, the total number of TLB misses is 1024+1024+1023+1023+1023=5117. Thus, the hit rate is $1 - \frac{5117}{1024 \times 1024} = 99.51$

6. **Bonus Question: Emerging Memory Systems.** A significant challenge in using the emerging non-volatile memories is their limited write endurance: *every memory cell can be rewritten only a limited number of times before it stops functioning.* This problem may translate into a short system lifetime. The following Figure shows the average and maximum rates of writes per block made by different applications.

   Assume the processor frequency is 2.8 GHz, the applications execute until the first block stops functioning, and every block may endure 10^8 writes.

   i Determine which application last longer and which shorter. **(5 points)**

   Answer: The shortest lifetime is achieved for an application with the maximum number of writes per cycle. In this example, OCN has the highest number of writes per cycle (0.00000125), for which the lifetime becomes $2.856 * 10^4$ seconds (equal to 7.93 hours). Likewise, we can find the application with the longest lifetime. RDX has the lowest number of writes per block (0.00000012766).

   ii Come up with a technique in hardware to increase the system lifetime. Recompute the lifetimes of applications from part i using your proposed technique. **(15 points)**

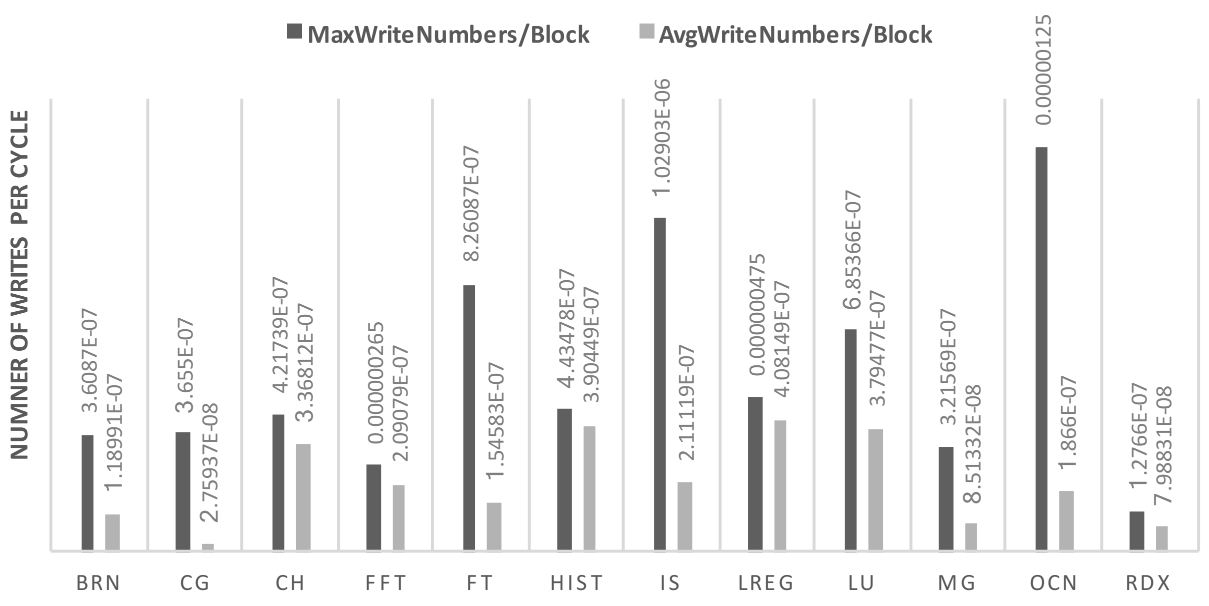


Figure 1: Average and maximum rates of writes per blocks.

Answer: Generally, a longer memory lifetime may be achieved by (1) employing materials that endure more writes and (2) distributing write operations across all memory location evenly (i.e., wear leveling technique) For example, the life time increases 100 times when using a device type that endures 10^{10} writes.