CS251 Assignment 06

Due: December 05 4:30pm

2% of Course Mark

40 Total Marks

Q1. (5 marks) Suppose we have a 32-bit MIPS computer with 1 GB of physical memory (main memory). For virtual memory on this computer, we need to translate a 32-bit virtual address into a 30-bit physical address, which is done via the page table. Below is part of the page table for translating the 32-bit virtual address to a 30 bit physical address. The page size is 4KB. Below the page table is a list of virtual addresses. Using the page table, convert the virtual addresses to physical addresses. The virtual and physical addresses are broken up into page numbers and offset bits.

If the table can not be used to convert to a physical address write: "Page Fault: Disk" on the line. The numbers to the left of the page table (VPN) are used as indices into the table. Complete the translations from virtual to physical addresses only, you do not need to update the page table.

Page Table

Virtual Page Number	Valid	Physical Page Number
0000 0000 0000 0000 0000	0	01 0000 1111 0000 0001
0000 0000 0000 0000 0001	1	00 0000 1100 1100 0100
0000 0000 0000 0000 0010	1	00 0000 0000 1000 0000
0000 0000 0000 0000 0011	1	00 1100 0000 1100 0000
. . .		. . .
0000 0000 1111 0000 0000	0	00 0110 0000 1100 0111
0000 0000 1111 0000 0010	1	00 0000 1100 1100 0101

Virtual Address	Physical Address
0000 0000 0000 0000 0001 0000 0001 1100	00 0000 1100 1100 0100 0000 0001 1100
0000 0000 1111 0000 0000 0000 0000 1000	Page Fault : Disk
0000 0000 0000 0000 0011 1100 0001 1000	00 0000 0000 1000 0000 1100 0001 1000
0000 0000 0000 0000 0000 1010 0101 1010	Page Fault : Disk
0000 0000 1111 0000 0010 1111 0001 1100	00 0000 1100 1100 0101 1111 0001 1100

Q2. (16 marks) In the table below are a sequence of instructions executed on a fully pipelined datapath with all necessary data forwarding, *load-use* stalling and branch flushing incorporated. We are interested in the data cache accesses only. The decimal addresses of data are provided for instructions that use data memory. Fill in the Hit/Miss columns in part(a) and (b).

a) (5 marks) The tables below the sequence of instructions represent data cache. Indicate what is loaded to cache for block size of 4 words. Mark any changes that occur in blocks coming and going from cache by using a new table when a block is removed. You must also update the dirty bit if a cache block is written to or leave the dirty bit at a zero if the cache block is not written to. You will not need any additional tables than the ones provided. If two blocks are removed from different indices, the changes can be indicated in one new table. You do not need to consider RAM update policies for dirty blocks.

Instruction	Memory Location	Tag	Index	Block	Byte	Hit/Miss
lw	52	000	011	01	00	Miss
lw	56	000	011	10	00	Hit
sw	44	000	010	11	00	Miss
lw	48	000	011	00	00	Hit
sw	176	001	011	00	00	Miss
lw	184	001	011	10	00	Hit
sw	180	001	011	01	00	Hit
sw	188	001	011	11	00	Hit
lw	48	000	011	00	00	Miss
lw	416	011	010	00	00	Miss

Index	Tag	D	00	01	10	11
000						
001						
010	000	1	✓	✓	✓	✓
011	000	0	✓	✓	✓	✓

Index	Tag	D	00	01	10	11
000						
001						
010	000	1	✓	✓	✓	✓
011	001	1	✓	✓	✓	✓

Index	Tag	D	00	01	10	11
000						
001						
010	011	0	✓	✓	✓	✓
011	000	0	✓	✓	✓	✓

b) (6 marks) In the table below are the same instructions now represented with block size of 8 words. Indicate what is loaded to cache in the next set of tables provided. Mark any changes that occur in blocks coming and going from cache by using a new table when a block is removed. You must update the dirty bit if a cache block is written to or leave the dirty bit at a zero if the cache block is not written to. You will not need any additional tables than the ones provided. If two blocks are removed from different indices, the changes can be indicated in one new table. Every time a cache block is replaced, a new table should be used. You do not need to consider RAM update policies for dirty blocks.

Instruction	Memory Location	Tag	Index	Block	Byte	Hit/Miss
lw	52	000	01	101	00	Miss
lw	56	000	01	110	00	Hit
sw	44	000	01	011	00	Hit
lw	48	000	01	100	00	Hit
sw	176	001	01	100	00	Miss
lw	184	001	01	110	00	Hit
sw	180	001	01	101	00	Hit
sw	188	001	01	111	00	Hit
lw	48	000	01	100	00	Miss
lw	416	011	01	000	00	Miss

Index	Tag	D	000	001	010	011	100	101	110	111
00										
01	000	01	✓	✓	✓	☑	✓	✓	✓	✓
10										
11										

Index	Tag	D	000	001	010	011	100	101	110	111
00										
01	001	1	✓	✓	✓	✓	✓	☑	✓	☑
10										
11										

Index	Tag	D	000	001	010	011	100	101	110	111
00										
01	000	0	✓	✓	✓	✓	✓	✓	✓	✓
10										
11										

Index	Tag	D	000	001	010	011	100	101	110	111
00										
01	011	0	✓	✓	✓	✓	✓	✓	✓	✓
10										
11										

c) (5 marks) When the store word *sw* instruction executes data is modified. Assume a *Write-Back* policy is used to update RAM with the modified data. If any single word in a block is written to, the dirty bit is turned on for this block. When it comes time to remove this block from the cache, the entire block is written back to RAM. State the total execution time in terms of the total number of clock cycles required by this sequence of code using a block size of 4 words (given in part a) and a block size of 8 words (shown in part b). You may assume the following RAM (memory) access times.

An individual RAM access to read or write a block of n words will take 100 + n. Therefore, a block of 4 will take 104cc. Writing back a block of n words to RAM, will require 100 + n clock cycles. You must calculate the total execution time including the execution of all instructions plus all data memory accesses plus possible additional time for *Write-Back*. You may assume all program instructions are in instruction cache and do not need to be loaded from RAM. Also, assume the pipeline has been running (ignore pipeline start-up time). You may safely assume no hazards or stalls occur.

Block Size	Total clock cycles explanation	Total
4 words	$10+(100+4)\times 5+(100+4)\times 2$	738
8 words	$10+(100+8)\times 4+(100+8)\times 2$	658

(the number of instructions)×1

+(Miss times)×(100+n)+(100+n)× Write back times.

Q3. (4 marks) AMAT (Average memory access time) is the average (expected) time it takes for a memory access considering both hits and misses. It can be calculated using the formula:

AMAT = hit_time + miss_rate × miss_penalty. The miss rate is the percentage of time that the cache is missing. The miss penalty is the additional time it takes for memory access to the next higher level in the hierarchy. Therefore, every cache miss takes additional time of (miss_rate × miss_penalty).

Suppose that you have a cache system with the following properties.
What is the AMAT in clock cycles.

a) Level_1 cache hits in 1 clock cycle with a miss_rate = 5%. Main Memory hits in 100 cycles and you may assume it always hits.

$1 + 100 \times 0.05 = 6$

b) Level_1 cache hits in 1 clock cycle with a miss_rate = 2.5%. Main Memory hits in 100 cycles and you may assume it always hits.

$1 + 100 \times 0.025 = 3.5$

4. (15 points)

Supposed you have a virtual memory system on a 32-bit computer with 4K pages, a 4 word TLB, and 16 pages of physical memory. For simplicity, assume that the page table is in memory on page 15 and addressed through a Page Table Register (not shown below) and an entry for the page table is never put in the TLB. Also assume the following:

- The page table and the TLB are initally empty (ie, all rows are invalid).
- A page read from disk is placed in the free page with the largest index.
- An entry put into the TLB is put in the first invalid row (ie, row with the lowest number that is invalid).

(a) (10 points) Below is shown a list of free pages, the first 10 entries of the Page table, the TLB, and four lines of code. Blank entries in the Page Table and TLB mean that these fields are 0. Simulate the execution of the code; our primary interest is in the page table, the TLB, and the mapping of virtual to physical addresses. As the code is executed, the page table and TLB below should be updated to reflect their memory accesses. In addition to completing the page table and TLB, you should fill in the virtual and physical memory addresses accessed for each instruction. Note whether the virtual to physical address mapping could be determined from the TLB, or if not in the TLB if the mapping was in the Page Table, or if in neither the TLB or Page table if the page had to be read from disk.

Free pages: 0 1 2 3 4 5 6 7 8 9

Page Table

Index	V	D	R	Page
00000	1	0	1	9
00001				
00002				
00003	1	0	1	8
00004				
00005				
00006	1	1	1	7
00007				
00008				
00009				

TLB

Row	V	D	R	Tag	Page
0	1	0	1	00000	9
1	1	0	1	00003	8
2	1	1	1	00006	7
3					

Code: (addresses in hexadecimal)

Address	Instruction	Virtual Address	Physical Address	TLB/PageTable/ Disk
0x00000200:	add $1,$2,$3	0x00000200	0x9200	Disk
0x00000204:	lw $1,0x00003000($0)	0x00000204 0x00003000	0x9204 0x8000	TLB Disk
0x00000208:	lw $1,0x00003004($0)	0x00000208 0x00003004	0x9208 0x8004	TLB TLB
0x0000020c:	sw $1,0x00006072($0)	0x0000020c 0x00006072	0x920C 0x7072	TLB Disk

(b) (5 points) Suppose the Free pages, the Page Table and TLB are as follows:

Free pages: 0 6 8

Page Table

Index	V	D	R	Page
00000	1	0	1	9
00001	1	0	1	3
00002	1	0	1	6
00003	1	1	1	4
00004	1	1	1	5
00005	1	1	1	1
00006	1	0	1	7
00007				
00008				
00009				

TLB

Row	V	D	R	Tag	Page
0	1	0	1	00001	3
1	1	1	1	00003	4
2	1	0	0	00006	7
3	1	0	1	00000	9

Suppose we execute the code below, starting with the Free pages, the Page Table and TLB above. Complete the page table, TLB, and Code table below. Be sure to fill in the Dirty bits in the page table for non-empty rows.

Page Table

Index	V	D	R	Page
00000	1	1	1	9
00001	1	0	1	3
00002	1	0	1	6
00003	1	1	1	4
00004	1	1	1	5
00005	1	1	1	1
00006	1	0	1	7
00007				
00008				
00009				

TLB

Row	V	D	R	Tag	Page
0	1	0	1	00001	3
1	1	1	1	00003	4
2	1	0	1	00002	6
3	1	1	1	00000	9

Code: (addresses in hexadecimal)

Address	Instruction	Virtual Address	Physical Address	TLB/PageTable/ Disk
0x000022d4:	addi $1,$0,4	0x000022d4	0x62d4	PageTable
0x000022d8:	lw $1,0x00003004($1)	0x000022d8 0x00003004	0x62d8 0x4004	TLB TLB
0x000022dc:	sw $1,0x00000072($0)	0x000022dc 0x00000072	0x62dc 0x9072	TLB TLB