Computer System Architecture
6.823 Quiz #2
October 21st, 2005

Professor Arvind

Dr. Joel Emer

Name:___________________

This is a closed book, closed notes exam.

80 Minutes

15 Pages

Notes:
- Not all questions are of equal difficulty, so look over the entire exam and budget your time carefully.
- Please carefully state any assumptions you make.
- Please write your name on every page in the quiz.
- You must not discuss a quiz's contents with other students who have not yet taken the quiz.

Writing name on each sheet	________	2 Points
Part A	________	20 Points
Part B	________	20 Points
Part C	________	18 Points
Part D	________	20 Points
TOTAL	________	**80 Points**

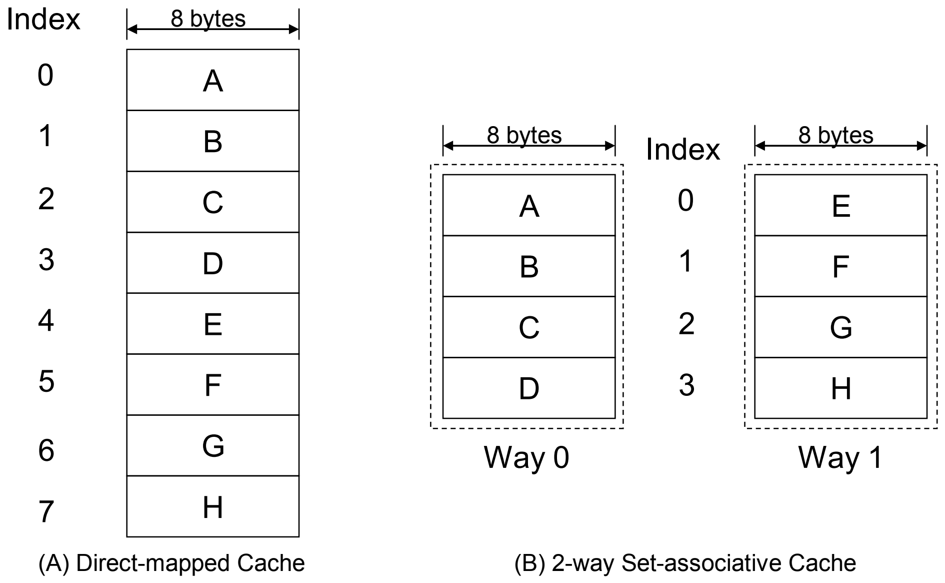
Part A: Cache Basics (20 Points)

Questions in Part A are about the operations of virtual and physical-address caches in two different configurations: direct-mapped and 2-way set-associative. The direct-mapped cache has 8 cache lines with 8 bytes/line (i.e. the total size is 64 bytes), and the 2-way set-associative cache is the same size (i.e. 32 bytes/way) with the same cache line size. The page size is 16 bytes.

Please answer the following questions.

Question 1. (6 Points)

For each cache configuration shown in the figure below, which block(s) can virtual address `0x34` be mapped to? Fill out the table at the bottom of this page. (Consider both virtually indexed and physically indexed cases for each configuration, and enumerate **all** possible blocks.)



	Virtually indexed	Physically indexed
Direct-mapped (A)		
2-way Set-associative (B)		

Name ____________________________

Question 2. (8 Points)

We ask you to follow step-by-step operations of the virtually indexed, physically tagged, 2-way set-associative cache shown in the previous question (Figure B). You are given a snapshot of the cache and TLB states in the figure below. Assume that the smallest physical tags (i.e. no index part contained) are taken from the high order bits of an address, and that Least Recently Used (LRU) replacement policy is used.
(Only valid (V) bits and tags are shown for the cache; VPNs and PPNs for the TLB.)

Index	V	Tags (way0)	V	Tags (way1)
0	1	0x45	0	
1	1	0x3D	0	
2	1	0x1D	0	
3	0		0	

Initial cache tag states

VPN	PPN	VPN	PPN
0x0	0x0A	0x10	0x6A
0x1	0x1A	0x20	0x7A
0x2	0x2A	0x30	0x8A
0x3	0x3A	0x40	0x9A
0x5	0x4A	0x50	0xAA
0x7	0x5A	0x70	0xBA

TLB states

After accessing the address sequence (all in virtual address) given below, what will be the final cache states? Please fill out the table at the bottom of this page with the new cache states. You can write tags either in binary or in hexadecimal form.

Address sequence: `0x34 -> 0x38 -> 0x50 -> 0x54 -> 0x208 -> 0x20C -> 0x74 -> 0x54`

Index	V	Tags (way0)	V	Tags (way1)
0				
1				
2				
3				

Final cache tag states

Name ____________________________

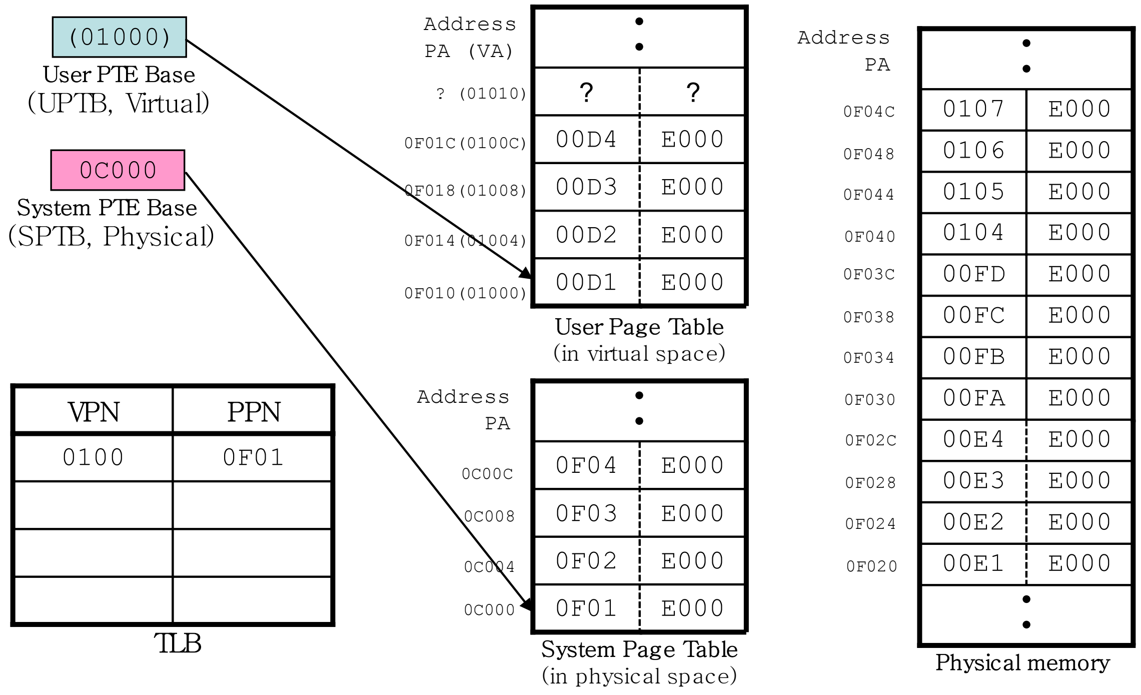
Question 3. (6 Points)

Assume that a cache hit takes one cycle and that a cache miss takes 16 cycles. What is the average memory access time for the address sequence of 8 words given in Question 2?

Name ____________________________

Part B: Handling TLB Misses (20 Points)

In the following questions, we ask you about the procedure of handling TLB misses. The following figure shows the setup for this part and each component's initial states.



VPN	PPN
0100	0F01

Address PA (VA)	User Page Table	
	• •	
? (01010)	?	?
0F01C(0100C)	00D4	E000
0F018(01008)	00D3	E000
0F014(01004)	00D2	E000
0F010(01000)	00D1	E000

Address PA	System Page Table	
	• •	
0C00C	0F04	E000
0C008	0F03	E000
0C004	0F02	E000
0C000	0F01	E000

Address PA	Physical memory	
	• •	
0F04C	0107	E000
0F048	0106	E000
0F044	0105	E000
0F040	0104	E000
0F03C	00FD	E000
0F038	00FC	E000
0F034	00FB	E000
0F030	00FA	E000
0F02C	00E4	E000
0F028	00E3	E000
0F024	00E2	E000
0F020	00E1	E000
	• •	

Notes 1. All numbers are in hexadecimal.
2. Virtual addresses are shown in parentheses, and physical addresses without parentheses.

For the rest of this part, we assume the following:

1) The system uses 20-bit virtual addresses and 20-bit physical addresses.
2) The page size is 16 bytes.
3) We use a linear (not hierarchical) page table with 4-byte page table entry (PTE). A PTE can be broken into the following fields. (Don't worry about the status bits, PTE[15:0], for the rest of Part B.)

31 … 16	15	14	13	12	11	10	9 … 0
Physical Page Number (PPN)	V	R	W	U	M	S	0000000000

4) The TLB contains 4 entries and is fully associative.

On the next page, we show a pseudo code for the TLB refill algorithm.

Name ____________________________

```
// On a TLB miss, "MA" (Miss Address) contains the address of that
// miss.  Note that MA is a virtual address.

// UTOP is the top of user virtual memory in the virtual address
// space.  The user page table is mapped to this address and up.
#define UTOP 0x01000

// UPTB and SPTB stand for User PTE Base and System PTE Base,
// respectively.  See the figure in the previous page.

if (MA < UTOP) {
   // This TLB miss was caused by a normal user-level memory access

   // Note that another TLB miss can occur here while loading a PTE.
   LW Rtemp, UPTB+4*(MA>>4);   // load a PTE using a virtual address
}
else {
   // This TLB miss occurred while accessing system pages (e.g. page
    // tables)

   // TLB miss cannot happen here because we use a physical address.
   LW_physical Rtemp, SPTB+4*((MA-UTOP)>>4);  // load a PTE using a
                                              // physical address
}

(Protection check on Rtemp); // Don't worry about this step here
(Extract PPN from Rtemp and store it to the TLB with VPN);
(Restart the instruction that caused the TLB miss);
```

TLB refill algorithm for Quiz #2

Question 4. (6 Points)

What will be the physical address corresponding to the virtual address `0x00030`? Fill out the TLB states below after an access to the address `0x00030` is completed.

Virtual address `0x00030` -> Physical address (`0x` __________)

VPN	PPN
0x0100	0x0F01

TLB states

Question 5. (8 Points)

What will be the physical address corresponding to the virtual address `0x00050`? Fill out the TLB states below after an access to the address `0x00050` is completed. (Start over from the initial system states in Page 5, not from your system states after solving the previous question.)

Virtual address `0x00050` -> Physical address (`0x` ___________)

VPN	PPN
0x0100	0x0F01

TLB states

Question 6. (6 Points)

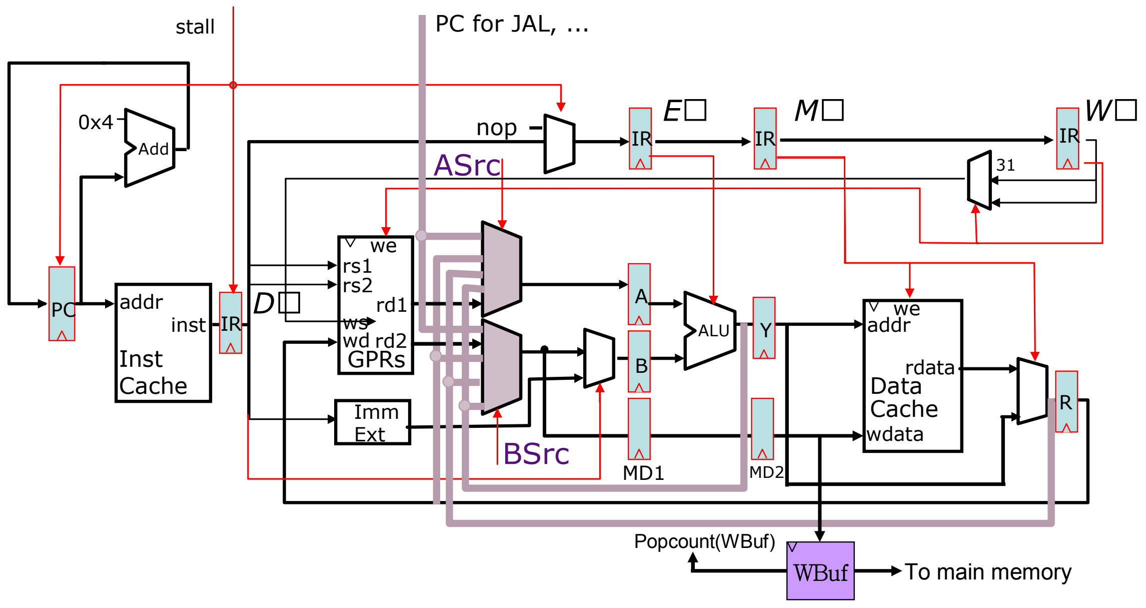
We integrate virtual memory support into our baseline 5-stage MIPS pipeline using the TLB miss handler. We assume that accessing the TLB does not incur an extra cycle in memory access in case of hits.

Without virtual memory support (i.e. he had only a single address space for the entire system), the average cycles per instruction (CPI) was 2 to run Program X. If the TLB misses 10 times for instructions and 20 times for data in every 1,000 instructions on average, and it takes 20 cycles to handle a TLB miss, what will be the new CPI (approximately)?

Part C: Write Buffer for Data Cache (18 Points)

In order to boost the performance of memory writes, Ben Bitdiddle has proposed to add a write buffer to our 5-stage fully-bypassed MIPS pipeline as shown below. Assuming a write-through/write no-allocate cache, every memory write request will be queued in the write buffer in the MEM stage, and the pipeline will continue execution without waiting for writes to be completed. A queued entry in the write buffer gets cleared only after the write operation completes, so the maximum number of outstanding memory writes is limited by the size of the write buffer.

Please answer the following questions.



Question 7. (6 Points)

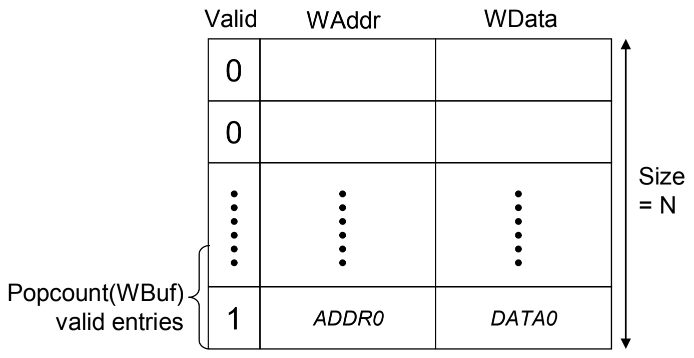
Ben wants to determine the size of the write buffer, so he runs benchmark X to get the observation below. What will be the average number of writes in flight (=the number of valid entries in the write buffer on average)?

1) The CPI of the benchmark is 2.
2) On average, one of every 20 instructions is a memory write.
3) Memory has a latency of 100 cycles, and is fully pipelined.

Name ___________________________

Question 8. (6 Points)

Based on the experiment in the previous question, Ben has added the write buffer with N entries to the pipeline. (Do not use your answer in Question 7 to replace N.) Now he wants to design a stall logic to prevent a write buffer overflow. The structure of the write buffer is shown in the figure below. `Popcount(WBuf)` gives the number of valid entries in the write buffer at any given moment.



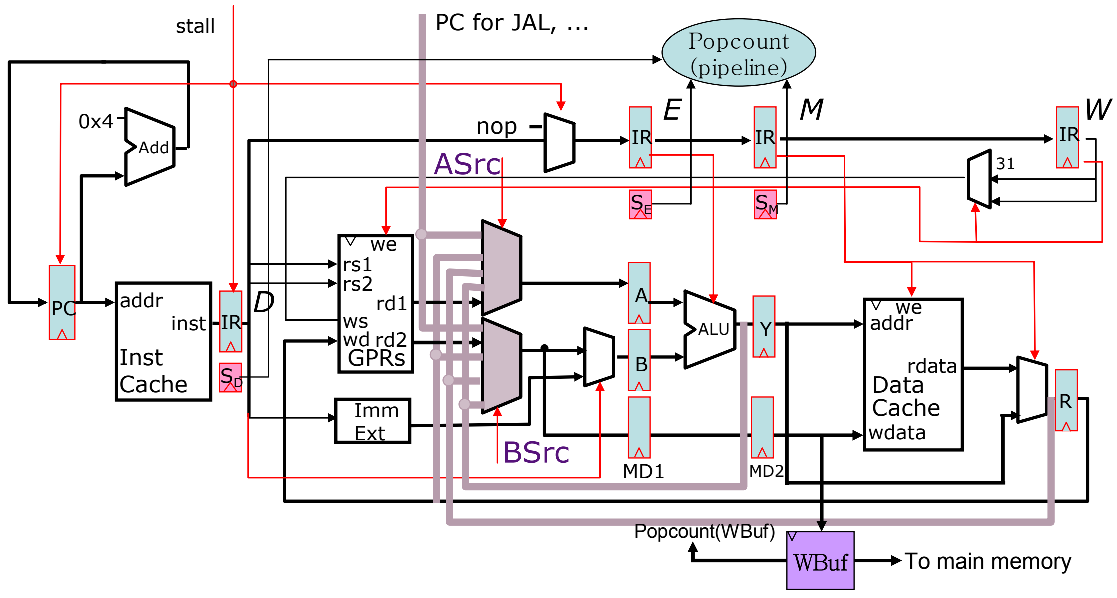
Please write down the stall condition to prevent write buffer overflows. You should derive the condition without assuming any modification of the given pipeline. You can use Boolean and arithmetic operations in your stall condition.

Stall =

Name ______________________________

Question 9. (6 Points)

In order to optimize the stall logic, Ben has decided to add a predecode bit to detect store instructions in the instruction cache (I-Cache). That is, now every entry in the I-Cache has a store bit associated with it, and it propagates through the pipeline with an S_{stage} bit added to each pipeline register (except the one between MEM and WB stages) as shown below. `Popcount(Pipeline)` gives the number of store instructions that are in flight (= number of S_{stage} bits set to 1).



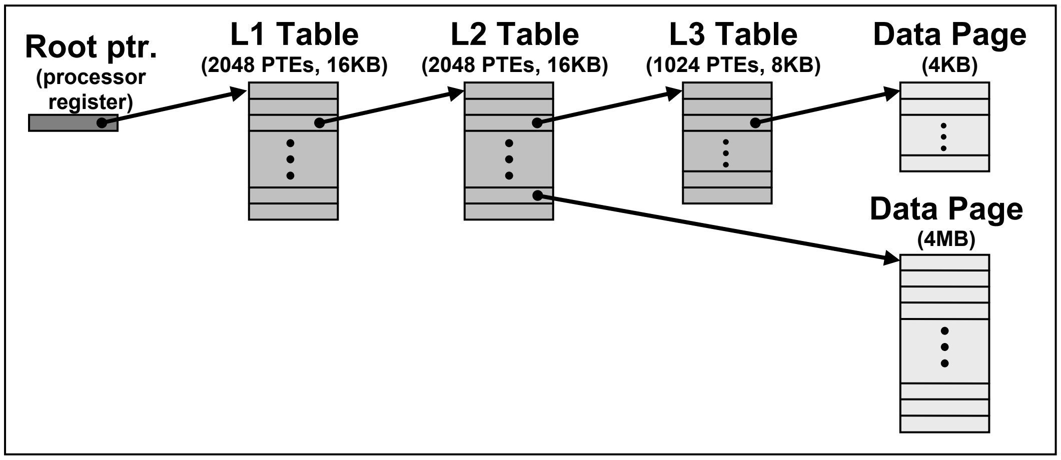
How will this optimization change the stall condition, if at all?

Stall =

Name ____________________________

Part D: Page Size and TLBs (20 points)

This problem evaluates a virtual memory system with two page sizes: 4KB and 4MB. The system uses 44-bit virtual addresses and 40-bit physical addresses. 4KB pages are mapped using a three-level hierarchical page table. 4MB pages are mapped using the first two levels of the same page table. An L2 PTE contains information which indicates if it points to an L3 table or a 4MB data page. All PTEs are 8 Bytes. The following figure summarizes the page table structure and indicates the sizes of the page tables and data pages (not drawn to scale):



The processor has a data TLB with 64 entries, and each entry can map either a 4KB page or a 4MB page. After a TLB miss, a hardware engine walks the page table to reload the TLB. The TLB uses a first-in/first-out (FIFO) replacement policy.

We will evaluate the memory usage and execution of the following program which adds the elements from two 1MB arrays and stores the results in a third 1MB array (note that, 1MB = 1,048,576 Bytes):

```
byte A[1048576]; // 1MB array
byte B[1048576]; // 1MB array
byte C[1048576]; // 1MB array

for(int i=0; i<1048576; i++)
  C[i] = A[i] + B[i];
```

We assume the A, B, and C arrays are allocated in a contiguous 3MB region of physical memory. **We will consider two possible virtual memory mappings**:

- **4KB**: the arrays are mapped using 768 4KB pages (each array uses 256 pages).
- **4MB**: the arrays are mapped using a single 4MB page.

For the following questions, assume that the above program is the only process in the system, and ignore any instruction memory or operating system overheads. Assume that the arrays are aligned in memory to minimize the number of page table entries needed.

Question 10. (4 Points)

This is the breakdown of a virtual address which maps to a 4KB page:

43 33	32 22	21 12	11 0
L1 index	**L2 index**	**L3 index**	**Page Offset**
11 bits	11 bits	10 bits	12 bits

Show the corresponding breakdown of a virtual address which maps to a 4MB page. Include the field names and bit ranges in your answer.

43   0

Question 11. (4 Points)

We define page table overhead (PTO) as:

$$\textbf{PTO} = \frac{\text{Physical memory that is allocated to page tables}}{\text{Physical memory that is allocated to data pages}}$$

For the given program, what is the PTO for each of the two mappings?

$$\textbf{PTO}_{\textbf{4KB}} = \frac{\qquad}{\qquad}$$

$$\textbf{PTO}_{\textbf{4MB}} = \frac{\qquad}{\qquad}$$

Question 12. (4 Points)

We define page fragmentation overhead (PFO) as:

$$\text{PFO} = \frac{\text{Physical memory that is allocated to data pages but is never accessed}}{\text{Physical memory that is allocated to data pages and is accessed}}$$

For the given program, what is the PFO for each of the two mappings?

$$\text{PFO}_{4KB} = \frac{\qquad}{\qquad}$$

$$\text{PFO}_{4MB} = \frac{\qquad}{\qquad}$$

Question 13. (4 Points)

Consider the execution of the given program, assuming that the data TLB is initially empty. For each of the two mappings, how many TLB misses occur, and how many page table memory references are required per miss to reload the TLB?

	Data TLB misses	Page table memory references (per miss)
4KB:		
4MB:		

Name ____________________________

Question 14. (4 Points)

Which of the following is the best estimate for how much longer the program takes to execute with the 4KB page mapping compared to the 4MB page mapping?
Circle one choice and **briefly explain** your answer (about one sentence).

1.01×	**10×**	**1,000×**	**1,000,000×**