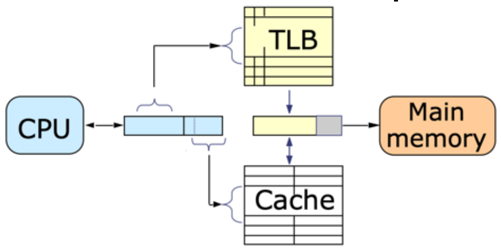
1	/16
2	/12
3	/20

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE



6.004 Computation Structures

Updated Spring 2022

Quiz #3

Name	*Athena login name*	*Score*
Solutions		

Recitation section

□ WF 10, 34-302 (Francis)	□ WF 2, 34-302 (Robert)	□ WF 12, 35-310 (Kendall)
□ WF 11, 34-302 (Francis)	□ WF 3, 34-302 (Robert)	□ WF 1, 35-310 (Kendall)
□ WF 12, 34-302 (Grace)	□ WF 10, 35-310 (Sara)	□ opt-out
□ WF 1, 34-302 (Grace)	□ WF 11, 35-310 (Sara)	

Please enter your name, Athena login name, and recitation section above. Enter your answers in the spaces provided below. Show your work for potential partial credit. You can use the extra white space and the backs of the pages for scratch work.

Problem 1. Fast Context Switches are Rocket Science (12 points)

NASA has hired you to design the processor that will control the spaceship that will fly the next mission to Mars. You start with a typical RISC-V design with paging-based virtual memory. The system has 32-bit virtual and physical addresses, 4 KB pages, a 64-entry fully associative TLB without process IDs, and a 32KB virtually-indexed, physically-tagged (VIPT) cache as shown to the right.

You profile this system and discover that about 90% of the time is spent on context switches. The code for this spaceship is structured in 128 different processes. When scheduled, each process performs a very simple action (like reading a sensor value) that takes just tens of user-level instructions, then yields control to the OS, which schedules the next process. Assume processes run in a round-robin fashion (i.e., process i+1 % 128 is scheduled after process i).

To make this system faster, you focus on reducing the time the OS spends on context switches. Each of the following questions lists a potential hardware change. Which ones would you implement? Answer yes if performance would increase, or no if performance would degrade or be unaffected. Briefly state the reason for your answer.

(A) (3 points) Using segmentation-based virtual memory instead of paging.

Beneficial? [✓ **YES**] **NO** **Explanation:**

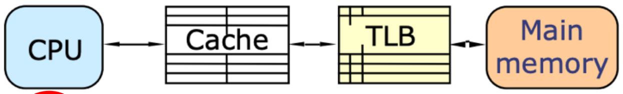
Segmentation will speed up context switches by not flushing the TLB and avoiding frequent TLB misses.

(B) (3 points) Adding process IDs to the TLB. The process ID is an extra field in the TLB that identifies the process that the translation corresponds to.

Beneficial? [✓ **YES**] **NO** **Explanation:**

Adding address space IDs means that the OS does not need to flush the TLB on a context switch, making context switches faster.
No is an acceptable answer only with an appropriate explanation: if you assume that TLB flushes are very cheap and since each process takes at least a page, no TLB entries will be reused across successive scheduling intervals of the same process.

(C) (3 points) Using a virtually addressed cache, shown below, instead of a VIPT cache.



Beneficial? **YES** [✓ **NO**] **Explanation:**

A virtually addressed cache would have to be flushed on context switches, adding overheads.

(D) (3 points) Using a different variant of the RISC-V ISA that has 16 general-purpose registers instead of the usual 32.

Beneficial? [✓ **YES**] **NO** **Explanation:**

Since general purpose registers must be saved and restored across context switches, having fewer registers reduces context switch cost.

Problem 2. Virtual Memory (20 points)

For the following questions, assume a processor with 40-bit virtual addresses, 26-bit physical addresses and page size of 2^{12} bytes per page.

(A) (2 points) Please calculate the following parameters relating to the size of the page table. You may assume each page table entry contains a dirty bit and a resident bit. *Your final answer can be a product or exponent.*

Number of entries in the page table: ____2^{28}_____

Size of each page table entry (in bits): ____16_____

(B) (1 point) What is the maximum fraction of virtual memory that can be resident in physical memory at any given time (assuming the page table is not in physical memory)?

Max fraction of virtual memory that can be resident in physical memory: ____$1/2^{14}$____

(C) (3 points) If we changed the RISC-V processor to use a page size of 2^{13} bytes, how would each of the following change? This is the only change being made to the system.

Number of virtual pages (select one of the choices below):

UNCHANGED … +1 … -1 … 2x … [✓ 0.5x] … CAN'T TELL

Number of physical pages (select one of the choices below):

UNCHANGED … +1 … -1 … 2x … [✓ 0.5x] … CAN'T TELL

Size of page table entries (in bits) (select one of the choices below):

UNCHANGED … +1 … [✓ -1] … 2x … 0.5x … CAN'T TELL

For the rest of this problem, assume a page size of 2^{12} bytes.

(D) (7 points) A program has been halted right before executing the following instruction, located at virtual address 0x3CA4.

```
. = 0x3CA4
sw x6, 0(x5) // x5 = 0x552C
```

The first 8 entries of the page table are shown to the right. The page table uses an LRU replacement policy and handles missing pages using a page fault handler. Assume that all physical pages are currently in use.

VPN	R	D	PPN
0	1	0	0x7F
1	0	--	--
LRU → 2	1	0	0xA
3	1	0	0x18
4	1	1	0x9
5	0	--	--
6	1	1	0x4
7	0	--	--
	…		

For the instruction and data accesses in the code above, please indicate the virtual address, the VPN, whether the access results in a page fault, the PPN, and the physical address. *If there is not enough information given to determine a given value, please write N/A.* Please write all numerical values in hexadecimal.

Virtual Address	VPN	Page Fault (Yes/No)	PPN	Physical Address
0x3CA4	**3**	**No**	**0x18**	**0x18CA4**
0x552C	**5**	**Yes**	**0xA**	**0xA52C**

Additionally, in the table below, please show the contents of the page table after the instruction has executed. **You can leave a row blank to indicate that the row is unchanged from the original page table.** You do not need to label the LRU entry.

VPN	R	D	PPN
0			
1			
2	**0**	**--**	**--**
3			
4			
5	**1**	**1**	**0xA**
6			
7			

All other rows are unchanged

Note the dirty bit for VPN 5 is set to 1 because data is written to the page when the store word instruction is executed

(E) (5 points) Now consider the same processor with and added 4-element, fully associative Translation Lookaside Buffer (TLB) with an LRU replacement policy. The TLB and the first 8 entries of the page table are shown below. The page table also uses an LRU replacement policy. Assume that all physical pages are currently in use.

TLB

	VPN	**V**	**R**	**D**	**PPN**
LRU →	0x3	1	1	0	0xA3
	0xFF	1	1	1	0x40
	0xB	1	1	1	0xBB
	0x12	1	1	0	0x7

Page Table

VPN	**R**	**D**	**PPN**
0	1	1	0x6
1	0	--	--
2	1	0	0x1A
3	1	0	0xA3
LRU → 4	1	1	0x5
5	1	1	0xB2
6	0	--	--
7	0	--	--
	…		

A program running on the processor is halted right before executing the following instruction located at address 0x1234:

```
. = 0x1234
lw x7, 0(x5) // x5 = 0xB0B0
```

For the instruction and data accesses in the code above, please indicate the virtual address, the VPN, whether the access results in a TLB hit, whether the access results in a page fault, the PPN, and the physical address. *If there is not enough information given to determine a given value, please write N/A.* Please write all numerical values in hexadecimal.

Virtual Address	**VPN**	**TLB Hit (Yes/No)**	**Page Fault (Yes/No)**	**PPN**	**Physical Address**
0x1234	**0x1**	**No**	**Yes**	**0x5**	**0x5234**
0xB0B0	**0xB**	**Yes**	**No**	**0xBB**	**0xBB0B0**

(F) (2 points) Using the page table from part (E), please indicate the PPN corresponding to the physical page that would be evicted upon a page fault, and if that physical page would need to be written back to disk. Briefly explain your answers.

Evicted physical page number (hex): ______**5**______

Explanation: **LRU page is VPN 0x4, PPN 0x5**

Writeback necessary (Yes/No): ____ **Yes** _____

Explanation: **Dirty bit for the evicted page is 1**

Problem 3. Synchronization (15 points)

To end the semester, you've decided to go big: make grilled cheese sandwiches for all of MIT. While you had a brilliant plan, you ran into a small problem: you only recruited two other people to help. However, since you remember synchronization, you've decided to parallelize some of the work!

You've decided to split up the work in the following way: one person (person A) will be responsible for the bread, another will be responsible for cheese and condiments, and a third responsible for tomatoes and condiments. This division of labor is roughly described below.

A:	**B:**	**C:**
Place first slice of bread Place second slice of bread Serve sandwich Goto A	Add one slice of cheese Add one serving of mustard Goto B	Add one layer of tomatoes Add one serving of mustard Goto C

(A) (3 points) Suppose all three processes run the code above without any synchronization. For each of the following situations, circle whether it is possible for the situation to occur or not.

1. You serve a sandwich that consists only of two slices of bread.

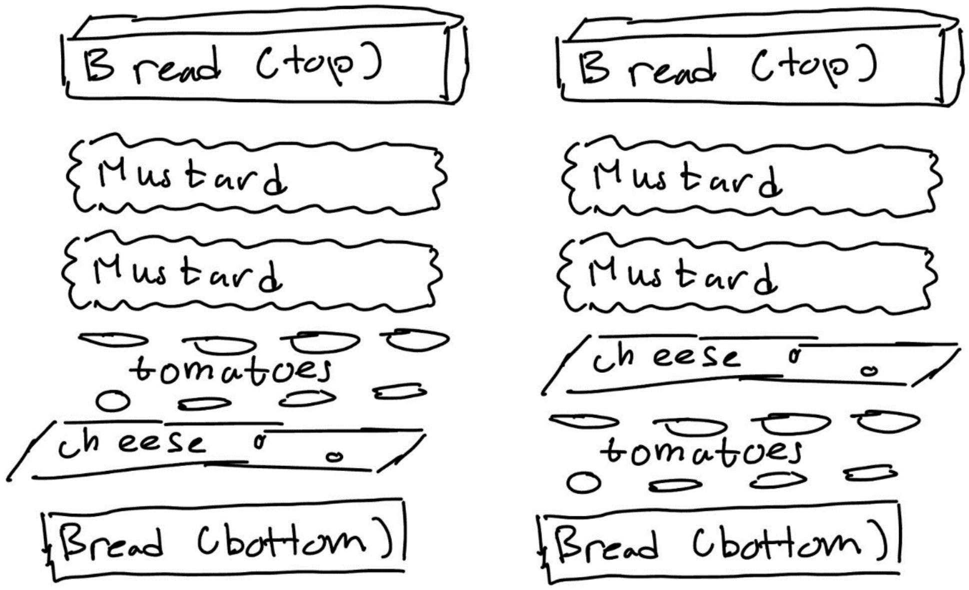
   [✓ Possible] / Not Possible

2. You serve a sandwich with two slices of cheese but no servings of condiments

   Possible / [✓ Not Possible]

3. You serve a sandwich with mustard on top of the second slice of bread

   [✓ Possible] / Not Possible



This diagram shows the two valid sandwich configurations allowed by the constraints in part B.

(B) (12 points) Having attempted to make sandwiches without any coordination, you quickly realize that you need some rules to create the perfect sandwich. You settle on the following constraints:

1. The bottom of the sandwich should be a single slice of bread. There should be no condiments, tomatoes, or cheese under this first slice.
2. After the first slice of bread, you should have a slice of cheese and one layer of tomatoes. **The order does not matter:** you can have cheese *then* tomatoes, or tomatoes *then* cheese.
3. You cannot place cheese and tomatoes on top of a sandwich at the same time.
4. On top of the cheese and tomatoes, you should have two servings of mustard. Both B and C **should be able apply mustard at the same time**.
5. On top of the mustard, you should have the final slice of bread.

The allowed sandwich configurations are illustrated on the previous page. In the code below, insert the appropriate semaphores, WAITs and SIGNALS to ensure that all 5 constraints are met while avoiding deadlock. Make sure to specify the initial value of your semaphores. For full credit, use five or fewer semaphores and don't introduce unnecessary precedence constraints.

5 semaphore solution:

Shared Memory (specify your semaphores and initial values)		
bottom_bread = 0, lock = 1, cheese = 0, tomatoes = 0, top_bread = 0		

A:
 	
Place first slice of bread
signal(bottom_bread)
signal(bottom_bread)
wait(top_bread)
wait(top_bread)
 	
Place second slice of bread
 	
Serve sandwich
 	
Goto A
	B:
	wait(bottom_bread)
	wait(lock)
	
	Add one slice of cheese
	signal(lock)
	signal(cheese)
	wait(tomatoes)
	
	Add one serving of mustard
	
	signal(top_bread)
	
	Goto B
		C:
		wait(bottom_bread)
		wait(lock)
		
		Add one layer of tomatoes
		signal(lock)
		signal(tomatoes)
		wait(cheese)
		
		Add one serving of mustard
		
		signal(top_bread)
		
		Goto C

4 semaphore solution:

Shared Memory (specify your semaphores and initial values) **lock = 0, cheese = 0, tomatoes = 0, top_bread = 0**		
A:	**B:**	**C:**
	wait(lock)	**wait(lock)**
Place first slice of bread	Add one slice of cheese	Add one layer of tomatoes
signal(lock)	**signal(lock)**	**signal(lock)**
wait(top_bread)	**signal(cheese)**	**wait(cheese)**
wait(top_bread)		**wait(lock)**
	wait(tomatoes)	**signal(tomatoes)**
Place second slice of bread	Add one serving of mustard	Add one serving of mustard
Serve sandwich	**signal(top_bread)**	**signal(top_bread)**
Goto A	**Goto B**	**Goto C**

The handling of the cheese and tomatoes semaphores flipped between processes B and C. The key is that the third wait(lock) must live between a wait(cheese/tomatoes) and a signal (tomatoes/cheese)

END OF QUIZ 3!