1	/10
2	/19
3	/17

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.191 Computation Structures

Updated Fall 2022

Quiz #3

Name	*Athena login name*	*Score*
Solutions		

Recitation section

☐ WF 10, 34-301 (Grace)	☐ WF 2, 13-5101 (Frances)	☐ WF 12, 36-155 (Shiqi)
☐ WF 11, 34-301 (Grace)	☐ WF 3, 13-5101 (Frances)	☐ WF 1, 36-155 (Shiqi)
☐ WF 12, 35-310 (Alexandra)	☐ WF 10, 13-4101 (Georgia)	☐ WF 1, 34-303 (Amelia)
☐ WF 1, 35-308 (Alexandra)	☐ WF 11, 13-4101 (Georgia)	☐ WF 2, 34-303 (Amelia)
		☐ opt-out

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. Operating Systems (10 points)

Consider the following two processes running RISC-V programs labeled with virtual addresses. Note that all pseudoinstructions in this code translate into a single RISC-V instruction.

Program for Process A

```
.=0x200
   li t0, 5000
   li t1, 0
loop:
   li a0, 0x300
   li a7, 0x13 // print system call
   ecall
   addi t1, t1, 1
   ble t1, t0, loop
   j exit // exit process

.=0x300
  .ascii “Hello from process A\n”
```

Program for Process B

```
.=0x450
    li t0, 50
    li t1, 10
    div t2, t0, t1
    sw t2, 0x900(x0)
    li a0, 0x600
    li a7, 0x13 // print system call
    ecall
    j exit // exit process

.=0x600
  .ascii “Hello from Process B\n”
```

Assume the OS schedules **Process B** first. *For the following questions, if you can’t tell a value based on the information given, write CAN’T TELL.*

(A) (3 points) A timer interrupt occurs just prior to the execution of `li t1, 10` in Process B. Process A runs for some time, then another timer interrupt occurs and control is returned to Process B. What are the values in the following registers immediately after returning to Process B?

t0: _______**50**__________

t1: __**CAN'T TELL or 0**_

pc: _____**0x454**_________

(B) (3 points) What are the values in the following registers just after the first `ecall` in Process A completes?

t0: ______**5000**_________

t1: _______**0**___________

pc: _____**0x214**_________

(C) (4 points) The RISC-V processor does not have hardware to support a `div` (integer division) instruction, so the OS must emulate it. What are the values in the following registers after `div` is emulated?

t0: _______**50**_________

t1: _______**10**_________

t2: ________**5**_________

pc: ____**0x45C**________

Problem 2. Virtual Memory (19 points)

For the following questions, assume a processor with 38-bit virtual addresses, 26-bit physical addresses, and page size of 256 (2^8) bytes per page. The Page Table of this processor uses an LRU replacement strategy and handles missing pages using a page fault handler.

(A) (2 points) What is the size of the page table? Assume that each page table entry includes a **dirty bit** and a **resident bit**. Specify the number of page table entries and the width of each entry.

Number of entries in the page table: _____2^{30}_____

Width of each page table entry (bits): _____20_____

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

Fraction of virtual memory that can be resident in physical memory: _____$1/2^{12}$_____

(C) (2 points) If we **double the size of our physical memory** but keep the same page size and virtual memory size, what effect will the change have on the size of a page table entry and on the number of entries in the page table? Use a letter "a" through "e" to indicate how the *new* value of the parameter compares to the *old* value of the parameter:

(a) doubled (b) increased by 1 (c) stays the same (d) decreased by 1 (e) halved

Number of entries in the page table: _____c_____

Width of each page table entry in bits: _____b_____

You now run a test program on the original processor that has a 38-bit virtual address, a 26-bit physical address and 256 bytes per page. Execution of this test program is halted just before executing the following two instructions. The first 8 locations of the page table at the time execution was halted are shown on the next page; the least recently used page ("LRU") and next least recently used page ("next LRU") are as indicated. Execution resumes and the following two instructions at locations 0x2FC and 0x300 are executed:

```
lw x12, 0x180(x0)   | PC = 0x2FC
sw x12, 0x60C(x0)   | PC = 0x300
```

(D) (6 points) The first 8 rows of the page table are shown below. Show all changes made to the first 8 rows of the page table after executing each of these instructions. If a row of the page table has not changed, you may write NO CHANGE over it instead of copying it.

Initial Page Table contents:

VPN	R	D	PPN
0	0	--	--
1	1	0	0x123
2	0	--	--
3	1	0	0xABC
LRU → 4	1	1	0xA0
5	1	1	0x380
6	0	--	--
next LRU → 7	1	1	0x651

Fill in Page Table contents after execution of `lw x12, 0x180(x0)    | PC = 0x2FC`

VPN	R	D	PPN
0			**No Change**
1			**No Change**
2	**1**	**0**	**0xA0**
3			**No Change**
~~LRU →~~ 4	**0**	**--**	**--**
5			**No Change**
6			**No Change**
~~next~~ LRU → 7			**No Change**

Fill in Page Table contents after execution of `sw x12, 0x60C(x0)    | PC = 0x300`

VPN	R	D	PPN
0			**No Change**
1			**No Change**
2			**No Change**
3			**No Change**
~~LRU →~~ 4			**No Change**
5			**No Change**
6	**1**	**1**	**0x651**
~~next LRU →~~ 7	**0**	**--**	**--**

Our processor has a **direct-mapped, 4-entry TLB**, so it uses VPN bits 1:0 as the index into the TLB.

(E) (8 points) The figure below shows the contents of the TLB prior to the execution of the two instructions in the previous question. For simplicity, we show the tag and index bits of the VPN, even though the TLB needs to store the tag bits only. Fill in the contents of the TLB after the execution of each of the two instructions. If an entry has not changed, you may write NO CHANGE over it instead of copying it. **Assume that the dirty bits in the TLB are kept in sync with the dirty bits in the page table.**

Note: The TLB caches page table entries, which the page fault handler may modify. To ensure correct behavior, most processors include a privileged instruction to invalidate a TLB entry. Assume that the page fault handler uses that instruction. Also assume that all page fault handler accesses happen through physical memory and do not go through the TLB.

Initial TLB contents:

Index	*VPN (Tag+Index)*	*V*	*D*	*PPN*
0	0x200	1	1	0xBC
1	0x1	1	0	0x123
2	0x36	1	0	0x412
3	0x23	1	1	0x99

Fill in TLB contents after execution of `lw x12, 0x180(x0)` | `PC = 0x2FC`

Index	*VPN (Tag+Index)*	*V*	*D*	*PPN*
0	**NO CHANGE**			
1	**NO CHANGE**			
2	0x2	1	0	0xA0
3	**NO CHANGE**			

Fill in TLB contents after execution of `sw x12, 0x60C(x0)` | `PC = 0x300`

Index	*VPN (Tag+Index)*	*V*	*D*	*PPN*
0	**NO CHANGE**			
1	**NO CHANGE**			
2	0x6	1	1	0x651
3	0x3	1	0	0xABC

Problem 3. Synchronization (17 points)

You and your friends are baking holiday cookies. Each of your friends has specific tasks for making the cookies. This is implemented as four threads shown below with pseudocode.

A:	**B:**	**C:**	**D:**
Put cookie in oven			
Bake cookie	Cut cookie out	Add sprinkles	Add icing
Remove cookie			
Goto A	Goto B	Goto C	Goto D

(A) (3 points) Without any synchronization, which of the following are possible to happen? Circle one for each question.

Sprinkles are added before a cookie is cut out **([✓ Possible] / Not Possible)**

A cookie is removed before it is baked **(Possible / [✓ Not Possible])**

Icing is added to a cookie that is cut but not baked **([✓ Possible] / Not Possible)**

(B) (6 points) Add semaphore waits and signals to the threads to ensure that multiple cookies can be baked using the threads and following these rules:

1. Only one cookie can be made at a time
2. All cookies must first be cut out, then placed in the oven, baked, removed, and finally an optional topping is added.
3. Each cookie may have no toppings, sprinkles, or icing, but not both
4. Your threads should be able to make an unlimited number of cookies
5. It must be possible for each cookie to have any of the allowed topping combinations, independent of the previous cookies

Under these rules, the valid cookies are made with the following sequences:

- Cut cookie out, Put cookie in oven, Bake cookie, Remove cookie
- Cut cookie out, Put cookie in oven, Bake cookie, Remove cookie, Add sprinkles
- Cut cookie out, Put cookie in oven, Bake cookie, Remove cookie, Add icing

Once one of these sequences is complete, a new sequence should start to create the next cookie. It should be possible for the next cookie to follow any of the above sequences.

To help you out, we have added a fifth process, E, that does nothing but can be used for additional synchronization.

State your semaphore initializations below and add waits and signals to the pseudocode table. You may only add wait and signal calls to the threads.

Semaphore Initializations:

startCookies = 1, cutCookies = 0, cookiesDone = 0

Threads:

A:	**B:**	**C:**	**D:**	**E:**
wait(cutCookies)				
Put cookie in oven	wait(startCookies)	wait(cookiesDone)	wait(cookiesDone)	
Bake cookie	Cut cookie out	Add sprinkles	Add icing	wait(cookiesDone)
				signal(startCookies)
Remove cookie	signal(cutCookies)	signal(startCookies)	signal(startCookies)	
signal(cookiesDone)				
Goto A	Goto B	Goto C	Goto D	Goto E

(C) (8 points) You decide you would like to allow cookies to have both icing and sprinkles as toppings, but only if the icing is added first. Add semaphore waits and signals to the threads to follow all other rules, but allow this new type of cookie to be produced as well.

The new valid sequences to make a cookie are as follows:

- Cut cookie out, Put cookie in oven, Bake cookie, Remove cookie
- Cut cookie out, Put cookie in oven, Bake cookie, Remove cookie, Add sprinkles
- Cut cookie out, Put cookie in oven, Bake cookie, Remove cookie, Add icing
- Cut cookie out, Put cookie in oven, Bake cookie, Remove cookie, Add icing, Add sprinkles

As before, your threads should create an unlimited number of cookies, each of which can be created from any of the sequences above independent of previous cookies.

To help you out, there are now two processes, E and F, that do nothing but can be used for additional synchronization.

State your semaphore initializations below and add waits and signals to the pseudocode table. You may only add wait and signal calls to the threads.

Semaphore Initializations:

startCookies = 1, cutCookies = 0, cookiesDone=0, nextTopping=0

Threads:

A:	**B:**	**C:**	**D:**	**E:**
wait(cutCookies)				
Put cookie in oven	wait(startCookies)	wait(nextTopping)	wait(cookiesDone)	
Bake cookie	Cut cookie out	Add sprinkles	Add icing	wait(cookiesDone)
Remove cookie	signal(cutCookies)	signal(startCookies)	signal(nextTopping)	signal(nextTopping)
signal(cookiesDone)	Goto B			
Goto A		Goto C	Goto D	Goto E
F:				
wait(nextTopping)				
signal(startCookies)				
Goto F				

END OF QUIZ 3!