Problem 2. Virtual Memory (20 points)

For the following questions, consider a RISC-V processor with 32-bit virtual addresses, 26-bit physical addresses and a page size of 4096 (=2¹²) bytes. The page table of this processor uses an LRU replacement policy to bring missing pages from disk, handled by a page fault handler.

(A) (3 points) What is the size of the page table in bits? Assume that each page table entry includes a dirty (D) bit and a resident (R) bit.

Page offset: 12 bits, VPN: 20 bits, PPN: 14 bits \( \rightarrow 2^{20} \) entries, \( 14+2 = 16 \) bits per entry

Size of page table (bits): \( 16 \times 2^{20} = 2^{24} \) bits

You now run a test program on this RISC-V processor. Execution of this test program is halted just before executing the following three instructions. The first 8 locations of the page table at the time execution is halted are shown to the right; the least recently used page (“LRU”), next least recently used page (“next LRU”), and third least recently used page (“3rd LRU”) are as indicated. Assume that all the code and data for handling page faults is located on physical page 0 and all the pages in physical memory are in use. Execution resumes and the following three instructions at locations 0x4FF8, 0x4FFC and 0x5000 are executed:

```
// lui sets x1 to 0x2000
lui x1, 0x2          | PC = 0x4FF8
lw x2, 0x7E4(x1)     | PC = 0x4FFC
sw x3, 0x7E8(x1)     | PC = 0x5000
```

(B) (6 points) What is the memory access that triggers the second page fault while executing the three instructions above? Provide the virtual address generated by the RISC-V processor that triggers the page fault and the physical address translated after the page fault is handled.

1st page fault: Data load (lw) from VA 0x27E4 (VPN=2)

LRU page VPN=5/PPN=0x456 evicted (not dirty) & PPN=0x456 reused for VPN 2.

2nd page fault: Instruction fetch (sw) from VA 0x5000 (VPN=5).

next LRU page VPN=6/PPN=0xA09 written to disk & PPN=0xA09 reused for VPN 5.

Virtual address generated by RISC-V: _____0x5000_____

Physical address translation: __0xA09000_____

(C) (3 points) Which physical pages have been written to disk during the execution of the three instructions above? List their physical page numbers, if any, in the order they are written.

Physical page numbers (in order) or NONE: __0xA09____________________
Our RISC-V processor has a **fully-associative, 4-entry TLB using an LRU replacement policy.** The TLB caches page table entries, which the page fault handler may modify. To ensure correct behavior, our RISC-V processor implements and uses a privileged instruction to invalidate a TLB entry. Assume that all page fault handler accesses do not go through the TLB.

The figure below shows the contents of the TLB after the execution of the first instruction from the previous question. The least recently used entry (“LRU”) and next least recently used entry (“next LRU”) in the TLB are as indicated. The TLB always selects the LRU entry for eviction, even if there are other invalid TLB entries that could be used instead.

(D) (8 points) Fill in the contents of the TLB after the execution of each of the two remaining instructions. If an entry has not changed, you may write NO CHANGE over it instead of copying it. You do not need to indicate the LRU entry.

**Fill in TLB contents after execution of  `lui x1, 0x2  |  PC = 0x4FF8`**

<table>
<thead>
<tr>
<th>VPN (tag)</th>
<th>V</th>
<th>D</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4</td>
<td>1</td>
<td>0</td>
<td>0x123</td>
</tr>
<tr>
<td>0x3</td>
<td>1</td>
<td>1</td>
<td>0x110</td>
</tr>
<tr>
<td>0x5</td>
<td>1</td>
<td>0</td>
<td>0x456</td>
</tr>
<tr>
<td>0x0</td>
<td>1</td>
<td>0</td>
<td>0xAEC</td>
</tr>
</tbody>
</table>

**Fill in TLB contents after execution of  `lw x2, 0x7E4(x1)  |  PC = 0x4FFC`**

<table>
<thead>
<tr>
<th>VPN (tag)</th>
<th>V</th>
<th>D</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO CHANGE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NO CHANGE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x5</td>
<td>0</td>
<td>0</td>
<td>0x456</td>
</tr>
<tr>
<td>0x2</td>
<td>1</td>
<td>0</td>
<td>0x456</td>
</tr>
</tbody>
</table>

**Fill in TLB contents after execution of  `sw x3, 0x7E8(x1)  |  PC = 0x5000`**

<table>
<thead>
<tr>
<th>VPN (tag)</th>
<th>V</th>
<th>D</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO CHANGE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x5</td>
<td>1</td>
<td>0</td>
<td>0xA09</td>
</tr>
<tr>
<td>NO CHANGE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2</td>
<td>1</td>
<td>1</td>
<td>0x456</td>
</tr>
</tbody>
</table>
Problem 3. Pipelining combinational circuits (12 points)

For each of the questions below, please create a valid K-stage pipeline of the given combinational circuit. This circuit has two inputs A and B, and one output C. Each component in the circuit is annotated with its propagation delay in nanoseconds.

Show your pipelining contours and place large black circles (●) on the signal arrows to indicate the placement of pipeline registers. Give the latency and throughput of each design, assuming ideal registers (tPD=0, tSETUP=0). Remember that our convention is to place a pipeline register on each output. (If you need them, additional copies of these circuit diagrams are available at the end of the quiz.)

(A) (3 points) Show a maximum-throughput 1-stage pipeline.

(B) (3 points) Show a maximum-throughput 2-stage pipeline using a minimal number of registers.
(C) (3 points) Show a maximum-throughput pipeline. For full credit, your design should achieve the smallest possible latency among all maximum-throughput pipelines.

Latency (ns): ___150______
Throughput (ns\(^{-1}\)): ___1/50______

(D) (3 points) To increase the throughput of the pipelined circuit, the component with propagation delay 50ns is internally pipelined into two components with propagation delay 20ns and 30ns, as shown below. Show a maximum-throughput pipeline. For full credit, your design should achieve the smallest possible latency among all maximum-throughput pipelines.

Latency (ns): ___120______
Throughput (ns\(^{-1}\)): ___1/30______
Problem 4. Pipelined Processors (32 points)

(A) (8 points) Consider a five-stage pipelined RISC-V processor (IF, DEC, EXE, MEM, WB), where:

- All branches are predicted not-taken.
- Branch decision are made in the EXE stage.
- The pipeline has full bypass paths.

Assume that beqz and beq instructions in the given portion of the program are always taken. **Complete the next 11 cycles of the pipeline diagram below. Draw an arrow showing any valid use of bypass paths from the EXE, MEM, or WB stages to the DEC stage. Denote killed instructions and stalls using NOPs.**

(If you need them, additional pipeline diagrams are available at the end of the quiz.)

(B) (8 points) Again assume that both beqz and beq instructions in the given code are always taken. **Complete the next 11 cycles of the diagrams below assuming the processor does not have any bypass paths.** Denote killed instructions and stalls using NOPs. **There is no need to fill out the blanks for the cycles that are same as in (A).**

(C) (4 points) The processor in (A) has a critical path delay of 5 ns, and the processor in (B) has a critical path delay of 4 ns. For the loop shown above, how many nanoseconds does each processor take to execute one loop iteration? Consider steady-state behavior only.

Nanoseconds per loop iteration for Processor (A): _25_________

5 cycles per loop iteration x 5 ns per cycle (clock period) = 25 ns

Nanoseconds per loop iteration for Processor (B): _28_________

7 cycles per loop iteration x 4 ns per cycle (clock period) = 28 ns
Problem 5. Synchronization (12 points)

You just got a new job working for Supreme selling limited quantities of hype merchandise. Box logo crewnecks are about to drop (go on the market). The crewnecks are only available in pink and blue, and it is your job to ensure that only 50 crewnecks of each color are made available for sale. In addition, since the blue ones are so much more hyped, Supreme wants to make sure that at least 3 pink ones are sold before the first blue one is made available for purchase.

You are provided with three semaphores defined in Shared Memory (shared among process A and process B). semP is used for the pink crewnecks, semB is for the blue crewnecks, and semX is used for whatever you want. You are also given the following skeleton code for processes A and B. Process A will be used by customers to purchase pink crewnecks and process B will be used to purchase blue crewnecks. Assume that buy() is properly synchronized (i.e., it can be called from multiple processes).

Your job is to initialize the semaphores and use them correctly in process A and B so that no more than 50 shirts of each color can be sold, and so that at least 3 pink shirts are sold before the first blue shirt can be sold. Remember that semaphore values must be non-negative.

Semaphore semP = ???, semB = ???, semX = ???;

Process A

wait(semX)
wait(semX)
wait(semX)

loopA:

wait(semP)

buy("pink")
signal(semX)

j loopA

(A)(3 points) What are appropriate initial values for each of the semaphores?

initial value for semP: ___50_________
initial value for semB: ___50_________
initial value for semX: ___0_________
(B) (6 points) Add any missing signal and wait calls to processes A and B above to ensure that all constraints are satisfied and no unnecessary constraints are introduced.

(C) (3 points) Assuming that the only process synchronization appearing in processes A and B is the use of the three semaphores provided, will the above implementation work without adding any additional constraints with multiple Process A? Multiple Process B?

Works with multiple Process A (circle one):  YES  NO

Works with multiple Process B (circle one):  YES  NO

END OF QUIZ 3. HAVE A NICE BREAK!