a [Pipelined Performance] (0.75pts) Three interns, α, β, and θ are in charge of performance characterization of the MN-4363 processor, which represents a basic MIPS implementation. First, all tried to estimate the average CPI (cycles per instruction). To this end, for a benchmarking program, all relied on the ratio of two measurements: $\frac{\texttt{the total number of cycles it took to execute the program}}{\texttt{the number of instructions}} = \frac{\texttt{C}}{\texttt{I}}$.

- α counted the number of instructions *fetched* to estimate `I`.
- β counted the number of instructions *issued* to estimate `I`.
- θ counted the number of instructions *retired* to estimate `I`.

Note: I(nstruction)F(etch)-I(nstruction)D(ecode)-EX(ecute)-MEM(ory)-W(rite)B(ack) constitute the 5 pipeline stages. An instruction is "issued" as it moves from ID to EX; and "retired" as it leaves WB. Recall that branches can earliest be resolved at ID.

a-1 Whose CPI estimation you think is the most accurate? Consider two cases: (i) if branch prediction is implemented; (ii) if branch prediction is not implemented.

a-2 How would the 3 estimates compare? How would each capture the impact of control flow hazards (assuming that branch prediction is implemented)? Consider different pipeline stages a branch can get resolved at.

a-3 Explain the implications of the three approaches under forwarding.

b [Pipelined Execution] (1pt) α, the infamous architect of the MN-4363 processor – a pipelined implementation of the basic MIPS ISA, wants to implement forwarding from EX(ecute) and MEM(ory) stages to EX. To this end he connected

- The inputs of EXMEM register to the inputs of the IDEX register.
- The inputs of MEMWB register to the inputs of the IDEX register.

He added muxes at the inputs of the IDEX register to control where the inputs to the EX stage should be coming from – i.e. I(nstruction)D(ecode) stage, forward from (the outputs of) EX, or forward from MEM. He claims that he will be able to cover any dependency by inclusion of these two forwarding paths.
Would this implementation work? Explain your answer considering clocking of pipeline registers.
Note: I(nstruction)F(etch)-I(nstruction)D(ecode)-EX(ecute)-MEM(ory)-W(rite)B(ack) constitute the 5 pipeline stages. The pipeline register between two stages is named by concatenating the acronyms for the preceding and succeeding pipeline stages. I.e. IDEX is the register between ID and EX stages; EXMEM, between EX and MEM; MEMWB, between MEM and WB. As opposed to α's suggestion, in a classic MIPS implementation, connecting the outputs of EXMEM to the output of IDEX (over muxes) takes care of forwarding from EX to EX. In a classic MIPS implementation, connecting the outputs of MEMWB to the output of IDEX (over muxes) takes care of forwarding from MEM to EX. The control inputs of the muxes are set to trigger forwarding as necessary.

c [Memory] (1pt) Your labmate, α, is in charge of the memory hierarchy design of the MN-4363 processor.

c-1 α claims that a write-through cache can be made to perform very similarly to a write-back cache if enhanced with a write buffer. If the write buffer is large enough, no stalls would be observed as a write-through cache processes a write hit.

- Explain how a write buffer can help.
- Explain whether α's observation is correct.

c-2 According to α, a write-through policy should be preferred for faster operation, since a write-through cache can handle the write and the tag check in parallel (which is not the case for a write-back cache).

- Is α correct that a write-back cache cannot handle the actual write and the tag check in parallel? Explain.
- Is α's insight about write-through caches correct? Explain.

d [Quantitative Analysis] (1pt) Your class-mate θ's study on usage of high-level language constructs suggests that procedure calls are one of the most expensive operations. He came up with an ISA which reduces the loads and stores normally associated with procedure calls and returns. The first thing he did was run some experiments with and without this optimization. His experiments deployed the same state-of-the-art optimizing compiler that will be used with either version of the computer. These experiments revealed the following information: (i) The clock cycle time of the optimized version is 5% lower than the unoptimized version. (ii) Thirty percent of the instructions in the unoptimized version are loads or stores. (iii) The optimized version executes two-thirds as many loads and stores as the unoptimized version. For all other instructions the dynamic execution counts are unchanged. (iv) Every instruction (including load and store) in the unoptimized version takes one clock cycle. (v) Due to the optimization, the procedure call and return instructions take one extra cycle in the optimized version, and these instructions account for 5% of total instruction count in the optimized version.

Is θ's optimization working? Justify your answer.