Name: ______________________________ Uniqname: ________________

EECS 570 Winter 2022 Final Exam

Total points: 100
Total time: 120 minutes
Closed book, closed notes.
Calculators are allowed, but no PDAs, Portables, Cell phones, etc.
For calculation questions, show your work and clearly indicate your final answers.
Use pen or a dark pencil.
Do not spend too much time on any one question.
State any reasonable assumptions that you need to make.
There are 12 pages in this exam. Please make sure that you have all pages.
Please try to write all your answers in the assigned spaces.
If you really need extra space, you may use the backs of pages (but please **clearly** indicate that you are doing so).

Please sign the Honor Code:

I have neither given nor received unauthorized aid on this examination, nor have I concealed any violations of the Honor Code.

____

Q1	/10
Q2	/20
Q3	/20
Q4	/30
Q5	/20
Total	/100

1 Memory Consistency I (10 points)

(a) (5 points) Consider the following program.

Core 0	**Core 1**
(i1) x = 1	(i3) y = 1
(i2) r1 = y	(i4) r2 = x

Is the outcome `r1=0, r2=0` forbidden under SC? Circle Yes or No as appropriate.

Yes No

(b) (5 points) Consider the following program.

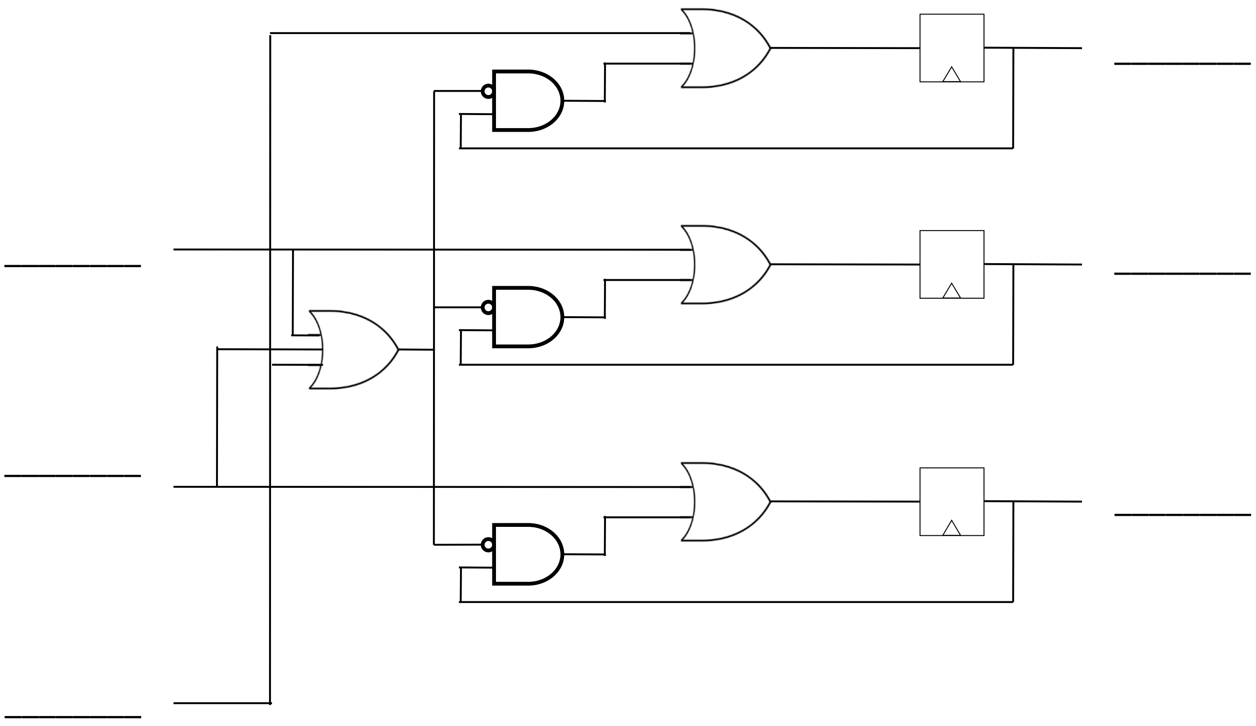
Core 0	**Core 1**
(i1) x = 1	(i3) r1 = y
(i2) y = 1	(i4) r2 = x

Is the outcome `r1=1, r2=0` forbidden under TSO? Circle Yes or No as appropriate.

Yes No

2 Interconnects I (20 points)

(a) (10 points) What is this circuit? Label its inputs and outputs (using the blanks provided), and briefly explain its operation.



(b) (5 points) The Occamy is a parallel processor with an on-chip network. The Occamy's network is a 2-D mesh, and uses Valiant's routing algorithm with X-Y routing. The routing is implemented using source routing. Tom is an architect working on the Occamy. He notices that the use of Valiant's routing algorithm requires that the packet exit the network at the intermediate node, only to be re-injected into the network from the intermediate node to proceed to its destination. This seems like extra overhead to Tom, so he optimizes the network implementation as follows: instead of exiting the network at the intermediate node, a packet is simply routed first from its source to an input of the intermediate node, and directly from there to its destination. Shortly after making this change, Tom observes that the Occamy starts hanging occasionally and needs to be rebooted whenever it hangs. What is going on? Be sure to explain your answer.

(c) (5 points) A network implements credit-based flow control, with a 5-cycle router pipeline, with a credit pipeline delay of 2 cycles. The network uses buffers of size 10, with a flit propagation delay of 2 cycles and a credit propagation delay of 1 cycle. What is the buffer turnaround time for this network?

—END OF QUESTION 2—

3 Fantastic Bugs and Where to Find Them (20 points)

Newt is working on the Thunderbird, a server-class multicore processor which implements TSO. The Thunderbird is running a web server. The Thunderbird currently runs workloads correctly, but it's a bit slow.

Answer the following questions. **Be sure to explain your answers.**

(a) (5 points) Newt wishes to improve the performance of the Thunderbird, so he adds a next-line prefetcher (for both instructions and data) to the processor. Puzzlingly to Newt, the modified processor's performance when running the web server remains almost exactly the same as before the modification. What is likely going on? (You may assume that Newt has implemented the next-line prefetcher correctly.) **The prefetcher remains in the processor for parts (b) and (c) of this question.**

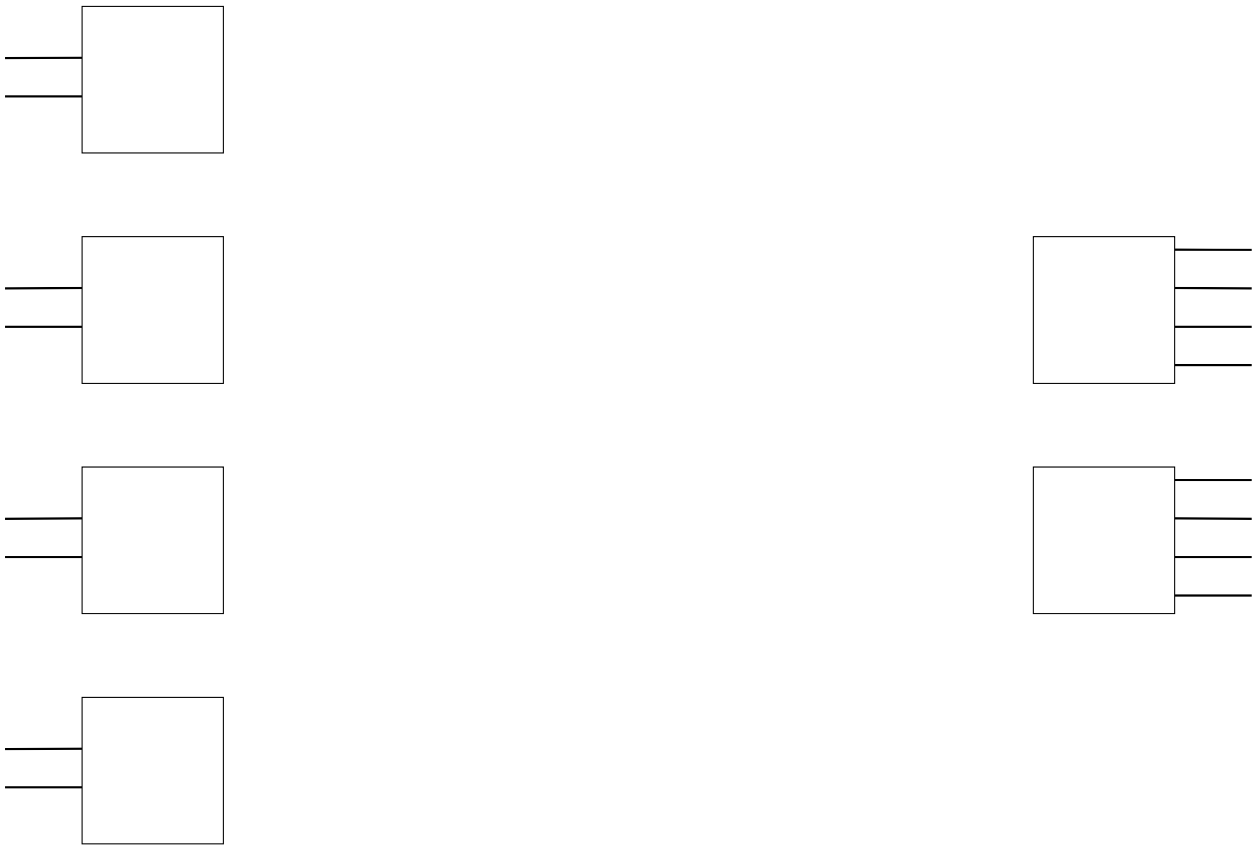
(b) (5 points) Newt notices that the coherence protocol of the Thunderbird is rather simplistic, and stalls in a number of different scenarios. Newt optimizes the coherence protocol to reduce stalls, including eliminating the stall (by adding transient states) that occurs when a core receives an invalidation for a line before it receives the data for that line. However, upon running some benchmarks with the new protocol, Newt notices that certain loads in the program end up staying in the pipeline forever and never complete, despite generating lots of memory requests and receiving data for those memory requests. What is the reason for this?

(c) (5 points) To fix the issue with the loads that don't finish, Newt decides to ensure that memory requests generated for loads can use the data provided to them (even if it's been invalidated) to service a load, as long as the cache discards the data immediately after the load operation completes. This does indeed fix the issue of loads never completing, but Newt now finds memory consistency violations (specifically, TSO violations) popping up in his benchmark runs, leaving him rather befuddled. What is the reason for these memory consistency violations?

(d) (5 points) What can Newt do to eliminate the memory consistency violations in (c) while also preventing the issue described in (b)?

4 Interconnects II (30 points)

(a) (10 points) The following diagram gives a skeleton of a 2:4 separable allocator (i.e., 2 requestors, 4 resources). Label the boxes and connect the missing wires to complete the diagram of the allocator. You do not need to label the wires.



(b) (5 points) SuperNet industries are developing a processor with an on-chip network, where routers have 4 ports. The network has good path diversity, and so multiple paths exist to every destination. Engineers at SuperNet want to develop a new feature called "turbo mode". In turbo mode, instead of transmitting at most one flit from one input to one output every cycle, the router attempts to send two flits from a given input to two different outputs in a single cycle (where each output corresponds to one of the paths to the packet's destination). You may assume that the flits are set up so that each flit contains the information necessary for its routing. You may also assume that flits are only taken from a maximum of 2 input ports every cycle (since up to 4 output ports will be needed to support turbo mode for flits from the 2 input ports).

Can the separable allocator from (a) be used by routers for switch allocation in such a network? Why or why not? Explain your answer.

(c) (5 points) Consider an on-chip network with 3 virtual channels (VC0, VC1, and VC2). The cores (and directory) connected by the network use an MSI coherence protocol that does not expect point-to-point ordering—in other words, a protocol like your base MSI protocol for PA2. Requests travel on VC0, forwarded requests on VC1, and replies on VC2.

The network's routing algorithm is not deadlock-free. To alleviate the deadlock, an engineer proposes making VC1 an escape VC. Will this make the system deadlock-free (as far as the processor and network are concerned)? Why or why not? Explain your answer.

(d) (5 points) A RISC-V startup, Fiver, is developing an MPSoC that has an on-chip network. They decide to use an irregular topology for their network. They begin with a 2D mesh that uses turn-based routing, and proceed by successively merging adjacent routers until doing so would violate performance or bandwidth constraints. Once the switch merging is complete, Fiver engineers run benchmarks on their model of the processor. To their surprise, they find that certain nodes are unable to send messages to each other despite a network path existing between the nodes. What is likely going on?

(e) (5 points) What is a solution to the issue that Fiver engineers are facing in (d)?

5 Memory Consistency II (20 points)

(a) (10 points) Engineers at Chips-R-Us are creating a multicore processor. Its pipelines are in-order and don't conduct speculative execution. Each core has its own FIFO store buffer from which it can read its own writes early. Only one store is allowed to be in-flight from a given core's store buffer to the memory hierarchy at any time. The cache hierarchy is private L1s and a shared L2.

The Chips-R-Us coherence protocol is based on MSI, but does not track sharers. On writes, the writing core obtains write permissions from the directory (ensuring a total order on all writes to a given address), but invalidations are not sent to current sharers. Instead, on any miss in its private L1, each core self-invalidates all Shared lines in its L1.Such self-invalidations also invalidate lines in transient states on the way to Shared (i.e., lines that are waiting for data responses to service loads). Cores do not use invalidated data. A fence results in a self-invalidation identical to that which occurs on an L1 miss.

Also note the following points:

- If a core has a copy of a line in Shared and wants to write to the line, it experiences a store miss. In response to its corresponding write request, the latest copy of the line is sent to the core along with write permissions.
- After a bounded number of reads to a Shared line in an L1, the line is evicted from that L1.
- You may ignore read-modify-write instructions for the purposes of this question.

Is Chips-R-Us's processor a valid implementation of TSO? If so, explain how the implementation maintains TSO's required orderings. If this implementation of TSO is buggy, describe a bug that could arise in it. (You may assume that the pipeline and store buffer are implemented correctly.)

(b) (10 points) **This question is not related to part (a).** Consider the following litmus test.

Core 0	**Core 1**
(i1) x = 1 (i2) FENCE (i3) y = 1	(i4) r1 = y (i5) y = 2 (i6) r2 = y <addr> (i7) r3 = x
Can r1=1, r2=2, r3=0?	

A microarchitecture maintains the following orderings:

- The `FENCE` (instruction `i2`) ensures that all cores observe `i1` before `i3`.
- The set of memory operations to a given address in the test obey SC with respect to each other. In other words, the outcomes of memory operations obey SC per location.
- As the test depicts, there is an address dependency between `i6` and `i7`. This dependency is preserved by the microarchitecture, i.e., `i6` and `i7` are executed in order.
- The microarchitecture implements rMCA write atomicity.
- The microarchitecture does not implement address prediction or value prediction.

Given these orderings, can the outcome `r1=1, r2=2, r3=0` be visible on this microarchitecture? If the outcome cannot occur, clearly explain why it cannot occur. If the outcome can occur, describe clearly how it could occur. (If you contend that the outcome can occur, you may not use obviously incorrect functionality—e.g., a buggy microarchitectural implementation of the FENCE operation—to cause it to occur.)