EECS 570 *Final Exam*

Answer Key

Winter 2021

Name: _____ unique name: _____

Sign the honor code:

I have neither given nor received aid on this exam nor observed anyone else doing so.

_____

Scores:

#	Points
1	**/20**
2	**/20**
3	**/15**
4	**/25**
5	**/20**
Total	***/ 100***

NOTES:

- Open lecture notes and open papers assigned for reading.
- Calculators are allowed.
- **Do not use web sources.**
- **Be sure to show work and explain what you've done when asked to do so.**

1. **Memory consistency I [20 points]**

Consider a four-processor system running the code below. A, B and C are global variables, initialized to 0. Note that P3 and P4 are performing the same sequence of memory operations.

P1	P2	P3	P4
`A = 1;`	`B = 1;`	`while (C==0) /*spin*/;`	`while (C==0) /*spin*/;`
`r = B;`	`C = 1;`	`s = B;`	`u = B;`
		`t = A;`	`v = A;`

a) If r is 0, what set(s) of result values for s, t, u, and v are *possible* on a sequentially consistent system? Enumerate all possible states. [5 points]

State 1: s = _____ t = _____ u = _____ v = _____

or

State 2: s = _____ t = _____ u = _____ v = _____

or (add more states below if necessary)

s = 1 t = 1 u = 1 v = 1

Loads in P3 and P4 cannot be reordered before the spin-loop.
Spin-loop exits only after "C = 1";
"B = 1" -> "C = 1" (program order); So, s and u must be 1
"r = B" -> "B = 1", because r's final state is 0, different from its initial state (100)
"A = 1" -> "r = B" (program order), So, t and v must be 1

b) If r is 0, what set(s) of result values for s, t, u and v are possible on a system which relaxes W R ordering, such as a processor consistent (PC) system? [5 points]

TSO (total store order) only relaxes Store->Load ordering compared to sequential consistency (SC), but ensures store atomicity. PC is more relaxed than TSO by allowing any processor to see the changes of a write before it is seen by all processors.

State 1: s = _____ t = _____ u = _____ v = _____

or

State 2: s = _____ t = _____ u = _____ v = _____

or (add more states below if necessary)

"A = 1" may or may not be seen by P3 and P4's loads, because "r = B" (load) can be moved ahead of "A = 1" (S -> L is relaxed)

s = 1 t = 1 u = 1 v = 1 (any SC reachable state is reachable under PC)

Also, under PC:
s = 1
t = 0 or 1
u = 1
v = 0 or 1

c) If r is 0, what additional set(s) of result values for s, t, u and v (beyond those in your answer for (b)) are possible on a system which relaxes all memory orderings? [5 points]

State 1: s = _____ t = _____ u = _____ v = _____

or

State 2: s = _____ t = _____ u = _____ v = _____

or (add more states below if necessary)

If all memoy orderings are relaxed, then

- loads in P3 and P4 may be reordered w.r.t spin-loop
- writes in P2 may be re-ordered
- Store and load may be re-ordered

Therefore, each load in P3 and P4 may return old or new value. All 16 combinations are possible.

s = 0 or 1
t = 0 or 1
u = 0 or 1
v = 0 or 1

d) Write in fence instructions (MEMBARs and/or WMBs) in the code listing above such that **s** and **u** are guaranteed to be 1 on a completely relaxed system. Your changes should minimize both the number of fence instructions and their performance impact. [5 points]

P2: Add a fence between two writes
P3 and P4: Add a fence immediately after the spin-while-loop exits and before the two loads.

2. **Memory Consistency II [20 points]**

In-window speculation allows loads to speculatively execute out-of-order. One way to check sequential consistency memory ordering constraints is to re-execute the load at the time of commit and check if the returned load value is the same as the speculatively executed load's value. Let us refer to this mechanism as value-based ordering.

One downside of value-based ordering is that, because all loads execute twice, the L1 cache access bandwidth is doubled.

Design a hardware filter that identifies loads that do not need to be re-executed during commit, because they could not possibly have caused a memory ordering violation; these filtered loads are executed only once.

Propose a design for such a hardware filtering method. In your answer, be sure to include:

- A description of which loads your filtering mechanism will not re-execute
- An explanation of why the filtered loads do not need to be re-executed (i.e., how do you know they couldn't have exposed a memory ordering violation.)
- An argument that your mechanism filters out a substantial number of load re- executions (you don't need to filter a majority, but you should argue you are capturing some kind of common case).
- A diagram and description of any new hardware structures you add

  Value Based Ordering
  https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.59.7955&rep=rep1&type=pdf

(empty page)

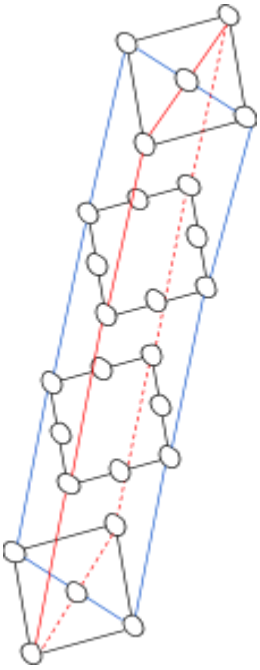
3. **Overlapping Ring Routing [15 points]**

Consider the following topology.

Horizontal rings (black): Two 4-node rings on the top and bottom, and two 6-node rings in the middle.

Vertical rings (red/blue): Two rings that connect the four horizontal rings, and intersect at two connecting nodes, one at the top and another at the bottom.

Create a deadlock-free routing algorithm for this topology with the **constraint** that you can only have virtual channels on the vertical (red/blue) links. You cannot use virtual channels for the horizontal rings.



Basic idea: primarily route through the red/blue vertical channels. Common correct answers typically involved either: limiting turns OR eliminating connections in the horizontal rings to remove the possibility of a cycle. Common mistakes: forgetting that delivering packets to nodes not connected to vertical rings need to be delivered in a manner which will not deadlock with other nodes in these rings.

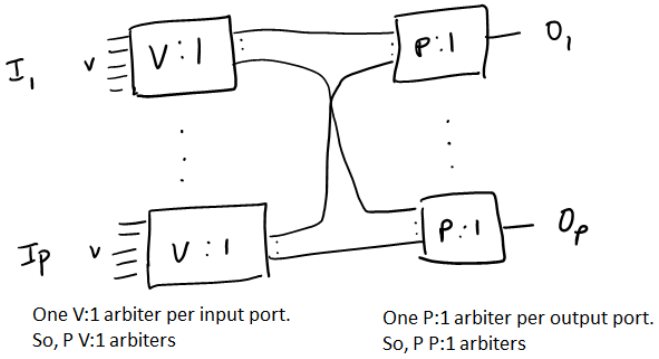
4. **Switch Allocation [25 points]**

What is separable switch allocation? (skip this if you are familiar with the concept)

Consider an on-chip network router with virtual channels (VCs) discussed in class. The crossbar switch (not shown in figures) within a router provides only one connection to each input port, even though each input has V virtual channels. Every input port is connected to every output port. Thus, a crossbar switch connects P inputs to P outputs, where P is the number of input (and output) ports.

Since each input port has V virtual channels, a switch allocator (SA) has **V * P "requests"** from input ports in any cycle. **P** output ports are the "**resources**" that it needs to allocate.

A separable switch allocator (shown below) has two phases. In the first phase, it selects one winner per input port (using P V:1 arbiters). These winners compete in the second phase for the output ports they need. A total of P (P:1) arbiters choose a winner for each output port in the second phase.



Problem:

Say we have 3 input/output ports and 3 virtual channels. The picture below shows the output port "requests" (labelled as O1, O2, or O3) in the virtual channels of input ports (packets P1-P7).

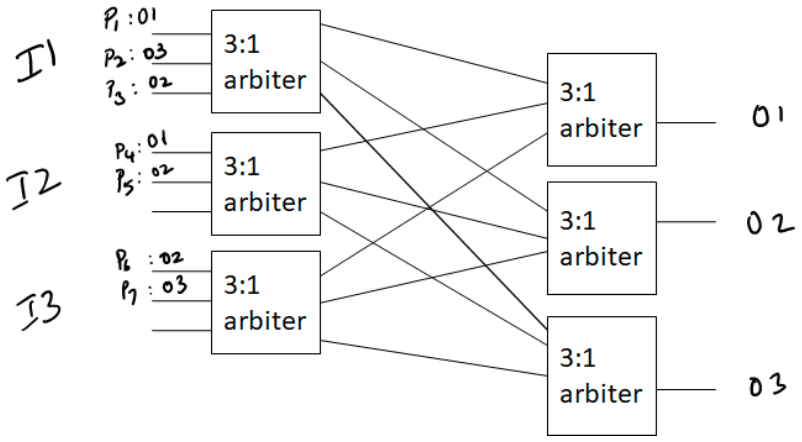


Figure: SA Example

Part A [5 points]

a) Determine the winners (packet IDs, P1-P7) in the first phase that would result in the worst possible switch allocation.

I1:_____ I2: _____ I3:____

I1: P3 (O2) I2: P5 (O2) I3: P6 (O2)

b) Determine the winners (packet IDs, P1-P7) in the first phase that would result in the best possible switch allocation.

I1:_____ I2: _____ I3:____

I1: P1 (O1) I2: P5 (O2) I3: P7 (O3)
I1: P2 (O3) I2: P4 (O1) I3: P6 (O2)
I1: P3 (O2) I2: P4 (O1) I3: P7 (O3)

I1: P2 I2: P1 I3: P3
I1:

Part B [6 points]

Improvement: In the first phase, instead of picking one winner per input port, say we select two winners. Both winners are sent to the second phase. The second phase still has to pick one winner per output port.

Instead of a 3 * 3 (i.e., P * P) crossbar, we need a 6 * 3 crossbar (i.e., 2P * P). Because, the crossbar connects 2 packets per input port.

a) Can you use a 3:1 arbiter to select two winners out of three requests? Explain.

Use 3:1 to select the "loser" input port.

Another way to reason about this is, if R1, R2, R2 are three requests, then there are only three possible output combinations, if you are going to select two winners:

(R1, R2)
(R1, R3)
(R2, R3)

3:1 selects one of these three combinations.

b) What is the arbiter size you need for the second phase (express it as X:Y)? How many of them do you need?

Arbiter for second phase (X:Y): _____

How many? _____ arbiters

3 6:1 arbiters.

c) Consider the SA example (see figure) under the improved design. Is it possible that none of the packets for input port I2 (P4, P5) get allocated? If so, specify the winners for the first and second phases. [6 points]

(State the winners in terms of packet IDs)

First phase:
I1: _____ _____ I2: _____ _____ I3: _____ _____

Second phase:

O1: _____ O2: _____ O3: _____

First phase winners:

I1: P1 P2/P3 I2: P4 P5 I3: P6 P7

Second phase winners:

O1: P1 O2: P3/P6 O3: P2/P7

P1

P1 P3/P6

d) Extend the SA example (shown in Figure) by inserting at most one packet to input ports I2 or I3 to illustrate a scenario where sub-optimal switch allocation exists. Sub-optimal allocation is one that does not utilize as many output ports as possible. [8 points]

Clearly specify two allocations for the revised example: one optimal and the other sub-optimal.

Input port where you want to insert a packet: _____

What is the new packet's output port: _____

Optimal allocation: (state the winners in terms of packet IDs)

First phase:
I1: _____ _____ I2: _____ _____ I3: _____ _____

Second phase:

O1: _____ O2: _____ O3: _____

Sub-Optimal allocation: (state the winners in terms of packet IDs)

First phase:
I1: _____ _____ I2: _____ _____ I3: _____ _____

Second phase:

O1: _____ O2: _____ O3: _____

There are several solutions. Two of them are presented here.

Solution1: Insert P8 (O3) into I2. I2 - O3 - O1

Optimal allocation:

First phase: any combination except the sub-optimal case described below.

I1: P1 P2 I2: P4 P5 I3: P6 P7

Second phase:

O1: P1 O2: P5 O3: P7

Sub-optimal allocation

First phase: I1: P2 P3 I2: P5 *P8* I3: P6 P7

Second phase: O1: - O2: P3/P5/P6 O3: P2/*P8*/P7

O1 is wasted

Solution 2: Insert P8 (O1) into I3. I3 - O1 - O3

Optimal allocation:

Similar to prevous solution.

```
Sub-optimal allocation

      First phase:    I1: P1 P3     I2: P4 P5    I3: P6 P8

      Second phase:     O1: P1/P4/*P8*      O2:   P3/P5/P6    O3: -

      O3 is wasted.
```

5. NoC Deadlock Elimination [20 Points]

Assume we have a ring network with no virtual channels. It uses a simple routing algorithm: route clockwise until a packet reaches its destination. This network is ***not*** deadlock-free.

Your company asks you to use a novel mechanism called *Synchronous Shift Mechanism* (SSM). The SSM takes a directed cyclic path in the network as an input. The mechanism will then synchronously shift (move) the packets in the nodes along the given cyclic path by one hop at the same cycle. This ensures that forward progress is made and no packets are lost/dropped.

a) Say, we detected a deadlock in our ring. Explain why we *may* be able to resolve that deadlock by applying a shift using SSM. [5 points]

Deadlocks are caused by cyclical resource dependencies in our network. When the SSM is used, we have a chance to deliver a packet and therefore break this cycle by freeing resources. This will end the deadlock. Many of you missed this point and just explained that the SSM would shift the deadlock - **this alone does not explain why the deadlock might be resolved.**

b) Say, you do not have a way to detect deadlock in a **mesh network**. We want to allow all the turns. It has no VCs. So the routing protocol is **not** deadlock-free. [7 points]

Describe a time/power efficient design for how you will use SSM to ensure deadlock-freedom in this **mesh network.** You may invoke the SSM shift in any cycle. Ignore live-lock.

DRAIN: Deadlock Removal for Arbitrary Irregular Networks
Is the inspiration for this question. The intended answer is to create a cycle which covers the entire mesh, and every time interval T (which we assume to be fairly long) we run the SSM on the entire network. Other solutions which relied on some approximation heuristic were also

accepted - more conservative heuristics (high threshold) typically received more points. Deadlocks are rare, so we should deal with them rarely to be energy efficient - many of your answers worked but ran the SSM too frequently to be efficient.

c) Your company is pleased with your new optimizations and wants to understand the implications. Explain why or why not they will need the following features in their NoC components and programs. **[8 points]**

i) Virtual Channels. Do you still need them if you have SSM to ensure deadlock-freedom?

No. We do not need VCs as there is no need for escape channels.

ii) Do you need deadlock-free routing algorithms?

No. If our network cannot deadlock, it's ok if our routing algorithm does - the SSM will take care of it.

iii) Do you need to ensure that dependencies between message classes (e.g., coherence requests/replies) don't cause deadlock in the network?

Yes. Unless you stated the assumptions listed from the DRAIN paper, we need to consider that some messages might get blocked by others as the coherence protocol might be stalling on a request while waiting for a response.

iv) Do you need to ensure that your program (software) is deadlock-free (e.g., due to improper locking)?

Yes. This should be obvious - even if our messages are delivered, software locks are completely divorced from this and can easily deadlock even if the network never does.