Name _____

Computer System Architecture
6.823 Quiz #4
May 14th, 2014
Professors Daniel Sanchez and Joel Emer

This is a closed book, closed notes exam.

80 Minutes
15 Pages

Notes:

- Not all questions are of equal difficulty, so look over the entire exam and budget your time carefully.
- Please carefully state any assumptions you make.
- Show your work to receive full credit.
- Please write your name on every page in the quiz.
- You must not discuss a quiz's contents with other students who have not yet taken the quiz.

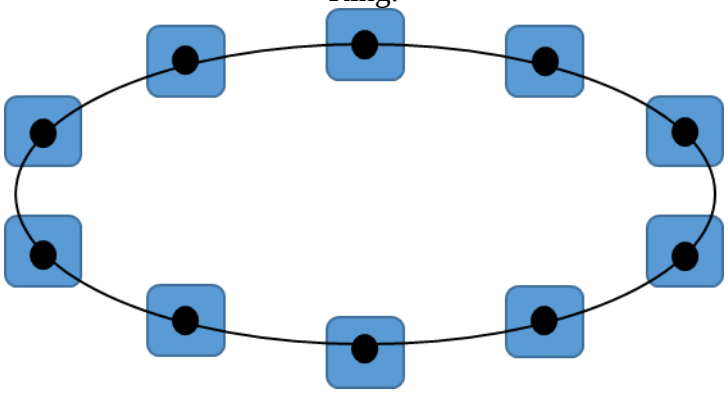
Part A	_____	25 Points
Part B	_____	55 Points
Part C	_____	20 Points
TOTAL	**_____**	**100 Points**

Name _____

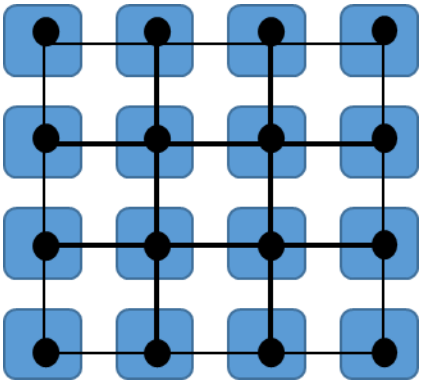
Part A: Network Effects (25 pts)

You are choosing between several network topologies for your on-chip network, shown below.

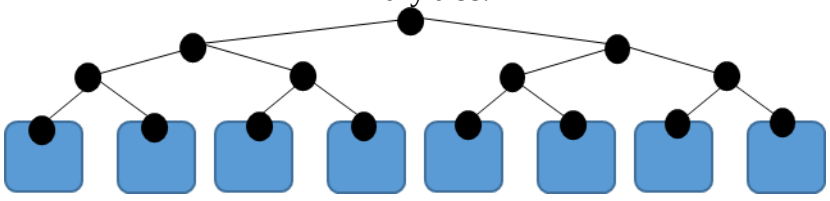
Ring:



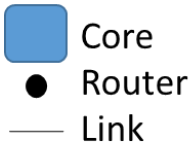
Mesh:



Binary tree:



Legend:



Name _____

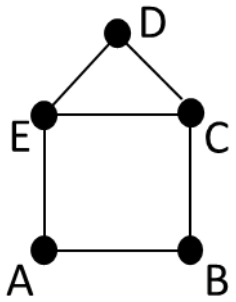
Question 1 (10 points):

Your first task is to evaluate these topologies along several important dimensions. Fill in the table below as a function of the number of nodes in the network, *N*. You can safely assume *N* is an even power of 2, giving a complete mesh and binary tree. *For partial credit, give the asymptotic growth instead.*

	Ring	**Mesh**	**Tree**
Number of links			
Diameter			
Average distance			
Bisection bandwidth			

Name _____

In a sudden flash of inspiration, you decide to use the following topology:



Having decided upon a topology, you now want to make sure your system works properly. All links are bidirectional.

Question 2 (5 points):

Show how deadlock could arise in the network by drawing an example on the graph above. Explain your answer in one or two sentences.

Name _____

Question 3 (10 points):

Draw the channel dependency graph (CDG) for your topology.

Show an example of how to eliminate routes to prevent deadlock on the CDG.

Name _____

Part Deux: Synchronicity (55 points)

(Same as Handout #14.) You are writing a queue to be used in a multi-producer/single-consumer application. (Producer threads write messages that are read by one consumer.) We assume here a queue with infinite space. The basic code is shown below.

`TST rs, Imm(rt)` is the test-and-set instruction, which *atomically* loads the value at `Imm(rt)` into `rs`, and if the value is zero, updates the memory location at `Imm(rt)` to 1. This atomic instruction is useful for implementing locks: a value of 1 at the memory location indicates that someone holds the lock, and a value of 0 means the lock is free.

Producer pushes a message onto queue: (memory operations in bold)

```
void push(int** tail_ptr, int* tail_write_lock, int message) {
      while (lock_try(tail_write_lock) == false);
      **tail_ptr = message;
      *tail_ptr++;
      lock_release(tail_write_lock);
}

# R1 – contains address of data to enqueue
# R2 – contains the address of the tail pointer of queue
# R3 – address of tail pointer write lock
P1 SpinLock:TST R4, 0(R3)          # try to acquire tail write lock
P2          BNEZ R4, R4, SpinLock
P3          LD R4, 0(R2)           # get tail pointer
P4          ST R1, 0(R4)           # write message to tail
P5          ADD R4, R4, 4          # update tail pointer
P6          ST R4, 0(R2)
P7          ST R0, 0(R3)           # release lock
```

Consumer pops a message off queue: (memory operations in bold)

```
int pop(int** head_ptr, int** tail_ptr) {
      while (*head_ptr == *tail_ptr);
      int message = **head_ptr;
      *head_ptr++;
      return message;
}

# R1 – will receive address contained in message
# R2 – contains the address of the head pointer of queue
# R3 – contains the address of the tail pointer of the queue
C1 Retry:   LD R4, 0(R2)           # get head pointer
C2          LD R5, 0(R3)           # get tail pointer
C3          SUB R5, R4, R5         # is there a message?
C4          BNEZ R5, Pop
C5          JMP Retry
C6 Pop:     LD R1, 0(R4)           # read message from queue
C7          ADD R4, R4, 4          # update head pointer
C8          ST R4, 0(R2)
```

Name _____

Question 1 (10 points):

You are trying to port this code to an architecture that does not have the `TST` instruction (but, happily, the rest of the ISA is unchanged). Instead the new architecture has load-reserve/store-conditional instructions. Implement `TST rs, 0(rt)` using load-reserve/store-conditional:

```
LR rs, Imm(rt):
     rs ← Memory[(rt) + Imm]
     Track address (rt) + Imm

SC rs, Imm(rt):
     If (rt) + Imm modified:
          rs ← 0                          # Fail
     Else:
          Memory[(rt) + Imm] = (rs)       # Succeed
          rs ← 1
```

Name _____

Question 2 (10 points):

This new architecture is also *not* sequentially consistent. Give an example of memory orderings between the producer and consumer that would result in incorrect behavior. *Explain your answer fully or you will not receive credit.*

Your answer should look something like:
`P1, P3, P4, C1, C2, P6, P7, C1, C2, C6, C8`
(Except that this is a sequentially consistent ordering, so it is not a correct answer.)

Name _____

Question 3 (10 points):

Show where memory fences should be added to the producer and consumer code to ensure correctness with a weak consistency model. Explain your answer fully.

```
P1 SpinLock:TST R4, 0(R3)          # try to acquire tail write lock

P2          BNEZ R4, R4, SpinLock

P3          LD R4, 0(R2)           # get tail pointer

P4          ST R1, 0(R4)           # write message to tail

P5          ADD R4, R4, 4          # update tail pointer

P6          ST R4, 0(R2)

P7          ST R0, 0(R3)           # release lock



C1 Retry:   LD R4, 0(R2)           # get head pointer

C2          LD R5, 0(R3)           # get tail pointer

C3          SUB R5, R4, R5         # is there a message?

C4          BNEZ R5, Pop

C5          JMP Retry

C6 Pop:     LD R1, 0(R4)           # read message from queue

C7          ADD R4, R4, 4          # update head pointer

C8          ST R4, 0(R2)
```

Name _____

Question 4 (5 points):

Let's next consider performance with a single producer thread and consumer thread. The following happens repeatedly:

1. The producer executes all instructions to push a message on the queue.
2. The consumer executes all instructions to pop a message off the queue.

Assume data, head, and tail pointers all lie in different, non-conflicting cache blocks.

First, after a few messages have been sent through the queue, will the consumer ever miss reading the head pointer? Will the producer ever miss reading the tail write lock, or fail to acquire the tail write lock? Explain in one or two sentences.

Question 5 (5 points):

We'll now focus on the tail pointer only. Assuming a MSI invalidate coherence protocol, show the state of the tail pointer in the producer and consumer cache after each operation in the sequence below. Show any data or permissions transfers, e.g. "Memory→C" or "C invalidates P".

Operation	**Producer tail pointer state**	**Consumer tail pointer state**	**Transfers**
	I	I	
P1 TST try lock			
P3 LD tail_ptr			
P4 ST message			
P6 ST new_tail			
P7 ST release lock			
C1 LD head_ptr			
C2 LD tail_ptr			
C6 LD message			
C7 ST new_head			
P1 TST try lock			
P3 LD tail_ptr			
P4 ST message			
P6 ST new_tail			
P7 ST release lock			
C1 LD head_ptr			
C2 LD tail_ptr			
C6 LD message			
C7 ST new_head			

How many state transitions occur per message in the steady state?

Name _____

Question 6 (5 points):

Stay focused on the tail pointer only. Assume an update coherence protocol where the state of each line is either valid (V) or invalid (I). Show the state of the tail pointer in the producer and consumer cache after each operation in the sequence below in the steady state. Show any data or permissions transfers, e.g. "Memory→C" or "C invalidates P".

Operation	**Producer tail pointer state**	**Consumer tail pointer state**	**Transfers**
	I	I	
P1 TST try lock			
P3 LD tail_ptr			
P4 ST message			
P6 ST new_tail			
P7 ST release lock			
C1 LD head_ptr			
C2 LD tail_ptr			
C6 LD message			
C7 ST new_head			
P1 TST try lock			
P3 LD tail_ptr			
P4 ST message			
P6 ST new_tail			
P7 ST release lock			
C1 LD head_ptr			
C2 LD tail_ptr			
C6 LD message			
C7 ST new_head			

How many state transitions occur per message in the steady state?

Name _____

Question 7 (10 points):

Your new architecture supports "remote access" for cached lines. This lets you assign a "home cache" for lines so that all memory operations will be sent *over the network* to operate remotely on the line *without allocating it in the requesting cache*.

For example, if line 0x100 is homed to processor A, and processor B writes 0x100, then *processor A's cache will be updated* and processor B's will be unchanged.

Assume the tail pointer is mapped to the producer's cache, and the cache uses an MSI invalidate protocol (similar to Question 5). Once again, show the state of the tail pointer for the sequence of operations in the steady state and data/permission transfers:

Operation	Producer tail pointer state	Consumer tail pointer state	Transfers
	I	I	
P1 TST try lock			
P3 LD tail_ptr			
P4 ST message			
P6 ST new_tail			
P7 ST release lock			
C1 LD head_ptr			
C2 LD tail_ptr			
C6 LD message			
C7 ST new_head			
P1 TST try lock			
P3 LD tail_ptr			
P4 ST message			
P6 ST new_tail			
P7 ST release lock			
C1 LD head_ptr			
C2 LD tail_ptr			
C6 LD message			
C7 ST new_head			

How many state transitions occur per message in the steady state?

Name _____

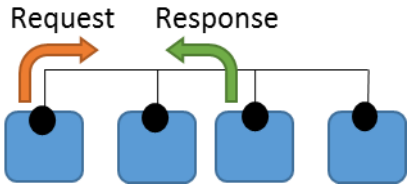
Part III: The Truth Will Set You Free (20 points)

Question 1 – Peanuts (5 points):

Snoopy coherence protocols rely on broadcast communication to detect sharing and updates. These are conventionally implemented using bus networks that allow for one message to be sent at a time to all nodes on the network.

Ben Bitdiddle is implementing a bus-based snoopy coherence protocol. One fifth of instructions access memory, and one quarter of these miss in the core's local cache (either because the line is invalid or doesn't have necessary permissions). Assuming each memory operation consists of a request and acknowledgement, the network traffic per core is therefore: $\frac{1}{5} \times \frac{1}{4} \times 2 = \frac{1}{10} \frac{\text{messages}}{\text{instruction}}$. Assume all messages fit within a single network flit.

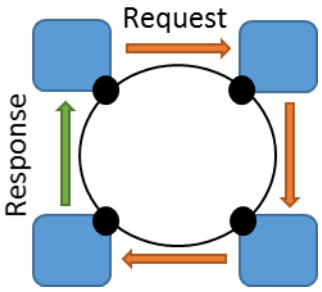
Assuming a fixed IPC of 1, perfect bus arbitration, and infinite buffers, how many cores can the bus support?



Name _____

Question 2 – …To rule them all (10 points):

Ben needs to build a larger system than the bus network will allow, so he changes the system to use a unidirectional ring network. In this design, the core issuing the memory operation sends the request around the ring, and each node along the way either forwards the request or replaces it with its response. Assuming fixed IPC of 1 and a single-cycle per hop in the network, at how many cores will this design saturate?



Name _____

Question 3 – Matryoshka (5 points):

Ben next explores the tradeoffs in cache design between an inclusive cache, where the parent always has a copy of every line in the child's cache, and non-inclusive caches, where this isn't guaranteed.

Give one advantage and one disadvantage of a non-inclusive cache design.