ELL 305 (Computer Architecture)
Major Exam
Ist Semester 2019-20
Time: 2 hours

<u>Maximum Marks:</u> 75

Please write your name in capital letters clearly.

Name:	Entry Number:

1. Answer the following questions briefly: **[5 x 3 marks]**

a. What are the three kinds of misses? (the three Cs) Define them.

Compulsory Misses: Misses incurred when a block is being read for the first time.

Conflict misses: Misses incurred when more than k blocks map to the same set in a K-way set associative cache.

Capacity Misses: Misses incurred when the working set of the program is larger than the size of the cache.

Note to TAs: 1 mark each.

b. If a system has infinite physical memory, does it still need virtual memory? Justify.

Yes, we still do (1 mark)
This is because we still can have the problem of two addresses in two different programs mapping to the same physical location. Even if the compiler somehow manages to separate the address spaces, we can always run two copies of the same program. Additionally, it is a security loophole, because we can read their values written by other programs.

c. Is sequential consistency practical? Why or why not? Justify your answer.

No it is not (1 mark).
It requires us to finish a memory operation completely, and only then issue the next memory instruction. This will make our pipeline very slow. Now we just hand over a piece of data to the memory system and the pipeline proceeds. If we want SC, this cannot be done.

d. Are hard disks good storage structures for files that are randomly accessed? A file is said to be randomly accessed when every access is to a random location in a file. The accesses do not have a pattern; there is no spatial locality. Justify your answer.

No (1 mark)

Reason: Hard disks use spinning platters. The disc head moves sequentially from cylinder to cylinder. As a result hard disks are good for sequential access. However, if we consider random accesses then we will spend a lot of time in *seeking* the new location.

e. What is the advantage of RAID 5?

1. It stores a parity block for each set of blocks. (1 mark)
2. The parity block is distributed across the disks. There is no single point of contention. (2 marks)

2. Derive the performance model of a memory system. **[2 x 5 marks]**

a. Derive the equation for the AMAT. Consider a system with an L1, L2, and L3 cache.

AMAT = L1_hittime + L1_missrate * (L2_hittime + L2_missrate * (L3_hittime + L3_missrate * main_memory_acc_latency))

Name:	Entry Number:

TAs: Cut one mark per mistake. Even a lack of justification is a mistake. The justification is required for both the parts (a) and (b).

Bonus of 1 mark, if they have added the TLB miss penalty to this equation (not a strict requirement though).

b. Derive the equation for the new CPI.

CPI_new = CPI_old + frac_mem_insts * (AMAT – L1_hit_time)

TAs: Cut 2 marks per mistake.

Here also the justification is required.
It should ideally be AMAT – L1_hit_time

Decide the grading scheme, and be consistent.

Name:	Entry Number:

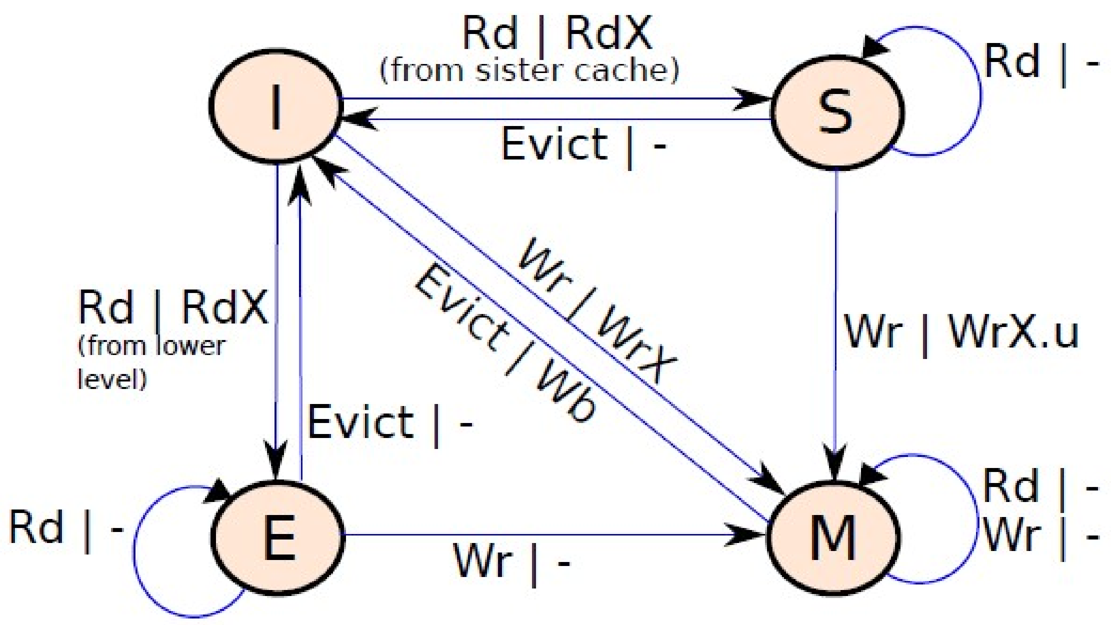
3. Assume that we have a write-invalidate coherence protocol on a Snoopy bus. We want to add an additional state: **Exclusive**. A block is said to be in the *exclusive* state when it is present with only one cache. That cache can only read the block. It needs to transition to the modified state if it wants to write to the block. **[3+10 marks]**

a. What is the benefit of the exclusive state? **[3 marks]**

1. It can be seamlessly evicted. (1 mark)
2. We can seamlessly transition to the modified state. There is no reason to sign any message on the bus. (2 marks)

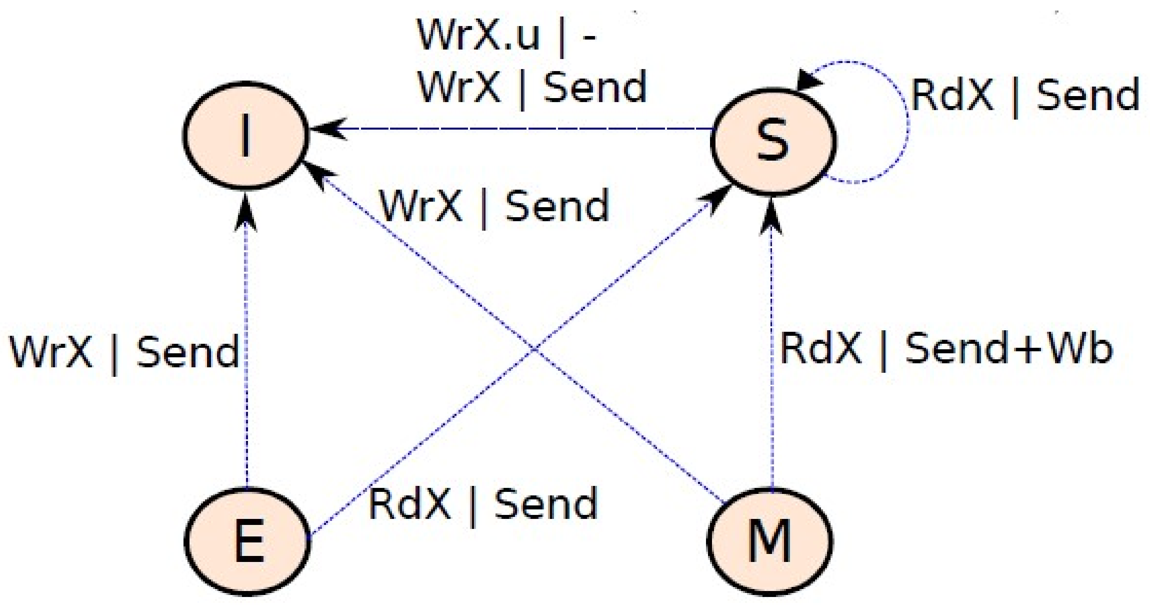
b. Draw the state diagrams for the new protocol that has four states: M, E, S, and I. (Transitions for messages received from the processor, and messages received from the bus) Justify all your decisions. **[10 marks]**

RdX → Read miss, WrX → Write miss, WrX.u → Write miss (don't need the data)



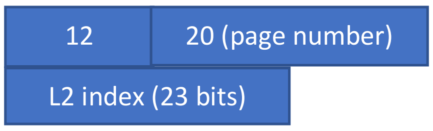
Name:	Entry Number:

TAs: 5 marks per diagram. Cut 0.5-1 marks per wrong transition. Rest on a case-by-case basis.



4. I claim that the mapping between virtual and physical address determines the L2 cache miss rate. This is in a typical system where the L1 cache is 32 KB, and the L2 cache is 8 MB. Assume all memory addresses are 32 bits. **Hint:** See the number of bits we require to address a page, and an L2 cache line (any connection?). **[3 x 4 marks]**

a. Prove that I am right. Draw a simple diagram that shows the bits we get from the virtual to physical mapping, and the bits that we use to access the L2 cache. Notice the connection.



1. There is an overlap of 11 bits between the LSB of the frame address and the L2 address (assuming it to be 1-way associative for the sake of simplicity).
2. These 11 bits are determined by the mapping engine. If these bits are random then the accesses will be distributed, otherwise they will map to the same set of locations (high miss rate).

b. How can we optimize the mapping (virtual $\rightarrow$ physical) to maximize the L2 hit rate?

Ensure that these 11 (23 – 12) bits are as uniformly distributed as possible. This means that the page mapping algorithm in the OS needs to ensure that we are generating a random sequence.

c. If a given application has a small working set (accesses a few pages only), can we do something better?

We can specifically ensure that those pages are mapped to different L2 cache sets. We can minimize the probability of a conflict miss.

Name:	Entry Number:

5. The off-chip main memory does not have a tag array. It has only a data array. Assume a 32-bit memory system, and a main memory that is smaller than 4 GB. **[2 x 5 marks]**

a. Is this a problem? If yes, what is the problem, and what issues can it cause unless something is done? Note that whenever two different addresses map to the same location, we need a tag that helps us disambiguate the addresses. In this case we do not have a tag.

If we make the naïve mistake that the physical address space is 32 bits, then there is a problem. This is because the physical address is 4GB, and the size of the main memory is 1 GB. We thus have four blocks mapping to each block in main memory. We definitely do need a tag, if we assume that our physical address is 32 bits. This is an issue, and this solution will not work.

b. How does the virtual memory system take care of these "problems"?
[HINT: Think of an example. A 32-bit memory system, a 1 GB main memory. How will you map the addresses to main memory? Any problems henceforth?]

The virtual memory system assumes that the physical address space is log_2(size of the main memory). In this case it is 30 bits. It maps each page number to a unique (30-12=18) bit integer. Each integer is the id of a frame in memory. This is a unique id, and there are no chances of ambiguity.

6. Consider the following piece of code: **[HARD]** **[3 x 5 marks]**

```
void lock(){
        int tid = get_thread_id();
        interested[tid] = 1;
        turn = tid;
        while ( (interested[1-tid] == 1) && (turn == tid)) {}
}
void unlock(){
        int tid = get_thread_id();
        interested[tid] = 0;
}
```

Here, tid is the thread id. There are only two threads in the system. The tid can either be 0 or 1. *interested* and *turn* are global variables.

This piece of code is used to get **exclusive access** to a piece of code. For example:

```
lock();
… piece of code …
unlock();
```

a. Prove that in a sequentially consistent system it is not possible for 2 threads to simultaneously execute the piece of code protected by the lock-unlock function call.
b. Does this code work in a weakly consistent system? Justify.
c. What changes need to be made to this code for it to work in a weakly consistent system? Where do you add the fences?

HINT:

1. Prove by contradiction. Assume that both the threads have entered the protected code at the same time.
2. With that assumption, work backwards. Both must have crossed the while loop.
3. This means that for both the threads one of the conditions must have been false.
4. If two threads are modifying the same variable, one must have modified the variable later (order axiom of coherence).
5. Only three statements in the code are of any value: interested[tid]=1, turn=tid, and the while loop. Let them be statements 1, 2, and 3. Look at the *happens before* ordering between these statements (directed graph).
6. A graph with happens-before edges cannot have a cycle. Both the statements cannot be simultaneously true: *a* happens before *b, b* happens before *a*.

Name:	Entry Number:

a. Assume SC $\rightarrow$ There are three instructions in the code: 1, 2, and 3.

Draw a graph like this, for threads 1 and 2.

$$\begin{tikzcd}
\boxed{1} \arrow[r] & \boxed{2} \arrow[r] \arrow[d] & \boxed{3} \\
\boxed{1'} \arrow[r] & \boxed{2'} \arrow[r] & \boxed{3'}
\end{tikzcd}$$

Let the events corresponding to thread be 1, 2, and 3, and the events corresponding to thread 2 be 1', 2', and 3' respectively. Because of our assumption (w.l.o.g) we have an arrow from 2 to 2'.

Now, in 3' (while loop for thread 2), one of the conditions must have been false. It cannot be (turn == tid) because thread 2 was the last to set the value of turn. It must be interested[1-tid] == 1.

This means that it must have read interested[1-tid] to be 0 (before it was set). This can only happen if 3' *happened before* 1. We shall thus have an arrow from 3' to 1. This will lead to a cycle in the happens-before graph. This is not possible. We shall thus have a contradiction. Hence, in this case two threads cannot enter the protected region simultaneously.

b. No it does not.
   This is because there are no arrows of the form 1$\rightarrow$ 2 $\rightarrow$ 3. Both the threads can read interested[1-tid] to be false.

c. Fix: Add a fence after every write in the lock and unlock codes.