CS 433 Final Exam: May 14, 2025

Professor Sarita Adve

Time: 3 Hours

Please print your Name and NetID and circle your course section below.

Name:		
NetID:		
Section:	S3 (Undergraduate)	S4 (Graduate)

Instructions

1. **Please write your Name and NetID on the top right corner of all pages in this exam.**
2. No books, papers, notes, or any other typed or written materials are allowed. No calculators or other electronic materials are allowed.
3. Please do not turn in loose scrap paper. Limit your answers to the space provided if possible. If this is not possible, please write on the back of the same sheet. You may use the back of each sheet for scratch work.
4. *In all cases, show your work. No credit will be given if there is no indication of how the answer was derived. Partial credit will be given even if your final solution is incorrect, provided you show the intermediate steps in reaching the final solution.*
5. If you believe a problem is incorrectly or incompletely specified, make a reasonable assumption and solve the problem. The assumption should not result in a trivial solution. In all cases, clearly state any assumptions that you make in your answers.
6. This exam has **6 problems** and **18 pages** (including this one). All students should solve problems 1 through 6B. **Only graduate students should solve problem 6C.** Please budget your time appropriately. Good luck!

Problem		**1**	**2**	**3**	**4**	**5**	**6**	**Total**
Points	**S3**	15	11	12	10	4	14	66
	S4	15	11	12	10	4	22	74
Score								

Name:
NetID:

Problem 1 [15 points]

Consider the following code:

```
double A[256][256];
double x = 0.0;

for (int i = 0; i < 64; i++) {
     for (int j = 0; j < 64; j++) {
          x += A[2*i][2*j];
     }
}
```

Assume the following:

- Array A contains double words (i.e., each element is 2 words long).
- Only accesses to array locations generate loads to the data cache. The rest of the variables are allocated in registers.
- The array A is stored in row-major order.
- The program is running on a machine with an L1 data cache that has 32 words per cache line.
- Memory is word addressable.
- Assume the array A starts at a cache line boundary.
- Assume an infinite data cache that is initially empty.

Part A [2 points]

How many L1 data cache misses occur for the above code? You must explain how you derived this number to receive any credit.

Part B [2 points]
Suppose our machine has a data prefetch instruction with the format prefetch(array[i][j]). This prefetches the entire cache line containing the word array[i][j] into the data cache. Assume it takes one cycle for the processor to execute this instruction and send it to the data cache. The processor can then execute subsequent instructions.

Consider inserting prefetch instructions for the inner loop of the above code; i.e., ignore the outer loop for this part. Explain why we may need to unroll the inner loop to insert prefetches. What is the minimum number of times you would need to unroll the loop for this purpose?

Part C [4 points]
Modify the original program by unrolling the inner loop the number of times identified in Part B and insert the minimum number of software prefetches to minimize execution time. The technique to insert prefetches is analogous to software pipelining. You do not need to worry about startup and cleanup code (i.e., epilog and prolog) and do not introduce any new loops. Assume that two iterations of your unrolled inner loop are sufficient to hide the full latency of an L1 cache miss (i.e., if you suffer a cache miss on a load in unrolled loop iteration i, the value for that load will be available in unrolled loop iteration i+2). Again, for this part, you may ignore the outer loop.

Note that if you obtained an incorrect answer for Part B that results in a trivial solution to this part, you will not get any credit for this part. So be careful about Part B. Again, the goal is to minimize the number of unnecessary prefetches (ignoring epilog/prolog issues and the outer loop).

Part D [3 points]

Consider your solution to Part C. How many of the data cache misses from the original code now have their latency fully hidden? Consider both the inner and outer loops for this part. You must explain how you derived this number to receive any credit. If an incorrect solution to Part C trivializes this problem or makes it too difficult, no credit will be given.

Part E [4 points]
Consider the data cache misses for which the latencies are not fully hidden in Part C. Suggest one enhancement to your code in Part C that can hide some of the exposed latency. Consider both the inner and outer loops for this part. You may answer with a clear description in words or with code that contains the enhancement accompanied with some explanatory text. Again, solutions to previous parts that trivialize this part will not get any credit.

Name:
NetID:

Problem 2 [11 points]

Consider a computer with a memory system with 16 bits for physical address, 32 bits for virtual address, page size of 4KB, 16 bit words, and word (16 bit) addressable memory. The computer system contains a 32KB data cache that is virtually indexed and physically tagged. The data cache is 8-way set associative and the cache line size is 16 words.

Part A [2 points]:
Specify what bit ranges of the address (virtual or physical) comprise the virtual page number and the physical page frame number. How many physical and virtual pages does this system have?

Part B [3 points]:
Assume each Page Table Entry (PTE) has 5 state bits (dirty bit, protection bits, etc.) in addition to the address translation. How many bits does each PTE consume? How much memory, in bytes, does the page table consume to map all the virtual pages? Assume only one level of page translation (as discussed in class) and assume each page table entry is word aligned.

Part C [3 points]

Specify what bit ranges within the virtual or physical address (whichever is appropriate) comprise the cache index and cache tag bits. State explicitly whether the bits used are from the physical or virtual address.

Part D [3 points]

If the architect were to decrease the cache associativity to 4-way, what would the cache index bits be then? Would this create a problem for the Virtually Indexed/Physically Tagged caching scheme? If so, describe the problem and a hardware-only way of solving it that does not require waiting for address translation before indexing the cache.

Name:
NetID:

Problem 3 [12 points]

This problem concerns MESIF, an invalidation-based snooping cache coherence protocol for bus-based shared-memory multiprocessors with a single level of cache per processor. The MESIF protocol has five states. A block can be in one of the following states in cache C:

- **M** *Modified*: The block is present in a single cache C. The block is dirty or modified; i.e., the data in the block reflects a more recent version than the copy in memory.
- **E** *Exclusive*: The block is present in a single cache C. The block is clean; i.e., memory has an up-to-date copy of the block.
- **S** *Shared*: The block is present in cache C and possibly present in other caches. The block is clean.
- **I** *Invalid*: The block is not valid in cache C. Space for the block may or may not be currently allocated in this cache.
- **F** *Forward*: The block is present in cache C and possibly present in other caches. The block is clean. Additionally, cache C must service the requests of other caches to this block; memory will not respond to requests for this block. Only one cache can have the block in Forward state.

If cache C has a block A in the Modified, Exclusive, or Forward state, then cache C responds to any requests for block A from other processors. If the request is a read, then the state of block A in cache C transitions to the Shared state, while the new copy of block A in the requester's cache assumes the Forward state. If cache C had block A in the Modified state, then memory updates its copy as well.

On replacement of a block in the Modified, Exclusive, or Forward state, memory resumes responsibility for servicing subsequent requests for that block.

For this problem, assume a system with three processors – P1, P2, and P3 – each with a single level cache that stores only a single block and starts empty.

Complete the following table for the MSI protocol studied in class and for the MESIF protocol described above. The first column gives memory accesses (Read or Write) initiated by P1, P2, or P3 to block A. For each protocol, you are to fill the entry for the states of block A in the caches of P1 and P2 after the access in the first column completes. You are also to fill the entry for the actions on the bus initiated by the processors or Memory (Mem) required to complete the access in column 1.

The possible states are: M, E, S, I, and F.

The possible bus actions are X / Y, where:

- X is the initiator of the action and can be: *P1, P2, P3, Mem, or None*
- Y is the action initiated by X and can be: *Get Block A, Send Block A, Invalidate, or None*

Name:
NetID:

For each entry, there can be multiple actions, possibly by multiple initiators.

Assume that the accesses occur in the order below and that no other memory accesses / traffic occur. The first row is filled out for you.

	MSI			**MESIF**		
Access on block A	**P1 State**	**P2 State**	**Bus Actions**	**P1 State**	**P2 State**	**Bus Actions**
P1 Read	S	I	P1 / Get Block A Mem / Send Block A	E	I	P1 / Get Block A Mem / Send Block A
P2 Read						
P3 Read						
P2 Write						
P2 Write						
P1 Write						
P2 Read						

Problem 4 [10 points]

You are to implement a queue using an array in a multiprocessor system. The elements of the array can be accessed in parallel by multiple processors. You are to write two functions:

- ***enqueue***, which will add an element to the tail of the queue, and
- ***dequeue***, which will remove an element from the head of the queue.

Consider the following function definitions:

```
int head; /* index for the head of the queue */
int tail; /* index for the tail of the queue */
int local-index; /* current index for enqueuing or dequeuing,
each processor has its own copy */

enqueue(item) {

     queue[local-index] = item;

}

dequeue() {

     item = queue[local-index];

     return item
}
```

Assume the queue always has at least one element; i.e., a *dequeue* is never called on an empty queue. Also assume that the queue never gets full; i.e., the array is infinitely long and you don't have to worry about calling an *enqueue* on a full queue.

Name:
NetID:

Part A [6 points]:

Implement the *enqueue* and *dequeue* functions using an atomic *test&set* instruction to achieve synchronization. Don't worry about using test&test&set, but otherwise, write the most efficient code possible. Use C-like pseudocode to implement the functions. Assume local-index is local to each processor and is not part of shared memory; i.e., each processor has its own copy of this variable (e.g., in its own register). Assume the system implements sequential consistency.

Part B [4 points]:

Repeat Part A using the atomic *fetch&increment* instruction instead of the test&set for synchronization.

Problem 5 [4 points]

This problem concerns the mini-project presentations in class. Circle the most appropriate choices for each question below. Some questions have multiple correct choices. Points will be given for a question only if all appropriate choices for that question are circled and no incorrect choice is circled.

Part A [1 point]
Which of the following branch predictors are used in AMD Zen 2?

a) Simple neural network (Perceptron)
b) Static, always-taken
c) G-share predictor
d) TAGE predictor

Part B [1 point]
Which of the following is true about Intel Lunar Lake?

a) It is built out of multiple smaller chiplets rather than one large monolithic chip
b) Its performance benefits come from Simultaneous Multithreading (Hyperthreading)
c) It includes a Neural Processing Unit dedicated for AI workloads
d) Its CPU includes 1000s of lightweight cores to achieve massive parallelism

Part C [1 point]
Which of the following is true about IBM POWER 10?

a) It moves toward a RISC architecture with 50 times the numbers of cores as POWER 9
b) It is the first IBM product with 3D stacking of 16 compute and 58 memory layers
c) It can be connected with 16 other POWER 10 chips to form a fully coherent system
d) Its CPU includes 1000s of lightweight cores to achieve massive parallelism

Part D [1 point]
Which of the following accelerators were present in the NVIDIA Ada Lovelace architecture?

a) Genomic sequencer
b) Tensor core
c) Ray tracing accelerator
d) Merge sort accelerator

Name:
NetID:

ALL STUDENTS SHOULD SOLVE PARTS A TO C.
ONLY GRADUATE STUDENTS SHOULD SOLVE PART D AND E.

Problem 6 [14 points for undergraduates, 22 points for graduates]

For this problem, consider a system that implements sequential consistency. You may assume that uniprocessor control and data dependencies are always respected. The following code fragment is executed on three processors:

```
                Initially X = Y = 0
         P1                P2                P3
    -------------------------------------------------
       X = 1             B = X             C = Y
       A = X             Y = 2             X = 3
```

Part A [8 points]

Consider an execution of the program where the **final value of C = 2** and fill the table below. For each row:

- In the third column, indicate (by writing *Yes* or *No*) whether the corresponding combination of the final value of A and B (and C = 2 as specified earlier) is possible.
- In the fourth column, justify your (Yes/No) answer in the third column with a reason. You may answer in words or justify with an example execution order of the program.

Final value of A	**Final value of B**	**Combination of A and B possible?**	**Reason for Yes/No answer in Column 3**
0	0		
1	1		
1	3		
3	1		

Part B [2 points]
What synchronization is required to ensure that all of P1's instructions are to be executed before any of P2's instructions, and all of P2's instructions are to be executed before any of P3's instructions? Modify the given program by inserting new (synchronization) instructions.

Do not reuse any of the variables in the original data accesses of the program to perform synchronization. Unnecessarily inefficient solutions will not get full credit.

Part C [4 points]

Consider a system that implements a relaxed consistency model, where any program-ordered pair of accesses to different variables may appear reordered to an external observer, except as restricted by a `fence` instruction. A fence instruction ensures all previous (by program order) reads and writes are complete to any observer in the system before subsequent (by program order) accesses can begin. You may assume that uniprocessor control and data dependencies are still respected.

Starting with the solution from Part B, modify the program to **introduce fence instructions** such that the values of A, B, and C at the end of an execution of the modified program on the system that implements the relaxed consistency model described above are the same as Part B (i.e., a system that implements sequential consistency). Ensure that fence instructions are used judiciously (only where necessary). Solutions with unnecessary fence instructions will not get full credit. Full credit for this part requires a reasonable answer to Part B.

ONLY GRADUATE STUDENTS SHOULD SOLVE PART D AND E.

Part D [4 points]

Describe how the use of a write buffer for Processor 1 could potentially violate sequential consistency for your solution in Part C, if the system ignored the fence instructions. Full credit will be given only if your solution in Part C is reasonable.

Now describe how you would consider the fence instructions in the design of the write buffer to avoid the above violation of sequential consistency.

Part E [4 points]
Again, consider your solution in Part C. Processor 3 is designed with an out-of-order pipeline employing Tomsaulo's algorithm with speculative execution of loads. Describe how this could potentially violate sequential consistency for your solution in Part C, if the system ignored the fence instructions. Full credit will be given only if your solution in Part C is reasonable.

Again, describe how you would consider the fence instructions in the design of the above pipeline for Processor 3 to avoid the above violation of sequential consistency.