YOUR NAME: _____

B-Number: _____

By signing this page, I am acknowledging that the work I am submitting for this exam was done exclusively me without the cooperation of other individuals:

Your Signature and Date: _____

CS 520 – Fall 2020 – Ghose
Exam 1: Mon, Oct. 26, 2:20 pm to 4:20 pm – INCLUDES 15 Minutes for Uploading Submission to Blackboard. Open Book, Open Notes

Suggested Length of answers are given for each problem. PLEASE *DO NOT* BE TOO VERBOSE AND STICK TO THE ANSWERS, INCLUDING EXAMPLES

ANSWER ALL 4 PROBLEMS

PLEASE KEEP YOUR CAMERA ON AND FOLLOW DIRECTIONS ANNOUNCED ON BLACKBOARD

Problem 1 (4 Points Each, Total 40 Points). Answer if each of the following statements is true or false. You can answer these questions on a separate piece of paper or use the empty column to the right in the table below to provide your answer and then send a scan of the page in PDF.

Indicate if each of the statements below are True or False	Answer
a) In a simple liner pipeline, like the 5-stage APEX pipeline, data forwarding can completely remove pipeline bubbles caused by data dependencies where the producer instruction is a register-to-register instruction and has a single-cycle execution latency.	
b) Software pipelining, where the loop is unrolled to facilitate software pipelining, produces binaries that require the use of additional architectural registers compared to the original, unrolled loop.	
c) All pipelined implementations of a given ISA should produce results that are consistent with the sequential execution model.	
d) If LOADs bypass earlier STOREs in the LSQ when bypassing conditions are satisfied, the compliance with the sequential execution model is not violated. (LOAD bypassing is defined on the next page)	
e) The use of a separate physical register file in out-of-order processor like the one shown for **Variation 2** (Page 203 in the notes) makes the individual ROB entries wider, compared to designs that are like Variation 1 (Page 202).	
f) For an out-of-order processor like the one shown as **Variation 3** (Pages 203-204 of the notes), the retirement register alias table (R-RAT) entries *may sometimes* point to the most recent instances of architectural registers created by instruction dispatches.	
g) The maximum level of speculation in an out-of-order processor that supports speculative execution (Page 271 of notes) is limited by data dependencies in the binaries that are run on the processor.	
h) Cache misses triggered by LOAD instructions can delay branch resolution.	
i) STORE instructions can write to memory in a processor that uses speculative execution as soon as the data and the memory address are both available.	
j) Branch mispredictions can take a serious toll on the performance of processors that use speculative execution.	

(OVER)

Explanation for 1(d): LOADs bypassing earlier STOREs in the LSQ:

A LOAD instruction can start memory operations before earlier STOREs queued up in the LSQ if it is known for certain that the memory address that will be accessed by the LOAD does not get updated by ANY of the STOREs that are ahead of it in the LSQ.

Problem 2. (20 Points, 1 Page) The ROB in an out-of-order processor can ensure that memory instructions (loads and stores) can access memory in the sequential execution order. In spite of this, most real-world out-of-order processors use a separate LSQ to maintain the program order of memory instructions. Give ***two reasons*** that justify the use of a separate LSQ.

Problem 3. (40 Points, 2 Pages) The ***renamer*** of an instruction, say I, that has an architectural register (say, R) as the destination is defined as the next committed instruction (say, J) that has the same architectural register (R) as its destination. I is called the ***renamee*** of J. Consider an out-of-order processor like the one shown as Variation 3 on Pages 203-204 of the notes. For this processor, the physical register allocated as the destination of an instruction (I) is freed up when the renamer instruction (J) commits. As an example, consider the ROB entries for following three instructions in the ROB, with the ADD instruction located at the current head of the ROB:

```
ADD R5, R2, R19     /* allocated physical register P9 for the destination */
SUB R8, R2, R5      /* allocated physical register P2 for the destination */
MUL R5, R1, R4      /* allocated physical register P14 for the destination */
```

The MUL instruction is the renamer for the ADD, as they have the same architectural register (R5) as destination. The ADD is the renamee of the MUL. The physical register P9 allocated to the ADD is freed up when the renamer MUL commits.

For a processor described above:

a) Provide arguments that this technique for physical register deallocation works correctly, that is, it meets the two conditions for freeing up a physical register. You can use the code fragment given above to explain your answer.

b) Propose a format for the ROB entry for register-to-register instructions.

c) Use the C-like pseudocode of the notes to show how register-to-register instructions are committed.

d) Are there performance issues introduced by this particular approach of releasing a physical register?

If you have to make any assumptions, please state them clearly. Please provide adequate explanations and use short code fragments, if need be, to explain your answer.

Problem 4. (30 Points, 1.5 Pages) For this problem, you are to design a branch predictor that retains the history of its past 4 executions in the BTB and predicts a branch to be taken only if the most recent execution results in a taken branch and if no more than (that is, at most) one of the three executions prior to the most recent one was not taken. For this predictor, answer the following questions:

a) What information will be stored in the BTB for this predictor?

b) What is the logic circuitry used for deriving the branch prediction from the information stored in the BTB?