CS 433 Midterm **Practice** Exam – Fall 2020

Student name:
Netid:
Grad or undergad:

Instructions

1. Connect to the exam zoom room before you download the exam from compass. Set your camera as instructed - with your hands, face, and the screen of the device you use to write your solutions visible on zoom - from before you download the exam to after you submit it on compass.
2. The exam is designed to be solved within two hours. We have allocated a four hour slot to allow for the online format.
3. No books, papers, notes, or any other typed or written materials are allowed. No calculators or other electronic materials are allowed.
4. Download the exam from compass 2g and save a copy on your device. It is acceptable to convert the word file into a google doc. Type your answers on the saved file/google doc (start by typing your name, netid, and grad/undergrad status above). Try to keep your answer within the space provided if possible. After you finish editing the document with your solutions, please save it, convert it to pdf, and upload the pdf back to Compass.
5. *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.*
6. 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.
7. If you need to ask a question, please use the "raise hand" feature on zoom or send a private message on zoom chat to Antonio and we will take you to a breakout room.
8. This exam has ***6 problems*** and **11 *pages*** (including this one). **All students should solve problems 1 through 5. Only graduate students should solve problem 6.** Please budget your time appropriately. Good luck!

Problem	Maximum Points
1	7
2	15
3	9
4	18
5	9
6	6
Total	58 for undergraduates 64 for graduates

Problem 1 [7 points]

Identify all the data dependences (potential hazards) in the code below. Specify the associated instructions, registers, and whether the dependence is a RAW, WAW, or WAR. State whether the dependence will cause a stall. Consider the 5-stage MIPS pipeline. Branches are resolved in the ID stage. All stages take 1 cycle. Assume full forwarding. If the identified pair does not form a dependence at all, you will get negative points (so don't give all possible instruction pairs!).

1: LD R1, 0(R6)

2: LD R2, 4(R6)

3: ADD R3, R2, R1

4: SUB R2, R2, R1

5: BEQZ R2, dest

Problem 2 [15 points]

This problem concerns Tomasulo's algorithm (with reservation stations) with the reorder buffer as discussed in detail in the lecture notes, with the following changes/additions/ clarifications.

Functional Unit Type	Cycles in EX (NON-PIPELINED)	Number of Functional Units
Integer	1	1
FP Add/Subtract	3	1
FP Divide	6	1

1) Assume dual issue and dual commit.
2) Assume unlimited reservation stations.
3) Loads use the integer function unit to perform effective address calculation during the EX stage. They also access memory during the EX stage. Loads stay in EX for 1 cycle.
4) If an instruction moves to its WB stage in cycle x, then an instruction that is waiting on the same functional unit (due to a structural hazard) can start executing in cycle x.
5) An instruction waiting for data on the CDB can move to EX in the cycle after the CDB broadcast.
6) Only one instruction can write to the CDB in one clock cycle. Branches/stores do not need the CDB.
7) When there is a conflict for a functional unit or the CDB, assume that the oldest (by program order) instruction gets access, while others are stalled.
8) Assume that the result from the integer functional unit is also broadcast on the CDB and forwarded to dependent instructions through the CDB (just like any floating point instruction).

Complete the **blank** entries in the following table using the above specifications. For each instruction, fill in the cycle numbers in each pipeline stage (CM stands for commit). For the last column, enter all the reasons that the instruction experiences a stall (leave blank if there is no stall). The first row is filled for you.

No	Instruction	IS	EX	WB	CM	Reason for stall
1	L.D F0, 0(R1)	1	2	3	4	---
2	ADD.D F0, F0, F3					
3	DIV.D F8, F0, F6					
4	L.D F6, 8(R1)					
5	ADD.D F4, F6, F2					
6	DIV.D F4, F6, F2					
7	L.D F6, 16(R1)					

Problem 3 [9 points]
Consider the following piece of code:

```
        DADDI  R1, R0, #100

L1:     DADDI  R1, R1, #-1
        BEQZ   R1, END      -- Branch 1
        DADDI  R12, R0, #2

L2:     DADDI  R12, R12, #-1
        BNEZ   R12, L2      -- Branch 2

        J      L1

END: ...
```

Assume R0 stores 0. Branch 1 is executed 100 times and branch 2 is executed a total of 198 times. For each branch, how many correct predictions will occur if we use the following prediction schemes (as discussed in class)? Assume separate prediction bits for each branch (i.e., no aliasing). Assume at the beginning of execution, the last branch was not taken. Please explain your answers.

Part A [3 points]

1-bit predictor initialized to T (taken).

Part B [3 points]

2-bit saturating predictor initialized to 10 (taken).

Part C [3 points]

(1, 1) global correlating predictor, initialized to T/T.

Problem 4: [18 points]

Consider the following code fragment:

```
Loop:  LD.D      F1, 0(R1)
       LD.D      F2, 0(R2)
       MUL.D     F3, F1, F10
       ADD.D     F4, F3, F2
       SD.D      F4, 0(R1)
       DADDUI    R1, R1, #8
       DADDUI    R2, R2, #8
       BNE       R1, R3, Loop
```

Consider a pipeline with the following latencies: 3 cycles between an FP multiply and its consumer, 1 cycle between an FP add and its consumer, and 0 cycles between all other pairs. Thus, there should be three stall cycles between the multiply and addition in the above code for correct operation. Assume that all functional units are pipelined.

Unroll the above loop 4 times and write the resulting code to the left of the table on the next page (the above loop is repeated on the next page for your convenience). You have access to temporary registers T0…T63. Assume that the total number of iterations for the original loop is a multiple of 4. Schedule the unrolled loop for a VLIW machine where each VLIW instruction can contain one memory reference, one FP operation, and one integer operation. Write the scheduled instructions in the table on the next page to minimize the number of stalls. You may use L for L.D, M for MUL.D, etc.

```
Loop:  LD.D      F1, 0(R1)
       LD.D      F2, 0(R2)
       MUL.D     F3, F1, F10
       ADD.D     F4, F3, F2
       SD.D      F4, 0(R1)
       DADDUI    R1, R1, #8
       DADDUI    R2, R2, #8
       BNE       R1, R3, Loop
```

Mem	FP ALU	Integer ALU

Problem 5: [9 points]
Consider the classic five stage pipeline as discussed in class, with the following extensions/clarifications:

- The MEM stage takes 4 cycles but is pipelined.
- Branches are resolved in the decode stage, and have one branch delay slot.
- All instructions take one cycle in the EX stage. Address calculation for memory instructions occurs in the EX stage as in the basic pipeline.
- Assume all forwarding paths as needed.

Consider the loop below. It calculates the sum of a list of numbers stored in an indirect representation. Instead of storing the number, the locations 0(R1), 4(R1), 8(R1), etc. contain the addresses of the values. This is similar to the memory access pattern of sparse matrix codes. The sum is accumulated in F1.

```
loop:
1)      LD R3, 0(R1)
2)      LD.D F2, 0(R3)
3)      ADD.D F1, F1, F2
4)      SUBIU R2, R2, #1
5)      BNEZ R2, loop
6)      ADDIU R1, R1, #4
```

Part A [3 points] List all the stalls incurred by the above loop and the reason for the stalls. If a dependence results in multiple stall cycles, indicate the number of cycles.

Part B [4 points]
Software pipeline the loop to minimize the stalls. Assume infinite registers are available. Only the most efficient solution will fetch a perfect score. Only provide the steady state code. Do not worry about start-up and finish-up code. (Do not unroll the loop for this part.).

Part C [1 point]
What is the advantage of using software pipelining over loop unrolling for the above code?

Part D [1 point]
What is the advantage of using loop unrolling over software pipelining for the above code?

ONLY GRADUATE STUDENTS SHOULD SOLVE THE NEXT PROBLEM.

Problem 6: [6 points]

You are a member of a team designing an out-of-order processor with dynamic scheduling and speculative execution. Your initial design was just reviewed by the circuit implementation team, and it turns out that you have some spare transistor budget! (A rare occurrence in practice.)

Your processor currently has a small 2-bit saturating counter-based branch predictor which performs moderately well. It has 8 Integer Functional Units and 4 Floating Point Units (FPUs), 256KB of on-chip caches, 4 reservation stations for the Integer Units, and 2 reservation stations for FPUs. The Reorder Buffer has 8 entries. The processor has a 25 stage pipeline.

The applications you care for have a small code size and work on small data sets in the range of 64 KB. These applications spend most of their time in loops whose iterations are independent of each other, but typically have only a limited amount of ILP within a single iteration (within the current processor implementation).

You can use the extra transistors in (possibly several of) the following ways:

1. Improve the branch predictor accuracy.
2. Add more reservation stations to your Tomasulo's Algorithm-based Dynamic Scheduler.
3. Add more FPUs and Integer Units.
4. Add more Reorder Buffer entries.

Some of these may be desirable additions, while others may not be too beneficial given the current configuration. There is a meeting coming up to discuss the proposed additions. Which of the above four additions should you support and which ones should you oppose (you can support/oppose multiple of these)? You need to justify your choices to receive credit.