CS 433 Midterm Exam – Oct 21, 2020

Professor Sarita Adve

Name	
NetID	
Grad/Undergrad	

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. You are allowed two sheets of scratch paper of 8.5”x11” (or less) each. They should initially be blank on both sides and be visible on zoom. No books, notes, or any other typed or written materials or 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 where possible 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 ***5 problems*** and **12 *pages*** (including this one). **For problem 2, only undergraduate students should solve part B and only graduate students should solve part C. Only graduate students should solve problem 5. All students should solve all the remaining problems.** Please budget your time appropriately. Good luck!

Problem	Maximum Points	Points Received
1	4	
2	12 for undergraduates 16 for graduates	
3	14	
4	26	
5	4 for graduates	
Total	56 for undergraduates 64 for graduates	

Problem 1 [4 points]

Consider the following instruction mix for a hypothetical benchmark.

Instruction type	Percentage of all instructions
Load from a memory address	15.5%
Store to a memory address	7.3%
Multiply	12.2%
Other	65%

You are considering making one of two possible enhancements to the machine. Enhancement E1 can make multiply operations run six times faster than before, while enhancement E2 can make memory accesses run two times faster than before. Determine the speedup from each enhancement. You can express your answer in terms of an equation with all the variables explicitly substituted. You are not required to perform numerical calculations.

Solution:

The enhancements mentioned only affect *multiply* and *memory access* instructions.

Recall Amdahl's law,
$\text{Speedup}_{\text{overall}} = 1 / ((1 - f) + f / S)$

First for our hypothetical benchmark:
$f_{\text{mul}}(E1) = 12.2\%$
$f_{\text{mem}}(E2) = 15.5\% + 7.3\% = 22.8\%$

$\text{Speedup}_{\text{mul}}(E1) = 1 / ((1 - f_{\text{mul}}(E1)) + f_{\text{mul}}(E1) / 6)$
$\text{Speedup}_{\text{mul}}(E1) = 1 / ((1 - 0.122) + 0.122 / 6)$
$\text{Speedup}_{\text{mul}}(E1) = 1.113$

$\text{Speedup}_{\text{mem}}(E2) = 1 / ((1 - f_{mem}(E2)) + f_{mem}(E2) / 2)$
$\text{Speedup}_{\text{mem}}(E2) = 1 / ((1 - 0.228) + 0.228 / 2)$
$\text{Speedup}_{\text{mem}}(E2) = 1.129$

So, E2 will give the best performance improvement for this benchmark.

Grading:

2 points for the Amdahl's law equation. 1 point for plugging in the correct values for enhancement E1, and 1 point for plugging in the correct values for enhancement E2.

ALL STUDENTS SHOULD SOLVE PART A OF PROBLEM 2. ONLY UNDERGRADUATES SHOULD SOLVE PART B AND ONLY GRADUATES SHOULD SOLVE PART C.

Problem 2 [12 points for undergraduates, 16 points for graduates]

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:

- Assume only the following functional units:

Functional Unit	Cycles in EX
Integer Div	7
Integer Mul	4
Other Integer	1

- There is one functional unit of each type.
- The processor can issue and commit at most one instruction per cycle.
- The CDB supports only one broadcast per cycle.
- Assume you have an unlimited number of reservation stations and reorder buffer entries.
- An instruction waiting for data on the CDB can move to EX in the cycle after the CDB broadcast.
- An instruction waiting to write to the CDB holds its execution unit until it gets the CDB; i.e., it prevents other instructions needing the same functional unit from beginning execution.
- If two instructions are ready to enter an execution unit in the same cycle, assume the older instruction (by program order) gets the execution unit.
- Assume that integer instructions also follow Tomasulo's algorithm (analogous to the floating point instructions discussed in class). That is, they can be issued out of order and the result from an integer functional unit is broadcast on the CDB and forwarded to dependent instructions through the CDB.

Part A [10 points] Fill in the missing entries in the chart below. Each entry should clearly show the cycles that the corresponding instruction spends in the corresponding pipeline stage. If there are any stalls, please indicate the reason in the last column. The issue stage and entries for the first instruction are filled in for you.

Instruction	IS	EX	WB	Commit	Reason for stalls
MUL R2, R1, R1	1	2-5	6	7	No stalls
DADDI R4, R3, #7	2	3	4	8	In-order commit
MUL R3, R2, R2	3	10-13	14	15	Wait for R2, structural hazard
ANDI R3, R4, #29	4	5	7	16	Wait for CDB after #1, in-order commit
MUL R1, R4, R1	5	6-9	10	17	In-order commit

Grading: Grading: 1/2 point for each entry/reason for stall. 1 additional point for fully correct solution. Cascading errors will not be penalized additionally as long as the relevant dependences are still observed.

Part B [2 points]

ONLY UNDERGRADUATE STUDENTS SHOULD SOLVE THIS PART

Consider a new block of instructions that the processor executes, issuing the first instruction on cycle 1. With the best choice of instructions, how early can the third instruction broadcast its result on the CDB? Use the following chart to show a sequence of instructions and the key stages/stalls to justify your answer.

Instruction	IS	EX	WB	Commit	Reason for stalls
ADD R1,R1,R1	1	2	3	4	No stall
ADD R2,R2,R2	2	3	4	5	No stall
ADD R3,R3,R3	3	4	***5***	6	No stall

Solution: Cycle 5. By making the third instruction a simple integer instruction that doesn't depend on earlier results, it will not stall.

Grading: Full credit for a correct sequence arriving at 5 cycles. No partial credit.

Part C [6 points]

ONLY GRADUATE STUDENTS SHOULD SOLVE THIS PART

Now find a sequence of instructions such that the third instruction broadcasts its result on the CDB as late as possible. Use the following chart to show the sequence and the key stages/stalls to justify your answer. You may use only the arithmetic instructions described above – do not use branches or memory instructions. Indicate clearly the cycle number where the third instruction broadcasts to the CDB.

Instruction	IS	EX	WB	Commit	Reason for stalls
DIV R3, R1, R2	1	2-8	9	10	No stall
DIV R5, R3, R4	2	16-22	23	24	RAW dep on R3 Structural hazard on #4
DIV R7, R5, R6	3	30-36	***37***		RAW dep on R5 Structural hazard on #5
DIV R8, R9, R10	4	9-15	16		Structural hazard on #1
DIV R12, R11, R11	5	23-29	30		Structural hazard on #1,#4, and #2

Solution: Cycle 37. Use divide instructions for the long execute time, have dependences between the first three instructions so each has to wait for the previous. Use out-of-order execution and the 1-cycle delay while a dependent instruction receives the broadcast to add structural hazards from instructions issued later in the program.

Grading: 1 point for recognizing that divides need to be involved. 2 points for delaying #3 due to a chain of two RAW dependences. 1.5 points for introducing each of the two later divides for delay due to structural hazards. Points are only allocated if proper explanations are given.

Problem 3 [14 points]

Consider a piece of code with three static branch instructions, B1, B2, and B3. During an execution of this code, the global history for these branches (i.e., the exact sequence in which the three branches were executed and their direction) is as follows:

T stands for the taken direction and N stands for not-taken

Branch instruction	B1	B2	B1	B2	B3	B1	B1	B2	B1
Direction	T	N	N	T	T	T	T	N	T

Thus, the execution starts with branch B1 being taken, then B2 is not taken, then B1 is not taken, etc. Assume that before the above execution, the outcome of all previous branches and the initial state of all predictors is not taken (N).

Assume that a correlating predictor of the form (2,1) is used for branch B1. Assume the state of the predictor is recorded in the form W/X/Y/Z where,

- W = state for the case where the last branch and the branch before the last are both TAKEN
- X = state for the case where the last branch is TAKEN and the branch before the last is NOT TAKEN
- Y = state for the case where the last branch is NOT TAKEN and the branch before the last is TAKEN
- Z = state for the case where the last branch and the branch before the last are both NOT TAKEN

Fill the blank entries in the two tables below assuming the (2,1) correlating predictor for branch B1 uses **local** branch history in Table 1 and **global** branch history in Table 2. (Recall that by local history for B1, we mean the history for only branch B1.)

Table 1: Assume the predictor uses LOCAL branch history				
Branch B1 invocation #	**History used for prediction**	**Prediction for B1**	**Actual direction of B1**	**New state of predictor**
1	N N	N	T	N/N/N/T
2	N T	N	N	N/N/N/T
3	T N	N	T	N/N/T/T
4	N T	N	T	N/T/T/T
5	T T	N	T	T/T/T/T

Table 2: Assume the predictor uses GLOBAL branch history				
Branch B1 invocation #	**History used for prediction**	**Prediction for B1**	**Actual direction of B1**	**New state of the predictor**
1	N N	N	T	N/N/N/T
2	T N	N	N	N/N/N/T
3	T T	N	T	T/N/N/T
4	T T	T	T	T/N/N/T
5	T N	N	T	T/N/T/T

Grading: 0.5 points for each correct entry. Cascading errors are not penalized.

Problem 4 [26 points]

In this problem, we will use a single-issue, in-order five stage pipeline similar to those studied in class, but with the following specification:

Functional Unit Type	Cycles in EX	Number of Functional Units	Pipelined
Integer	1	1	Yes
FP Add/Subtract	3	1	Yes
FP/Integer Multiplier	8	1	Yes
FP/Integer Divider	24	1	No

- The integer functional unit performs integer addition (including effective address calculation for loads/stores), subtraction, and logic operations.
- There is full forwarding and bypassing, including forwarding from the end of an FU to the MEM stage for stores.
- Loads and stores complete in one cycle. That is, they spend one cycle in the MEM stage after the effective address calculation.
- There are as many registers, both FP and integer, as you need.
- Branches are resolved in ID and there is one branch delay slot.
- While the hardware has full forwarding and bypassing, it is the responsibility of the compiler to schedule such that the operands of each instruction are available when needed by each instruction
- If multiple instructions finish their EX stages in the same cycle, then we will assume they can all proceed to the MEM stage together. Similarly, if multiple instructions finish their MEM stages in the same cycle, then we will assume they can all proceed to the WB stage together. In other words, for the purpose of this problem, you are to ignore structural hazards on the MEM and WB stages.
- This problem explores the ability of the compiler to schedule code as efficiently as possible for such a pipeline. We will consider the following code.

```
Loop:   L.D      F4, 0 (R1)
        MUL.D    F8, F4, F0
        L.D      F6, 0 (R2)
        ADD.D    F10, F6, F2
        ADD.D    F12, F8, F10
        S.D      F12, 0 (R3)
        DADDUI   R1, R1, #8
        DADDUI   R2, R2, #8
        DADDUI   R3, R3, #8
        DSUB     R5, R4, R1
        BNEZ     R5, Loop
```

Part A [6 points]

Rewrite the above loop, but let every row take a cycle (each row can be an instruction or a stall). If an instruction can't be issued on a given cycle (because the current instruction has a dependency that will not be resolved in time), write STALL instead, and move on to the next cycle (row) to see if it can be issued then. Assume that a NOP is scheduled in the branch delay slot (effectively stalling 1 cycle after the branch). *Explain the cause of all stalls*, but don't reorder instructions. How many cycles elapse before the second iteration begins?

Solution:

```
Loop:       L.D F4, 0(R1)
            stall RAW F4
            MUL.D F8, F4, F0
            L.D F6, 0(R2)
            stall RAW F6
            ADD.D F10, F6, F2
            stall RAW F8, F10
            stall RAW F8, F10
            stall RAW F8
            stall RAW F8
            ADD.D F12, F8, F10
            stall RAW F12
            S.D F12, 0(R3)
            DADDUI R1, R1, #8
            DADDUI R2, R2, #8
            DADDUI R3, R3, #8
            DSUB R5, R4, R1
            stall RAW R5 since branch resolved in ID
            BNEZ R5, Loop
            NOP – stall for branch delay
```

20 cycles elapse before the next iteration begins.

Grading: 1 point for each sequence of stalls. ½ point partial credit for indicating that there is a stall between a pair of instructions, but with an incorrect number of cycles. Negative ½ point for each unnecessary sequence of stalls.

Part B [6 points]

Now reschedule the loop to compute the same results as quickly as possible. You can change immediate values and memory offsets and reorder instructions, but don't change anything else. Show any stalls that remain. How many cycles elapse before the second iteration begins? Show your work.

Solution:

```
Loop:   L.D F4, 0(R1)
        L.D F6, 0(R2)
        MUL.D F8, F4, F0
        ADD.D F10, F6, F2
        DADDUI R1, R1, #8
        DADDUI R2, R2, #8
        DADDUI R3, R3, #8
        DSUB R5, R4, R1
        stall RAW F8
        stall RAW F8
        ADD.D F12, F8, F10
        BNEZ R5, Loop
        S.D F12, -8(R3)
```

13 cycles elapse before the second iteration begins

Grading: Full points for any correct sequence with minimum number of stalls. Partial credit only if the sequence does the same computation and reduces some stalls. Deduct ½ point for each error (e.g., incorrect index), and deduct ½ point for each stall in excess of 2.

Part C [6 points]

Now unroll the loop the minimum number of times needed to eliminate all stalls (with rescheduling). Show the unrolled and rescheduled loop. You can, and should, remove redundant instructions. How many original iterations of the loop are in an iteration of your new unrolled loop? How many cycles elapse before the next iteration of the unrolled loop begins? Don't worry about start-up or clean-up code outside the unrolled loop. Assume a very large number of iterations for the original loop. Show your work.

Solution: Note that in the solutions below, the registers used could be different and there is some flexibility in scheduling the instructions.

Loop:

```
L.D F4, 0(R1)
L.D F6, 0(R2)
MUL.D F8, F4, F0
L.D F14, 8(R1)
L.D F16, 8(R2)
MUL.D F18, F14, F0
ADD.D F10, F6, F2
ADD.D F20, F16, F2
DADDUI R1, R1, #16
DADDUI R2, R2, #16
DADDUI R3, R3, #16
ADD.D F12, F8, F10
DSUB R5, R4, R1
ADD.D F22, F18, F20
S.D F12, -16(R3)
BNEZ R5, Loop
S.D F22, -8(R3)
```

There are two original iterations in an iteration of the new loop. 17 cycles elapse before the next iteration of the new loop begins.

Grading: 1 point for the correct iteration count. Deduct ½ point for every error or stall cycle. Give partial credit (3 points) if three iterations are used instead of two and the solution is correct with three iterations.

Part D [8 points]

Consider a VLIW processor in which one instruction can support two memory operations (load or store), one integer operation (addition, subtraction, comparison, or branch), one floating point add or subtract, and one floating point multiply or divide. There is no branch delay slot. Now unroll the original loop four times (i.e., four original iterations in one new iteration), and schedule it for this VLIW to take as few stall cycles as possible. How many cycles do the four iterations take to complete? Use the following table template to show your work.

Solution:

```
Unrolled loop:  L.D        F4, 0 (R1)
                MUL.D      F8, F4, F0
                L.D        F6, 0 (R2)
                ADD.D      F10, F6, F2
                ADD.D      F12, F8, F10
                S.D        F12, 0 (R3)

                L.D        F14, 8 (R1)
                MUL.D      F18, F14, F0
                L.D        F16, 8 (R2)
                ADD.D      F20, F16, F2
                ADD.D      F22, F18, F20
                S.D        F22, 8 (R3)

                L.D        F24, 16 (R1)
                MUL.D      F28, F24, F0
                L.D        F26, 16 (R2)
                ADD.D      F30, F26, F2
                ADD.D      F32, F28, F30
                S.D        F32, 16 (R3)

                L.D        F34, 24 (R1)
                MUL.D      F38, F34, F0
                L.D        F36, 24 (R2)
                ADD.D      F40, F36, F2
                ADD.D      F42, F38, F40
                S.D        F42, 24 (R3)

                DADDUI     R1, R1, #32
                DADDUI     R2, R2, #32
                DADDUI     R3, R3, #32
                DSUB       R5, R4, R1
                BNEZ       R5, Loop
```

MEM 1	MEM 2	INTEGER	FP ADD	FP MUL
L.D F4, 0(R1)	L.D F6, 0(R2)			
L.D F14, 8(R1)	L.D F16, 8(R2)			
L.D F24, 16(R1)	L.D F26, 16(R2)		ADD.D F10, F6, F2	MUL.D F8, F4, F0
L.D F34, 24(R1)	L.D F36, 24(R2)		ADD.D F20, F16, F2	MUL.D F18, F14, F0
			ADD.D F30, F26, F2	MUL.D F28, F24, F0
			ADD.D F40, F36, F2	MUL.D F38, F34, F0
		DADDUI R1, R1, #32		
		DADDUI R2, R2, #32		
		DADDUI R3, R3, #32		
		DSUB R5, R4, R1		
			ADD.D F12, F8, F10	
			ADD.D F22, F18, F20	
S.D F12, -32(R3)			ADD.D F32, F28, F30	
S.D F22, -24(R3)			ADD.D F42, F38, F40	
S.D. F32, -16(R3)				
S.D F42, -8(R3)		BENZ R5, Loop		

Four iterations will take 16 cycles.

Grading:

- 1 point for scheduling all the loads correctly
- 1 point for scheduling all the ADDs that add from the load values correctly
- 1 point for scheduling all the MULT that multiply from the load values correctly
- 1 point for scheduling all the ADDs that add from the ADD and MULT correctly
- 1 point for scheduling all the DADDUIs correctly
- 0.5 point for scheduling the DSUB correctly
- 1 point for scheduling all the stores correctly
- 0.5 point for scheduling the branch correctly
- 1 point for correctly unrolling the loop with proper register renaming and offsets

Do not take off points for “DADDIU R1, R1, #32” or “DADDIU R1, R1, #32” as long as they are scheduled at cycle 5 or later. Do not take off points for “DADDIU R3, R3, #32” as long as it is scheduled on or before cycle 12. Do not take off points for “DSUB R5, R4, R1” as long as it is scheduled after “DADDUI R1, R1, #32” and there is one cycle between the DSUB and the branch (since the branch needs that value in ID stage).

ONLY GRADUATE STUDENTS SHOULD SOLVE PROBLEM 5.

Problem 5 [4 points]

Consider the following format for predicated MIPS instructions:

```
(pT) ADD R1, R2, R3
```

where the ADD instruction is predicated on the predicate register pT. Assume a set of 1-bit predicate registers.

Assume compare instructions that set a pair of predicate registers to complementary values:

```
CMP.NE pT, pF = R8, R0
```

The above compare sets the 1-bit predicate registers, pT and pF, based on the "not equal" (NE) comparison relation as follows:

```
pT = (R8 != R0)
pF = !(R8!= R0)
```

So pT is true if R8 is not equal to R0, and pF is the complement of pT.

For the following problem, you can assume the availability of any comparison relation with two operands; e.g., .LE for less than or equal to and .GT for greater than.

Using the predicated instructions described above, write the three basic blocks of the following code fragment as a single basic block; i.e., eliminate all branches using predicated instructions.

```
        SUB     R1, R13, R14
        BLT     R1, R4, L1      //branch if R1 < R4
        ADDI    R2, R2, #1
        SW      R2, 0(R7)
        J       L2
L1:     DIV.D   F0, F0, F2
        ADD.D   F0, F4, F2
        S.D     F0, 0(R8)
L2:      ...
```

Solution:

The code fragment with predicated instructions is as below

```
        (1) SUB R1, R13, R14
        (2) CMP.LT pT, pF = R1, R4
        (3) (pF) ADDI R2, R2, #1
        (4) (pF) SW 0(R7), R2
        (5) (pT) DIV.D F0, F0, F2
        (6) (pT) ADD.D F0, F4, F2
        (7) (pT) S.D 0(R8), F0
L2:...
```

Grading:

1/2 point for correctly translating each of the 8 instructions in the original code. (Note that for the jump instruction, J, the correct translation is to not have any instruction.