CprE 381 Homework 8

[Note: This assignment is targeted to help solidify your knowledge of exam 2 topics as well as pipeline control hazard. Such understanding is useful for both part 2 of your term project and to understand transient execution attacks such as Spectre.]

1. Exam 2 Rework
   Rework and submit your worst graded problem (i.e., 2, 3, ..., 12) from Exam 2. I leave it up to you to select an appropriate problem where you had a genuine misconception. You must rework the entire problem and submit it as a legible, one- or two-page pdf. **Include a description of your misconception.** This problem's grade will be the percent score from your best revised problem. Since you have time and the ability to access resources, partial credit will be less available. In particular, if I made comments on your question and took off few to no points, there may still be mild errors in your solution! Please ask questions in OHs, before class, and on the Homework Help channel to make sure you understand the problem.

   Solutions to exams not provided and many unique misconceptions possible.

2. Control Hazards
   The following problem was sourced from the fifth edition of our textbook, problem 4.14.

   This exercise is intended to help you understand the relationship between delay slots, control hazards, and branch execution in a pipelined processor. In this exercise, we assume that the following MIPS code is executed on a pipelined processor with a 5-stage pipeline, full forwarding, and a predict-taken branch predictor:

```
        lw $2,0($1)
label1: beq $2,$0,label2 # not taken once, then taken
        lw $3,0($2)
        beq $3,$0,label1 # taken
        add $1,$3,$1
label2: sw $1,0($2)
```

   a. 4.14.1: Draw the pipeline execution diagram for this code, assuming there are no delay slots and that branches execute in the EX stage.

Instruction	Cycle													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lw r2, 0(r1)	IF	ID	EX	MEM	WB									
beq r2, r0 label 2 (NT)		IF	ID	**	EX	MEM	WB							
lw r3, 0(r2) (squashed)			IF	X	X	X	X	X						
sw (squashed)				IF	ID	X	X	X	X					
Inst following sw (squashed)					IF	X	X	X	X	X				

lw r3, 0(r2)						IF	ID	EX	MEM	WB				
beq r3, r0, label1 (T)							IF	ID	**	EX	MEM	WB		
beq r2, r0, label 2 (T)								IF	**	ID	EX	MEM	WB	
sw r1, 0(r2)										IF	ID	EX	MEM	WB

For the first beq, a stall is needed following the ID stage to allow the values to be forwarded from the output of the memory read from the preceding lw. Additionally, due to the branch not being taken, the sw needs to be squashed. The only other stalls needed are in the two consecutive beq's to allow the loaded value in the second lw to be forwarded.

b. 4.14.2: Repeat 4.14.1, but assume that delay slots are used. In the given code, the instruction that follows the branch is now the delay slot instruction for that branch.

Instruction	Cycle													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lw r2, 0(r1)	IF	ID	EX	MEM	WB									
beq r2, r0 label 2 (NT)		IF	ID	**	EX	MEM	WB							
lw r3, 0(r2)			IF	**	ID	EX	MEM	WB						
sw (squashed)					IF	X	X	X	X					
beq r3, r0, label1 (T)						IF	ID	EX	MEM	WB				
add r1, r3, r1							IF	ID	EX	MEM	WB			
beq r2, r0, label 2 (T)								IF	ID	EX	MEM	WB		
lw r3, 0(r2)									IF	ID	EX	MEM	WB	
sw r1, 0(r2)										IF	ID	EX	MEM	WB

Here, the same stall is needed for the initial beq; however, due to the delay slot being the consecutive instruction the lw is also in the pipeline. Then, after the delay slot instruction (the lw), the sw is fetched, due to the predict taken branch predictor. Once the initial beq enters the EX stage, the correct instruction can be loaded (the second beq) and the sw can be squashed. With the lw instruction already in the pipeline, the second beq does not need to be stalled, the values can be forwarded from the WB stage.

c. Repeat 4.14.1 but assume no prediction and no hardware detection or prediction of control hazards (data hazards are still detected and forwarded). Insert the appropriate NOPs and draw the corresponding pipeline diagram.

```
        lw $2,0($1)
label1: beq $2,$0,label2
        nop
        nop
        lw $3,0($2)
        beq $3,$0,label1
        nop
        nop
        add $1,$3,$1
label2: sw $1,0($2)
```

Instruction	Cycle																	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
lw r2, 0(r1)	IF	ID	EX	MEM	WB													
beq r2, r0, label2 (NT)		IF	ID	**	EX	MEM	WB											
NOP				IF	ID	EX	MEM	WB										
NOP					IF	ID	EX	MEM	WB									
lw r3, 0(r2)						IF	ID	EX	MEM	WB								
beq r3, r0, label1 (T)							IF	ID	**	EX	MEM	WB						
NOP									IF	ID	EX	MEM	WB					
NOP										IF	ID	EX	MEM	WB				
beq r2, r0, label2 (T)											IF	ID	EX	MEM	WB			
NOP												IF	ID	EX	MEM	WB		
NOP													IF	ID	EX	MEM	WB	
sw r1, 0(r2)														IF	ID	EX	MEM	WB

Since branches are executed in the EX stage, the 2 instructions following a branch instruction will enter the pipeline in the IF and ID stages before the branch decision is completed. Therefore, 2 NOPs are needed after each branch to prevent the incorrect instructions from executing in case the branch is taken.

Section (COD 4.9) describes how the severity of control hazards can be reduced by moving branch execution into the ID stage. This approach involves a dedicated comparator in the ID stage, as shown in COD Figure 4.62. However, this approach potentially adds to the latency of the ID stage, and requires additional forwarding logic and hazard detection.

d. 4.14.4: Using the first branch instruction in the given code as an example, describe the hazard detection logic needed to support branch execution in the ID stage as in COD Figure 4.62. Which type of hazard is this new logic supposed to detect?

The hazard detection logic must detect situations when the branch depends on the result of the previous R-type instruction, or on the result of two previous loads. When the branch uses the values of its register operands in its ID stage, the R-type instruction's result is still being generated in the EX stage. Thus, we must stall the processor and repeat the ID stage of the branch in the next cycle. Similarly, if the branch depends on a load that immediately precedes it, the result of the load is only generated two cycles after the branch enters the ID stage, so we must stall the branch for two cycles. Finally, if the branch depends on a load that is the second-previous instruction, the load is completing its MEM stage when the branch is in its ID stage, so we must stall the branch for one cycle. In all three cases, the hazard is a data hazard.

Note that in all three cases we assume that the values of preceding instructions are forwarded to the ID stage of the branch if possible.

e. 4.14.5: For the given code, what is the speedup achieved by moving branch execution in the ID stage? Explain your answer. In your speedup calculation, assume that the additional comparison in the ID stage does not affect clock cycle time.

Instruction	Cycle														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw r2, 0(r1)	IF	ID	EX	MEM	WB										
beq r2, r0 label 2 (NT)		IF	**	**	ID	EX	MEM	WB							
sw r1, 0(r2) (squashed)					IF	X	X	X	X						
lw r3, 0(r2)						IF	ID	EX	MEM	WB					
beq r3, r0, label1 (T)							IF	**	**	ID	EX	MEM	WB		
beq r2, r0, label 2 (T)										IF	ID	EX	MEM	WB	
sw r1, 0(r2)											IF	ID	EX	MEM	WB

Speedup computed as follows:
14/15 = .93

f. 4.14.6: Using the first branch instruction in the given code as an example, describe the forwarding support that must be added to support branch execution in the ID stage. Compare the complexity of this new forwarding unit to the complexity of the existing forwarding unit in COD Figure 4.62.

Branch instructions are now executed in the ID stage. If the branch instruction is using a register value produced by the immediately preceding instruction, as we described for part d the branch must be stalled because the preceding instruction is in the EX stage when the branch is already using the stale register values in the ID stage. If the branch in the ID stage, it depends on an R-type instruction that is in the MEM stage, we need forwarding to ensure correct execution of the branch. Similarly, if the branch in the ID stage depends on an R-type of load instruction in the WB stage, we need forwarding to ensure correct execution of the branch. Overall, we need another forwarding unit that takes the same inputs as the one that forwards to the EX stage. The new forwarding unit should control two Muxes placed right before the branch comparator. Each Mux selects between the value read from Registers, the ALU output from the EX/ MEM pipeline register, and the data value from the MEM/WB pipeline register. The complexity of the new forwarding unit is the same as the complexity of the existing one.

3. Computer Architecture in the Media (***Extra Credit*** to be applied globally to the course)

   Find an online article from the past two years that has some relation to computer architecture. Make sure you can relate it to at least three of the following terms: CPI, frequency/clock cycle, speculative execution, pipeline/pipelining, execution time, instruction, and power. Summarize the article in one paragraph (4-6 complete sentences) and, in a second paragraph, describe how the article relates to the above terms and what you have been learning in class.