DEPARTMENT OF COMPUTER SCIENCE
COLORADO SCHOOL OF MINES
EARTH • ENERGY • ENVIRONMENT

CSCI 341: Computer Organization
WS 12: Data Hazards and Control Hazards

1	Name one difference between forwarding and hazard detection. Solution: Forwarding does not stall the pipeline whereas hazard detection does.

2	What are the conditionals for an EX and MEM hazard that the forwarding unit has to implement?
	
	Solution:
	EX hazard:
	`if (EX/MEM.RegWrite and`
	`(EX/MEM.RegisterRd ≠ 0) and`
	`(EX/MEM.RegisterRd = ID/EX.RegisterRs1)) ForwardA = 10`
	`if (EX/MEM.RegWrite and`
	`(EX/MEM.RegisterRd ≠ 0) and`
	`(EX/MEM.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 10`
	
	MEM hazard:
	`if (MEM/WB.RegWrite and`
	`(MEM/WB.RegisterRd ≠ 0) and`
	`(MEM/WB.RegisterRd = ID/EX.RegisterRs1)) ForwardA = 01`
	`if (MEM/WB.RegWrite and`
	`(MEM/WB.RegisterRd ≠ 0) 578 and`
	`(MEM/WB.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 01`

3	What is a control hazard? Why do they arise? Why does the single cycle processor not have control hazards? Solution: A control hazard occurs with branch instructions, where the pipelined processor will start running instructions that it should not run because the outcome of the branch is not known until late in the pipeline. A single cycle processor does not have control hazards because only one instruction is running at a time.
4	How many instructions are run before a branch decision is made with the original pipeline architecture? Why is this important?

Solution:
3 instructions. This is important because this creates a large delay in processing, especially considering how often branches occur in code. The overhead may seem small with only one branch, but with many branches the effect will be noticeable.

5 List the three strategies used to reduce the amount of instructions run before a branch decision is made.

Solution:
- Delayed branch
- Static branch prediction
- Dynamic branch prediction

6 An assembler may change the following assembly to produce the optimized code shown below.

Original	**Optimized**

`addi t0, zero, zero`
`addi t1, zero, 10`
`forLoop:`
`  … code …`
`  addi t0, t0, 1`
`  lw t2, 0(s1)`
`  bne t0, t1, forLoop`
	`addi t0, zero, zero`
	`addi t1, zero, 10`
	`forLoop:`
	`  … code …`
	`  addi t0, t0, 1`
	`  bne t0, t1, forLoop`
	`lw t2, 0(s1)`

Why does this work for pipelined processors? Will this work for a single cycle processor? What is this an example of?

Solution:
The lw instruction was moved down because it has no effect on the execution of the for loop. By doing this there is no need to stall the pipeline to wait for the result of the bne instruction.
This is an example of a delayed branch.

7 Explain how dynamic branch prediction works. Why does it work?

Solution:
Dynamic branch prediction takes into account previous branch decisions to make a guess as to how the current branch will make its decision. There are one and two bit

	prediction systems that take into account one and two previous decisions respectively. The prediction bit(s) are updated after each branch decision. It works because most code has repeated decisions (for loops, while loops, if statements inside loops) whose behavior is mostly the same until some boundary condition is me (end of loop, found item in list). (or the answer to the problem worked out in class)
8	Will increasing the number of bits in a branch predictor eventually result in 100% accuracy? Solution: No.