In-Class Activity 13-2

1 Activity 1 - Mechanisms for Stall

Simulate the following sequence of instructions. In this case, we will focus on the stall logic in the ID phase. For each instruction, determine

- Is a stall required because of an LW hazard? (You can assume that data forwarding is implemented, so you don’t need to worry about the more “ordinary” EX-to-EX forwarding cases.)
- If a LW stall is required, what pipeline registers could you read to find that out? What is the key information?
- Do we have a SW-data-register hazard?
- If we have a SW-data-register hazard, can we solve it with forwarding? If so, what registers would you need to read in the MEM phase to build this?

```
lw  $s0, 0($t0)  
addi $s1, $s0,1  
addi $s2, $s0,2  
sw  $s0, 4($t0)  

lw  $s3, 8($t0)  
addi $s4, $s3,100  

lw  $s5, 12($t0)  
nop  
addi $s5, 16($t0)  
```
2 Activity 2 - Branch Hazards, Part 1

Assume that the branch design is as we’ve shown you so far (we’re about to change it): we use the ALU to compare the two registers. Thus, we can’t know whether to take a branch or not, until the EX phase is complete.

Question 1
Simulate two instructions - BEQ, followed by anything else. How many stalls do we need to insert so that we don’t fetch until we know the correct address to fetch from?

Question 2
Then, consider a JUMP instruction: simulate a J, followed by anything else. We don’t need the ALU - but we still need to realize that a jump is occurring. How many stalls do we need to insert in a J instruction?

Question 3
Now, consider JR. What do we need to do? How many stalls are required?

Question 4
We’ve overlooked a detail. If we decide to change the PC (due to a taken BEQ, or because of J or JR), we already have instructions in the pipeline. Brainstorm ways to handle those instructions.