ECE552 – Lab 1 Prelab
Bing Li – 1004191910

1. A dependence describes a relationship where one instruction might need the result of another (whether it's a branch or just using the same registers), while a hazard is a potential problem in a pipeline that may result from a dependence. For example, given instructions:

   ```
   ADD r2,r3 -> r1
   SUB r1,r2 -> r6
   ```

   The second instruction is dependent on the first, and in the event that it's run in the 5 stage pipeline model (F,D,X,M,W), the r1 in the SUB instruction will not have the correct value as it may be read before the previous ADD instruction loaded the result into the register, resulting in a data hazard

2. A data hazard is when an instruction depends on the result of another instruction still in the pipeline (like the example from 1.), and a structural hazard is when two instructions attempt to use the same hardware resource in the same cycle

3. In a simple 5 stage pipeline (assuming in-order), WAW hazards are not possible. This is because a WAW hazard occurs when 2 instructions write to the same register, and due to some scheduling problem, these 2 instructions are executed out of order, resulting in the wrong value being left in the register. However, since a simple, in-order pipeline executes the instructions in the same order as written, a WAW hazard should never happen

4. Since the Decode stage reads the values from the register to send to the later stages in the pipeline, it must be able to read the correct value. By assuming that the Write stage writes in the first half of a cycle while the Decode stage reads in the second half, we can safely assume that whatever is meant to be written to the register will be read correctly in the same cycle. This in turn will decrease the number of stalls required by 1 cycle.

   For q1, assuming no forwarding, ADD will be complete and r1 will be written in cycle 5, and the SUB can then decode the instruction and read the values from r1 in cycle 5 with no worries, resulting in a stall of 2 cycles

Instruction	1	2	3	4	5	6	7	8	9
`ADD r2,r3 -> r1`	F	D	X	M	W				
`SUB r1,r2 -> r6`		F	d*	d*	D	X	M	W	

   However, if the assumption is not the case and it is unsure if Decode will read the correct value when in the same cycle as Write, we will need to add one more cycle of stall to be sure that there isn't a hazard, as D will need to wait until it's sure W completes (i.e. wait until the next cycle) before reading

Instruction	1	2	3	4	5	6	7	8	9
`ADD r2,r3 -> r1`	F	D	X	M	W				
`SUB r1,r2 -> r6`		F	d*	d*	**d***	D	X	M	W

This does not apply to q2, which has forwarding so it doesn't have to wait until the Write stage to use the value

5. For q1, where it is using a simple 5 stage pipeline, it has the possibility of an instruction needing 1 cycle of stalling and an instruction needing 2 cycles of stalling, depending on whether the dependent instruction comes right after the first instruction or 2 instructions after:

2 cycle stall scenario:

Instruction	1	2	3	4	5	6	7	8	9
`ADD r2,r3 -> r1`	F	D	X	M	W				
`SUB r1,r2 -> r6`		F	d*	d*	D	X	M	W	

1 cycle stall scenario:

Instruction	1	2	3	4	5	6	7	8	9
`ADD r2,r3 -> r1`	F	D	X	M	W				
`ADD r4,r3 -> r5`		F	D	X	M	W			
`SUB r1,r2 -> r6`			F	d*	D	X	M	W	

For q2, since it has full bypassing, we can use a MX bypass to pass the value from M to EX1, result in only one cycle of stall (for EX2 of the first instruction):

Instruction	1	2	3	4	5	6	7	8	9
`ADD r2,r3 -> r1`	F	D	EX1	EX2	M	W			
`SUB r1,r2 -> r6`		F	d*	D	EX1	EX2	M	W	

We can count the number of these different types of stalls and use that to calculate the penalties added to execute the program, and thus generating a new CPI by adding all the stalling penalties

Thus, the general algorithm is:

- For each instruction
    - note down the Input and Output registers
    - Check if any of the input registers are not ready, if not ready:
        - q1: If the number of instructions between current instruction and "ready" state is 2, increment counter for 2 cycle stall, otherwise increment counter for 1 cycle stall
        - q2: Increment counter for 1 cycle stall
    - For each output register, mark it as "ready" in 3 instructions / cycles for q1, and 2 cycles for q2
- Compute CPI
    - q1: CPI = 1 + (single stall frequency) * 1 + (double stall frequency) * 2
    - q2: CPI = 1 + (single stall frequency) * 1