**Karan Vora (kv2154)**
**Computer Systems Architecture Assignment 6**

**Problem 1):**

The number of NOPs required to optimize the n-instruction without forwarding to handle data hazards is = 0.4 * n

**Solution 1.1):**

The cycle time of pipeline without forwarding is 250 ps

The number of NOPs required to optimize the n-instruction with forwarding to handle data hazards is 0.05 * n

The cycle time of pipeline with forwarding is 300 ps

Speed of the pipeline for n instructions without forwarding
= 250 * ( n + 0.4 * n )
= 250 * 1.4 * n
= 350n ps

Speed of the pipeline for n instructions with forwarding
= 300 * ( n + 0.05 * n )
= 300 * 1.05 * n
= 315n ps

The speedup of the new pipeline compared to the without forwarding is

$$= \frac{350-315}{350} * 100$$

$$= \frac{35}{350} * 100$$

= 10%

Therefore, the speedup of the new pipeline compared to the without forwarding by 10%

**Solution 1.2):**

Suppose that n-instruction program with xn NOPs and (1-x)n non-NOP instructions can run faster on the pipeline with forwarding ( x( 0.05 / 0.4 ) )n NOPs and without forwarding ( 1 - x )n non – NOPs

Number of NOPs can remain in the typical program before that programs runs slower on the pipeline with forwarding is
= 250 ps * n < 300 ps * [( 1 - x ) + ( 0.125x )]n
= 5 < 6( 1 – 0.875x )
= 5.25x < 1

= x < 0.1905
≈ 19.05% of Instructions

Therefore about 19.05% of code instructions

**Solution 1.3):**

The number of NOPs required to optimize the n-instruction with forwarding to handle data hazards is = 0.075 * n

Speed of the pipeline for n instructions with forwarding
= 300 * ( n + 0.75 * n )
= 300 * 1.75 * n
= 525n ps

The speedup of the new pipeline compared to the without forwarding is

$$= \frac{525-315}{525} * 100$$

$$= \frac{210}{525} * 100$$

= 40%

Therefore, the speedup of the pipeline compared to the without forwarding by 40%

**Solution 1.4):**

Speedup of the new pipeline compared to the without forwarding by 40%

The cycle time of pipeline with forwarding is 300 ps

Speed of the pipeline for n-instructions with forwarding = 300 * 40 / 100 = 120 NOPs

**Solution 1.5):**

The larger the number of NOPs per instruction, x without forwarding, the easier it is for forwarding hardware to work and lower the number of NOPs per instruction. oAs the number of NOPS per instruction without forwarding, x decreases, it becomes physically impossible to go faster with forwarding if there are very few data hazards. This minimum occurrence of data hazards is when x is at a minimum corresponding to the case of γ = 0.

==========

**Problem 2):**

**Solution 2.1):**

| Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sd x29, 12(x16) | IF | ID | EX | MEM | WB | | | | | | | | |
| ld x29, 8(x29) | | IF | ID | EX | MEM | WB | | | | | | | |
| sub x17, x15, x14 | | | IF | ID | EX | MEM | WB | | | | | | |
| beqz x17, label | | | | Stall | Stall | Stall | IF | ID | EX | MEM | WB | | |
| add x15, x11, x14 | | | | | | | | IF | ID | EX | MEM | WB | |
| Sub x15, x30, x14 | | | | | | | | | IF | ID | EX | MEM | WB |

**Solution 2.2):**

Structural hazards are due to resource sharing. This cannot be eliminated by reordering instructions as every instructions will go through all stages of pipelining. This can be solved either by increasing hardware resources or by stalling the pipeline.

**Solution 2.3):**

It has to be handled by hardware. NOPs can handle data hazards but not structural hazards because NOPs is also an instruction which needs to be fetched like other instructions => to eliminate structural hazard a hardware hazard detection unit has to be used.

**Solution 2.4):**

3 stall cycles.

==========

**Problem 3):**

**Solution 3.1):** The clock period won't change because we aren't making any changes to the slowest stage.

**Solution 3.2):** Moving the MEM stage in parallel with the EX stage will eliminate the need for a cycle between loads and operations that use the result of the loads. This can potentially reduce the number of stalls in a program.

**Solution 3.3):** Removing the off set from ld and sd may increase the total number of instructions because some ld and sd instructions will need to be replaced with a addi/ld or addi/sd pair.

==========

**Problem 4):**

Choice 2 describes pipeline hazard detection in a better way. Because the different cycles of the instruction executed are mentioned as next instruction was started before the first on get executed.

ld x11, 0(x12) : IF ID EX ME WB
add x13, x11, x14 : IF ID EX ME WB
or x15, x16, x17 : IF ID EX ME WB

==========

**Problem 5):**

**Solution 5.1):**

Since perfect branch prediction is used, we do not lose any cycles due to branch hazards, that is, the branch is always predicted and taken correctly. Availability of full forwarding support is assumed, so we can assume data hazards that can be resolved with forwarding do not stall the pipeline. Load-use-hazards cannot be resolved and are identified in next slide in red boxes - resolved by stalling the pipeline. pipeline stages unused by any instruction are identified in blue. Any clock cycle that does not have all of the pipeline stages utilized is identified with ‘N’.

**Solution 5.2):**

==========

**Problem 7):**

**Solution 7.5):**

The hazard detection unit additionally needs the values of rd that comes out of the MEM/WB register. The instruction that is currently in the ID stage needs to be stalled if it depends on a value produced by (or forwarded from) the instruction in the EX or the instruction in the MEM stage. So, we need to check the destination register of these two instructions. The Hazard unit already has the value of rd from the EX/MEM register as inputs, so we need only add the value from the MEM/WB register. No additional outputs are needed. We can stall the pipeline using the three output signals that we already have. The value of rd from EX/MEM is needed to detect the data hazard between the add and the following ld. The value of rd form MEM/WB is needed to detect the data hazard between the first ld instruction and the or instruction.

**Solution 7.6):**

| Clock Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| add | IF | ID | EX | MEM | WB | | | | |
| ld | | IF | ID | - | - | EX | MEM | WB | |
| ld | | | IF | - | - | ID | EX | MEM | WB |

(1) PCWrite = 1; IF/IDWrite = 1; control mux = 0
(2) PCWrite = 1; IF/IDWrite = 1; control mux = 0
(3) PCWrite = 1; IF/IDWrite = 1; control mux = 0
(4) PCWrite = 0; IF/IDWrite = 0; control mux = 1
(5) PCWrite = 0; IF/IDWrite = 0; control mux = 1