Problem 1. (20 points) The following instructions executed on a 5-stage pipelined datapath:

I. If there is no forwarding or hazard detection, insert NOPS to ensure correct execution. (Hint: for the first sw instruction, register r1 needs to be written first by "add" instruction and then sw instruction can read it.)

II. In this case, use NOPS only when changing or rearranging instructions cannot avoid a hazard. You can assume register R7 can be used to hold temporary values in your modified code.

```
add  r1, r5, r3
sw   r1, 0(r2)
lw   r1, 4(r2)
add  r5, r5, r1
sw   r1, 0(r2)
```

a. If there is no forwarding or hazard detection, insert NOPS to ensure correct execution.

| **Clock cycle** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *ADD r1, r5, r3* | IF | ID | EXE | MEM | WB | | | | | | | | |
| *SW r1, 0(r2)* | | nop | nop | IF | ID | EXE | MEM | WB | | | | | |
| *LW r1,4(r2)* | | | | | IF | ID | EXE | MEM | WB | | | | |
| *ADD r5, r5, r1* | | | | | | nop | nop | IF | ID | EXE | MEM | WB | |
| *SW r1, 0(r2)* | | | | | | | | | IF | ID | EXE | MEM | WB |

b. In this case, use NOPS only when changing or rearranging instructions cannot avoid a hazard.

- In this case, I used r7 to get a result of the sum between r5 and r3. So that, I can move up the LW instruction before the first SW instruction because there is no dependence when I use register r7.

| **Clock cycle** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
|---|---|---|---|---|---|---|---|---|---|---|
| *ADD **r7**, r5, r3* | IF | ID | EXE | MEM | WB | | | | | |
| *LW r1, 4(r2)* | | IF | ID | EXE | MEM | WB | | | | |
| *SW **r7**, 0(r2)* | | | nop | IF | ID | EXE | MEM | WB | | |
| *ADD r5, r5, r1* | | | | | IF | ID | EXE | MEM | WB | |
| *SW r1, 0(r2)* | | | | | | IF | ID | EXE | MEM | WB |

Problem 2. (20 points) Calculate how many clock cycles will take execution of the following segment one RISC-V simple pipeline without forwarding when the outcome of branch instruction is available after EXE stage. Draw the pipelined diagram for this question.

```
lw   r1, 0(r4)
lw   r2, 400(r4)
addi r3, r1, r2
sw   r3, 0(r4)
sub  r4, r4, 4
beq  r4, r5, L1
```

| Instruction | Clock cycle number | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** |
| *LW r1, 0(r4)* | IF | ID | EXE | MEM | WB | | | | | | | | | |
| *LW r2, 400(r4)* | | IF | ID | EXE | MEM | WB | | | | | | | | |
| *ADD r3, r1, r2* | | | * | * | IF | ID | EXE | MEM | WB | | | | | |
| *SW r3, 0(r4)* | | | | | | * | * | IF | ID | EXE | MEM | WB | | |
| *SUB r4, r4, 4* | | | | | | | | | IF | ID | EXE | MEM | WB | |
| *BEQ r4, r5, L1* | | | | | | | | | | IF | ID | EXE | MEM | WB |

- Number of clock cycles in one loop = 14 clock cycles.
- Number of clock cycles for segment execution on pipelined processor = 1 clock cycle (IF stage of the initial instruction) + (Number of clock cycles in one loop L1) * (Number of loop cycles) = 1 + 14 * 400/4 = 1401 clock cycles.

Problem 3. (20 points) The individual stages of datapath have the following latencies:

| IF | ID | EXE | MEM | WB |
|---|---|---|---|---|
| 250ps | 350ps | 150ps | 300ps | 200ps |

Also, assume that instructions executed by the processor are broken down as follows:

| ALU | Branch | Load | Store |
|---|---|---|---|
| 45% | 20% | 15% | 20% |

a) What is the clock cycle time in a pipelined and non-pipelined processor?

b) What is the total latency of **ld** instruction in a pipelined and non-pipelined processor?

c) Assuming there are no hazards or stalls, what is the utilization of data memory?

(Utilization is the fraction of clock cycles in which data memory is used).

d) Assuming there are no hazards or stalls, what is the utilization of register block's write port?

a. What is the clock cycle time in a pipelined and non-pipelined processor?

- Pipelined:
  - Pipelining to 5 stages reduces the cycle time to the length of the longest stage. Therefore, clock cycle is determinate by the slowest stage.
  - Instruction decode is the slowest processor => Clock cycle = 350ps
- Non-pipelined:
  - There is no pipelining, so the cycle-time has to allow an instruction to go through all the stages each cycle. Therefore, clock cycle is determinate by sum of all stages.
  - Clock cycle = 350 + 250 + 150 + 200 + 300 = 1250ps

b. What is the total latency of load instruction in a pipelined and non-pipelined processor?

- Pipelined:
  - Since a load instruction needs to go through 5 pipeline stages, spending one cycle in each, before it commits; therefore, the latency for load instruction is 5 * cycle time.
  - Latency = 5 * 350 = 1750ps
- Non-pipelined:
  - Latency for a load instruction is also the same since each instruction takes one cycle to go from beginning fetch to the end of writeback.
  - Latency = 1250ps

c. Assuming there are no hazards and stalls, what is the utilization of data memory?
Data memory is utilized only by Load and Store instructions. The utilization is 35% of the clock cycles.

d. Assuming there are no hazards and stalls, what is the utilization of register block's write port?
The write port may be utilized by ALU and Load instructions. The utilization is 60% of the clock cycles.

Problem 4. (20 points) Consider the following sequence of instructions being processed on the 5-stage RISC-V pipelined processor:

```
lw  r4, 100(r2)
add r5, r2, r3
sub r6, r4, r5
and r7, r2, r5
```

a) Identify all the data dependencies in the above instruction sequence. For each dependency, indicate the two instructions and the register that causes the dependency.

b) Assume that the pipelined uses full forwarding. Draw a pipelined diagram that represents the flow of instructions through the pipeline during each clock cycle. Indicate forwarding by arrows.

a. Identify all data dependencies in the above instruction sequence. For each dependency, indicate the two instructions and the register that causes the dependency.
There are three data dependencies in this instruction sequence:

- Subtract instruction: depends on Load instruction for register r4
- Subtract instruction: depends on Add instruction for register r5
- And instruction: depends on Add instruction for register r5

b. Assume that the pipelined uses full forwarding. Draw a pipelined diagram that represents the flow of instructions through the pipeline during each clock cycle. Indicate forwarding by arrows.

| **Clock cycle** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|
| *LW r4, 100(r2)* | IF | ID | EXE | MEM *(r4 is ready)* | WB | | | |
| *ADD r5, r2, r3* | | IF | ID | EXE *(r5 is ready)* | MEM | WB | | |
| *SUB r6, r4, r5* | | | IF | ID | EXE | MEM | WB | |
| *AND r7, r2, r5* | | | | IF | ID | EXE | MEM | WB |

**Problem 5 (20 points):**

One particular branch (i.e., one specific PC) has the following actual outcomes. Show the predictions for both a one-bit counter and a two-bit counter (no history). For the two-bit counter, use **T** for strongly taken **t** for weakly taken, **n** for weakly not-taken, and **N** for strongly not taken. Finally fill in the accuracy (percentage of predictions that were correct) at the bottom of the table. The first prediction is done for you.

| Branch Outcome | 1-bit predictor | | 2-bit predictor | |
|---|---|---|---|---|
| | Prediction | Correct? | Prediction | Correct? |
| T | T | Yes | T | yes |
| T | | | | |
| T | | | | |
| N | | | | |
| T | | | | |
| N | | | | |
| T | | | | |
| T | | | | |
| T | | | | |
| N | | | | |
| Accuracy | | | | |

| **Number** | **Branch Outcome** | **1-bit predictor** | | **2-bit predictor** | |
|---|---|---|---|---|---|
| | | **Prediction** | **Correct?** | **Prediction** | **Correct?** |
| *1* | **T** | T | Yes | T | Yes |
| *2* | **T** | T | Yes | T | Yes |
| *3* | **T** | T | Yes | T | Yes |
| *4* | **N** | T | No | T | No |
| *5* | **T** | N | No | t | Yes |
| *6* | **N** | T | No | T | No |
| *7* | **T** | N | No | t | Yes |
| *8* | **T** | T | Yes | T | Yes |

| | | | | | |
|---|---|---|---|---|---|
| *9* | **T** | T | Yes | T | Yes |
| *10* | **N** | T | No | T | No |
| ***Accuracy*** | | 50% | | 70% | |