NAME: MURAGURI JOEL
REGISTRATION NUMBER: SCT212-0042/2020
UNIT: Computer Architecture (BCT 2408)

Lab 3

I) E1 – Problem

Task:
Analyze each of the following MIPS code fragments, each consisting of two instructions. Identify any data hazard present, specify its type, and list the registers involved.

Data Hazard Types:

1. **RAW (Read After Write):** Occurs when an instruction tries to read a register before a previous instruction writes to it. This causes pipeline delays.
2. **WAW (Write After Write):** Occurs when two instructions write to the same register, and the second one writes before the first completes.
3. **WAR (Write After Read):** Happens when an instruction writes to a register before a previous instruction reads it.

a.

```
LD R1, 0(R2)
DADD R3, R1, R2
```

Hazard Type: RAW
Explanation: The `LD` instruction loads data into register `R1`. The subsequent `DADD` instruction attempts to use the value of `R1` before the load is completed, creating a Read After Write (RAW) hazard.

b.

```
MULT R1, R2, R3
DADD R1, R2, R3
```

Hazard Type: WAW
Explanation: Both instructions write to register `R1`. The `DADD` instruction could overwrite the result of the `MULT` operation before it completes, resulting in a Write After Write (WAW) hazard.

c.

MULT R1, R2, R3
MULT R4, R5, R6

Hazard Type: None
Explanation: These instructions operate on completely different registers. No dependencies exist between them, so there is no hazard.

d.

DADD R1, R2, R3
SD 2000(R0), R1

Hazard Type: RAW
Explanation: The `SD` instruction stores the value of `R1`, which is being computed by the `DADD` instruction. There is a RAW hazard since `SD` may attempt to use `R1` before `DADD` finishes.

e.

DADD R1, R2, R3
SD 2000(R1), R4

Hazard Type: RAW
Explanation: In this case, `R1` is used as a memory address in `SD` after being calculated by `DADD`. The hazard exists because `SD` relies on the correct computation of the memory address stored in `R1`.

II) E2 – Problem

a.
Topic: 2-Bit Saturating Counter – Branch Predictor Behavior

Explanation:
A 2-bit saturating counter is commonly used in branch prediction to decide whether a branch is likely to be taken or not. The predictor maintains one of four states:

State	Meaning
00	Strongly Not Taken
01	Weakly Not Taken

10	Weakly Taken
11	Strongly Taken

State Transitions:

- **If the branch is taken:**
  - 00 → 01
  - 01 → 10
  - 10 → 11
  - 11 → 11 (remains unchanged)
- **If the branch is not taken:**
  - 00 → 00 (remains unchanged)
  - 01 → 00
  - 10 → 01
  - 11 → 10

Transition Table:

Current State	Prediction	Branch Outcome	New State
00	Not Taken	Not Taken	00
00	Not Taken	Taken	01
01	Not Taken	Not Taken	00
01	Not Taken	Taken	10
10	Taken	Not Taken	01
10	Taken	Taken	11
11	Taken	Not Taken	10
11	Taken	Taken	11

b.

Given Code:

```
for (i = 0; i < N; i++) {
  if (x[i] == 0)
    y[i] = 0.0;
```

```
  else
    y[i] = y[i] / x[i];
}
```

Generated Assembly (simplified):

```
loop: L.D F1, 0(R2)       # Load x[i]
      L.D F2, 0(R3)       # Load y[i]
      BNEZ F1, else       # Branch if x[i] != 0
      ADD.D F2, F0, F0    # y[i] = 0.0
      BEZ R0, fall        # Unconditional jump
else: DIV.D F2, F2, F1    # y[i] = y[i] / x[i]
fall: DADDI R2, R2, 8     # i++
      DADDI R3, R3, 8
      DSUBI R1, R1, 1
      S.D -8(R3), F2      # Store result
      BNEZ R1, loop       # Loop if R1 != 0
```

Assumptions:

- `x[i]` alternates between 0 and non-zero starting with 0.
- Initial branch predictor state: **00** (strongly not taken).
- Prediction is made for the branch `BNEZ F1, else`.

Outcome of Each Iteration Using the Predictor:

Iteration	x[i] == 0	Actual Outcome	Predictor State	Prediction	New State
1	Yes	Not Taken	00	Not Taken	00
2	No	Taken	00	Not Taken	01
3	Yes	Not Taken	01	Not Taken	00
4	No	Taken	00	Not Taken	01
5	Yes	Not Taken	01	Not Taken	00
6	No	Taken	00	Not Taken	01

Conclusion:

Since `x[i]` alternates between 0 and non-zero, the branch outcome alternates between *Not Taken* and *Taken*. As a result, the predictor frequently mispredicts because it is not able to adapt quickly enough due to only moving one state per misprediction