10 Branch Prediction [70 points]

A processor implements an *in-order* pipeline with multiple stages. Each stage completes in a single cycle. The pipeline stalls after fetching a *conditional branch instruction* and resumes execution once the condition of the branch is evaluated. There is no other case in which the pipeline stalls.

10.1 Part I: Microbenchmarking [30 points]

You create a microbenchmark as follows to reverse-engineer the pipeline characteristics:

```
LOOP1:
    SUB R1, R1, #1  // R1 = R1 - 1
    BGT R1, LOOP1   // Branch to LOOP1 if R1 > 0

LOOP2:
    B   LOOP2       // Branch to LOOP2
                    // Repeats until program is killed
```

The microbenchmark takes one input value `R1` and runs until it is killed (e.g., via an external interrupt).

You carefully run the microbenchmark using different input values (`R1`) as summarized in Table 2. You terminate the microbenchmark using an external interrupt such that each run is guaranteed to execute exactly 50 *dynamic instructions* (i.e., the instructions actually executed by the processor, in contrast to *static instructions*, which is the number of instructions the microbenchmark has).

Initial R1 Value	Number of Cycles Taken
2	71
4	83
8	107
16	155

Table 2: Microbenchmark results.

Using this information, you need to determine the following two system characteristics. *Clearly show all work to receive full points!*

1. How many stages are in the pipeline?
2. For how many cycles does a conditional branch instruction cause a stall?

1. 10 pipeline stages
2. 6 cycles

Explanation: We have a system of equations in the variables:

- C is the total number of cycles taken
- P is the total number of pipeline stages
- I is the total number of dynamic instructions executed
- B is the number of conditional branch instructions executed
- D is the number of cycles stalled for each conditional branch

The total number of cycles can be expressed as $C = P + I - 1 + B * D$.
We know that $I = 50$, and Table 2 gives us B and C, which we can use to solve the following system of equations:

- $71 = P + 50 - 1 + 2 * D$
- $83 = P + 50 - 1 + 4 * D$

Solving this system, we obtain $P = 10, D = 6$

10.2 Part II: Performance Enhancement [40 points]

To improve performance, the designers add a *mystery* branch prediction mechanism to the processor. All we know about this *mystery* branch predictior is that it does not stall the piepline at all if the prediction is correct. They keep the rest of the design exactly the same as before. You re-run the same microbenchmark with $R1 = 4$ for the same number of total dynamic instructions with the new design, and you find that the microbenchmark executes in 77 cycles.

Based on this given information, determine which of the following branch prediction mechanisms could be the *mystery* branch predictor implemented in the new version of the processor. For each branch prediction mechanism below, you should circle the configuration parameters that makes it match the performance of the mystery branch predictor.

(a) [10 points] **Static Branch Predictor**

Could this be the mystery branch predictor: `YES` `NO`

If `YES`, for which configuration below is the answer `YES`?

(I) Static Prediction Direction

`Always taken` `Always not taken`

Explain:

> `YES`, if the static prediction direction is `always not taken`.
> **Explanation:** The execution time corresponds to 3 mispredictions and 1 correct prediction. The correct prediction occurs when the branch condition evaluates to FALSE and execution falls through to the following instruction (i.e., NOT TAKEN).

(b) [10 points] **Last Time Branch Predictor**

Could this be the mystery branch predictor?

`YES` `NO`

If YES, for which configuration is the answer *YES*? Pick an option for each configuration parameter.

(I) Initial Prediction Direction

`Taken` `Not taken`

(II) Local for each branch instruction (PC-based) or global (shared among all branches) history?

`Local` `Global`

Explain:

> *NO.*
> **Explanation:** The last-time predictor will make a correct prediction at least three times, which means that it cannot be the mystery predictor.

(c) [10 points] **Backward taken, Forward not taken (BTFN)**

Could this be the mystery branch predictor?

YES NO

Explain:

NO.

Explanation: The BTFN predictor makes exactly one *mis*prediction, which is the opposite of what the mystery predictor achieves.

(d) [10 points] **Forward taken, Backward not taken (FTBN)**

Could this be the mystery branch predictor?

YES NO

Explain:

YES.

Explanation: The FTBN predictor makes exactly one correct prediction, which is what we observe from the microbenchmark.