Branch Predictor Design and Analysis

Project 2 Part 1

Adil Hydari

Department of Computer Engineering

Rutgers University

December 5, 2024

Contents

1 Introduction

Modern processors heavily rely on branch predictors to reduce pipeline stalls and improve instruction-level parallelism. This project explores a simple branch predictor design based on a two-bit state machine. We progressively increase the complexity and correlation history in the predictor to observe its impact on prediction accuracy and overall performance. The implemented predictors include variations with no history and up to three bits of history, capturing both temporal and spatial correlations.

2 Design Methodology

2.1 Overview of the Predictor

The baseline predictor design is a 2-bit saturating counter, initialized to a weakly not-taken state (‘01‘). Each predictor variant differs in the number of history bits used:

- **(0,2) Predictor:** No global or per-branch history. Predictions rely solely on a single 2-bit counter for each branch pattern index.
- **(1,2) Predictor:** One bit of branch history (BHR) is used. This incorporates temporal/spatial correlation by indexing into the Pattern History Table (PHT) using one history bit.
- **(2,2) Predictor:** Two bits of history, allowing the predictor to capture more complex patterns.
- **(3,2) Predictor:** Three bits of history, attempting to exploit even deeper correlation.

The PHT is indexed by the BHR combined with the branch ID. For simplicity, our design uses a global BHR and updates it after every branch is resolved. The final prediction for a

branch is taken if the 2-bit counter is in the upper half of its state space (i.e., states ‘10‘ or ‘11‘), and not-taken otherwise.

2.2 Temporal and Spatial Correlation Incorporation

Temporal correlation is captured by using the same branch's past outcomes to predict its future behavior. As we add more history bits, the predictor can identify repeating patterns (e.g., always taken, alternating taken/not-taken) and improve accuracy.

Spatial correlation is achieved by allowing the BHR to represent outcomes of recently executed branches, influencing the prediction of the current branch. This is useful in scenarios where the behavior of one branch might inform the likely behavior of the next.

2.3 2-bit Saturating Counter FSM

Each PHT entry is a 2-bit counter:

State	Binary	Meaning	Transition
Strong NT	00	Predict Not-Taken	If taken, move to 01
Weak NT	01	Predict Not-Taken	If taken, move to 10; if not-taken, move to 00
Weak T	10	Predict Taken	If taken, move to 11; if not-taken, move to 01
Strong T	11	Predict Taken	If not-taken, move to 10

All counters are initialized to the weak Not-Taken state (‘01‘).

3 Experimental Setup

3.1 Workload and Patterns

We tested the predictor using a program that includes multiple branches:

- Branches with no correlation: random taken/not-taken sequences.
- Branches with strong temporal correlation: repeating patterns (e.g., always taken).
- Branches with alternating patterns to test medium complexity correlation.
- Branches designed to exhibit spatial correlation, where one branch's outcome influences the predictability of another.

The simulation runs each sequence of branch outcomes twice:

1. **Baseline Execution:** No prediction, each branch costs 3 CCs, each non-branch 1 CC. (In our test scenario, we focused mainly on branch instructions.)
2. **Predicted Execution:** Using the implemented predictor. Correctly predicted branches cost 1 CC; mispredicted branches cost 9 CCs due to pipeline flush penalties.

3.2 Metrics Collected

We collected the following metrics:

- **Correct Predictions:** Number of times the predictor outcome matched the actual branch outcome.
- **Mispredictions:** Number of times the predictor outcome differed from the actual branch outcome.

- **Predicted Execution Cycles:** Total simulated cycles with the branch predictor enabled.
- **Speedup:** Ratio of baseline cycles to predicted cycles.

Speedup indicates the performance improvement gained by using the predictor compared to the baseline.

4 Results

4.1 Tabulated Results

After running the simulation, we obtained the following results:

History Size	Predicted Cycles	Correct Predictions	Mispredictions	Speedup
0	200	11	21	0.48
1	80	26	6	1.20
2	80	26	6	1.20
3	104	23	9	0.92

Table 1: Results of Branch Prediction for Different History Sizes

The baseline execution cycles (no predictor) were measured to be 96 CCs.

4.2 Analysis

Baseline: The baseline scenario, where each branch costs 3 CCs and no prediction is used, resulted in a total of 96 cycles.

(0,2) Predictor: With no history bits, the predictor lacks context to accurately predict branches. This resulted in many mispredictions and a significant penalty, increasing total cycles to 200 and yielding a speedup of only 0.48 compared to the baseline. Essentially, the overhead of mispredictions overshadowed any benefit gained.

(1,2) Predictor: Introducing one bit of history improved predictions dramatically, especially for consistently taken or alternating patterns. The predicted cycles reduced to 80, and the speedup rose to 1.20. This indicates that even a small amount of temporal/spatial correlation helps significantly.

(2,2) Predictor: Increasing history to two bits did not further reduce execution cycles compared to one bit. The results remained at 80 predicted cycles, 26 correct predictions, and 6 mispredictions, identical to the (1,2) scenario. This suggests that the tested branches did not exhibit more complex patterns that two bits of history could exploit beyond what one bit already captured. However, this result also be due to the fact that the program used to test the predictor is not complex enough to fully exploit all 2 bits of branch history.

(3,2) Predictor: Using three bits of history introduced slightly more complexity. While one might expect further improvement, the results actually slightly worsened to 104 cycles and a speedup of 0.92. The additional complexity may have overfitted to certain patterns or introduced scenarios where additional history bits did not match the actual patterns tested. Thus, more history is not always beneficial, especially if the pattern does not warrant it or if noise in the pattern negates the benefits of longer history.

5 Conclusions

This study demonstrates how branch prediction performance is strongly influenced by the complexity and nature of the branch patterns in a program. Key takeaways include:

- Adding some history (1 or 2 bits) can greatly improve prediction accuracy for regular patterns, resulting in reduced execution cycles and speedup over the baseline.
- Not all increases in history length lead to further improvements. In some cases, longer history can reduce accuracy if the pattern is not sufficiently correlated.

- The optimal predictor configuration depends on the branch behavior of the workload. For simpler or well-structured patterns, a 1-bit history predictor can provide substantial gains.

The results suggest that carefully choosing the predictor complexity based on the expected branch patterns is crucial. In a real system design scenario, this might involve profiling and adaptive mechanisms to select the appropriate history length dynamically.