Lab 4 Report: Branch Predictors

Yanwen Zhu(yz2949) Yibin Xu(yx623)

Section 1: Introduction

In processors that prioritize high performance, managing control flow hazards is essential to avoid costly delays. Dynamic branch prediction plays a vital role in this context, helping to minimize stalls and ensure efficient throughput. The purpose of branch predictors is to enhance processor performance by precisely forecasting the outcomes of branching commands. This allows the processor to continue operations smoothly, even when faced with unresolved branches. The focus of this lab exercise is to explore and assess various branch prediction methods. We will implement three distinct architectures of varying sizes: the Bimodal Branch Predictor, the Global Branch Predictor, and the GShare Branch Predictor. Additionally, we will employ a robust testing strategy to validate these predictors and carry out a comparative analysis of their performance. After evaluation we can see that the bimodal predictor, known for its simplicity and speed, may falter in accuracy under complex conditions; the global predictor, while more accurate in intricate situations, can be prone to aliasing issues; the GShare predictor offers a compromise, mitigating aliasing concerns while still maintaining considerable accuracy.

Section 2: Baseline Design

Section 2.1 Design of Bimodal Branch Predictor

In the design of the Bimodal Branch Predictor (figure 1), the clk (clock signal) is used in synchronizing the updates to the predictor's state. The reset is used to clear the internal state of the branch predictor, returning it to its initial condition. The pc (program counter) represents the address of the branch that is currently being predicted. The signal update_en (update enable) serves as an indicator for the branch predictor, signaling when to modify its internal state; this occurs when the signal is asserted. The update_val (update value) carries the actual result of the branch operation, with 0 indicating a branch not taken and 1 indicating a branch taken. The output of the branch predictor, referred to as the prediction, reflects the decision made by the predictor. It uses a binary representation where 00/01 signifies a prediction of the branch not being taken, and 11/10 indicates a prediction of the branch being taken.

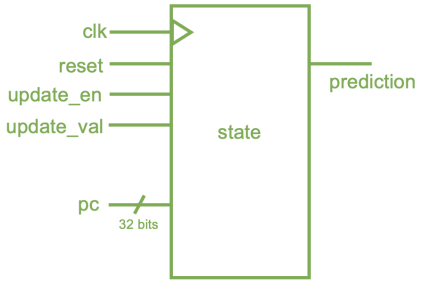

Figure 1. Block diagram

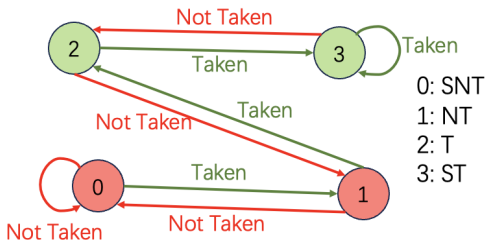

Figure 2. The states in a two-bit prediction scheme

The Bimodal Predictor operates by utilizing the pc (program counter) value and its internal state for making branch predictions. The prediction process is combinatorial, relying on the current pc value and the predictor's internal state. The output of the predictor is binary: 0 indicates a prediction of the branch not being taken, and 1 signifies a prediction of the branch being taken. The internal state of the predictor undergoes updates at the positive edge of the clock cycle, provided that the update_en signal is active. These updates depend on the pc value at the time and the provided update_val. Moreover, the predictor is designed to accommodate a parameter named pht_size, which defines the number of entries in the Pattern History Table (PHT). Adjusting pht_size allows for an analysis of the predictor's accuracy across different scenarios. The PHT exclusively employs two-bit saturating counters for its entries.

In the Bimodal Predictor, the two-bit counter method is essential for understanding branch behaviors. This method is elucidated in figure 2. In this scheme, every conditional branch occurrence is associated with a two-bit counter in the PHT. These counters, with values 0, 1, 2, or 3, correspond to the states of Strong Not Taken, Not Taken, Taken, and Strong Taken, respectively. If a branch is taken, the associated counter increments; conversely, it decrements when a branch is not taken. Thus, a prediction needs to be incorrect twice before it undergoes a change. The decision-making process is primarily influenced by the most significant bit of the counter. Additionally, the state register plays a crucial role in gathering historical data, which it then uses to influence future predictions. Consequently, as branch instructions are executed, the state of each branch's entry in the buffer dynamically evolves.

The Bimodal Predictor maintains a counter table where the state of each counter is recorded, capturing the history of all branches. In this setup, every branch is associated with a distinct counter. The indexing of the branch history table relies on specific bits of the branch's program counter (pc) address. When the counter table is sufficiently large, such as being over 128K bytes, each branch can be accurately mapped to a unique counter without issues. However, in cases where the table size is smaller, the effectiveness of precise prediction can be compromised due to multiple branches sharing the same counter. Despite this limitation, a smaller-sized predictor still outperforms the basic two-bit scheme in terms of efficiency. The structural design of the Bimodal Predictor is depicted in figure 3. This structure outlines how branches are mapped to their respective counters and illustrates the mechanism through which the predictor manages and utilizes the history of branch executions to inform its predictions. The balance between table size and prediction accuracy becomes a key consideration in the design and effectiveness of the Bimodal Predictor.

Section 2.2 Design of Global Branch Predictor

The Global Predictor distinguishes itself by utilizing the collective history of all branches for prediction purposes, rather than relying on the history of just a single branch. This approach involves a Pattern History Table (PHT) that aligns global history patterns with their respective predictions. In contrast to the Bimodal Predictor, which considers the pc (program counter), the Global Predictor exclusively depends on its internal state to predict branch behavior. This prediction scheme capitalizes on the recent activities of various branches. It consolidates the history information into a single shift register, named GR, which chronicles the direction (either taken or not taken) of the most recent n conditional branches. The presence of GR allows the Global Predictor to achieve a level of precision in forecasting that is on par with local predictors. Additionally, the use of a single global register in place of a traditional history table helps minimize the complications arising from multiple branches competing for the same history table entries, a common occurrence since branches often follow similar paths. However, the global register approach presents its own set of challenges. Notably, it struggles more with pinpointing the exact current branch address compared to local predictors. This difficulty arises because the global register does not maintain a comprehensive log of all branch addresses. The design and operational mechanism of the Global Predictor, which synthesizes global branch history for making predictions, is detailed in figure 4.

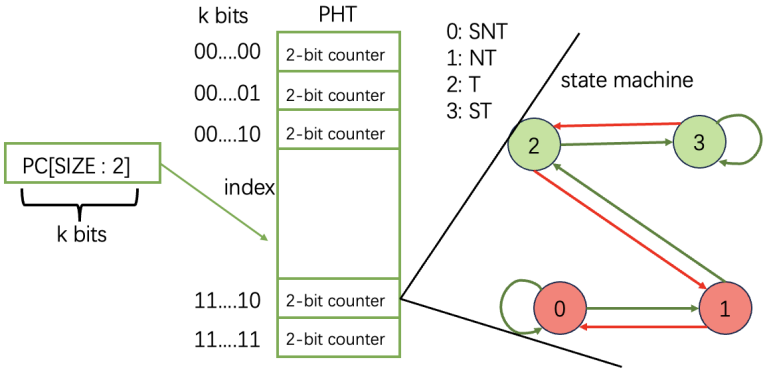

Figure 4. Global Predictor Structure

Section 2.3 Design of GShare Branch Predictor

The GShare Predictor, a variant of the global predictor, enhances its approach by amalgamating the global history with the branch address to determine the indexing into the Pattern History Table (PHT). This combination is achieved by "xor-ing" (exclusive OR operation) the global history with the branch address, an effective strategy to diminish aliasing effects. In the GShare predictor's design, the global history register (GHR) is XORed with the pc (program counter) for indexing into the PHT. Recognizing the challenges faced by the global predictor in accurately identifying branch addresses, the GShare predictor, as coined by McFarling, advances the methodologies of both global and Bimodal branch prediction schemes. There are two primary modifications in the GShare scheme compared to the global and Bimodal approaches. The first is the method of integrating branch address bits with history bits. Instead of simple concatenation, the GShare predictor employs the XOR operation to merge these bit types, which results in a more distinct indexing process. The second key modification is the inclusion of a greater number of branch address bits in this amalgamation, enhancing the GShare predictor's ability to easily pinpoint the current branch address. The architectural design of the GShare or global sharing branch predictor is detailed in figure 5, showcasing how it innovatively combines elements of global history and branch addresses for more refined prediction accuracy.

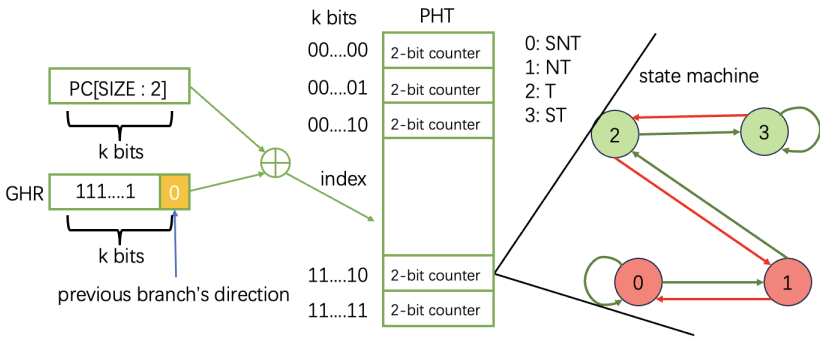

Figure 5. GShare Predictor Structure

Section 4: Testing Strategy

We have developed three distinct test benches to evaluate different versions of branch prediction: utb_BranchBimodal.v, utb_BranchGlobal.v, and utb_BranchGShare.v. These test benches comprehensively address the scenarios outlined in the provided handout. Each branch prediction version is subjected to three testing strategies: a loop with a single branch, a loop with multiple branches (N-branches), and a sequence of varied branches. These strategies include test patterns like always taken, never taken, alternating, and complex sequences such as AAB, BBA, AAAB, BBBA, ABBA, and ABAB. While similar testing approaches are applied across all versions, notable differences exist. For instance, the utb_BranchGlobal.v shows limited significant results in the N-branches and varied branches tests, as it relies more on prediction history than the Program Counter (PC) number.

- single branch
    - We only have one branch, it is always taken.
    - We only have one branch, always not taken.
    - We only have one branch, alternating between taken and not taken.
    - loop with various other patterns, AAB, BBA, AAAB, BBBA, ABBA, and ABAB.
- loop with N-branches
    - We only have four branches, always taken.
    - We only have four branches, always not taken.
    - We only have four branches, alternating between taken and not taken.
    - loop with various other patterns, AAB, BBA, AAAB, BBBA, ABBA, and ABAB.
- a series of branches
    - We loop through the 2048 number of branches always taken.
    - always not taken.
    - alternating between taken and not taken.

To enhance clarity, we suggest incorporating a table summarizing the test cases and their outcomes. The testing methodology section could start with an overview paragraph, outlining the cohesive strategy behind these tests. The single branch test case, our simplest, validates basic prediction functionality before progressing to more complex scenarios. The loop with N-branches tests the predictor's response to closely spaced branch instructions (e.g., PC: 200, 204, 208, 20c), while the series of branches assesses the system's handling of a broader array of instructions to identify any potential edge cases in our design.

	single branch	loop with N-branches	a series of branches
utb_BranchBimodal	38 test case	53 test case	53 test cases
utb_BranchGlobal	97 test case	49 test cases	97 test cases
utb_BranchGShare	304 test case	304 test cases	304 test cases

Table 1. Number of test for each branch prediction

Section 5: Evaluation

We used the script to run all 20 benchmarks, and calculated the average prediction rates of the three predictors at each size (from 4 bytes to 16384 Bytes), as shown in the table and line chart below.

	PHT Size (Bytes)												
Design	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
Bimodal	76.8	81.4	84.3	86.6	88.9	90.4	91.4	92.1	92.4	92.6	92.7	92.8	92.8
Global	75.1	77.8	79.8	82.1	85.1	87.2	88.9	90.5	91.9	93.0	93.8	94.2	94.6
GShare	72.8	75.4	79.0	82.0	84.7	87.4	89.6	91.9	93.0	94.1	94.8	95.2	95.5

Table 2. Average Prediction Accuracy

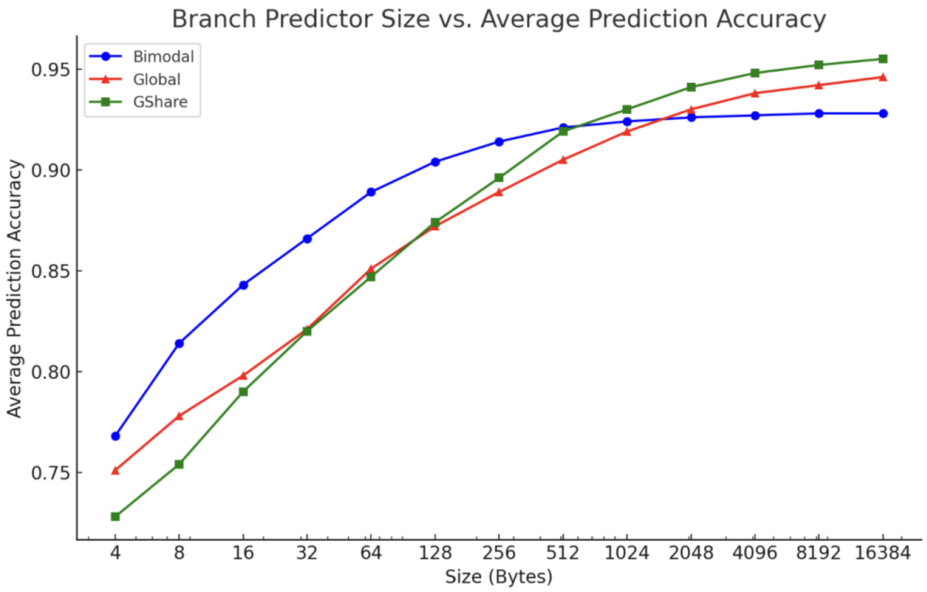

Figure 6. Branch Predictor Size vs. Average Prediction Accuracy

Based on the data presented in the line graph and table, we can evaluate the three branch predictors (Bimodal, Global, and GShare) as follows:

1. Performance with Increasing Size:
   Bimodal: This predictor performs well at smaller sizes, but the rate of improvement in average prediction accuracy diminishes as size increases. This suggests that beyond a certain point, increasing size has a limited effect on enhancing performance. Global: The Global predictor shows a significant improvement in performance as size increases. Especially in the medium size range, it demonstrates the most noticeable improvement, surpassing Bimodal and GShare. GShare: The GShare predictor performs moderately at smaller sizes but shows significant improvement with increased size, achieving the highest accuracy at larger sizes.

2. Performance Comparison:
   At smaller sizes (e.g., 4 Bytes and 8 Bytes), the Bimodal predictor performs the best. In medium to larger sizes (starting from around 256 Bytes), the GShare predictor exhibits the highest accuracy. The Global predictor performs well in medium sizes (e.g., 64 to 1024 Bytes) but its improvement is not as pronounced as GShare at very large sizes.

3. Optimization Suggestions:
   For resource-constrained environments, the Bimodal predictor is a good choice as it offers high accuracy at smaller sizes. When more resources are available, the GShare predictor is the best, as it provides the highest prediction accuracy at larger sizes. The Global predictor is well-suited for situations where a balance between performance and resource allocation is needed, especially in medium size ranges.

Section 6: Conclusion

Bimodal is simple and fast but less accurate in complex scenarios especially in cases where branch behavior is dependent on the global history of branch outcomes. So they are suitable for scenarios where branch behavior is relatively consistent and not influenced by the outcomes of other branches. Global offers higher accuracy in complex scenarios than bimodal predictors in complex scenarios where branch behavior is influenced by the outcomes of other branches. But they can suffer from aliasing issues where different branch histories map to the same pattern. So they are effective in applications with complex control flow where branch decisions are influenced by the global execution context. GShare provides a good balance between the simplicity of the bimodal approach and the accuracy of the global predictor. The use of XOR operation helps in distinguishing between different branch histories, which can reduce aliasing while maintaining good accuracy which is useful in applications where both the individual branch behavior and global history are important for prediction accuracy. In summary, choosing the appropriate branch predictor depends on the specific resource constraints and performance requirements. Each predictor has its own application scenarios and advantages.

Section 7 Work Distribution

Yibin Xu and Yanwen Zhu have collaborated seamlessly on both the predictors designs and unit tests. Yibin Xu took the lead in unit tests and Yanwen Zhu focused on predictors design.