Dynamic Branch Prediction Within RISC-V

Jesus Arias, Jaeden Carpenter, James Fulton, Eric Lawson
Electrical and Computer Engineering 562
University of Arizona
Tucson, Arizona, USA
{jearias, carpenterjaeden, jamesfulton, emlawson1}@arizona.edu

***Abstract*— RISC-V has become increasingly popular for small embedded microcontrollers. Dynamic branch prediction is one of the many implementations used to improve performance to determine whether a branch instruction will be taken or not. This branch predictor is a 2-bit bimodal branch predictor coded in Verilog and ran using ModelSim. For the average speedup and mispredicts, the 2-bit branch predictor has a better performance than the 1-bit branch predictor and static not taken predictor. The paper concludes that the 2-bit branch predictor on average optimizes the performance of the pipeline depending on the workload. The paper also suggests possible enhancements and outlines potential future developments for dynamic branch prediction.**

***Keywords*— dynamic branch prediction, 2-bit branch predictor, RISC-V, ModelSim, Verilog**

1. INTRODUCTION

Static branch prediction is a simple technique that predicts the outcome of a conditional branch instruction based solely on the branch instruction itself. It does not take into account any previous branch predictions or outcomes. The prediction is made at compile time and stored in the instruction cache along with the branch instruction. If the prediction is correct, the processor can continue executing instructions along the predicted path without delay; but if it is incorrect, a pipeline stall occurs as the processor discards the incorrectly fetched instructions and redirects the program to the correct path. The effectiveness of static branch prediction is limited to programs with predictable behavior and static branch prediction does not provide significant performance improvements for complex programs with unpredictable branch behaviors.

In contrast, dynamic branch predictors use the program's execution behavior to adjust their branch predictions. By storing information about previous branch behavior, including the program counter indicating the current instruction, dynamic branch predictors can adapt to changes in the program's behavior to make more accurate dynamic predictions. Furthermore, dynamic branch predictors can detect patterns in the behavior of conditional branches, such as loops or function calls, and use this information to improve overall pipeline performance [1]. As a result, dynamic branch predictors are generally considered more effective than static branch predictors for improving performance in modern processors, especially for complex programs with unpredictable branch behavior. Also, RISC-V architecture is designed to have a simple instruction set, which makes it easier for branch predictors to accurately predict the outcome of conditional statements and loops. Additionally, RISC-V allows for flexible implementation of branch prediction strategies [2], which can help optimize performance for different types of programs and workloads.

Dynamic branch prediction is a technique that helps to increase the efficiency of pipelining in modern processors by minimizing the number of stalls in the pipeline that can be caused by a missed fetch [3]. By predicting the outcome of branch instructions before they are executed, dynamic branch prediction can help to reduce the number of stalls in the pipeline, leading to better performance for the processor.

A 1-bit predictor only has two states: not taken (0) and taken (1). If the state is not taken, the branch will predict not taken, and if the decision is also not taken, it stays in the same state. However, if it predicts taken, then it moves to the taken state. In the taken state, it stays there if the branch is taken. But if the predictor predicts taken and sees that the branch is not taken, then it goes to the not taken state and this cycle could repeat many times which results in a decrease in performance. Essentially, one outcome can change the state if the branch outcome is different from the state. This results in two mispredictions per anomaly such as taken → not taken → taken.

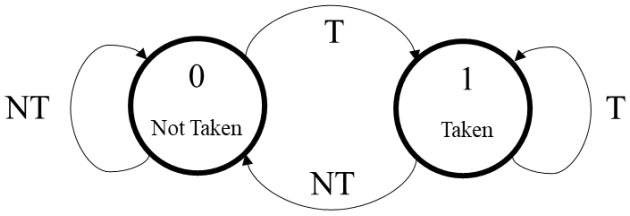

Figure 1: 1-bit predictor flow diagram..

A popular type of dynamic branch predictor is the 2-bit predictor or 2-bit counter (2BP or 2BC), which improves upon the behavior of the 1-bit predictor. The 2-bit predictor uses a two-bit counter that counts up from zero to three and saturates when the branch is continuously taken, and counts down from three to zero and saturates when the branch is not taken. The most significant bit of the counter represents the prediction bit, which behaves like a normal 1-bit predictor, while the lower bit is called the hysteresis or conviction bit, which determines how certain the prediction is based on the previous branch.

A 2-bit branch predictor has four states, as shown in Figure 2. Typically, state 00 is treated as a strong not taken state, meaning that the predictor is highly convinced that the next branch will not be taken. State 01 represents a weak not taken prediction, indicating that the predictor is less certain that the branch will not be taken. The other two states, 10 and 11, correspond to weak taken and strong taken predictions, respectively. By using these four states, the 2-bit branch predictor can make more accurate predictions and reduce the number of mispredictions, thereby improving the performance of the processor.

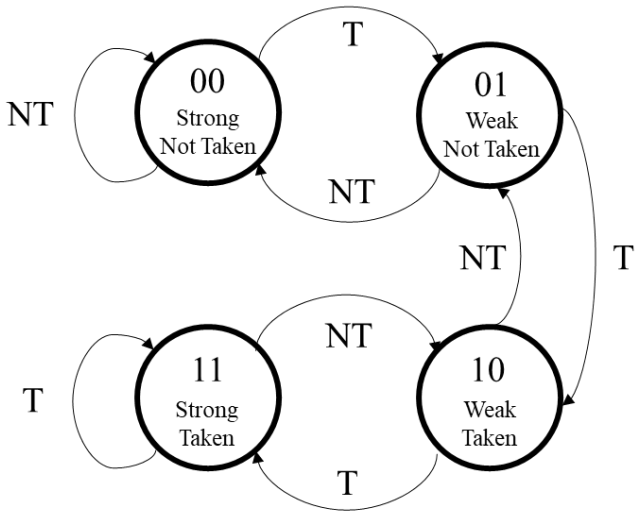

Figure 2: 2-bit Branch Predictor FSM

2. Related Works

Pan, So, and Rahmeh [4] provide two different types of dynamic branch prediction: Counter-Based Branch Prediction and Correlation-Based Branch. They mention in their results that Counter-Based Branch Prediction is limited by the counter size, where a larger counter-size does not necessarily give better results, and Correlation-Based Branch Prediction is better for "Scientific/Engineering workloads where program execution is dominated by inner-loops". Both of these findings, specifically their similarities with *repeated actions increasing confidence,* give understanding for the use of a confidence based 2-bit predictor.

Yeh and Pat [5], reference the work of Pan, So, and Rahmeh [4] and expand on the work of their Correlation-Branch Prediction by explaining that there are actually nine variations of the basic prediction model of Correlation-Branch Prediction. They devised a two-level table where a combination of the differentiating ways branch history information is kept. The nine variations were a two-level combination of Global (G), Per-Address (A), and Per-Set (S). For example GAs, Global Adaptive Branch Prediction using per-set pattern history tables was determined to be the most accurate.

In the paper, Nair [6] conducts a thorough review of the various types of branch predictors, with a particular focus on the 2-bit dynamic branch predictors that are commonly used. The paper presents an expression to calculate the probability of a misprediction and analyzes the trade-offs between different models. Additionally, the paper includes the code used for the project and presents the results from various benchmarks that demonstrate the effectiveness of the 2-bit branch predictor. Nair also explores different variations of the model, such as the starting point for the 2-bit dynamic branch predictor, and highlights their benefits for various applications.

Different initial states were considered for the 2-bit branch predictor in the project like the ones used in Nair's paper [6], including the bimodal initial state (11) or the weakly taken initial state (01). However, to maintain consistency across experiments and for simplicity, the strong not taken state (00) was chosen as the initial state. While alternative initial states could potentially have resulted in different performance outcomes, the focus of the project was to compare different branch predictors such as the 1-bit predictor or no predictor with the 2-bit predictor. By using a consistent initial state for the 2-bit predictor, a direct comparison of these different predictors could be made under the same conditions.

The paper by Jiménez, Keckler, and Lin [7] discusses the number of history bits and the size of the pattern history table and how they are key design parameters that can affect the accuracy of the predictor as well as its access time and storage requirements. It highlights how there are different tradeoffs between the accuracy, access time, and storage requirements based on different types of branch predictors, including 2-bit branch predictors. Jiménez and Lin [8] also published another paper talking about how the 2-bit branch predictor is not always the most efficient or accurate for every application. They discuss an alternative variation between the 2-bit and 1-bit branch predictor for complex workloads. But they note how the 2-bit branch predictor can still provide good optimal performance based on the application.

Despain and Su [9] in their paper introduce different comparisons of static vs dynamic branch predictors integrated into a superpipelined processor. They present different modeling schemes of the branch target buffer and discuss the accuracy of the different models. They conclude in their research that the dynamic branch schemes have a higher prediction accuracy than those of static schemes. In our research when we compare the static vs dynamic predictors, the dynamic branch predictors are more accurate than the static predictor.

3. Methodology

a. Benchmark Application

Benchmarks to test the accuracy of each predictor (1-bit, 2-bit, and static predictor) were created for two reasons.

1. There are no widely available benchmarks in RISCV assembly; most are in c++ and need to be compiled. Existing examples may have unknown interactions with unsupported instructions in our datapath.
2. Ease of debugging since the final behavior and current interaction with our datapath is known.

Our benchmarks will be consistently used to test and analyze each predictor separately. Each of our own benchmarks introduces incrementally different looping branch behaviors to analyze the behavior of different models of branch predictors.

b. Branch Predictor Integration

For this project, we used ModelSim to simulate the behavior and the performance of the 2-bit, 1-bit, and Static branch predictors. The branch predictor contains a register named table1 that contains two bits. The two bits are the previous prediction and the conviction bit. The implemented logic allows the predictor to use these values to properly implement 2-bit dynamic prediction. The base RISCV data-path that was implemented came from Mohammed Mujtaba Roohani's public template on GitHub[10]. The tables below shows the files that we added or modified and why:

TABLE 1: Files added to the project.

Added Files	Reason
XOR.v	Branch Predictor Logic
DataOutput.v	Keeps track of number of cycles, branches, and branch mispredicts
Branch_Predictornone.v	Static not taken prediction
Branch_Predictor.v	2-bit branch predictor
Branch_Predictor1bit.v	1-bit branch predictor
InstructionMemory.v	Allows for use of .mem files (replaced the previous InstructionMemory file)
Branch_Control.v	Decodes branch in IF stage for prediction

Table 2: Modified files to the project.

Modified Files	Reason
Control_Unit.v	Added arithmetic immediate instruction decode capability
ALU_64_bit.v	Fixed zero flag so that it works properly
EXRegister.v	added flushing
IDRegister.v	added flushing
extractor.v	added arithmetic immediate value extraction capability
RISC_V_Processor.v	Added necessary wiring to incorporate the branch predictor as well as other changes to datapath.

The program that was used for simulating the data path is ModelSim. In order to run the simulation in ModelSim, the directory must first be opened by entering: "cd "[/path/to/directory]/PipelinedProcessor" into the transcript window. Then run the simulation by entering: "do run.do". The simulation should run BenchMark6 using the 2-bit branch predictor. To change the benchmark, the InstructionMemory.v file must be edited so that it is reading from the correct file. To change the branch predictor model, the Predictor unit in the RISC_V_Processor.v file can simply be changed from a Branch_Predictor module to either a Branch_Predictornone or a Branch_Predictor1bit module. None of the wirings change between the different predictors. The project can be opened by entering: "project open [/path/to/directory]/PipelinedProcessor/ECE_562" into the transcript window.

The code for the 2-bit branch predictor is shown below:

```
module Branch_Predictor(
        input clk,
        input branchex,
        input outcome,
        output reg prediction
);

 reg [1:0] table1; // 2-bit counter register
initial begin
  table1 <= 2'b00;
  prediction <= 0;
end
```

```
always @(posedge clk) begin
  if (branchex)begin
    case (table1)
      2'b00: begin
        if (outcome) begin
          prediction <= 0;
          table1[1:0] <= 2'b01;
        end
        else begin
          prediction <= 0;
          table1[1:0] <= 2'b00;
        end
      end
      2'b01: begin
        if (outcome) begin
          prediction <= 1;
          table1[1:0] <= 2'b10;
        end
        else begin
          prediction <= 0;
          table1[1:0] <= 2'b00;
        end
      end
      2'b10: begin
        if (outcome)begin
          prediction <= 1;
          table1[1:0] <= 2'b11;
        end
        else begin
          prediction <= 0;
          table1[1:0] <= 2'b01;
        end
      end
      2'b11: begin
        if (outcome) begin
          prediction <= 1;
          table1[1:0] <= 2'b11;
        end
        else begin
          prediction <= 1;
          table1[1:0] <= 2'b10;
        end
      end
    endcase
  end
end
endmodule
```

This model ran a series of simulations where different inputs were fed into the system and outputs were analyzed to determine the system's behavior under different conditions and test cases. After running the simulations, the results were analyzed to determine the accuracy of the 2-bit dynamic branch predictor. Then, a comparison of the 2-bit branch predictor between the 1-bit branch predictor and the static branch predictor was performed.

4. Results

The experiment compared the performance of three different branch prediction schemes - a 2-bit dynamic branch predictor, a 1-bit dynamic branch predictor, and static branch-not-taken predictor. These predictors were implemented using Verilog and evaluated using six different benchmarks. Several metrics were used to evaluate their performance, including instruction count, number of cycles simulated, percent of branches, and percent of mispredicts.

As seen in Figure 3, in every benchmark besides benchmark 3 the 2-bit branch predictor has a lower percentage of mispredicted branches versus the 1-bit and static branch predictor. In benchmark 3, one of the test cases was switching back and forth between taken and not taken multiple times and as a result the 1-bit branch predictor had a lower percentage of mispredictions as expected. This benchmark proves in certain applications the 1-bit branch predictor has a specific case where it performs better. However, in the majority of cases the 2-bit branch predictor performs the best. The reason for this is because the 2-bit branch predictor remembers the previous two branches taken and is able to decide based on those branches. The 2-bit branch predictor keeps track of the branch history predicting the outcome based on the past behavior of the branch. This allows it to make more accurate predictions than a 1-bit predictor that only remembers the last outcome. Additionally, the 2-bit predictor has more states and can adjust its prediction quickly in response to changes in branch behavior. These factors contribute to the lower percentage of mispredicts on average compared to the 1-bit predictor and static predictor.

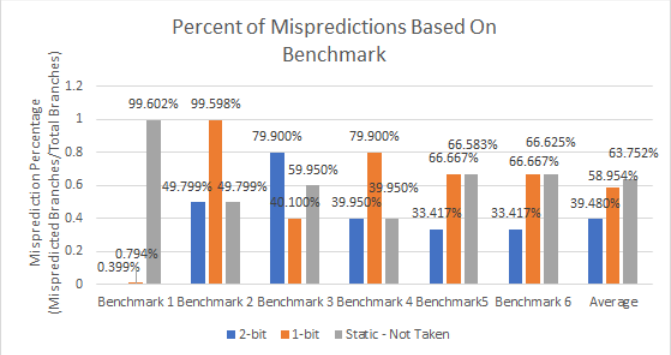

Figure 3: Misprediction percentages for each benchmark.

Figure 4 was used to establish a baseline for comparison between the 2-bit and 1-bit branch predictors, using the static branch predictor. The findings are consistent with those in Figure 3, showing that, with the exception of benchmark 3, the 2-bit branch predictor outperformed the 1-bit branch predictor in terms of speedup. Notably, for benchmarks 2 and 4, both figures indicate that the 2-bit branch predictor was equivalent to the static branch predictor. This is due to an equal number of mispredicts by both predictors, resulting in equivalent speedup times. The averages in Figure 4 have similar reasoning to that stated for Figure 3, where the 2-bit branch predictor is able to make more accurate predictions

based on the past two branches resulting in a faster execution time.

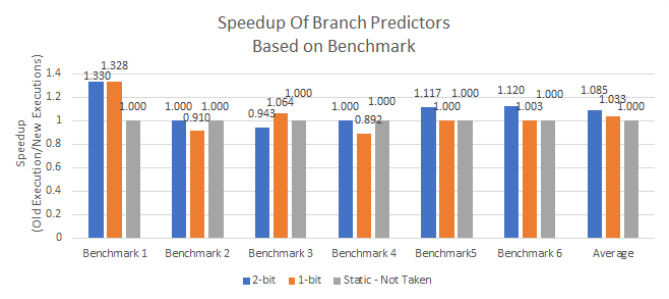

Figure 4: Speedup of branch predictors.

The results showed that the 2-bit dynamic branch predictor outperformed the other two predictors in a majority of benchmarks. On average, the 2-bit dynamic branch predictor had lower misprediction rates and required fewer instructions and cycles to execute. The 1-bit dynamic branch predictor on average performed better than the no predictor case, but the 1-bit dynamic branch predictor had higher averaged misprediction rates than the 2-bit predictor. The no predictor case had the highest misprediction rates and required the most cycles and instructions to execute. Overall, the average of the benchmarks demonstrated the selective effectiveness of using a 2-bit dynamic branch predictor to improve processor performance.

5. Conclusion

The results of our experiment demonstrate the effectiveness of using a 2-bit dynamic branch predictor to improve processor performance over a wide range of workloads. In the benchmarks, the 2-bit predictor on average outperformed the 1-bit predictor and the no predictor case with lower misprediction rates and requiring fewer instructions and cycles to execute. One of the key advantages of using a 2-bit dynamic branch predictor in pipelining is that it reduces pipeline stalls caused by branch instructions [5]. Pipeline stalls occur when the processor encounters a branch instruction that changes the program counter, but the target of the branch is not yet known. This can cause the pipeline to stall as the processor waits for the branch target to be computed.

By using a 2-bit dynamic branch predictor, the processor can make an educated guess about the target of the branch, allowing the pipeline to continue executing instructions without waiting for the target to be computed. Therefore, by reducing the number of pipeline stalls, a 2-bit dynamic branch predictor can increase the number of instructions that the pipeline can execute per unit of time, improving the overall efficiency of the processor. In conclusion, our experiment demonstrates that using a 2-bit dynamic branch predictor can significantly improve the performance of a pipelined processor over an average of different workloads.

6. Future Works

There are several avenues for future work related to 2-bit branch predictors. One potential area is the use of machine learning techniques to improve the accuracy of 2-bit predictors. The use of neural networks to enhance branch prediction accuracy, and similar approaches could be applied to 2-bit predictors. Another potential area for future work is the exploration of different initial state configurations for 2-bit predictors. While the strong not taken state is a commonly used initial state and is what was used in this experiment, other configurations may provide better performance for certain types of applications or programs. Additionally, it may be valuable to explore the use of 2-bit predictors in different processor architectures, such as those with multiple cores or heterogeneous architectures. By refining and improving 2-bit branch predictors, the performance of modern processors can enhance their ability to execute complex workloads.

One area of future research for 2-bit branch predictors is the investigation of alternative storage schemes that may improve their accuracy or reduce their storage requirements. For example, research has shown that some hybrid predictor schemes, which combine elements of 2-bit and other predictor types, can provide improved accuracy while using relatively small amounts of storage [11]. Additionally, research has explored the use of history-based predictors that utilize patterns in the sequence of branch instructions to improve their accuracy. By exploring alternative storage schemes and prediction strategies, it may be possible to further optimize the performance of 2-bit branch predictors and extend their applicability to a wider range of programs and workloads.

Another area of future work for 2-bit branch predictors is the exploration of their performance in the context of emerging processor architectures. For example, the use of multiple cores or heterogeneous processing elements may pose new challenges for branch prediction, and it may be valuable to investigate the effectiveness of 2-bit predictors in these contexts. Additionally, the continued growth of machine learning and artificial intelligence applications may require specialized branch prediction strategies that are tailored to these workloads [12]. By exploring the use of 2-bit predictors in different processor architectures and application domains, it may be possible to uncover new opportunities for performance optimization and innovation in computer architecture.

References

[1] James E. Smith. 1981. “A study of branch prediction strategies.” *In Proceedings of the 8th annual symposium on Computer Architecture (ISCA '81)*. IEEE Computer Society Press, Washington, DC, USA, 135–148.

[2] K. Hepola, J. Multanen and P. Jääskeläinen, "OpenASIP 2.0: Co-Design Toolset for RISC-V Application-Specific Instruction-Set Processors," *2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, Gothenburg, Sweden, 2022, pp. 161-165, doi: 10.1109/ASAP54787.2022.00034.

[3] Lee and Smith, "Branch Prediction Strategies and Branch Target Buffer Design," in *Computer*, vol. 17, no. 1, pp. 6-22, Jan. 1984, doi: 10.1109/MC.1984.1658927.

[4] Pan, Shien-Tai, Kimming So and Joseph T. Rahmeh. “Improving the accuracy of dynamic branch prediction using branch correlation.” *ASPLOS V* (1992).

[5] Tse-Yu Yeh and Y. N. Patt, "A Comparison Of Dynamic Branch Predictors That Use Two Levels Of Branch History," *Proceedings of the 20th Annual International Symposium on Computer Architecture*, San Diego, CA, USA, 1993, pp. 257-266, doi: 10.1109/ISCA.1993.698566.

[6] R. Nair, "Optimal 2-bit branch predictors," in *IEEE Transactions on Computers*, vol. 44, no. 5, pp. 698-702, May 1995, doi: 10.1109/12.381956.

[7] D. A. Jimenez, S. W. Keckler and C. Lin, "The impact of delay on the design of branch predictors," *Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-33 2000*, Monterey, CA, USA, 2000, pp. 67-76, doi: 10.1109/MICRO.2000.898059.

[8] D. A. Jimenez and C. Lin, "Dynamic branch prediction with perceptrons," Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture, Monterrey, Mexico, 2001, pp. 197-206, doi: 10.1109/HPCA.2001.903263.

[9] Ching-Long Su and A. M. Despain, "Minimizing branch misprediction penalties for superpipelined processors," Proceedings of MICRO-27. The 27th Annual IEEE/ACM International Symposium on Microarchitecture, San Jose, CA, USA, 1994, pp. 138-142, doi: 10.1109/MICRO.1994.717451.

[10] Mohammed Mujtaba Roohani, “RISC-V processor: A Verilog based 5-stage pipelined RISC-V processor code.,” GitHub. [Online]. Available: https://github.com/MMujtabaRoohani/RISC-V-Processor. [Accessed: 27-Mar-2023].

[11] P. Trivedi and S. Shah, "Reduced-hardware Hybrid Branch Predictor Design, Simulation & Analysis," 2019 *7th International Conference on Smart Computing & Communications (ICSCC)*, Sarawak, Malaysia, 2019, pp. 1-6, doi: 10.1109/ICSCC.2019.8843638.

[12] Y. Mao, H. Zhou, X. Gui and J. Shen, "Exploring Convolution Neural Network for Branch Prediction," in IEEE Access, vol. 8, pp. 152008-152016, 2020, doi: 10.1109/ACCESS.2020.3017196.