A Survey of Branch Prediction Techniques on Out-of-Order Processors with RISC-V Core (SURGE)

Raja Kantheti *University of Colorado at Colorado Springs*
rkanthet@uccs.edu

Abstract—Branch prediction is a critical aspect of performance tuning in modern processors. The performance of a processor is highly dependent on the accuracy of the branch prediction. This paper presents a survey of branch prediction techniques on out-of-order processors with RISC-V core. This survey paper, provides an evaluation of the branch prediction techniques, with emphasis on their Strengths and Weakenesses and offers insight into their implementation complexity and hardware overhead.

Index Terms—Branch Prediction, Computer Architecture, Performance, High Performance Computing.

1 Introduction

As the complexity in the workloads on the processors increases, the need for efficient branch prediction techniques becomes more important. The system throughput is highly dependent on the accuracy of the branch prediction. Branch mispredictions can cause control hazards and incur penalties such as pipeline stalls until all the instructions in the pipeline are flushed which impacts the performance of the processor negatively. As micro architectures are becoming deeply pipelined, a need for an accurate branch prediction mechanism is essential. [1].

This paper aims to study the below listed branch prediction techniques on out-of-order processors with RISC-V core:

1) Local
2) L-TAGE
3) BiMODE
4) Tournament
5) TAGE-SC-L
6) Multi-Perspective Perceptron
7) Multi-Perspective Perceptron with TAGE

Out-Of-Order CPU is a type of processor that allows the execution of instructions in an order that is different from the order in which they appear in the program. They basically segregate the instructions based on dependencies and executes the independent instructions rapidly. Almost all the modern processors are out-of-order processors. RISC-V is an open-source instruction set architecture (ISA) based on reduced instruction set computing (RISC) principles. It is a simple and clean ISA that is easy to understand and implement. The techniques discussed in this paper are as follows.

The rest of the paper is organised as follows: Section 3 presents related research work on branch prediction. Section 4 includes a detailed explanation on the branch prediction techniques simulated. Section 5 the exprimental setup in gem5 simulator. Section 7 presents the results of the simulation. Section 8 gives perspectives and concludes the paper.

2 Background

Here, I present a overview of branch predictors and their classification with respect past research.

2.1 Useful Terms and Concepts

- **Branch Prediction**: A technique used in processors to guess the direction of branch instructions (whether they will be taken or not) to improve instruction flow in pipelined architectures.
- **Misprediction**: The event when the processor incorrectly predicts the outcome of a branch instruction, leading to performance penalties such as pipeline stalls or flushes.
- **Static Prediction**: A branch prediction method that relies on fixed heuristics, determined at compile time, rather than runtime behavior. Common strategies include using the direction of the branch (e.g., forward branches are likely taken).
- **Dynamic Prediction**: A more sophisticated prediction method that utilizes historical information from previous branch executions to inform future predictions. This can be achieved through global history, local history, or hybrid approaches.
- **Global History Buffer (GHB)**: A structure that maintains a record of the outcomes of recent branches to improve the accuracy of dynamic predictions.
- **Local History Buffer**: A mechanism that stores the behavior of individual branches to make predictions based on their unique patterns.
- **Reorder Buffer (ROB)**: A crucial component in out-of-order processors that allows instructions to complete in a different order than they were issued, helping to maintain the correct program order at the time of writing results back to the register file.
- **Instruction-Level Parallelism (ILP)**: The ability to execute multiple instructions simultaneously within a processor, which can be enhanced through effective branch prediction techniques.
- **Hardware Overhead**: Refers to the additional area, power, and complexity that branch prediction techniques introduce into a processor architecture, impacting design trade-offs.
- **Speculative Execution**: A performance optimization technique where the processor executes instructions before it is known whether they are needed, often based on branch predictions, to reduce idle cycles.
- **Performance Metrics**: Metrics such as prediction accuracy, misprediction rate, and throughput that evaluate the effectiveness of branch prediction strategies.
- **gem5 Simulator**: An open-source simulator used for computer architecture research, which can model various aspects of processors, including branch prediction techniques in RISC-V architectures. [2]

2.2 Classification

The table below represents the specific research on the respective branch prediction techniques. It is to be noted that the references are not exhaustive and are only indicative of the research conducted on the respective branch predictors. All the researfch cited in this paper has mentions, opinions, and Quantifiable data on, if not all, atleast 4 of the branch predictors.

Predictor	**References**
Local	*Nearly all*
Tournament	*Nearly all*
BiMODE	[3]
L-TAGE	[4]–[7]
TAGE-SC-L	[8]–[10]
Multi-Perspective Perceptron	[11]–[14]
Multi-Perspective Perceptron with TAGE	[11]–[14]

3 Related Research

A review of several dynamic BP systems, such as gshare, two-level BPs, Smith BP, and perceptron, is presented by Sparsh M. [15]. For the majority of the applications employed, it is evident that the perceptron BP has the highest precision.

The OoO cpu model in gem5 is modelled as follows. Its pipeline stages are fetch, decode, rename, and retirement. The instruction issue, dispatch, and execute stages are carried out out-of-order in an OoO pipeline (Power et al. [2]). The branch prediction unit is handled by the OoO pipeline's fetch step. The pipeline's decode stage handles unconditional branches, whose branch target is unavailable, and the execute stage finds the conditional branch mispredictions. The program counter (PC) is updated whenever a branch misprediction is detected, the pipeline is squashed as a whole, and the entry is removed from the branch target buffer (BTB).

A comparison of several BPs, such as the bimodal, gshare, YAGs, and meta-predictor, is presented by Kotha A [16]. The JPEG Encoder, G721 Speech Encoder, Mpeg Decoder, and Mpeg Encoder apps are used to assess the BPs' performance. For certain applications, YAGS Neo, a new BP, performs better than the others. The meta predictor presented in the research, which combines different predictor combinations, performs better than the others.

4 Branch Predictors

4.1 Local

The local branch predictor is a simple branch predictor that uses a table of 2-bit saturating counters to predict the outcome of a branch.

Modern processors have specialised mechanisms called local branch predictors that use the past performance of particular branch instructions to improve branch prediction accuracy.

The predictors employ a Local History Table (LHT) to save the results of branches that are taken and those that are not. Because every entry in the LHT relates to a particular branch instruction, the predictor can efficiently catch distinct patterns.

The LHT's output is sent into a Pattern History Table (PHT), which makes predictions about the branch's future occurrences using saturating counters. Local predictors can achieve good prediction accuracy with this architecture, especially for branches with unique behaviours. Though they typically use less storage than global predictors, they still have some overhead, and they can have trouble with branches that exhibit inconsistent patterns impacted by program input or state.

Because local branch predictors reduce misprediction-related execution pauses, they play a critical role in improving out-of-order processor efficiency. To increase overall prediction accuracy, they are frequently used in conjunction with global predictors or tournament predictors. [15]

4.2 L-TAGE

In order to attain more accuracy in forecasting branch outcomes, L-TAGE (Local TAGE) is an enhanced branch prediction technique that expands upon the ideas of both local and global prediction strategies. It tracks both local and global history at the same time by using a hybrid technique that involves keeping numerous tables. L-TAGE combines the advantages of global history tracking—which takes into account more extensive execution patterns—with local history tables, which may effectively record the unique behaviours of individual branches. When compared to typical local or global predictors alone, L-TAGE's accuracy is improved due to its ability to adapt to a wide range of branching behaviours according to this dual approach.

Using numerous Pattern History Tables (PHTs) to optimise prediction for different sorts of branches by utilising varying historical lengths is one of L-TAGE's primary characteristics. To make sure that each branch instruction uses the most pertinent previous data, the predictor uses a tagging method. Because of this, L-TAGE is able to manage complicated branch behaviours, such as correlated branches, in which the course of one branch may influence the course of other branches.

Modern out-of-order processors can incorporate L-TAGE because of its ability to maximise prediction accuracy and save hardware overhead. It successfully strikes a trade-off between performance and complexity, which is crucial in high-performance computing settings where each cycle matters.

To sum up, L-TAGE is a major breakthrough in branch prediction technology that offers a reliable way to improve processor performance through precise branch outcome prediction. Examine pertinent literature that covers its architecture and performance reviews for additional insights.

4.3 BiMODE

The Bi-Mode Predictor is a branch prediction method that improves prediction accuracy by utilising two different prediction modes. The two main parts of this strategy are a global predictor that takes into account the branches' larger execution context and a local predictor that monitors the history of individual branches. The Bi-Mode Predictor is especially useful in a variety of settings since it can adjust to different branching behaviour patterns by using both local and global history.

The local predictor in the Bi-Mode Predictor architecture usually uses a local history table (LHT) to store the historical results of certain branch instructions. In the meantime, the global predictor records the recent actions of several branches using a global history registry (GHR). The decision-making process of the Bi-Mode technique is crucial. Upon encountering a branch instruction, the predictor has the option to employ the local or global prediction, depending on which mode is anticipated to produce a more precise outcome for that particular branch.

The Bi-Mode Predictor's ability to manage the trade-offs between accuracy and hardware complexity is one of its main features. It can outperform conventional single-mode predictors, like solely local or purely global predictors, in terms of accuracy while keeping hardware overhead to a minimum by combining two modes of prediction. Because of this feature, the Bi-Mode Predictor is especially well-suited for out-of-order processors, where it's crucial to maintain a high instruction throughput.

Studies show that when compared to less complex prediction methods, Bi-Mode Predictors can dramatically lower branch misprediction rates, improving processor performance overall. The approach remains relevant in the design of contemporary high-performance computing systems due to its efficiency and versatility. You can read up on the architecture and performance assessments of the Bi-Mode Predictor in greater depth in the literature.

4.4 Tournament

A sophisticated branch prediction method called the Tournament Predictor was created to improve branch outcome predictions on contemporary CPUs. Through a competitive selection procedure, this strategy combines the capabilities of both local and global predictors. The Tournament Predictor's main concept is to use several predictors and dynamically select the best one based on how well it has performed in the past for a particular branch instruction.

This architecture combines a "tournament" selection process with one or more predictors, usually a local predictor and a global predictor. Every predictor keeps track of its own prediction algorithms and history tables. The Tournament Predictor evaluates each of its component predictors' predictions when it encounters a branch instruction. The final prediction is made by the predictor who has previously shown the highest accuracy for branches that are comparable to it.

The Tournament Predictor's versatility is one of its main benefits. The Tournament Predictor is particularly helpful in complex applications where the behaviour of branch instructions might vary greatly, as it can handle a wide range of branching behaviours by utilising numerous prediction algorithms and choosing the best-performing one. This dynamic technique maximises processor performance by reducing the penalty linked to incorrect predictions and increasing prediction accuracy.

Studies have indicated that the Tournament Predictor performs better than a lot of conventional prediction techniques, especially in settings with a lot of branch variability. It is an important tool in the design of high-performance computing systems and out-of-order processors because of its flexibility in responding to shifting patterns. Consult the literature on the Tournament Predictor's performance comparisons and implementation techniques for a more thorough grasp of its architecture and efficacy.

4.5 TAGE-SC-L

4.6 Multi-Perspective Perceptron

4.7 Multi-Perspective Perceptron with TAGE

5 Experimental Setup

The experiments were conducted using the gem5 simulator, which is a cycle-accurate simulator for computer architecture research. [2] The gem5 simulator was used to model an out-of-order processor with a RISC-V core, which is a common architecture for high-performance computing systems. The branch prediction techniques were implemented in the simulator to evaluate their performance in terms of prediction accuracy, misprediction rate, and throughput.

TABLE 1: Experimental Setup and Metrics

Component	Description
Simulator	gem5
Architecture	RISC-V core with out-of-order execution
Branch Prediction Techniques	Local, BiMode, Tournament, L-TAGE, Multi-Perspective Perceptron
Benchmarks	SPEC CPU benchmarks for validation
Metrics	• Prediction Accuracy • Misprediction Rate • Performance Improvement (IPC) • Execution Time • Hardware Overhead (Area and Power)
Analysis Tools	Statistical analysis tools for data interpretation

6 Workload

7 Experiment Results

8 Perspectives and Conclusion

References

[1] E. Sprangle and D. Carmean, "Increasing processor performance by implementing deeper pipelines," in *Proceedings 29th Annual International Symposium on Computer Architecture*, 2002, pp. 25–34.

[2] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, B. Beckmann, S. Bharadwaj, G. Black, G. Bloom, B. R. Bruce, D. R. Carvalho, J. Castrillon, L. Chen, N. Derumigny, S. Diestelhorst, W. Elsasser, C. Escuin, M. Fariborz, A. Farmahini-Farahani, P. Fotouhi, R. Gambord, J. Gandhi, D. Gope, T. Grass, A. Gutierrez, B. Hanindhito, A. Hansson, S. Haria, A. Harris, T. Hayes, A. Herrera, M. Horsnell, S. A. R. Jafri, R. Jagtap, H. Jang, R. Jeyapaul, T. M. Jones, M. Jung, S. Kannoth, H. Khaleghzadeh, Y. Kodama, T. Krishna, T. Marinelli, C. Menard, A. Mondelli, M. Moreto, T. Mück, O. Naji, K. Nathella, H. Nguyen, N. Nikoleris, L. E. Olson, M. Orr, B. Pham, P. Prieto, T. Reddy, A. Roelke, M. Samani, A. Sandberg, J. Setoain, B. Shingarov, M. D. Sinclair, T. Ta, R. Thakur, G. Travaglini, M. Upton, N. Vaish, I. Vougioukas, W. Wang, Z. Wang, N. Wehn, C. Weis, D. A. Wood, H. Yoon, and Éder F. Zulian, "The gem5 simulator: Version 20.0+," 2020. [Online]. Available: https://arxiv.org/abs/2007.03152

[3] S. Nain and P. Chaudhary, "Implementation and comparison of bi-modal dynamic branch prediction with static branch prediction schemes," *Int. j. inf. tecnol.*, vol. 13, no. 3, pp. 1145–1153, Jun. 2021. [Online]. Available: https://doi.org/10.1007/s41870-021-00631-z

[4] K. Matsui, M. Ashraful Islam, and K. Kise, "An Efficient Implementation of a TAGE Branch Predictor for Soft Processors on FPGA," in *2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, Oct. 2019, pp. 108–115. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8906770

[5] A. Seznec, "Storage free confidence estimation for the TAGE branch predictor," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, Feb. 2011, pp. 443–454, iSSN: 2378-203X. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5749750

[6] ——, "A 64 Kbytes ISL-TAGE branch predictor."

[7] ——, "A new case for the TAGE branch predictor," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: Association for Computing Machinery, Dec. 2011, pp. 117–127. [Online]. Available: https://dl.acm.org/doi/10.1145/2155620.2155635

[8] C.-K. Lin and S. J. Tarsa, "Branch Prediction Is Not a Solved Problem: Measurements, Opportunities, and Future Directions," Jun. 2019, arXiv:1906.08170 version: 1. [Online]. Available: http://arxiv.org/abs/1906.08170

[9] A. Seznec, "TAGE-SC-L Branch Predictors Again," 2016. [Online]. Available: https://inria.hal.science/hal-01354253

[10] ——, "TAGE-SC-L Branch Predictors," Jun. 2014. [Online]. Available: https://inria.hal.science/hal-01086920

[11] D. Jimenez and C. Lin, "Dynamic branch prediction with perceptrons," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer*

Architecture, Jan. 2001, pp. 197–206, iSSN: 1530-0897. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/903263

[12] D. Tarjan and K. Skadron, "Merging path and gshare indexing in perceptron branch prediction," *ACM Trans. Archit. Code Optim.*, vol. 2, no. 3, pp. 280–300, Sep. 2005. [Online]. Available: https://dl.acm.org/doi/10.1145/1089008.1089011

[13] E. Garza, S. Mirbagher-Ajorpaz, T. A. Khan, and D. A. Jiménez, "Bit-level perceptron prediction for indirect branches," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, Jun. 2019, pp. 27–38. [Online]. Available: https://dl.acm.org/doi/10.1145/3307650.3322217

[14] R. Joseph, "A Survey of Deep Learning Techniques for Dynamic Branch Prediction," Dec. 2021, arXiv:2112.14911. [Online]. Available: http://arxiv.org/abs/2112.14911

[15] S. Mittal, "A Survey of Techniques for Dynamic Branch Prediction," Apr. 2018, arXiv:1804.00261. [Online]. Available: http://arxiv.org/abs/1804.00261

[16] A. Kotha, "A Comparative Study of Branch Predictors."

[17] "A Survey of Techniques for Dynamic Branch Prediction." [Online]. Available: https://ar5iv.labs.arxiv.org/html/1804.00261

[18] D. D. Penney and L. Chen, "A Survey of Machine Learning Applied to Computer Architecture Design," Sep. 2019, arXiv:1909.12373. [Online]. Available: http://arxiv.org/abs/1909.12373

[19] "Documentation page not found - Read the Docs." [Online]. Available: https://marss-riscv-docs.readthedocs.io/en/latest/sections/branch-pred

[20] "3. Branch Prediction Unit — MARSS-RISCV 4.1a documentation." [Online]. Available: https://marss-riscv-docs.readthedocs.io/en/latest/sections/branch-pred.html

[21] Y. He and X. Chen, "Survey and Comparison of Pipeline of Some RISC and CISC System Architectures," in *2023 8th International Conference on Computer and Communication Systems (ICCCS)*, Apr. 2023, pp. 785–790. [Online]. Available: https://ieeexplore.ieee.org/document/10150975

[22] Emma and Davidson, "Characterization of Branch and Data Dependencies in Programs for Evaluating Pipeline Performance," *IEEE Transactions on Computers*, vol. C-36, no. 7, pp. 859–875, Jul. 1987, conference Name: IEEE Transactions on Computers. [Online]. Available: https://ieeexplore.ieee.org/document/1676981

[23] J. Hoogerbrugge, "Dynamic branch prediction for a VLIW processor," in *Proceedings 2000 International Conference on Parallel Architectures and Compilation Techniques (Cat. No.PR00622)*, Oct. 2000, pp. 207–214, iSSN: 1089-795X. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/888345

[24] M. Evers, P.-Y. Chang, and Y. N. Patt, "Using hybrid branch predictors to improve branch prediction accuracy in the presence of context switches," *SIGARCH Comput. Archit. News*, vol. 24, no. 2, pp. 3–11, May 1996. [Online]. Available: https://dl.acm.org/doi/10.1145/232974.232975

[25] B. Calder and D. Grunwald, "Fast and accurate instruction fetch and branch prediction," in *Proceedings of the 21st annual international symposium on Computer architecture*, ser. ISCA '94. Washington, DC, USA: IEEE Computer Society Press, Apr. 1994, pp. 2–11. [Online]. Available: https://dl.acm.org/doi/10.1145/191995.192011

[26] M. Mohammadi, S. Han, T. M. Aamodt, and W. J. Dally, "On-Demand Dynamic Branch Prediction," *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 50–53, Jan. 2015, conference Name: IEEE Computer Architecture Letters. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6834760?casa_token=lHuqEKUMvOkAAAAA

[27] A. K. Uht, "Disjoint Eager Execution: what it is / what it is not," *SIGARCH Comput. Archit. News*, vol. 30, no. 1, pp. 12–14, Mar. 2002. [Online]. Available: https://dl.acm.org/doi/10.1145/511120.511124

[28] D. Grunwald, D. Lindsay, and B. Zorn, "Static methods in hybrid branch prediction," in *Proceedings. 1998 International Conference on Parallel Architectures and Compilation Techniques (Cat. No.98EX192)*, Oct. 1998, pp. 222–229, iSSN: 1089-795X. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/727254

[29] T.-Y. Yeh and Y. N. Patt, "Two-level adaptive training branch prediction," in *Proceedings of the 24th annual international symposium on Microarchitecture*, ser. MICRO 24. New York, NY, USA: Association for Computing Machinery, Sep. 1991, pp. 51–61. [Online]. Available: https://dl.acm.org/doi/10.1145/123465.123475

[30] ——, "Alternative implementations of two-level adaptive branch prediction," *SIGARCH Comput. Archit. News*, vol. 20, no. 2, pp. 124–134, Apr. 1992. [Online]. Available: https://dl.acm.org/doi/10.1145/146628.139709

[31] C. Egan, G. Steven, P. Quick, R. Anguera, F. Steven, and L. Vintan, "Two-level branch prediction using neural networks," *Journal of Systems Architecture*, vol. 49, no. 12, pp. 557–570, Dec. 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138376210300095X

[32] T.-Y. Yeh and Y. N. Patt, "A comparison of dynamic branch predictors that use two levels of branch history," in *Proceedings of the 20th annual international symposium on computer architecture*, ser. ISCA '93. New York, NY, USA: Association for Computing Machinery, May 1993, pp. 257–266. [Online]. Available: https://dl.acm.org/doi/10.1145/165123.165161

[33] "Efficient integration of bimodal branch prediction and pipeline analysis | IEEE Conference Publication | IEEE Xplore." [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1541054

[34] A. Choudhury, S. V. Siddamal, and J. Mallidue, "An optimized RISC-V processor with five stage pipelining using Tournament Branch Predictor for efficient performance," in *2022 International*

Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER), Oct. 2022, pp. 57–60. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9974891

[35] N. M. Dang, H. X. Cao, and L. Tran, "BATAGE-BFNP: A High-Performance Hybrid Branch Predictor with Data-Dependent Branches Speculative Pre-execution for RISC-V Processors," *Arab J Sci Eng*, vol. 48, no. 8, pp. 10 299–10 312, Aug. 2023. [Online]. Available: https://doi.org/10.1007/s13369-022-07593-9

[36] "Branch Prediction — RISCV-BOOM documentation." [Online]. Available: https://docs.boom-core.org/en/latest/sections/branch-prediction/

[37] K. Matsui, M. Ashraful Islam, and K. Kise, "An Efficient Implementation of a TAGE Branch Predictor for Soft Processors on FPGA," in *2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, Oct. 2019, pp. 108–115. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8906770

[38] D. A. Jiménez and C. Lin, "Neural methods for dynamic branch prediction," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 369–397, Nov. 2002. [Online]. Available: https://dl.acm.org/doi/10.1145/571637.571639

[39] A. Butko, A. Gamatié, G. Sassatelli, L. Torres, and M. Robert, "Design Exploration for next Generation High-Performance Manycore On-chip Systems: Application to big.LITTLE Architectures," in *2015 IEEE Computer Society Annual Symposium on VLSI*, Jul. 2015, pp. 551–556, iSSN: 2159-3477. [Online]. Available: https://ieeexplore.ieee.org/document/7309629/?arnumber=7309629