

A PROJECT REPORT

ON

**“Dynamic Branch Predictor Using Perceptron Learning Algorithm”**

**Submitted in partial fulfillment of the requirement for the Course CSE360**

**(Course title:** Computer Architecture**, Section:**01**)**

Of

BACHELOR OF SCIENCE

IN

COMPUTER SCIENCE AND ENGINEERING

**Submitted By: Group-5**

1. Shanta islam (2022-1-60-288)

2. Urmi kirtonia (2022-1-60-184)

3. Md Sifatullah Sheikh (2022-1-60-029)

**Submission date: 21 January, 2025**

**Under the supervision of**

**Dr. Md. Nawab Yousuf Ali**

(**Professor**, Department of Computer Science and Engineering)

**Title: Implement a dynamic branch predictor using a perceptron learning algorithm in C**

**Introduction**

Branch prediction is a methodology employed in computing processors to predict the flow of a program even before the judgment of a particular decision is made. As for instance, upon reaching conditional branches that cause programs to proliferate in between two branches, the CPU typically predicts one and continues to enhance performance while waiting for the condition to be evaluated. If it has been correctly predicted then the processor saves the time it would have otherwise spent waiting, otherwise, it backs out and corrects the result of its work.  
  
Dynamic branch prediction makes use of algorithms to further enhance those guesses through learning from previously taken outcomes on the branches. Thus, in this project, we have set up a dynamic branch predictor on top of the perceptron learning algorithm. Simply put, the perceptron is a simple machine learning model that alters its predictions according to past errors. This allows the predictor to catch up on the emergence of patterns in the branch behavior that improves its accuracy with time. This project, thus, aims to demonstrate how this method is going to operate and how effective it will be through simulations.

**Objectives**

* To understand the perceptron-based branch prediction mechanism.
* To design and implement a dynamic branch predictor capable of learning branch behavior.
* To evaluate its performance through simulation with varying branch patterns.

**Theory**

Branch prediction has become an essential ingredient in modern processors for reducing the number of stalls based on control hazards. In this project, the perceptron learning algorithm is a simple machine learning model for branch prediction. A perceptron makes a prediction based on the sign of a weighted sum of inputs-the global branch history. It iteratively adjusts its weights to minimize prediction errors.

**Perceptron Learning Algorithm**  
The perceptron is a type of linear classifier used in supervised learning. It maps an input x to an output y using a set of weights w:

y= sign (w ⋅ x)

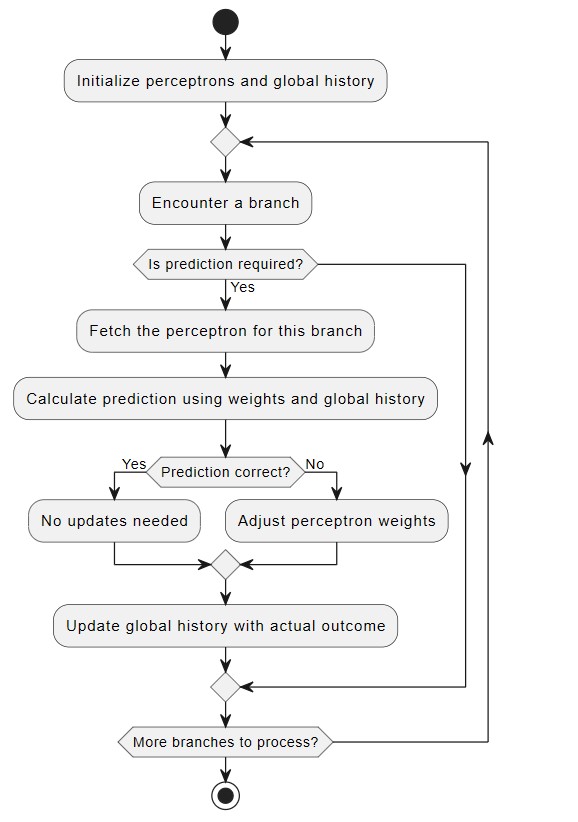
In branch prediction:

* **Inputs**: A global history of past branch outcomes (+1 for taken, −1 for not taken).
* **Output**: Predicted branch outcome (+1 for taken, −1 for not taken).
* **Training**: If the prediction is incorrect or close to a predefined threshold, the weights are updated using: wi ← wi + Δwi, where Δwi = yactual ⋅ xi​​

**Advantages of Perceptron-based Prediction**

* Handles long global histories effectively.
* Adapts to complex patterns in branch behavior.

**Diagram**

****

**Implementation**

1. **Functional Modules**

* Initialization Module (initialize):  
  This module sets up the environment for branch prediction by initializing all perceptron weights and global history to zero.It makes sure that all perceptrons start unbiased, untrained weights, and the global history is empty for an accurate simulation.
* Prediction Module (predict):  
  Computes the dot product of the perceptron's weights and the global history.  
  Includes the bias weight in its calculation.If the result is non-negative, the prediction is "Taken"; otherwise, it is "Not Taken." This gives a very fast and efficient way of estimating branch behavior based on past history.
* Training Module (train):  
  Updates the weights of the perceptron in case of an incorrect or weakly confident prediction. It has a tunable threshold to decide the confidence level for predictions.Uses the actual outcome of the branch to modify both bias and history-dependent weights to make sure the perceptron learns from its mistakes.
* Global History Management (update\_history):

It keeps track of a shift register of recent branch outcomes  
It shifts the older outcome and appends the newest one at the head.  
This is to make the past behavior represented consistently for both prediction and training.

* Simulation and Control (main):  
    
  This is an interactive simulation interface.  
  The user inputs the number of perceptrons, history length, branch ids, and outcome to see predictions and weight updates.  
  Returns prediction and updated history after every simulation of branches.

1. **Hierarchical Relationships**  
   The main function is the orchestrator of this whole process, and it calls initialize at the very beginning. Main calls predict to identify the predicted result, train to update the perceptron when necessary, and update\_history to keep the track of history for each branch. The perceptrons are the heart of the system and have been duplicated between the predictor and the training parts.
2. **System Requirements**

* **Programming Language**: C
* **Operating System**: Linux/Windows
* **Compiler**: GCC or any C compiler supporting ANSI C

1. **Built-In Documentation**  
   Comments stating what a function does and critical logic blocks are provided in the code.

**Debugging-Test-run**

**Testing Process**

* A fixed history length and number of perceptions was given.
* Multiple branches with random branch IDs and outcomes were simulated.
* Predictions against actual outcomes were verified.
* Weight updates and history shifts were manually traced for correctness.

**Debugging**

* Initialization of perceptions and history were verified.
* Edge cases, such as invalid branch IDs and boundary outcomes (+1 and −1), were tested.
* Predictions were checked to be within expected thresholds.

**Results Analysis**  
The perceptron predictor effectively learned from its past outcomes to improve its prediction accuracy.

**Complexity Analysis**

* Time Complexity: O (n ⋅ h), where n is the number of branches and h is the history length.
* Space Complexity: O (p ⋅ h), where p is the number of perceptions.

**Robustness**

* Little accurate predictions for patterns longer than those handled by traditional 2-bit predictors.
* Handles noise and erratic patterns reasonably well.

**Conclusion and Future Improvements**

**Conclusion**  
  
The perceptron-based branch predictor showed better performance in complex branch patterns. By using a machine learning-like approach, it outperforms traditional predictors regarding adaptability and accuracy.

**Future Improvements**

* Implement hybrid predictors that will combine the perceptions with the simpler models of prediction.
* Extend the design to handle multi-core architecture with shared history.
* Optimize memory for perceptions in large-scale simulations.
* Evaluate the performance using real-world branch traces.

**Bibliography**

* Smith, J.E. "A Study of Branch Prediction Strategies." IEEE Micro, 1998.
* Daniel A. Jiménez, Calvin Lin. "Dynamic Branch Prediction with Perceptions." HPCA, 2001.
* Hennessy, J.L., & Patterson, D.A. Computer Architecture: A Quantitative Approach. Morgan Kaufmann, 2020.
* Online Resources:
  + [GeeksforGeeks: Branch Prediction](https://www.geeksforgeeks.org)
  + [Wikipedia: Perceptron Learning Algorithm](https://en.wikipedia.org/wiki/Perceptron)