Branch Prediction Analysis

Introduction

Branch prediction is a fundamental concept in computer architecture and processor design aimed at improving the performance of modern processors. In essence, it involves predicting the outcome of conditional branch instructions before their actual resolution during program execution. Conditional branch instructions determine the flow of control in a program, leading to different paths of execution based on certain conditions or states.

Advantages of Branch Prediction:

★ **Improved Instruction Flow:**
- Predicting branch outcomes enables speculative execution of instructions along the predicted path.
- Reduces pipeline stalls and enhances instruction throughput.

★ **Enhanced Performance:**
- Effective branch prediction optimizes the utilization of processor resources.
- Results in overall performance gains across various computational tasks.

★ **Reduced Latency:**
- Accurate branch prediction minimizes the latency from mispredicted branches.
- Mitigates pipeline flushes and restarts, enhancing program execution efficiency.

Disadvantages of Branch Prediction:

★ **Prediction Errors:**
- Despite advancements, prediction errors can occur, leading to wasted computation cycles.
- May result in potential degradation of system performance.

★ **Complexity Overhead:**
- Sophisticated branch prediction mechanisms add complexity to processor design.
- This complexity may increase manufacturing costs and power consumption.

Various Branch Predictors

We have covered the four basic branch prediction strategies :

- **Always Taken:**

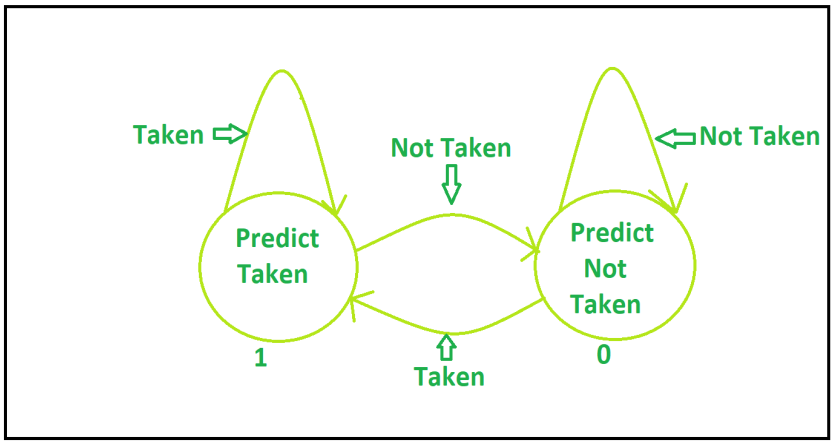
  - Assumes all conditional branches will be taken.
  - Effective in loops or conditional statements where true condition is common.
  - May lead to suboptimal performance when branches are infrequently taken.
  - Can cause unnecessary pipeline stalls and speculative execution of incorrect instructions.

- **Always Not Taken:**

  - Predicts that conditional branches will never be taken.
  - Suitable for scenarios with predominantly not taken branches.
  - Suffers from similar limitations as Always Taken strategy when applied across all branches.

- **1-Bit Branch Predictor:**

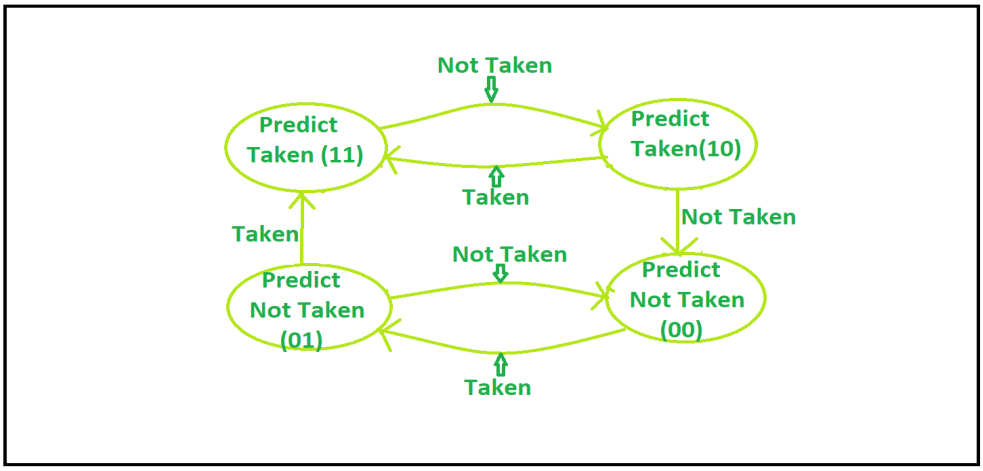
  - Maintains a single-bit state for each branch.
  - State predicts either taken or not taken.



  - Updates prediction based on actual outcome of the branch.
  - Simple but can exhibit poor performance in dynamic branch behavior scenarios.

- **2-Bit Branch Predictor:**

  - Extends 1-bit predictor using a two-bit saturating counter.
  - Tracks history of branch outcomes for more nuanced predictions.



  - Better captures patterns of branch behavior over time.
  - Exhibits greater resilience to short-term fluctuations, leading to improved prediction accuracy compared to 1-bit predictor.

Branch Buffer Table

The branch buffer table in the code stores details about branch instructions during program execution. It links Program Counter (PC) values with target addresses and tracks prediction history.

1. Program Counter (PC):

- The PC value represents the address of the current instruction being executed.
- It serves as the key in the branch buffer table to uniquely identify each branch instruction.

2. Target Address:

- The target address is the destination address of the branch instruction.
- It is typically the address to which the control flow will be transferred upon successful prediction of the branch outcome.

3. Prediction History:

- The prediction history provides information about the prediction outcome of the branch instruction.
- It indicates whether the branch was predicted as taken ('T') or not taken ('N').

The branch buffer table helps track branch instructions and their predictions during program execution. It stores information about branch outcomes and target addresses, enabling analysis of prediction accuracy and behavior of branch predictors. This helps evaluate prediction strategies and improve processor performance.

Results

The accuracy of each branch predictor on different test cases is as follows :

- **`Bubble Sort:`**
  - Always Taken Predictor: `41.4791`%
  - Always Not Taken Predictor: `58.5209`%
  - 1 Bit Predictor: `97.1092`%
  - 2 Bit Predictor: `98.3933`%

*

- **`Fact_Test_Lab:`**
  - Always Taken Predictor: `63.7374`%
  - Always Not Taken Predictor: `36.2626`%
  - 1 Bit Predictor: `88.3983`%
  - 2 Bit Predictor: `90.1616`%

*

- **`qsort_test_Lab`**:
  - Always Taken Predictor: `67.9452`%
  - Always Not Taken Predictor: `32.0548`%
  - 1 Bit Predictor: `93.7701`%
  - 2 Bit Predictor: `94.7281`%

- **`Recursing_test_Lab`**:
    - Always Taken Predictor: `72.0708`%
    - Always Not Taken Predictor: `27.9292`%
    - 1 Bit Predictor: `96.2123`%
    - 2 Bit Predictor: `96.6429`%

*

- **`recursion_test_Lab`**:
    - Always Taken Predictor: `72.0988`%
    - Always Not Taken Predictor: `27.9012`%
    - 1 Bit Predictor: `59.0024`%
    - 2 Bit Predictor: `60.5109`%

*

- **`sqrt_test_Lab`**:
    - Always Taken Predictor: `56.9508`%
    - Always Not Taken Predictor: `43.0492`%
    - 1 Bit Predictor: `95.4861`%
    - 2 Bit Predictor: `96.444`%

*

- **`wikisort_test_Lab`**:
    - Always Taken Predictor: `51.8256`%
    - Always Not Taken Predictor: `48.1744`%
    - 1 Bit Predictor: `95.1311`%
    - 2 Bit Predictor: `96.3269`%

Links (References- Branch Buffer Tables)

- Bubble_Sort_Branch
- Fact_Test_Lab
- qsort_test_Lab
- Recursing_test_Lab
- recursion_test_Lab
- sqrt_test_Lab
- wikisort_test_Lab

Conclusion

- The Always Not Taken Predictor shows the lowest average accuracy among the predictors.
- The 2 Bit Predictor consistently outperforms the 1 Bit Predictor across all test cases.
- Branch prediction significantly impacts program execution efficiency and performance.
- Additionally, the choice of predictor can vary depending on the characteristics of the program being executed.

Collaborators:

★ Ahire Vrushank (2022csb1002@iitrpr.ac.in)
★ Patel Ayush (2022csb1101@iitrpr.ac.in)
★ Pranav Menon (2022csb1329@iitrpr.ac.in)