**Branch Predictor Simulation**

This project implements a branch predictor simulation using two prediction models: Global Predictor (GAp) and Per-Address Predictor (PAp). The implementation focuses on accurately predicting branch outcomes while evaluating the trade-offs between prediction accuracy and hardware cost.

**1.Data Structures**

**Branch History Table (BHT):**

* Implemented using an unsigned integer for the Branch History Register (BHR).

**Pattern History Table (PHT):**

* Implemented as an array of unsigned characters (unsigned char pht[]) with 2-bit saturating counters.

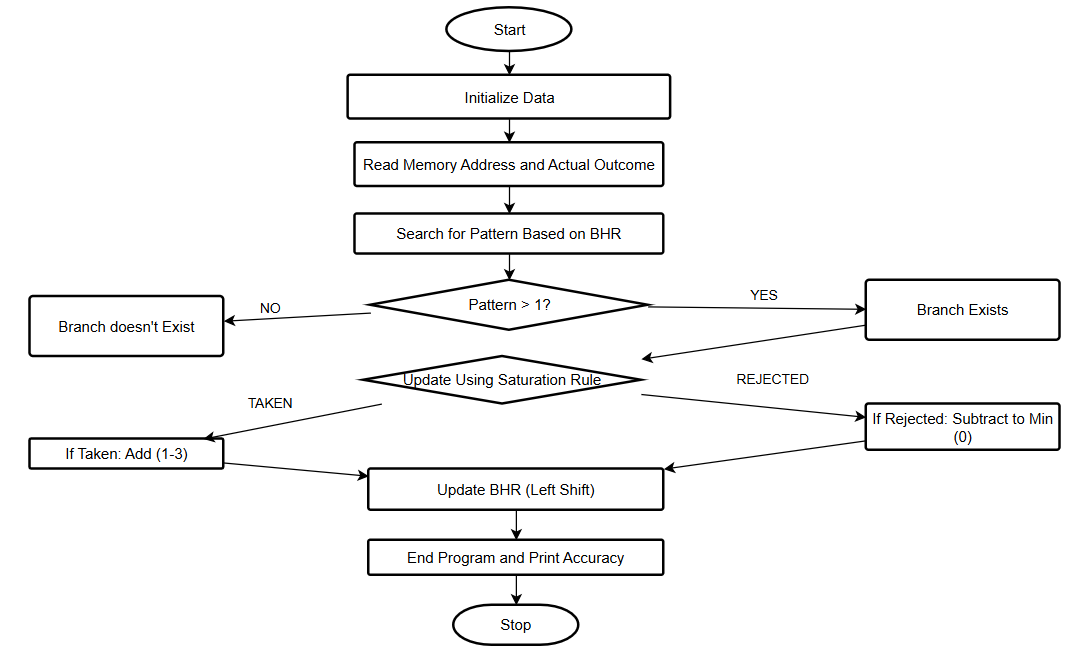
The simulation uses a Branch History Register (BHR) and a Pattern History Table (PHT) to store branch prediction data. The BHR is implemented as an unsigned integer, while the PHT is an array of 2-bit saturating counters. In GAp, a single global BHR and PHT are shared across all branches. For PAp, each branch maintains its own BHR and PHT, allowing for more accurate predictions at the cost of higher memory usage.

**2. Algorithm**

**Branch Prediction:**

1. **Input:** Branch Address and Branch Outcome.
2. **Prediction Process:**
   * For GAp:
     + Use a global BHR to index into the PHT.
     + Predict "Taken" if the PHT value ≥ 2, else "Not Taken."
   * For PAp:
     + Use a per-address BHR (based on the branch address) to index into a PHT for that specific branch.
     + Predict "Taken" if the PHT value ≥ 2, else "Not Taken."
3. **Update Process:**
   * Update the BHR by shifting in the actual outcome.
   * Update the PHT entry based on the actual outcome:
     + Increment for "Taken" (up to a maximum of 3)
     + Decrement for "Not Taken" (down to a minimum of 0)

**Flowchart:**

****

**3.Source Code**

The source code for the branch predictors is written in C and stored in the file branch\_predictor.c. It contains implementations for both GAp and PAp predictors. The code includes functions for initializing the predictors, predicting branch outcomes, and updating the predictors based on the actual outcomes. The program processes the trace file gccSmall.trace and outputs the results in two files: GApOutput.txt for the global predictor and PApOutput.txt for the per-address predictor.

**4.Compilation**

The project is implemented in C, with the main source code file named branch\_predictor.c. The program is compiled using the GCC (GNU Compiler Collection) with the command:

gcc branch\_predictor.c -o branch\_predictor

Ensure that the trace file gccSmall.trace is in the same directory as the source code before running the program.

**5.Installation**

To install the project, first ensure that GCC or MinGW is installed on your system. Download the files branch\_predictor.c (source code) and gccSmall.trace (branch trace file). Place these files in the same directory. No additional dependencies are required for the program to run.

**6.How to Run the Program**

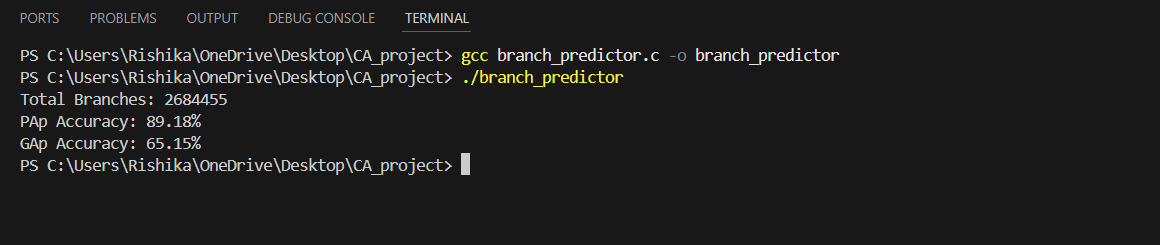
After compiling the program using the GCC command, run the executable with the following command:

./branch\_predictor

The program reads the branch trace file, processes it, and generates output files containing the prediction results for both GAp and PAp predictors.

**7.Results**

The program processes a total of 2,684,455 branch instructions. The prediction accuracy for the GAp predictor is 65.15%, while the PAp predictor achieves a much higher accuracy of 89.18%. These results highlight the trade-off between hardware efficiency and prediction performance.



**8.Hardware Cost**

The hardware cost is calculated based on the memory requirements of the BHT and PHT. The GAp predictor uses a single BHT (8 bits) and a global PHT with 256 entries (each 2 bits). This results in a total memory cost of 520 bits.

For the PAp predictor, each branch has its own BHT (8 bits) and its own PHT with 256 entries. With 1024 branches, the total memory requirement for the PAp predictor is significantly higher, at 532,480 bits. This higher memory usage is the cost of achieving greater prediction accuracy.

**1. Global Predictor (GAp):**

* **BHR (Branch History Register):**
  + Size = 8 bits
* **PHT (Pattern History Table):**
  + Entries = 256 (2^BHR\_SIZE, i.e., 2^8)
  + Each entry = 2 bits (for the 2-bit saturating counter)
  + Total size = 256 × 2 = **512 bits**
* **Total Memory for GAp:**

BHR size + PHT size = 8 + 512 = 520 bits

**2. Per-Address Predictor (PAp):**

* **BHR (Branch History Register):**
  + Size per branch = 8 bits
  + Number of branches = 1024
  + Total size = 8 × 1024 = **8192 bits**
* **PHT (Pattern History Table):**
  + Entries per branch = 256 (2^BHR\_SIZE)
  + Each entry = 2 bits (for the 2-bit saturating counter)
  + Total size per branch = 256 × 2 = 512 bits
  + Total size for all branches = 512 × 1024 = **524288 bits**
* **Total Memory for PAp:**

BHR size + PHT size = 8192 + 524288 = 532480 bits

**9.Analysis**

The results demonstrate the trade-offs between accuracy and hardware efficiency. The GAp predictor is more memory-efficient, requiring only 520 bits of storage, but its prediction accuracy is limited by interference between branches due to the shared BHT and PHT. On the other hand, the PAp predictor achieves much higher accuracy by using separate BHTs and PHTs for each branch, but it comes with a significant increase in memory requirements (532,480 bits).

For systems with limited resources, the GAp predictor may be a better choice. However, for performance-critical applications where accuracy is paramount, the PAp predictor is more suitable.

**10.Output Files**

The results of the simulation are saved in GApOutput.txt and PApOutput.txt. Each file contains four columns:

1. **Branch Address**: The address of the branch instruction.
2. **Branch Outcome**: The actual branch outcome (1 = Taken, 0 = Rejected)
3. **Your Prediction**: The predicted branch outcome (1 = Taken, 0 = Rejected)
4. **Correct/Miss**: Whether the prediction was correct (1 = Correct, 0 = Miss)

This project demonstrates the implementation, evaluation, and trade-offs of branch predictors, providing insights into their accuracy and hardware efficiency.