# CMPEN 431: Programming Project 2

# Branch Predictor Simulator

## Overview

The Branch Predictor Simulator is a Python-based simulation tool to evaluate the performance of different branch prediction algorithms. This document will guide you through the steps needed to run the simulator, generate branch traces, and understand how to implement each branch predictor. This simulator helps to gain insights into branch prediction mechanisms used in modern computer architecture, suitable for educational purposes.

## Running the Simulator

The process of running the simulator involves the following steps:

### 1. Generating the Branch Trace

The `branch_trace_generator.py` script generates synthetic branch traces that are used by the simulator to evaluate each branch predictor. To generate a branch trace, you can use the command below:

```
python branch_trace_generator.py --trace <trace_file.csv> --branches
<number_of_branches> --seed <random_seed>
```

- `--trace` (optional): path to the branch trace file (default to branch_trace.csv)
- `--branches` (optional): Specifies the number of branches to generate. Default is 10,000.
- `--seed` (compulsory): Specifies the random seed as your PSU ID for reproducibility.

The generated trace is stored in a file called `branch_trace.csv` and contains two columns:

- **BranchAddress**: The address of the branch instruction.
- **Outcome**: The actual outcome (taken or not taken).

### 2. Running the Branch Predictor Simulator

Once the trace file has been generated, you can run the main simulator using the `main_simulator.py` script.

```
python branch_simulator.py --trace <trace_file.csv> --x <interval_length> --
fast
```

- `--trace` (optional): path to the branch trace file (default to branch_trace.csv)

- `--x` (optional): Specifies the number of branches for calculating interval-based accuracy (default is 10).
- `--fast` (optional): Skips the 2-second pause between intervals, making the simulation run faster.

The simulator reads the `branch_trace.csv` (unless a different name was provided) file and runs each of the implemented branch predictors, providing cumulative accuracy statistics during and after the simulation.

## Logs and Output Data

### Real-Time Statistics

The simulator logs real-time statistics in a file named `realtime_stats.txt`. This file contains cumulative accuracy information for each branch predictor during the simulation, formatted as follows:

```
Predictor, Branches Processed, Cumulative Accuracy (%)
```

### Predictor-Specific Logs

Each predictor generates a detailed log of predictions during the simulation. These logs are stored in the `logs` directory, with one file per predictor, e.g., `logs/One_Bit_log.txt`. Each file contains information in the format:

```
Branch: <branch_number>, Correct: <0_or_1>
```

### Branch History Table (BHT) Logs

The simulator also saves the state of the Branch History Table (BHT) for applicable predictors in the `bht_logs` directory. Each predictor's BHT log provides insight into the internal state of the predictor after the simulation.

## Analysis

You can inspect the generated log files to plot the data using external tools like Python, MATLAB, or spreadsheet software for more detailed analysis.

## Branch Predictors Explained

### 1. Static Predictors

- **Static Taken / Not Taken**: These predictors always predict the branch will be taken (or not taken). No learning occurs.

### 2. One-Bit Branch Predictor

- Maintains a **Branch History Table (BHT)** that stores a single bit for each branch address. This bit represents whether the branch was previously taken or not. The predictor simply repeats the last outcome.

### 3. Two-Bit Branch Predictor

- Utilizes a **two-bit saturating counter** for each branch address. The counter ranges from `00` (strongly not taken) to `11` (strongly taken). Prediction is considered taken if the counter value is `10` or higher. The counter should be incremented or decremented based on the actual outcome.

### 4. Bimodal Branch Predictor

- Uses a fixed-size **BHT** indexed by the lower bits of the branch address. Each entry in the BHT has a two-bit counter similar to the Two-Bit Predictor. The prediction accuracy is improved by reducing aliasing in the prediction table.

### 5. GShare Branch Predictor

- Employs **global branch history** to determine prediction outcomes. It should XOR the global history register with the branch address to generate an index into the BHT. This approach helps to correlate predictions across different branches.

### 6. Hybrid Branch Predictor

- Combines the **GShare** and **Bimodal** predictors. A **choice table** determines which predictor (GShare or Bimodal) should be trusted for each branch. The choice table is updated to improve the accuracy of prediction based on which predictor was correct for each branch.

## Anticipated Steps

These steps can serve as a high-level guideline to aid you during the project:

1. Run `branch_trace_generator.py` script to generate the trace file given your PSU ID as the seed parameter.
2. Implement branch predictors: **static, 1-bit, 2-bit, bimodal, gshare, and hybrid** branch predictors
3. Run `branch_simulator.py` script to test out the different branch predictors using the trace file generated in step 1.
4. Complete the report.

## Submission Requirements

1. Project Report
2. Code Implementations of the branch predictors mentioned above.

## Code Implementations

You need to implement various branch predictors in `branch_predictors.py` file. For each predictor, you need to implement three functions

- `__init__(self)`: constructor for the predictor. You can utilize this function to initialize the predictor
- `predict(self, address)`: given an address return a prediction (either 0 for not-taken, or 1 for taken)
- `update(self, address, actual_outcome)`: this function is utilized to update the state of the predictor with the actual outcome of the branch instruction.

## Report Minimum Requirements

1. Describe in 100 words or less how the provided simulator enable testing various branch predictions.
2. Table with the overall accuracy of each predictor of the generated trace file.
3. Plots that show the branch predictor accuracy over time. The x-axis should be the Number of Branches and the y-axis should be the Prediction Accuracy (%).
4. Elaborate on the results of the predictors and why some predictors performed better than others.

## Directory Structure

The simulator organizes its files and logs as follows:

```
.
├── branch_predictors.py         # Branch predictor implementations
├── branch_trace_generator.py    # Generates branch trace files
├── branch_simulator.py          # Main branch predictor simulator
├── branch_trace.csv             # Generated branch trace file
├── logs/                        # Logs for each branch predictor
│   ├── One_Bit_log.txt          # Detailed logs for the One-Bit predictor
│   └── ...
├── bht_logs/                    # Logs for BHT states
│   ├── GShare_bht.txt           # GShare BHT state
│   └── ...
└── realtime_stats.txt           # Real-time statistics log
```

## System Requirements

- Python 3.x
- `tabulate` for tabular progress display

To install the dependencies, run:

```
pip install tabulate
```