

**Project – IV****“A Trace Driven Branch Predictor Simulator”***Chandini Shetty**Kiran Kumar Gangadevi***1. Introduction**

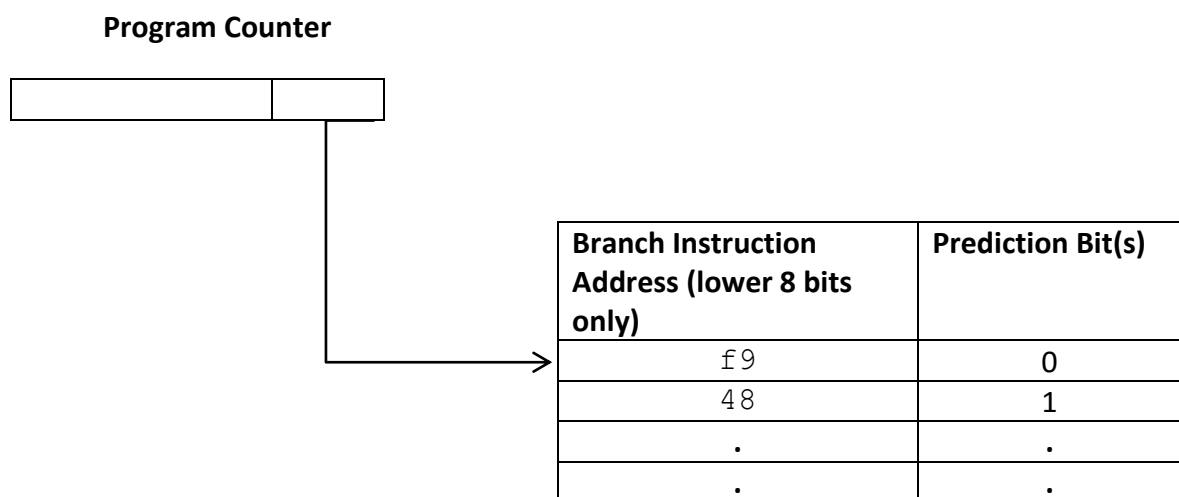
Branch prediction is a technique used in some processors with instruction prefetch to guess whether a conditional branch will be taken or not and prefetch code from the appropriate location. When a branch instruction is executed, its address and that of the next instruction executed (the chosen destination of the branch) are stored in the Branch Target Buffer. This information is used to predict which way the instruction will branch the next time it is executed so that instruction prefetch can continue. When the prediction is correct (and it is over 90% of the time), executing a branch does not cause a pipeline break. Branch Prediction can be categorized as Static Branch Prediction and dynamic branch prediction.

**Static Branch Prediction:** In static branch prediction, all the decisions are made at compile time.

**Dynamic Branch Prediction:** In dynamic branch prediction, all the decisions are made during the execution.

**Branch Prediction Buffer (Branch History Table):**

It is a buffer that stores some bit(s) to indicate whether a branch is taken or not recently. These entries in the buffer can be indexed by lower bits of address of branch instruction.



*Figure. 1 Branch Prediction Buffer (Branch History Table)*

If a branch is mispredicted, the prediction bit is inverted and stored back. If a single bit is used for prediction, chances are that the predictor will likely mispredict twice when then branch is almost always

taken. This problem could be solved by using a 2-bit predictor. This scheme could also be extended to design a n-bit predictor where we could keep track of an n-bit saturating counter for each branch.

## 2. Implementation

As a part of this project, we have implemented two types of Dynamic Branch Prediction Scheme

### Type1: Saturating Counter based Branch prediction

### Type2: Two Level Correlating Predictor with configurable Global History Buffer size

Below is a screenshot of the config.properties file where user can tweak the parameters

[illegible]

### Type 1: High-level Overview of the Predictor Implementation Logic

**Type 1:** The Idea here is to maintain a Buffer cache indexed by the lower portions of the PC address. And for each entry maintain a 1-bit, 2-bit or n-bit Saturating counter.

To implement this in Java code, we first parse the trace file and store the PC counter and Branch direction for all the branch instruction in a Java List. Then we build a Java Map- $(\text{Map}\langle\text{String}, \text{Integer}\rangle \text{ BHT})$  which has: key = the last 8 bits of PC and value= n-bit saturating counter (Figure1- visualize). We initialize the n-bit counter to 0 (NT) at the beginning of the processing.

For each branch, if branch is TAKEN (actual as per trace) and predicted is less than  $2^n/2$ , we increment the saturating counter. Else we decrement the counter. i.e If the counter is greater than or equal to half its maximum value, predict the branch as taken.

Snippet of Code below:

```
//n-bit Branch predictor

public static void nBit_predictor(List<String> tracelist, int n){

    Map<String,Integer> BHT = new LinkedHashMap<String,Integer>();
    int total_count=0;
    int mispred_count =0 ; //to count number of mispredictions
    int pred_value;

    BHT = initialize_BHT(256, BHT);

    for (String entry : tracelist){

        String index_bht = entry.substring(4, 6);
        pred_value = BHT.get(index_bht);
        // System.out.println("BHT Index :" +index_bht+ " Branch direction:" + entry);
        if(entry.contains("T")){

            if(pred_value<(n/2)){
                mispred_count++;
                pred_value = (pred_value < (n-1))? pred_value+1 : n-1; //Increment count but saturate at n-1.Saturating counter logic
                BHT.put(index_bht, pred_value);
            }
            else{
                //counter still needs to increment- case of strongly taken
                pred_value = (pred_value < (n-1))? pred_value+1 : n-1; //Saturating counter logic
            }
            total_count++;
        }
        else{//entry.getValue().equals("NT")

            if(pred_value>(n/2)){
                mispred_count++;
                pred_value = (pred_value ==0)? 0: pred_value-1; //Decrement counter but saturate at 0
                BHT.put(index_bht, pred_value);
            }
            else{
                //counter still needs to decrement- case of weakly taken- still needs to decrement and remain at 0
                pred_value = (pred_value ==0)? 0: pred_value-1; //Saturating counter logic
            }
            total_count++;
        }

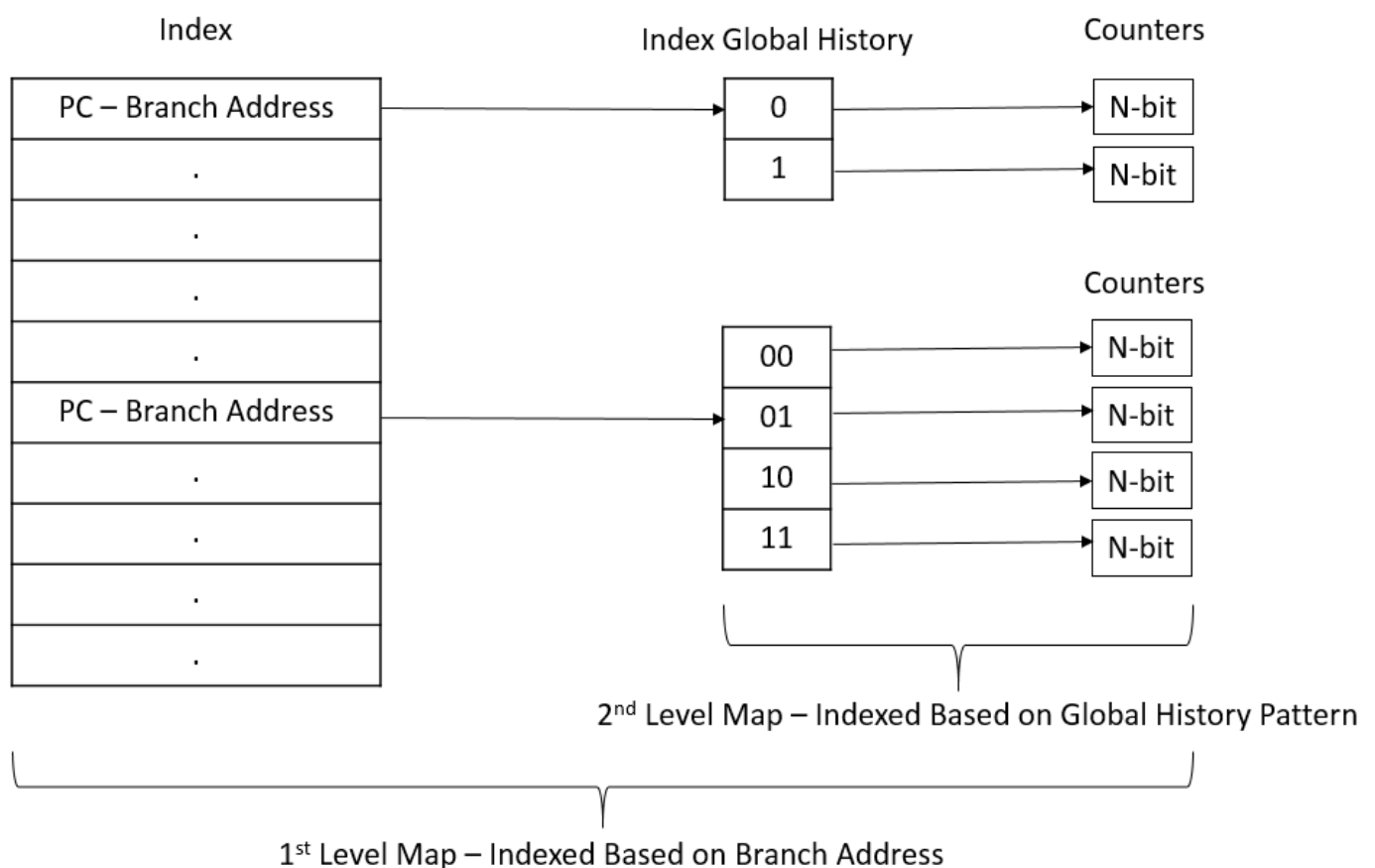
    }

    Utils.printStatistics(total count, mispred count);
}
```

**Type 2:** High-level Overview of the Correlating Predictor (m,n) Implementation Logic

In two-level predictors, we keep track of the behavior of previous branches, and use that to predict the behavior of the current branch. To track the behavior we use global history buffer whose size can be configurable from the config.properties file. For the (m,n) predictor, say  $m=1$ ,  $n=2$  i.e (1,2) bit predictor, we use 1 bit to indicate direction of previous branch and a 2-bit saturating counter.

For (m,n) we use behavior of last m branches to choose the n-bit saturating counter per branch. This is maintained in a Pattern History Table(PHT) which is indexed per branch's least significant bits. The logic is depicted below in figure. This is representative of how the data structure we have used in Java code. We have shown two examples – one where Global history buffer size is 1 and another when it is 2. This is implemented using Java Nested Maps.



## Code snippet:

```

public static void correlating_m_nPredictor(List<String> tracelist,int m, int n){
    //This will be a map of Maps . First Map's key is PC's least significant bits, second
    //map's key is global history value,
    Map<String,HashMap<String,Integer>> PHT = new LinkedHashMap<String,HashMap<String,Integer>>();

    int pred_value = 0; //value as given by n-bit counter in the Pattern history Table.
    int mispred_count = 0;
    int total_count = 0;
    int globhist_buffer = 0; // this will be decimal value in this variable but binary string in mapindex

    //Initialize PHT Map
    PHT=initialize_PHT(256,m);

    for(String entry:tracelist){
        String index_bht = entry.substring(4, 6);
        String new_global;
        Map<String,Integer> subMap= new HashMap<String,Integer>();
        Map<String,Integer> updatedMap= new HashMap<String, Integer>();

        subMap = PHT.get(index_bht);
        pred_value = subMap.get(Utils.decToBin(globhist_buffer,m)); // indexing based on the global history value- gives
                                                                    //value of saturating counter

        //n=1 is special case handle separate
        if(n > 1){

            if(entry.contains("T")){
                if(pred_value<(n/2)){//change this to pow(2) later
                    mispred_count++;
                    pred_value = (pred_value < (n-1))? pred_value+1 : n-1; //Saturating counter logic
                    updatedMap = PHT.get(index_bht); // has to be done like this else all branches gets updated
                    updatedMap.put(Utils.decToBin(globhist_buffer,m), pred_value );

                    //Update global history buffer

                    new_global = Utils.shiftreg(Utils.decToBin(globhist_buffer,m), 1); // this value is 1 because branch was supposed to be taken
                    globhist_buffer = Utils.binToDec(new_global);

                    //System.out.println("Global in shiftreg: "+ new_global+ " Global in buffer: "+globhist_buffer);
                }
                else{
                    //counter still needs to increment- case of strongly taken
                    pred_value = (pred_value < (n-1))? pred_value+1 : n-1; //Saturating counter logic

                    //Update global history buffer- still
                    new_global = Utils.shiftreg(Utils.decToBin(globhist_buffer,m), 1); // this value is 1 because branch was supposed to be taken
                    globhist_buffer = Utils.binToDec(new_global);
                    // System.out.println("Global in shiftreg: "+ new_global+ " Global in buffer: "+globhist_buffer);
                }
                total_count++;
            }
            else{//entry.contains("N")
                if(pred_value>(n/2)){//this will be mis-prediction
                    mispred_count++;
                    pred_value = (pred_value ==0)? 0: pred_value-1; //Decrement counter but saturate at 0
                    subMap.put(Utils.decToBin(globhist_buffer,m), pred_value);

                    //Update global history buffer

                    new_global = Utils.shiftreg(Utils.decToBin(globhist_buffer,m), 0); // this value is 0 because branch was supposed to be taken
                    globhist_buffer = Utils.binToDec(new_global);

                    //System.out.println("Global in shiftreg: "+ new_global+ " Global in buffer(dec value): "+globhist_buffer);
                }
                else{
                    //counter still needs to increment- case of weakly taken- still needs to decrement and remain at 0
                    pred_value = (pred_value ==0)? 0: pred_value-1; //Saturating counter logic
                    //Update global history buffer - Still
                    new_global = Utils.shiftreg(Utils.decToBin(globhist_buffer,m), 0);
                    globhist_buffer = Utils.binToDec(new_global);

                    //System.out.println("Global in shiftreg: "+ new_global+ " Global in buffer(dec value): "+globhist_buffer);
                }
                total_count++;
            }
        }
    }
}

```

### 3. Running The Simulator

**Step 1:** Open the Virtual Machine. Make sure the Jar file, configuration file and trace file are in the same directory

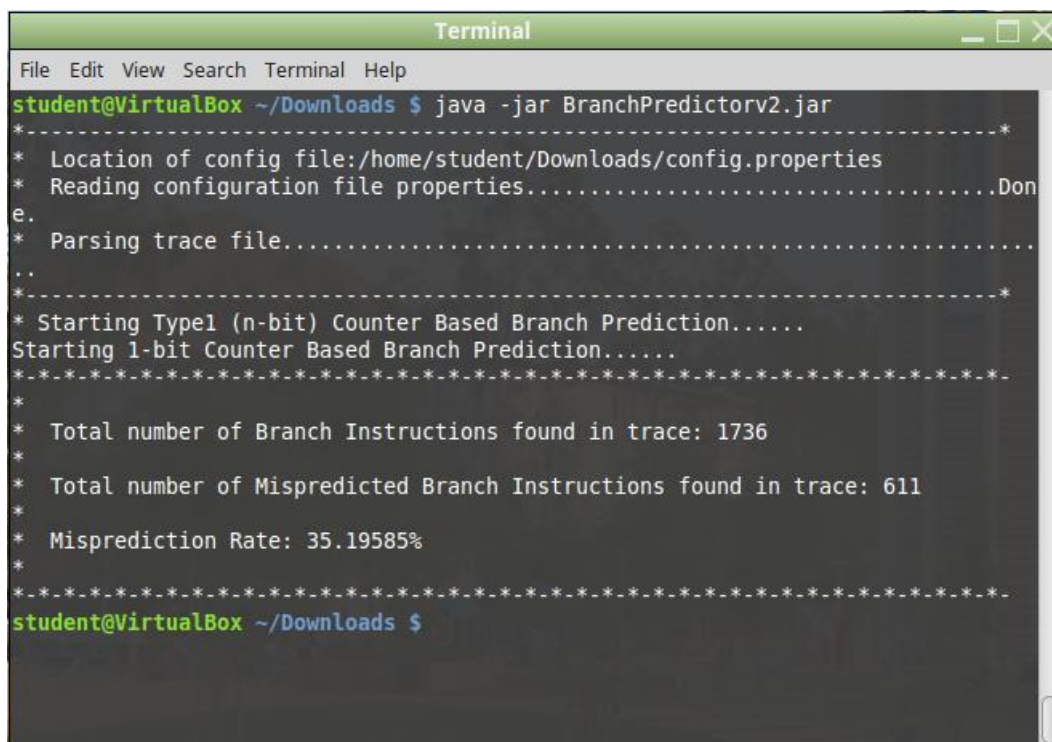
**Step 2:** Set the configuration file for the required settings. ( **config.properties** )

**Step 3:** To run the simulator: `java -jar BranchPredictorv2.jar`

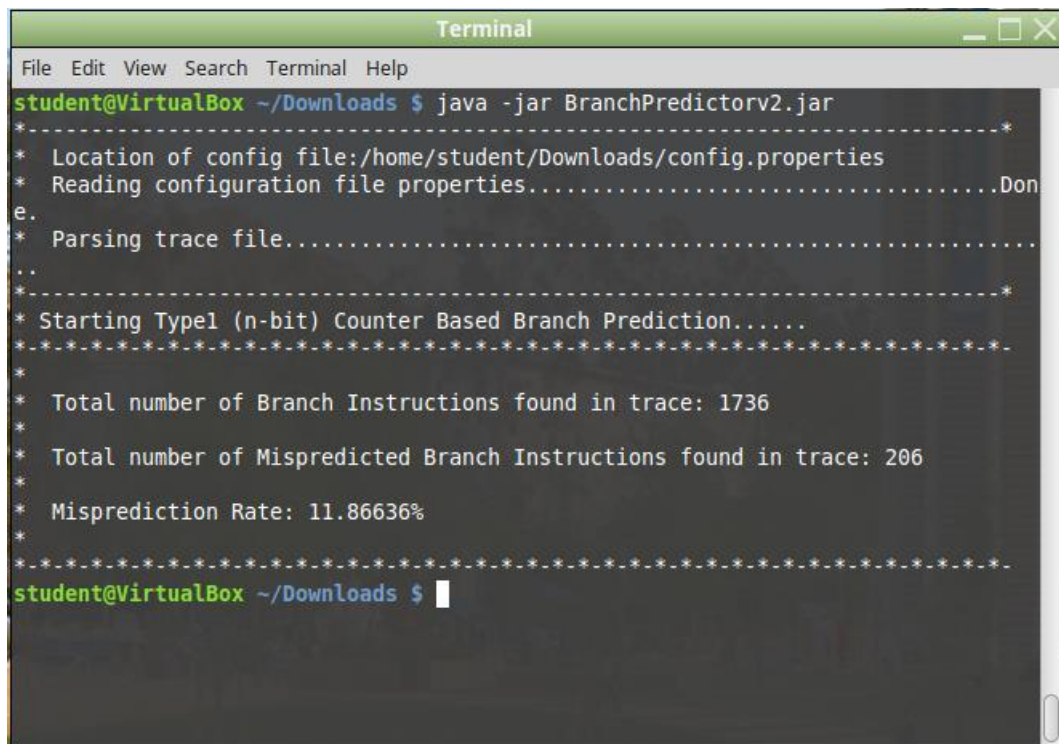
### 4. Testing & Output

We have performed rigorous testing with various configurations and found that the simulator was working as expected. Below are few test cases we have performed.

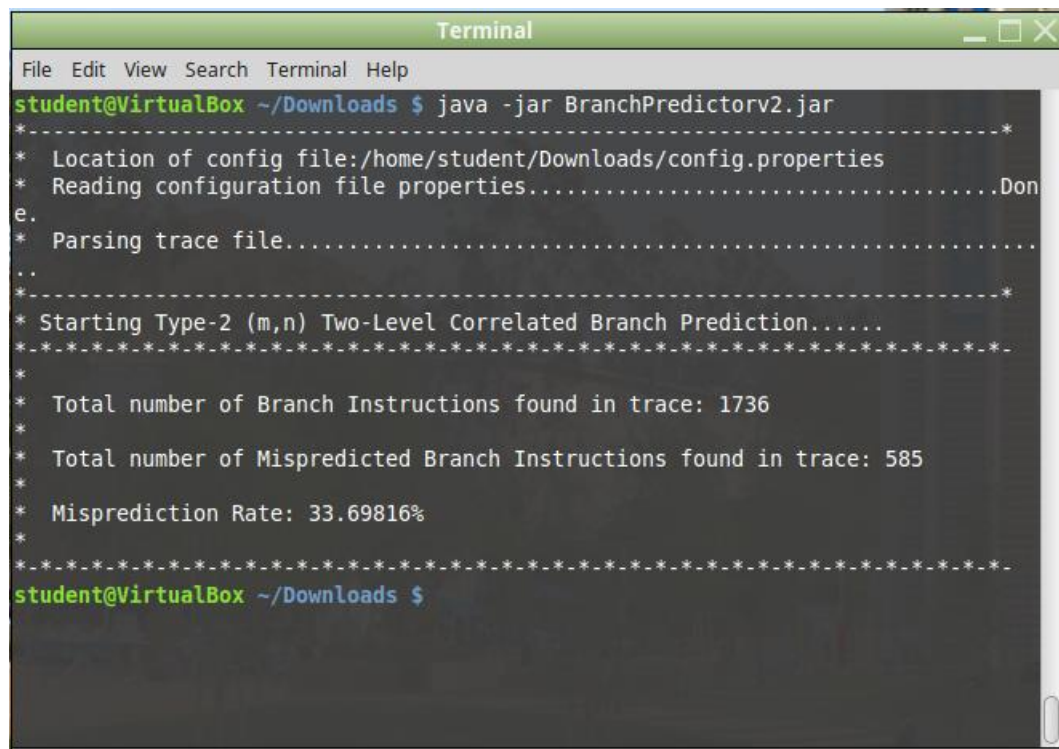
#### 1-Bit Predictor:



```
Terminal
File Edit View Search Terminal Help
student@VirtualBox ~/Downloads $ java -jar BranchPredictorv2.jar
*-----*
* Location of config file:/home/student/Downloads/config.properties
* Reading configuration file properties.....Done.
* Parsing trace file.....
* ..
*-----*
* Starting Type1 (n-bit) Counter Based Branch Prediction.....
Starting 1-bit Counter Based Branch Prediction.....
*-----*
*
* Total number of Branch Instructions found in trace: 1736
*
* Total number of Mispredicted Branch Instructions found in trace: 611
*
* Misprediction Rate: 35.19585%
*
*-----*
student@VirtualBox ~/Downloads $
```

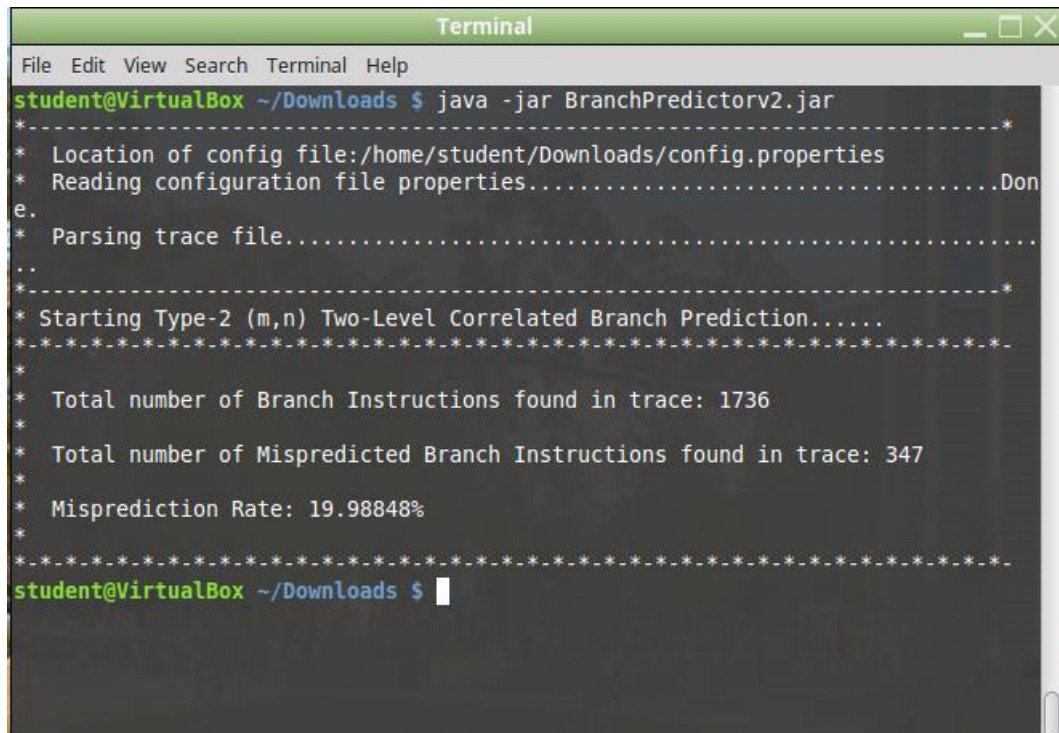
**2-Bit Predictor:**

```
Terminal
File Edit View Search Terminal Help
student@VirtualBox ~/Downloads $ java -jar BranchPredictorv2.jar
*-----*
* Location of config file:/home/student/Downloads/config.properties
* Reading configuration file properties.....Done.
* Parsing trace file.....
*
*-----*
* Starting Type1 (n-bit) Counter Based Branch Prediction.....
*-----*
* Total number of Branch Instructions found in trace: 1736
* Total number of Mispredicted Branch Instructions found in trace: 206
* Misprediction Rate: 11.86636%
*-----*
student@VirtualBox ~/Downloads $
```

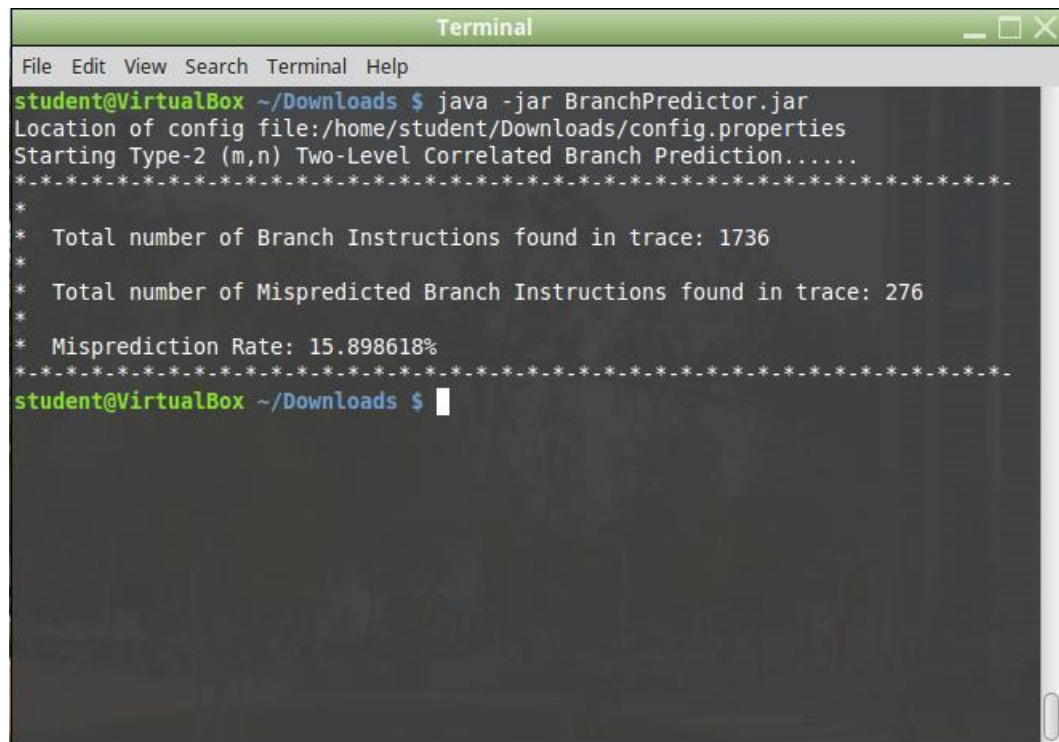
**(1,1) Correlation Predictor (m,n):**

```
Terminal
File Edit View Search Terminal Help
student@VirtualBox ~/Downloads $ java -jar BranchPredictorv2.jar
*-----*
* Location of config file:/home/student/Downloads/config.properties
* Reading configuration file properties.....Done.
* Parsing trace file.....
*
*-----*
* Starting Type-2 (m,n) Two-Level Correlated Branch Prediction.....
*-----*
* Total number of Branch Instructions found in trace: 1736
* Total number of Mispredicted Branch Instructions found in trace: 585
* Misprediction Rate: 33.69816%
*-----*
student@VirtualBox ~/Downloads $
```



**(2,2) Correlation Predictor (m,n):**

```
Terminal
File Edit View Search Terminal Help
student@VirtualBox ~/Downloads $ java -jar BranchPredictorv2.jar
*-----*
* Location of config file:/home/student/Downloads/config.properties
* Reading configuration file properties.....Done.
* Parsing trace file.....
* ..
*-----*
* Starting Type-2 (m,n) Two-Level Correlated Branch Prediction.....
*-----*
* Total number of Branch Instructions found in trace: 1736
* Total number of Mispredicted Branch Instructions found in trace: 347
* Misprediction Rate: 19.98848%
*-----*
student@VirtualBox ~/Downloads $
```

**(1,3) Correlation Predictor (m,n):**

```
Terminal
File Edit View Search Terminal Help
student@VirtualBox ~/Downloads $ java -jar BranchPredictor.jar
Location of config file:/home/student/Downloads/config.properties
Starting Type-2 (m,n) Two-Level Correlated Branch Prediction.....
*-----*
* Total number of Branch Instructions found in trace: 1736
* Total number of Mispredicted Branch Instructions found in trace: 276
* Misprediction Rate: 15.898618%
*-----*
student@VirtualBox ~/Downloads $
```