# Computer Architecture
## Assignment-3

Submitted by:

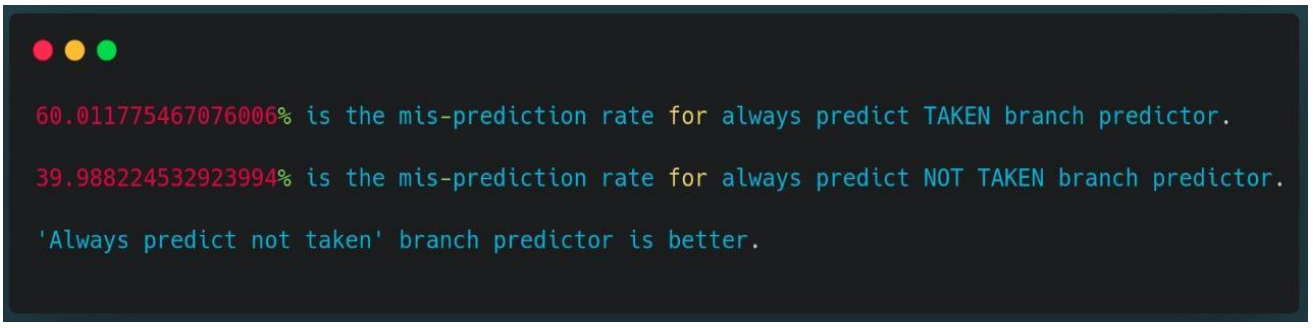
Madhav Sood (IMT2021009), Nilay Kamat (IMT2021096) and Anuj Arora (IMT2021050)

# Q2a.

## Logic used in the code :

- First, we open the trace file using the open command. **f_contents** is a list of all the lines present in the given trace file - **"file0.trace"**. Each element in **f_contents** contains the decimal equivalent of the 32-bit address and the branch outcome.
- We then take two counters, one to store mis-predictions in the "Always Taken" branch predictor and one for the "Always Not Taken" branch predictor.
- For the "Always Taken" branch predictor, if the line in the trace file shows **N**, then we increment the number of wrong(mis) predictions, i.e., **NotTaken** counter by 1.
  We then calculate the percentage of the mis-prediction by this branch predictor.
- Now for the "Always Not Taken" branch predictor, we check for the occurrence of **T** in the list and repeat the same process.
- We then proceed to check which mis-prediction rate was lesser. The branch predictor whose miss rate is lower is more accurate.
- Finally, we outputted the values and results we obtained from the calculations onto the output file - **"IMT2021009_IMT2021050_IMT2021096_output_2a.txt"**

## Output Screenshot :




## Results and Observations :

We observed the values of the mis-prediction rates to be 60.011% and 39.988% for "Always Taken" and "Always Not Taken" branch predictors respectively. From these observations, we can deduce that the "Always Not Taken" branch predictor is more accurate as its mis-prediction rate is lesser than the "Always Taken" branch predictor.

# Q2b.

## Logic used in the code :

- Similar to step 1 of Q2a, we open the trace file using the open command. **f_contents** is a list of all the lines present in the given trace file - **"file0.trace"**. Each element in **f_contents** contains the decimal equivalent of the 32-bit address and the branch outcome.
- We have to find the miss rates for predictors of size 2^2, 2^3 and so on till 2^20. Variable **index_size** is used to store the index bits which will be required for these predictors. Number of index bits is the **log** of the size of the predictor, thus from 2 to 20.
- Initially, we define **index_size** = 2 and then run a while loop to calculate miss rates for all predictors till **index_size** = 20. (**included**)
- We take two counters, namely **count_correctP** and **count_wrongP** to store the number of correct predictions and wrong(mis) predictions respectively and initialize both to 0 for every predictor. We also create an empty list that has dictionaries of the decimal value of index bits as key and the current prediction as the value.
- As given in the question, we set the initial value of prediction to Strongly Taken(0) for all indices present in the list.
- Now for every element in **f_contents** we find the index of the space between address and T/N and then calculate the address using string slicing. Then we convert this address(string with numbers in decimal format) into a 32-bit binary string using the function **getBinary_32().**
- Now every predictor has a different number of index bits. We use the **getIndex()** function to extract the required number of bits from the end of the binary string and then convert it into decimal using the function **BinaryToDecimal()**. This stores the value corresponding to which we have to check the following branch predictor states:

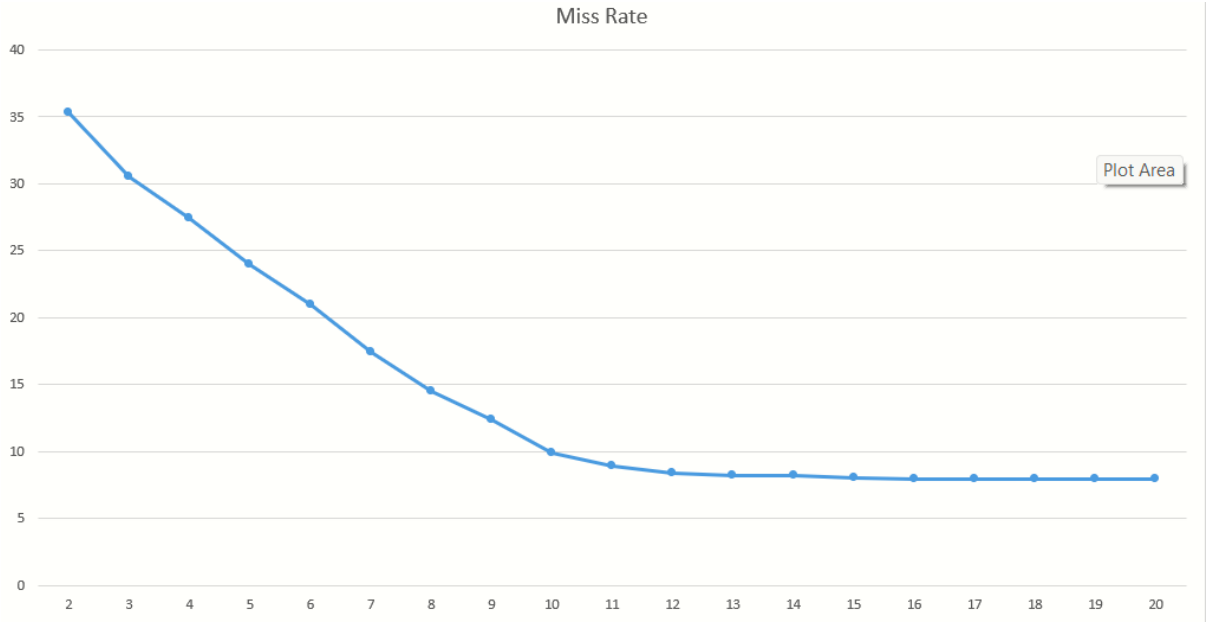
  0 - Strongly Taken
  1 - Weakly Taken
  2 - Weakly Not Taken
  3 - Strongly Not Taken

- If the branch outcome is T, the prediction is correct if the value corresponding to the index(key) is 0 or 1, that is, weakly taken or strongly taken. In that case, we increment count_correctP. Else, we increment count_wrongP.
- We also have to keep in account that we have to change the state of the predictor value. If the predictor has anything other than Strongly Taken(0), we reduce it by 1 as the branch outcome is T.
- Now if the branch outcome is N, we follow similar steps to change the state and increment the counters. Here, we increment count_correctP if the value is 2 or 3, that is, weakly not taken or strongly not taken. Else, we increment count_wrongP.

The state is increased by 1 if the predictor is in any state other than Strongly Not Taken(3).

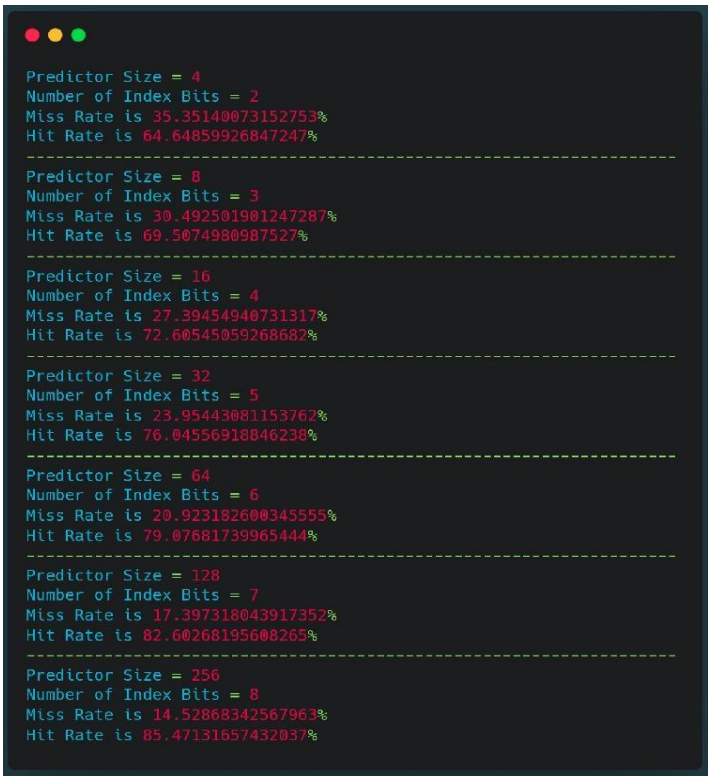
- Then we calculated miss and hit rates for the predictor whose index bits are the current index_size. We also calculate the predictor size for the current predictor.(predictor size = 2^index_size).
- Finally, we outputted the values and results we obtained from the calculations into the output file - **"IMT2021009_IMT2021050_IMT2021096_output_2b.txt"**

## Line Plot :

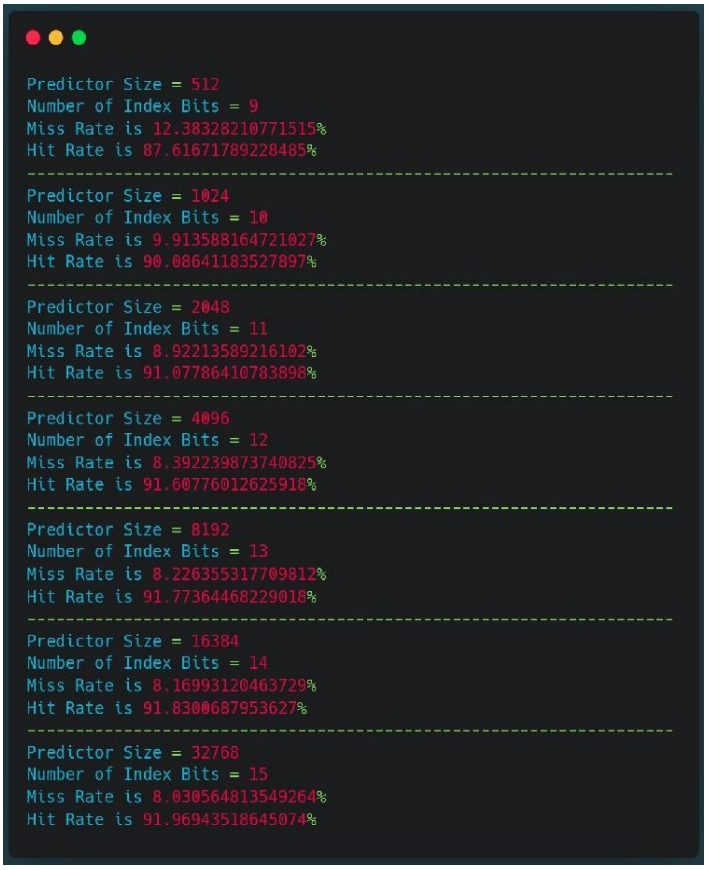



**Y-Axis:** Percentage of branches mis-predicted
**X-Axis:** Log of the Predictor Size (number of Index Bits)

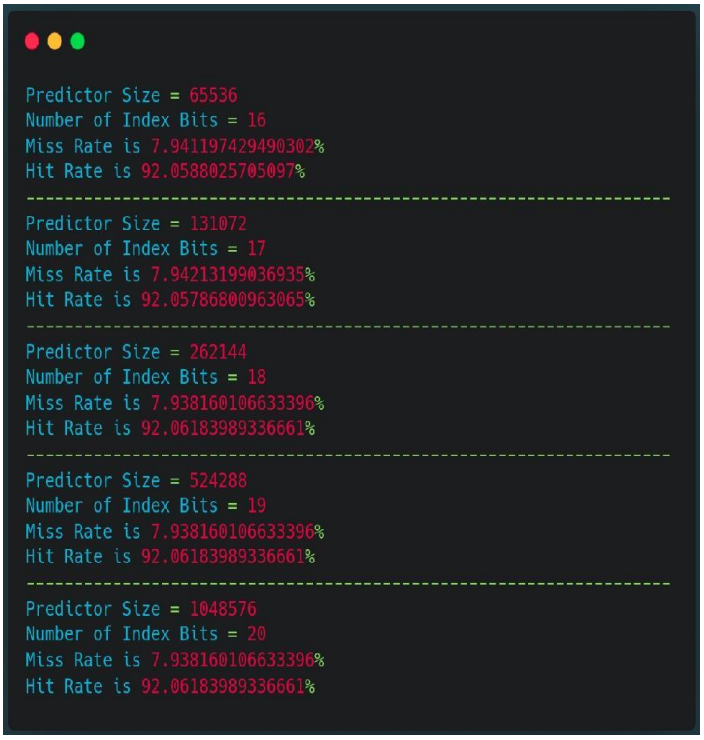
## Output Screenshots :



```
Predictor Size = 4
Number of Index Bits = 2
Miss Rate is 35.35140073152753%
Hit Rate is 64.64859926847247%
------------------------------------------------------------------------
Predictor Size = 8
Number of Index Bits = 3
Miss Rate is 30.492501901247287%
Hit Rate is 69.5074980987527%
------------------------------------------------------------------------
Predictor Size = 16
Number of Index Bits = 4
Miss Rate is 27.39454940731317%
Hit Rate is 72.60545059268682%
------------------------------------------------------------------------
Predictor Size = 32
Number of Index Bits = 5
Miss Rate is 23.95443081153762%
Hit Rate is 76.04556918846238%
------------------------------------------------------------------------
Predictor Size = 64
Number of Index Bits = 6
Miss Rate is 20.923182600345555%
Hit Rate is 79.07681739965444%
------------------------------------------------------------------------
Predictor Size = 128
Number of Index Bits = 7
Miss Rate is 17.397318043917352%
Hit Rate is 82.60268195608265%
------------------------------------------------------------------------
Predictor Size = 256
Number of Index Bits = 8
Miss Rate is 14.52868342567963%
Hit Rate is 85.47131657432037%
```



```
Predictor Size = 512
Number of Index Bits = 9
Miss Rate is 12.38328210771515%
Hit Rate is 87.61671789228485%
------------------------------------------------------------------------
Predictor Size = 1024
Number of Index Bits = 10
Miss Rate is 9.913588164721027%
Hit Rate is 90.08641183527897%
------------------------------------------------------------------------
Predictor Size = 2048
Number of Index Bits = 11
Miss Rate is 8.92213589216102%
Hit Rate is 91.07786410783898%
------------------------------------------------------------------------
Predictor Size = 4096
Number of Index Bits = 12
Miss Rate is 8.392239873740825%
Hit Rate is 91.60776012625918%
------------------------------------------------------------------------
Predictor Size = 8192
Number of Index Bits = 13
Miss Rate is 8.226355317709812%
Hit Rate is 91.77364468229018%
------------------------------------------------------------------------
Predictor Size = 16384
Number of Index Bits = 14
Miss Rate is 8.16993120463729%
Hit Rate is 91.8300687953627%
------------------------------------------------------------------------
Predictor Size = 32768
Number of Index Bits = 15
Miss Rate is 8.030564813549264%
Hit Rate is 91.96943518645074%
```




## Results and Observations :

- The best misprediction rate obtained in the analysis carried out by us was found to be 7.938%(rounded to 3 decimals).

- The better of "Always Taken" and "Always Not Taken" was the latter with a miss rate of 39.988% as calculated in Q2a. A predictor with approximately half mispredictions would be one with a miss rate of approx. 20%.
  From the results we obtained, we can observe that the predictor with 6 index bits or 64 counters has a miss rate of 20.923%. Thus, this is the required predictor.

- From the values of miss rates obtained, we can notice that predictors with index_size as 18, 19 and 20 have the same miss rate of 7.938%. This conclusion is also visible in the line graph as the line flattens as it approaches 20 index bits. Therefore, the predictor needs to be of size 2^18 (262144) before it basically captures almost all of the benefits of a much larger predictor.