CA-ASSIGNMENT-3

REPORT

-Written by Pranav Bhutada (IMT2021105)

Yogesh Goyal (IMT2021542)

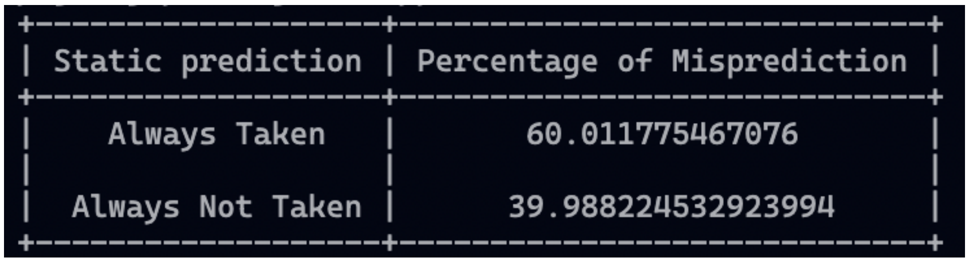
Yash Gupta (IMT2021514)

Q2)

A) In this question we have implemented static branch predictors for two cases on the same input data. One assuming the branch will be always taken and in the other case assuming the branch will be always not taken. For each case, we have calculated the percentage of misprediction.

*Percentage of misprediction= (Total Mispredictions/Total branches) *100*

A snapshot of the result for the same is attached below:

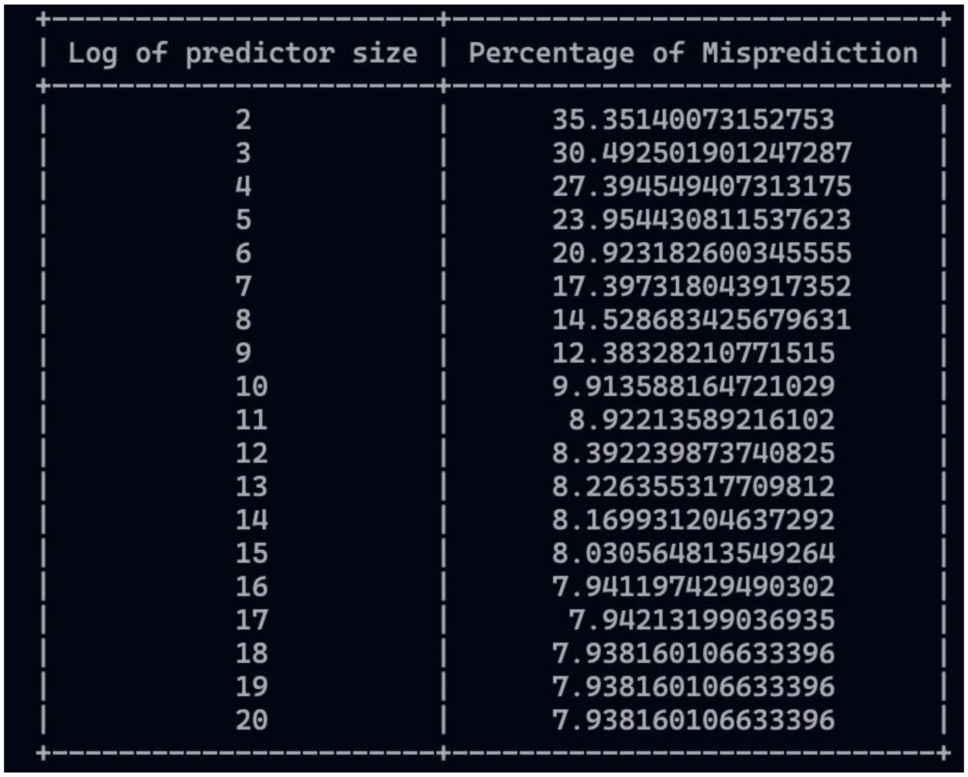


```
+------------------+----------------------------+
| Static prediction | Percentage of Misprediction |
+------------------+----------------------------+
|   Always Taken    |      60.011775467076       |
|                   |                            |
|  Always Not Taken |     39.988224532923994     |
+------------------+----------------------------+
```

<u>Observation</u> – As we can see in the above snapshot the <u>*ALWAYS NOT TAKEN*</u> policy is more accurate as the percentage of misprediction for the same input data is lower.

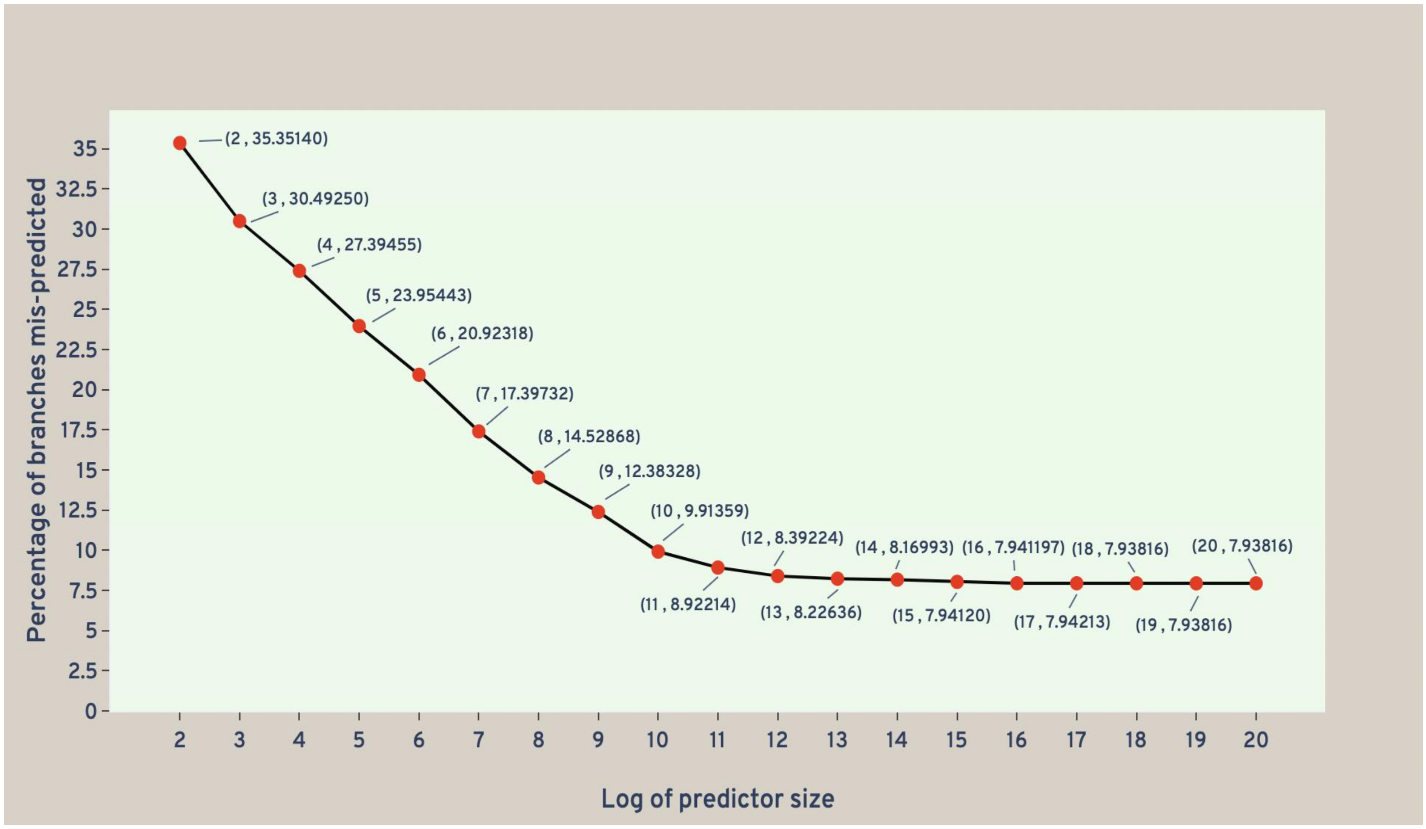
B) In this question, we implement a simple dynamic branch direction predictor, an array of 2^n two-bit counters. We initialize the predictor to “strongly taken” at the start. To make a prediction, the predictor selects a counter from the table using the lower-order n bits of the instruction's address (its program counter value). After each branch (correctly predicted or not), the hardware increments or decrements the corresponding counter to bias the counter toward the actual branch outcome (the outcome given in the trace file).

A snapshot of the output:



```
+-----------------------+----------------------------+
| Log of predictor size | Percentage of Misprediction |
+-----------------------+----------------------------+
|           2           |     35.35140073152753      |
|           3           |     30.492501901247287     |
|           4           |     27.394549407313175     |
|           5           |     23.954430811537623     |
|           6           |     20.923182600345555     |
|           7           |     17.397318043917352     |
|           8           |     14.528683425679631     |
|           9           |     12.38328210771515      |
|          10           |     9.913588164721029      |
|          11           |      8.92213589216102      |
|          12           |     8.392239873740825      |
|          13           |     8.226355317709812      |
|          14           |     8.169931204637292      |
|          15           |     8.030564813549264      |
|          16           |     7.941197429490302      |
|          17           |      7.94213199036935      |
|          18           |     7.938160106633396      |
|          19           |     7.938160106633396      |
|          20           |     7.938160106633396      |
+-----------------------+----------------------------+
```

A snapshot of the graph:



This graph shows the trend of the percentage of branches mis predicted over the increasing values of (log of predictor size -that is n). The percentage of misprediction decreases as the value of n increases (that is the slope of the graph decreases), but after a point, the slope of the graph almost becomes constant. At this point, the performance of the predictor is pretty much maxed out.

- What is the best mis-prediction rate obtained in the analysis carried out?

--> (7.938160106633396) is the best prediction rate obtained in the analysis.

- How large must the predictor be to reduce the number of mis-predictions by approximately half as compared to the better of "always taken" and "always not taken"? Give the predictor size both in terms of number of counters as well as bits.

--> The better result of the percentage of misprediction between “always taken” and “always not taken” is approximately 39.988 for the “always not taken” prediction. Half of this is = 19.994. So, to obtain this, the value of n must be at least 7, that is 7 bits and 2^7 =128 number of counters.

- At what point does the performance of the predictor pretty much max out? That is, how large does the predictor need to be before it basically captures almost all the benefits of a much larger predictor.

-->The performance of the predictor is maxed out at n=15. As can be seen, the slope of the graph becomes almost constant from this value.