Aadu Pirn
Jonathan Selling

All of our data is located in 1541_Project1_Data.xlsx that we will be referring to. Comparing the results between the sample traces and each prediction method, we can see a variability that is common. It is always better to have the branch predictor since on average you have 8.34% less cycles when using the 1-bit predictor compared to predicting all branches as not taken. This would show that having a branch predictor would be beneficial and make programs run faster than if they automatically predict not taken. Comparing the 2-bit predictor and the 1-bit predictor, it is slightly better because you have 0.63% less cycles.

The last two tables on our excel sheet show the data for a size-32 section table, and a size-128 table. When implementing this, the size-32 section table required more cycles than the size-64 table and the size-128 table required the least. Using 1-bit prediction, you have 1.99% less cycles using the size-32 table compared to the size-64 table. Switching from the size-32 table to the size-128 table there are 2.00% less cycles. Having a bigger table than 32 is better, but the improvement from a size-64 to size-128 table is not worth implementing if the cost is not roughly the same.

Looking at this data you can see that using a predictor at all is definitely beneficial. You use significantly less cycles to run the same code. Modifying the predictor by adding bits or expanding the size of the prediction array helps but does so marginally. Improving the predictor would probably be beneficial if it was cheap but if it requires significantly more power, cost or causes other downsides you would probably be better off leaving the predictor alone.

Improvements to the predictor should probably be done on a case by case basis. The jump from a 32 to 64 sized array was a lot bigger than the jump to 128. Some improvements may be worth it.