

Figure 1: Accuracies of different testbenches with the different methods

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | fppp | gcc | go | Vpr |
| predacc\_i | 0.8728 | 0.8584 | 0.7329 | 0.8627 |
| predacc\_ii | 0.9164 | 0.8944 | 0.7963 | 0.8933 |
| predacc\_iii | 0.8796 | 0.8731 | 0.7509 | 0.8765 |
| predacc\_iv | 0.9311 | 0.9254 | 0.8342 | 0.9193 |
| predacc\_v | 0.9467 | 0.9405 | 0.8631 | 0.9367 |

The four different parts of the assignment added bits to the branch predictor to increase accuracy. The single bit predictor is the baseline seen in part i. Part ii is a saturating predictor that takes 2 bits similarly to part iii. It should be noted that having the saturating counter made a larger difference in predictions than history (for 2bits used prediction), meaning that the previous branch mattered less in determining the next branch than ignoring single differences. From part iv, combining the two methods increased the prediction accuracy greatly (note that a 1% change with only a 10% miss is practically a 10% difference). Part v took multiple attempts and it seems that the more history with the 2-bit saturating predictor the more accuracy that is found. This basically means that many situations often repeat and the longer the history the more situations that are covered.

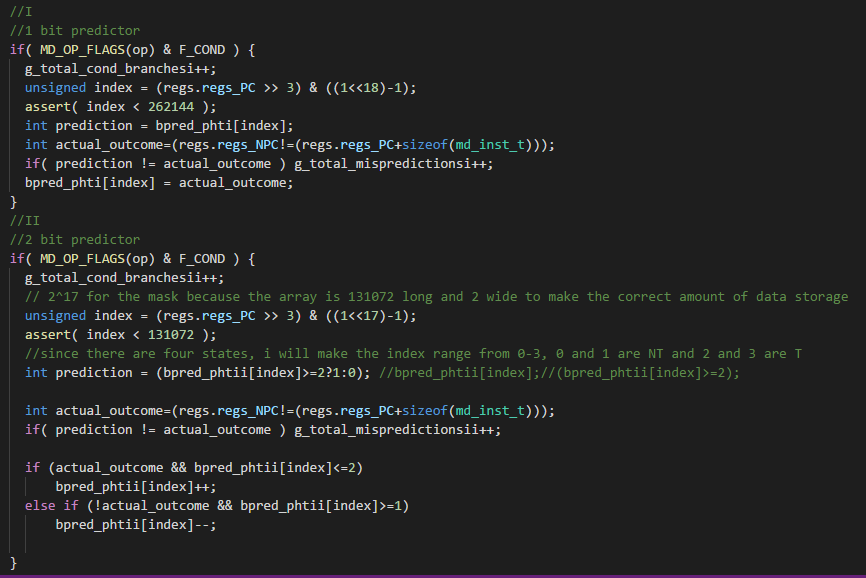


Figure 3: Implementation of parts i and ii

Part i was the basic predictor that was just one bit. It checked if a conditional branch was occurring and then make the index value for the predictor. Using the previous result stored at the index value the next outcome was predicted and then compared for correctness. The two bit predictor did the same however would always try to change the value at the index depending on the current value there and the actual outcome.

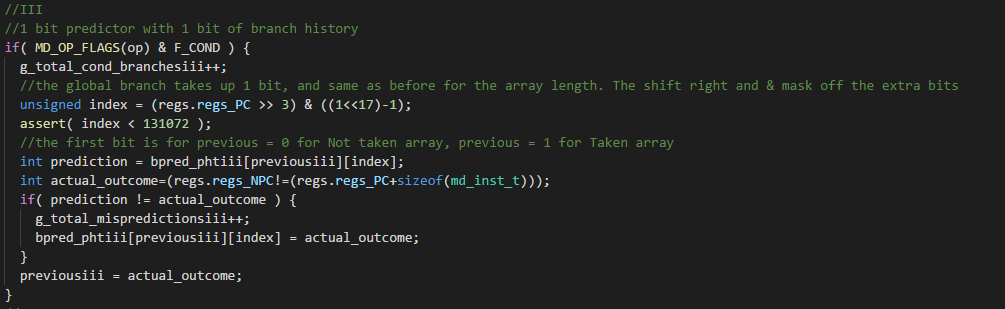


Figure 4: Implementation of part iii

Part iii used the one bit predictor with one bit of history. Instead of using one array with index values from 0 to 262071, it used two arrays of 131072 the first array being for previously Not Take and the second array being for Previously taken. The value in each array was changed at their appropriate times.

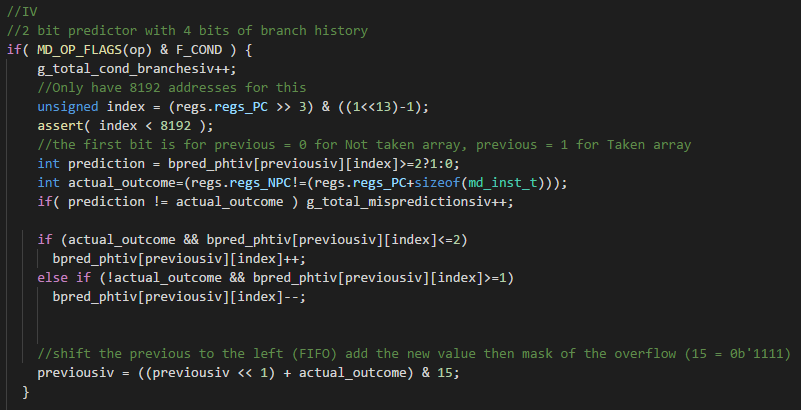


Figure 5: Implementation of part iv

Part iv combined ii and iii. This is just a mix of both of the codes taking into consideration the change of length and masking of the index array for the saturating counters. In this case each was 8192 because of 16 “columns” for the history and 2 bits per row. At the bottom the previous history was made by shifting to the left and then adding the actual outcome and masking off the overflow with and AND.

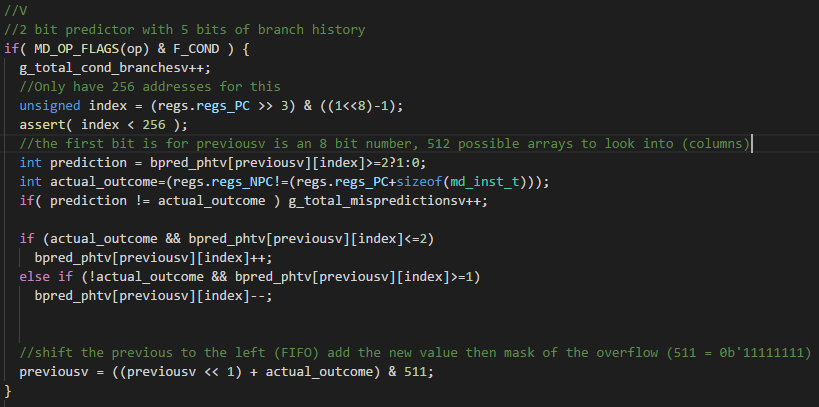


Figure 6: Implementation of v

Part v was done exactly the same as part iv and was found to be effective at 256 rows of 2bits each and 512 columns.