1. The code (commented out) is poor because it selects one of 4 different if statements during runtime. As we know from branch prediction, having a mispredicted branch is incredibly costly to the processor and results in a high CPI. The processor has to fill the pipeline with nops. Instead, I replaced these if statements with 4 statements that all execute. This decreases the branch mispredictions by a factor of 4. In the previous implementation, there are 4 if statements added in addition to each original if statement, so this reduces the branch count by 4x.
2. Since the storage cap is 33KB, I created a storage space of 32000 bits for a 2 bit correlational prediction buffer. Essentially, each branch is directly mapped into the array (each index modulus (32000 divided by 5)). The reason 32000 is divided by 5 is because each index carries 5 bits. The first bit indicates a whether this index correlates to a branch. The next two bits acts as a 2 bit branch predictor in the case that the last branch was not taken. The next two bits acts as a 2 bit branch predictor in the case that the last branch was taken. This can predict sequences with history of 3 bits. In the case that a new branch is found and has not yet been set in the prediction buffer, a standard 2 bit branch predictor suffices. This implementation is advantageous to a standard 2 bit branch predictor, which based on my tests, had a very low prediction rate (~70%). A one bit prediction buffer brought the prediction rate to ~90%, and making it a 2 bit correlational prediction buffer brought it to ~95%. Notice the tradeoff. With a fixed storage size, the more bits of history I can have for each branch in the buffer, the fewer number of indices I can have.

http://web.engr.oregonstate.edu/~benl/Projects/branch_pred/

3. Implementing the 32000 bit prediction buffer would be the most costly thing to implement in silicon. It would require 5 memory accesses per branch instruction (since there are 5 bits stored for every single index in the prediction buffer). Since memory accesses take the largest number of cycles, it could be an issue. The other portion of the implementation, which is the 2 bit predictor which acts independently of the prediction buffer, would run fine since it is basic combinational logic that only would need two flip flops for the two bits.
4. A real branch predictor might have a tournament style branch predictor which runs multiple predictions, then picks the prediction that has been most successful based on past history. This would certainly require more space on the CPU, which is the hardware downside to having multiple predictions. Since my predictor is not tournament style, it may work well for certain patterns of code, yet not very well for other patterns. My code assumes a degree of regularity for the last 3 branch options for each branch. If a different pattern is more dominant, such as changing regularity which can be noted in a global predictor, my predictor will not be so as useful. Even if my predictor takes up less space, because it could mispredict when taking a global look at the code, the lower space

tradeoff would not be as advantageous as a better predictor, since misprediction is incredibly costly and wastes the pipeline with nops.