Saturating Bit Counter Simulator: An Evaluation on the Effects of Varying Bits

Syth Ryan

Department of Electrical and Computer Engineering

Iowa State University

[syth@iastate.edu](mailto:syth@iastate.edu)

**Abstract**

*Since the introduction of the Saturating Bit Counter in the 1980’s [1] there have been numerous ideas using this algorithm in branch prediction. Something that can be modified with the saturating bit counter is the number of bits. I wanted to build a simulator that could aid in deciding what amount of bits is right for projects on a more individual basis.*

*Expanding on this idea, we could possibly pair the saturating bit counter simulator’s best prediction with a compiler. That compiler could be recommended a bit size for their project specific saturating bit counter.*

*Whenever a prediction is correctly made, we drastically reduce computing time. It is possible to try and run both directions of a branch simultaneously and take the correct side after computation but this is expensive in terms of resources. This tells me that even correctly predicting only a small amount of branch instructions has the potential to benefit the system in computation time.*

**1. Introduction**

In life, we can prepare for many things if we have the prior knowledge. We know that a usual day would entail; waking up, brushing teeth, getting dressed, going to work. Sometimes the usual day could be interrupted, for example there may be a snow storm that prevents you from going to work, and your day has now changed.

Besides the few unexpected events such as a snow storm, history tends to repeat itself. If we have a good knowledge on the past we can better predict the future. If we try to predict what will happen in a day, it would be safer to say that we have a usual day. One snow storm may affect one or maybe two days but shouldn’t we still prepare for work tomorrow?

The Saturating Bit Counter tends to follow a similar pattern, the more bits the counter has the more tolerance there is to changing its prediction. This idea has been widely used in branch prediction. One common use for this is the Tournament Predictor where there are two or more types of predictions and based on the Saturating bit counter, it chooses the side more heavily weighted or is correct more than the other.

Sometimes there are programs that require more tolerance than others. We want to find out if modifying the number of bits used for a saturating bit counter for specific programs has any branch prediction improvement.

If the result of manipulating the size of the Saturating Bit Counter significantly increases branch prediction, I hope to move onto adjusting the Simulator to help compilers predict what size of bit to use.

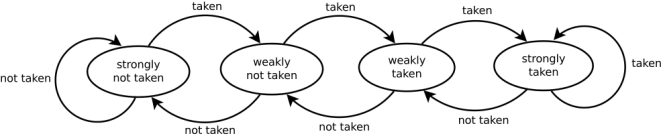
**2. Saturating Bit Counter**

The Saturating Bit counter was introduced by James E. Smith in his paper *A Study of Branch Prediction Strategies* in 1981[1]. He described the many prediction algorithms that were currently in place and proposed new strategies of predicting branches that would take a fraction of the space but still have the ability to accurately predict.

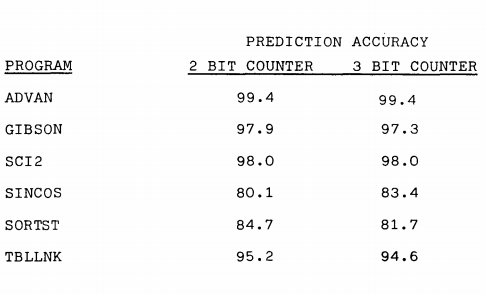
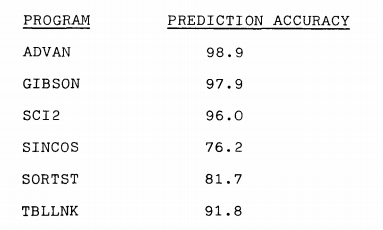
The need for branch prediction comes from the increasing need for faster computers. Conditional branch instructions take up valuable time; instructions have to come to a standstill in order to determine what set of instructions to execute next in the pipeline. An idea that has been continuously used to speed up this process is to start execution of a predicted branch path’s next instructions. There is a down side to this, incorrect predictions take even longer to correct since there now has to be instructions purged from the pipeline. This is why it is very important to have a high percentage of correct predictions.

There were two strategies related to what is being investigated in Smith’s paper [1]. Strategy 6 explained how a one bit history prediction could be used. The simple logic 0 meaning predict not taken and 1 equaling predict taken. The history bit is determined by the previous branch, if the prior branch was taken then predict taken for the next and vice versa. Strategy 7 is the same basic idea but using a 2 bit and 3 bit counter. *Figure 1* shows a two bit counter. We now have a longer history. The more taken branches in the recent last 2 branches mean we should predict taken. This could solve issues if there are multiple taken branches and only a single or very few not taken branches. There is now more of a tolerance to branch prediction.

Smith explains that the number of bits don’t show significant improvement for prediction accuracy larger than two bits. His results are shown in *Figure* 2 for a comparison between a 2 bit counter and a 3 bit counter and *Figure 3* shows the single bit history.



*Figure* 1



*Figure 3*

*Figure* *2*