Joseph Lewis

11567186

EE 334

Project Final Report

1. Introduction

The problem of this project is to create a simulation of branch prediction and creating a BTB table. The input data received are benchmarks which represents a list of current PC’s. Using a specific state machine, we can make prediction on each entry of a BTB to determine if a branch was taken. I created my simulation of a state machine and a branch predictor using the coding language python. Branch prediction is effective due to the fact data involving PC’s are regular and dynamic prediction in the BTB is more effective than static. Since BTB is a dynamic predictor, it will be more accurate and prevent stalls from happening.

2. Parameters and Observations

The following parameters were based on the least significant bit of my student id number. Since my id number was even used Epresso\_int and Spice\_FP as my benchmarks. The state machines I used for this project was State machines A and Class State machine. The following observations after running both bench marks and both state machines are as follows:

When using state machine A:

Results from Espresso\_int:

Instruction count = 809370

Number of Hits = 130256

Number of Misses = 452

Number of Correct Predictions = 123301

Number of Wrong Predictions = 6955

Number of Collisions = 110

Number of Wrong Address Predictions = 671

Results from Spice\_FP:

Instruction count = 799451

Number of Hits = 141760

Number of Misses = 10085

Number of Correct Predictions = 124103

Number of Wrong Predictions = 17657

Number of Collisions = 9485

Number of Wrong Address Predictions = 10847

Now using the Class state Machine:

Results from Espresso\_int:

Instruction count = 809370

Number of Hits = 130256

Number of Misses = 452

Number of Correct Predictions = 123404

Number of Wrong Predictions = 6852

Number of Collisions = 110

Number of Wrong Address Predictions = 671

Results from Spice\_FP:

Instruction count = 799451

Number of Hits = 141760

Number of Misses = 10085

Number of Correct Predictions = 124243

Number of Wrong Predictions = 17517

Number of Collisions = 9485

Number of Wrong Address Predictions = 10847

3. Result of the simulations

For each of the four simulations I have gotten the following results involving hit rate, prediction accuracy, and Incorrect address percent:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Hit Rate | Prediction accuracy | Incorrect address |
| Machine A Espresso\_int | 99.65% | 94.66% | 9.64% |
| Machine A  Spice\_FP | 93.36% | 87.54% | 61.43% |
| Machine Class Espresso\_int | 99.65% | 94.74% | 9.79% |
| Machine Class Spice\_FP | 93.36% | 87.64% | 61.92% |

The BTB tables are on the attachment representing each of the following simulations tested. Each BTB table is in a text document named by the type of state diagram and the benchmark used. Each entry on the table has the entry value, PC, target PC, and a two bit prediction. The two bit prediction represents what state that entry was in when it comes to the state machine. There is two different python simulations representing the specific state machine used in the branch prediction.

4. Quantitative assessment of results

The results told us information about the effectiveness of the two state machines and benchmarks. The Class state machine was more effective than state machine A. When using Espresso\_int the Class machine had a higher prediction accuracy by .08%. When using Spice\_FP the Class machine had a higher prediction accuracy by .10%. The reason why I think this happened is because when state machine A goes directly to binary node “11” from “01”, it takes two branches taken in a row to get to a taken prediction. This makes this state machine less effective than the class state machine. The class state machine was more effective because it only changed the binary value of the nodes by only “01”. When a branch is taken, we subtract “01”, except when the current node is “00”. When a branch is not taken, we add “01”, except when the current node is “11”.

The Esprosso\_int benchmark overall had better results than the Spice\_FP when it comes to the number of stalling situations. Espresso\_int had a higher hit rate than the Spice\_FP by 6.29%. When using the Class state machine the Esprosso\_int had a higher prediction accuracy by 7.1%. Also, the Espresso\_int had a higher prediction accuracy then the Spice\_FP by 7.12%, when using State machine A. I also found that the Spice\_FP has a greater incorrect address percent prediction than the espresso int. When using the class state machine, the Spice\_FP had a greater incorrect address percent than the Espresso\_int by 52.13%. Also, the Spice\_FP had a greater incorrect address percent than the Espresso\_int by 51.79%, when using state machine A.

5. Concluding remarks

In this project I learned how useful the prediction scheme can be in processors. A more effective prediction scheme dramatically increases the performance of the processor. Using dynamic branch prediction in the pipeline processor is more effective than static prediction because branches in programs have a dynamic behavior. An effective prediction scheme prevents stalls in the CPI. The prediction scheme used a BTB to reduce the number misses, wrong branch prediction, and wrong address predictions. One reason why stall occur in a pipeline processor, are when a branch is not in the BTB and needs to be added. The second reason why a stall can occur is because of wrong branch predictions involving a wrong address or jump not occurring. Overall, the prediction scheme in processers help prevent stalls, which increases the effectiveness and speed of the pipeline processor.