ECE 585 – Final Project - Spring 2017

HDL Implementation of Dynamic Branch Predictors

V1.0

Due date: May 1, 2017

This is an individual project. Each student will work alone.

1. INTRODUCTION

Branch prediction is used to overcome the fetch limitation imposed by control hazards in order to expose instruction level parallelism (ILP). Branch prediction can be thought as a sophisticated form of prefetch or a limited form of data prediction that attempts to predict the result of branch instructions so that a processor can speculatively fetch across basic block boundaries. Once a prediction is made, the processor can speculatively execute instructions depending on the predicted outcome of the branch. If branch predictions rates are high enough to offset misprediction penalties, the processor will have better performance. Without branch prediction, such a processor must stall whenever there are unresolved branch instructions. This imposes a substantial penalty on performance of processors.

There are many branch prediction schemes proposed. **Static Branch prediction** in general is a prediction that uses information that was gathered before the execution of the program. Simplest predictors are to predict that the branch is always taken (MIPS) or to predict that the branch is always not taken (Motorola MC68000). **Dynamic Branch Prediction** on the other hand uses information about taken or not taken branches gathered at run-time to predict the outcome of a branch. Some examples for dynamic predictors include One-level branch predictors, correlated predictors and tournament predictors.

2. YOUR ASSIGNMENT

Your task in this project is hardware implementation of several **dynamic branch predictors** in VHDL or Verilog and compare their performance. Branch prediction schemes are:

i. 1-bit Branch Prediction
ii. 2-bit Branch Prediction (2-bit saturating counters)
iii. (4,2) correlating predictor (4-bit global branch history register and 2-bit counters)

2.1 1-bit Branch Prediction:

- Uses Branch History Table (BHT): Lower bits of PC address index a table of 1-bit values.
- Prediction based on whether or not branch taken last time (T-Taken, N – Not Taken)

2.2 2-bit Branch Prediction:

Uses 2-bit saturating counters as shown in Figure 1.

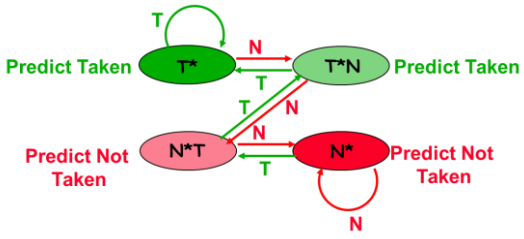


Fig. 1. 2-bit Saturating Counter

Branch History Table is now a table of 2-bit Predictors (saturating counters) indexed by PC address of Branch as shown in Figure 2.

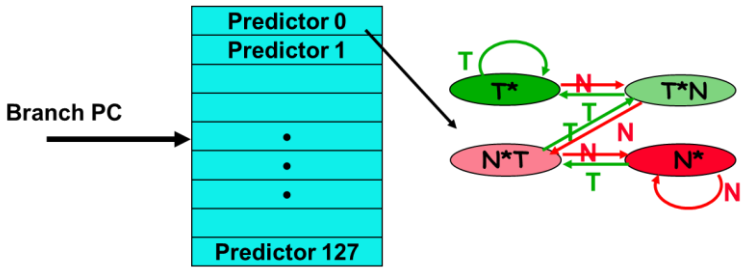


Fig. 2. 2-bit Branch History Table

2.3 (m,n) Correlating Predictors

Correlating Predictors keep track of the behavior of previous branches and use that to predict the behavior of the current branch. For example, a (2,1) correlated branch predictor:

- Uses the behavior of the last 2 branches to choose from 2^2 different predictions
- Uses a 1 bit predictor for each of the 4 prediction buffers

For a (2,n) branch predictor, the last two branches are relevant and this can be implemented using 2-bit shift register as shown in Fig. 3.

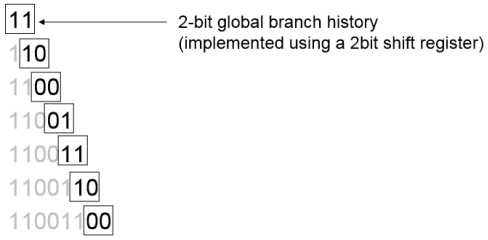


Fig. 3. Global Branch History implemented with 2-bit shift register

Fig. 4 shows the implementation of a (2,2) predictor. Last 2 branch outcomes select between 4 predictions of next branch, updating just that prediction. Each entry in the local predictor is the 2-bit saturating counter value (incremented with each T and decremented with each NT.)

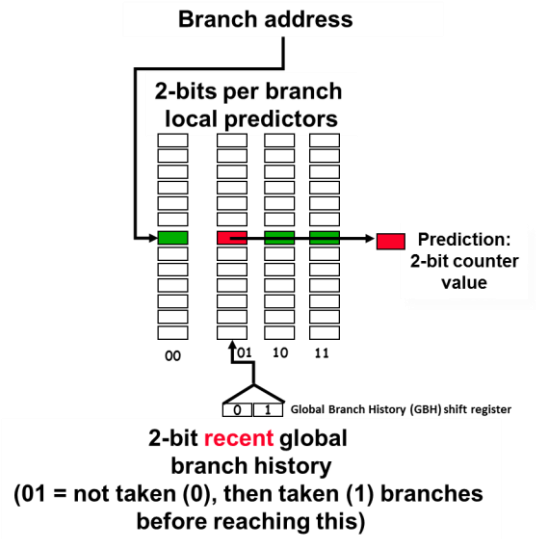
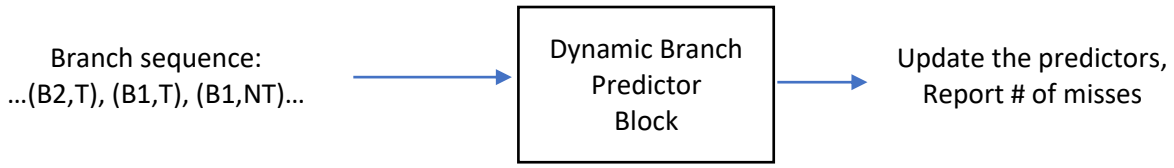


Fig. 4. (2,2) predictor implementation

In this project, you will build a **(4,2) correlating predictor**. Therefore, 4 previous branch outcomes need to be monitored. A 4-bit shift register is required for GBH. Each branch will have 2^4=16 predictions (16 columns). Prediction value is based on the 2-bit saturating counter which is initialized to 0 (i.e. NT predicted).

2.4 Implementation

You are only responsible for branch predictor implementation. Dynamic predictor block will be built with counters, shift registers and multiplexors. You don't need to implement any CPU datapath. Therefore, the input to the system will be a sequence of branch number (id #) and the correct corresponding outcome for a given program. You can generate this sequence in any way, (C code, Matlab code, python script etc…).



An example is given in the following table. This can be used as input to your dynamic branch predictors.

Branch #	Outcome:
1	T
1	T
1	T
1	NT
2	T
1	T
1	T
.	.
.	.
.	.

2.5 Test Program:

Consider the following code which has **2 unique branches** and **~4,000 branch decisions** in total.

```
int c;
int main () {
        int    i, j;
        for (i=0; i<1000; i++) {
                for (j=0; j<4; j++) {
                        c++;
                }
        }
        return c;
}
```

You have four main tasks for the test program:

1. Convert it to the assembly code in MIPS language.

2. Generate all branch decisions using an external program/script.

3. Use this generated sequence as input to you HDL architecture.

4. Find out number of mispredictions for all three predictor schemes; 1-bit predictor, 2-bit predictor, and (4,2) predictor and analyze their performance.

For HDL implementation, use Modelsim PE (Student Edition) software from Mentor Graphics. You can download the software from:

https://www.mentor.com/company/higher_ed/modelsim-student-edition

Please check the **Modelsim Tutorial** posted to the blackboard webpage.

3. DELIVERABLES

i. MIPS assembly code for the test program
ii. Script for generating the branch outcome sequence
iii. HDL source code for each branch predictor implementation
iv. HDL simulation results and waveforms for each branch predictor implementation
v. Table comparing # of mispredictions

4. REPORT

Project reports are due on Monday, May 1st, 2017 at midnight. ***Only blackboard submissions are accepted***. You also need to upload your HDL (Verilog or VHDL) files as well. Your project will be graded on how well your design works, the efficiency of your design, the level of detail of your design, improvements or extra features in your design, the quality and correctness of your report, and the thoroughness of your testing methodology.

The report should be limited to 15 pages at most and be legible when printing on letter-size papers. Please use common sense to format your report, e.g. report with small fonts/figures will not be graded.

All the writings, results, codes, and screen shots should be by yourself. COPY without proper CITATION, and extensive COPY from other materials including but not limited to project instructions and textbooks, will be treated as PLAGIARISM and called for DISCIPLINARY ACTION. You should clearly separate your contribution from existing works, e.g. to separate your implementation from the implementation that is already given. NEVER share your reports with others.

The following sections are recommended as an effective way to organize your writing in your reports.

- **Abstract**

In less than 100 words, briefly discuss your contributions in the project.

- **Introduction**

Summarize the motivation of the project. Highlight the engineering and design challenges in this project and the methods to overcome these challenges.

- **Background**

  Give concise descriptions of the branch prediction. Note that you should cite various references properly, e.g. the project instructions and the textbook.
- **Architectural Exploration of dynamic predictors**

  Discuss the trade-offs among the different dynamic branch predictors.
- **Functional Validation and verification**

  Discuss the approaches taken to validate and verify your designs, e.g. testing and the waveforms. Justify your claims of correctness with simulation results.
- **Results**

  Discuss and explain the differences of the designs in terms of performance and cost. Create charts/tables to tabulate the performance results.
- **Conclusion and Future Work**

  Summarize your contributions and discuss possible future works.
- **Appendix**

  HDL code/Simulation screen shots/results listing.
- **References**

Good luck and have fun!