ECE 486/586

Tournament Branch Predictor

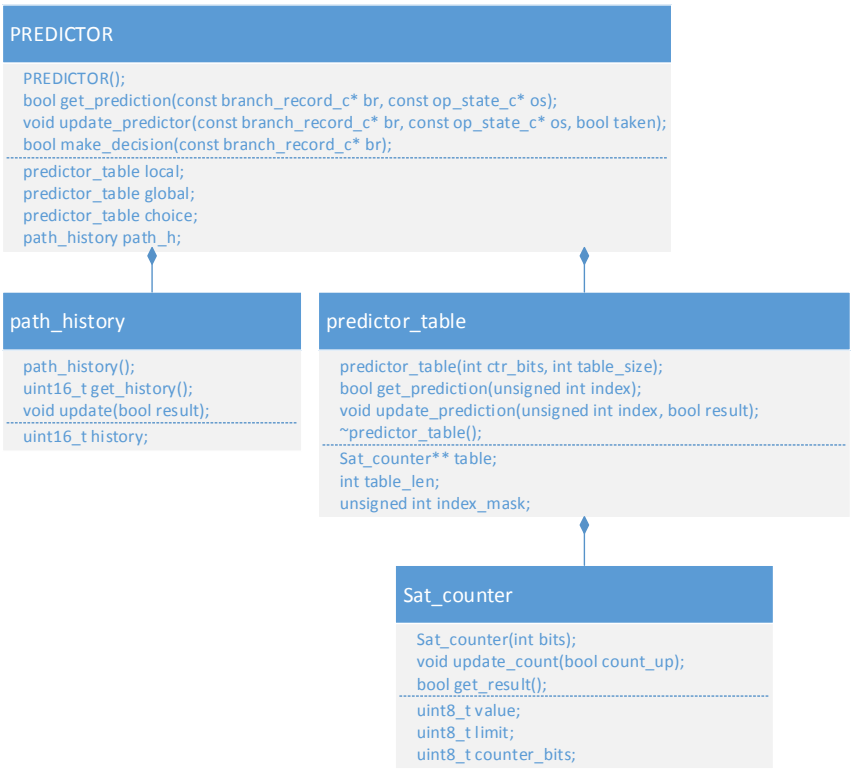
Jordan Fluth, Brett Dunscomb, Scott Lawson and Robert Gaskell

Contents

Program Design

A tournament branch predictor is by nature a state machine. The team elected to write the program in C++ and take an object-oriented approach to maintain state between function calls. The implementation of the tournament branch predictor is based on the design of the Alpha 21264 microprocessor's branch predictor. See Algorithms section for differences between the simulation and the Alpha 21264 predictor.

Class Diagram

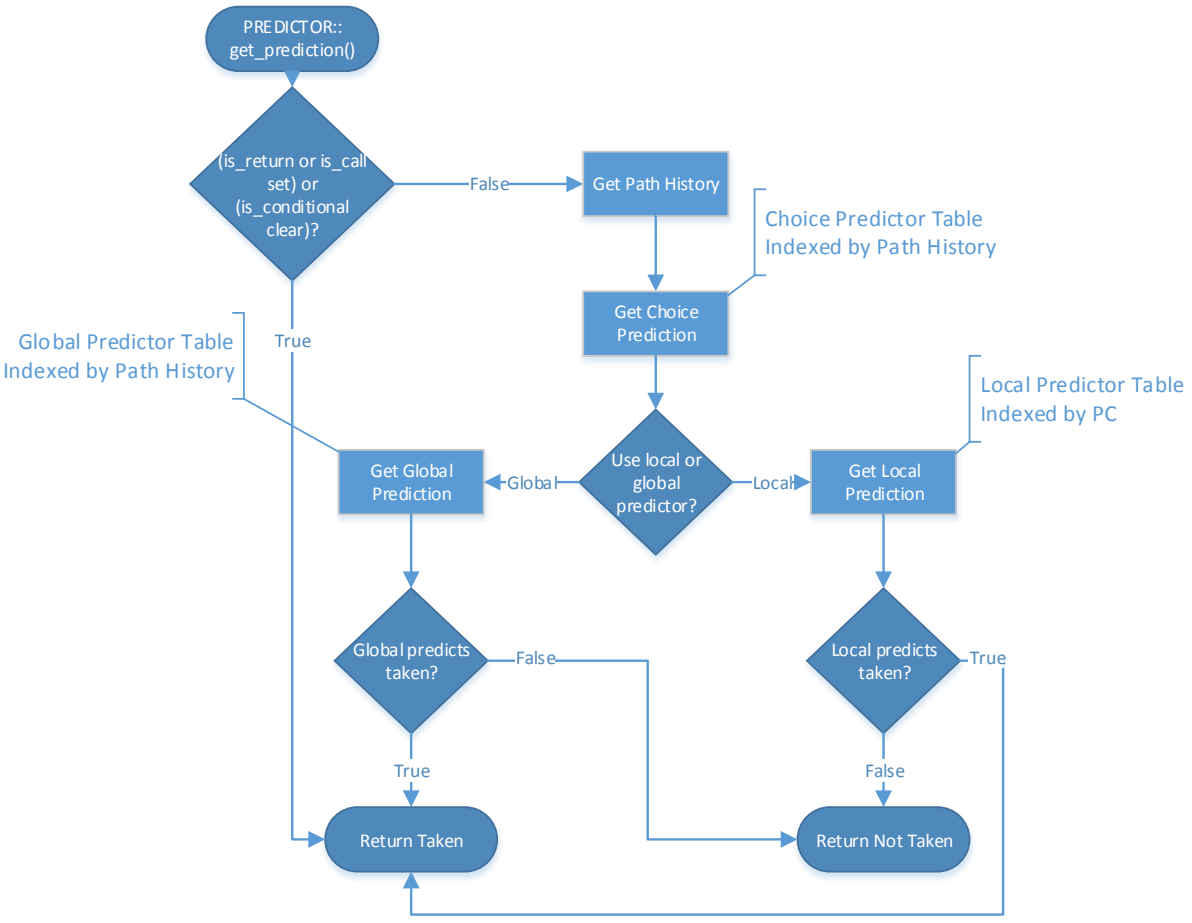


Algorithms

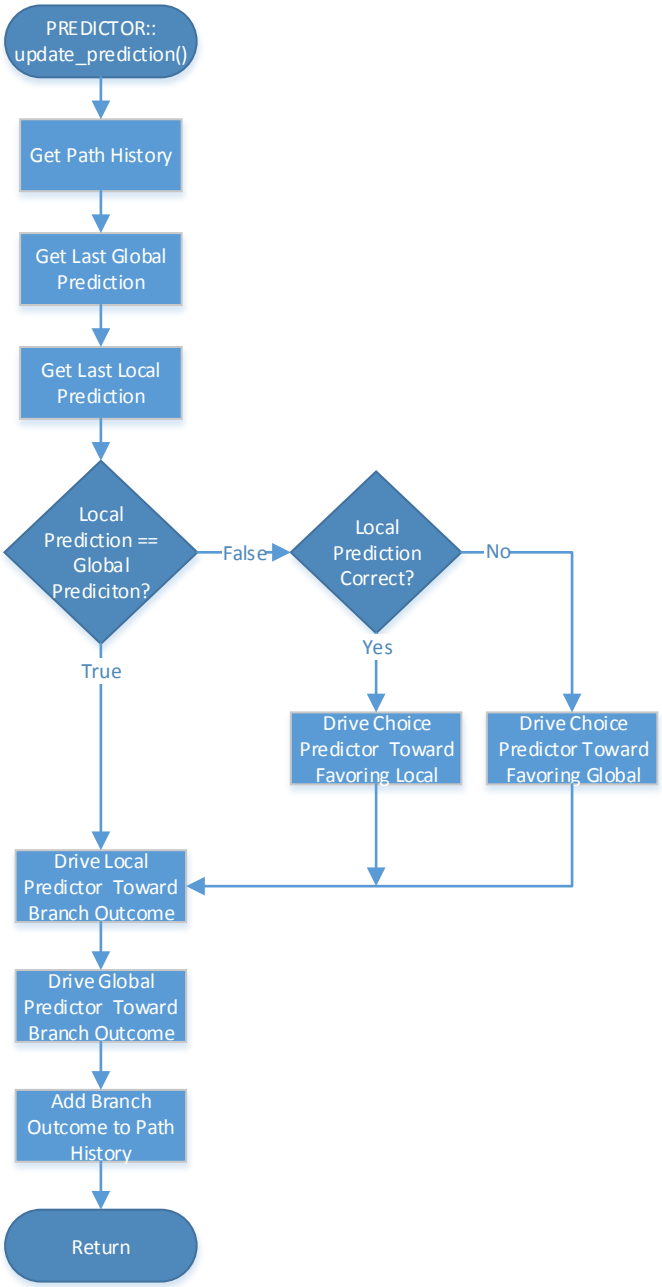
The algorithms used are based on the Alpha 21264 branch predictor. The individual predictors are saturating counters. A table of 1024 three-bit counters make up the local predictor table, and the choice and global predictors each have 4096 two-bit counters. There are two significant differences between the simulated branch predictor and the Alpha 21264 branch predictor:

1. The local prediction table is index directly by the program counter, rather than a local history table as is done by the Alpha 21264 branch predictor.
2. In the simulation, the path history is not speculatively updated with branch predictions. It is only updated when the branch outcome is known.

Get Prediction



Update Prediction



Space Budget

Structure	Contents	Size (Bits)
Choice Predictor Table	4096 x 2-bit Counters	8192
Global Predictor Table	4096 x 2-bit Counters	8192
Local Predictor Table	1024 x 3-bit Counters	3072
Path History	1 x 12-bit Register	12
Total	19468 bits ≈ 2.38 KB	

Program Usage

Program usage and options are defined by the framework, not by the code generated by the branch predictor team. See framework README for instructions.

Benchmark Results

Benchmark	Mispredict Rate
DIST-FP-1	4.187%
DIST-INT-1	8.572%
DIST-MM-1	8.597%

Test Plan

The branch predictor simulator was tested by creating an input file (default.ascii) that tests the following conditions:

Condition Number	Condition
1	Global Prediction
2	Local Prediction
3	Global Saturation High
4	Global Saturation Low
5	Local Saturation High
6	Local Saturation Low
7	Selector Saturation High
8	Selector Saturation Low
9	Special Case
10	Path History Indexing
11	Mispredict Not Taken
12	Mispredict Taken
13	Local Aliasing

For testing, the framework is bypassed and default.ascii is read in by main() in tester.cpp. Each line in default.ascii specifies all of the information found in a branch_record_c class, and get_prediction() and update_predictor() are called in the same manner as in the framework. As lines are acted on, their effects are observed through a debugger. The resulting states of each predictor table are compared to the table Appendix A. Any differences in table state indicate an error condition.

Source Code

The branch predictor source code is attached, and can be found here:
https://github.com/JustRob83/branch_predictor

Team Member Roles

Jordan Fluth	path_history class
Brett Dunscomb	Sat_counter class
Scott Lawson	PREDICTOR class, predictor_table class
Rob Gaskell	Test Plan