Lab Project 3: Dynamic Branch Predictors
Due Monday, Dec. 5 before class

Introduction

In this project, two dynamic branch predictors will be implemented in C/C++: a gshare predictor and an Alpha 21264 predictor. The performance of the predictors will be evaluated on several different program traces. The performance metric used is the misprediction rate. You will be given a standard interface and a set of program traces. Several functions to get the predictor to operate correctly are expected to be implemented by the students.

This project must be completed individually.

Project Details

A standard C/C++ interface is provided and can be downloaded in Canvas. The job of students will be to implement several functions (init_predictor(), make_prediction(), train_predictor() in predictor.c) in this interface to simulate different branch predictors. Upon implementing the required functions, you will compile the branch prediction simulator (via a makefile) and test your predictor on several different traces. These traces were generated from running real programs and tracking if branches were taken or not taken. The trace files themselves are huge text files but are in a very simple format. Each line consists of the address (PC) of the branch instruction and whether the branch was taken (1) or not taken (0). The trace files have been compressed with bzip.

There are two specific branch predictors that you must implement: gshare and Alpha 21264. The description of the two predictors can be found in [1] and [2].

For the gshare predictor, the number of bits for PC and global history is 14bits. Thus the branch history table has 16K entries and each entry is a 2-bit saturating counter. The total memory budget is (32K+14)bits

The parameters of the Alpha 21264 predictor are shown in the following table. The total memory budget is (29K+12)bits.

Local History Table	1024 entries, indexed by PC; each entry is a 10-bit history of a specific branch's set of previous outcomes
Local Prediction Table	1024 entries, indexed by the 10-bit history from the local history table, each entry is a 3-bit saturating counter
Global History Table	A single 12-bit register that encodes the last 12 branch outcomes
Global Prediction Table	4096 entries, indexed by the global history register, each entry is a 2-bit saturating counter
Chooser Prediction Table	4096 entries, indexed by the global history register, each entry is a 2-bit saturating counter (00-01: Use local table, 10-11: Use global table)

Initialize all tables and the global history to all 0s.

You should implement the predictors only in predictor.C and predictor.h. Other parts of the interface should not be changed. Once the implementation and testing of a predictor is done, rename the two files to **predictorg** or **predictor21264**.

Please read the READEME file of the standard interface on how to make the executable and test the predictor.

Project Deliverables

Report:

Provide a brief introduction of each predictor and a table showing the misprediction rate for each individual trace and the arithmetic mean over all traces. Also, provide a graph comparing the performance of the two predictors on each of the traces.

A note on graphs: Graphs are supposed to make large amounts of information comprehensible very quickly. This means they must clear, uncluttered, easy to decipher, and well labeled. Perhaps the most important and most frequently broken rule for graphs is "Label your axes." Your graphs MUST have properly labeled axes, or they will receive no credit.

Tarball:

Please place your report, two executables, and source code (**predictorg.C** and **predictorg.h, predictor21264.C** and **predictor21264.h**) in a gzipped tarball using the following naming convention: "lastname-cse331-F14-P3.tar.gz". Upload the tarball in Canvas.

References

[1] McFarling, Combining Branch Predictors, WRL TN-36. Good description of local, correlating, gshare, and combining predictors.

[2] Kessler, The Alpha 21264 Microprocessor, IEEE Micro, 1999. Uses variant of McFarling's combining (tournament) predictor.