Project 2: Branch Simulator

1 Overview

This lab will help you better understand branch predictors.

2 Logistics

This is a partner project. You may use C or C++ for this assignment, and you should only rely on standard libraries. The base GitHub repo has been updated with the necessary files to support this assignment.

3 Branch Prediction Simulator

You will be implementing a branch predictor simulator. This simulator is based on a 2-bit saturating counter, aka a "Smith" counter. [1]

Arguments:

-s <val> : Log2 of the predictor size

A branch instruction's PC address is mapped to a predictor and branch target buffer (BTB). The reference ignores the 3 least significant bits, and then uses the next s bits to lookup a single BTB entry and as input to find a specific predictor (depending on the model).

-b <val> : Size of BHR register in bits / length of history (default 0)

The BHR register is used in models other than 2-bit.

-g <val> : Type of predictor being modeled (common/branch.h). You are required to implement at least two models (2-bit and either GSHARE, GSELECT, or Yeh-Patt):

0 : 2-bit counters
1 : GSHARE
2 : GSELECT
3: Yeh-Patt

-p <val> : Number of processors being modeled

Each counter will default to state 01. As an exercise for the reader, the initial value for each BTB entry does not matter.

For simplicity, a branch that is predicted not taken is indicated by "predicting" the current PC address + 4 bytes.

4 Interface

You should only modify files within your branch's subdirectory, along with the CMakeLists.txt. Modifying other files may result in simulator instability or merge conflicts at later points.

The following functions may need further explanation:

```
branch* init(branch_sim_args* bsa)
```

> Uses the arguments in bsa to initialize the branch simulator, including the allocation of any structures. On success, returns a pointer to the allocated branch struct, including the interface functions. On failure, returns NULL.

```
uint64_t branchRequest(trace_op* op, int processorNum)
```

> op - Branch operation being executed by the processor
>
> processorNum - currently 0

Unlike the cache, the delay time for a branch does not directly depend on the predictor design itself, but rather the rest of the processor. A mispredicted branch is not detected until the branch's instruction completes its execution. Therefore, the predictor only returns the predicted PC address. The operation contains the true next address, so the predictor and processor can both determine the accuracy as well as any necessary updates.

5 Running the Simulator

Example commandline:

```
$ /usr/bin/time ./cadss-engine -s ex_branch.config -t ls.trace -b refBranch

Ticks - 192100

{"real":0.04, "user":0.04, "sys":0.00, "mem":1736 }
```

6 Experiments

For each trace in the trace directory*, study the impact of varying the design parameters:

1. You have a total budget of 512 smith counters. With GSELECT, the size of the GHR will limit the predictor size. With GSHARE, the BHR cannot exceed the predictor size.

2. Compare the prediction accuracy with and without the BHR, as well as the benefit of increasing its size.

3. Other design parameters / explorations are possible.

7 What to hand in

1. A short report (PDF) on the design and experiments for your simulator, to be submitted to Gradescope. About 2 pages in "technical paper" formatting (suggested but not required - https://www.acm.org/publications/proceedings-template).

2. The commented source code for the simulator program itself. Typing "cmake .; make" should be sufficient to build your simulator. We recommend committing your work regularly and downloading the zip from github as your submission file.

3. **Autolab will use grace days for late submissions. After the deadline, Prof Railing will review submissions and provide feedback. Any simulator that is at the "experiment stage", will have the opportunity to resubmit.**

8 Grading (approximate)

0%	You do not hand in anything by the deadline
+40%	Your simulator doesn't run, does not work, but you hand in significant commented code
+22%	Your branch simulator runs (12%), Your simulator supports 2-bit and at least one other model (10%)
+8%	Your simulator's output "matches" the reference
+10%	You ran all experiments and analyzed the difference in branch models / parameters
+20%	The report demonstrating key experimental results, simulation design, etc.

[1]
https://courses.cs.washington.edu/courses/cse548/11au/Smith-A-Study-of-Branch-Prediction-Strategies.pdf