CMU 18-741: Advanced Computer Architecture

Handout 9: Branch Prediction

**** Due 10am, 12/6/2005 ****

1. Introduction

Branch prediction continues to be an ongoing area of research and many new ideas are being proposed today. In this project, you will (1) design a basic tournament predictor based off the Alpha 21264 and (2) participate in a branch prediction competition. The objective of the competition is to design a branch predictor that maximally reduces mispredictions given a fixed storage constraint.

This project must be completed in groups of 2 or 3 students. After forming a group, one member should email the TAs with the names and email addresses of all members by November 8. Use the subject heading "Project 4 Group".

2. Branch Prediction

In this project, you will design branch predictors (basic and competition) to optimize for a set of benchmarks under a given storage constraint. Your branch predictors should provide a branch decision when presented with the program counter and other info about a branch from a trace of branch instructions. After each prediction, your predictor then will be given the actual branch outcome for learning/training.

We provide a driver loop that manages trace processing used to drive your predictors. There will be two sets of traces: basic and competition. The basic "non-competition" traces are available in gzipped format under `/afs/ece/class/ece741/.vol1/` for your perusal and possible off-line analysis. They consist of a suite of integer, floating-point, multimedia, and server workloads. These traces also include branches executed within the operating system. **Half of the competition traces will be released to you at a later date, and the other half will be revealed only after submission.**

If you wish to write your own traces, branch traces are formatted as follows:

```
[PC] [Branch Target] [Is Indirect] [Is Conditional] [Is Call] [Is Return] [Is Taken]
```

The following is an example branch trace:

```
40ee8a 40eeb8 0 1 0 0 0
40eea9 40eed3 0 0 0 0 1
40eed7 40f306 0 1 0 0 0
40efb4 488420 0 1 0 0 0
40f308 40f34f 0 1 0 0 0
```

Note: the Branch Target is additional information that is included but is not always necessary or useful for prediction.

Altogether, there are four functions your predictors must implement:

void initPredictor()
int getPrediction(branch_record_t *branch)
void updatePredictor(branch_record_t *branch, bool is_taken)
void dumpStats()

initPredictor( ) is called before any processing begins for you to initialize your data structures.

int get_prediction(branch_record_t *branch) is called once every branch in the trace processed by our driver. **Do not free the branch_record_t pointer!** The argument provided is a branch record that contains information on the branch PC. The following is a table with the branch record fields:

uint instruction_addr	The branch's program counter
int is_indirect	1 if the target is not PC-relative; 0 if it's PC-relative
int is_conditional	1 if the branch is conditional type
int is_call	1 if the branch is a call type
int is_return	1 if the branch is a return type

If the branch is predicated to be taken, return 1, otherwise return 0.

void updatePredictor( ) is called right after get_prediction() to return the actual outcome to your branch predictor for training.

void dumpStats( ) is called after all of the branch traces have been processed. Use this function to print out any internal statistics at the end of the simulation.

Additional details will be included in the source package for the skeleton code. The following code fragment gives you an idea of the sequence of events in the simulation loop:

```
initPredictor();

while(…trace is not EOF…) {
        branch_record_t *branch = a single branch from the trace file
        prediction = getPrediction(branch)
        … check your prediction
        … count statistics
        … notify your predictor of the outcome
        updatePredictor(branch, isTaken);
}

… print out stats including your misprediction rate …
dumpStats();
```

3. Alpha 21264 Tournament Predictor (basic)

The first phase of your project is to implement a "basic" branch predictor based off the Alpha 21264 Tournament Branch Predictor. Please refer to Kessler, R. E., "The Alpha 21264 Microprocessor", IEEE Micro, March/April 1999, pp. 24-36 (available at ieeexplore.ieee.org).

The following is a summary of the predictor design:

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 register	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

The local history table maintains history entries for individual branches. Each 10-bit history (which encodes the last 10 outcomes for this branch) is used to index into a local prediction table, which maintains entries that have 3-bit saturating counters. When a 3-bit saturating counter reaches 100_2 or above, a "taken" prediction is made.

The global history register maintains a single 12-bit history (which encodes the outcomes for the previous 12 branches) used to index into the global prediction table, which has 2-bit saturating counters per entry. When a 2-bit saturating counter reaches 10_2 or above, a "taken" prediction is made.

The local and global predictions are compared against one another, and if they agree, the prediction is made. However, if they disagree, they should consult the chooser prediction table, which is also indexed by the global history register. If the saturating counter is 10_2 or above, the global predictor's decision should be used; otherwise, the local predictor's decision is picked. The chooser prediction table is updated depending on the outcome of disagreeing predictors. Depending on whether a predictor chose correctly, the saturating counter is incremented or decremented.

Note some specific implementation decisions are open to your implementation. If your predictor is designed correctly, it should have an overall branch misprediction rate of at most **5.5% on the basic traces** (see section 4 on how we compute the branch misprediction rate).

4. Competition Branch Predictor

In this phase of the project, you must design a competition branch predictor that minimizes the branch misprediction rate across all of the benchmarks with a maximum storage budget of **33792 bits.** Your score will be penalized for going over. The following is a summary of the restrictions your predictor should abide by:

Design constraints
All storage tables must have power-of-2 entries
8-way set-associativity and less do not incur extra cost over direct-mapped tables
If a table is greater than 8-way set-associative, its cost is multiplied by 2. Such tables cannot have more than 128 entries.
Random replacement policy (if used) must be reproducible
Extravagant logic functions (such as dividing/multiplying by non-constants or non-powers of 2) are not allowed
There is no extra cost for dual-ported memory. If memory is more than dual ported, cost is multiplied by (# of ports – 1)

The measure of success is the branch misprediction count aggregated for all of the benchmarks—that is, the branch misprediction rate if you were to run each benchmark under the `/afs/ece/class/ece741/.vol1` folder once, in sequence. This means adding up all of the branch mispredictions for all of the benchmarks and dividing over the total number of branches for all of the benchmarks. **Do not average the branch misprediction rates individually.**

Your misprediction rate will determine the competition portion of your grade (see section 6). **Half of the competition traces will be released to you at a later date and the other half will not be known before submission.**

Feel free to re-implement any existing branch predictor design. Portal.acm.org and ieeexplore.iee.org are excellent places to start. If you do decide to leverage any prior work, please include proper citations in your report.

5. Requirements

The following is a list of what we expect for a completed project:

- All source files
- Makefile
- Readme file
- Design review answers (see section 7)
- Report
- Presentation (more details later)

You may use any operating system for your project, but your project will be tested on the Greek cluster (e.g. alpha.ece.cmu.edu), so make sure your final submission works on these machines. You may use any language as long as you can provide a wrapper that will interface to our object code developed for C, but you must use the provided predictor_sim.o.

When building your predictors, we will be executing:

```
make basic
make competition
```

The executables generated should be "**predictor_basic**", and "**predictor_comp**", respectively. Each of these executables should run standalone and execute as follows:

```
        make basic
        zcat tracefile.gz | ./predictor_basic
```

Please include a README file with your code, which should contain the names of the group members, and e-mail addresses at which you can be contacted in case there are any problems. If there are any other comments about the project (e.g. project does not work), please include it in the README file. A sample README file can be found at `/afs/ece/class/ece741/project4/`.

6. Grading

Your project will be graded according to the following:

40%	Both predictors build and run without crashing
20%	Alpha 21264 Tournament Branch Predictor with branch misprediction rate of at most 5.5% for basic non-competition traces
20%	Competition Branch Predictor
20%	Design Review
20%	Report
30%	Presentation
150%	Total (this is worth 1.5x previous projects)

To achieve the first 40%, the make commands (see section 5) should generate the correct executables and execute benchmarks without crashing. 20% of your grade will depend on a demonstration that your basic branch predictor (Alpha 21264 Tournament Branch Predictor) has a branch misprediction rate of at most 5.5% on the basic non-competition traces.

For the competition score, if you are in the top quartile for all **competition traces (released and unreleased)**, you will receive the full 20%. If you are in the second quartile, you will receive 15%. If your misprediction rate is at least as good as the TAs' branch predictor misprediction rate, you will receive at least 10%. If your branch predictor misprediction rate is worse than the TAs' branch predictor, you will receive 0%. (To be fair, the TAs will not optimize their branch predictor for the unreleased traces.) **Half of all competition traces and the TAs' score will be released in two weeks**. If you are a top-scoring team, you will be rewarded with a token prize and bragging rights.

Finally, the remainder of your grade will depend on the quality of your design review answers as well as a report and presentation, documenting the design and results of your predictor. The report should include design methodology and how you arrived at your competition predictor. Clearly outline how you used your storage budget. Presentations will be given on December 6 and December 8. More details will be provided later.

7. Design Review

1. List five reasons why Project 4 does not accurately reflect true branch prediction.
2. What is aliasing? Describe cases where aliasing is desirable and undesirable.
3. Does increasing the number of bits used for the saturation counter always improve performance? Give an example supporting your position.
4. There are 3 two-level adaptive branch predictors named PA*. For each, describe a trace where it has least mispredictions.

8. Submission

The final submission will be done electronically. Copy all relevant files to the directory in `/afs/ece/class/ece741/project4/submission/your_email` which matches the email of the student who e-mailed the group names. The directory is unlocked until the deadline so you can make changes until then if you submit early.