Project rules

1. All students are encouraged to work in teams of two, using pair programming. Pair programming means programming where two people sit at the same workstation, writing the code collaboratively.

2. ECE 521 students will have additional parts to do; therefore both members of a pair should be from the same class (463 or 521).

3. You may not work with the same partner on more than one project this semester.

4. You must register your partnership by posting on the Pair-Programming Partners message board, under the topic Project 2.”

5. Sharing of code between teams will be considered cheating, and will be penalized in accordance with the Academic Integrity policy.

6. It is acceptable for you to compare your results with other groups to help debug your program. It is not acceptable to collaborate on the final experiments.

7. You must do all your work in the C/C++ or Java languages. C++ (or Java) is encouraged because it enables straightforward code-reuse and division of labor.

8. Homework will be submitted over the Wolfware Submit system and run in the Eos/Unity environment.

Project description

In this project you will construct a branch predictor simulator and use it to design branch predictors.

Specification of the simulator

1. *(All students)* Model a local branch predictor with parameters \((m, n)\)

   \[ m \] = the number of low-order PC bits used to index into the local branch-history table.

   \[ n \] = number of bits in the local branch-history register.

Refer to the diagram of the local branch predictor at the top of the next page.
The table of local branch history registers is indexed by $m$ low-order bits of the address of the branch instruction (i.e., PC value when the branch is fetched, not the branch target). These $m$ bits are the least-significant bits of the address, no including the two least-significant bits of the PC, since these are always zero (that is, use bits $m+1$ through 2 of the PC).

The value in a branch-history register is used to index into an array of counters. Thus, if $n$ is the number of bits in a branch-history register, there will be $2^n$ counters.

The branch history registers are all initialized to zero at the beginning of the simulation.

Every time a branch is predicted, the branch outcome is shifted into the most-significant bit position of the local branch-history register containing the history of that branch. (Note: In real hardware, we would not be able to shift the branch outcome in until the EX stage of the pipeline. However, this is just a simulation, and we read the branch outcome from the trace file, so we can use it right away.)

There are no tags and no tag-checking (no miss signals for the predictor).

Each of the $2^n$ counters contains 2 bits.

(a) The counter is incremented when the branch is taken and decremented when it is not taken.
(b) The counter saturates at 0 and 3
(c) The counter is updated after the prediction is made.
(d) If the counter value is greater than or equal to two, the branch is predicted taken otherwise it is predicted not taken.
(e) All counters should be set to 2 when the simulation begins.

2. (All students) Model a global branch predictor with parameter $n$.

\[ n = \text{number of bits in the global branch history register.} \]
• Every time a branch is predicted, the most recent branch outcome is shifted into the most-significant bit position of the global branch-history register for a branch.
• The global history register is initialized to zero at the start of the simulation.
• There are no tags and no tag checking (no miss signals for the predictor).
• Each of the $2^n$ counters contains 2 bits.
  (a) The counter is incremented when the branch is taken and decremented when it is not taken.
  (b) The counter saturates at 0 and 3 (the counter will never go below zero or above three).
  (c) The counter is updated after the prediction is made.
  (d) If the counter value is greater than or equal to two, the branch is predicted taken otherwise it is predicted not taken.
  (e) All counters should be set to 2 when the simulation begins.

(3) \textit{(ECE521 students only)} Model a hybrid predictor that selects between the local and the global branch predictors using a choice predictor.

• The choice predictor is a table of $2^k$ two-bit counters.
• If the choice counter is either 2 or 3, it selects the global branch predictor; otherwise it selects the local branch predictor.
• The choice counter is updated using the following rule:

<table>
<thead>
<tr>
<th>Result from predictors</th>
<th>Both incorrect or both correct</th>
<th>Local correct, global incorrect</th>
<th>Global correct, local incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choice counter update policy</td>
<td>No change</td>
<td>Decrement</td>
<td>Increment</td>
</tr>
</tbody>
</table>
(a) The counter saturates at 0 and 3 (the counter will never go below zero or above three).
(b) All choice counters are initialized to 1 at the beginning of the simulation
(c) The index for the chooser table is bit $k+1$ to bit 2 of the address of the branch (the value of the PC when the branch is encountered).
(d) Only the predictor that was selected (local or global) is updated with the branch behavior.
(e) The global history register will be updated even if the local branch predictor is selected.

You should be able to select any of the predictors (local, global, or hybrid) based on a command-line parameter. The parameters of the simulation are:

- $\{m, n_1\}$ for local branch predictor.
- $\{n_2\}$ for global branch predictor.
- $\{\{m, n_1\}, \{n_2\}, k\}$ for the hybrid predictor.

### Input
The simulator reads a trace file in the following format:

```
0x<hex branch PC> t/n
0x<hex branch PC> t/n
...
```

where `<hex branch PC>` holds the hexadecimal form of address of the branch instruction in memory.

- “t” indicates that the branch is actually taken.
- “n” indicates that the branch is not taken.

**Example**

```
1100023b t
0a050201 n
```

### Output
The simulation output should include:

1. The number of branches encountered.
2. The number of branch mispredictions (predicted taken when not taken, predicted not taken when taken).
3. Misprediction rate ($\#$mispredictions / $\#$branches).

### Program interface requirements

1. Your code must be able to run in the Unix environment on the Eos/Unity system.
2. A makefile must be provided.
3. A “make” will create an executable file--“brn_predictor”.
4. The program must be executable from the command line, using the following format.
local:
brn_predictor [tracefile name] [m1] [n1]

global:
brn_predictor [tracefile name] [n2]

hybrid:
brn_predictor [tracefile name] [m1] [n1] [n2] [k]

Example:
For local branch predictor, eos% brn_predictor gcc_trace.txt 12 8
The first parameter is the number of low-order PC bits \(m = 12\).
The second parameter is the number of bits in history register \(n = 8\).

Experiments

For the traces given in the course locker (directory
/afs/eos/courses/ece/ece521/common/www/homework/projects/2/traces), use
your simulator as follows:

(1) **(All students)** Do a preliminary design-space search using the following
configurations.

  *Local Branch Predictor*: \(5 \leq m \leq 15, 2 \leq n \leq m\), where \(n\) is even.

    - Produce one graph for each benchmark.
      - \(y\)-axis: Branch-misprediction rate
      - \(x\)-axis: \(m\)
    - Draw conclusions and discuss the results. Compare and contrast with
      (2).

(2) **(All students)** Do a preliminary design-space search using the following
configurations:

  *Global Branch Predictor*: \(5 \leq n \leq 15\)

    - Produce one graph for each benchmark.
      - \(y\)-axis: Branch-misprediction rate
      - \(x\)-axis: \(n\)
      - There should be one curve with data points for each value of \(n\).
    - Draw conclusions and discuss the result. Compare and contrast with (a).

(3) **(All students)** Do an intelligent search of the design space to minimize
misprediction rate and minimize predictor cost in bits given a maximum budget of
(a) 16 kilobytes of storage, and (b) 32 kilobytes of storage. (That is, do one
analysis for 16 KB and another for 32 KB.)

What to include in the report: For each of the benchmarks individually, report the best
two predictors based on the constraints. Justify your choice with graphs and/or
tables and discussion.
4) (ECE521 only)

*Hybrid Branch Predictor:* For the hybrid predictor, find the best hybrid configuration within the maximum budgets of 16 KB, and 32 KB, of storage. Do an intelligent search of the design space to minimize misprediction rate and minimize predictor cost in bits.

*What to include in the report:* For each of the benchmarks individually, report the best two predictors based on the constraints. Justify your choice with graphs/tables and discussion.

What to hand in:

*On Friday, March 31 at 11 PM:* **Project design,** which includes
  - a list of the major data structures and function headers in the program (complete with parameters)
  - a paragraph or two about the approach you will take in writing the code. This should be about one page long.

*On Monday, April 3 at 11 PM:* **Code,** which includes
  - The commented source code of the simulator.
  - The validation runs.

*On Friday, April 7 at 11 PM:* **Report,** which includes
  - The experimental runs.
  - A written narrative of what you found in your experiments, as explained above.

**Grading**

0% You do not hand in (submit electronically) anything by any of the due dates.
+10% You submit a reasonable design by the design due date, and follow that design in writing your simulator.
+10% Your Makefile works, and creates simulator (executable file).
+10% Your simulator can read trace files from the command line and has the proper interface for parameter settings.
+40% Your simulator produces the correct output.
+30% You have done a good analysis of the experiment.