IIT Madras
Department of Computer Science and Engineering

CS6600: July-Nov '24
Project 2: Branch Predictor Simulator
Due: Sunday, October 20 at 11:59 PM

1. Ground rules

1. All students must work alone for this project.
2. Sharing of code between students and/or copying code from public git repositories is considered cheating. The TAs will scan source code through various tools available to us for detecting cheating. Source code that is flagged by these tools will receive the following actions:
   - Zero credits for the project.
   - Final grade will be two grades lower than the actual grade obtained by the student.
3. Students are encouraged to engage in white board discussions, which is an essential aspect of the project.
4. It is recommended that you do all your work in the C or C++ languages. Exceptions (in rare cases) must be approved by the faculty.
5. Students must get the code to work on the provided Docker.

2. Project Description

In this project, you will build a branch predictor simulator and use it to design branch predictors that are well suited for the SPECint95 benchmarks.

3. Specification of the Simulator

Model a *gshare* branch predictor with parameters `{m, n}`, where:

- `m` is the number of low-order PC bits used to form the prediction table index. Ensure to discard the lowest two bits of the PC, since these are always zero, i.e., use only bits `m+1` through `2` of the PC.
- `n` is the number of bits in the global branch history register. Note:
  - `n` is typically less than or equal to `m`, i.e., `n <= m`.
  - `n` may be equal to zero, in which case we have the simple *bimodal* branch predictor.

3.1 Bimodal Branch Predictor (`n=0`)

When `n=0`, the *gshare* predictor reduces to a simple *bimodal* predictor. In this case, the index is based on only the branch's PC, as illustrated in Fig. 1 below.

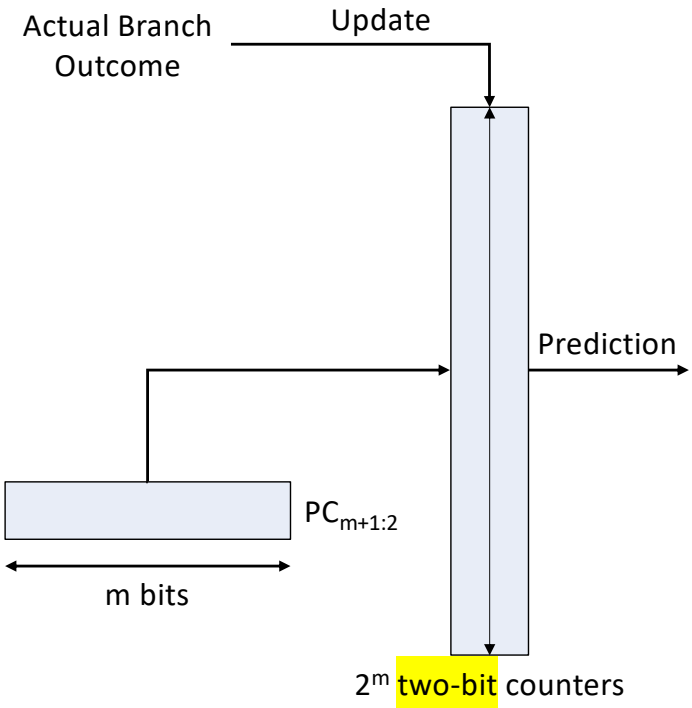

Figure 1. Bimodal branch predictor.

Entry in the prediction table:
Each entry in the prediction table contains a single 2-bit counter. All entries in the prediction table should be initialized to 2 ("*weakly taken*") at the beginning of the simulation.

Sequence of operation:
When you get a branch from the (provided) trace file, the following three steps need to be performed:

1. Determine the branch's index into the prediction table, as shown in Fig. 1.
2. Make a prediction based on the branch's counter value.
3. Update the branch predictor counter based on the branch's actual outcome.

3.2 Gshare Branch Predictor (n>0)

When `n>0`, we have the *gshare* branch predictor with n-bit global branch history register. In this case, the index is based on both the branch's PC and the global branch history register, as shown in Fig. 2 below. The global branch history register should be initialized to all zeroes (`00…0`) and all entries in the prediction table should be initialized to 2 ("*weakly taken*") at the beginning of the simulation.

Sequence of operation:
When you get a branch from the (provided) trace file, the following steps need to be performed:

1. Determine the branch's index into the prediction table, as shown in Fig. 2.
2. Make a prediction based on the branch's counter value.
3. Update the branch predictor counter based on the branch's actual outcome.
4. Shift the global branch history register to the right by 1 bit, and place the branch's actual outcome into the most significant bit position of the register.

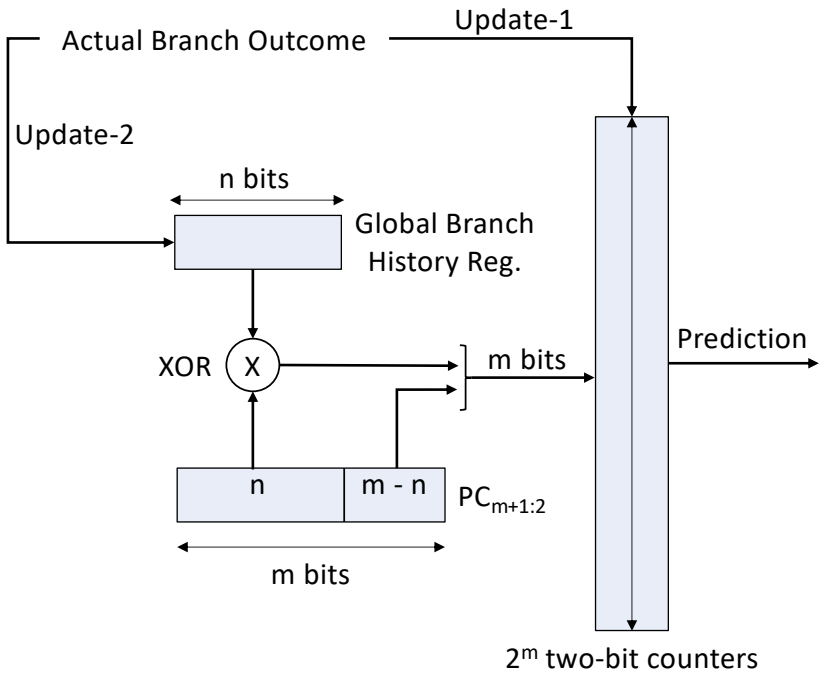

Figure 2. Gshare branch predictor.

4. Inputs to Simulator

The simulator reads a trace file in the following format:

```
<branch PC in hex> t | n
<branch PC in hex> t | n
…
```

Where:

- <branch PC in hex> is the (32-bit) address of the branch instruction in memory. This field is used to index into the predictors.
- "t" indicates that the branch is actually taken.
- "n" indicates that the branch is actually not-taken.

Example trace:

```
00edebe4 t
00ed0a0c n
00dd0bc8 n
```

5. Outputs from Simulator

The simulator outputs the following measurements after completion of the run:

a. number of branches
b. number of branch mispredictions
c. branch misprediction rate (# mispredictions / # branches)

6. Validation Requirements

Sample simulation outputs will be provided to validate the correctness of your simulator. These are called validation runs. You must run your simulator and debug it until it passes all the given validation runs.

Each validation run includes:

1. The branch predictor configuration.
2. The final contents of the branch predictor.
3. All measurements described in Section 5.

Your simulator output must match both numerically and in terms of formatting, because the TAs will only *diff* your simulator's output with the correct output. You must confirm the correctness of your simulator by following these two steps for each validation run:

1. Redirect the console output of your simulator to a temporary file. This can be achieved by placing "> *your_output_file*" after the simulator command.
2. Test whether or not your outputs match properly, by running this unix command:
   diff –iw <your_output_file> <provided_validation_output_file>
   The *–iw* flags tell *diff* to treat upper-case and lower-case as equivalent and to ignore the amount of whitespace between words. Therefore, you do not need to worry about the exact number of spaces or tabs as long as there is some whitespace where the validation runs have whitespace.

6.1 Simulator Compilation and Execution Requirements

You will hand in source code, and the TAs will compile and run your simulator. You must meet the following requirements without fail:

1. You must be able to compile and run your simulator on the provided Docker.
2. You must provide a *Makefile* that automatically compiles the simulator. This *Makefile* must create a simulator named "*bpsim*". The TAs should be able to type only "*make*" and the simulator will successfully compile. An example *Makefile* will be provided to you. Please feel free to modify it based on your needs.
3. Your simulator must accept the following command-line arguments in the specified order.

   To simulate a *bimodal* predictor:
   ./bpsim bimodal <M> <trace_file>
   - M : Number of PC bits used to index into the bimodal table.
   - trace_file : Character string specifying the full name of trace file.

   To simulate a *gshare* predictor:
   ./bpsim gshare <M> <N> <trace_file>
   - M : Number of PC bits used to index into the gshare table.
   - N : Number of global branch history register bits.
   - trace_file : Character string specifying the full name of trace file.

6.2 Compiling and Running the Simulator on the Docker

The following are the steps to be followed to compile and run the simulator on Docker.

Step 1 : Install Docker. For example, if you are installing on Ubuntu based machine, you can follow this: https://docs.docker.com/engine/install/ubuntu/. Docker desktop can also be used; in the latest version it comes with an integrated terminal in which the following commands can be run.

Step 2 : Build the docker image. For this, go to the folder containing Dockerfile and run:
$ docker build -t assign2 .
Where, assign2 is the name for the docker image.

Step 3 : Create a container and run. The command for it is:
$ docker run -it assign2 /bin/bash

Step 4 : Now, we will get a shell inside the docker container. If you run make, it should build the simulation executable.
$ cd Assignment_files
$ make
$ ./bpsim 6 gcc_trace.txt

Note: Docker container is a temporary instance, so any changes made is not saved.

7. Project Experiments and Report

7.1 Bimodal Predictor

Question #1(a):

Simulate the *Bimodal* Predictor configurations for $7 \leq m \leq 12$ using the *gcc* and *jpeg* traces. Plot the misprediction rate (on the Y axis) vs. "m" (on the X axis) for each of the two benchmarks.

Note:

- Produce a separate plot for each benchmark.
- Provide the plot title and clearly label the axes.

Discuss the trends (change in misprediction rate with increasing "m") in each plot and describe the similarities/ differences between benchmarks.

Question #1(b):

For each benchmark/trace, propose a *bimodal* predictor design that minimizes misprediction rate and predictor cost (in terms of storage requirement of the predictor table). Assume that you have a maximum budget of 16kB of storage for the predictor table.

Hint: While looking at the drop in misprediction rate with "m", look for the point at which you start to get diminishing returns. Remember that your goal as an architect is to obtain improved performance at minimal (or reasonable) storage/power overheads.

7.2 Gshare Predictor

Question #2(a):

Simulate the *Gshare* predictor configurations for $7 \leq m \leq 12$, $2 \leq n \leq m$ using the *gcc* and *jpeg* traces. Please simulate only those configurations for which "n" is even. Plot the misprediction rate (on the Y axis) vs. "m" (on the X axis) for each value of "n".

Note:

- Produce a separate plot for each benchmark.
- Provide the plot title, clearly specify the legend, and label the axes.
- For each "n", include only those values of "m" that are legal. For example, if the value of "n" is 10, the legal values for "m" are 10, 11, and 12.

Discuss the trends in each plot and describe the similarities/ differences between benchmarks.

Question #2(b):

For each benchmark/trace, propose a *gshare* predictor design that minimizes misprediction rate and predictor cost (in terms of storage requirement of the predictor table). Assume that you have a maximum budget of 16kB of storage for the predictor table.

Hint: While looking at the drop in misprediction rate with "m" (for a given "n"), look for the point at which you start to get diminishing returns. Remember that your goal as an architect is to obtain improved performance at minimal (or reasonable) storage/power overheads.

8. Submission Requirements

You must submit a single zip file, named <StudentID>_assignment2.zip, which should contain the following:

- **Source code.** You must include all the source code files (.cc/.h or .c/.h)
- **Makefile.** The Makefile needs to be updated to compile all your source code files.
- **Project report.** This should be a single *report.doc* or *report.pdf* file, which contains the plots and the discussions.

9. Grading Policy

The grading policy for this project is described in the following table (Table 1).

Description	Points
Bimodal predictor validation runs (3) + experiments (2).	50
Gshare predictor validation runs (3) + experiments (2).	50

Table 1. Grading policy for the branch predictor project.