COL718 Architecture of High Performance Computers
Assignment 1
Devise a Branch Predictor

In this assignment, you need to devise a branch predictor, that satisfies given area constraints and achieves a given minimum accuracy. This assignment is to be done in groups of two.

Framework

- Download the assignment from `palasi.cse.iitd.ac.in`. The file can be found at `/home/phd/rajshekar/public_html/COL718_assignment1.tar.gz`. Extract it.
- The folder *traces* contains five traces of branch instructions against which your predictor is to be tested. You **DO NOT** have to perform any file operations. The given framework does all that for you. Please concern yourself with only modifying the files *Predictor1200.java*, *Predictor2400.java* and *Predictor4800.java* . No other file is to be modified.
- Run the following commands:
  - `ant clean`: to clean class files and jar files from previous builds.
  - `ant`: to compile the java files.
  - `ant make-jar`: to create a jar file (akin to an X86 executable).
  - `java -jar jar/BranchPredictor.jar traces/trace1 2400`: this runs the simulator with the input trace *trace1*. It also tells the simulator that the maximum allowed predictor size is 2400 bits. More on this later.

Designing a Branch Predictor

As mentioned earlier, you need to design a branch predictor that achieves good accuracy within an area budget. Specifically, the question has three sub-parts: maximum predictor size = (i) 1200 bits, (ii) 2400 bits and (iii) 4800 bits. So you need to design a predictor with any number of tables and registers with the

constraint – the sum of the table sizes (number of entries in the table × bits per entry) and register sizes (number of bits in the register) **MUST** be less than the specified maximum.

Implementing Your Design

- All your changes are to be limited to the files *Predictor1200.java*, *Predictor2400.java* and *Predictor4800.java*. **DO NOT** change any other file.
- For sub-part
  - (i), viz. 1200 bits, implement your predictor in the file Predictor1200.java .
  - (ii), viz. 2400 bits, implement your predictor in the file Predictor2400.java .
  - (iii), viz. 4800 bits, implement your predictor in the file Predictor4800.java .
- Each of these *Predictor<XXXX>.java* files have three functions that need to be implemented:
  - `Predictor()`: This is the constructor. Here you instantiate your tables and registers. **IMPORTANT:** you are allowed to instantiate objects of only the given `Table` and `Register` classes. Instantiating any objects or arrays of your own will result in a zero. You may declare your own variables, both as class members and locally in the functions.
  - `boolean predict(long address)`: Perform the branch prediction based on the branch instruction's program counter, i.e. `address`. Return `true` for taken, `false` for not taken. To access your tables and registers, use the public functions defined in *Table.java* and *Register.java*.
  - `void train(long address, boolean outcome, boolean predict)`: Train the predictor. `address` refers to the branch instruction's program counter, `outcome` is the actual outcome of the branch, `predict` is the prediction your predictor made.
- You may declare any other functions you may need within the *Predictor<XXXX>.java* files. You may declare your own variables, both as class members and locally in the functions.
- Run the commands specified above ("ant clean", "ant"....) to run the simulator. Try for the five different traces, as well as the three different values for the maximum predictor size.

Evaluating Your Design

Create a .tar.gz archive of the files *Predictor1200.java*, *Predictor2400.java* and *Predictor4800.java*. Name it <entry-number-1>_<entry-number-2>.tar.gz . Run the command *python test.py <path-to-archive>*. It should tell you for the fifteen different runs (five traces, three predictor sizes):

- if you instantiated objects or arrays other than `Table` and `Register`
- if compilation succeeded
- if there was a runtime error
- if you violated the maximum size constraint
- number of branches simulated and the accuracy achieved
- if the achieved average accuracy for each predictor size (calculated over the five traces) is greater than the stipulated minimum.

Run the command *python test.py clean* to clean the logs, class files and jar files from the previous runs (deletes the directories *bin*, *jar* and *logs*).

Submitting Your Assignment

We will be evaluating using the same *test.py* script, but with a different set of traces. So make sure your implementation does not violate any of the rules. Submit only the .tar.gz archive, created and named as described above on moodle. The deadline is 23:55 hours, 10th February 2016. There will be no negotiations regarding the deadline. Please do not ask for extensions.

Grading Scheme

Submissions achieving the desired accuracy within the area budget will get full marks. Additionally, the three most accurate submissions will receive a bonus of 2.5 marks.

Championship Branch Prediction

This assignment was inpired by the "Championship Branch Prediction" competition (`www.jilp.org/cbp2014`). You are encouraged to participate as well.