# Simulator Report

Thomas Cumming (Matric No: s1230426)

When the simulator runs, it first validates the command line arguments and reads in the instructions in the trace file into a list, where each entry is a string. Then it carries out a small amount of preprocessing; the raw instructions are of the form 'B <addr> <outcome>'. The 'B' is unnecessary, so the code strips this away and formats each individual instruction into a tuple (addr, outcome), to facilitate future processing. So, the result of preprocessing is the list [(addr, outcome)] for each instruction in the trace file.

After these initial steps, the code follows one of two paths depending on whether or not the user selected static or dynamic branch prediction.

## ***Static Prediction***

First, the user is prompted to choose a static prediction scheme: either always taken, always not taken or profile guided. Once they've made a choice, the code builds a ' static prediction map', which is a dictionary of the form {*addr* : *prediction*}, for each unique address in the trace file that is currently being processed. *prediction* is our guess on whether or not the branch should be taken when we encounter an instruction with branch address *addr*.

### *Building the static prediction map*

Building the map is trivial if the user chose schemes always taken or always not taken - it would be {*addr : 0*} for each unique address appearing in the trace file for always not taken and {*addr : 1*} for always taken. If the user chose to use a profile guided approach, the code invokes a function to tally the number of times, for each unique address, the instruction was taken and not taken and then uses this information to make a prediction for the map: for each instruction in the trace, if it was taken more than 50% of the time, we predict taken, otherwise we predict not taken.

### *Calculating the result*

Figuring out the final result is simple: the simulator invokes a function which goes through each instruction in the trace file, and indexes into the prediction map with it's address to see if we predicted taken or not taken for that address. If the guess was correct we continue, otherwise add one to a count of incorrect guesses (mispredictions). Finally, the number of mispredictions is divided by the total number of instructions in the trace to obtain the misprediction rate and the result is output.

***Dynamic Prediction***
Dynamic prediction works slightly differently. First, there are two classes defined: one to model the global history register (GHR) and one to model the pattern history table (PHT).

*The GHR*
This encapsulates a list representing a register k bits wide. It incorporates two functions, one to shift the contents left whilst shifting in a new value supplied as an argument and one that returns a string representation of the register's contents.

*The PHT*
The PHT is represented by a dictionary within this class: the keys are indexes and the values are the row contents. It has two functions. One allows us to index into the table and obtain the corresponding value, and the other allows us to increase or decrease the 'taken' level.

*Calculation*
After the GHR and PHT have been initialised as per the coursework handout, we loop through each instruction in the trace file. First, we index into the PHT with the contents of the GHR using the functions the above classes provide, and check to see if, in the past given this history, the branch was strongly/weakly taken/not taken. If it was strongly/weakly taken, we predict taken for the instruction we are examining now; we predict not taken if in the past it was weakly/strongly not taken. Next, we evaluate if we our guess was correct. If it was, we increase the 'taken level', otherwise we decrease it and add one to a misprediction counter (since we were wrong). Finally, the GHR is updated with the actual outcome of the instruction we were examining and then continue looping.

After the code has looked at every instruction, the number of mispredictions is divided by the number of instructions in the trace to obtain the misprediction rate, and the result is printed to the screen.