Lab 5: Branch Prediction

In this lab you will implement a simulator for a 2-level branch predictor with a 2-bit saturating counter. You are provided with a text file containing a trace of branch instructions consisting of the PC at which each branch occurs, and whether the branch is Taken (1) or Not Taken (0).

Your goal is to write code to simulate a branch predictor on this trace. Your output file indicates, for each branch in the trace, whether it was predicted as Taken or Not Taken.

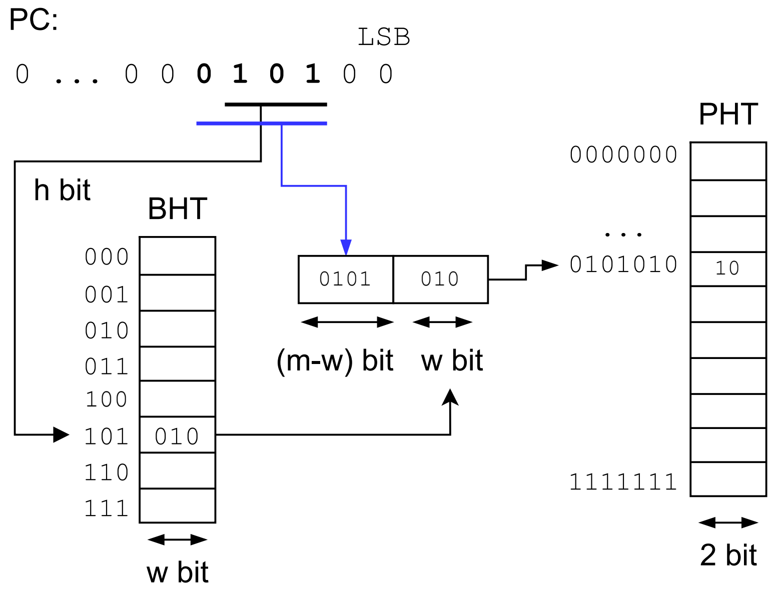
Two-Level Branch Predictor

Your design must consist of a PHT that has 2^m 2-bit saturating counter, and a global BHT that has 2^h w-bit branch history. Each saturating predictor in the PHT should be initialized in 2-bit-“10” (Predict Weak Taken) state and is updated as per the **finite state machine** discussed in class. The value of m is specified in the config.txt.

00	01	10	11
Strong Not Taken	Weak Not Taken	Weak Taken	Strong Taken

Each BHT entry should be initialized to **0**.

Architecture Example: h=3 w=3 m=7



We will start from the branch address (PC) to access the saturating counter for a prediction. Here are more details for the figure:

Step 1: The BHT is indexed using h bits from the PC, starting from the 3rd LSB. The BHT has 2^h entries with each entry having w bits.

Step 2: The w bits from the BHT are concatenated with (m-w) bits from the PC (again starting from the 3rd LSB) as shown in the figure. **Be careful to place the (m-w) bits in the upper portion of the concatenation and the w bits from the BHT into the lower portion of the concatenation as shown in the Figure.**

Step 3: We use these m bits to index into the PHT which has 2^m entries. Each PHT entry gives us our 2-bit prediction for the current branch. Both Strong Not Taken (00) and Weak Not Taken (01) mean we predict the branch to be Not Taken. On the other hand, both Strong Taken (11) and Weak Taken (10) indicate that we predict the branch to be Taken.

Step 4: Compare the prediction with the actual branch action as provided in the trace file. If the branch was indeed taken, update the PHT entry towards the taken side. If the branch was not taken, update the PHT entry towards the not taken side. **Example**: Suppose our PHT entry was 01 (Weak Not Taken) and the actual branch action was Taken (1). We would update the PHT entry to be 10 (Weak Taken).

Step 5: Update the BHT. **Example**: Suppose the current BHT entry is 010 and the branch action is Not Taken (0), then you need to update the BHT entry to 100 by left-shifting the BHT and placing 0 in the LSB.

Config File

The config file config.txt contains 3 lines with the value of m, h, w.

Trace File

The trace file, trace.txt, contains one branch per line. Each line has the branch address (PC) for the corresponding branch (in hex format) followed by a single bit indicating Taken (1) or Not Taken (0). A sample trace file is provided.

Output Format

The output from your simulator should be a file trace.txt.out that has one line per branch. Each line has a single bit that indicates whether that branch was predicted as Taken (1) or Not Taken (0).

What to Submit

1. Code
    a. Your source code. On compilation, we should be able to execute your simulator with the following command

       ```
       ./branchsimulator.out config.txt trace.txt
       ```

    b. Your simulator should output a file trace.txt.out in the same directory.

    c. You must upload **ONLY** one file on Gradescope: "branchsimulator.cpp". Do not zip the file or upload any other file.