UNIVERSITY OF CENTRAL FLORIDA

ECE Department, UCF

EEL 4768: Computer Architecture

Due: Nov 25, 2025 11:59pm

Project 2

Project Objective:

Design and implementation of a branch predictor simulator for testing and verification of this architectural component.

Rules and Regulations:

1. The work must be done alone.

2. Any form of cheating and plagiarism such as sharing of your code or using an available code is reported to the school for consequences and results in getting zero.

3. You can communicate with the instructor and the TA through WebCourses regarding your issues.

4. You can use any high-level programming language for implementation.

5. Accomplish your work in either Windows or Linux environment.

6. Your project report should be comprehensive and include the demanded results and their corresponding analyses and discussion.

7. Your simulator outputs should nearly match the reference outputs.

8. All your source codes along with the executable file must be delivered.

9. A compressed file with the "***.zip**" format can contain all your files.

10. All the specifications of your simulator executable file should be specified through passing "**arguments**".

Project Description:

In this project, the main goal is development of a simulator for modeling a global branch predictor. For a global branch predictor, separate history records are not kept for different conditional jumps. In fact, a shared history record is maintained for all of the conditional jumps. With this feature, any correlation between different conditional jumps helps in making more accurate predictions. However, the shared history record may be filled with useless information and the needed information for a branch may be lost due to limited space. The global branch predictor can use a two-level adaptive mechanism. This scheme is beneficial only for large table sizes. The size of the pattern history table has exponential relationship with the size of the history buffer. The pattern history table should be large enough to be shared among all conditional jumps. There are two types for a two-level adaptive predictor with globally shared history buffer and pattern history table: (a) gshare, when it performs XOR operation on the global history and branch program counter; and (b) gselect, when it concatenates them. In this project, we will focus on gshare.

For simplicity, you only need to model a gshare branch predictor with two main parameters: "**M**" and "**N**". The "**M**" parameter is the number of lower part of the program counter bits. They are used to form the prediction table index. The lowest two bits of the PC can be ignored since they are always equal to zero, which means using the bits "**M+1**" through "**2**". The "**N**" parameter is the number of bits in the global branch history (GBH) register. This parameter should be less than or equal to the "**M**" parameter. If it is equal to zero, then the branch predictor is called a simple bimodal branch predictor. Always the "**M**" number of bits from the lower part of PC is utilized to form the prediction table index. The index to the prediction table is calculated as the XOR operation between the PC index and "**Global History Record << (m-n)**".

When the "N" parameter is greater than zero, we have an N-bit global branch history register with the type of **gshare**. For this case, the index is determined based on both the branch's PC and the global branch history register. Also, the global branch history register should be initialized to zero value at the start of simulation. After getting the trace file, these processes should be computed by the simulator: (a) determining the branch's index into the prediction table, which is calculated by **performing the XOR operation between the current n-bit global branch history register and the uppermost n bits of the PC index bits**. This means accessing the block **{GBH ^ PCIndex_UpperNBit , Size(PCIndex) - N}**. (b) making a prediction based on using index to get the branch's counter from the prediction table. If the counter value is greater than or equal to two, then the branch is predicted taken else it is predicted as not-taken. (c) updating the branch predictor according to the real outcome of the branch. In this process, the counter is incremented if the branch was taken and vice versa. The counter saturation happens at zero or three. (d) updating the global branch history register based on shifting the register to the right by one bit and placing the branch result into the position of the most-significant bit. The branch predictor operation is illustrated in Figure 1.

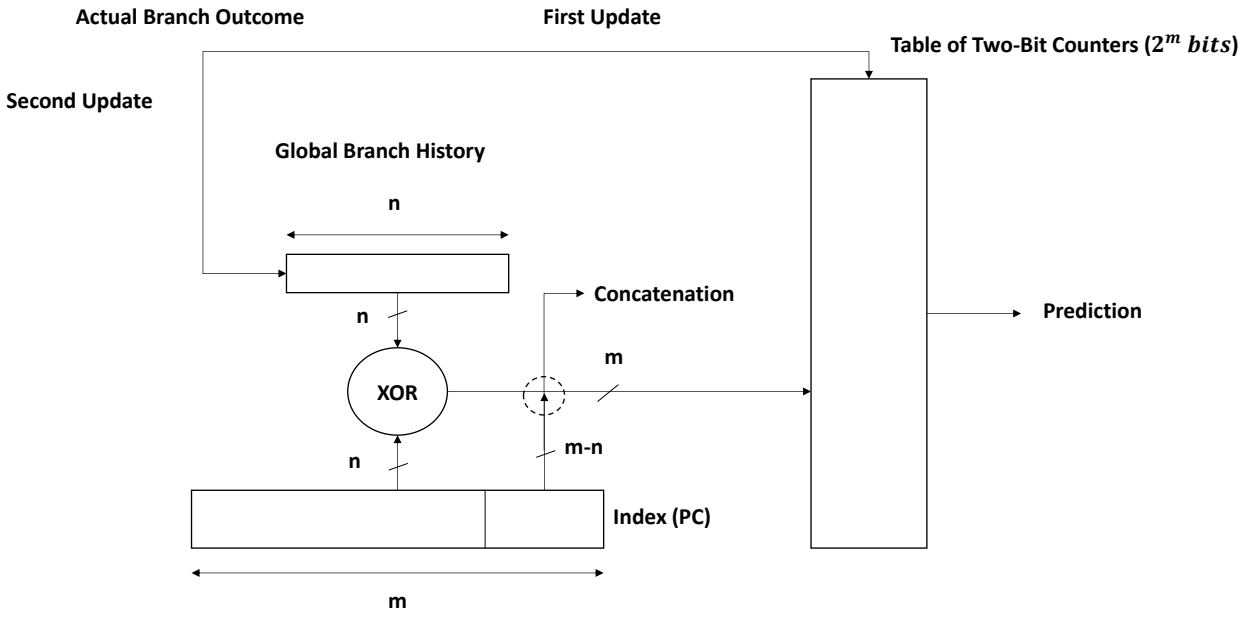

Figure 1: The structure of branch predictor operation

Note: On each entry of the history table is a 2-bit counter/predictor and is initialized to state 2 (Weakly taken).

Inputs to Simulator

The executable file of your simulator along with its arguments should be called in a command-based terminal according to the following:

./sim gshare <GPB> <RB> <Trace_File>

1) GPB = The number of PC bits used to index the gshare table.
2) RB = The global history register bits used to index the gshare table.
3) Trace_File = The trace file name along with its extension.

The simulator should be able to read a trace file in the following format:

<hex branch PC> t|n, in which "hex branch PC" is the address of the branch instruction in the memory (which is used to point to the predictors), "t" indicates that the branch is actually taken, and "n" indicates that the branch is actually not-taken.

Example:

00A4C6FB t

00A6D80D n

…

Outputs from Simulator

The following outputs are expected from your simulator:

<M> <N> <Misprediction Ratio>

For examples:

4 12 0.22

Grade Breakdown

The grade breakdown is as following:

1- (30%) A functional code that runs cleanly, i.e., showing serious efforts in implementation
2- (40%) 4 test cases that must exactly match your simulation output. We will provide you with 3 validation tests and the fourth will be a secret test.
3- (40%) Report
    - A) For each of the provided traces/applications fix M at 4 bits (means you have 16 entries table of 2-bit predictors) and vary N to 1, 2, 3 and 4. Plot the mis-prediction rate of each trace where the X-Axis is N.
    - B) For each of the provided traces/applications fix N at 4 bits (means you history of last 4 outcomes) and vary M to 4, 5, 6 and 7. Plot the mis-prediction rate of each trace where the X-Axis is M.
    - C) For each of the provided traces, assume we have N to be 0 and M to be varied to 4, 5, 6 and 7. Plot the mis-prediction rate with X-Axis being M. Compare with the results in part B.

Good Luck