**Title: To study a Branch Prediction Controller**

**Objective: Branch prediction is to reduce the probability of making a wrong decision, to avoid fetching instructions that eventually have to be discarded.**

**References :** Computer Architecture and Organization –Zaky /Hamacher

Computer Architecture and Organization -Stallings

**Theory:**

In dynamic branch prediction schemes, the processor hardware assesses the likelihood of a given branch being taken by keeping track of branch decisions every time that instruction is executed.

In its simplest form, the execution history used in predicting the outcome of a given branch instruction is the result of the most recent execution of that instruction. The processor assumes that the next time the instruction is executed; the result is likely to be the same. Hence, the algorithm may be described by the two-state machine in Figure *a*.

The two states are:

LT: Branch is likely to be taken

LNT: Branch is likely not to be taken

This simple scheme, which requires one bit of history information for each branch instruction, works well inside program loops. Once a loop is entered, the branch instruction that controls looping will always yield the same result until the last pass through the loop is reached. In the last pass, the branch prediction will turn out to be incorrect, and the branch history state machine will be changed to the opposite state. Unfortunately, this means that the next time this same loop is entered, and assuming that there will be more than one pass through the loop, the machine will lead to the wrong prediction.

Better performance can be achieved by keeping more information about execution history. An algorithm that uses 4 states, thus requiring two bits of history information for each branch instruction, is shown in Figure *b*.

The four states are:

ST: Strongly likely to be taken

LT: Likely to be taken

LNT: Likely not to be taken

SNT: Strongly likely not to be taken

Again assume that the state of the algorithm is initially set to LNT. After the branch instruction has been executed, and if the branch is actually taken, the state is changed to ST; otherwise, it is changed to SNT. As program execution progresses and the same instruction is encountered again, the state of the branch prediction algorithm continues to change as shown. When a branch instruction is encountered, the instruction fetch unit predicts that the branch will be taken if the state is either LT or ST, and it begins to fetch instructions at the branch target address. Otherwise, it continues to fetch instructions in sequential address order. It is instructive to examine the behavior of the branch prediction algorithm in some detail. When in state SNT, the instruction fetch unit predicts that the branch will not be taken. If the branch is actually taken, that is if the prediction is incorrect, the state changes to LNT. This means that the next time the same branch instruction is encountered; the instruction fetch unit will still predict that the branch will not be taken. Only if the prediction is incorrect twice in a row will the state change to ST. After that, the branch will be predicted as taken. Let us reconsider what happens when executing a program loop. Assume that the branch instruction is at the end of the loop and that the processor sets the initial state of the algorithm to LNT. During the first pass, the prediction will be wrong (not taken), and hence the state will be changed to ST. In all subsequent passes the prediction will be correct, except for the last pass. At that time, the state will change to LT. When the loop is entered a second time, the prediction will be correct (branch taken). We now add one final modification to correct the mispredicted branch at the time the loop is first entered. The cause of the misprediction in this case is the initial state of the branch prediction algorithm. In the absence of additional information about the nature of the branch instruction, we assumed that the processor sets the initial state to LNT. The information needed to set the initial state correctly can be provided by any of the static prediction schemes discussed earlier. Either by comparing addresses or by checking a prediction bit in the instruction, the processor sets the initial state of the algorithm to LNT or LT. In the case of a branch at the end of a loop, the compiler would indicate that the branch should be predicted as taken, causing the initial state to be set to LT. With this modification, branch prediction will be correct all the time, except for the final pass through the loop. Misprediction in this latter case is unavoidable.

The state information used in dynamic branch prediction algorithms may be kept by the processor in a variety of ways. It may be recorded in a look-up table, which is accessed using the low-order part of the branch instruction address.

