CENG331 Take-Home Final

Fatih YILDIZ
2306793

Q3.1

I picked the value prediction scheme from the following source:

Lipasti, M. H., Wilkerson, C. B., & Shen, J. P. (1996, September). Value locality and load value prediction. In *Proceedings of the seventh international conference on Architectural support for programming languages and operating systems* (pp. 138-147).

In this value prediction scheme, the Load Value Prediction Unit is utilized. Three main components of this unit are Load Value Prediction Table (LVPT), Load Classification Table (LCT), and Constant Verification Unit (CVU).

This scheme is speculative. Execution starts with the predicted value, but its predictions need to be verified, and if a correction is needed, the pipeline slows down and fixes the errors before continuing. One advantage of this scheme is that it uses load instructions' addresses as the key to a value while storing the predicted values. This lets us do an early lookup because we know the instruction address at the very beginning.

LVPT holds an entry for each static load instruction. Instructions address is used as the key, and the history (of depth one in our case) is the value. History of depth 1 means that only the last read value is used to guess the next bit (somewhat similar to the Last Value Prediction scheme).

LCT assigns the static load instructions into three classes. These are non-predictable, predictable, and constant (highly predictable). Thanks to LCT, we avoid making false predictions and fixing them later for non-predictable loads. We also avoid the memory access overhead if the load is highly predictable. A counter with four states, "don't predict" "don't predict" "predict" and "constant", is used to keep track of the predictability of a load instruction. The counter is decremented in case of a false prediction and incremented otherwise.

CVU stores the index of the LVPT and the data address for constant predictors. The fact that we delete the matching entry in the CVU upon a store instruction lets us keep the CVU consistent with the actual memory, thus letting us avoid the conventional memory access overhead. In case we don't find a match in CVU, the state of the instruction is decremented to "predictable", leading to the conventional memory access to check the correctness of the prediction.