How branching statistics were collected?

The branch predictor mechanism works as follow:

The Program counter asks BPU whether there’s an existing entry. If an entry is found in the table, program counter hands on the decision of whether to branch to the branch predictor. BPU looks up the table to decide whether to take the branch according to previous history. If no entry is found, the program counter assumes the branch will not be taken. And moves the program counter to the next instruction. During the execute state, whether the branching shall take place is answered by the calculation of specific register values. The EXE stage sends back signals to the branch prediction unit to update the BPU table entry.

To analyze how the branching mechanism works, counter must be set. The counters for analysis in this work are placed at the execute stage, for the following reasons:

1. Execute stage is the first stage to certify the assumption of branching or not. There are signals telling whether the branch is miss predicted, the original decision could be obtained from decoding pipeline stage.
2. Decode stage has decoders signals telling which kind of branch it is, a forward or backward jumping branch, or an unconditional jump.

By referencing the branching address extracted from the instruction and the instruction type, we have a clear view which kind of branching instruction it is. These branching type signals are best collected in the decode stage, and send to the execute stage through pipeline registers.

Analysis of branching statistics

From Table 1. Turning on the branch prediction unit improves the Coremark score, from 74.965 to 84.837. The improvement shows on the total cycle counts on executing the program. Without the branch prediction unit, Aquila has to spend extra 78,263,922 cycles to complete the program. From Table 1, we may conclude that branch prediction unit in Aquila DO work to improve the IPC of Aquila.

Table 2. show the distribution of different branch types. Most of the branch jumps to a larger PC address (forward jump), unconditional jumps like JAL and JALR are the least type of branching type, sharing only 9.3% of the overall branch/jump instructions.

Another discovery is most of the branch address is actually not indexed in the BPU memory. From Table 2, we may see that over 90% of the branching instruction do not appear in the BPU table. The branching decision in most of the time is made by the program counter (which violently assume not taken), and guessed it right most of the time. Table 2 shows how high percentage of these unregistered branch are not taken. A reasonable explanation is that these unheard-of branches indicates control sequence that are seldom used, and do not work most of the time.

Do those indexed branches, those branches whose address appear in the BPU table, do a better job? Table 2 shows how the default one-level bimodal branch predictor perform

From Table 2 we can tell the accuracy of three branch types. Forward jumping branch guessed 55% right, only 5% better than flipping a coin. Forward j

Bimodal branch predictor handles Forward branching very poorly, 55 % of accuracy is collected running Coremark program. True Negative and False Positive rate are very low, indicating that most forward branches recorded in BPU would take the branch. This explains why false negative rate in forward branching is also very high, guessing branch not taken in forward jumping is apparently not a good idea. Forward branches are the most common branching type, but the BPU handles it poorly. The overall branching accuracy (include those not in BPU) is 97.376 in this category.

Backward jump is also

Plan on using a two-level predictor to improve performance