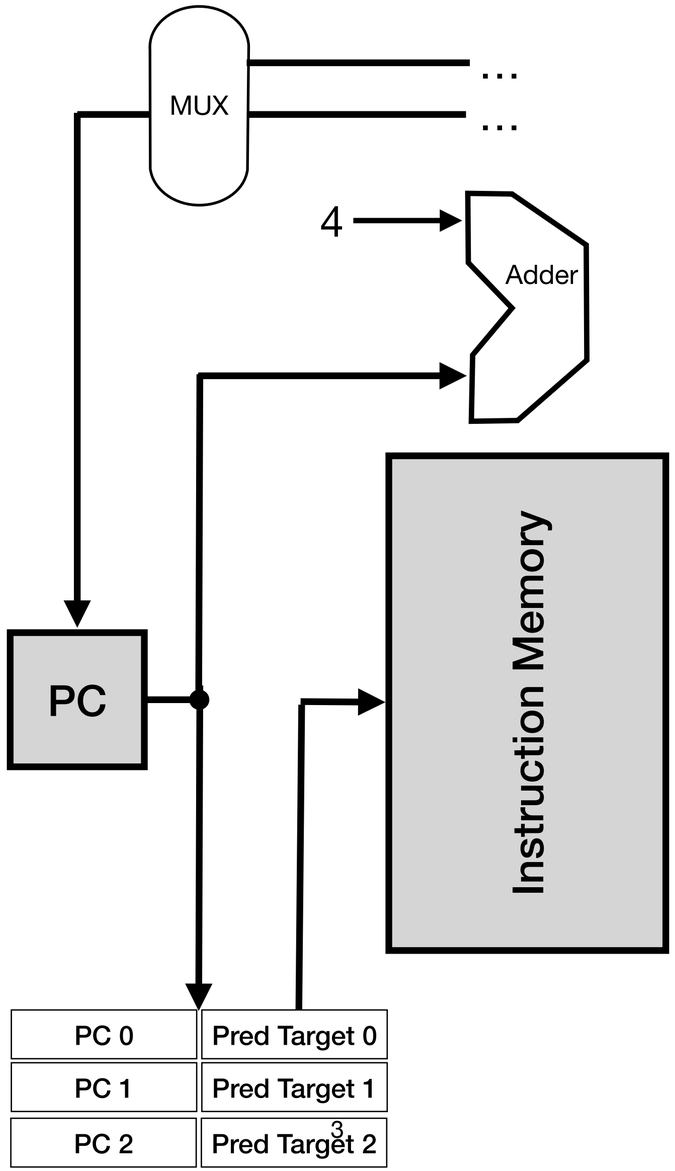
Branch Prediction

Lab Tonight: Homework 3 Gear-Up Session! (Autograder coming by the end of the week)

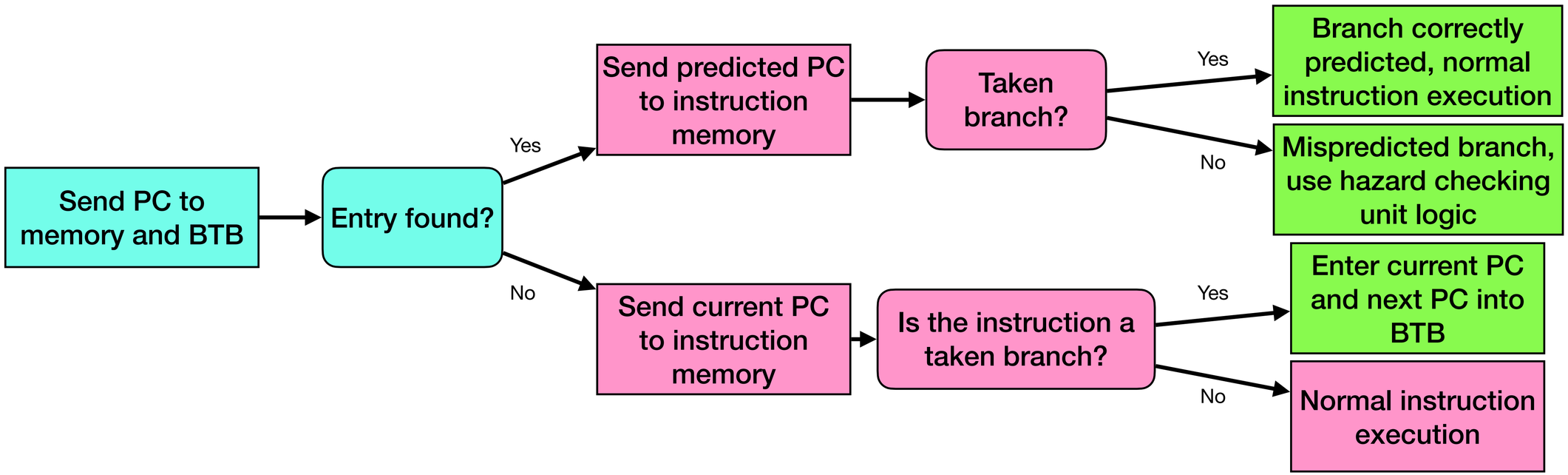
Outline

- Finish formalization of the BTB
- Motivate branch prediction strategies
- Overview branch prediction algorithms

From Monday: Branch-Target Buffers



Data Path Pipeline with BTB



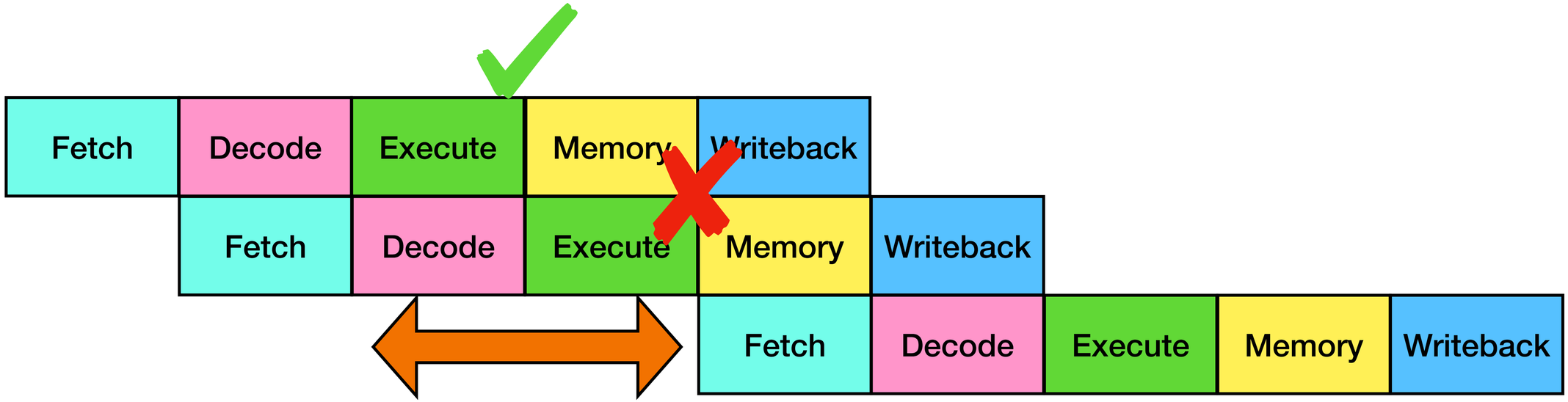
Chat with your neighbor(s)!

Our BTB is essentially a cache for branch instruction targets. Do all hits to the BTB provide the same benefit? Why or why not?

Branch Target Buffers and Instruction Memory

- Recall, instruction memory does not necessarily fetch instructions in a well-defined amount of time
  - ➡️ if the processor issues a fetch for an instruction at `0xff00` then `0xff40`, the memory system may respond with `0xff40` first if it was an instruction cache hit
- We *could* start executing `0xff40` before `0xff00` and later correcting ourselves if a dependence existed between these instructions ➡️ this processor "pre-processing" is called *runahead*
- Implementing runahead uses same mechanism as a control hazard! We are "executing the unknown"

Towards Effective Branch Prediction



Cost of misprediction: the number of cycles to delay in order to fetch from the true next instruction

Goal: reduce the number of mispredictions so that the cost of misprediction is reduced

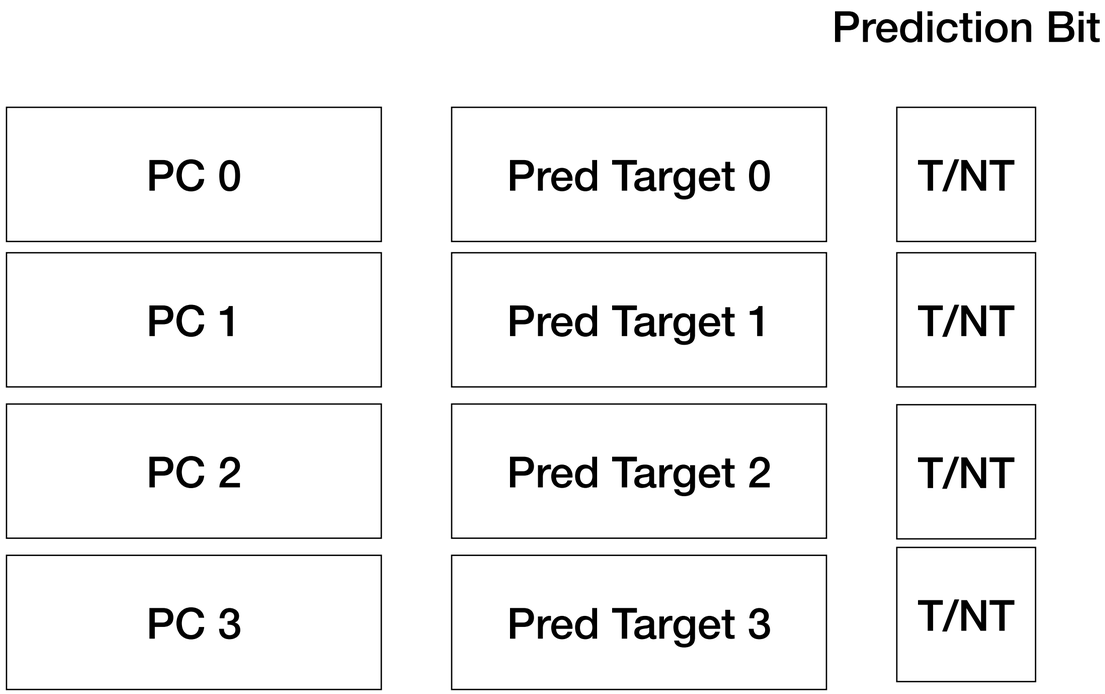
Branch Prediction

How are we going to deal with conditional branches??

- To minimize the number of mispredicted branches, we deploy *branch prediction algorithms* to face the minimum number of branch delays that we possibly can
- We can make these decisions on a *local* (e.g., per program counter) or *global* (e.g., for all branches) granularity
- These algorithms may be *static* in which heuristics of application behavior drive the decision making procedure or *dynamic* in which application runtime behaviors influence the subsequent predictions
  - ➡️ (static) Assume Branch Not Taken: if we assume that most instructions are not branches, then our predicted next program counter should be correct in these cases
  - ➡️ (static) Assume Branches Hit and are Unconditional: if we track PCs and their target in the BTB, assume that these branches are unconditional and use the target associated with this PC as the next address to fetch

One-Bit Branch Predictor

1. Lookup current PC in BTB
2. If not there, assume not a branch and fetch current PC
3. If predicting taken, then fetch the predicted target
4. Otherwise fetch the current PC



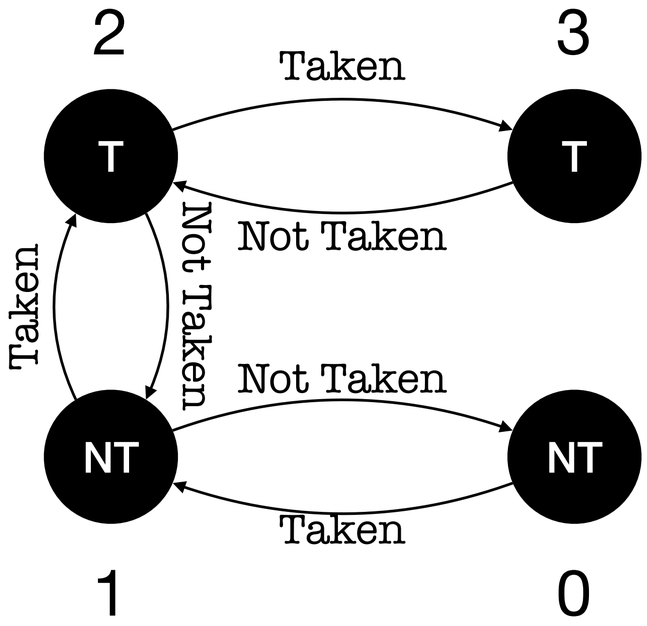
Chat with your neighbor(s)!

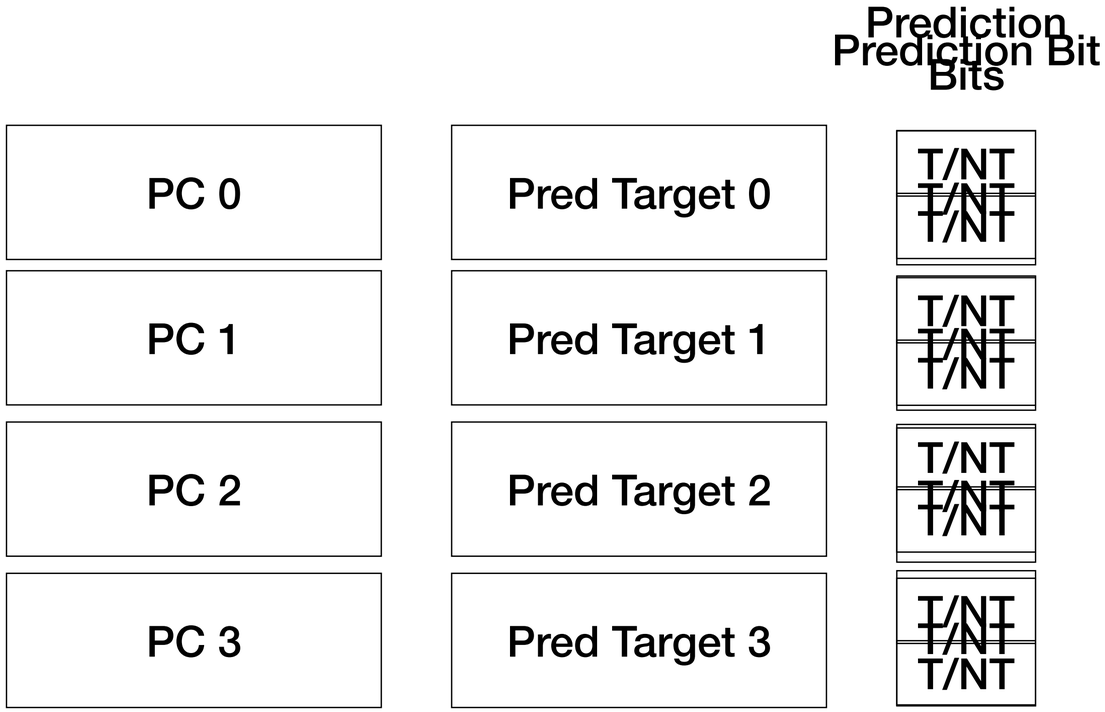
Think about the types of branches that we've described thus far in terms of the high-level language syntax leading to their use. Given this, come up with a code snippet in which the one-bit branch predictor does poorly.

As soon as we have a difference in behavior for one-off cases, it messes with the predictor for that PC!

Two-Bit Branch Predictor

Our 2-bit branch predictor is “sticky” ➡️ don’t make rash decisions based on one behavior





Advanced Branch Prediction

- Correlating Branch Predictors: a one- or two-bit branch predictor examines branches in isolation, but in practice the outcome of many branches depends on the outcome of *other branches* (think the inside of a nested `for`-loop or an `if`-statement within a `while`-loop)
  - ➡️ for the current PC, track the outcome (taken versus not taken) from the last *n* loops into account in your prediction bits
- Tournament Branch Predictors: maintain the information to implement all strategies of branch prediction (local/global, 1-bit, 2-bit, correlating/non-correlating) and *predict* which prediction scheme works best!

On the Effectiveness of Branch Prediction

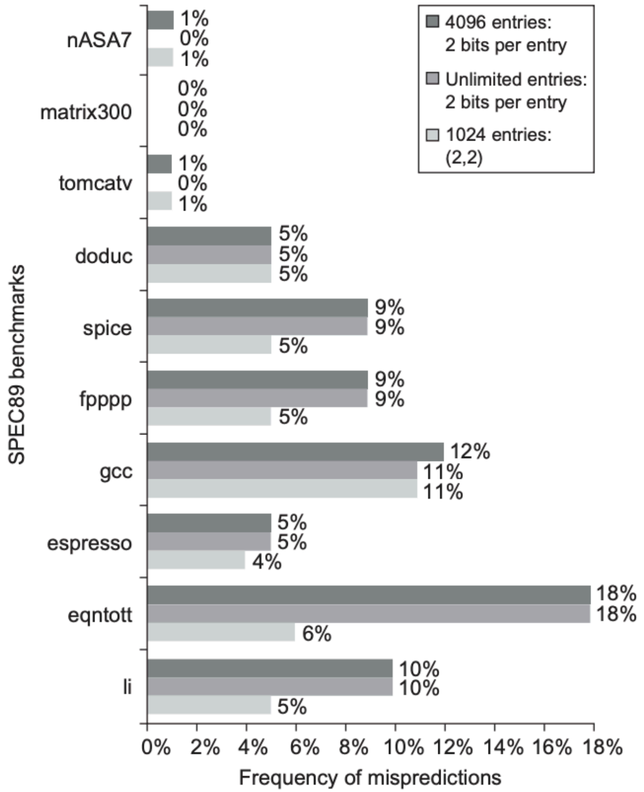


Figure 3.3 Comparison of 2-bit predictors. A noncorrelating predictor for 4096 bits is first, followed by a noncorrelating 2-bit predictor with unlimited entries and a 2-bit predictor with 2 bits of global history and a total of 1024 entries. Although these data are for an older version of SPEC, data for more recent SPEC benchmarks would show similar differences in accuracy.

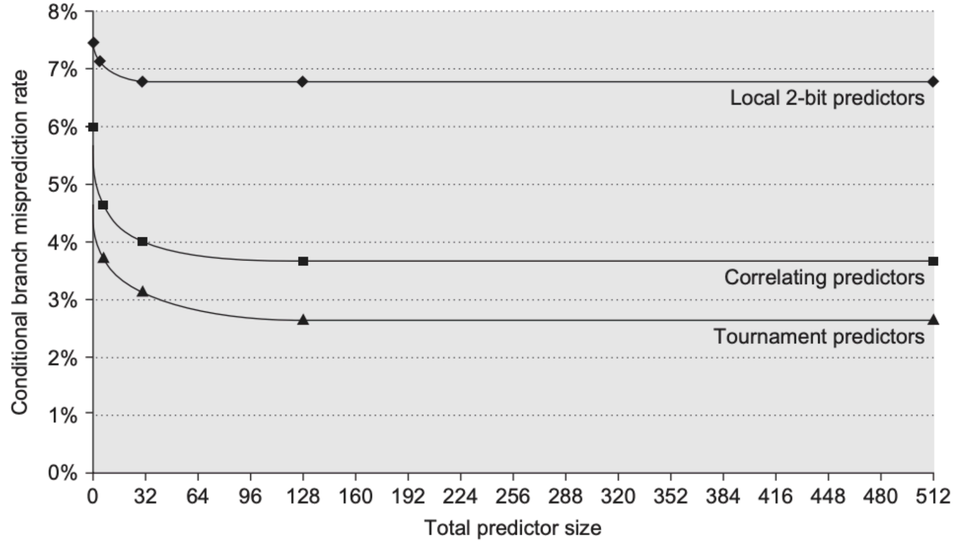


Figure 3.6 The misprediction rate for three different predictors on SPEC89 versus the size of the predictor in kilobits. The predictors are a local 2-bit predictor, a correlating predictor that is optimally structured in its use of global and local information at each point in the graph, and a tournament predictor. Although these data are for an older version of SPEC, data for more recent SPEC benchmarks show similar behavior, perhaps converging to the asymptotic limit at slightly larger predictor sizes.

Image Credit: CA: AQA (course textbook)

Takeaways

- To mitigate control hazards, we can predict the target next instruction based on tracking execution history in the processor
- We can implement *dynamic branch predictors* to implement this behavior with very small modifications to the BTB
- Branch prediction is an extraordinarily well studied problem and is, for intents and purposes, solved ➡️ modern branch predictors almost guarantee the correct outcome of a predicted branch