Branch Prediction (Part 2)

HW3 (Part 2) + Written due Friday;
CS Department Climate Survey due Friday (free t-shirt!)

Accurate Indirect Branch Prediction

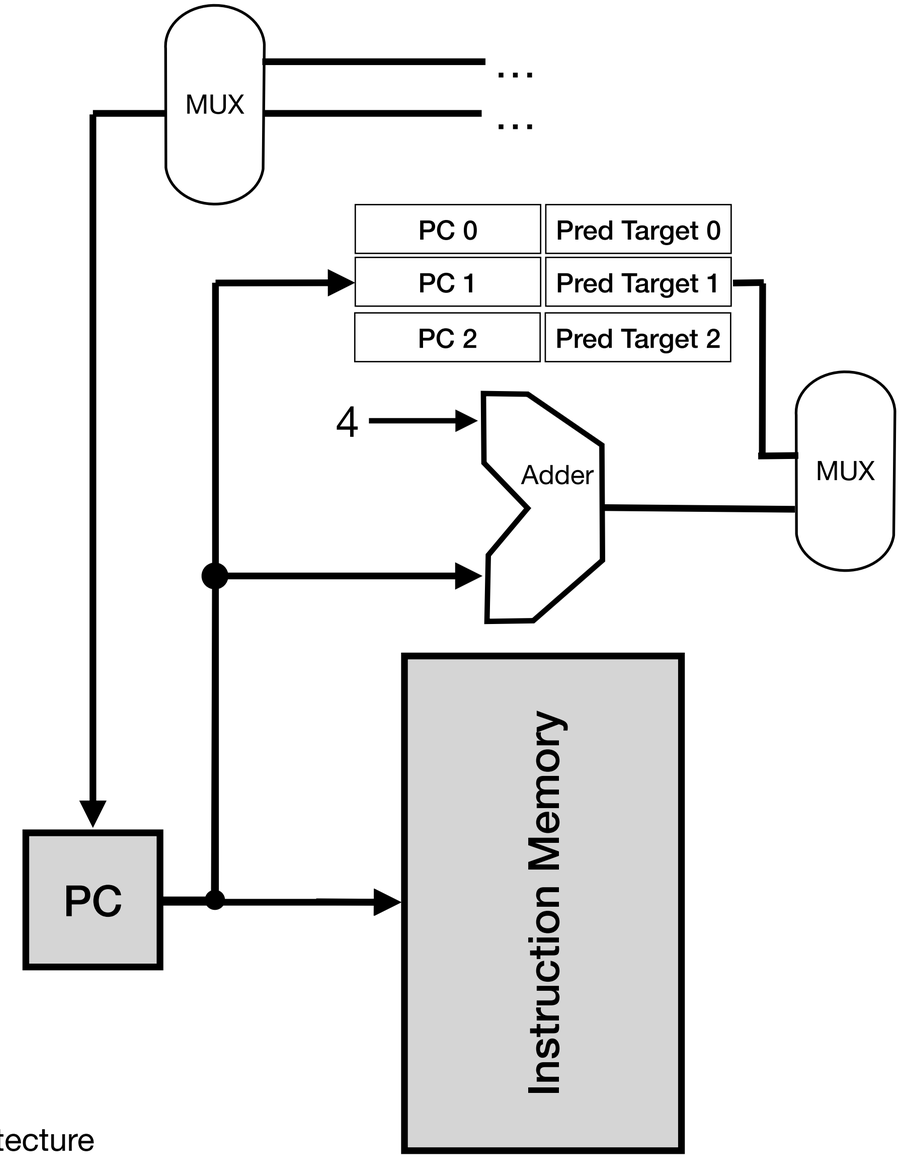
Karel Driesen and Urs Hölzle
Department of Computer Science
University of California
Santa Barbara, CA 93106

Conditional direct branches, whose target is encoded in the instruction itself, can already be predicted with reported hit rates of up to 97% ([YP93]). In contrast, indirect branches, which transfer control to an address stored in a register, are harder to predict accurately. Unlike conditional branches, they can have more than two targets so that prediction requires a full 32-bit or 64-bit address rather than just a "taken" or "not taken" bit. Current processors predict indirect branches with a branch target buffer (BTB) which caches the most recent target address of a branch. Unfortunately, BTBs typically have much lower prediction rates than the best predictors for conditional branches. For example, an ideal (unconstrained) BTB achieves an average prediction hit ratio of only 64% on the SPECint95 benchmarks.

Outline

- Review BTB in the context of branch prediction
- Overview branch prediction algorithms
- Branch prediction effectiveness

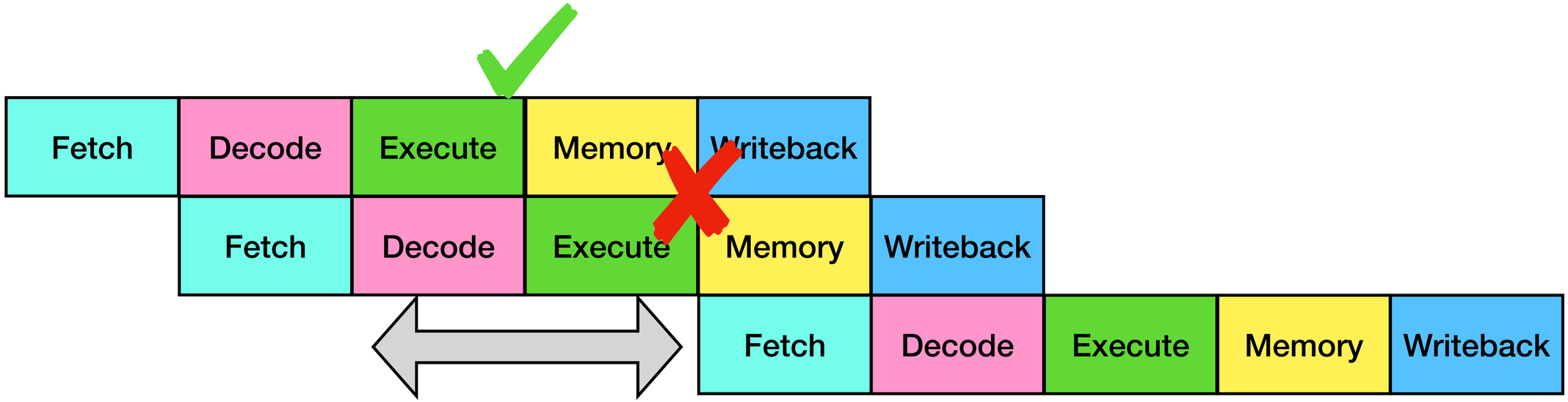
From Monday: Branch-Target Buffers




```
FetchStage::predictTarget() {
    nextPC = PC + 4
    if (BTB.contains(PC)) {
        nextPC = BTB.get(PC)
    }
}
```

```
Decode/ExecuteStage::updateBTB() {
    if (branchTaken) {
        BTB[PC] = branchTarget;
    } else if (!branchTaken) {
        BTB.remove(PC);
    }
}
```

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

Chat with your neighbor(s)!

What are some of the limitations of using the BTB as the sole mechanism for branch prediction? Think about program behaviors that could lead to negative performance.

The BTB serves as a cache for all currently executing instructions. If we fill the BTB, then we will suffer capacity misses which implies predicting branch not taken… this seems undesirable!

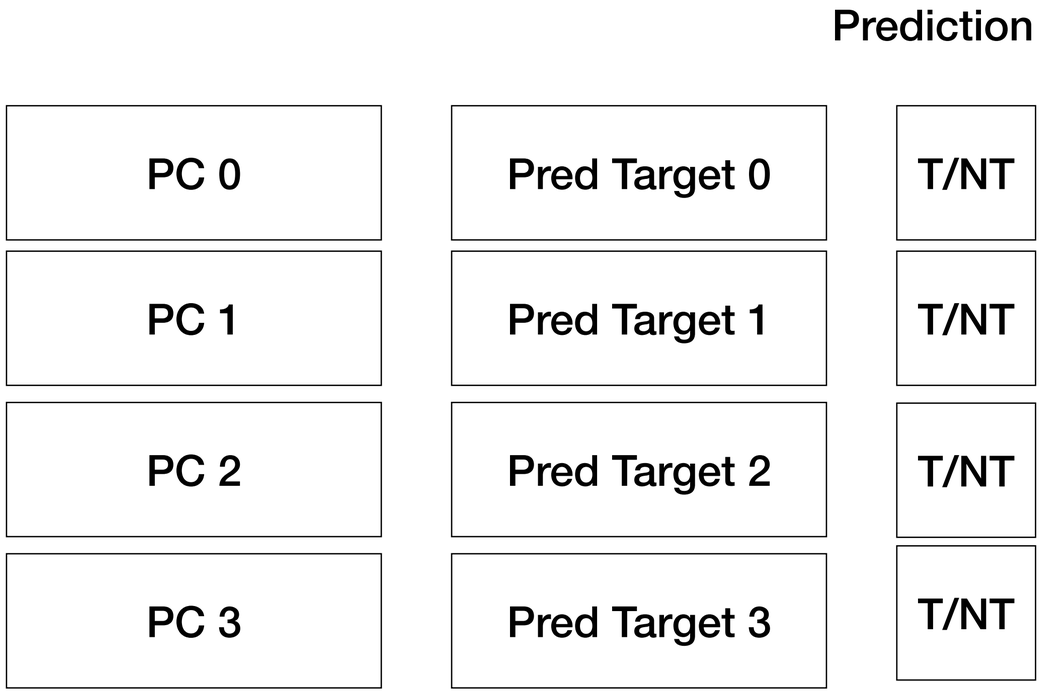
Caches require cache maintenance… if we have multiple stages all trying to access the same line of the BTB, who wins?

Branch Prediction

- 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
  - ➡️ (dynamic) 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



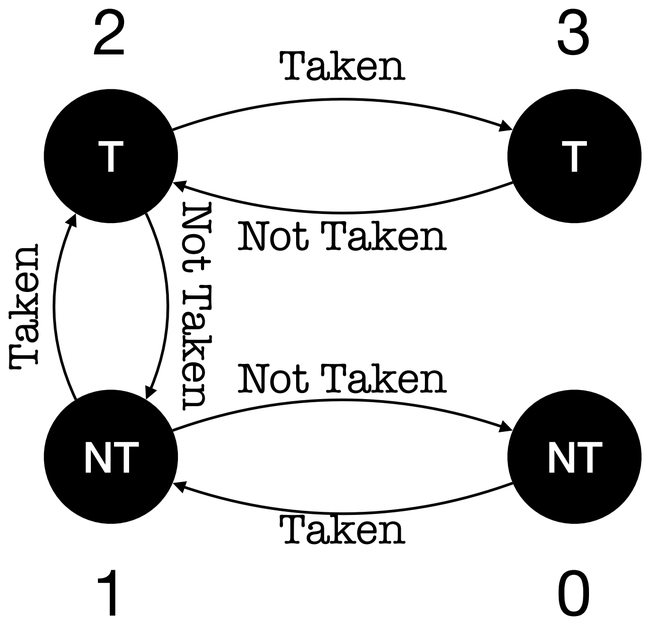
✅ We solve the issue of contention for the same BTB line across execution stages

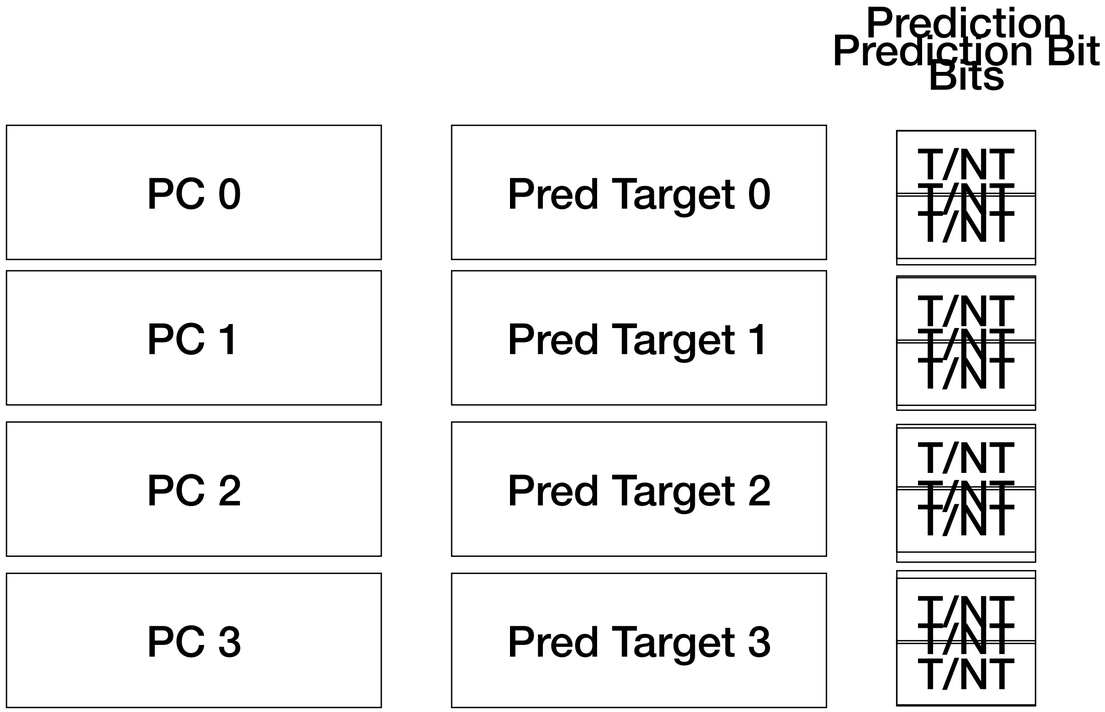
If we have a BTB miss, we still predict not taken ➡️ not based on application behavior…

Our predictions are highly sensitive to slight variations in behavior… can we do better?

Two-Bit Branch Predictor

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



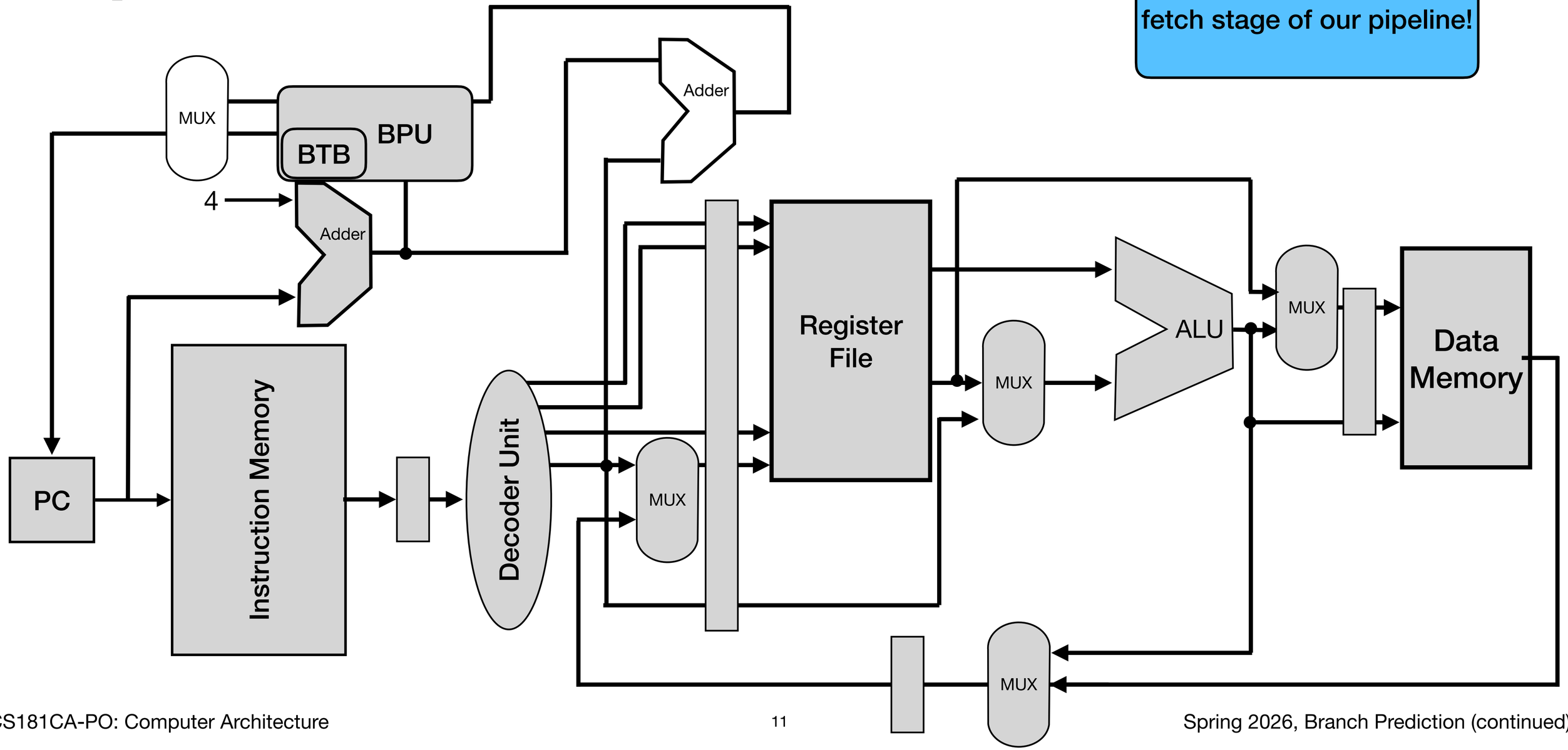


Advanced Branch Prediction

- 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!

Updated Data Path

A lot more work to be done in the instruction fetch stage of our pipeline!



On the Effectiveness of Branch Prediction

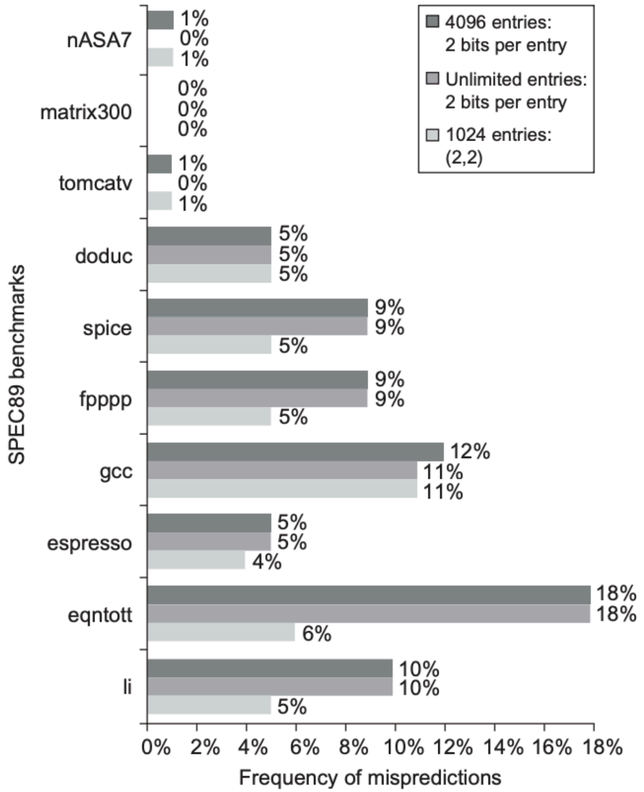


Figure 3.3 Comparison of 2-bit predictors. A noncorrelating predictor for 4096 bits is first, followed by a noncorrelating 2-bit predictor with unlimited entries and a 2-bit predictor with 2 bits of global history and a total of 1024 entries. Although these data are for an older version of SPEC, data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

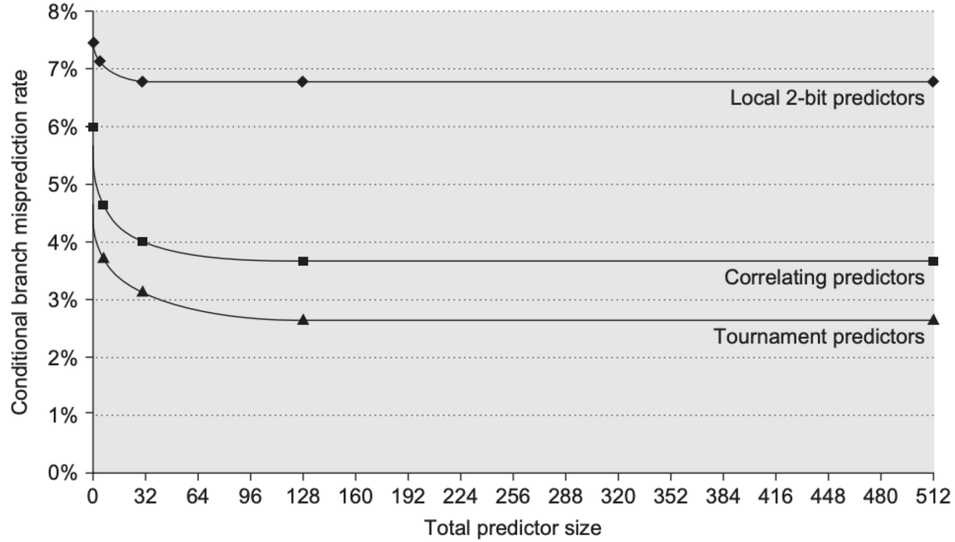


Figure 3.6 The misprediction rate for three different predictors on SPEC89 versus the size of the predictor in kilobits. The predictors are a local 2-bit predictor, a correlating predictor that is optimally structured in its use of global and local information at each point in the graph, and a tournament 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 or in a dedicated branch prediction unit
- 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