Branch handling strategies

- Stall until branch direction is known
- Predict “NOT TAKEN”
  - Execute fall-off instructions that follow the branch (at PC+4, PC+8, …)
  - Squash (or cancel) instructions in pipeline if branch is actually taken
  - (PC+4) already computed, so use it to get next instruction
- Predict “TAKEN”
  - 67% MIPS branches taken on average
  - Start fetching from the branch target as soon it is available (useful if target address is computed earlier than the branch condition).
- Delayed branch

Delayed branch

- Change the branch semantics such that the “N” instructions after a branch are always executed before branching takes place.
  - The N instructions are executed regardless of the branch outcome
  - N is set to be the number of cycles needed to resolve the branch
- The compiler will try to fill the N slots with useful work
  - If can’t find instructions to fill the slots, use NOPs.
- Simplifies hardware (semantics eliminates hazards)
- Possibly good performance – if slots are filled with useful instructions
- Code size will increase??
Finding instructions to fill the delay slots

Branch prediction

- Goal
  - Predict the branch outcome (taken or not taken) and the branch target address (if taken, where should we go?)
  - In short, just figure out what is the next PC (target address or PC+4)?

- When do we predict?
  - Compiler may associate with each branch a “mostly taken” or “mostly not-taken” indicator.
  - May dynamically predict the next PC when we are fetching from the current PC

- How to dynamically predict?
  - Branch condition
  - Branch target address
What to predict – T/NT

- Let’s first focus on predicting taken (T)/not taken (NT)

- Branch outcome may be biased for individual branches
  - Example: in loops, branches may be mostly taken or mostly not taken

- Dynamic prediction using a simple 1-bit predictor
  - Remember the last behavior (taken/not-taken) using a hash table
    - \(2^N\) entries – index using \(N\) bits from PC (for some \(N\))
    - Hashing \(\rightarrow\) may have collision
    - Collision \(\rightarrow\) problem??
  - When should we check the table?
  - If prediction is wrong
    - Take corrective action
    - update the prediction buffer

Target prediction (Branch Target Buffer – BTB)

- We would like to check the table while fetching the instruction
  - How to make sure that the BTB entry refers to a branch instruction?
  - How to make sure it refers to the correct branch instruction?

- Should store and match all bits of PC.

- Add the last branch address to the table (to predict the target)
Target prediction with Branch Target Buffer

Records the target address for the last execution of the branch instruction

Branch Target Buffer (BTB)

- If match and Predict Taken, use the stored target as the next PC
- If match and Predict not Taken, use PC+4 as the next PC
- If no match, then use PC+4 because either
  - The instruction is not a branch
  - It is a branch, but no prediction → assume branch is not taken

After resolving a branch and finding that it was mispredicted
- Undo the damage (flush wrong instruction)
- Restore correct PC
- Update BTB with new information
Shortcoming of 1-bit dynamic predictor

- Remember the last behavior of the branch

```c
for (i=0; i< 100; i++) {
   A[i] = B[i] * C[i];
   D[i] = E[i] / F[i];
}
```

- How many prediction hits and misses? Prediction accuracy?

```c
for (j=0; j<100; j++) {
   for (i=0; i< 5; i++) {
      A[i] = B[i] * C[i];
      D[i] = E[i] / F[i];
   }
}
```

2-bit predictor

- Requires two consecutive miss-predictions to flip direction

![Diagram of 2-bit predictor with two levels of prediction: Predict taken and Predict not taken.](image)

Miss-predicted once

Change prediction

Miss-predicted once

Change prediction

Miss-predicted once

Taken

Not taken

Predict taken

11

Predict not taken

01

Predict taken

10

Predict not taken

00

Taken

Not taken

Taken

Not taken

Taken

Not taken

Taken

Not taken
2-Bit Branch prediction buffer

Branch prediction & pipelining

Assuming that branch condition and target are resolved in ID stage
(a similar chart may be drawn if branch condition/target are resolved in EX)