Branch prediction

- Longer pipelines can’t readily determine branch outcome early
  - Stall penalty becomes unacceptable

- Two possible solutions:
  1. Static branch prediction (taken or not taken)
  2. Dynamic branch prediction
1. A Static branch prediction (not taken case)

**Prediction correct**

- Program execution order (in instructions)
  - add $4, $5, $6
  - beq $1, $2, 40
  - lw $3, 300($0)

**Prediction incorrect**

- Program execution order (in instructions)
  - add $4, $5, $6
  - beq $1, $2, 40
  - lw $3, 300($0)

Convert the incorrectly fetched instruction into a bubble (flush it)
1.B Static branch prediction (taken case)

- Assume branch will be taken

36: sub $10, $4, $8
40: beq $1, $3, 7
44: and $12, $2, $5
48: or $13, $2, $6
52: add $14, $4, $2
56: slt $15, $6, $7
    ...
72: lw $4, 50($7)
Branch taken: cycle 3

and $12, $2, $5
beq $1, $3, 7
sub $10, $4, $8
before<1>
before<2>
Branch taken: cycle 4

There is still 1 cycle penalty to compute target address
Eliminating 1-cycle stall for taken-prediction policy with branch target buffer

- Even with predictor, still need to calculate the target address
  - 1-cycle penalty for a taken branch
- **Branch target buffer**
  - Cache of target addresses
  - Indexed by PC when instruction fetched
    - If hit and instruction is branch predicted taken, can fetch target immediately – no 1-cycle penalty
2. Dynamic branch prediction

- In deeper and superscalar pipelines, branch penalty is more significant
- Use dynamic prediction:
  - **Branch prediction buffer** (aka branch history table) indexed by recent branch instruction addresses and stores outcome (taken/not taken)
  - To execute a branch:
    - Check table, expect the same outcome
    - Start fetching from fall-through or target
    - If wrong, flush pipeline and flip prediction
1-bit predictor

- Inner loop branches mispredicted twice!

  outer: ...
  ...
  inner: ...
  ...
  beq ..., ..., inner
  ...
  beq ..., ..., outer

- Mispredict as taken on last iteration of inner loop
- Then mispredict as not taken on first iteration of inner loop next time around
2-bit predictor

- Only change prediction on two successive mispredictions
Data hazards for branches

- If a comparison register is a destination of 2\textsuperscript{nd} or 3\textsuperscript{rd} preceding ALU instruction

\begin{align*}
\text{add } &\, \$1, \$2, \$3 \quad \text{IF} \quad \text{ID} \quad \text{EX} \quad \text{MEM} \quad \text{WB} \\
\text{add } &\, \$4, \$5, \$6 \quad \text{IF} \quad \text{ID} \quad \text{EX} \quad \text{MEM} \quad \text{WB} \\
\text{beq } &\, \$1, \$4, \text{target} \quad \text{IF} \quad \text{ID} \quad \text{EX} \quad \text{MEM} \quad \text{WB}
\end{align*}

- Hazard can resolved using forwarding and stalling
Data hazards for branches

- If a comparison register is a destination of preceding ALU instruction or 2\textsuperscript{nd} preceding load instruction
  - Need to modify forwarding hardware to forward from EX/MEM or MEM/WB pipeline registers to the comparator
  - Need 1 stall cycle in this example

```
lw $1, addr
add $4, $5, $6
beq stalled
beq $1, $4, target
```
Data hazards for branches

- If a comparison register is a destination of immediately preceding load instruction
  - Need 2 stall cycles
  - Need to modify forwarding hardware to forward from MEM/WB

```
lw  $1, addr
beq stalled
beq stalled
beq $1, $0, target
```
Summary

• Pipelining for speedup

• Ideal speedup = number of stages, but actual speedup depends on delay balance between stages (clock frequency), delays introduced by pipeline registers, and number of stalls (CPI).

• Hazards (structural, data, and control) can increase CPI

• Hazards can be eliminated or mitigated using code reorganization, stalling, flushing, forwarding / bypassing

• Branch prediction, branch prediction buffer, branch target buffer can reduce stalls arising from control hazards