Control Hazards

**Cause** of the hazard:

- evaluation of the branch condition & calculation of the branch target is not completed before the next instruction is fetched
- conflict as to which instruction to fetch next
- called a **delayed branch** if the hazard can’t be eliminated

**Hardware solutions:**

- stall until the result of the condition & target are known (unacceptable delay: no computer does this now)
- assume the branch is not taken
- redesign the pipeline
- dynamic branch prediction

---

The Problem

The diagram illustrates the control hazards in a pipeline. The `beq $1, $3, target` instruction is highlighted, showing the branch condition is evaluated here and the target address is calculated here. The instructions following the branch are shown as conditional, and the diagram indicates that these instructions should not be executed if the branch is taken.
Just Stall

beq $1, $3, target

Even worse:
beq $1, $3, target

Assume a Branch Outcome

Technique:
- assume the branch will not be taken
- continue fetching sequential instructions
- flush them if the branch is taken after all
- fetch the target instruction

Performance savings
- no cost if the condition is false & branch isn’t taken
- 40% of conditional branches are not taken
- 3 cycle penalty if the branch is taken

Implementation (for flushing)
- change control signals for EX, MEM & WB stages
  in IF/ID, ID/EX & EX/MEM pipeline registers to 0
  (similar to what we did for stalling after a load data hazard)
Redesign the Pipeline

Purpose of the redesign:
• determine the branch outcome earlier
• reduce the branch cost of a taken branch

Hardware changes:
• add a target address shifter & adder to ID stage
  ⇒ will know where to branch to in ID stage
• add combinational logic in ID stage to determine the outcome of simple comparisons
  • equal/not equal
  • less than 0
  ⇒ know whether to branch in ID stage
• do the more complicated comparisons in the ALU
  ⇒ know whether to branch in EX stage
• what architectural design principle is being used here?

How did the branch penalty change?

Branch Prediction

Definition:
• Resolve a branch hazard by predicting which path will be taken
• Proceed under that assumption
• Have a mechanism to back out if wrong

Dynamic branch prediction:
• branch prediction implemented in hardware
  (static branch prediction is done by the compiler)
• the prediction changes as program behavior changes
• common algorithm:
  • predict the branch taken if branched the last time
  • predict the branch not-taken if didn’t branch the last time
Branch Prediction Buffer

Branch prediction buffer

- small memory indexed by the lower bits of the address of a branch instruction
- contains a prediction bit/address
- do what the bit says to do
- if the prediction was wrong
  - toggle the bit
  - flush the pipeline
- accessed in IF stage
  - What is the penalty if predict not taken & prediction is correct?
  - What is the penalty if predict taken & prediction is correct?
  - What is the penalty if mispredict?
- branch prediction buffer predicts correctly most of the time

Two-bit Prediction

A single prediction bit does not work well with loops

Two-bit branch prediction for loops

- must be wrong twice to toggle the bit

• What pattern is bad for two-bit branch prediction?
Control Hazards, in Summary

Goals of the solutions to eliminate control hazards:
• assume the common-case outcome
• determine the branch outcome & target address earlier so can branch to the target earlier
• predict the branch direction

Control hazards can occur with all transfers of control:
• jumps
• procedure calls
• returns
• as well as taken conditional branches