Dynamic Branch Prediction

- **Dynamic branch prediction**: guess outcome
  - Start fetching from guessed address
  - Flush on **mis-prediction**
Inside A Branch Predictor

- Two parts
  - **Target buffer**: maps PC to taken target
  - **Direction predictor**: maps PC to taken/not-taken
- What does it mean to “map PC”?  
  - Use some PC bits as index into an array of data items (like Regfile)
More About “Mapping PCs”

- If array of data has N entries
  - Need log(N) bits to index it
- Which log(N) bits to choose?
  - Least significant log(N) after the least significant 2, why?
  - LS 2 are always 0 (PCs are aligned on 4 byte boundaries)
  - Least significant change most often → gives best distribution
- What if two PCs have same pattern in that subset of bits?
  - Called *aliasing*
  - We get a nonsense target (intended for another PC)
  - That’s OK, it’s just a guess anyway, we can recover if it’s wrong
Updating A Branch Predictor

• How do targets and directions get into branch predictor?
  • From previous instances of branches
  • Predictor “learns” branch behavior as program is running
    • Branch X was taken last time, probably will be taken next time

• Branch predictor needs a write port, too (not in my ppt)
  • New prediction written only if old prediction is wrong
Types of Branch Direction Predictors

- Predict same as last time we saw this same branch PC
  - 1 bit of state per predictor entry (take or don’t take)
  - For what code will this work well? When will it do poorly?

- Use 2-level saturating counter
  - 2 bits of state per predictor entry
    - 11, 10 = take, 01, 00 = don’t take
    - Why is this usually better?

- And every other possible predictor you could think of!
  - **ICQ:** Think of other ways to predict branch direction

- Dynamic branch prediction is one of most important problems in computer architecture
Branch Prediction Performance

- Same parameters
  - **Branch**: 20%, load: 20%, store: 10%, other: 50%
  - 75% of branches are taken

- Dynamic branch prediction
  - Assume branches predicted with 75% accuracy
  - CPI = 1 + 0.20*0.75*2 = 1.15

- Branch (esp. direction) prediction was a hot research topic
  - Accuracies now 90-95%
Pipelining And Exceptions

- Remember exceptions?
  - Pipelining makes them nasty

- 5 instructions in pipeline at once

- Exception happens, how do you know which instruction caused it?
  - Exceptions propagate along pipeline in latches
  - Two exceptions happen, how do you know which one to take first?
    - One belonging to oldest insn

- When handling exception, have to flush younger insns
  - Piggy-back on branch mis-prediction machinery to do this

- Just FYI – we’ll solve this problem in ECE 252
Pipeline Performance Summary

- Base CPI is 1, but hazards increase it

- Remember: nothing magical about a 5 stage pipeline
  - Pentium4 (first batch) had 20 stage pipeline

- Increasing **pipeline depth** (#stages)
  - Reduces clock period (that’s why companies do it)
  - But increases CPI
  - Branch mis-prediction penalty becomes longer
    - More stages between fetch and whenever branch computes
  - Non-bypassed data hazard stalls become longer
    - More stages between register read and write
  - At some point, CPI losses offset clock gains, question is when?
Instruction-Level Parallelism (ILP)

- Pipelining: a form of instruction-level parallelism (ILP)
  - Parallel execution of insns from a single sequential program

- There are ways to exploit ILP
  - We’ll discuss this a bit more at end of semester, and then we’ll really cover it in great depth in ECE 252

- We’ll also talk a bit about thread-level parallelism (TLP) and how it’s exploited by multithreaded and multicore processors
Summary

• Principles of pipelining
  • Pipelining a datapath and controller
  • Performance and pipeline diagrams

• Data hazards
  • Software interlocks and code scheduling
  • Hardware interlocks and stalling
  • Bypassing

• Control hazards
  • Branch prediction

Next up: Memory Systems (caches and main memory)