CS 3220
Branch Prediction
Control Hazards Revisited

• Forwarding helps a lot with data hazards
  – Can have a deep pipeline (high clock rate)
  – Need to stall rarely (few wasted cycles)

• But we must still flush due to control hazards!
  – Each flush wastes several cycles worth of work
  – We fetch from PC+4, so we must flush all insts that were fetched after a branch that is taken!
  – Problem: there are lots of taken branches
    • Think loops!
Branch Prediction

• Idea: try to fetch the correct instructions
  – Guess what should be fetched
  – If we guessed correctly, no flush
  – If we guessed wrong, must flush
  – The trick is to be correct most of the time

• Lucky us - things are easy to guess
  – JAL – almost always goes to the same address
  – Branch – if taken, always goes to the same address
    • And taken/not-taken behavior highly repeatable!
But we predict that the next PC should be PC+4!

```verilog
always @(posedge clk) begin
  if(reset)    PC<=STARTPC;
  else if(mispred_A)PC<=pcgood_A;
  else if(!stall_F) PC<=pcpred_F;
end

wire [(DBITS-1):0] pcplus_F=PC+32'd4;
wire [(DBITS-1):0] pcpred_F=pcplus_F;

assign pcgood_A=dobranch_A?brtarg_A:
  isjump_A?jmptarg_A:
  pcplus_A;
assign mispred_A=(pcgood_A!=pcpred_A) && !isnop_A;
```
How to Predict?

• We need to fetch the correct next instruction!
  – (Almost) every cycle
  – Without looking at the previous instruction

• So we need to guess the PC of the next instruction while we are fetching the current one
  – We only know the PC of the current instruction, but nothing else about it!
  – But… we can remember what it did when we executed it before…

• Use a table to remember what happened!
  – At some point, we know the correct next-PC (let’s call it pcgood)
  – When inst done, update predictor: bpred[PC_X]<=pcgood_X;
    • Where stage X is where we want to update the predictor, e.g. the last stage

• Look up the prediction instead of using pcplus
  assign pcpred_F=pcplus_F;

  assign pcpred_F=bpred[PC];
• Branch Prediction Benefit #1
  – Fewer flushes => fewer cycles to execute Sorter2.mif
• Branch Prediction Benefit #2
  – Takes time to figure out pcgood, ~ late in ALU stage
    • Then need time for PC selection and to feed that iMem address input
    • Often this is the critical path:
      get forwarding-corrected reg values from D->A pipeline latches,
      compare for EQ/NE, use this to select between btarg and pcplu,
      see if this pcgood is equal to pcplu, get this mispred signal to where the
      PC is selected, do selection, get that PC value to where the memory is,
      latch it as address for next fetch
  – Moving some of this to next stage (MEM) would fix this
    • E.g. can determine pcgood, then do the rest early in the M stage
    • Didn’t work well before: wastes another cycle on each taken branch
    • Now: wastes another cycle rarely: only if prediction wrong