CS 3220
Forwarding
Back to our data hazard example…

ADD R1, R2, R3
SUB R4, R2, R1

What happens in our two stage pipeline?

C1: aluout_M <= R2 + R3

C2: The correct result for R2 + R3 is in aluout_M
    But it is not ready to be read from R1!
    R1 <= aluout_M; aluout_M <= R2 + R1 (problem!)

Forwarding – the SUB takes the value from aluout_M!
Forwarding

• If result is computed but not yet written to regs, let data-dependent insts use that result instead of waiting for it to be written to regs
• Adding forwarding to our processor
  – First, get the processor working without forwarding!
  – You can finish Project 3 just fine w/o forwarding!
  – But… Project 4 will require more performance
ADD R1, R2, R3
SUB R4, R2, R1

• Without forwarding, we have something like this
  assign regval1_D = regs[rregno1_D];

• Now, we still read the register value, but we’re not done yet…
  assign regout1_D = regs[rregno1_D];

• Then check if we use that value or a forwarded one…
  – With only two stages, we would have something like:
    assign regval1_D = forw1_M2D?
      aluout_M:
        regout1_D;

• What’s the code for forw1_M2D? We’ll get to it!
Our forwarding logic from last slide is
assign regval1_D=forw1_M2D?aluout_M:regout1_D;
What happens for this code
LW R1,0(R2)
ADD R3,R1,R4
We still need to stall!
- This logic only works for forwarding ALU result!
• We could use this forwarding code:
  assign regval1_D=forw1_M2D?wregval_M:regout1_D;
• Now all data dependences fixed by forwarding
  – No need to stall… but does this result in a faster processor?
  – Execution Time = CPI * CycleTime * NumInsts
    • NumInsts stays the same
    • Fewer stalls => CPI is improved
  – Does CycleTime change?
    • It does if this new forwarding becomes the critical path!
  – How do we know?
When to Forward and When to Stall

• You need to know the answers to two questions:
  – At what point in the clock cycle do we need the result of the forwarding logic
  – At what point in the clock cycle does the value we are giving to the forwarding logic become available

• Example #1 in a two-stage pipeline
  assign regval1_D=forw1_M2D?aluout_M:regout1_D;
  – When do we use regval1_D?
    • Around the middle of the clock cycle (we feed it to ALU)
  – When do we have aluout_M available?
    • At the very beginning of the clock cycle (comes from a pipeline FF)
  – So we have some time for the forwarding logic to do its job!
Example #2 in a two-stage pipeline
assign regval1_D = forw1_M2D ? wregval_M : regout1_D;

- When do we use regval1_D?
  - Around the middle of the clock cycle (we feed it to ALU)

- When do we have wregval_M available?
  - Around the middle or towards the end of the clock cycle (we need to read memory, then select wregval_M)

- So the new critical path for CycleTime may now be read mem -> select wregval_M -> forwarding -> ALU

- Note: may, not is
  - If it is not, forwarding wregval_M is a win (smaller CPI, same CycleTime)
  - Even if critical, CycleTime may increase little and this may still be a win
  - But it may be that CycleTime increases more than CPI decreases…
  - The only way to know is to try… or figure out the exact timing numbers
• When to use forwarded value instead of regout?
  – If there is a dependence, must forward or stall
  – So we need to know if there is a dependence:
    assign dep1_M2D=wrreg_M&&(wregno_M==rregno1_D);
• If our forwarding logic is
  assign regval1_D=forw1_M2D?aluout_M:regout1_D;
• Then
  assign forw1_M2D=dep1_M2D&&selaluout_M;
  assign stall1_M2D=dep1_M2D&&((selmemout_M||selpcplus_M);
  – If no dependence, don’t stall and use regout1
  – If dependence and selaluout_M is 1, forward and don’t stall
  – If dependence and it’s not aluout that we need, we stall
  – Note: assign stallto_D=stall1_M2D||stall2_M2D;
Forwarding with more stages

• Let’s look at a 3-stage pipeline
  – Fetch/Decode/ReadRegs, ALU, MEM/WRReg

• Where does forwarding logic go
  – Can go at the end of F/D/R stage
    • Read regs, forward, latch value, then in next cycle feed it to ALU
  – Can go at start of A stage
    • Read regs, latch, then in next cycle forward and feed to ALU
  – Which one to choose?
    • Not a clear-cut decision (depends on critical path)
    • Usually, better if done in a stage that has more slack (less critical)
Example: Forward at the end of F/D/R stage

- Now, there are three possible values for `regval1`
  - If inst that is in A stage will write to our reg, we need that value
  - If inst that is in A stage does not write to our reg, but the inst that is now in M stage does, we need that value
  - Otherwise, we have the correct value from regs (regout1_D)

```plaintext
assign regval1_D = forw1_A2D?aluout_A:
                   forw1_M2D?wregval_M:
                   regout1_D;
```

Notes

- For A stage, no choice about aluout or wregval
  - We only have aluout_A, wregval selected in M stage
  - We have to stall if we need memout or pcplus from A stage!
- When do we need `regval1_D`?
- When do we produce `aluout_A` and `wregval_M`?
We don’t have to forward everything!

- We could forward only from previous inst, e.g. assign regval_D=forw1_A2D?aluout_A?regout_D;
- And stall if we need the result from M stage
- Wait… isn’t forwarding always better than stalling
  - Usually it is… but if forwarding logic is the critical path
  - Simpler forwarding logic => shorter critical path
  - What if we don’t have many INST_n => INST_{n+2} deps?
wire [(DBITS-1):0] regval1_D= 
  dep1A1_D?wregval_A1: 
  dep1A2_D?wregval_A2: 
  dep1M_D?wregval_M: 
  dep1W_D?wregval_W: 
  regout1_D;

• **Problem 1:** More selection logic makes it slower  
  – More layers of selection logic (logic delay)  
  – Lots of wires go to this logic (wire delay)

• **Problem 2:** Slowest wregval determines speed  
  – E.g. wregval_M generated after Read-Mem delay, plus selection between that and ALU result, plus wires to where the forwarding logic is!