CS250 Computer Architecture

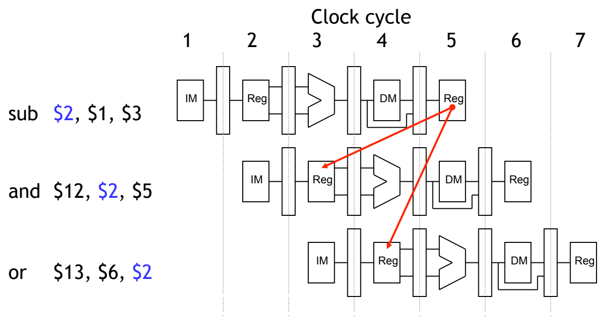
Stalls and Flushes

Stalls and flushes

- Last time, we discussed data hazards that can occur in pipelined CPUs if some instructions depend upon others that are still executing.
  - Many hazards can be resolved by forwarding data from the pipeline registers, instead of waiting for the writeback stage.
  - The pipeline continues running at full speed, with one instruction beginning on every clock cycle.
- Today we' ll see some real limitations of pipelining.
  - Forwarding may not work for data hazards from load instructions.
  - Branches affect the instruction fetch for the next clock cycle.
- In both of these cases we may need to slow down, or stall, the pipeline.

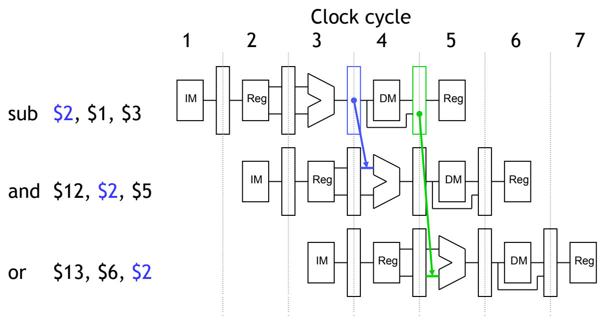
Data hazard review

- A data hazard arises if one instruction needs data that isn' t ready yet.
  - Below, the AND and OR both need to read register $2.
  - But $2 isn' t updated by SUB until the fifth clock cycle.
- Dependency arrows that point backwards indicate hazards.



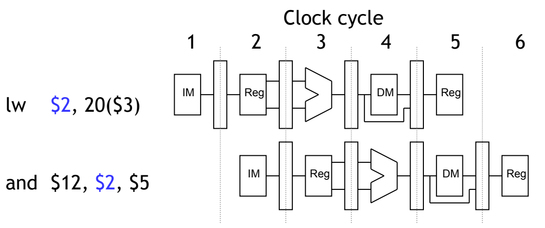
Forwarding to the rescue!

- The desired value ($1 - $3) has actually already been computed—it just hasn' t been written to the registers yet.
- Forwarding allows other instructions to read ALU results directly from the pipeline registers, without going through the register file.



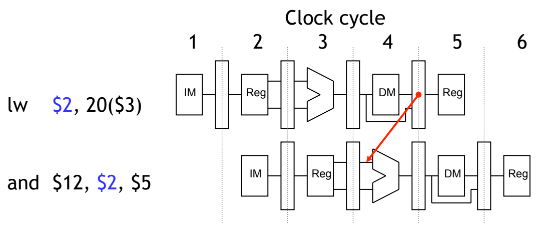
What about loads?

- Imagine if the first instruction in the example was LW instead of SUB.
  - How does this change the data hazard?



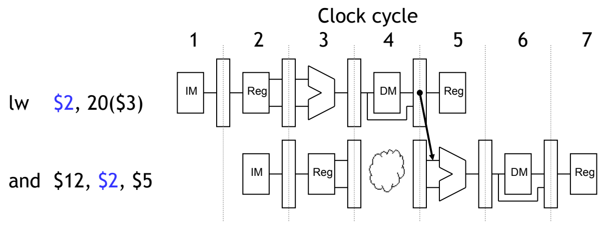
What about loads?

- Imagine if the first instruction in the example was LW instead of SUB.
  - The load data doesn't come from memory until the *end* of cycle 4.
  - But the AND needs that value at the *beginning* of the same cycle!
- This is a "true" data hazard—the data is not available when we need it.



Stalling

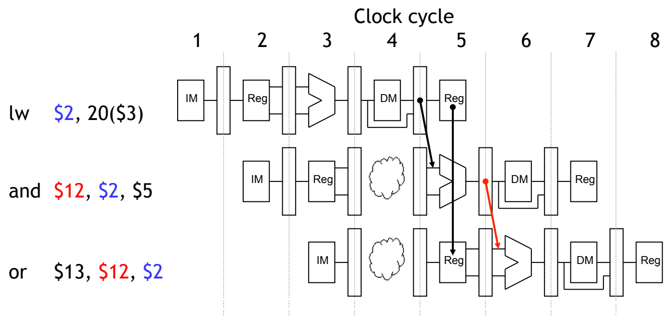
- The easiest solution is to stall the pipeline.
- We could delay the AND instruction by introducing a one-cycle delay into the pipeline, sometimes called a bubble.



- Notice that we're still using forwarding in cycle 5, to get data from the MEM/WB pipeline register to the ALU.

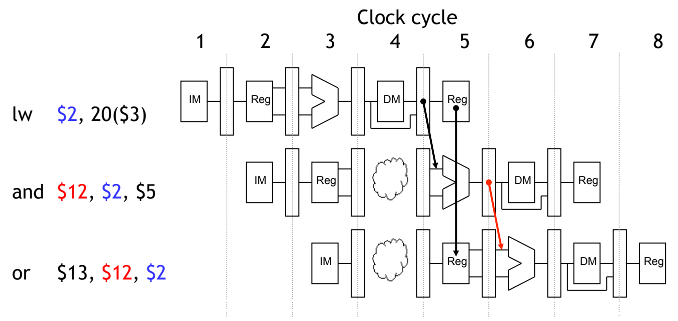
Stalling delays the entire pipeline

- If we delay the second instruction, we'll have to delay the third one too.
  - Why? (two reasons)



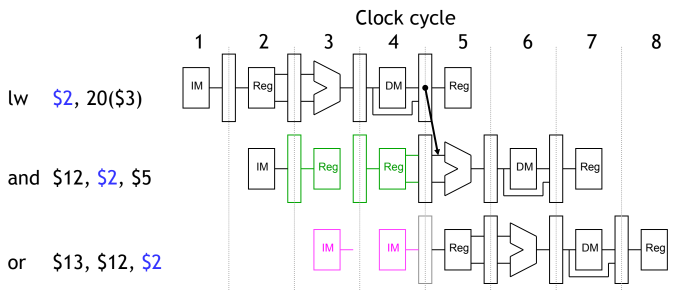
Stalling delays the entire pipeline

- If we delay the second instruction, we' ll have to delay the third one too.
  - This is necessary to make forwarding work between AND and OR.
  - It also prevents problems such as two instructions trying to write to the same register in the same cycle.



Implementing stalls

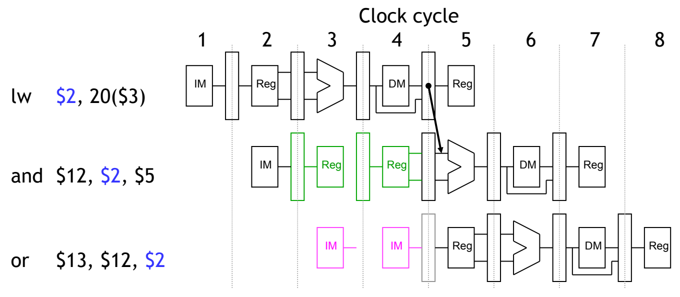
- One way to implement a stall is to force the two instructions after LW to pause and remain in their ID and IF stages for one extra cycle.



- This is easily accomplished.
  - Don' t update the PC, so the current IF stage is repeated.
  - Don' t update the IF/ID register, so the ID stage is also repeated.

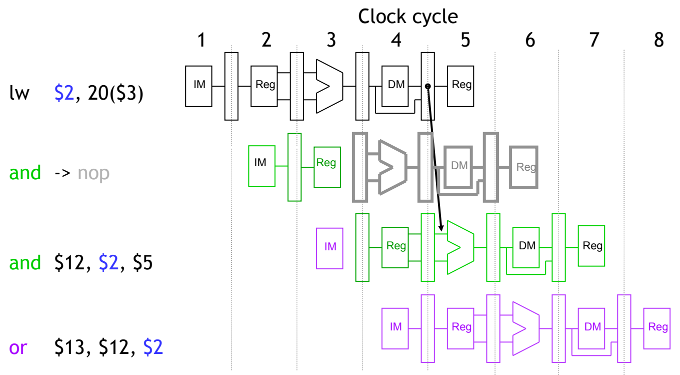
What about EXE, MEM, WB

- But what about the ALU during cycle 4, the data memory in cycle 5, and the register file write in cycle 6?



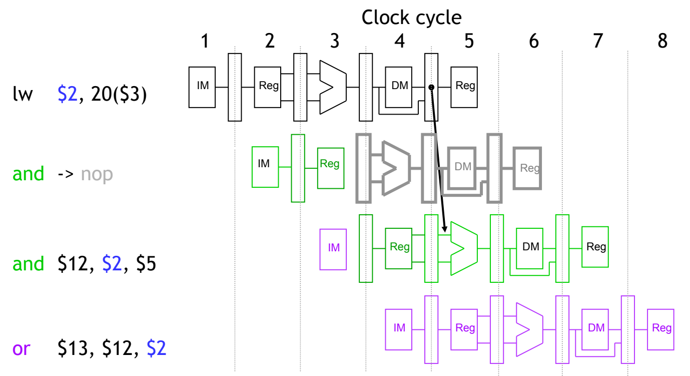
- Those units aren' t used in those cycles because of the stall, so we can set the EX, MEM and WB control signals to all 0s.

Stall = Nop conversion



- The effect of a load stall is to insert an empty or nop instruction into the pipeline

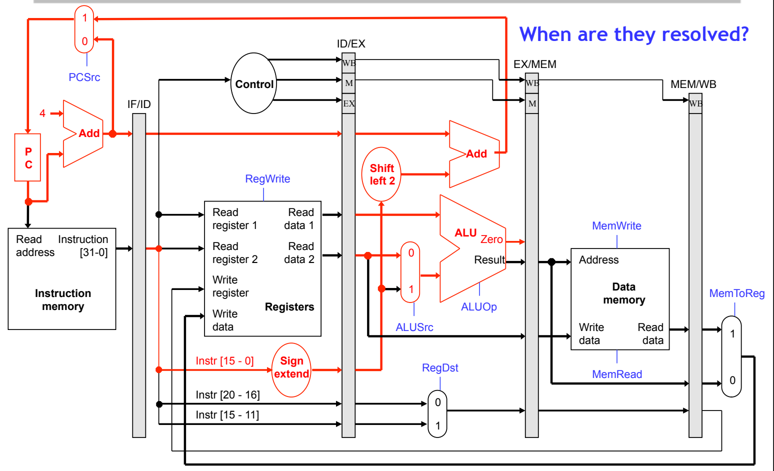
Stall = Nop conversion



- The effect of a load stall is to insert an empty or nop (“no operation”) instruction into the pipeline

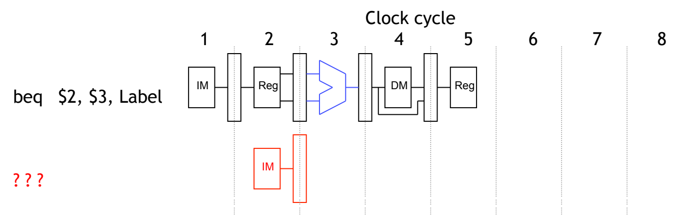
Branches in the original pipelined datapath

When are they resolved?



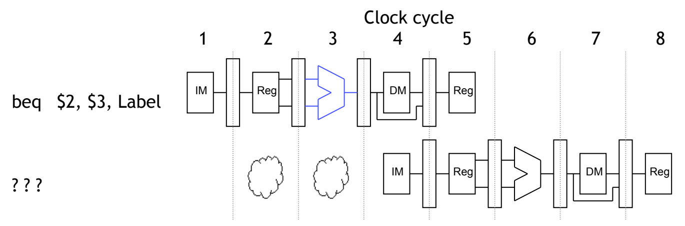
Branches

- Most of the work for a branch computation is done in the EX stage.
  - The branch target address is computed.
  - The source registers are compared by the ALU, and the Zero flag is set or cleared accordingly.
- Thus, the branch decision cannot be made until the end of the EX stage.
  - But we need to know which instruction to fetch next, in order to keep the pipeline running!
  - This leads to what' s called a control hazard.



Stalling is one solution

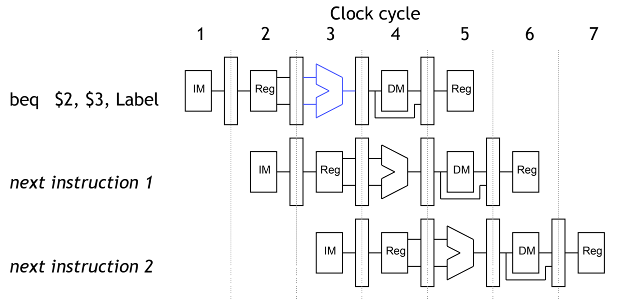
- Again, stalling is always one possible solution.



- Here we just stall until cycle 4, after we do make the branch decision.

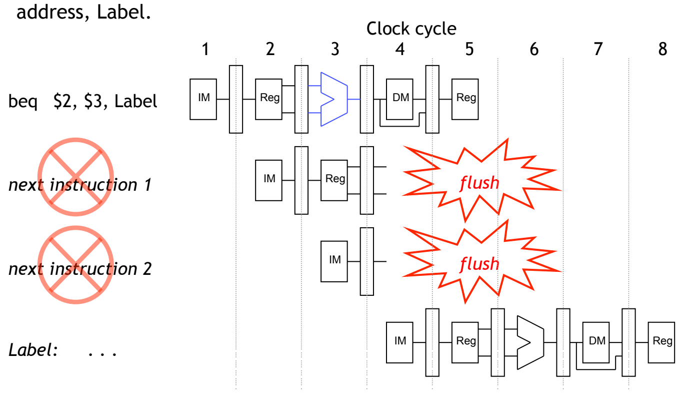
Branch prediction

- Another approach is to *guess* whether or not the branch is taken.
  - In terms of hardware, it's easier to assume the branch is *not* taken.
  - This way we just increment the PC and continue execution, as for normal instructions.
- If we're correct, then there is no problem and the pipeline keeps going at full speed.



Branch misprediction

- If our guess is wrong, then we would have already started executing two instructions incorrectly. We'll have to discard, or flush, those instructions and begin executing the right ones from the branch target address, Label.



Performance gains and losses

- Overall, branch prediction is worth it.
  - Mispredicting a branch means that two clock cycles are wasted.
  - But if our predictions are even just occasionally correct, then this is preferable to stalling and wasting two cycles for *every* branch.
- All modern CPUs use branch prediction.
  - Accurate predictions are important for optimal performance.
  - Most CPUs predict branches dynamically—statistics are kept at run-time to determine the likelihood of a branch being taken.
- The pipeline structure also has a big impact on branch prediction.
  - A longer pipeline may require more instructions to be flushed for a misprediction, resulting in more wasted time and lower performance.
  - We must also be careful that instructions do not modify registers or memory before they get flushed.

Summary

- Two kinds of hazards conspire to make pipelining difficult.
- Data hazards can occur when instructions need to access registers that haven' t been updated yet.
  - Hazards from R-type instructions can be avoided with forwarding.
  - Loads can result in a "true" hazard, which must stall the pipeline.
- Control hazards arise when the CPU cannot determine which instruction to fetch next.
  - We can minimize delays by doing branch tests earlier in the pipeline.
  - We can also take a chance and predict the branch direction, to make the most of a bad situation.