Hazards

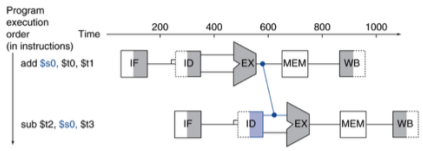
Hazards are situations that prevent the next instruction in the pipeline from executing in the next cycle.

Structure Hazards arises when there is a conflict for the use of a resource. Load/Store operations require data access, causing the instruction fetch to stall for that cycle.

Data Hazards occur when an instruction depends on the completion of data access by a previos instruction. For example, in the instructions `add $s0, $t0, $t1` followed by `sub $t2, $s0, $t3`, the latter must wait for the former to complete

Forwarding (Bypassing) passes a computed result from one stage to another, avoiding the wait for register storage and reduces stalls

- Use result when it is computed
  - Don't wait for it to be stored in a register
  - Requires extra connections in the datapath



A load-use-hazard occurs when an instruction depends on a value being loaded by previous instruction before its available. They can always be avoided by forwarding. For example, `lw $s0, 20($t1)` and `sub $t2, $s0, $t3` the second instruction must wait for the 1st unless resolved with forwarding

Code Scheduling involves reordering instructions to avoid the use of a load result in the subsequent instruction.

Stall on Branch: To mitigate control hazards, waiting until the branch outcome is determined before fetching the instruction is an approach but it introduces stalls.

Branch Prediction guesses the outcome of branch instructions which helps decide wether to fetch the next instruction or branch target early

Predict branch not taken reduces control hazard cost by assuming which is efficient when misprediction penalties are low

Static branch prediction uses fixed rules: backwards branches are predicted as taken, forward as not taken.