Hazards

- Structural hazards
- Data hazards
- Control hazards
Control Hazards

- Simple techniques to handle control hazard stalls:
  - for every branch, introduce a stall cycle (note: every 6th instruction is a branch on average!)
  - assume the branch is not taken and start fetching the next instruction – if the branch is taken, need hardware to cancel the effect of the wrong-path instructions
  - predict the next PC and fetch that instr – if the prediction is wrong, cancel the effect of the wrong-path instructions
  - fetch the next instruction (branch delay slot) and execute it anyway – if the instruction turns out to be on the correct path, useful work was done – if the instruction turns out to be on the wrong path, hopefully program state is not lost
Branch delay slot

(a) From before

DADD R1, R2, R3
if R2 = 0 then

Delay slot

becomes

if R2 = 0 then

DADD R1, R2, R3

(b) From target

DSUB R4, R5, R6

if R1 = 0 then

Delay slot

becomes

DSUB R4, R5, R6

(c) From fall-through

DADD R1, R2, R3
if R1 = 0 then

Delay slot

OR R7, R8, R9

DSUB R4, R5, R6

becomes

DADD R1, R2, R3
if R1 = 0 then

DSUB R4, R5, R6

OR R7, R8, R9

DSUB R4, R5, R6
Multicycle Instructions
Effects of Multicycle Instructions

- Potentially multiple writes to the register file in a cycle
- Frequent RAW hazards
- WAW hazards (WAR hazards not possible)
- Imprecise exceptions because of o-o-o instr completion

Note: Can also increase the “width” of the processor: handle multiple instructions at the same time: for example, fetch two instructions, read registers for both, execute both, etc.
Precise Exceptions

• On an exception:
  ➢ must save PC of instruction where program must resume
  ➢ all instructions after that PC that might be in the pipeline
    must be converted to NOPs (other instructions continue
    to execute and may raise exceptions of their own)
  ➢ temporary program state not in memory (in other words,
    registers) has to be stored in memory
  ➢ potential problems if a later instruction has already
    modified memory or registers

• A processor that fulfils all the above conditions is said to
  provide precise exceptions (useful for debugging and of
  course, correctness)
Dealing with these Effects

• Multiple writes to the register file: increase the number of ports, stall one of the writers during ID, stall one of the writers during WB (the stall will propagate)

• WAW hazards: detect the hazard during ID and stall the later instruction

• Imprecise exceptions: buffer the results if they complete early or save more pipeline state so that you can return to exactly the same state that you left at
Slowdowns from Stalls

- Perfect pipelining with no hazards → an instruction completes every cycle (total cycles ~ num instructions) → speedup = increase in clock speed = num pipeline stages
- With hazards and stalls, some cycles (= stall time) go by during which no instruction completes, and then the stalled instruction completes
- Total cycles = number of instructions + stall cycles
- Slowdown because of stalls = $1/ (1 + \text{stall cycles per instr})$
Pipelining Limits

Assume that there is a dependence where the final result of the first instruction is required before starting the second instruction.
Thank you!