Processor Architecture IV: Pipeline (1)

Kai Zhang
Fudan University
zhangk@fudan.edu.cn

from https://csapp.cs.cmu.edu/3e/home.html

Overview

- **General Principles of Pipelining**
  - Goal
  - Difficulties
- **Creating a Pipelined Y86-64 Processor**
  - Rearranging SEQ
  - Inserting pipeline registers
  - Problems with data and control hazards

Real-World Pipelines: Car Washes

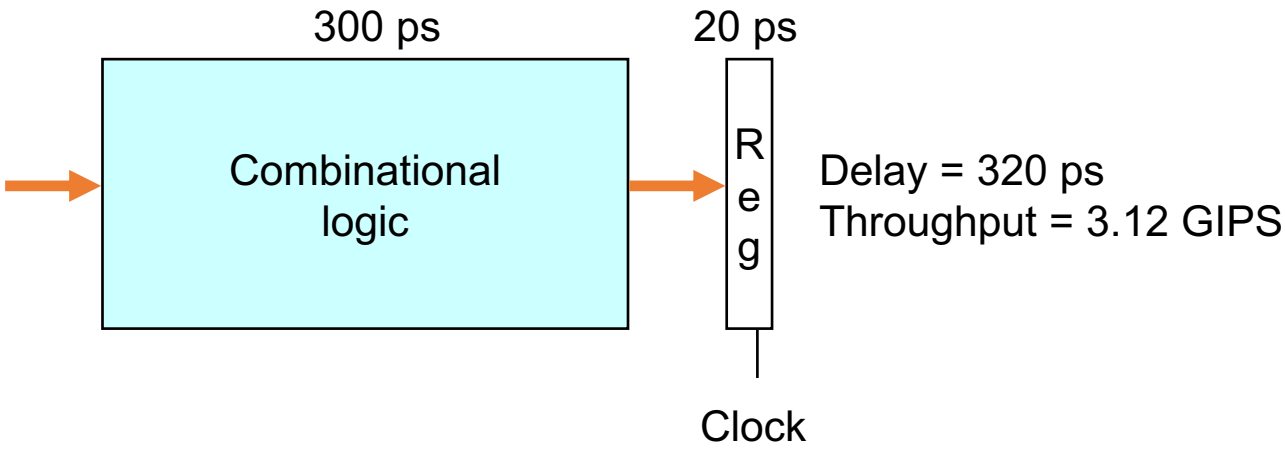
Sequential

Parallel

Pipelined

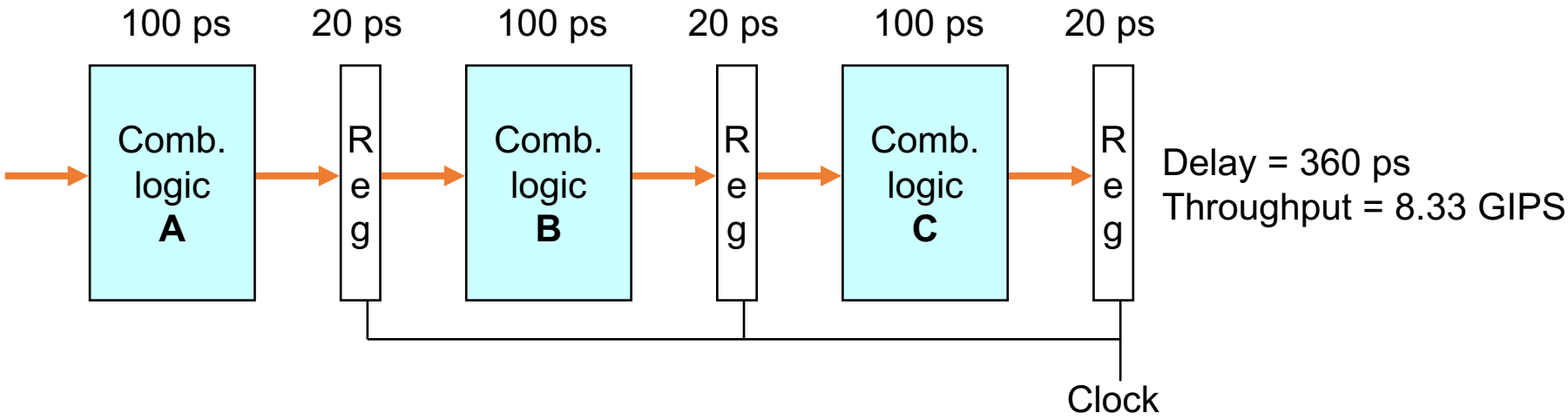
- **Idea**
  - Divide process into independent stages
  - Move objects through stages in sequence
  - At any given times, multiple objects being processed

Computational Example



- **System**
  - Computation requires total of 300 picoseconds (1ps =10^-3ns)
  - Additional 20 picoseconds to save result in register
  - Must have clock cycle of at least 320 ps
  - Throughput calculation P283

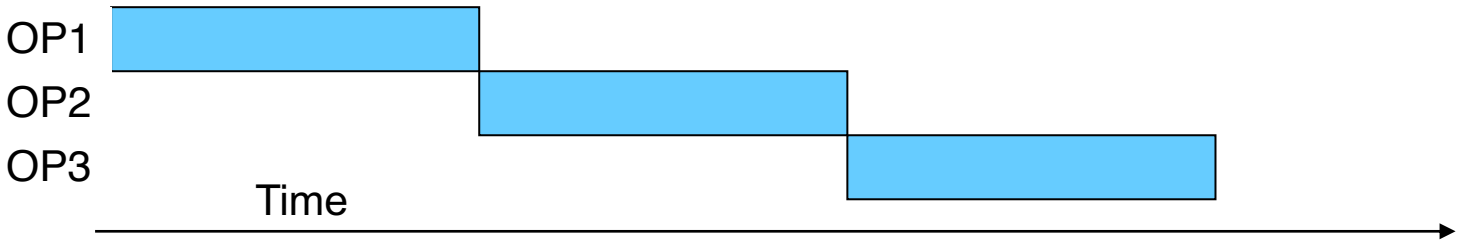
3-Way Pipelined Version



- **System**
  - Divide combinational logic into 3 blocks of 100 ps each
  - Can begin new operation as soon as previous one passes through stage A.
    - Begin new operation every 120 ps
  - Overall latency increases
    - 360 ps from start to finish

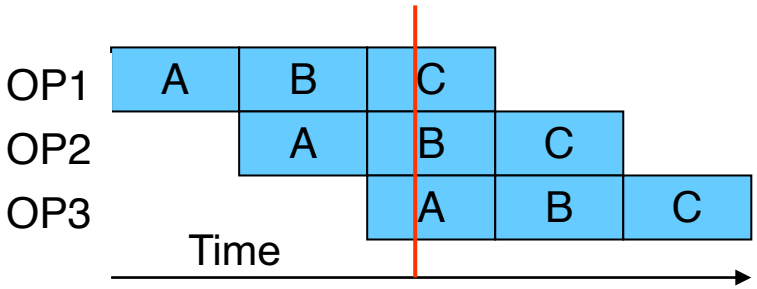
Pipeline Diagrams

- **Unpipelined**



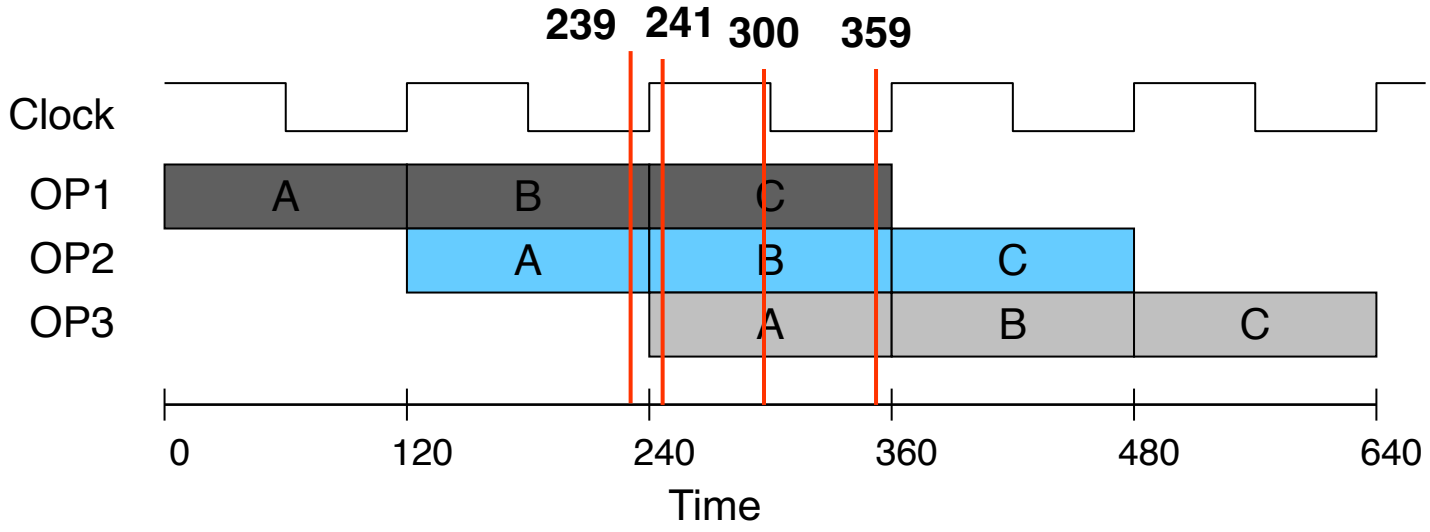
  - Cannot start new operation until previous one completes

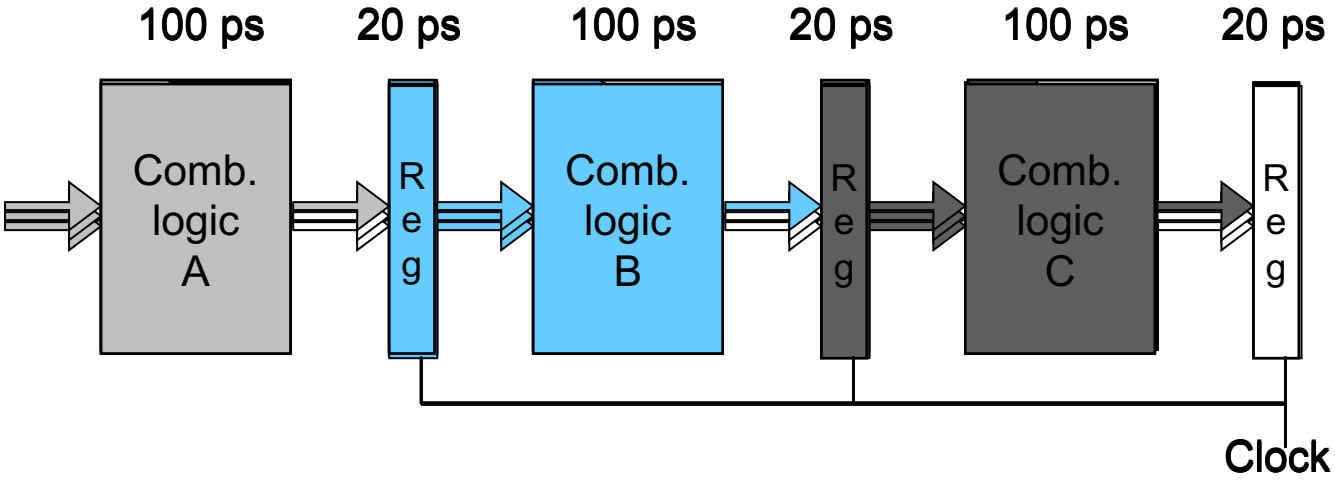
- **3-Way Pipelined**



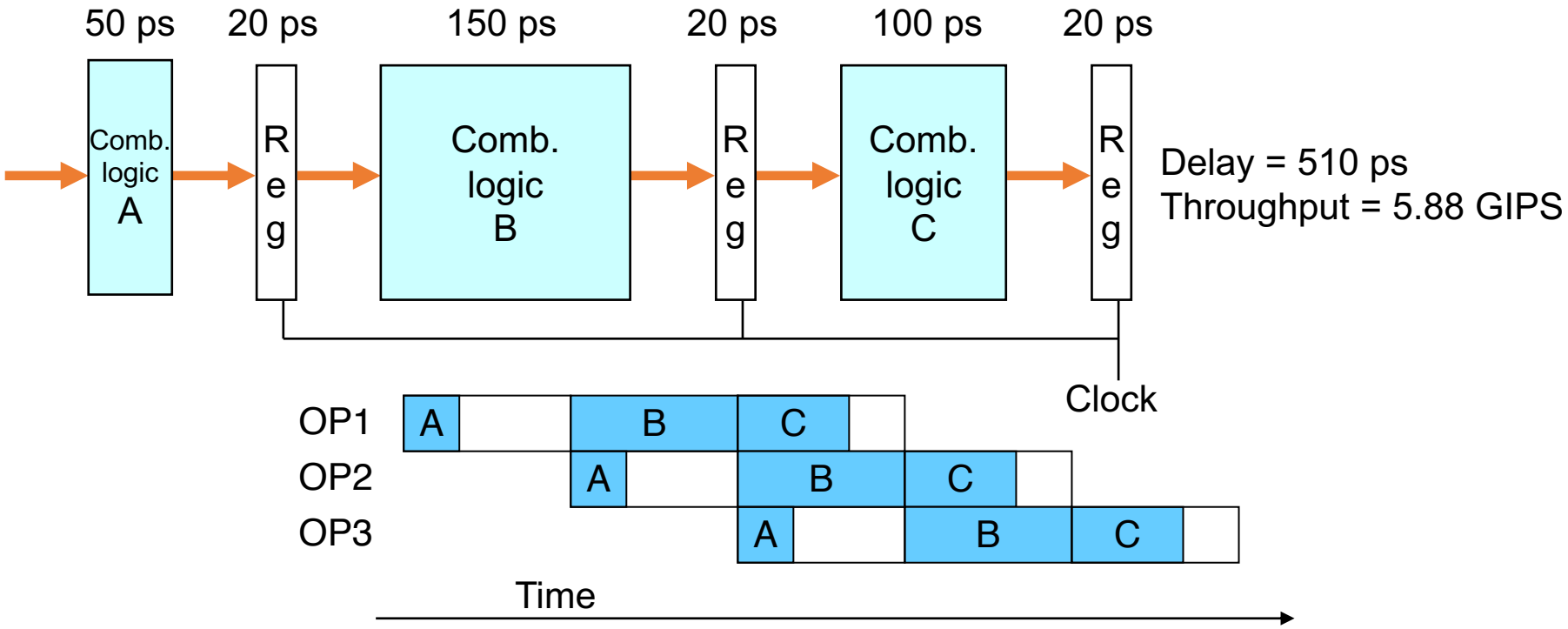
  - Up to 3 operations in process simultaneously

Operating a Pipeline





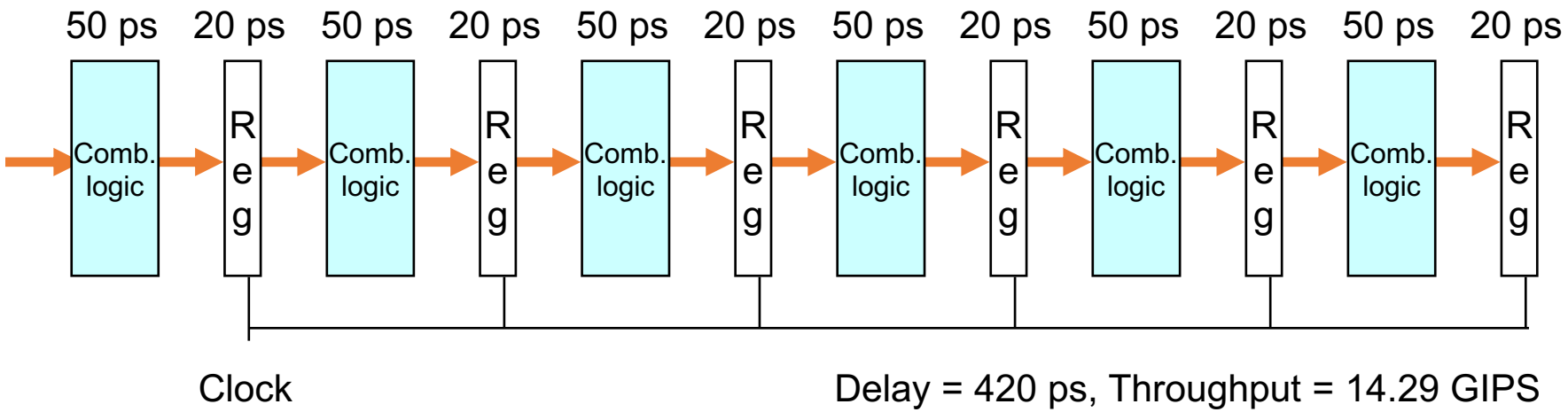
Limitations: Nonuniform Delays



- Throughput limited by slowest stage
- Other stages sit idle for much of the time
- Challenging to partition system into balanced stages

How to deal with an imbalanced pipeline?

Limitations: Register Overhead

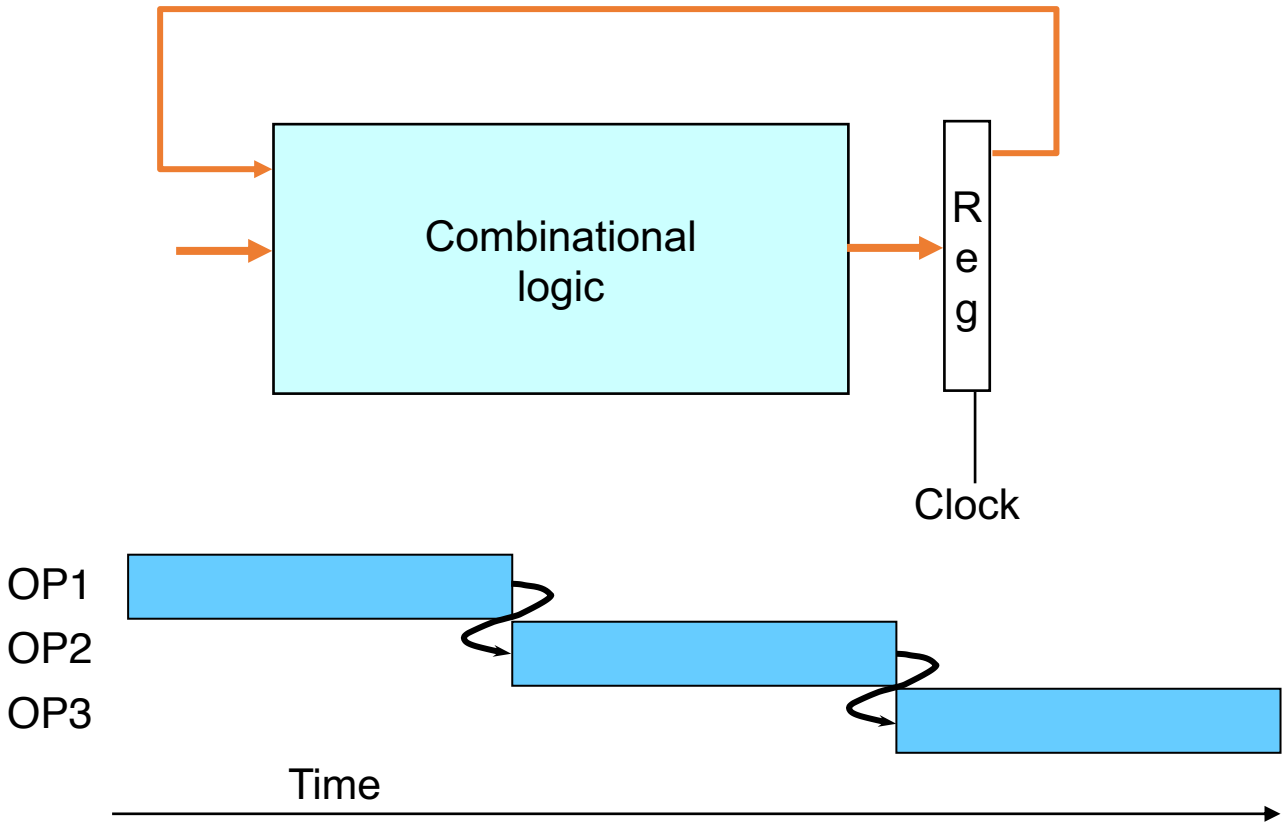


- As try to deepen pipeline, **overhead of loading registers** becomes more significant
- Percentage of clock cycle spent loading register:
  - 1-stage pipeline: 6.25%
  - 3-stage pipeline: 16.67%
  - 6-stage pipeline: 28.57%
- High speeds of modern processor designs obtained through very deep pipelining

Quiz Time!

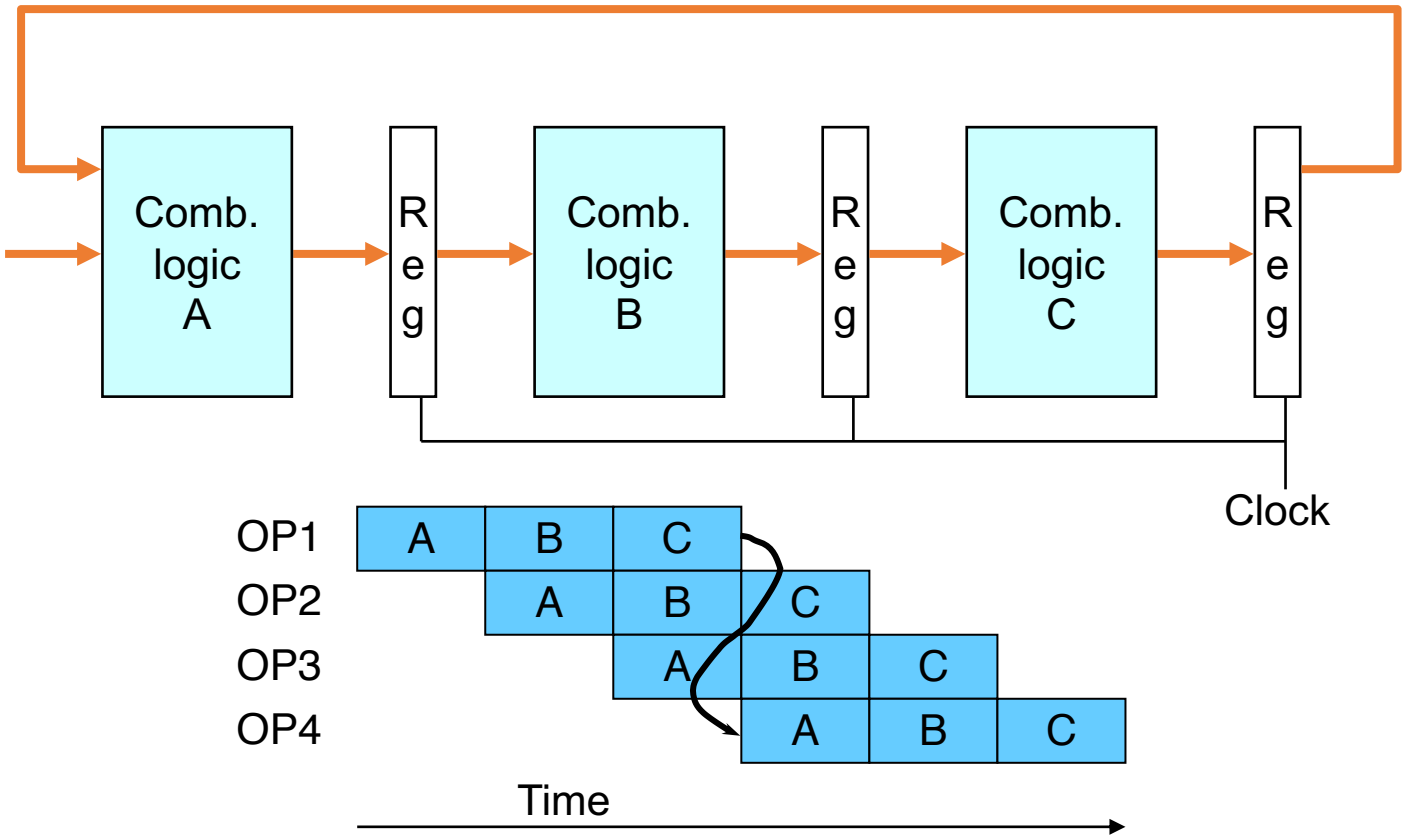
Exercise 4.28

Data Dependencies



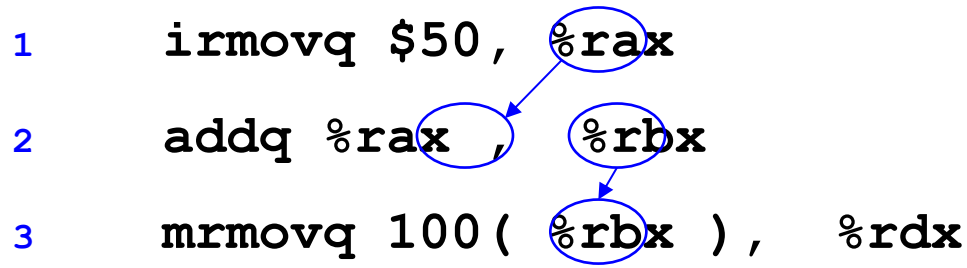
- **System**
  - Each operation depends on result from preceding one

Data Hazards



- Result does not feed back around in time for next operation
- Pipelining has changed behavior of system

Data Dependencies in Processors



- Result from one instruction used as operand for another
  - Read-after-write (RAW) dependency
- Very common in actual programs
- Must make sure our pipeline handles these properly
  - Get correct results
  - Minimize performance impact
- Solutions
  - Adding more lines in the hardware to get the results in the middle, e.g., the output of an ALU can be used directly for the next instruction
  - Inserting more instructions between the instructions with data dependencies

Control Dependency

- Start fetch of new instruction after current one has completed fetch stage
  - Not enough time to reliably determine next instruction

- Predicting the PC: Guess which instruction will follow
  - Recover if prediction was incorrect

Pipeline Summary

- **Concept**
  - Break instruction execution into 5 stages
  - Run instructions through in pipelined mode
- **Limitations**
  - Can't handle dependencies between instructions when instructions follow too closely
  - Data dependencies
    - One instruction writes register, later one reads it
  - Control dependency
    - Instruction sets PC in way that pipeline did not predict correctly
    - Mispredicted branch and return
- **Fixing the Pipeline**
  - In the class of Computer Organization and Architecture