Systems I

Pipelining I

Topics

- Pipelining principles
- Pipeline overheads
- Pipeline registers and stages

Overview

What's wrong with the sequential (SEQ) Y86?

- It's slow!
- Each piece of hardware is used only a small fraction of time
- We would like to find a way to get more performance with only a little more hardware

General Principles of Pipelining

- Goal
- Difficulties

Creating a Pipelined Y86 Processor

- Rearranging SEQ
- Inserting pipeline registers
- Problems with data and control hazards

Real-World Pipelines: Car Washes

Sequential

Parallel

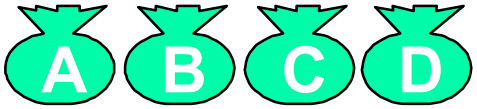
Pipelined

Idea

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

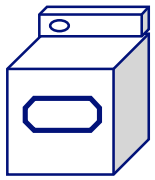
Laundry example

Ann, Brian, Cathy, Dave
each have one load of clothes
to wash, dry, and fold



Washer takes 30 minutes

Dryer takes 30 minutes

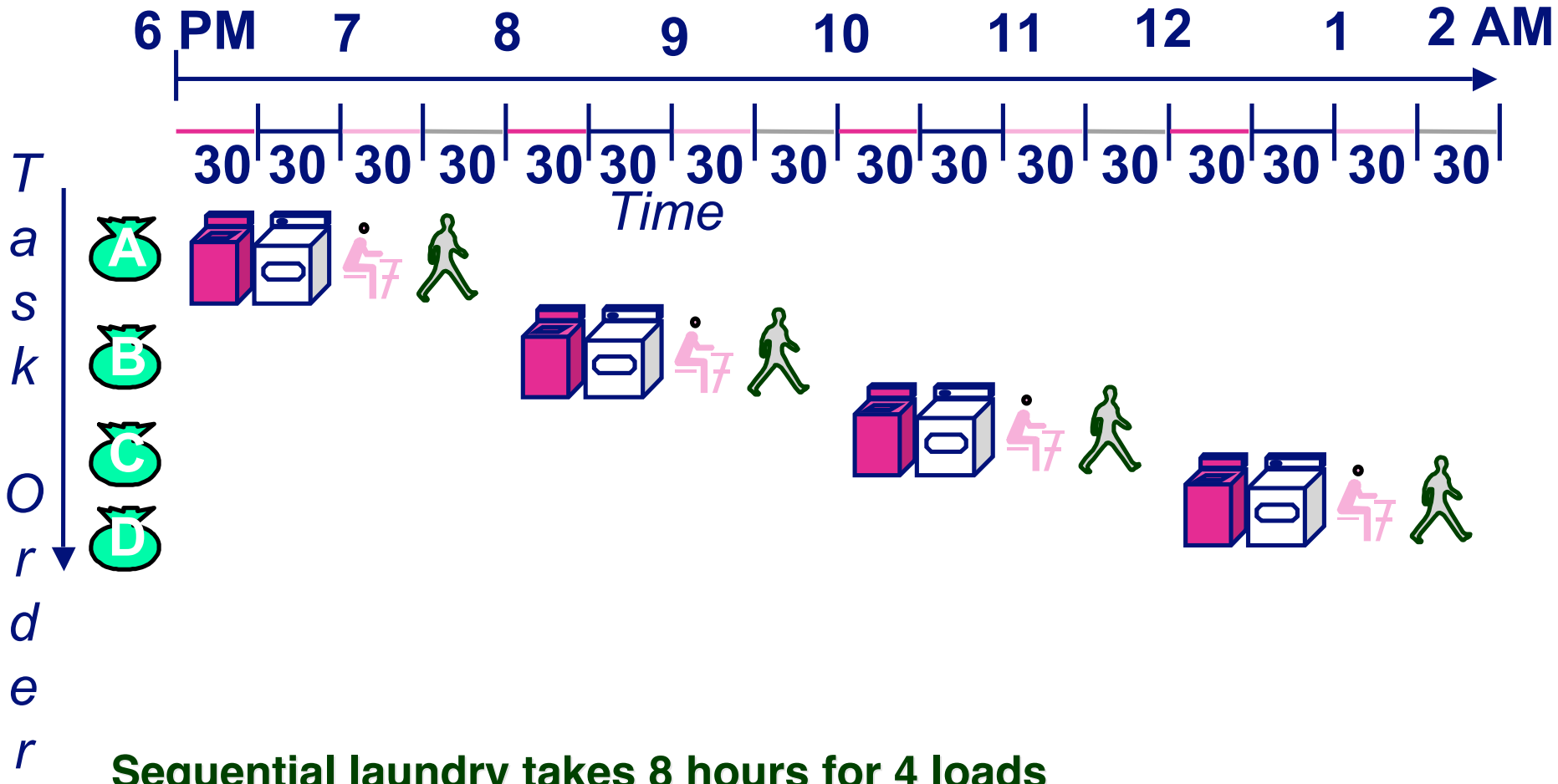


"Folder" takes 30 minutes

"Stasher" takes 30 minutes
to put clothes into drawers

Slide courtesy of D. Patterson

Sequential Laundry

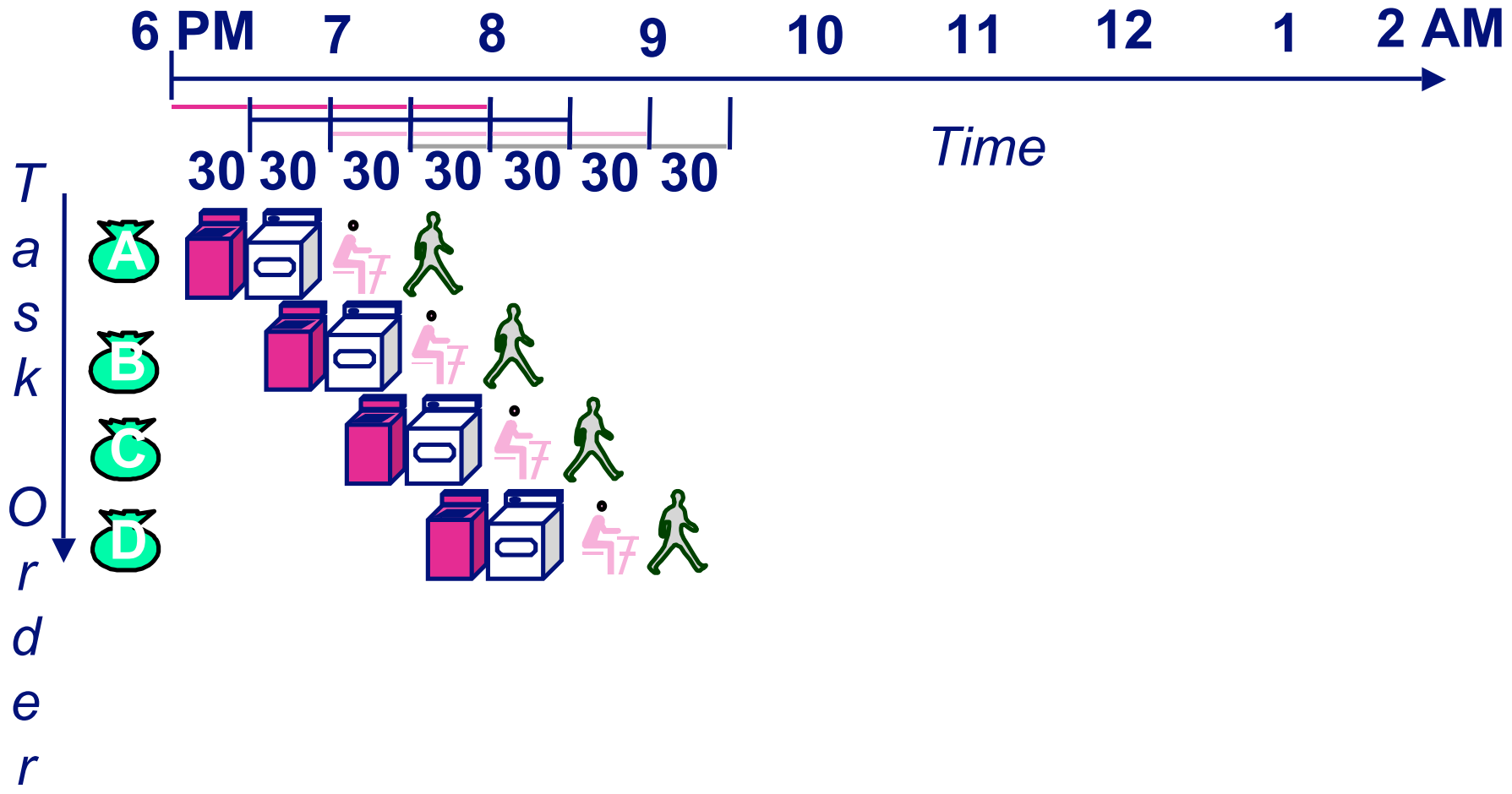


Sequential laundry takes 8 hours for 4 loads

If they learned pipelining, how long would laundry take?

Slide courtesy of D. Patterson

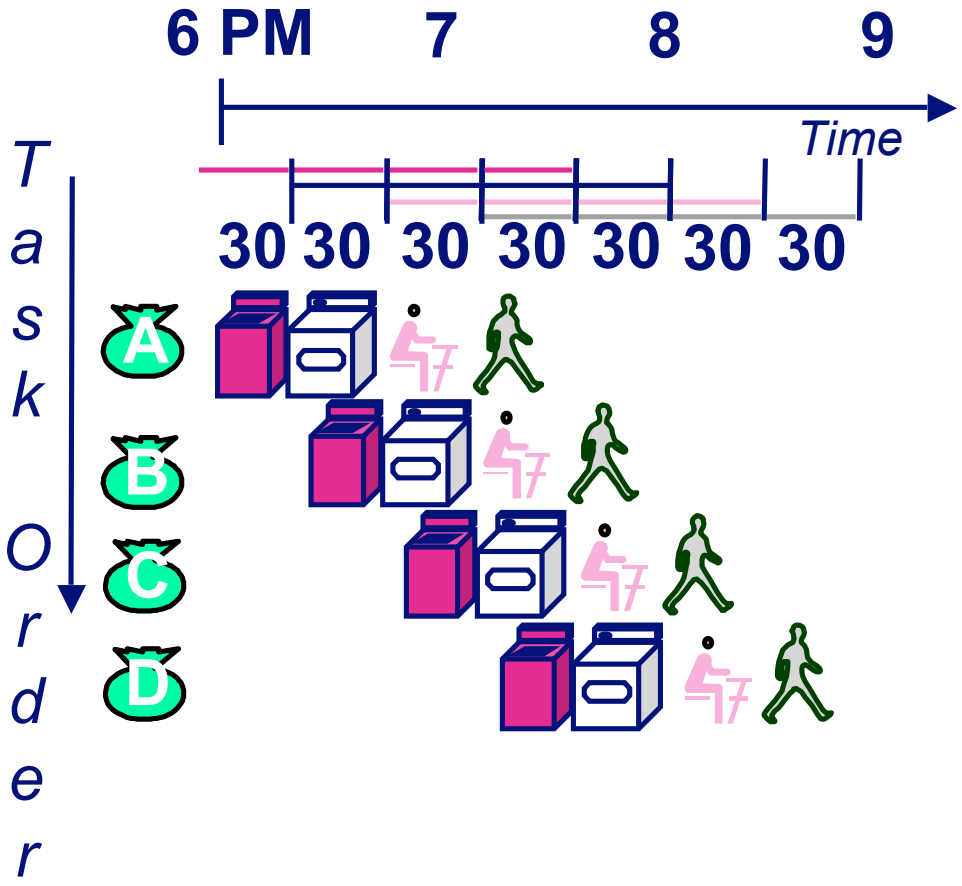
Pipelined Laundry: Start ASAP



Pipelined laundry takes 3.5 hours for 4 loads!

Slide courtesy of D. Patterson

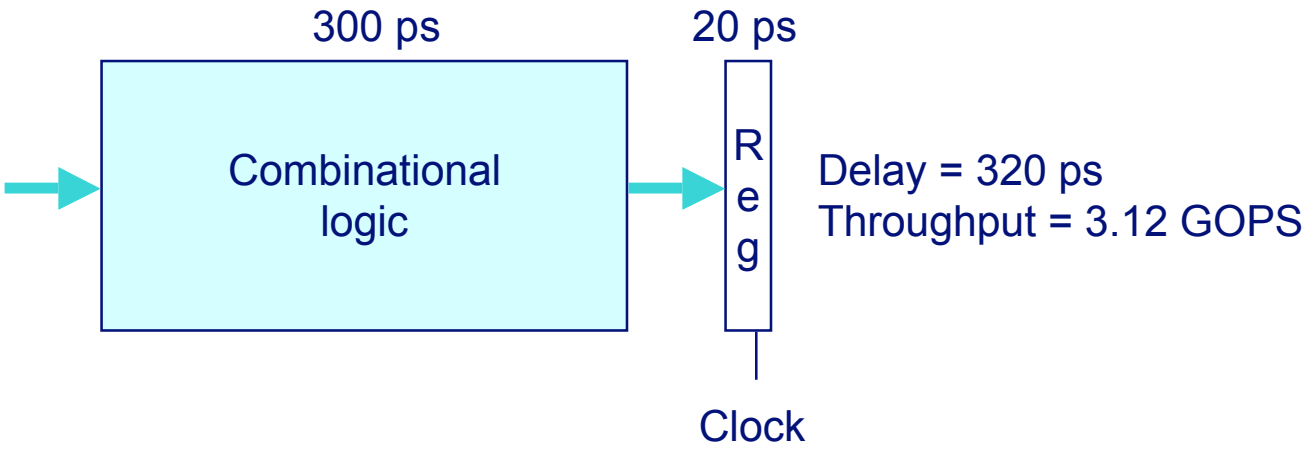
Pipelining Lessons



- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Multiple tasks operating simultaneously using different resources
- Potential speedup = Number pipe stages
- Pipeline rate limited by slowest pipeline stage
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup
- Stall for Dependences

Slide courtesy of D. Patterson

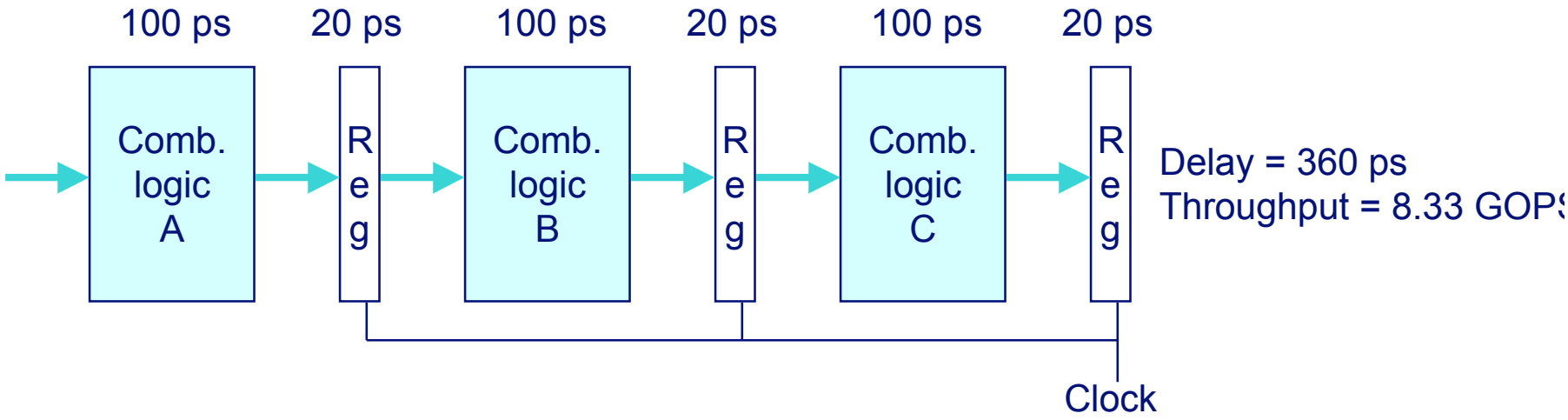
Computational Example



System

- Computation requires total of 300 picoseconds
- Additional 20 picoseconds to save result in register
- Must have clock cycle of at least 320 ps

3-Way Pipelined Version

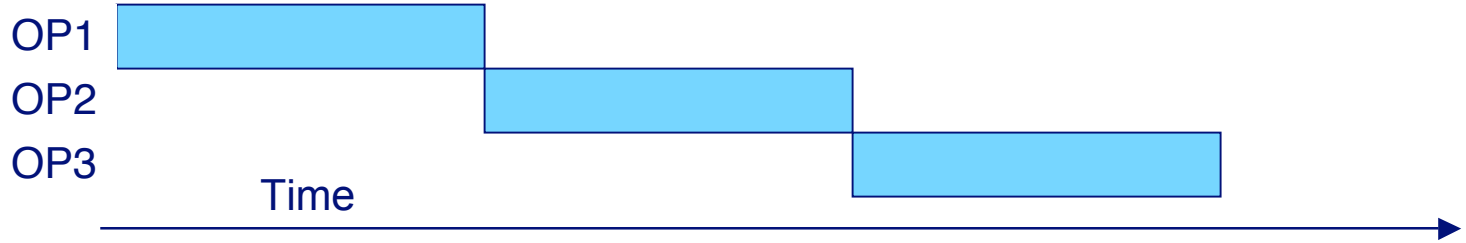


System

- 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

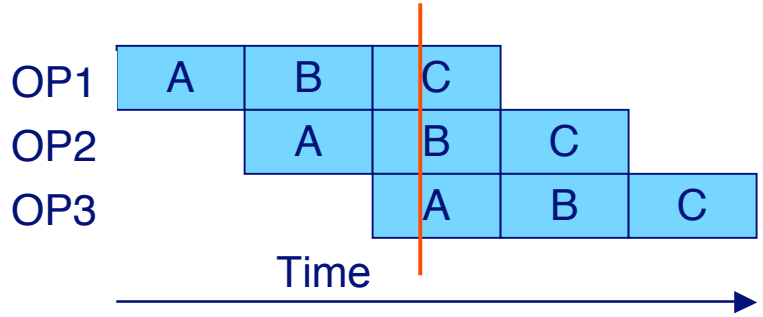
Pipeline Diagrams

Unpipelined



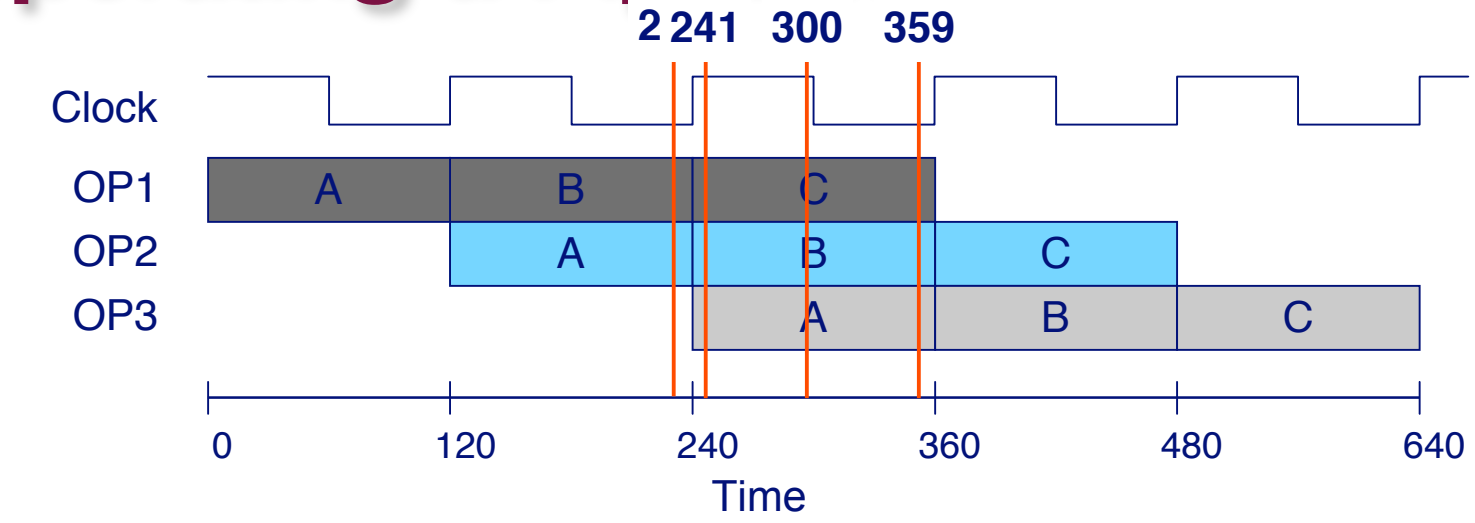
- Cannot start new operation until previous one completes

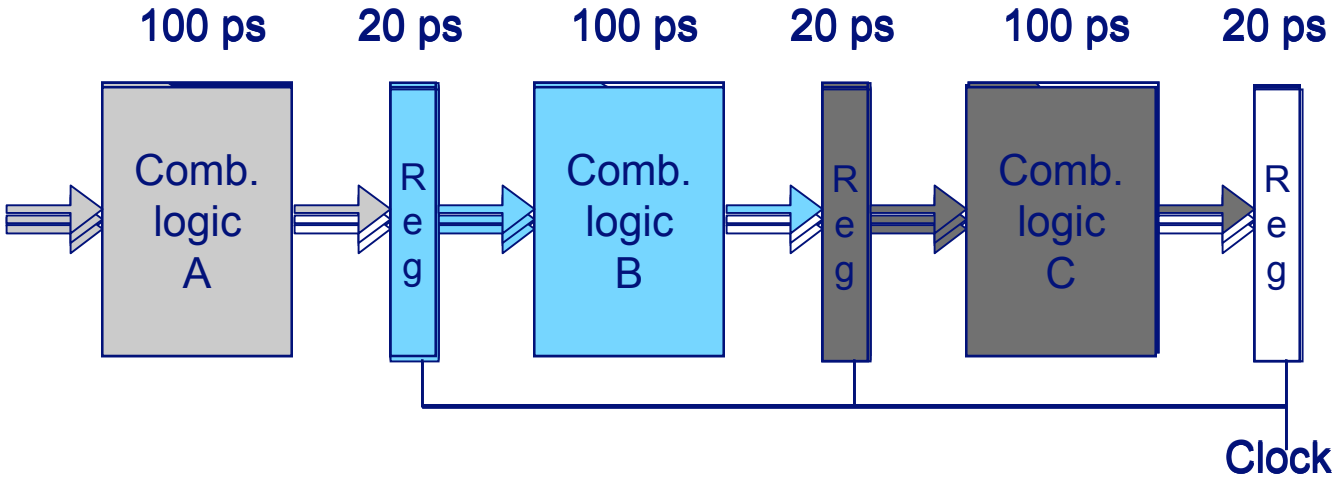
3-Way Pipelined



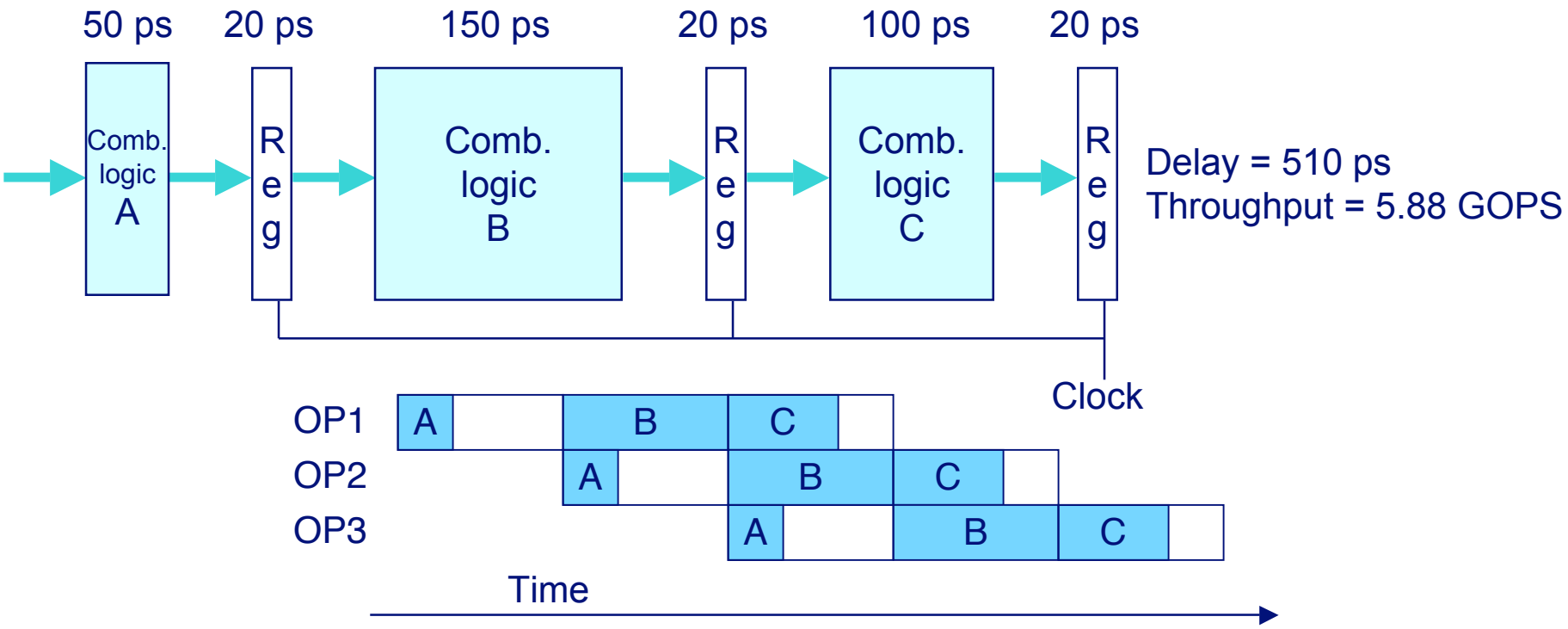
- Up to 3 operations in process simultaneously

Operating a Pipeline



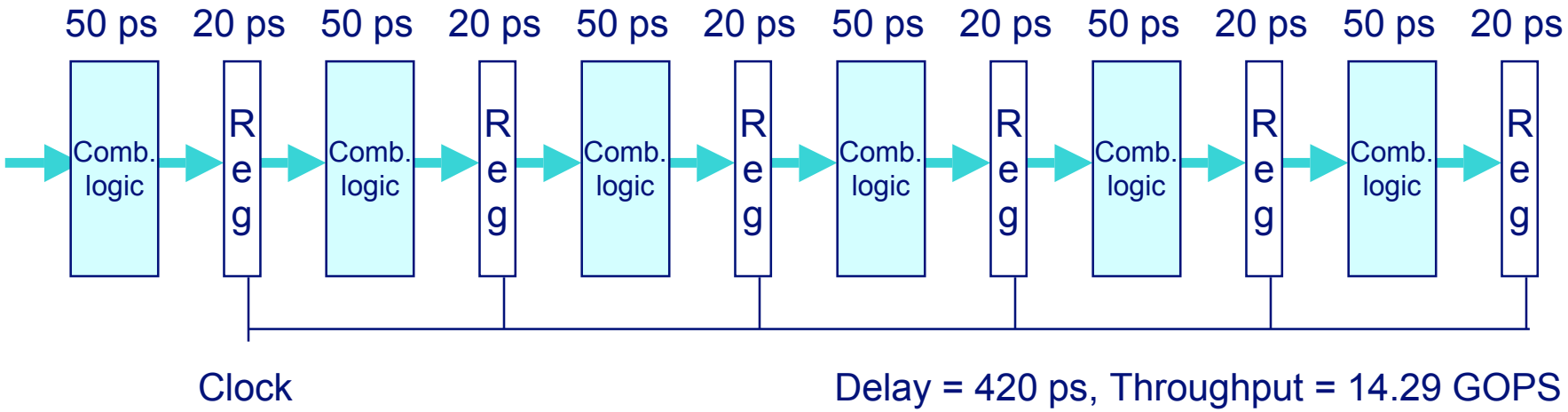


Limitations: Nonuniform Delays



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

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

Revisiting the Performance Eqn

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Seconds}}{\text{Cycle}}$$

Instruction Count: No change

Clock Cycle Time

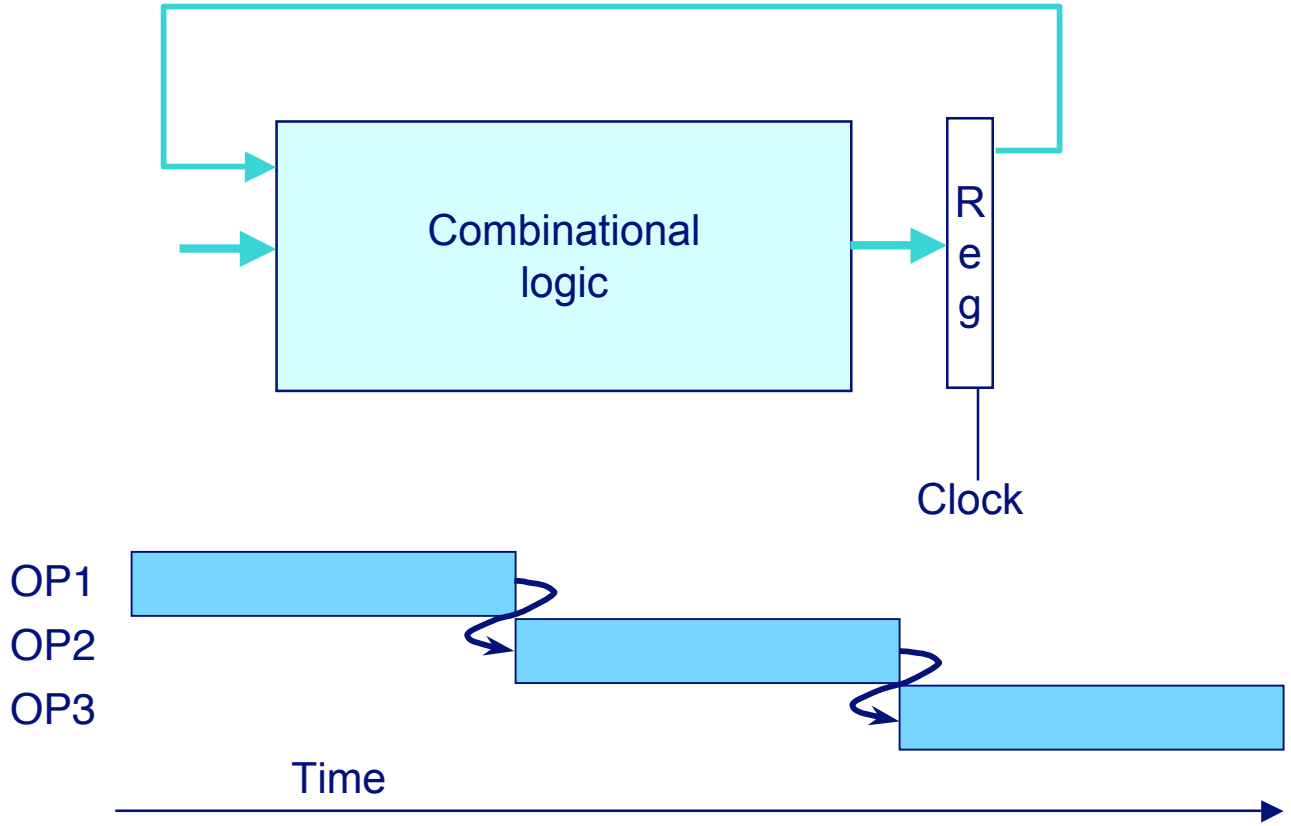
- Improves by factor of almost N for N-deep pipeline
- Not quite factor of N due to pipeline overheads

Cycles Per Instruction

- In ideal world, CPI would stay the same
- An individual instruction takes N cycles
- But we have N instructions in flight at a time
- So - average $CPI_{pipe} = CPI_{no\_pipe} * N/N$

Thus performance can improve by up to factor of N

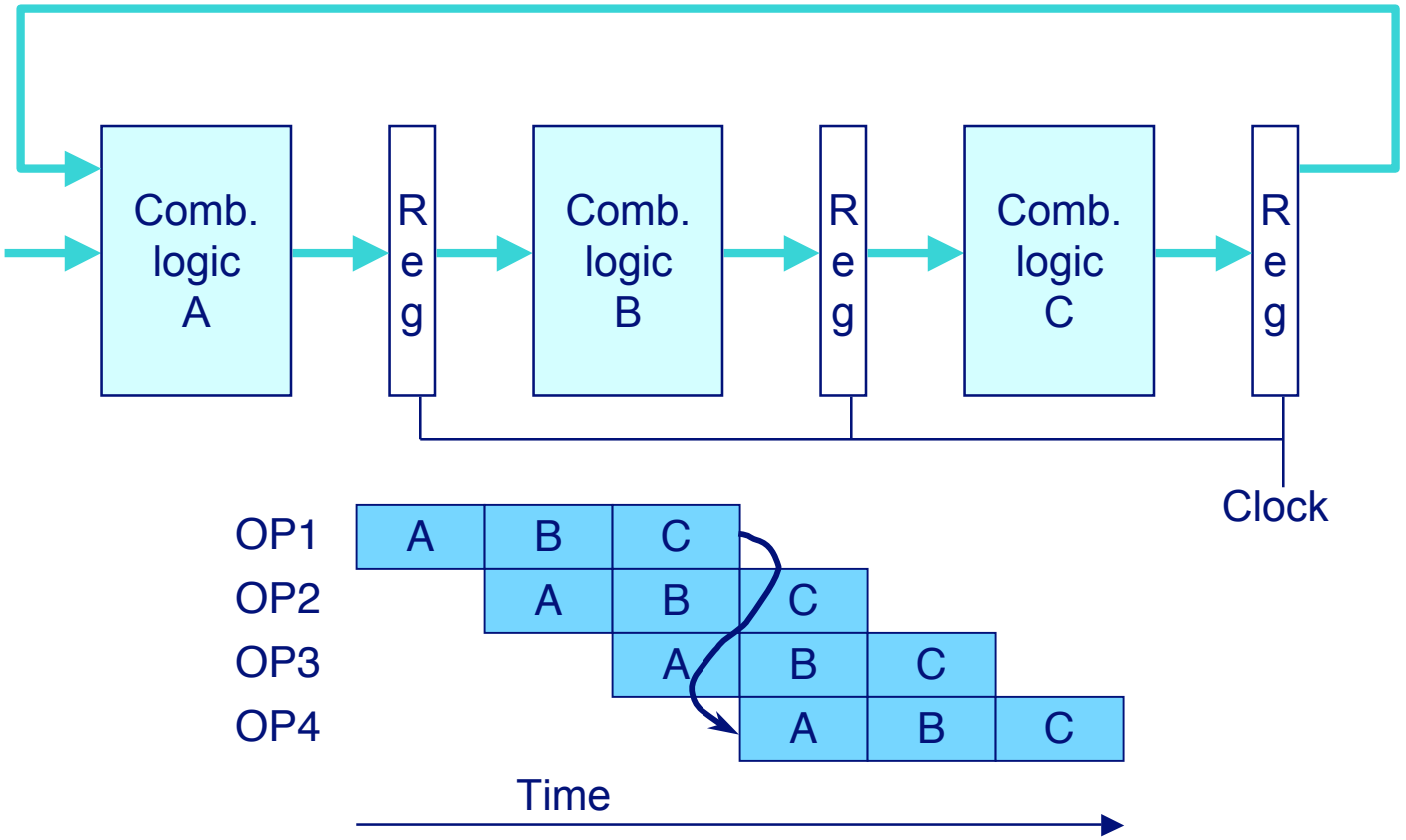
Data Dependencies



System

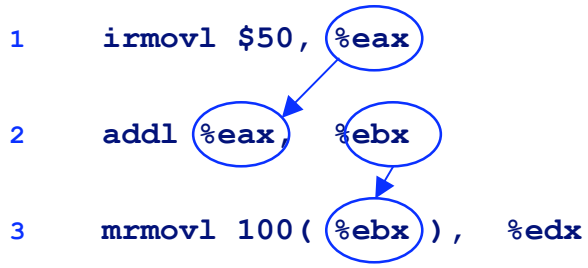
- **Each operation depends on result from preceding one**

Data Hazards



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

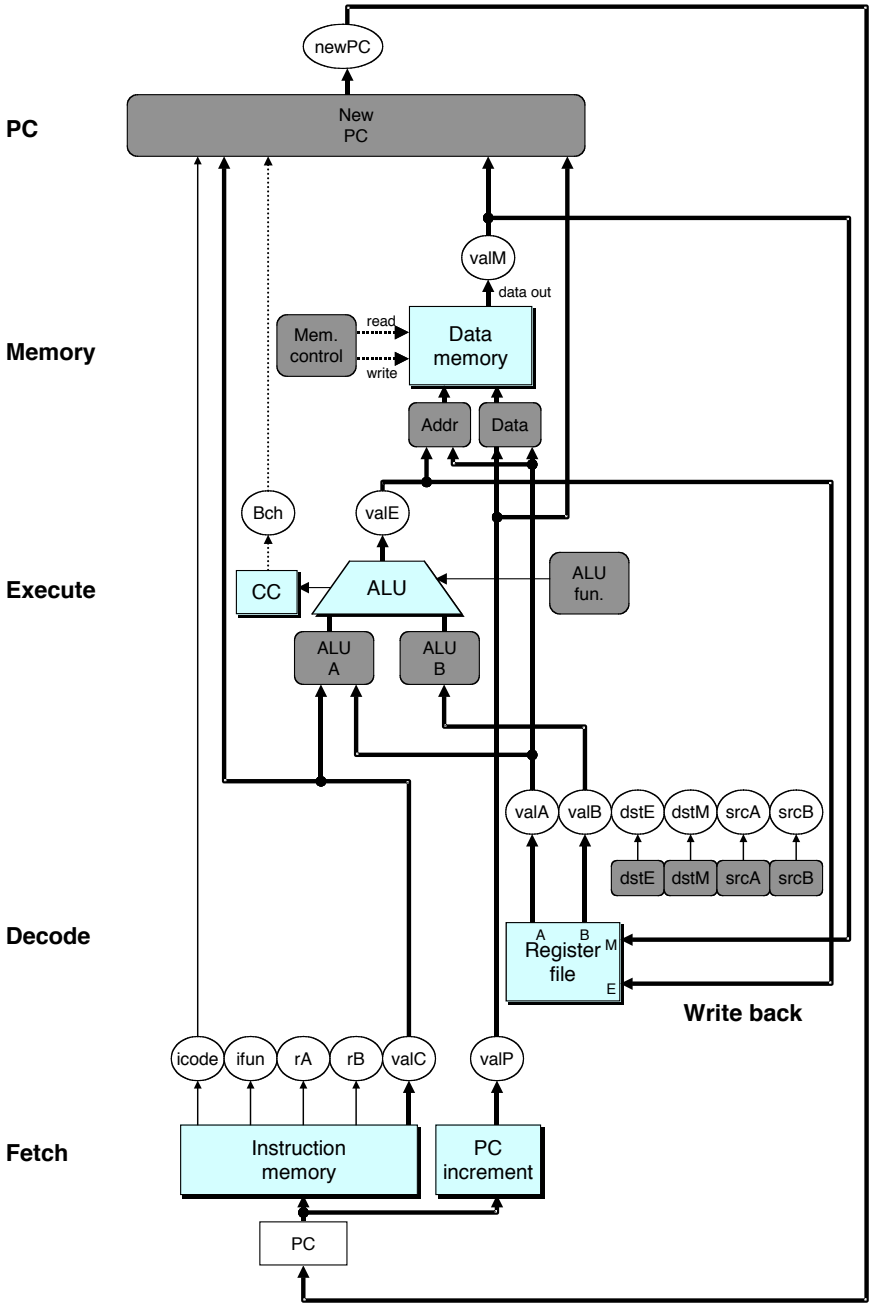
Data Dependencies in Processors



- 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

SEQ Hardware

- Stages occur in sequence
- One operation in process at a time
- One stage for each logical pipeline operation
  - Fetch (get next instruction from memory)
  - Decode (figure out what instruction does and get values from regfile)
  - Execute (compute)
  - Memory (access data memory if necessary)
  - Write back (write any instruction result to regfile)



SEQ+ Hardware

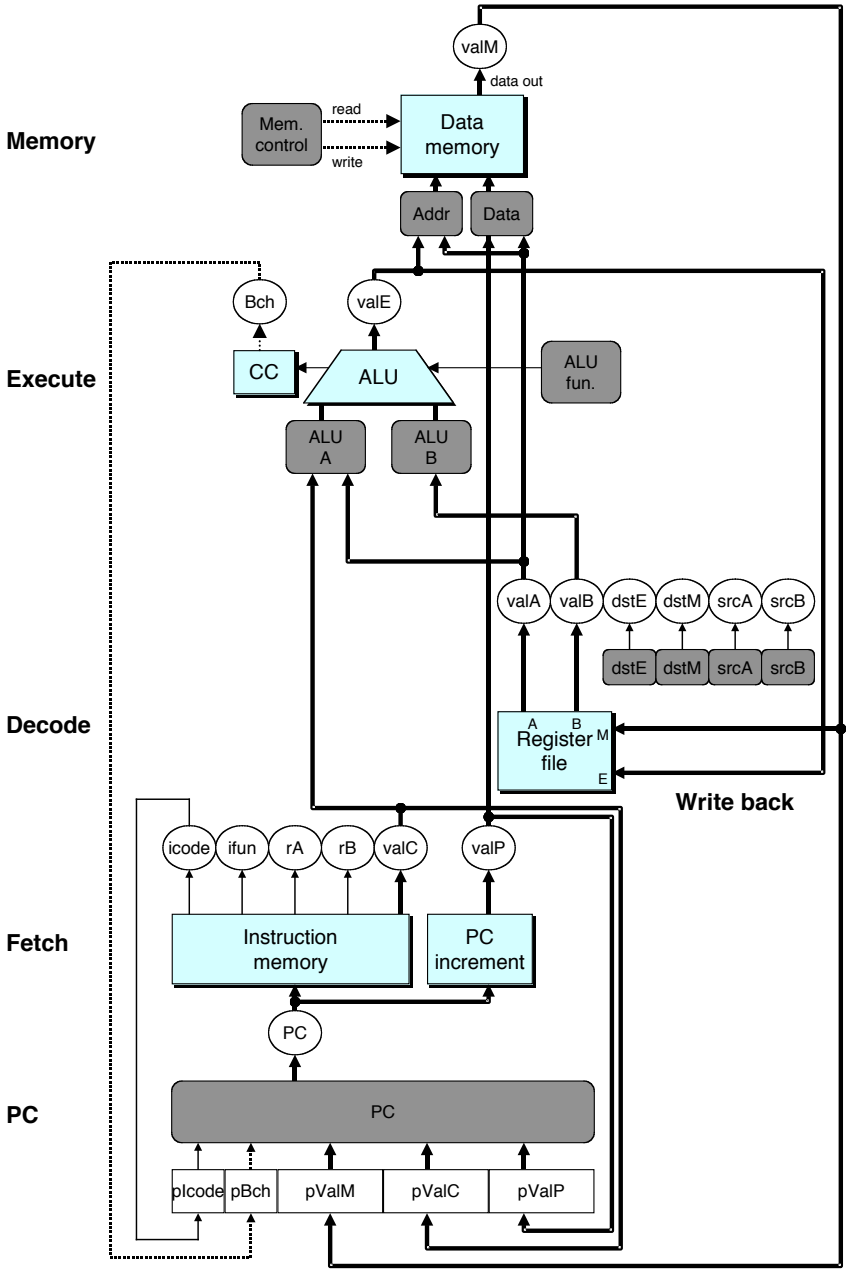
- Still sequential implementation
- Reorder PC stage to put at beginning

PC Stage

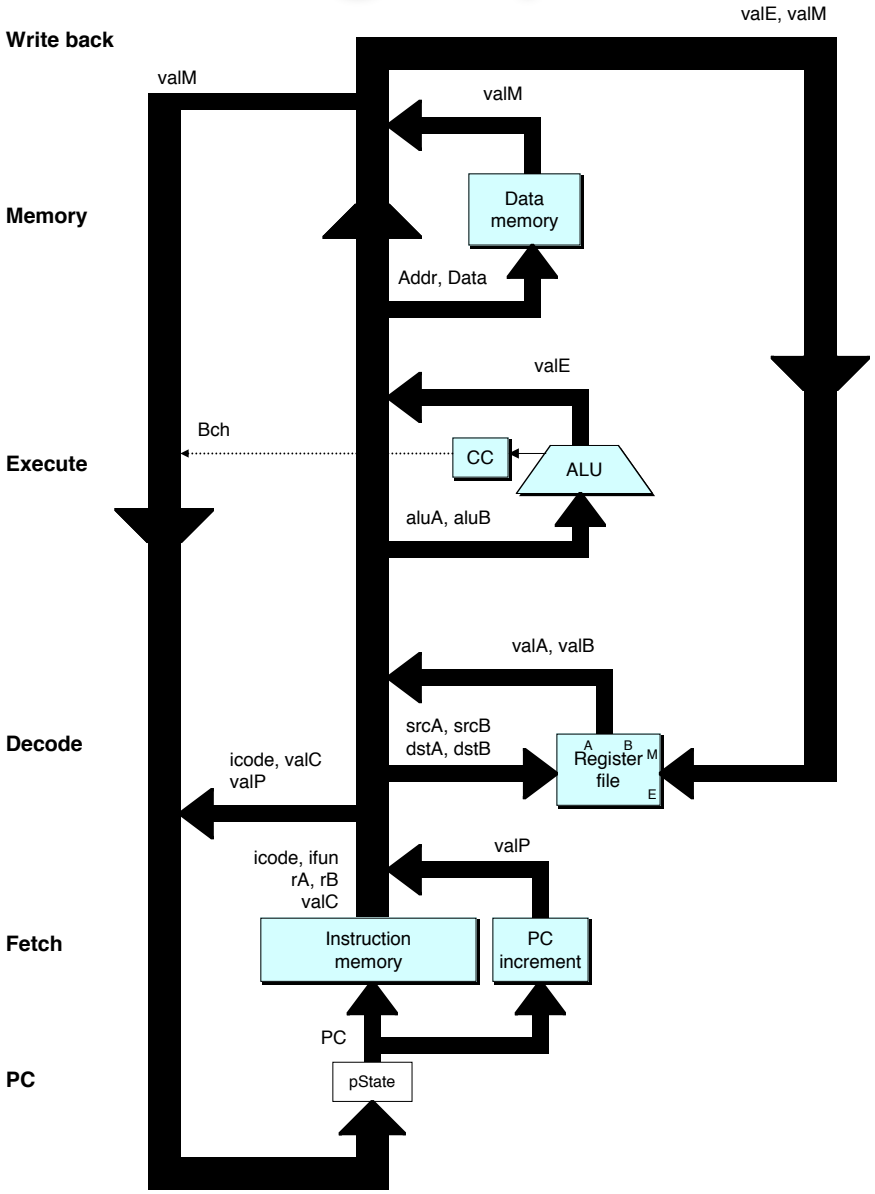
- Task is to select PC for current instruction
- Based on results computed by previous instruction

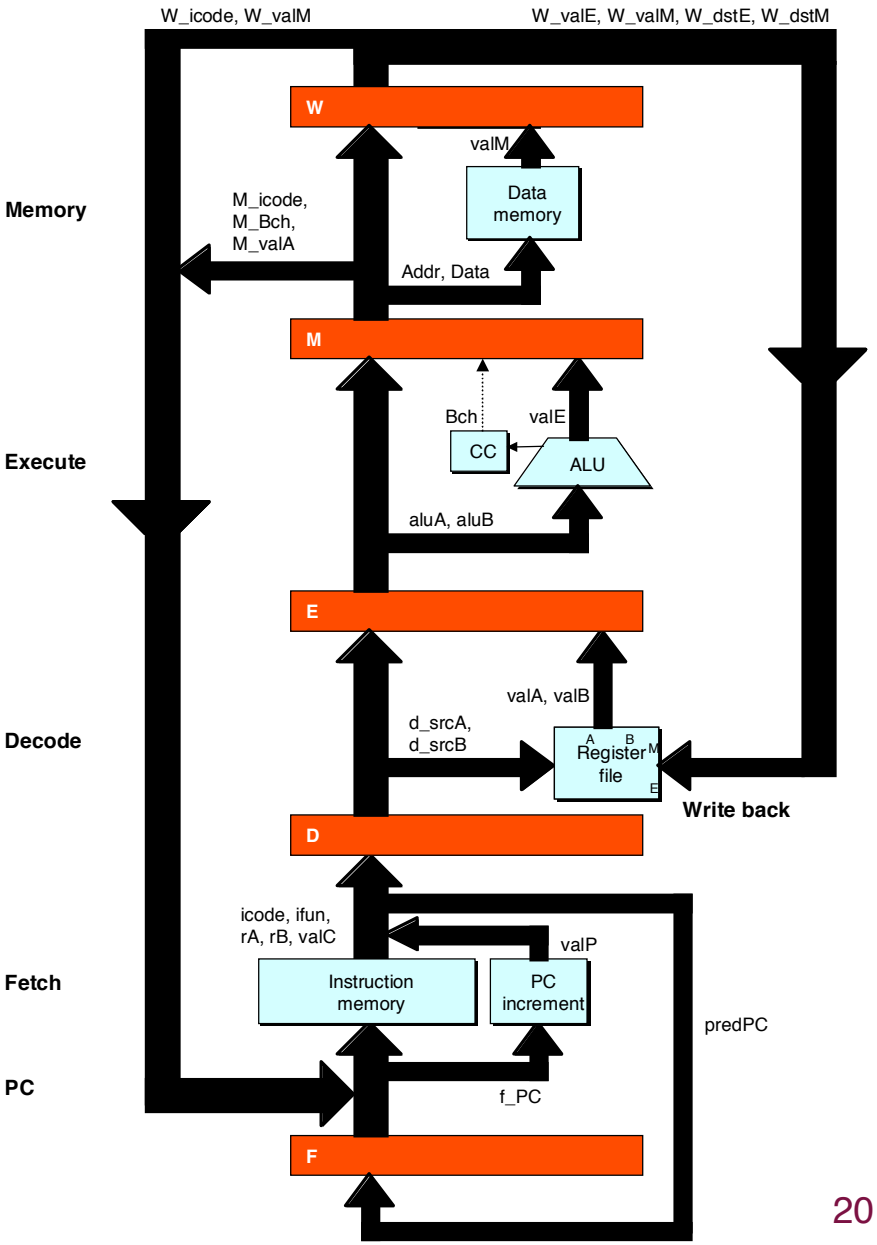
Processor State

- PC is no longer stored in register
- But, can determine PC based on other stored information



Adding Pipeline Registers





Pipeline Stages

Fetch

- Select current PC
- Read instruction
- Compute incremented PC

Decode

- Read program registers

Execute

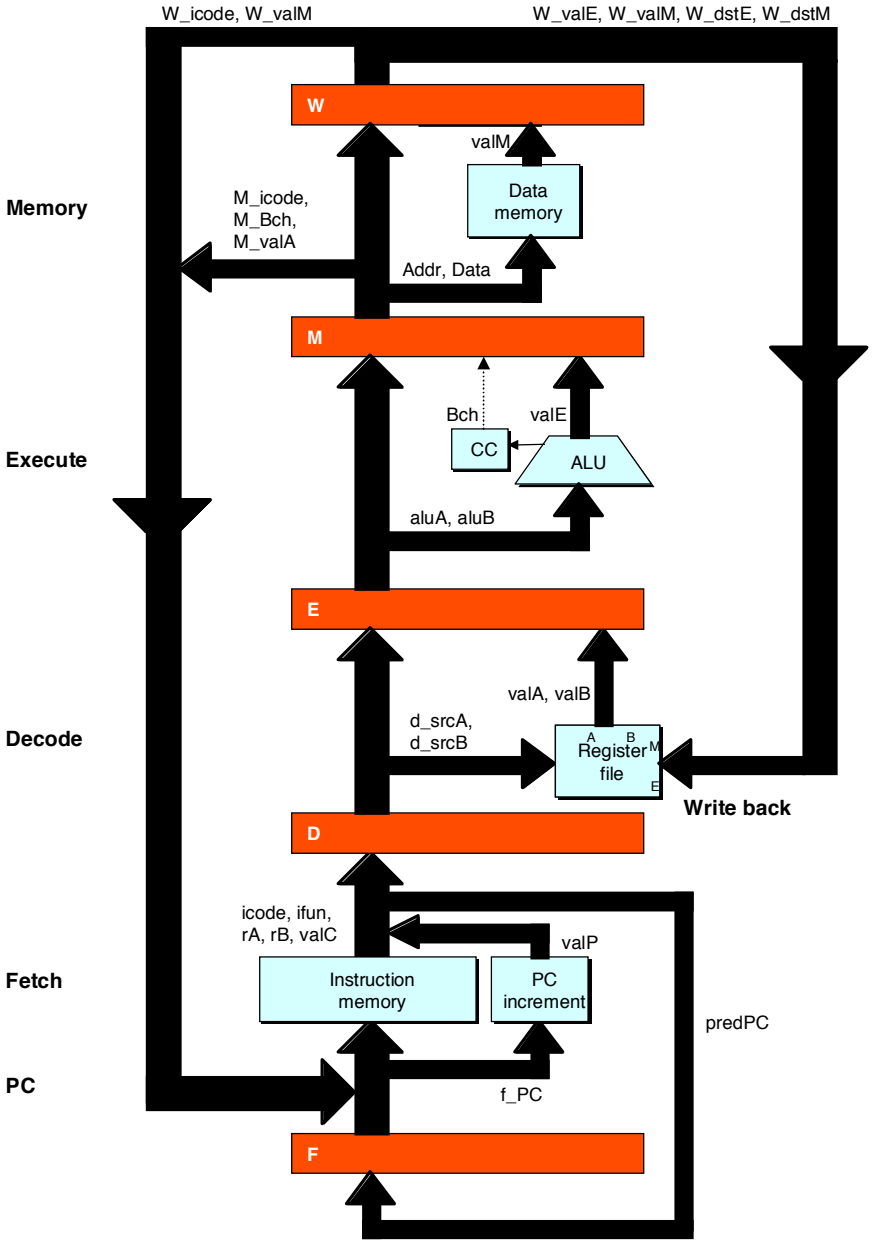
- Operate ALU

Memory

- Read or write data memory

Write Back

- Update register file



Summary

Today

- Pipelining principles (assembly line)
- Overheads due to imperfect pipelining
- Breaking instruction execution into sequence of stages

Next Time

- Pipelining hardware: registers and feedback paths
- Difficulties with pipelines: hazards
- Method of mitigating hazards