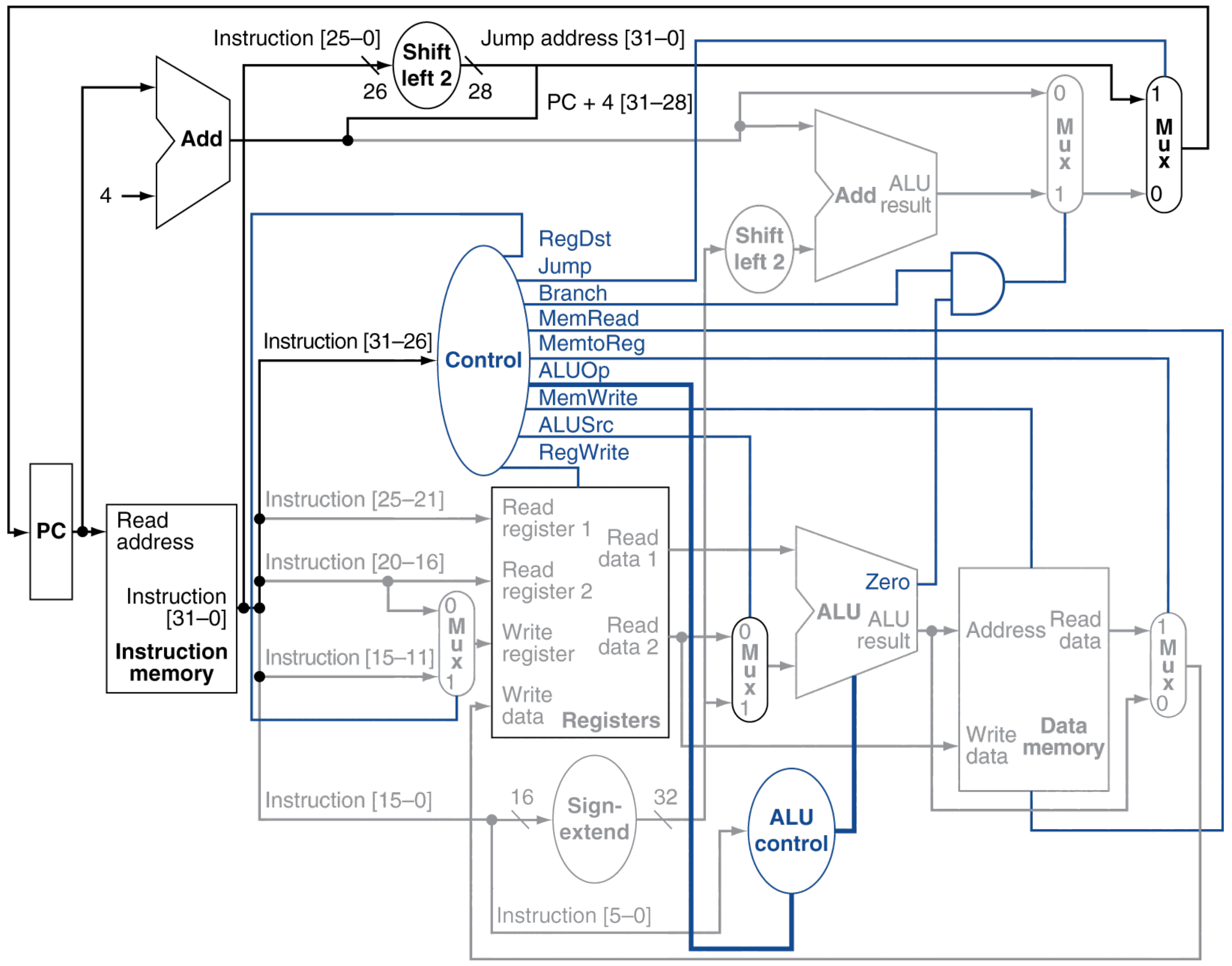
CPT_S 260 Intro to Computer Architecture
Lecture 32

Intro to Pipelining
April 4, 2022

Ganapati Bhat
School of Electrical Engineering and Computer Science
Washington State University

Datapath with Jumps Added



Performance Issues

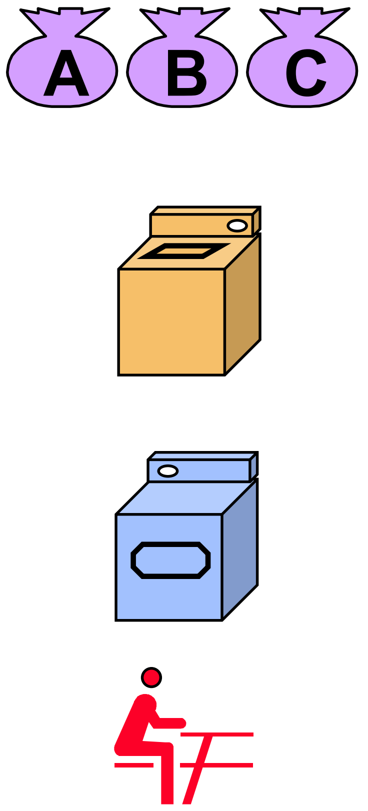
- **Longest delay determines clock period**
  - Critical path: load instruction
  - Instruction memory $\rightarrow$ register file $\rightarrow$ ALU $\rightarrow$ data memory $\rightarrow$ register file
- **Not feasible to vary period for different instructions**
- **Violates design principle**
  - Making the common case fast
- **We will improve performance by pipelining**

Pipelining & Pipelined MIPS Architecture

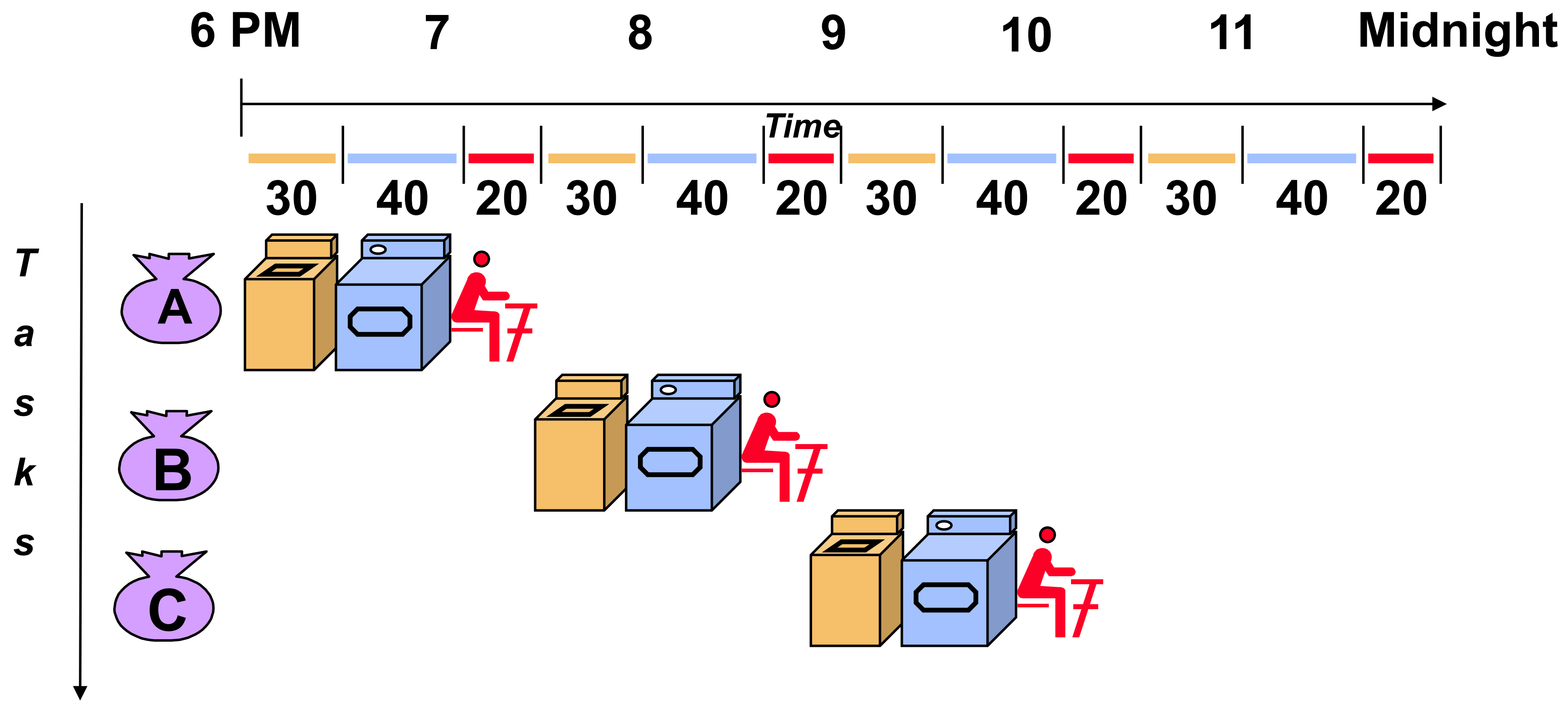
- **Topics:**
  - Pipeline Analogy
  - Pipeline Performance
  - Hazards
    - Structural Hazards
    - Data Hazards
    - Control Hazards
  - MIPS Pipelined Datapath
    - Different Instructions
  - Pipelined Control

Pipelining is Natural!

- Laundry Example
- Three friends each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- Folding takes 20 minutes



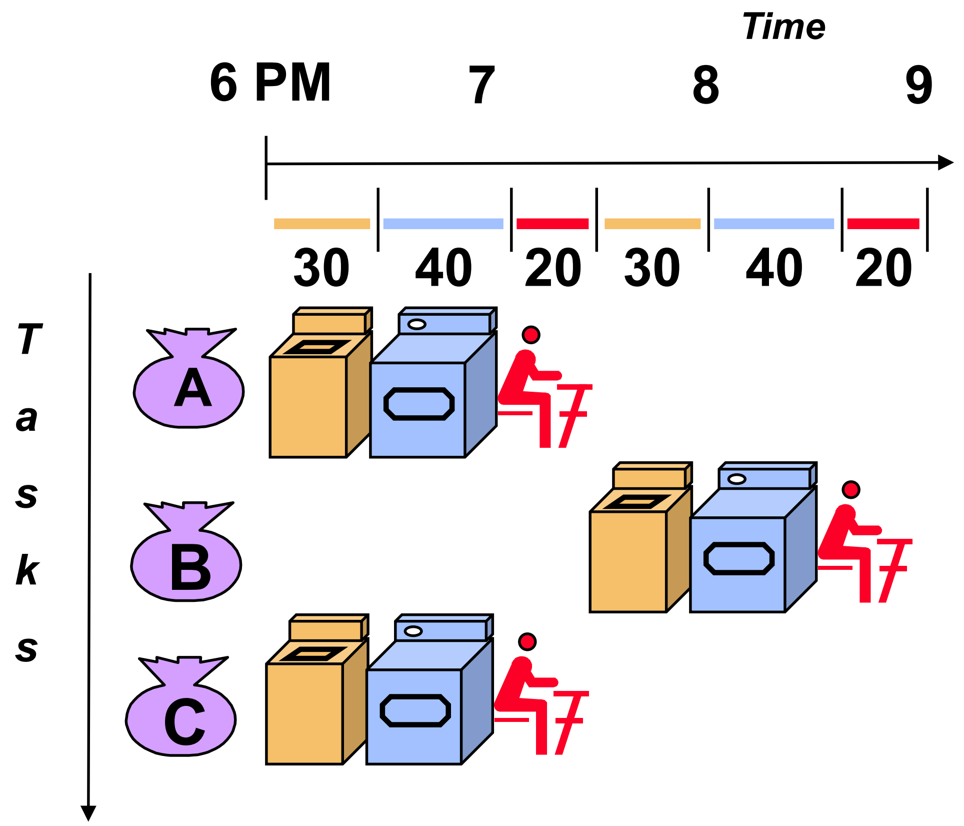
Sequential Laundry



- Sequential laundry takes 4.5 hours

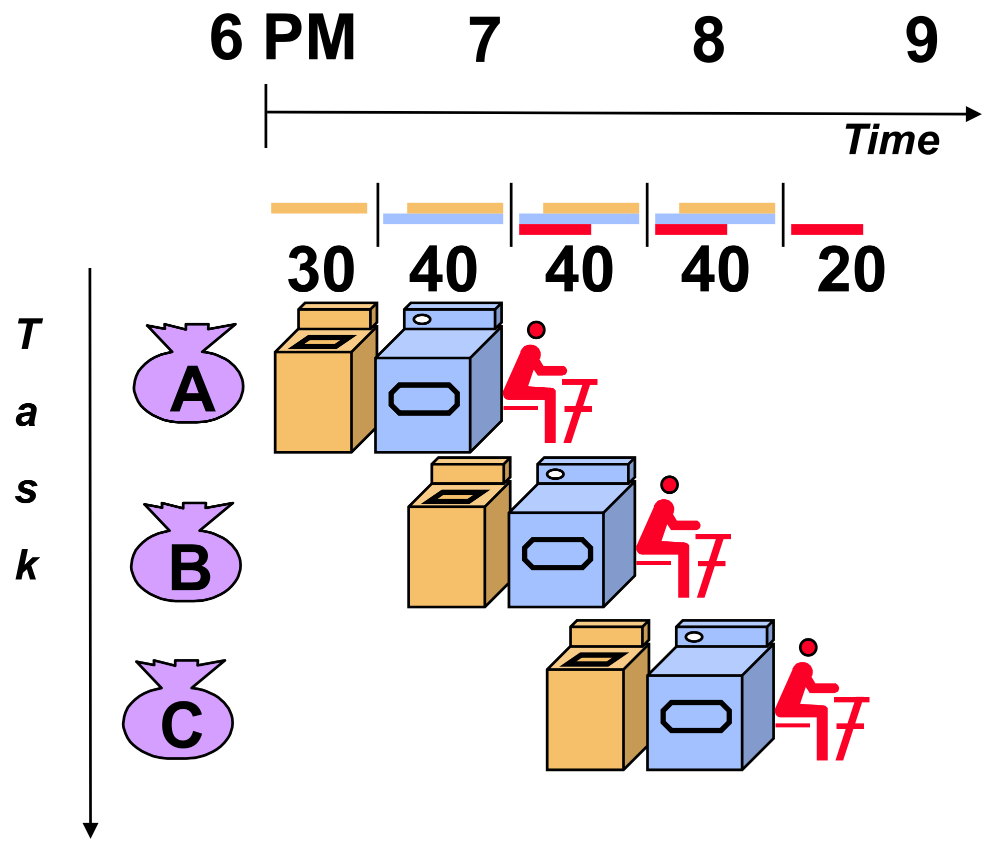
How do we Increase the Throughput?

- Use 2 washers, 2 dryers



- Throughput improves 2x
- Total time is 3 hours
- However,
  - Cost is also doubled
  - Uses more resources

Pipelined Laundry

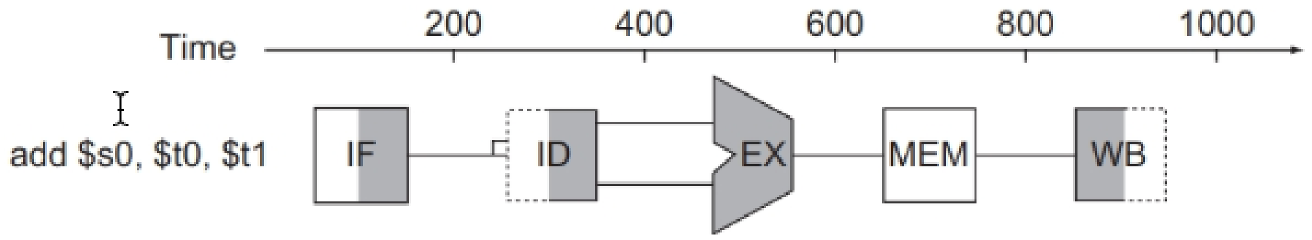


- Pipelined laundry takes 2 hours and 50 minutes

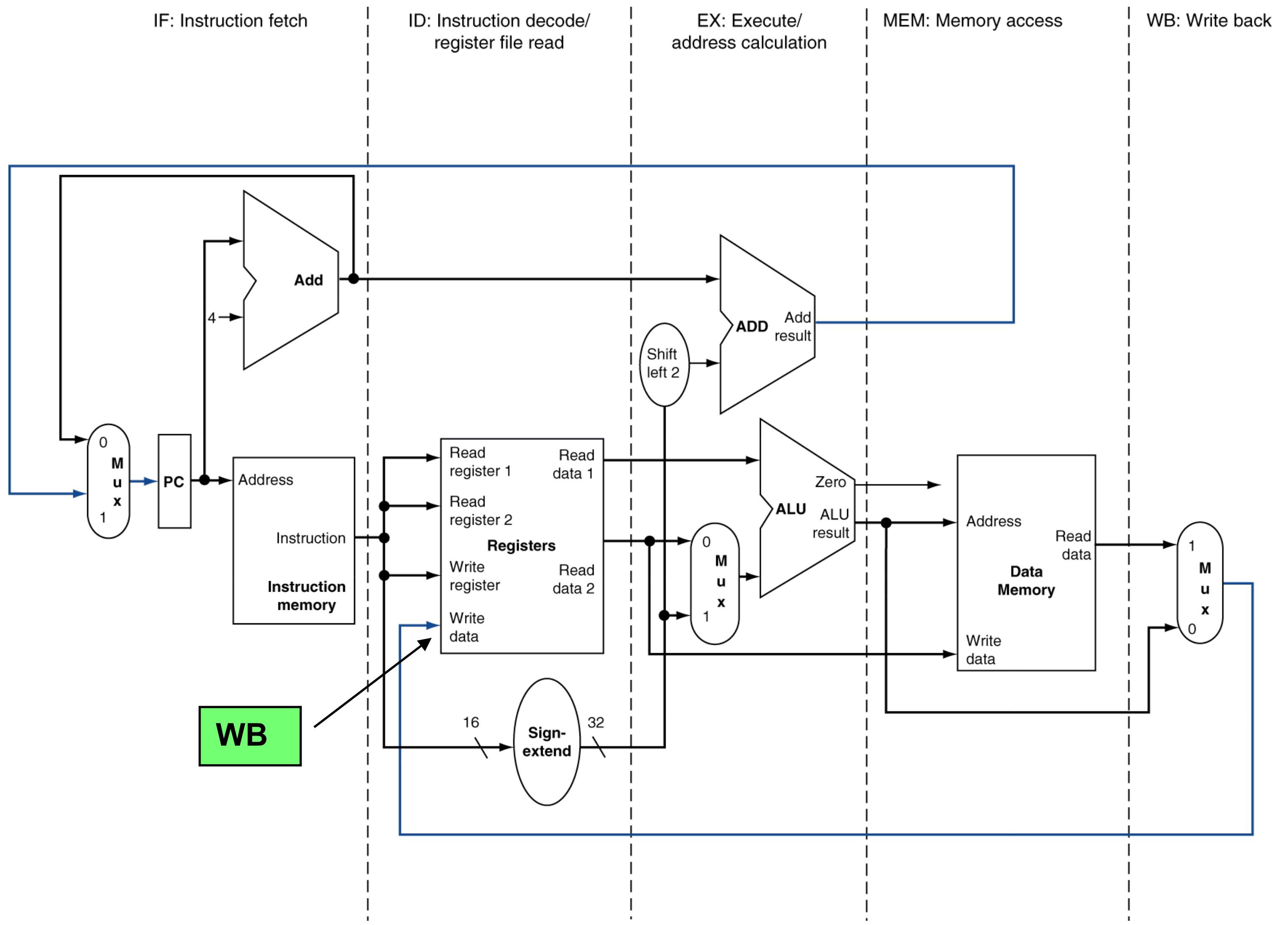
- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously using different resources
- Potential speedup = Number of pipeline stages
- Unbalanced lengths of pipe stages reduces speedup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

MIPS Pipeline

- **Five stages, one step per stage**
    1. IF: Instruction fetch from memory
    2. ID: Instruction decode & register read
    3. EX: Execute operation or calculate address
    4. MEM: Access memory operand
    5. WB: Write result back to register



Overall Idea



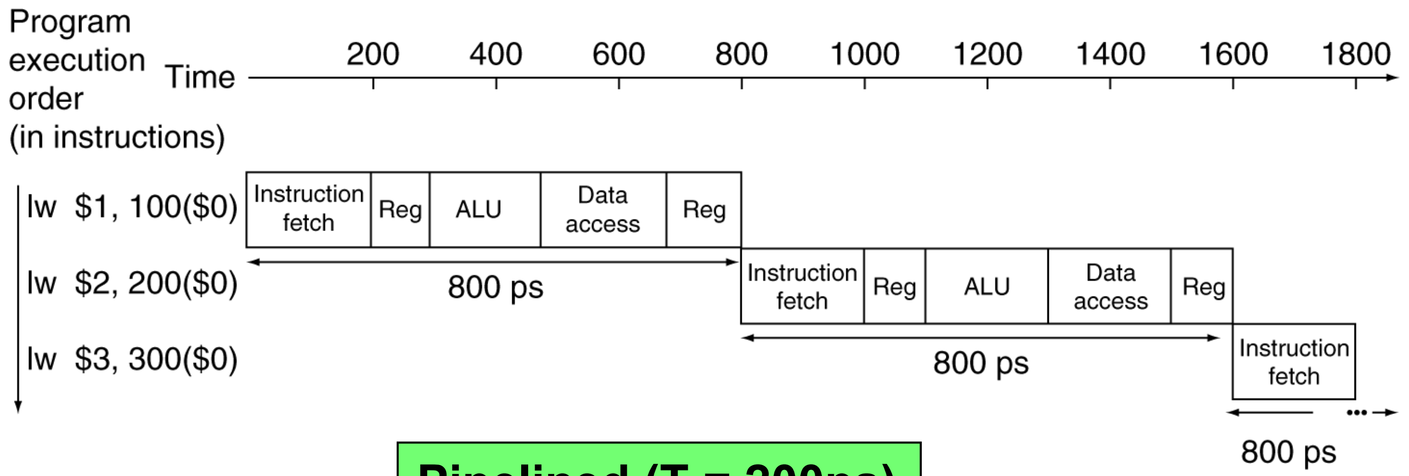
Pipeline Performance

- **Assume time for stages is**
  - 100ps for register read or write
  - 200ps for other stages
- **Compare pipelined datapath with single-cycle datapath**

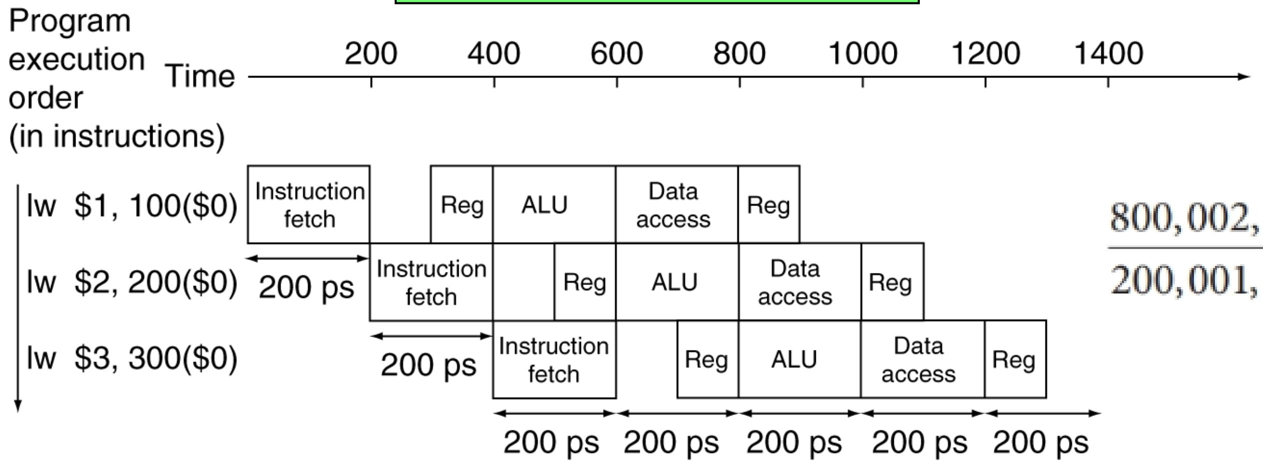
Instruction	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

Pipeline Performance

Single-cycle (T_c= 800ps)



Pipelined (T_c= 200ps)



$$\frac{800,002,400\,ps}{200,001,400\,ps} \simeq \frac{800ps}{200ps} \simeq 4.00$$

Pipeline Speedup

- **If all stages are balanced**
  - i.e., all take the same time

$$\text{Time between instructions\_pipelined} = \frac{\text{Time between instructions\_nonpipelined}}{\text{Number of stages}}$$

- **If not balanced, speedup is less**
- **Speedup due to increased throughput**
  - Latency (time for each instruction) does not decrease

Pipeline Speedup

- **Pipelining improves performance by increasing instruction throughput, as opposed to decreasing the execution time of an individual instruction**

- ***instruction throughput* is the important metric because real programs execute billions of instructions.**

Pipelining and ISA Design

- **MIPS ISA designed for pipelining**
  - All instructions are 32-bits
    - » Easier to fetch and decode in one cycle
    - » c.f. x86: 1- to 17-byte instructions
  - Few and regular instruction formats
    - » Can decode and read registers in one step
  - Load/store addressing
    - » Can calculate address in 3rd stage, access memory in 4th stage
  - Alignment of memory operands
    - » Memory access takes only one cycle

Pipeline Hazards

Hazards

- **Situations that prevent starting the next instruction in the next cycle**
- **Structure hazards**
  – A required resource is busy
- **Data hazard**
  – Need to wait for previous instruction to complete its data read/write
- **Control hazard**
  – Deciding on control action depends on previous instruction

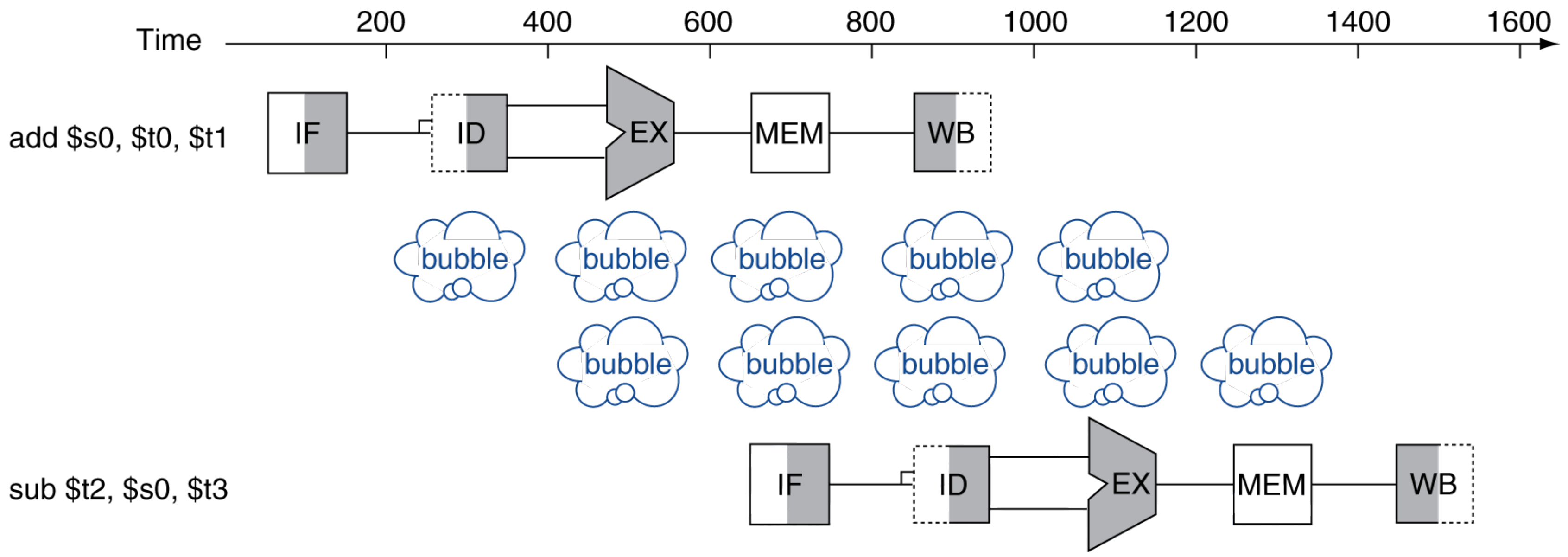
Structure Hazards

- **Conflict for use of a resource**
- **In MIPS pipeline with a single memory**
  - Load/store requires data access
  - Instruction fetch would have to *stall* for that cycle
    - » Would cause a pipeline "bubble"
- **Hence, pipelined datapath require separate instruction/data memories**
  - Or separate instruction/data caches

Data Hazards

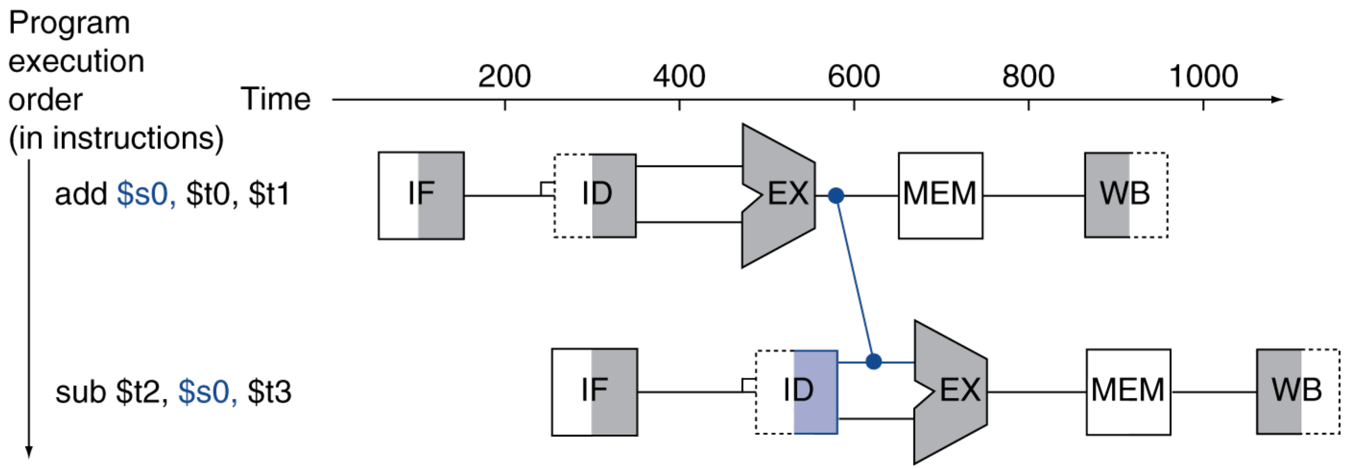
- An instruction depends on completion of data access by a previous instruction

  -
    ```
    add     $s0, $t0, $t1
    sub     $t2, $s0, $t3
    ```



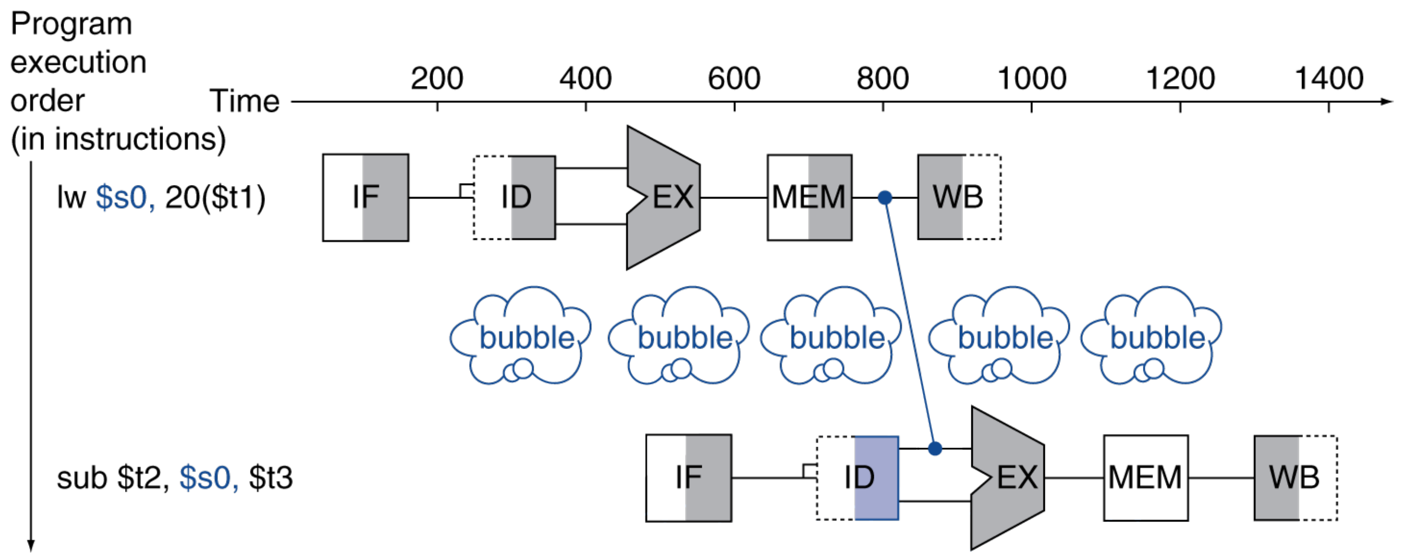
Forwarding

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



Load-Use Data Hazard

- **Can't always avoid stalls by forwarding**
  - If value not computed when needed
  - Can't forward backward in time!



Program execution order (in instructions)
Time
200
400
600
800
1000
1200
1400
lw $s0, 20($t1)
IF
ID
EX
MEM
WB
bubble
bubble
bubble
bubble
bubble
sub $t2, $s0, $t3
IF
ID
EX
MEM
WB