CSCI 210: Computer Architecture
Lecture 27: Control Path

Stephen Checkoway

Slides from Cynthia Taylor

CS History: The IBM 7030 Stretch

- First pipelined computer
  - Three stages: Fetch-Decode-Execute
- Fastest computer in the world from 1961 until 1964
- Much slower than IBM anticipated
  - Dropped price from \$13.5 million to \$7.5 million
  - PC World named it one of the biggest IT project management failures in history
- Many ideas from the Stretch, including pipelining, were used in the very successful IBM System/360

Instruction Critical Paths

- What is the clock cycle time assuming negligible delays for muxes, control unit, sign extend, PC access, shift left 2, wires, setup and hold times except:
  - Instruction and Data Memory (200 ps)
  - ALU and adders (200 ps)
  - Register File access (reads or writes) (100 ps)

Instr.	I Mem	Reg Rd	ALU Op	D Mem	Reg Wr	Total
R-type						
load						
store						
beq						
jump						

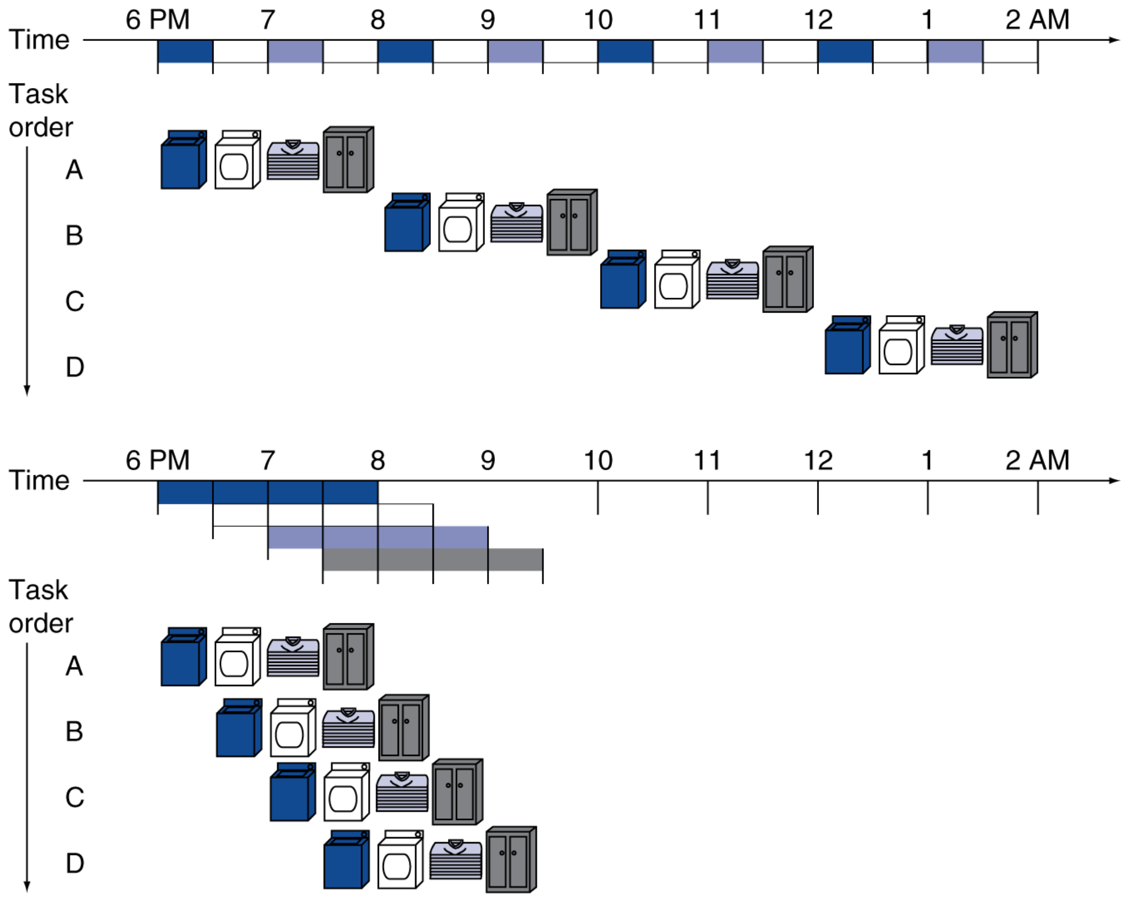
Times are just examples and not representative of real-world latencies

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 Analogy

- Pipelined laundry: overlapping execution
  - Parallelism improves performance

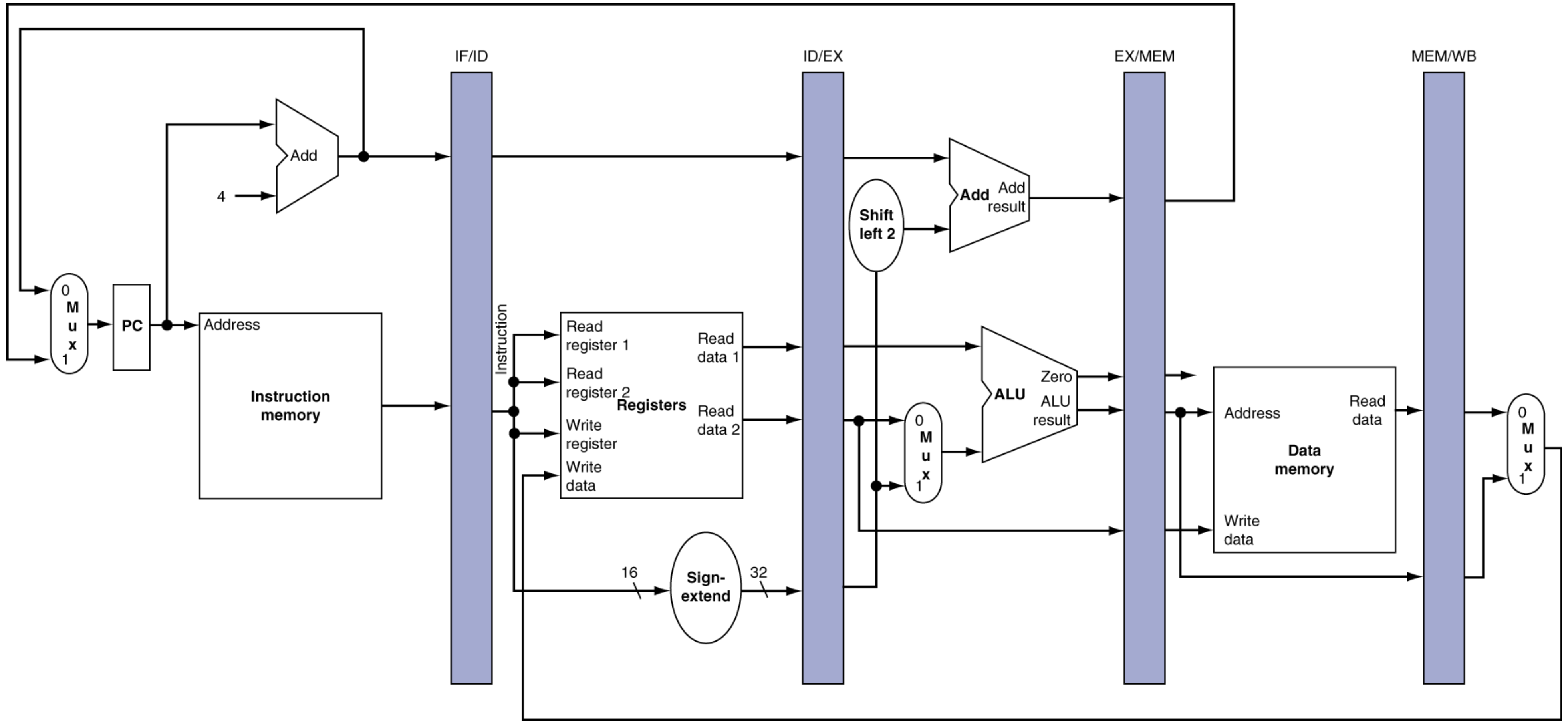


MIPS Pipeline

- 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

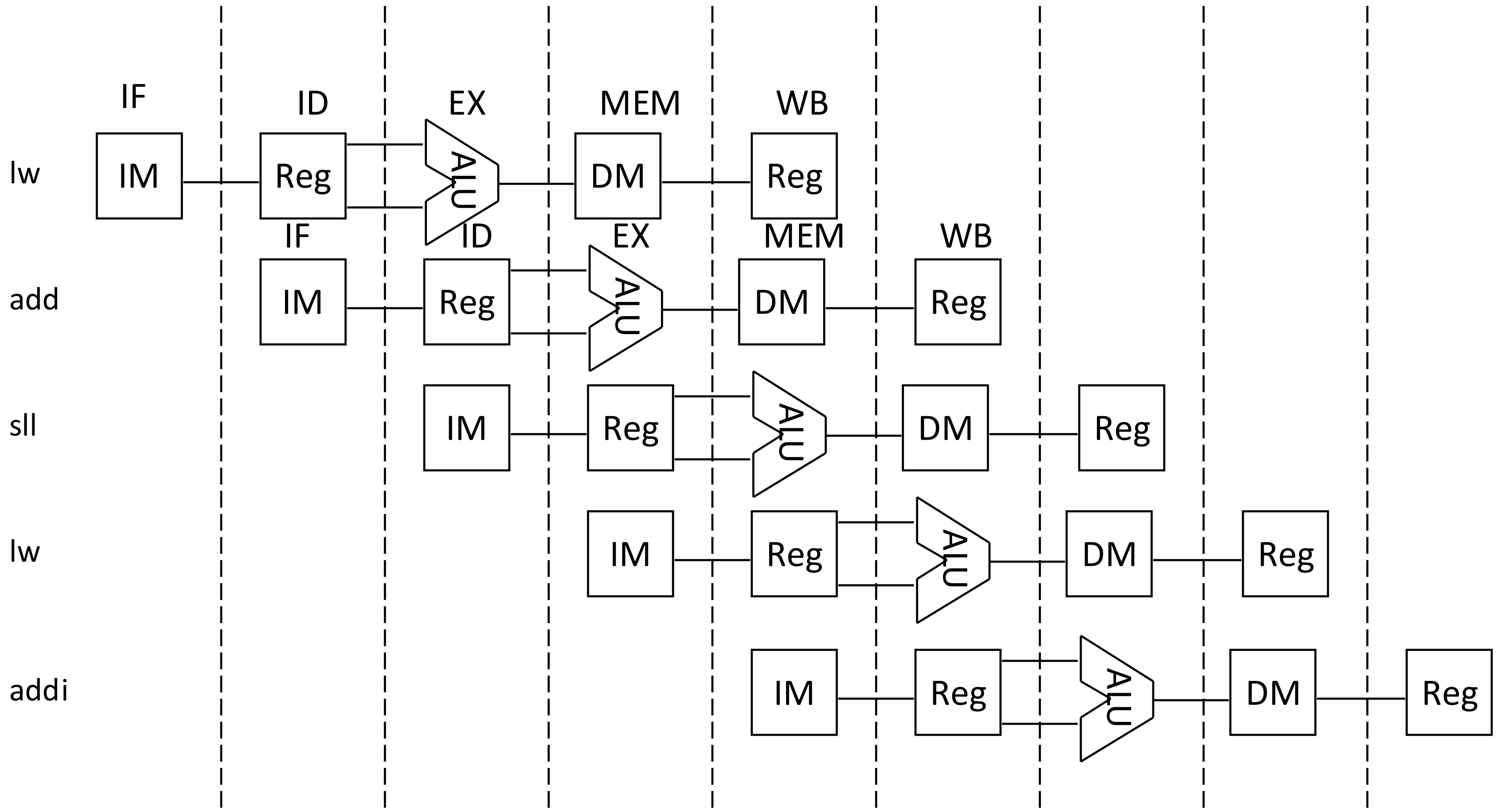
- Move from one instruction per clock cycle, to one stage per clock cycle (with shorter cycle period!)

Pipelined Datapath

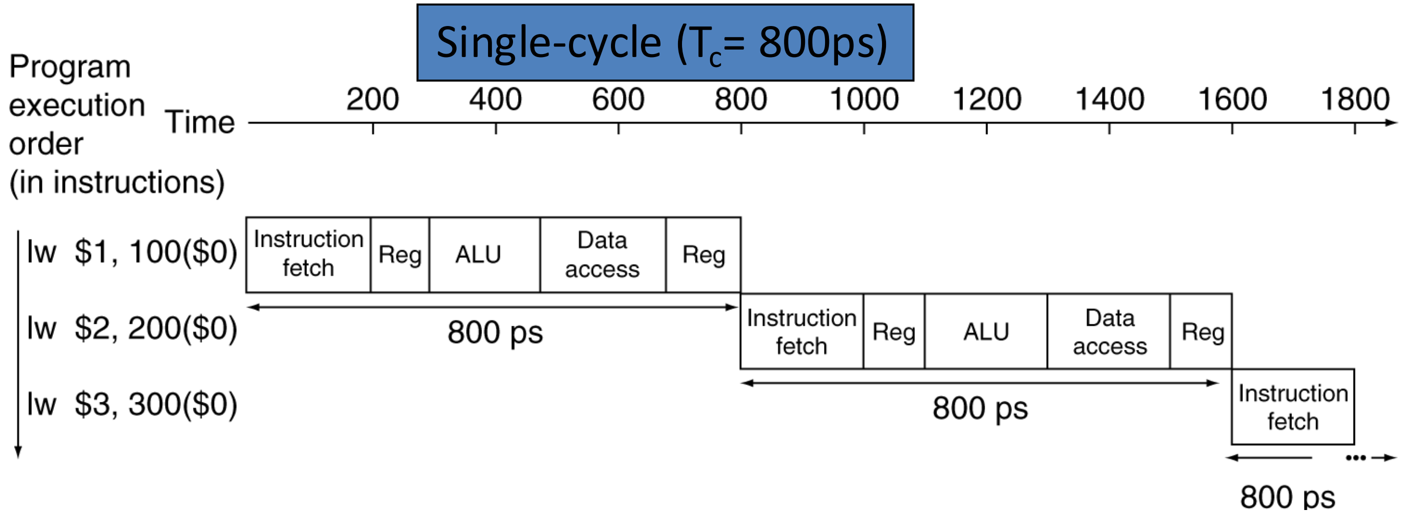


IF: Instruction Fetch   ID: Instruction Decode   EX: Execute   MEM: Memory   WB: Writeback

Execution in a Pipelined Datapath



Pipeline Performance



If we pipeline by running different stages at the same time (i.e., running instruction fetch for the next instruction during the Reg stage of the first instruction), running two instructions will take us

A. 900 ps

B. 1000 ps

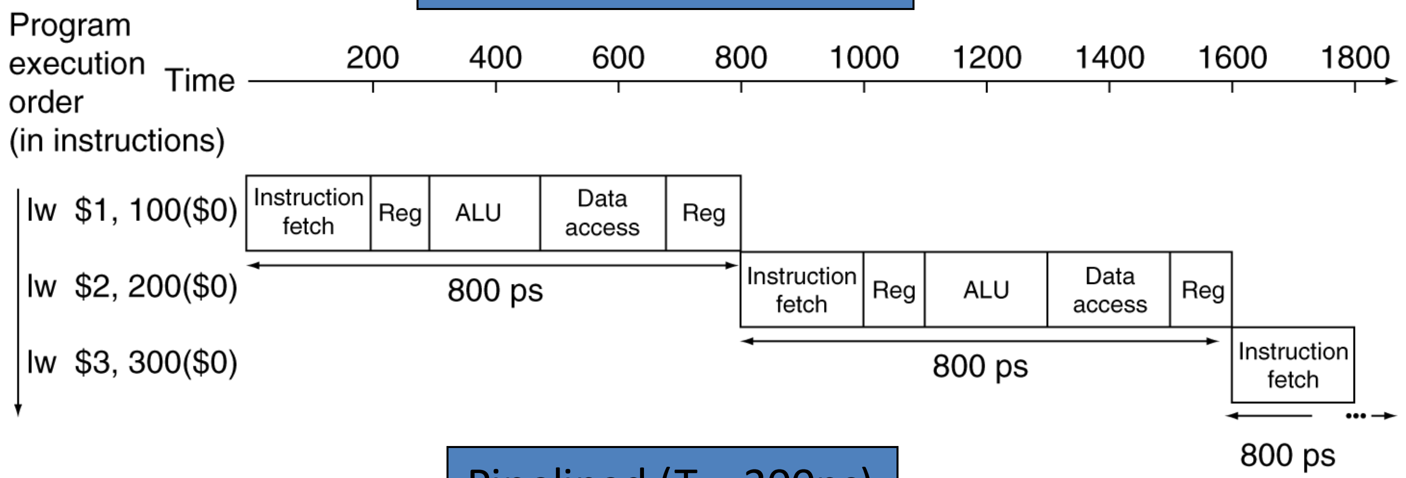
C. 1200 ps

D. 1600 ps

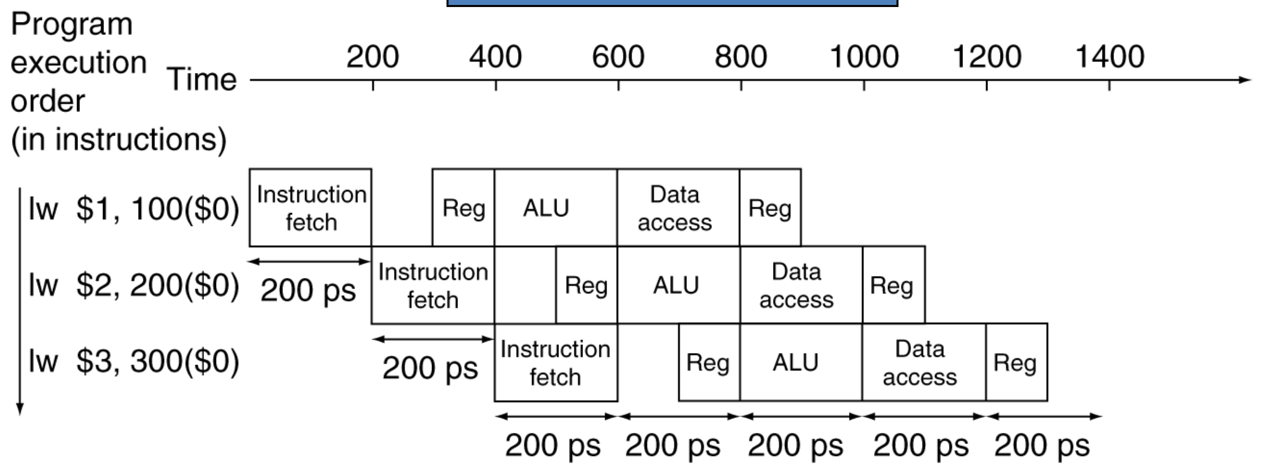
- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages
- HINT: How long will the clock cycle be in the pipelined version?

Pipeline Performance

Single-cycle (T_c= 800ps)



Pipelined (T_c= 200ps)



In our pipelined datapath, latency (ms per instruction) __________, but throughput (instructions per second) __________

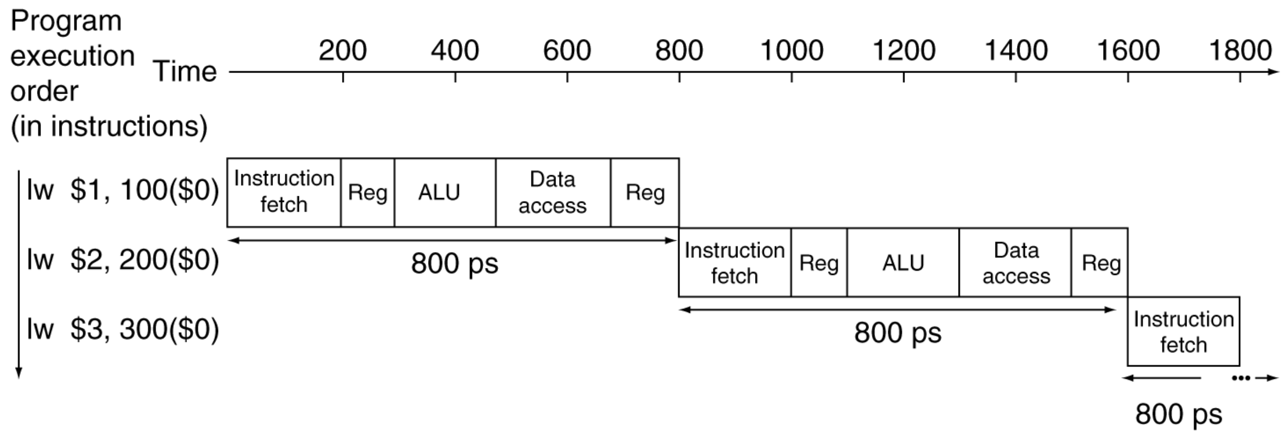
A. Improves, gets worse

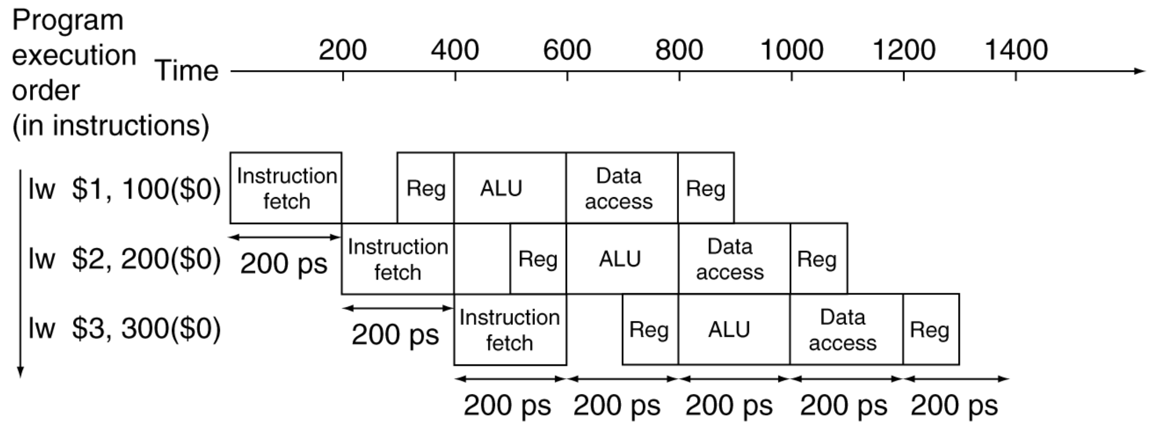
B. Improves, stays the same

C. Gets worse, improves

D. Stays the same, improves

E. None of the above





Maximum Pipeline Speedup

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

$$\text{Time between instructions}_{\text{pipelined}} = \frac{\text{Time between instructions}_{\text{nonpipelined}}}{\text{Number of stages}}$$

- Speedup = time$_{\text{pipelined}}$ / time$_{\text{nonpipelined}}$ = number of stages
- If not balanced, speedup is less
- Speedup due to increased throughput
  - Latency (time for each instruction) does not decrease

Pipelining and ISA Design

- MIPS ISA designed for pipelining
  - MIPS stands for Microprocessor without Interlocked Pipelined Stages
    - This was aspirational; the pipeline stages ended up being interlocked
  - All instructions are 32-bits
    - Easier to fetch in one cycle
    - c.f. x86: 1- to 15-byte instructions
  - Few and regular instruction formats
    - Can decode and read registers in one cycle
  - Load/store addressing
    - Can calculate address in 3rd stage, access memory in 4th stage

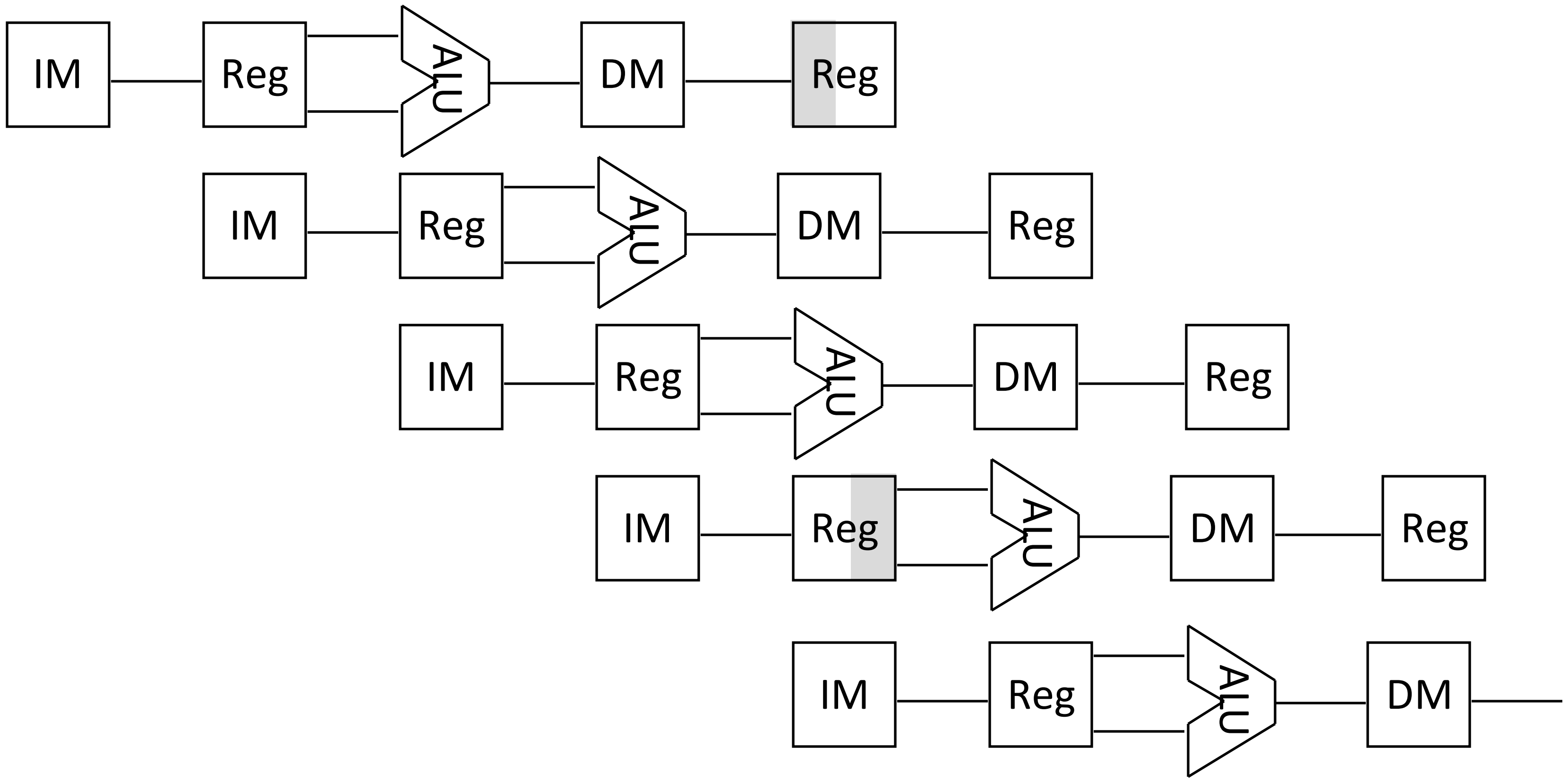
```
sub $2, $1, $3

and $12, $2, $5

or  $13, $6, $2

add $14, $2, $2

sw  $15, 100($2)
```



What just happened here which is problematic (BEST ANSWER)?

A. The register file is trying to read and write the same register
B. The ALU and data memory are both active in the same cycle
C. A value is used before it is produced
D. Both A and B
E. Both A and C

Hazards

Situations that prevent starting the next instruction in the next cycle

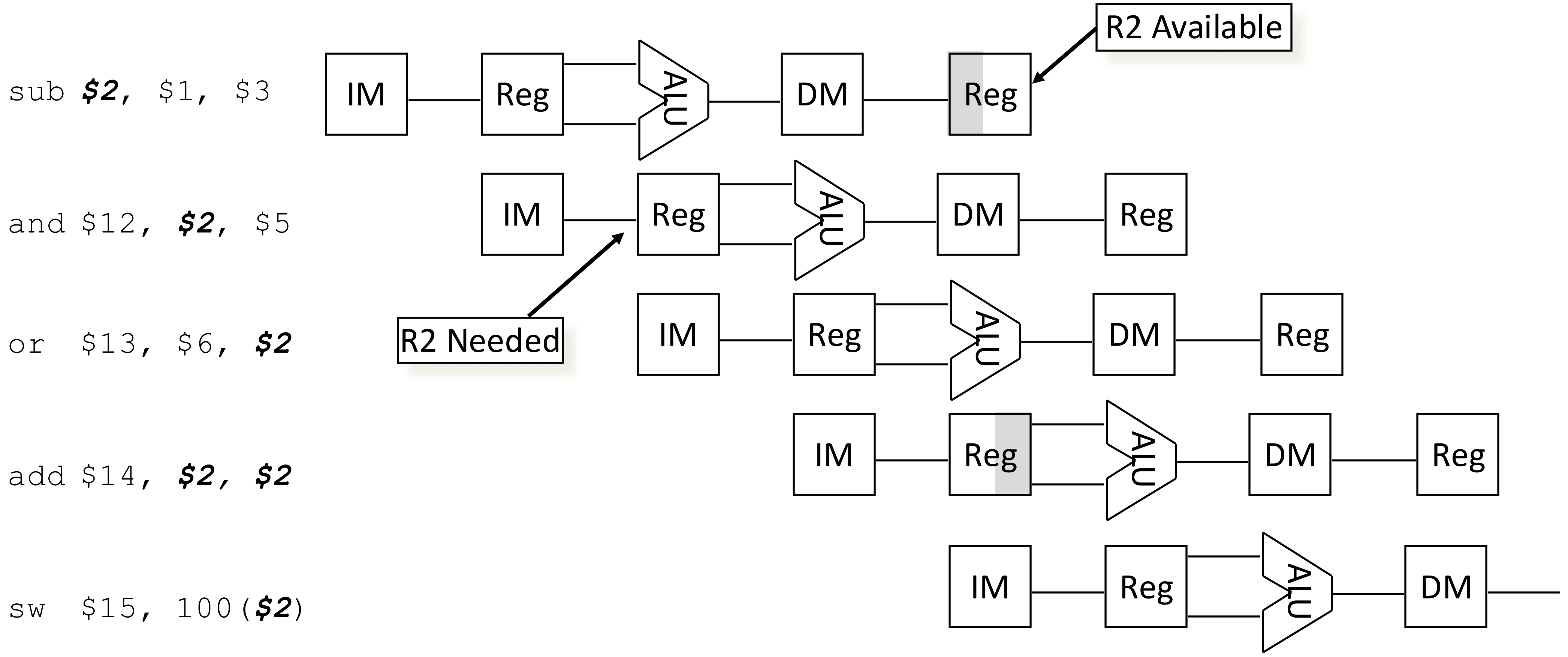
- Structure hazards
  - A required resource is busy
- Data hazard
  - Need to wait for previous instruction to complete its data read/write
- Control hazard
  - Deciding on control action depends on previous instruction

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 datapaths require separate instruction/data memories (or “caches” which we’ll talk about later in great detail)

Data Hazards

- When a result is needed in the pipeline before it is available, a "data hazard" occurs.



We could solve data hazards by

A. Reordering instructions

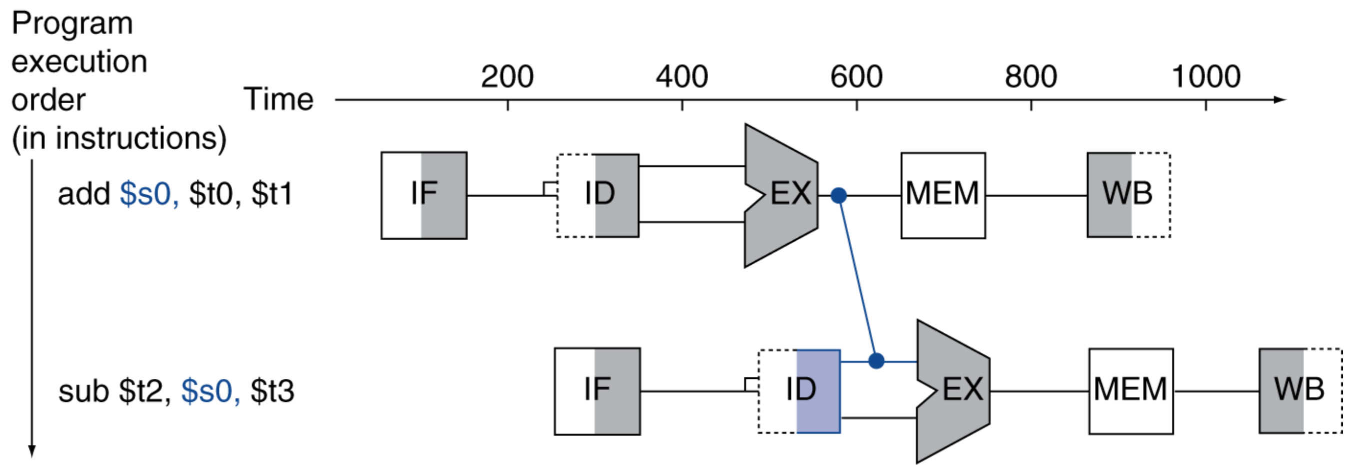
B. Not running the second instruction until the data is ready

C. Sending the calculated value straight from the ALU to the next instruction, skipping the registers

D. More than one of the above

Forwarding (a.k.a. Bypassing)

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



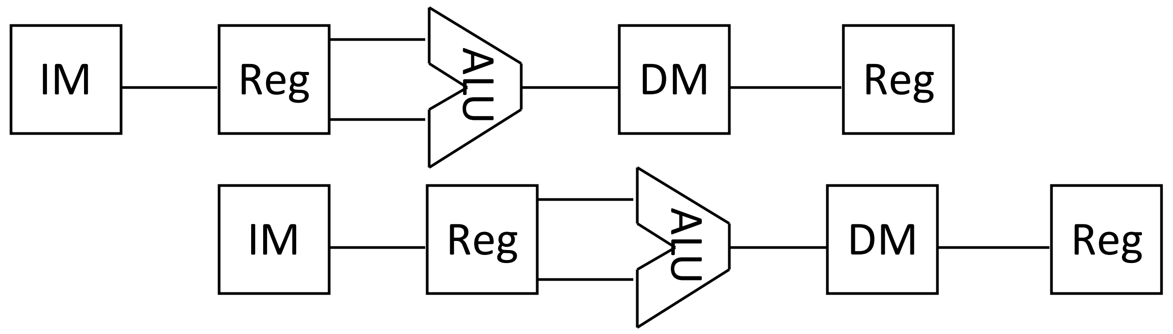
Would forwarding work if our instructions were

```
lw  $s0, 20($1)
sub $t2, $s0, $t3
```

A. Yes

B. No

C. Depends on the value loaded



Reading

- Next lecture: Pipeline
  - Section 5.7