

PIPELINING

CSE 240A

Dean Tullsen

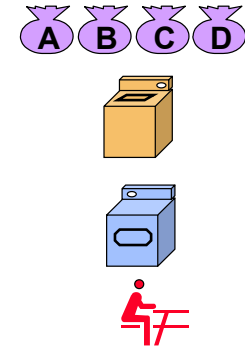
Pipelining: Natural Phenomenon

Laundry Example:

Ann, Brian, Cathy, Dave
each have one load of clothes
to wash, dry, and fold
Washer takes 30 minutes

Dryer takes 40 minutes

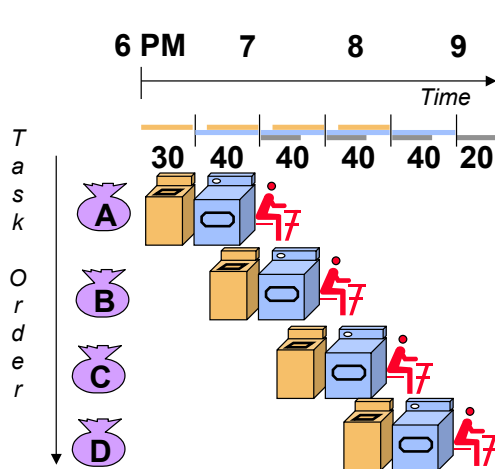
“Folder” takes 20 minutes



CSE 240A

Dean Tullsen

Pipelining Lessons



CSE 240A

Dean Tullsen

- 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
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to “fill” pipeline and time to “drain” it reduces speedup

Pipelining

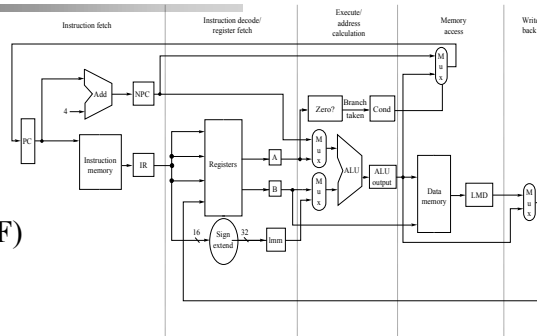
- Requires separable jobs/stages
- Requires separate resources
- Achieves parallelism without replication
- Improves throughput
- Often increases single-task (e.g., instruction, laundry load) latency
- Pipeline efficiency (keeping the pipeline full) critical to performance

CSE 240A

Dean Tullsen

5 Steps of a DLX/MIPS Instruction

- **Instruction Fetch (IF)**
 - $IR \leftarrow M[PC]$
 - $NPC \leftarrow PC + 4$
- **Instruction Decode/register fetch (ID)**
 - $A \leftarrow \text{Reg}[IR6..10]$
 - $B \leftarrow \text{Reg}[IR11..15]$
 - $\text{Imm} \leftarrow \text{Sign_extend}(IR16..31)$

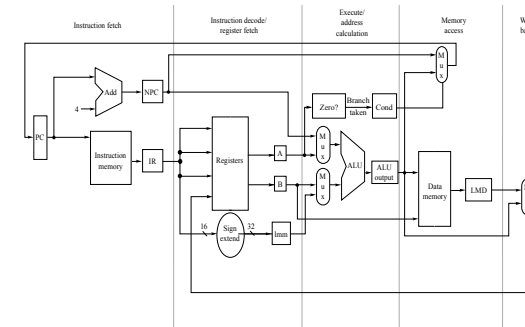


CSE 240A

Dean Tullsen

5 Steps of a DLX/MIPS Instruction

- **Execute/Effective Address (EX)**
 - $\text{ALUOutput} \leftarrow A + \text{Imm}$ (memory ref)
 - $\text{ALUOutput} \leftarrow A \text{ op } B$ (register-register alu instruction)
 - $\text{ALUOutput} \leftarrow A \text{ op } \text{Imm}$ (register-immediate alu instruction)
 - $\text{ALUOutput} \leftarrow \text{NPC} + \text{Imm}; \text{Cond} \leftarrow (A \text{ op } 0)$ (Branch)

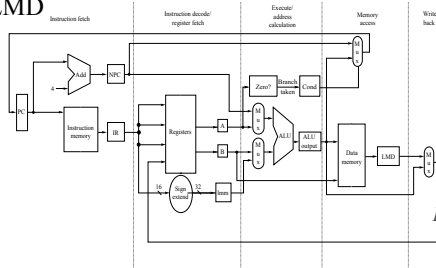


CSE 240A

Dean Tullsen

5 Steps of a DLX/MIPS Instruction

- **Memory access/branch completion (MEM)**
 - $\text{LMD} \leftarrow M[\text{ALUOutput}] \text{ or } M[\text{ALUOutput}] \leftarrow B$ (load or store)
 - if (cond) $PC \leftarrow \text{ALUOutput}$ (branch)
 - else $PC \leftarrow \text{NPC}$
- **Write-Back (WB)**
 - $\text{Reg}[IR16..20] \leftarrow \text{ALUOutput}$ (reg-reg alu instruction)
 - $\text{Reg}[IR11..15] \leftarrow \text{ALUOutput}$ (reg-imm alu instruction)
 - $\text{Reg}[IR11..15] \leftarrow \text{LMD}$

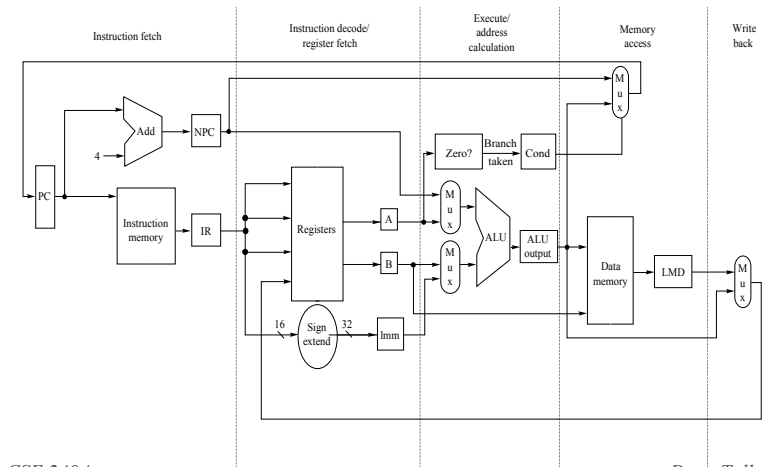


CSE 240A

Dean Tullsen

ADDI R7, R2, #35

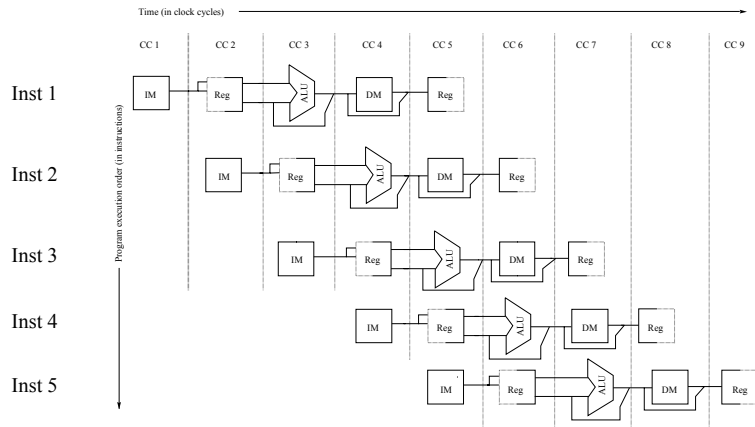
(Rd) (Rs1)



CSE 240A

Dean Tullsen

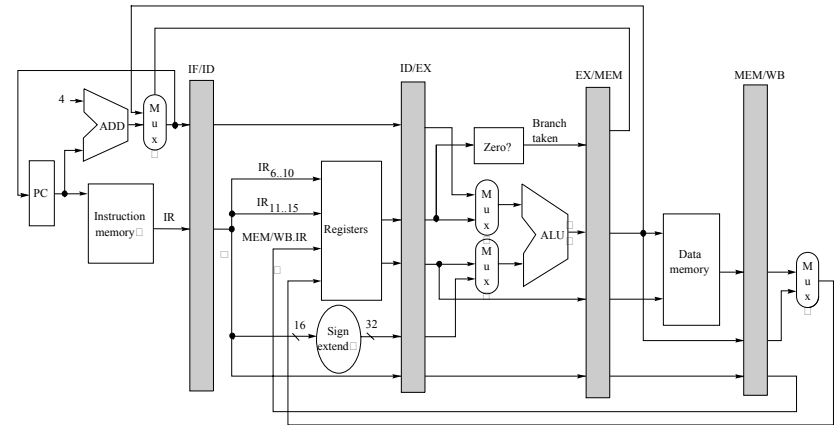
Visualizing Pipelining



CSE 240A

Dean Tullsen

The Pipelined DLX Datapath



CSE 240A

Dean Tullsen

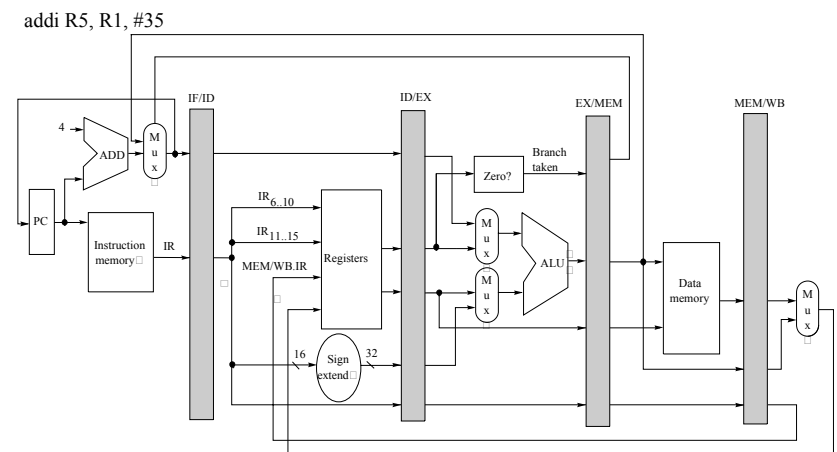
The Pipeline in Motion

- addi R5, R1, #35
- add R6, R2, R1
- lw R8, 10000(R3)

CSE 240A

Dean Tullsen

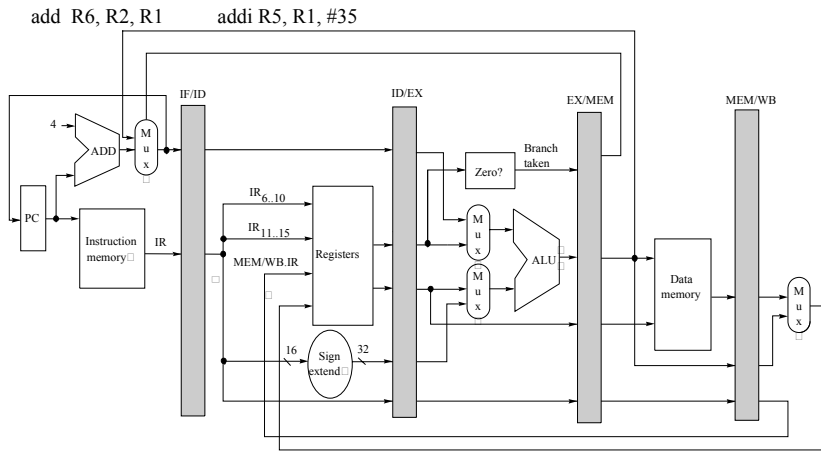
The Pipeline In Motion



CSE 240A

Dean Tullsen

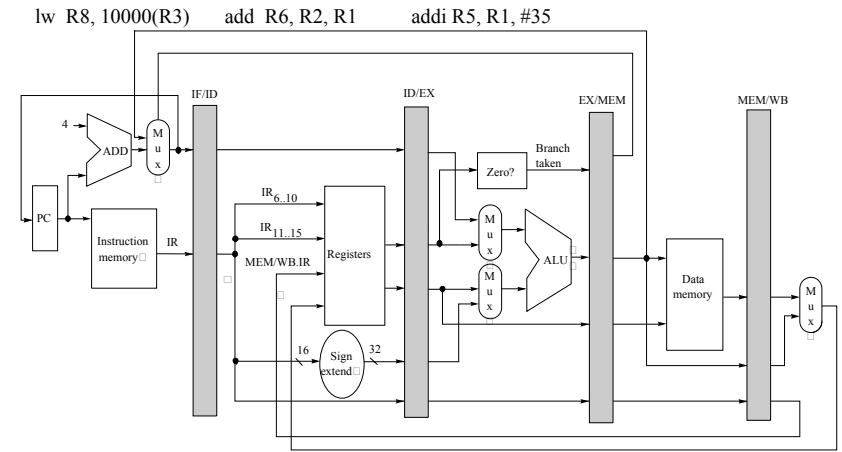
The Pipeline In Motion



CSE 240A

Dean Tullsen

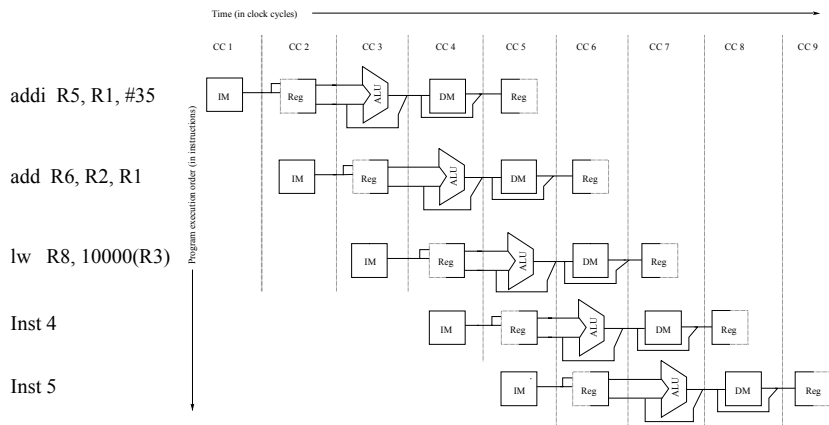
The Pipeline In Motion



CSE 240A

Dean Tullsen

The Pipeline In Motion



CSE 240A

Dean Tullsen

Pipeline Performace

- $ET = IC * CPI * CT$
 - single-cycle processor
 - multiple-cycle processor
 - pipelined processor
- complexity has a cost
 - e.g., latch overhead
 - uneven stage latencies
- Can't always keep the pipeline full
 - why not?

CSE 240A

Dean Tullsen

When Things Go Wrong -- Pipeline Hazards

- Limits to pipelining: **Hazards** prevent next instruction from executing during its designated clock cycle
 - *Structural hazards*: HW cannot support this combination of instructions
 - *Data hazards*: Instruction depends on result of prior instruction still in the pipeline
 - *Control hazards*: Pipelining of branches & other instructions that change the PC
- Common solution is to stall the pipeline until the hazard is resolved, inserting one or more “bubbles” in the pipeline

Key Points

- Pipeline improves throughput rather than latency
- Pipelining gets parallelism without replication
- $ET = IC * CPI * CT$
- Keeping the pipeline full is no easy task
 - structural hazards
 - data hazards
 - control hazards