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

Pipelining Lessons

• 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
5 Steps of a DLX/MIPS Instruction

- **Instruction Fetch (IF)**
  - IR <- M[PC]
  - NPC <- PC + 4

- **Instruction Decode/register fetch (ID)**
  - A <- Reg[IR11..15]
  - B <- Reg[IR6..10]
  - Imm <- Sign_extend(IR16..31)

- **Execute/Effecitve Address (EX)**
  - ALUOutput <- A + Imm (memory ref)
  - ALUOutput <- A op B (register-register alu instruction)
  - ALUOutput <- A op Imm (register-immediate alu instruction)
  - ALUOutput <- NPC + Imm; Cond <- (A op 0) (Branch)

- **Memory access/branch completion (MEM)**
  - LMD <- M[ALUOutput] or M[ALUOutput] <- B (load or store)
  - if (cond) PC <- ALUOutput (branch)
    else PC <- NPC

- **Write-Back (WB)**
  - Reg[IR11..15] <- ALUOutput (reg-reg alu instruction)
  - Reg[IR11..15] <- LMD
  - Reg[IR11..15] <- LMD
The Pipeline in Motion

- `addi R5, R1, #35`
- `add R6, R2, R1`
- `lw R8, 10000(R3)`
The Pipeline In Motion

add R6, R2, R1
add R5, R1, #35

The Pipeline In Motion

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

The Pipeline in Motion

Pipeline Performance

- 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?
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