Introduction to Instruction Pipelining

By

Mr. Priyajit Sen
Assistant Professor in Computer Science,
DDE, Vidyasagar University

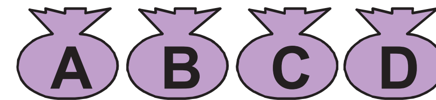
Pipelining

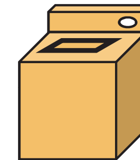
- **Pipelining is an implementation technique in which multiple instructions are overlapped in execution**

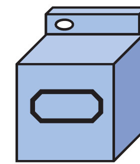
- **Here we will consider RISC architecture**
  - Memory Access by Load/Store
  - All other instructions use registers.

Pipelining is Natural!

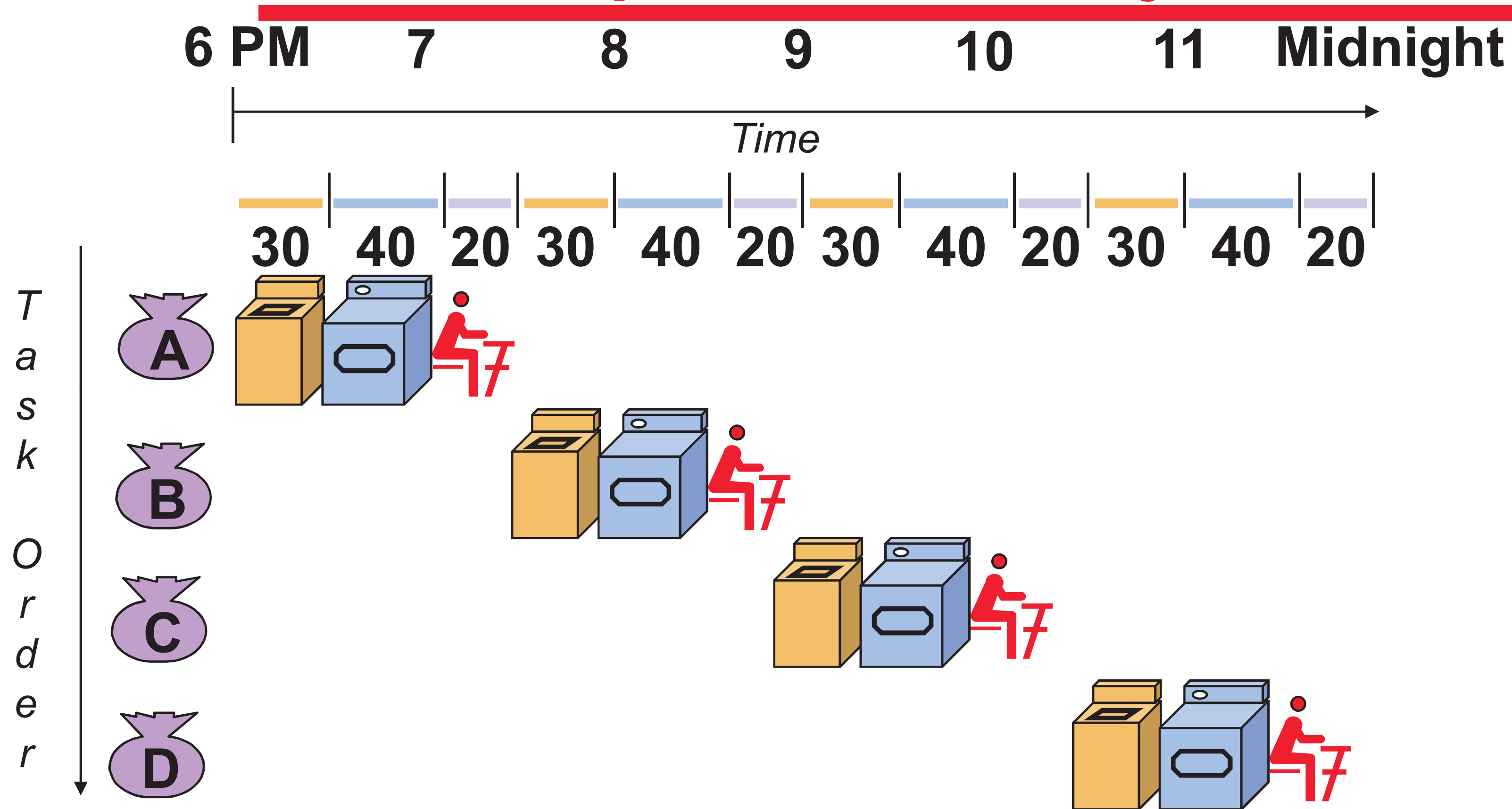
- Laundry Example
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- "Folder" takes 20 minutes





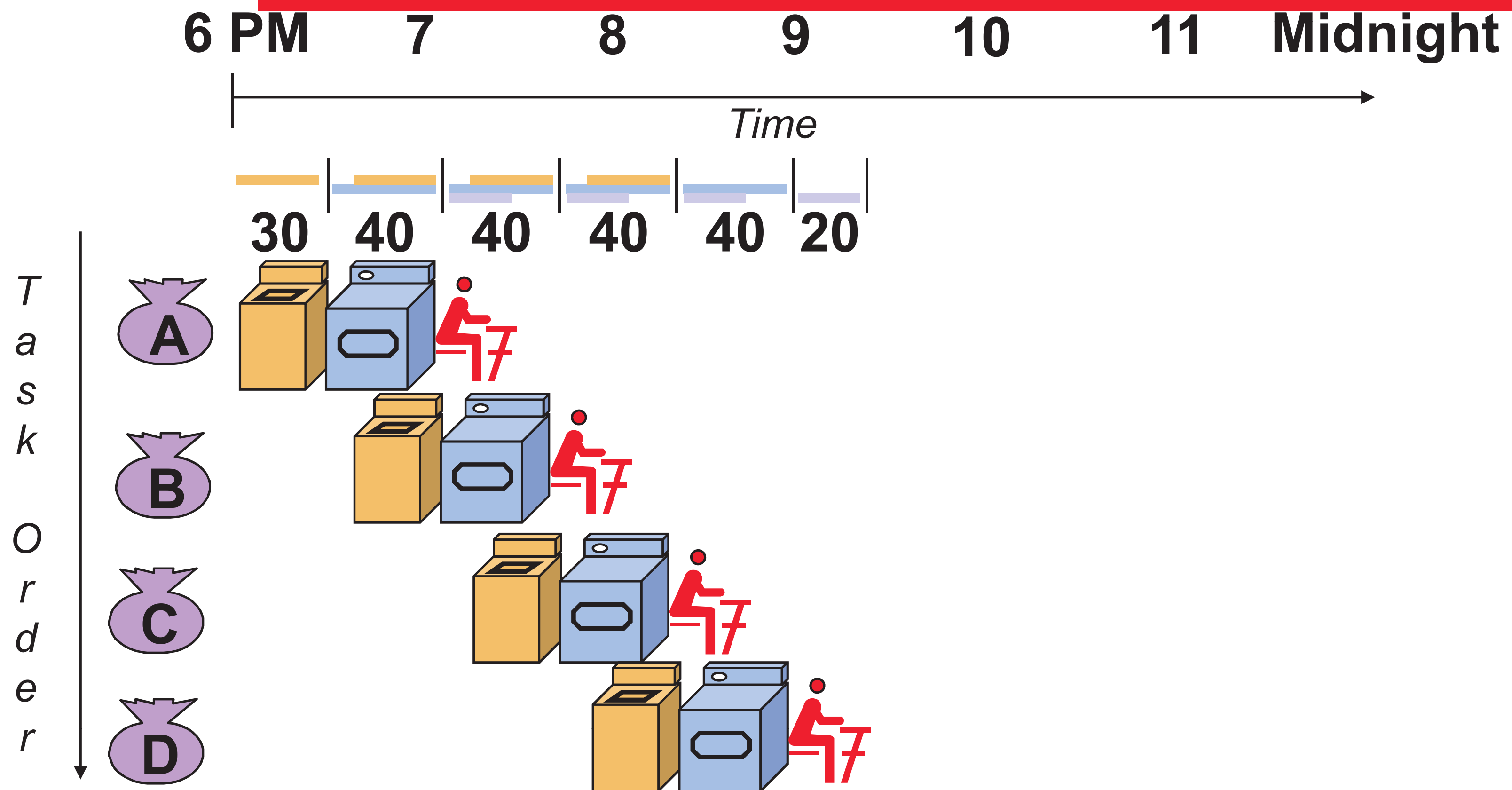


Sequential Laundry



- **Sequential laundry takes 6 hours for 4 loads**
- **If they learned pipelining, how long would laundry take?**

Pipelined Laundry: Start work ASAP



- Pipelined laundry takes 3.5 hours for 4 loads

Linear Pipelining

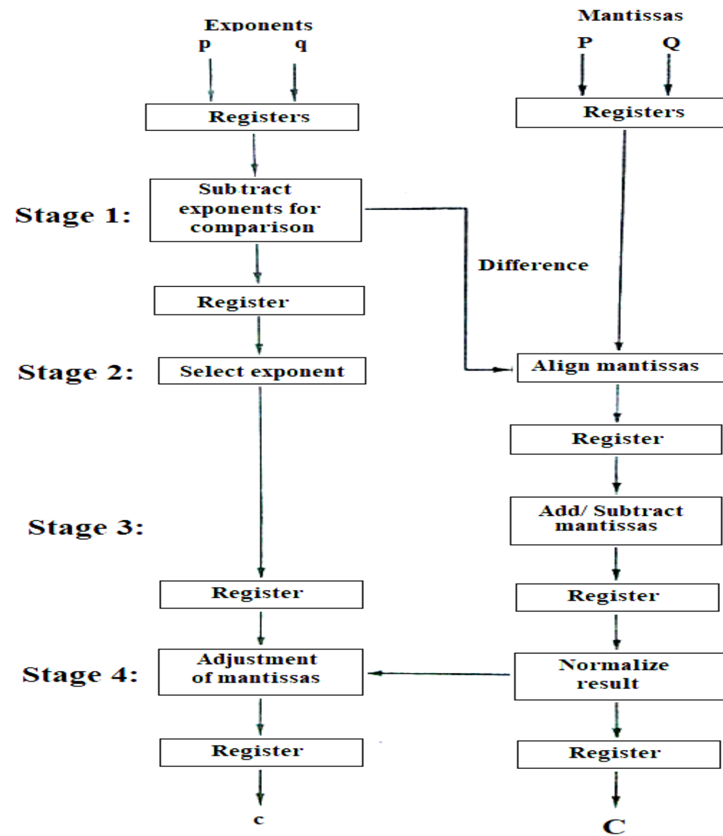
- Types of linear pipeline models
  - **Asynchronous Model**
  - **Synchronous Model**
- Reservation table for Linear Pipelining

Non-Linear Pipelining

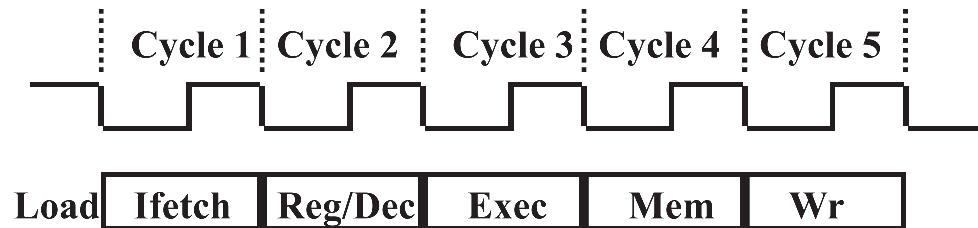
- Reservation table for Non-Linear Pipelining
- Latency Analysis in Non-Linear Pipelining
- Collision Free Scheduling

Arithmetic Pipeline

- Floating point adder-subtractor

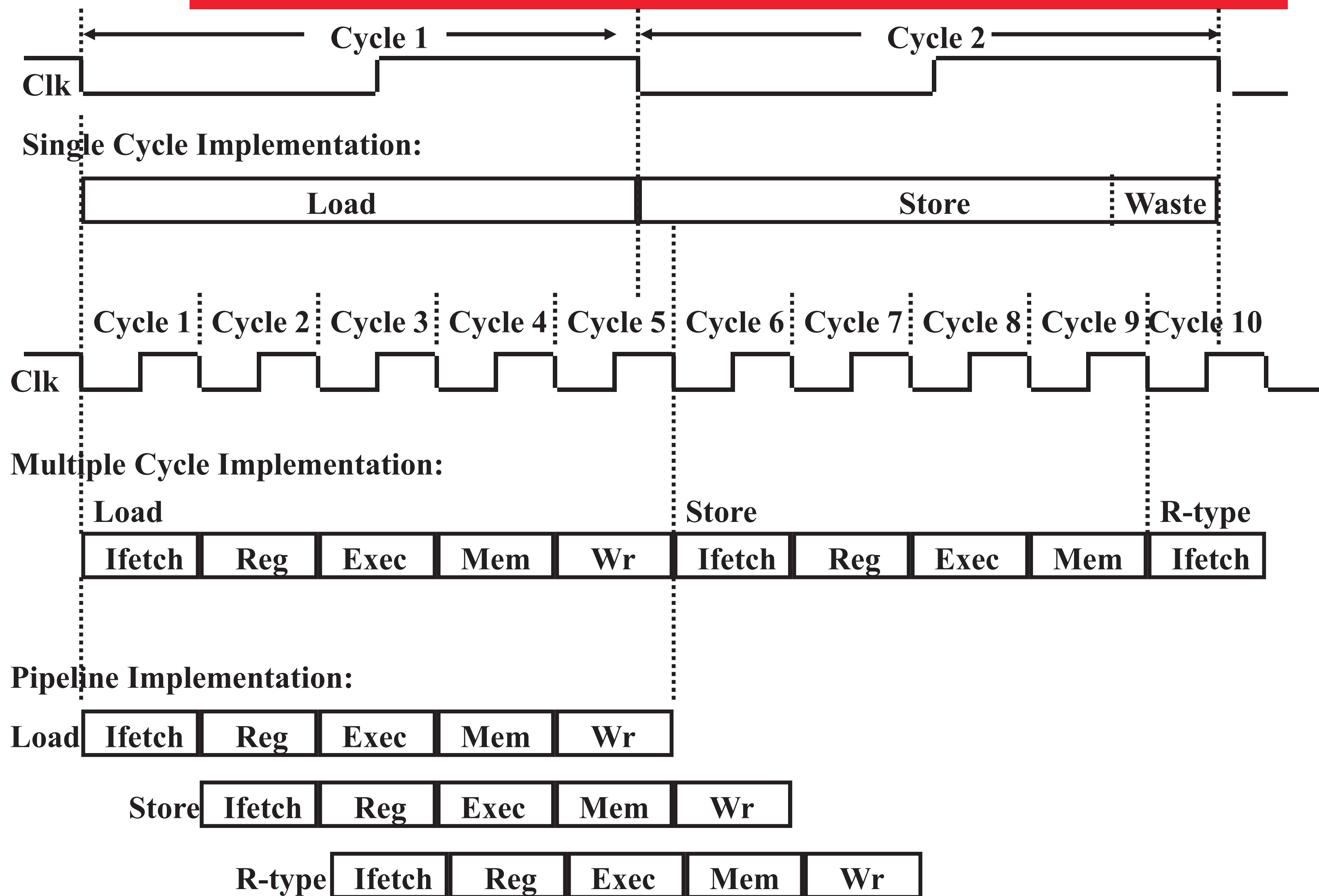


The Five Stages of Load

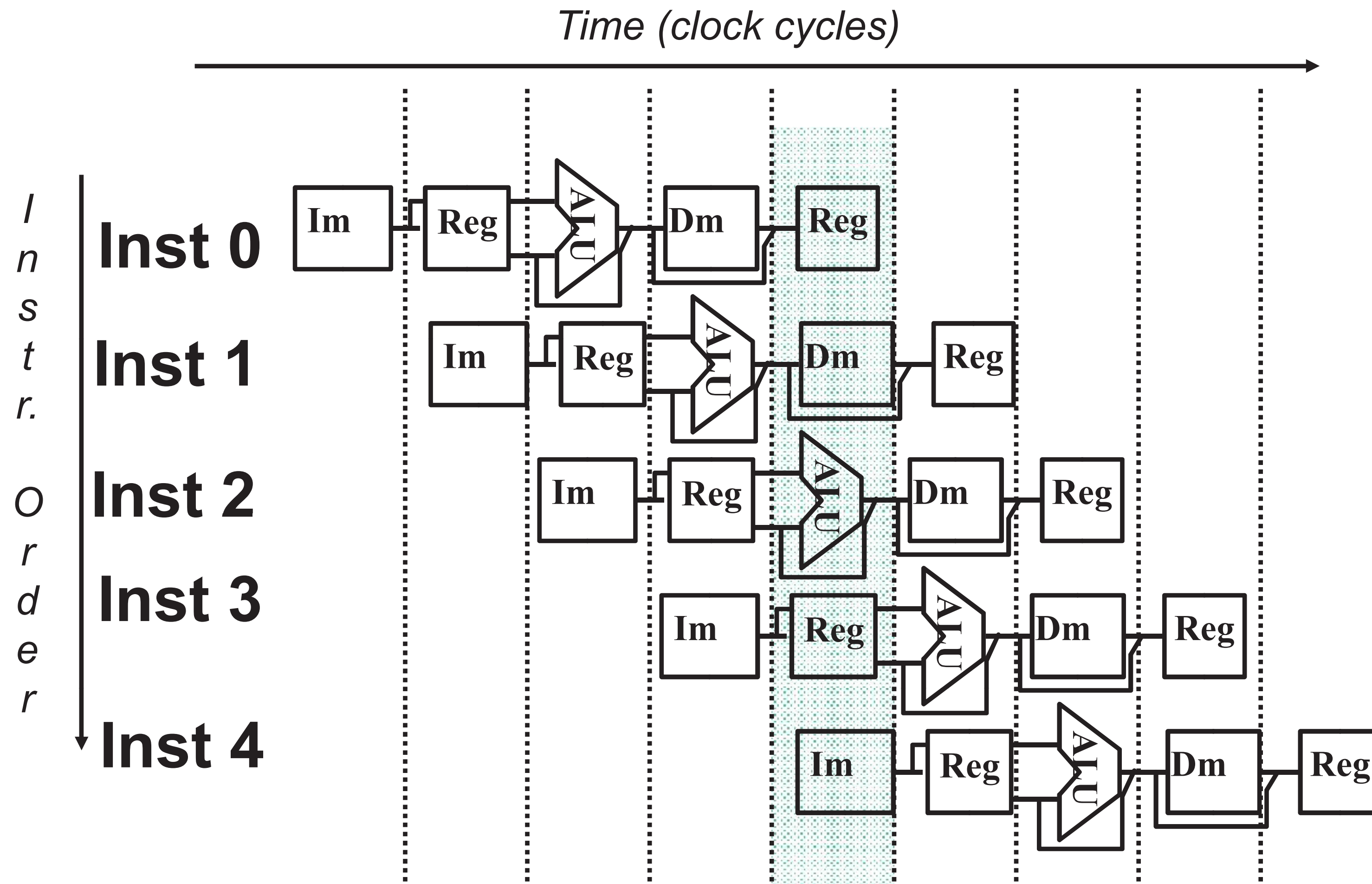


- **Ifetch: Instruction Fetch**
  - Fetch the instruction from the Instruction Memory
- **Reg/Dec: Registers Fetch and Instruction Decode**
- **Exec: Calculate the memory address**
- **Mem: Read the data from the Data Memory**
- **Wr: Write the data back to the register file**

Single Cycle, Multiple Cycle, vs. Pipeline



Why Pipeline? Because the resources are there!



Speedup and Efficiency

k-stage pipeline processes *n* tasks in k + (n-1) clock cycles:

k cycles for the first task and *n-1* cycles for the remaining *n-1* tasks

Total time to process *n* tasks $T_k = [k + (n-1)]\tau$

For the non-pipelined processor $T_1 = n k \tau$

Speedup factor

$$S_k = \frac{T_1}{T_k} = \frac{n k \tau}{[k + (n-1)]\tau} = \frac{n k}{k + (n-1)}$$

If n is very large (n >> k), then $S_k \approx \frac{n k}{n} \approx k$

Efficiency and Throughput

Efficiency *of the k-stages* pipeline:

$$E_k = \frac{S_k}{k} = \frac{n}{k + (n-1)}$$

Pipeline throughput (the number of tasks per unit time):

$$H_k = \frac{n}{[\,k + (n-1)]\,\tau} = \frac{n\,f}{k + (n-1)}$$

Slow Down From Stalls

- Perfect pipelining with no hazards → an instruction completes every cycle (total cycles ~ num instructions) $k + (n-1) \approx n$

→ speedup = increase in clock speed = num pipeline stages

$$S_k \approx \frac{n\,k}{n} \approx k$$

- With hazards and stalls, some cycles (= stall time) go by during which no instruction completes, and then the stalled instruction completes

- Total cycles = number of instructions + stall cycles

- Slowdown because of stalls = 1/ (1 + stall cycles per instr)

Pipeline Hazards

Structural Hazards: Attempt to use the same resource (hardware unit) two different ways at the same time. E.g., two instructions try to read the same memory at the same time

Data Hazards: Attempt to use item before it is ready instruction depends on result of prior instruction still in the pipeline

addr1, r2, r3

sub r4, r2, r1

Control Hazards: Attempt to make a decision before condition is evaluated branch instructions

beq r1,r2,loop

add r3, r4, r5

Types of Data Hazards

- Write After Read (WAR) Hazard
- Read After Write (RAW) Hazard
- Write After Write (WAW) Hazard

Pipeline Performance Improvement

- Super pipelined Design
- Super scalar design
- Very long Instruction word (VLIW) Processor