Pipeline: Introduction

These slides are derived from:
CSCE430/830 Computer Architecture course by Prof. Hong Jiang and Dave Patterson ©UCB

Some figures and tables have been derived from :
Computer System Architecture by M. Morris Mano

Pipelining Outline

Introduction

Defining Pipelining

Pipelining Instructions

Hazards

- Structural hazards
- Data Hazards
- Control Hazards

What is Pipelining?

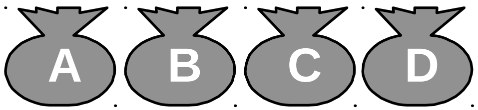
A way of speeding up execution of instructions

Key idea:

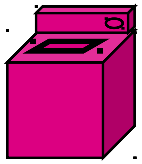
overlap execution of multiple instructions

The Laundry Analogy

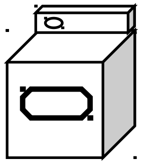
- Ann, Brian, Cathy, Dave
  each have one load of clothes
  to wash, dry, and fold



- Washer takes 30 minutes

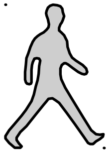


- Dryer takes 30 minutes

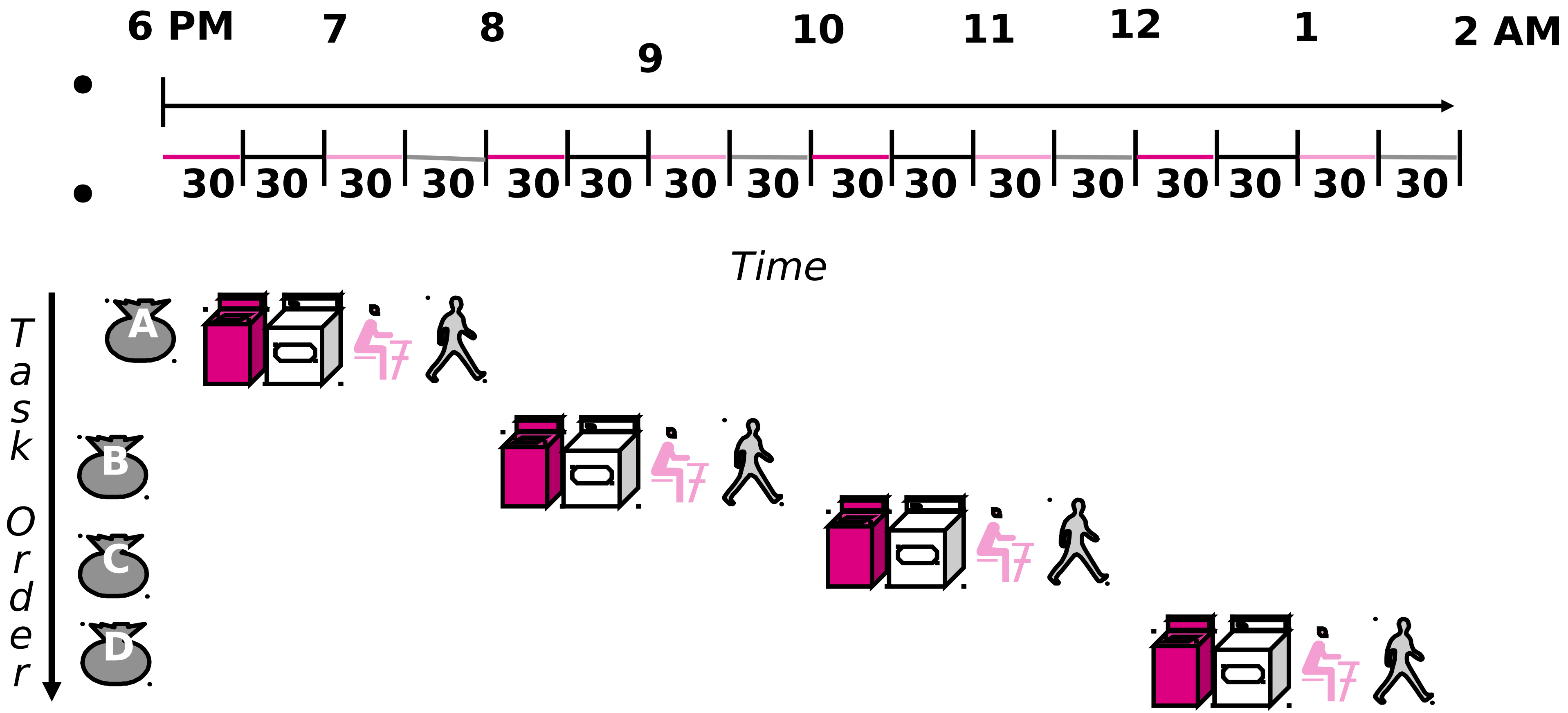


- “Folder” takes 30 minutes

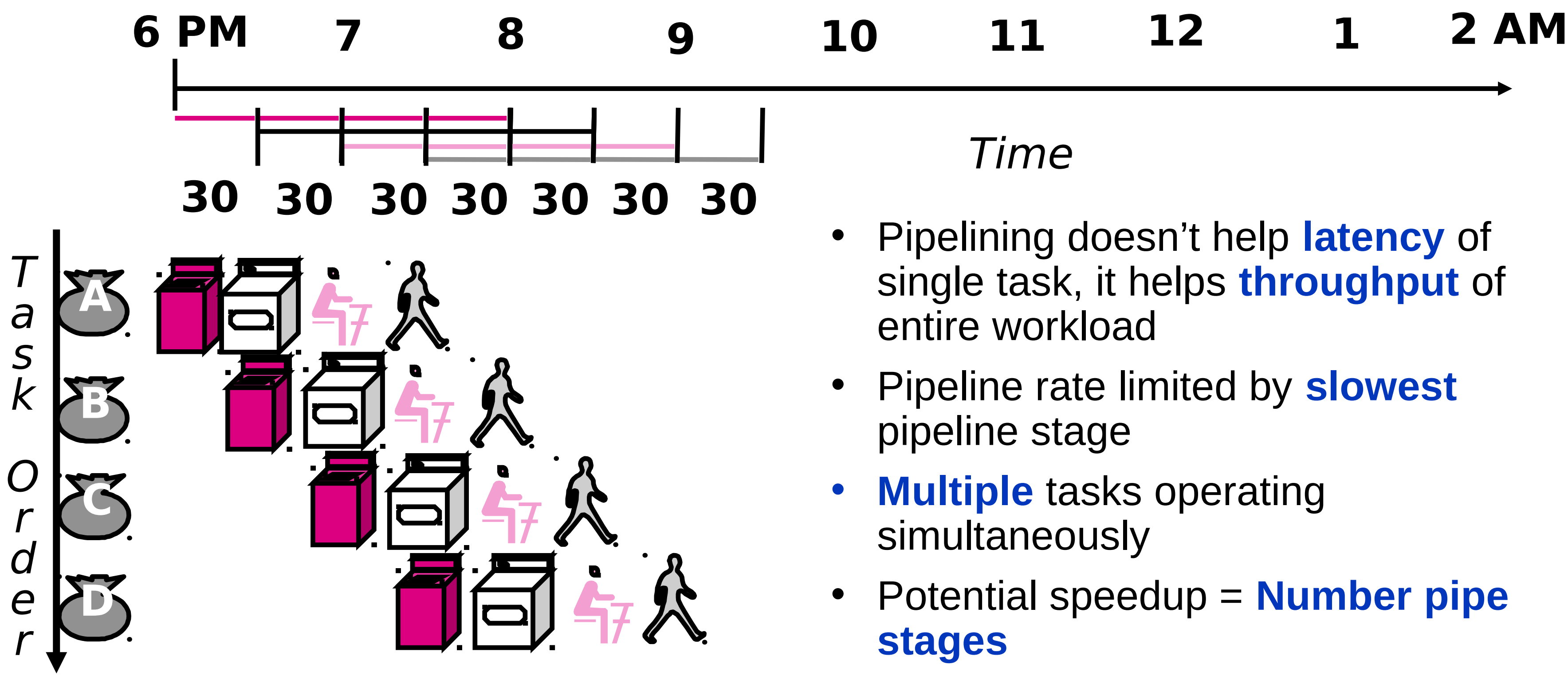
- “Stasher” takes 30 minutes
  to put clothes into drawers



If we do laundry sequentially...



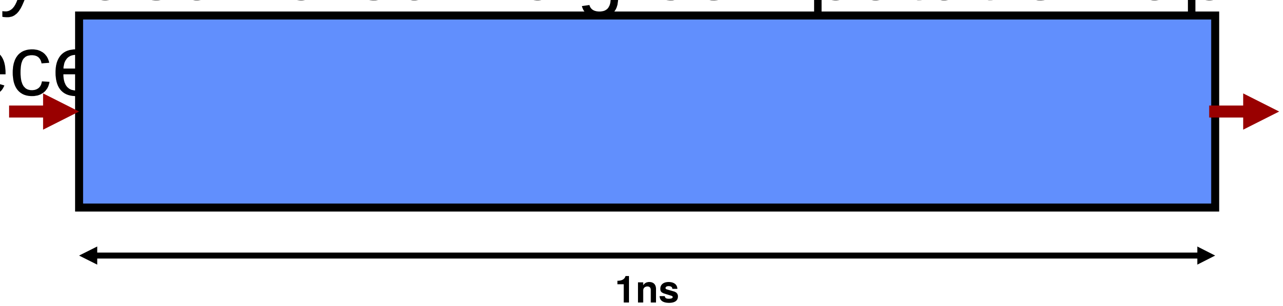
To Pipeline, We Overlap Tasks



- 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

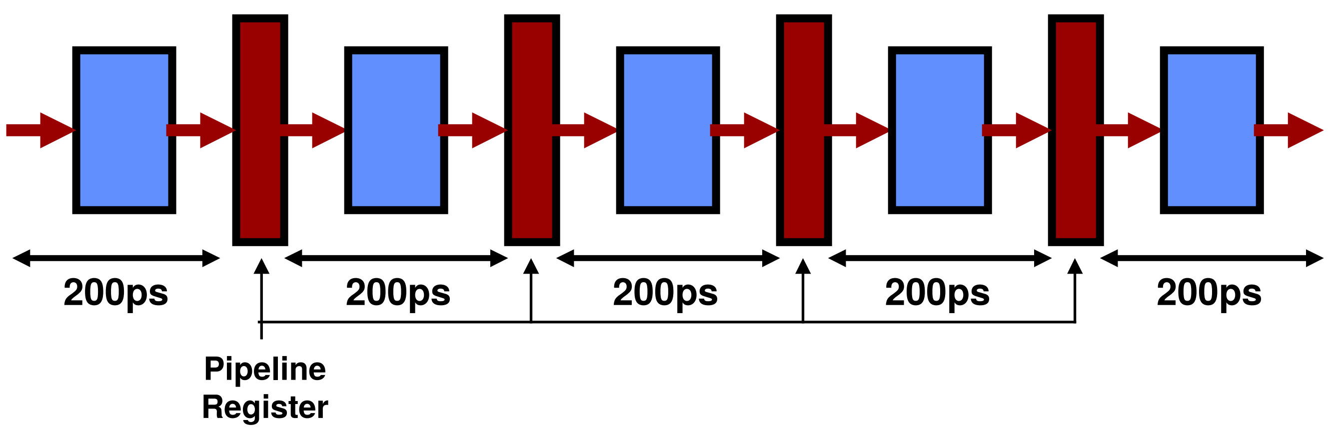
Pipelining a Digital System

-
- Key idea: break big computation up into piece



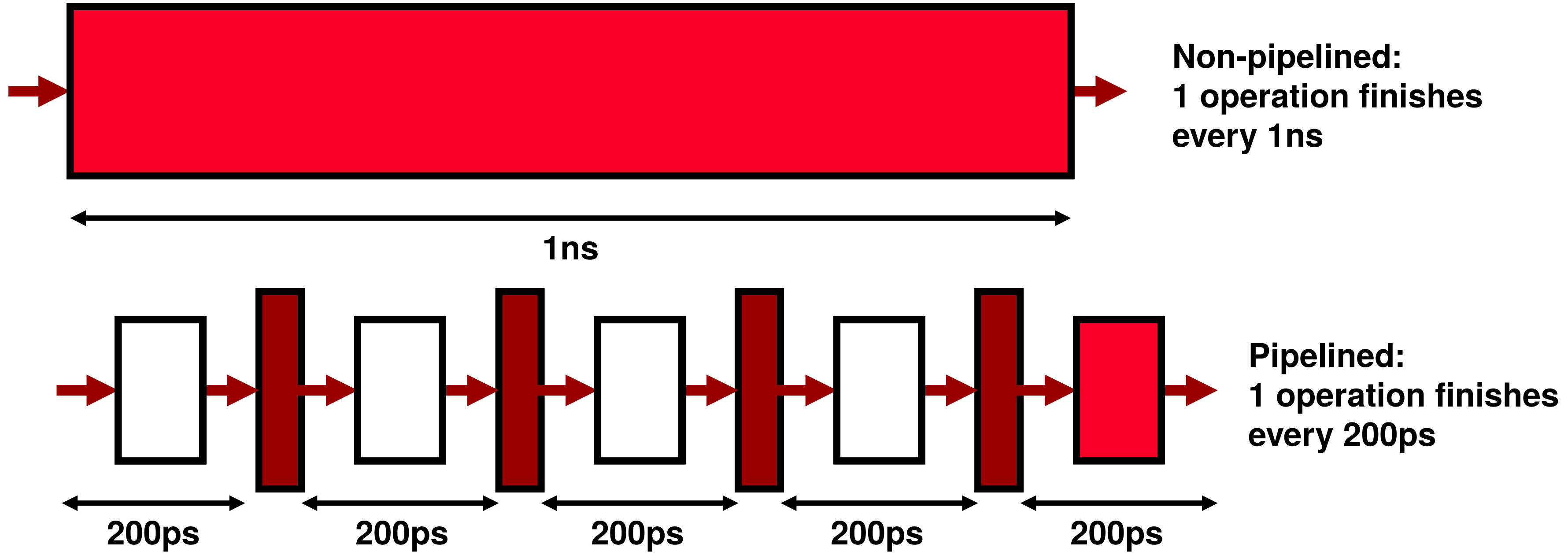
  Separate each piece with a pipeline register

-



Pipelining a Digital System

Why do this? Because it's <u>faster</u> for repeated computations



Comments about pipelining

Pipelining increases throughput, but not latency
Answer available every 200ps, BUT

-A single computation still takes 1ns

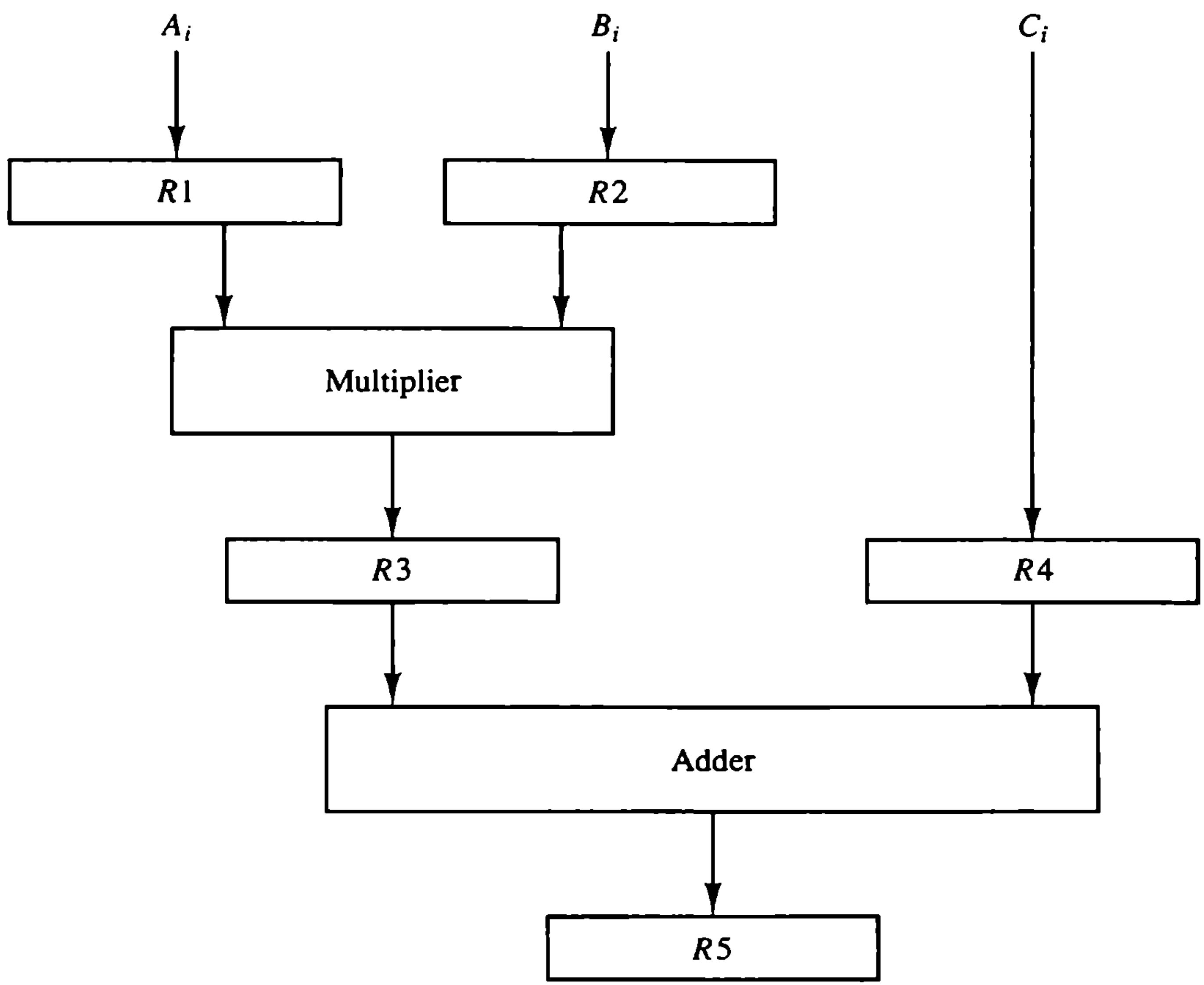
Limitations:
-Computations must be divisible into stage size

-Pipeline registers add overhead

- Suppose we need to perform multiply and add operation with a stream of numbers
- $A_i * B_i + C_i \quad \text{for } i = 1, 2, 3, \ldots, 7$
- Each subinstruction is implemented in a segment within the pipeline. Each segment has one or two regsiters and a combinational circuit
- The sub operations performed in each segement are as follows

$R1 \leftarrow A_i, \quad R2 \leftarrow B_i$	Input A_i and B_i
$R3 \leftarrow R1 * R2, \quad R4 \leftarrow C_i$	Multiply and input C_i
$R5 \leftarrow R3 + R4$	Add C_i to product

Example of Pipeline Processing



Content of Registers in Pipeline

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	$R1$	$R2$	$R3$	$R4$	$R5$
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	—	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$

Space Time Diagram of Pipeline

Segment:	1	2	3	4	5	6	7	8	9	→ Clock cycles
1	T_1	T_2	T_3	T_4	T_5	T_6				
2		T_1	T_2	T_3	T_4	T_5	T_6			
3			T_1	T_2	T_3	T_4	T_5	T_6		
4				T_1	T_2	T_3	T_4	T_5	T_6	

Speedup

Speedup from pipeline

= Average instruction time unpiplined/Average instruction time pipelined

Consider a case for k-segment pipeline with a clock cycle time tp to execute n tasks. The first task T1 requires a time equal to k*tp to complete its operation since there are k segments in pipeline. The remaining n-1 tasks emerge from the pipe at a rate of one task per clock cycle and they will be completed in k+n-1 clock cycles.

Next, to concsider an unpipeline unit that performs the same operation and takes a time equal to tn to complete the task. The total time required fro n tasks is n*tn. The speed up of a pipeline processing over an equivalent non-pipeline processing is defined by the ratio

$$S = \frac{nt_n}{(k + n - 1)t_p}$$

Speedup

- As the number of tasks increase n becomes much larger than k-1, and k+n-1 approaches

  the value of n. Under this condition, the speed up becomes
- $$S = \frac{t_n}{t_p}$$

  If we assume the the time taken to process the task is the same as in the pipeline and nonpipeline circuits, we will have $t_n = kt_p$
- The speedup then reduces to numer of stages of pipeline

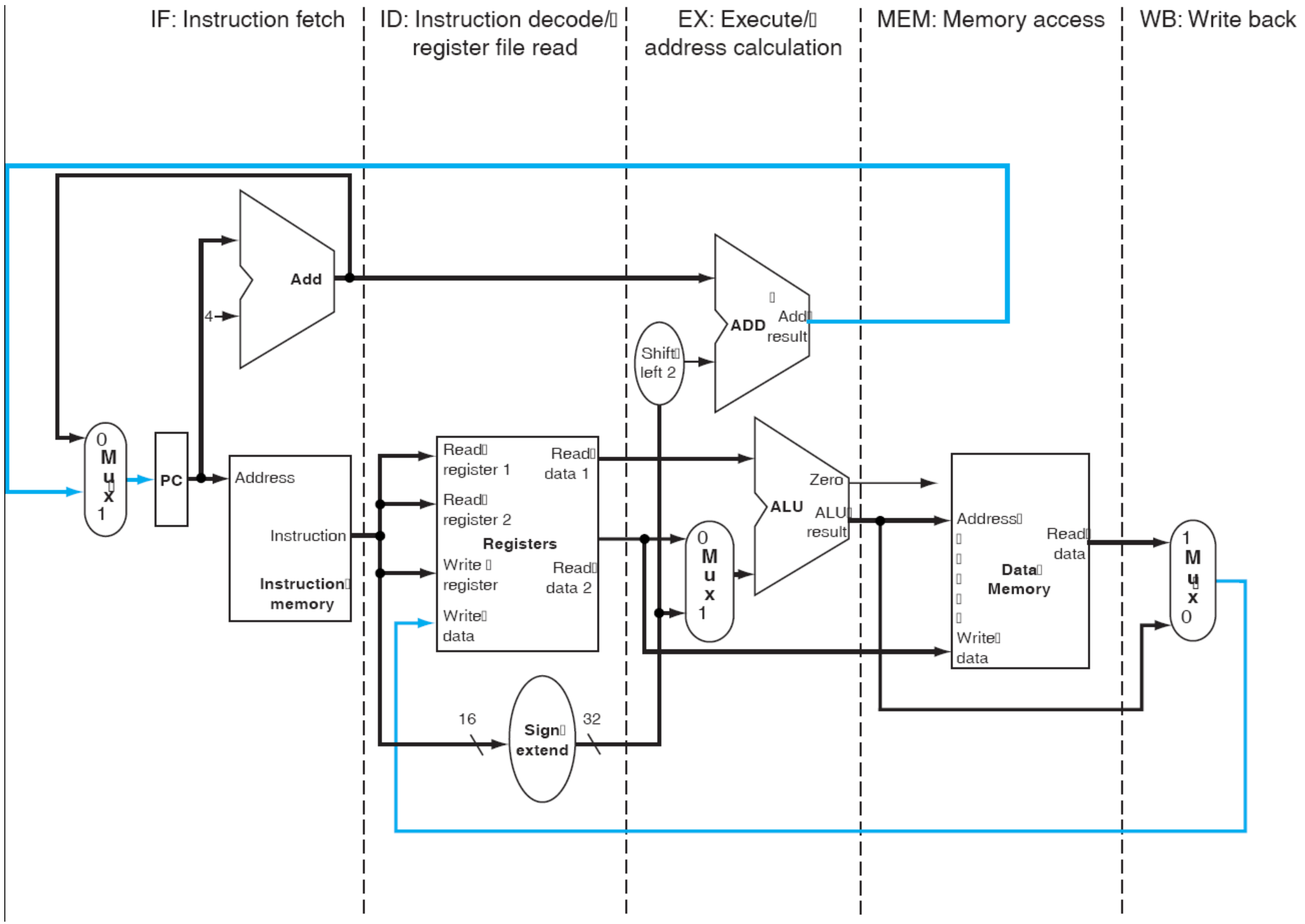
$$S = \frac{kt_p}{t_p} = k$$

Pipelining a Processor

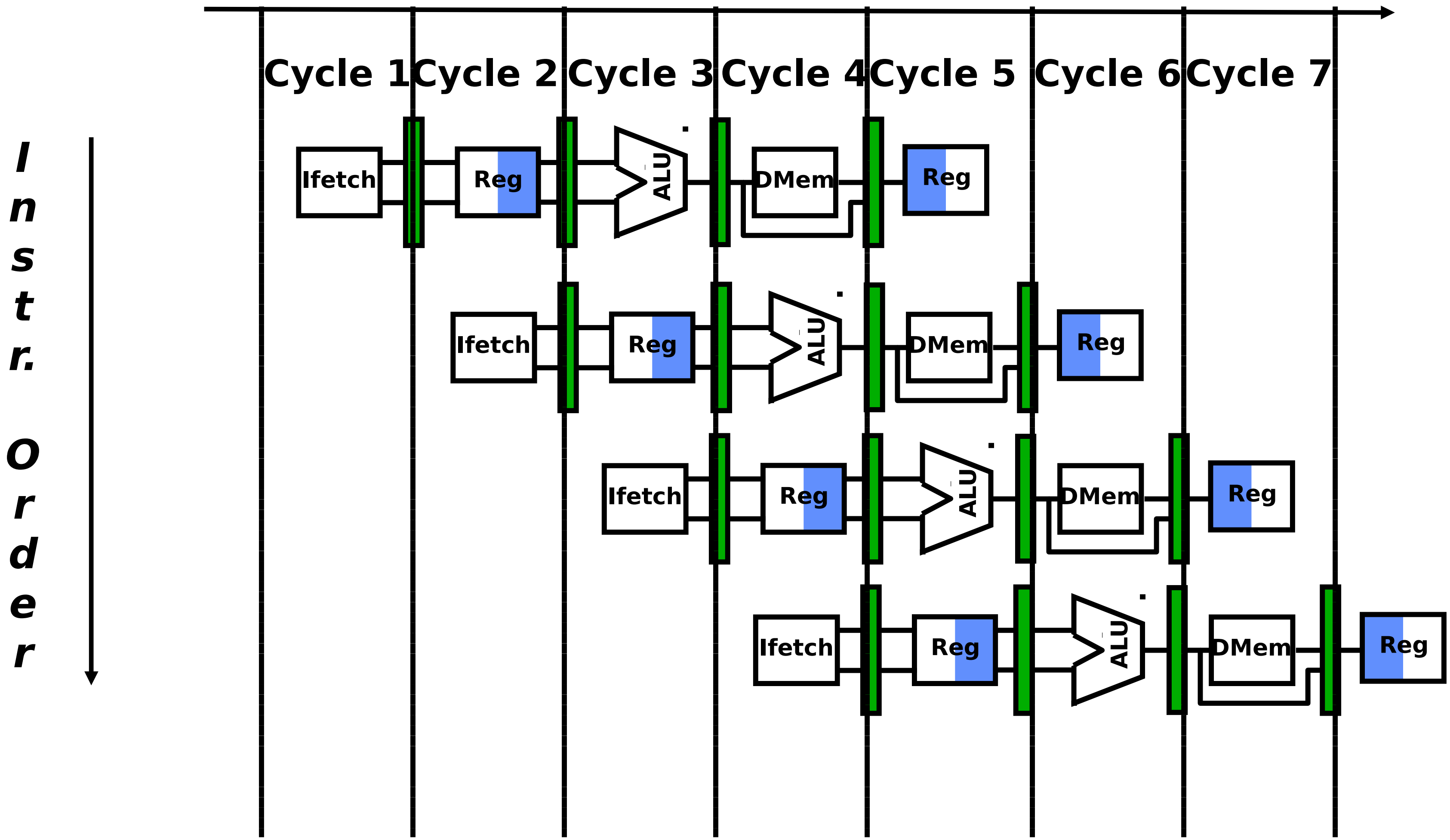
- Recall the 5 steps in instruction execution:
  1. Instruction Fetch (IF)
  2. Instruction Decode and Register Read (ID)
  3. Execution operation or calculate address (EX)
  4. Memory access (MEM)
  5. Write result into register (WB)

- Review: Single-Cycle Processor
  – All 5 steps done in a single clock cycle
  – Dedicated hardware required for each step

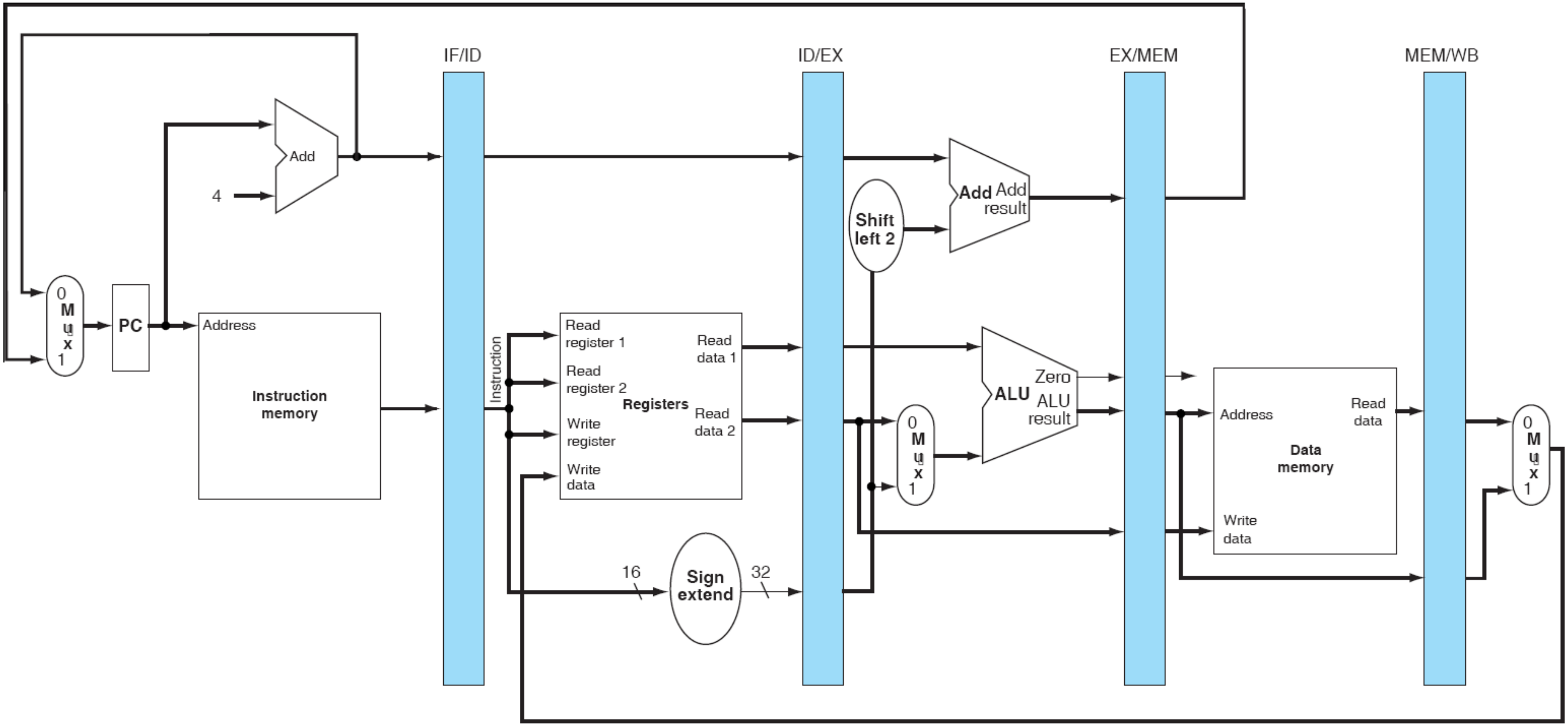
Review - Single-Cycle Processor



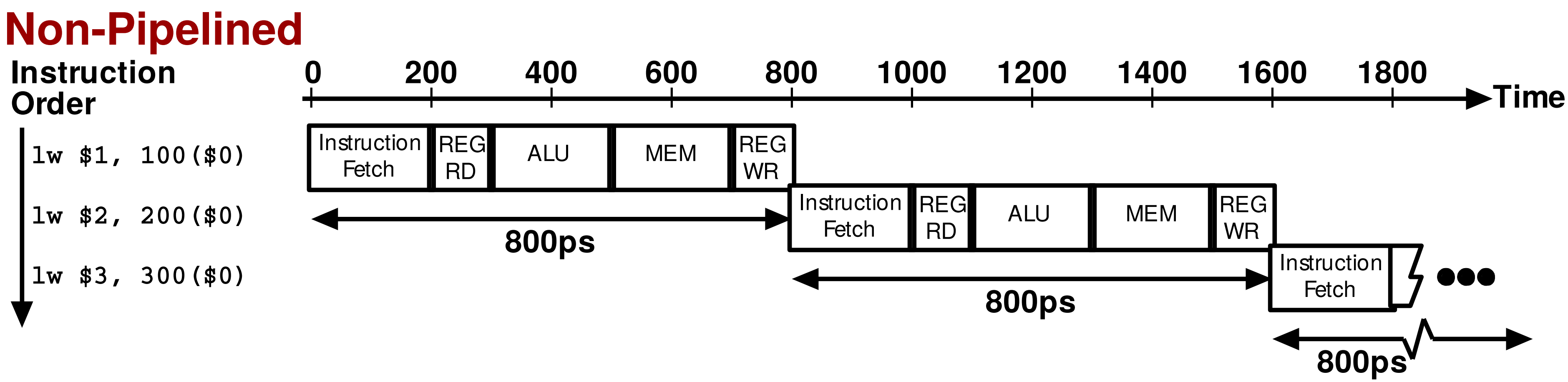
The Basic Pipeline For MIPS

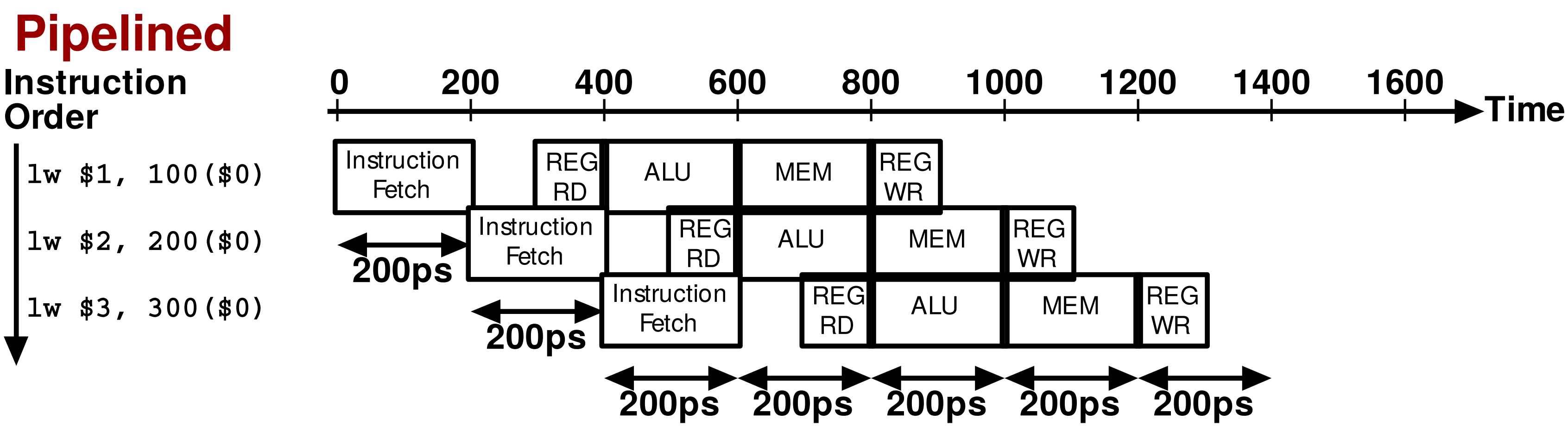


Basic Pipelined Processor



Single-Cycle vs. Pipelined Execution





Comments about Pipelining

The good news

- Multiple instructions are being processed at same time
- This works because stages are isolated by registers
- Best case speedup of N

The bad news

- Instructions interfere with each other - hazards

  Example: different instructions may need the same piece of hardware (e.g., memory) in same clock cycle

  Example: instruction may require a result produced by an earlier instruction that is not yet complete

Pipeline Hazards

Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle

Structural hazards: two different instructions use same h/w in same cycle

Data hazards: Instruction depends on result of prior instruction still in the pipeline

Control hazards: Pipelining of branches & other instructions that change the PC