

Design of Digital Circuits and Systems

Pipelining

Instructor: Vikram Iyer

Teaching Assistants:

Ariel Kao

Josh Wentzien

Selim Saridede

Jared Yoder

Derek Thorp

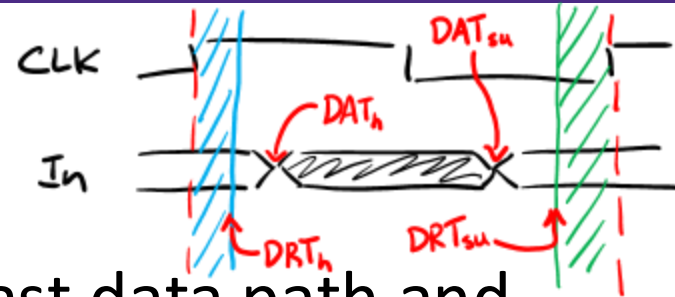
Adapted from material by Justin Hisa

Relevant Course Information

- ❖ Quiz 3 starts at 11:50 am
- ❖ Lab 4 due Friday (5/9), demos next week
- ❖ Homework 5 released today, due next Friday (5/16)
 - Static Timing Analysis and Pipelining
- ❖ Lab 5 released today, due in two weeks (5/23)
 - Hardest lab for many students
 - You will need to use the VGA interface on LabsLand
 - There's a creative component and opportunity for extra credit

Review: Timing Closure

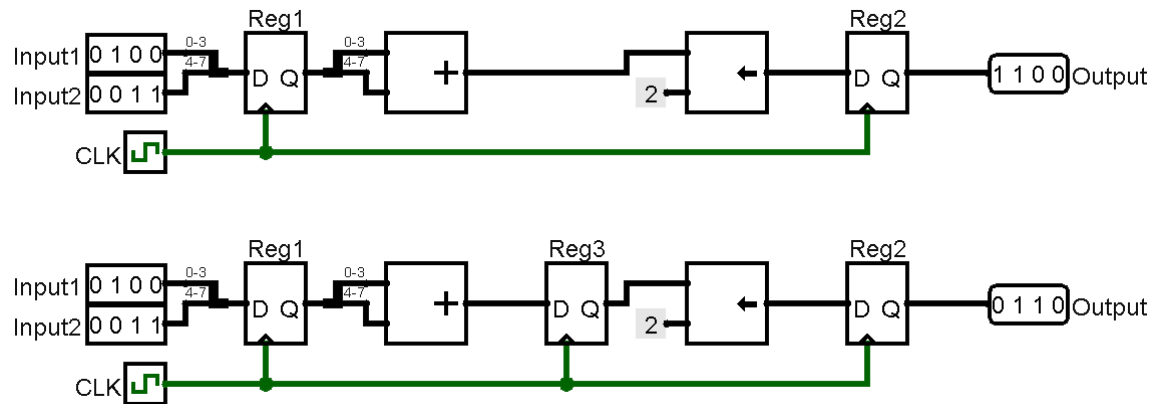
(setup and hold slack)



- ❖ Fixing hold violations: caused by fast data path and destination register's clock latency
 - Add delay in the data path with buffers or pairs of inverters (done automatically by Quartus)
- ❖ Fixing setup violations: data arrives too late compared to the destination register's clock speed
 - Slow down the clock (undesirable)
 - Tell fitter to try harder or confine logic to a smaller area
 - Rewrite code to simplify logic
 - ■ Add *pipelining* (today!)

Pipelining

- ❖ **Pipelining** is a set of data processing elements connected in series with buffer storage inserted between
 - In digital systems, the buffer storage are FFs & registers and data processing elements are stages of combinational logic
 - In its simplest form, can be thought of as adding registers in the middle of a computation to reduce our clock period



Performance

- ❖ What does it mean to say X performs better than Y?
- ❖ Silly example: a Tesla vs. a school bus
 - 2015 Tesla Model S P90D
 - 5 passengers, 2.8 secs in quarter mile
 - 2011 Type D school bus
 - Up to 90 passengers, quarter mile time?



Performance

- ❖ What does it mean to say X performs better than Y?
- ❖ Silly example: a Tesla vs. a school bus
 - 2015 Tesla Model S P90D
 - 5 passengers, 2.8 secs in quarter mile
 - 2011 Type D school bus
 - Up to 90 passengers, quarter mile time?

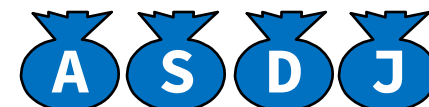


Measurements of Performance

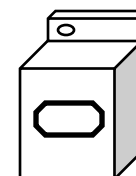
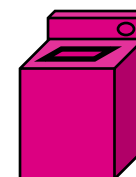
- ❖ *Latency* (or *response time* or *execution time*)
 - Time to complete one task
- ❖ *Throughput* (or *bandwidth*)
 - Tasks completed per unit time

Analogy: Doing Laundry

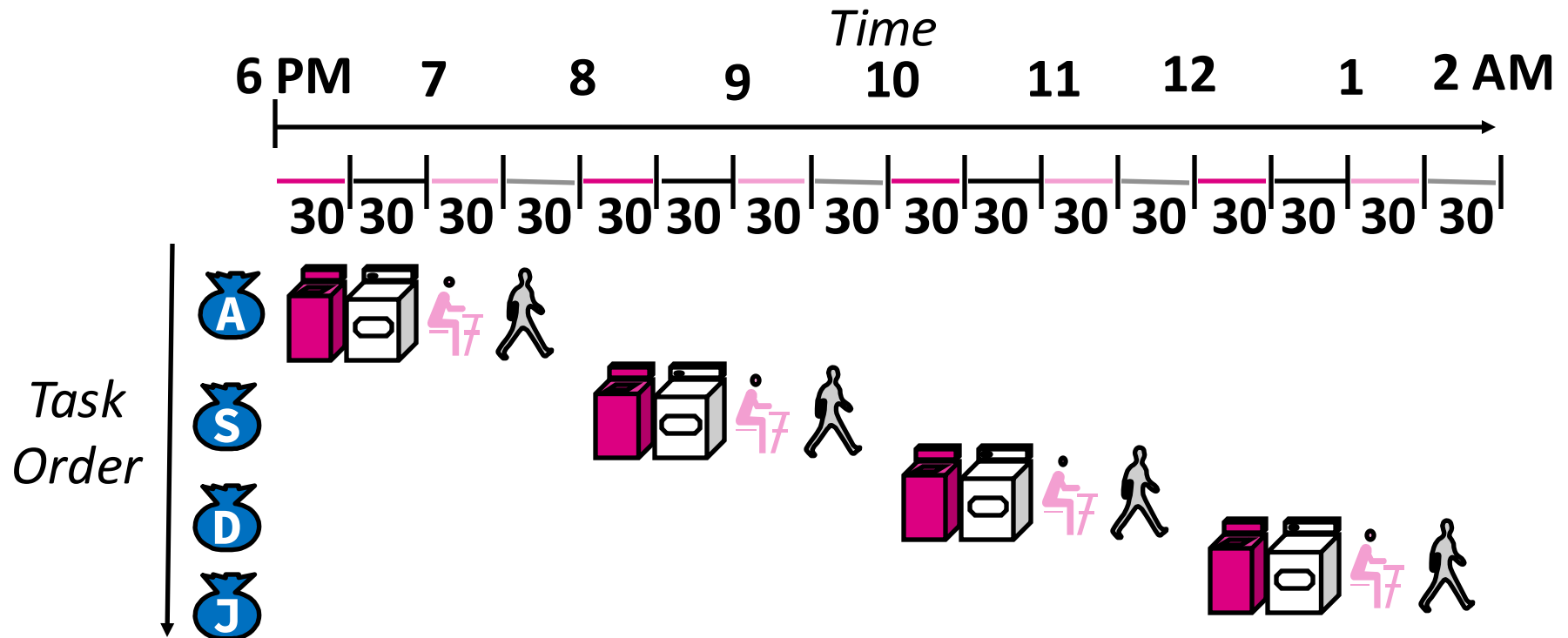
❖ Ariel, Selim, Derek and Josh/Jared each have one load of clothes to wash, dry, fold, and put away



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

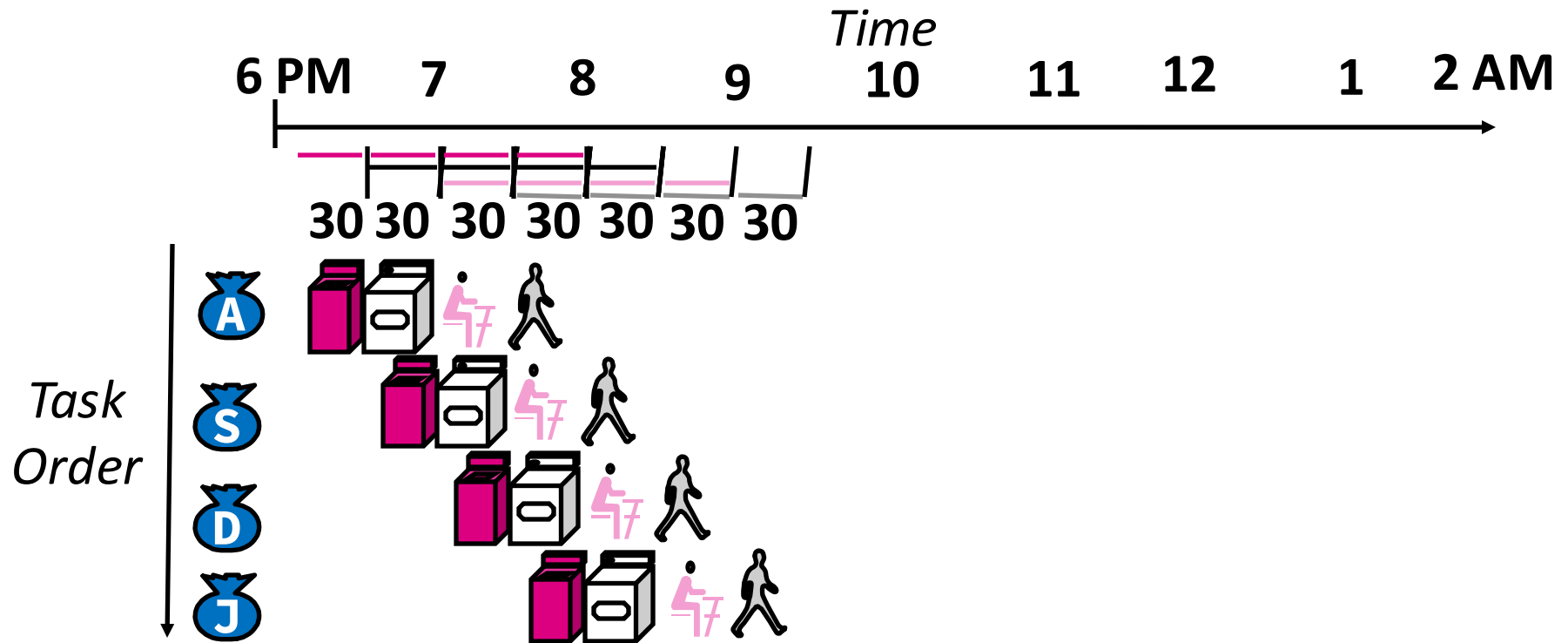


Sequential Laundry



- Sequential laundry takes 8 hours for 4 loads

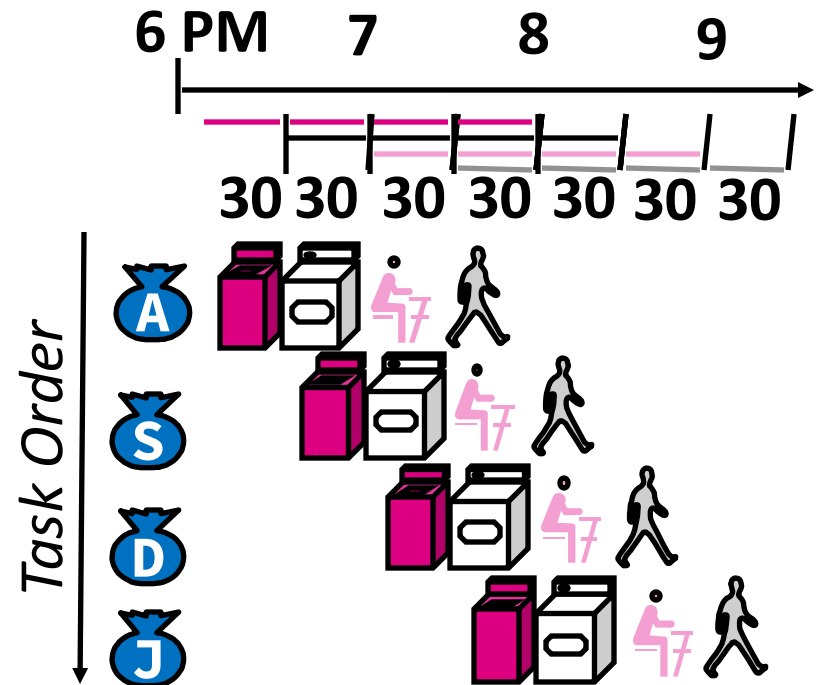
Pipelined Laundry



- Pipelined laundry takes 3.5 hours for 4 loads!

Pipelining Notes

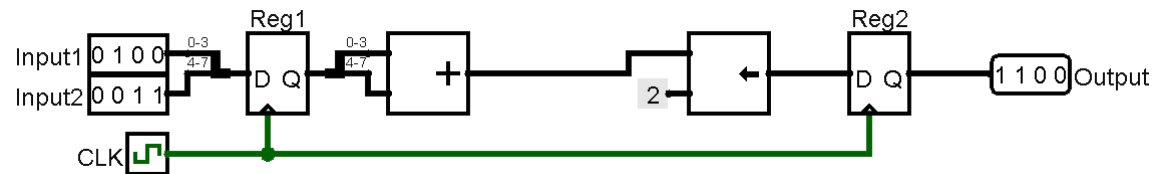
- ❖ Pipelining helps throughput of overall workload, but not latency of single task
 - Reduction in critical pathway allows for shorter clock period
- ❖ *Multiple* tasks operating simultaneously using different resources
 - Executing different parts of multiple computations at the same time using the same hardware – like an assembly line
 - Greater utilization of logic resources



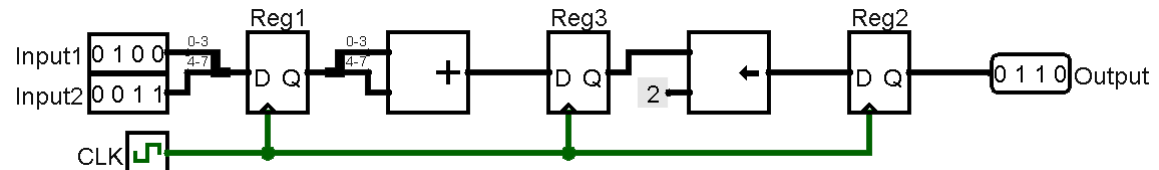
Pipelined Performance Example

- ❖ Assume $t_{CO} = 10$ ns, $t_{add} = 90$ ns, $t_{shl} = 50$ ns
 - For simplicity, assume $t_{clk} = t_{wire} = t_h = t_{su} = 0$
- ❖ Solve for the minimum clock period for each circuit
 - Given this minimum clock period, solve for the latency and throughput of each circuit

■ Circuit 1:



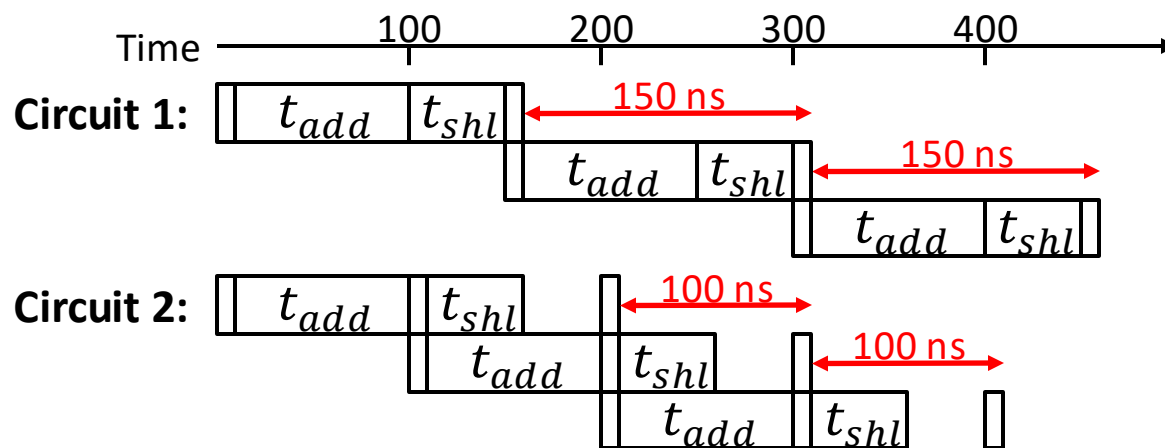
■ Circuit 2:



Pipeline Performance

❖ In theory, can measure “speedup” as the ratio in time per completion (TC) of computations

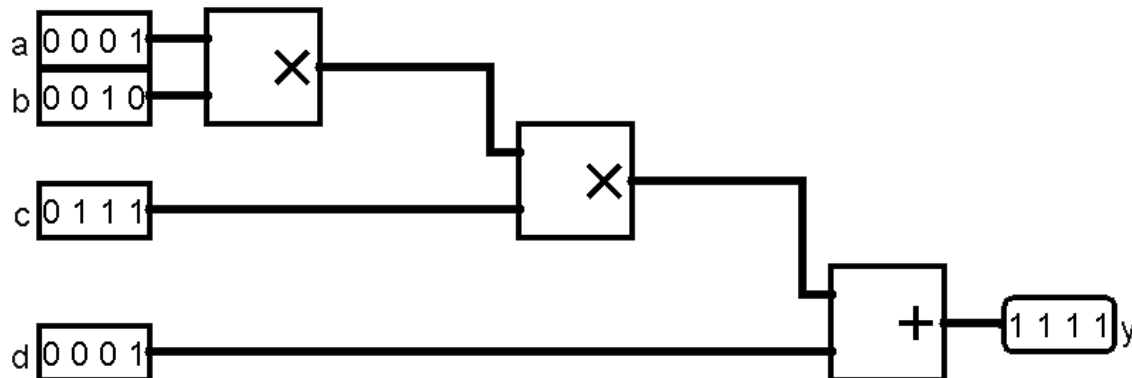
- $\text{speedup} = \frac{TC_{\text{original}}}{TC_{\text{pipelined}}}$
- $\text{speedup}_{\text{max}} = \# \text{ of pipeline stages}$
- Speedup is reduced by *unbalanced* stages (and t_{CO}):



Technology Break

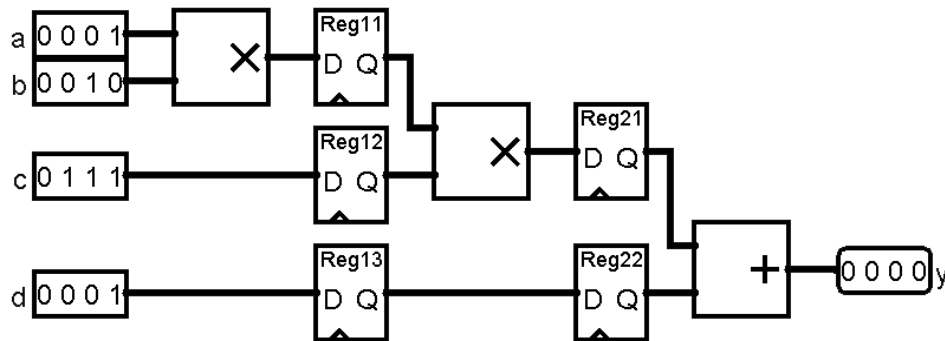
Pipeline Registers

- ❖ Where to add pipeline registers?
 - For a given computation, all paths from any input to output must pass through the *same number* of pipeline registers
- ❖ Example: $y_i = (a_i \times b_i) \times c_i + d_i$



Pipeline Registers

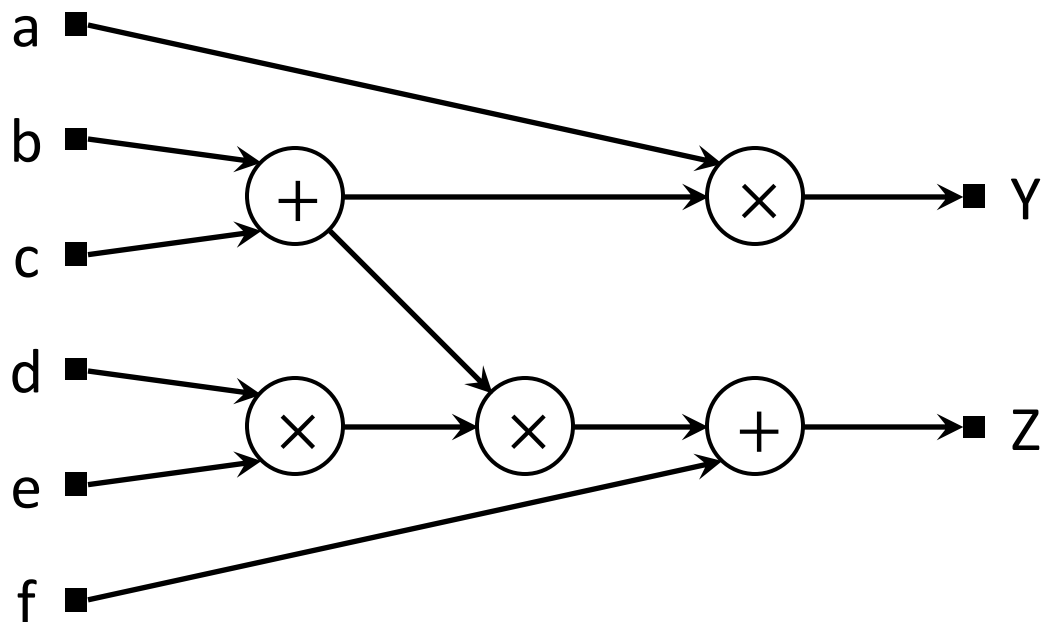
- ❖ Where to add pipeline registers?
 - For a given computation, all paths from any input to output must pass through the *same number* of pipeline registers
- ❖ Example: $y_i = (a_i \times b_i) \times c_i + d_i$
 - Signal flow:



Cycle	1	2	3	4
Reg11				
Reg12				
Reg13				
Reg21				
Reg22				

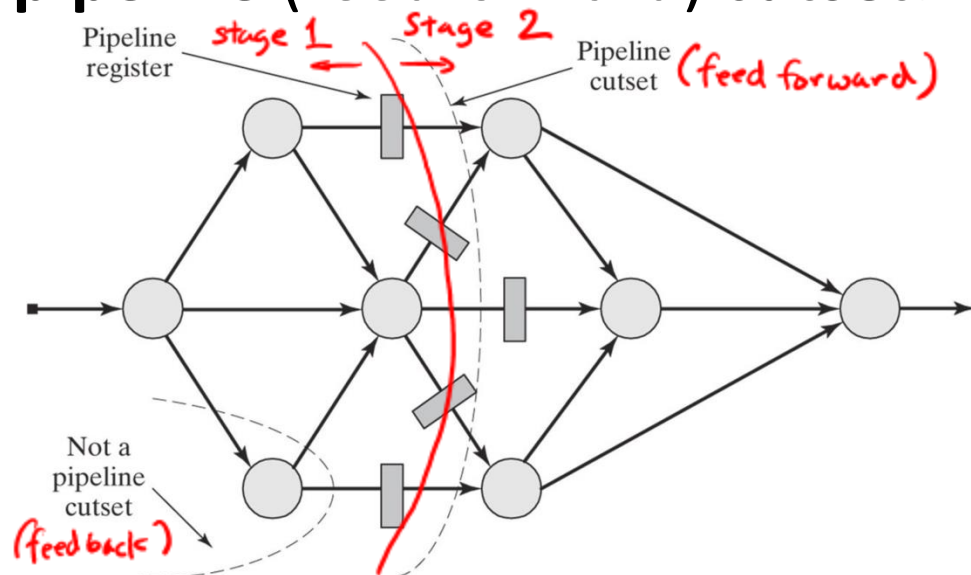
Data Flow Graph

- ❖ A **data flow graph** (DFG) is a visualization tool that can be used to simplify circuits into *directed graphs*
 - Nodes are computations (and their delays)
 - Edges represent data dependencies



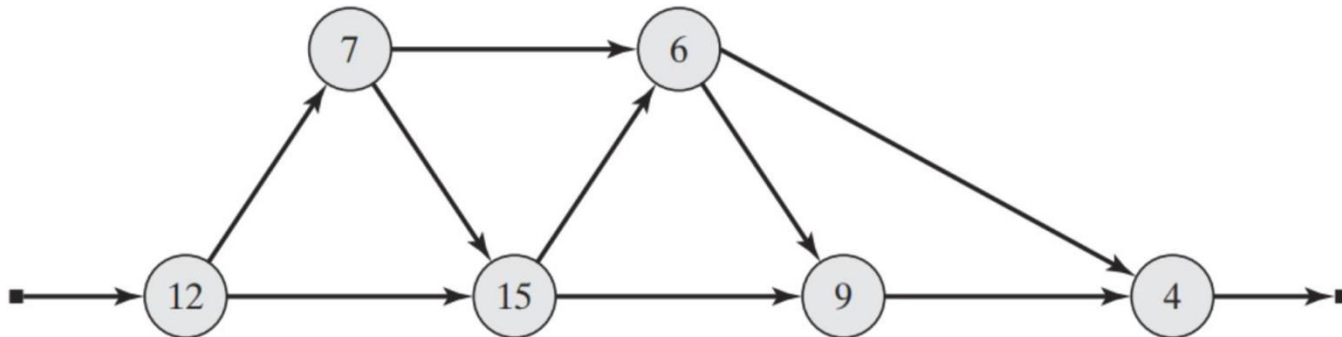
Pipeline Cutset

- ❖ A **cutset** is a set of edges that form two disjoint graphs when removed/cut
 - *Feedforward* cutset: data travels only forward in the cutset
 - *Feedback* cutset: data travels in both directions in the cutset
- ❖ Pipelining is done by placing a register along every edge in a **pipeline (feedforward) cutset**:



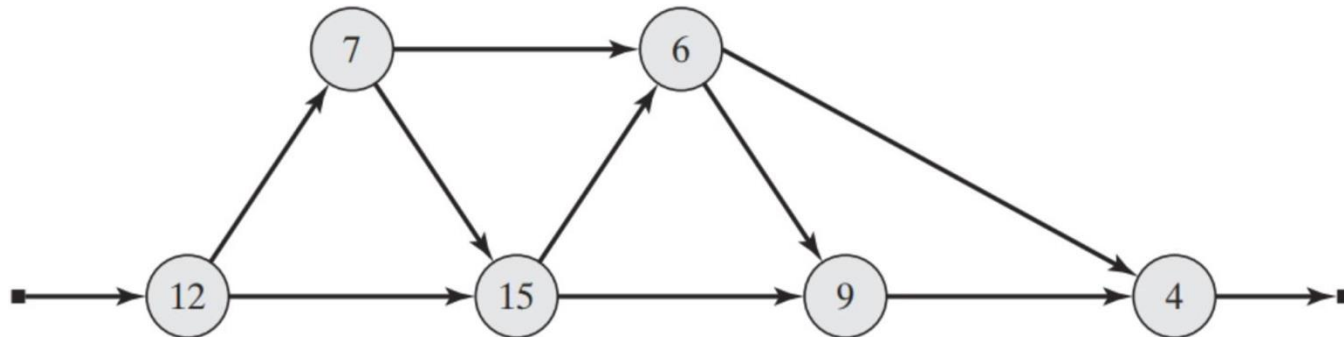
Pipeline Cutset Example

- ❖ The following data flow graph shows the propagation delay in each node
 - For simplicity, assume $t_{CO} = 0$
 - Original (non-pipelined) performance:



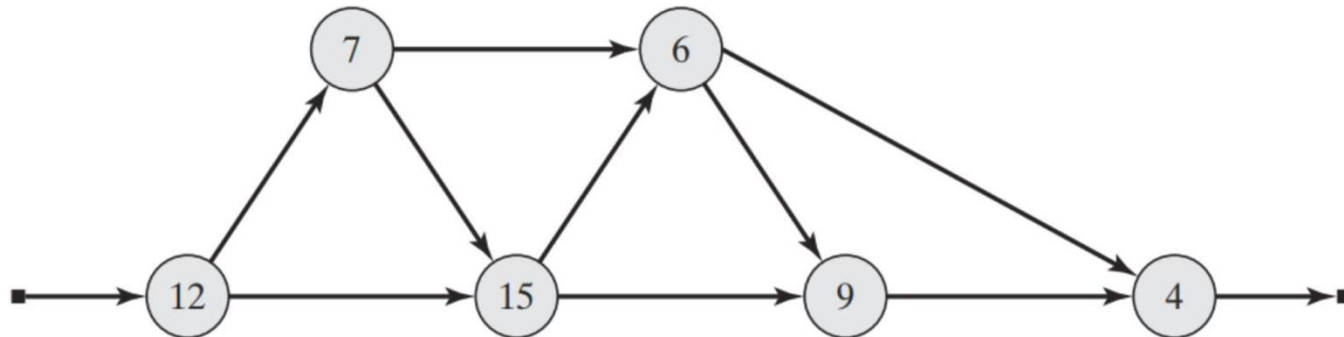
Pipeline Cutset Example

- ❖ The following data flow graph shows the propagation delay in each node
 - Create 2-3 different pipelined versions of this DFG and compute the maximum delay of each stage and minimum clock period for the pipelined computation
 - For simplicity, assume $t_{CO} = 0$



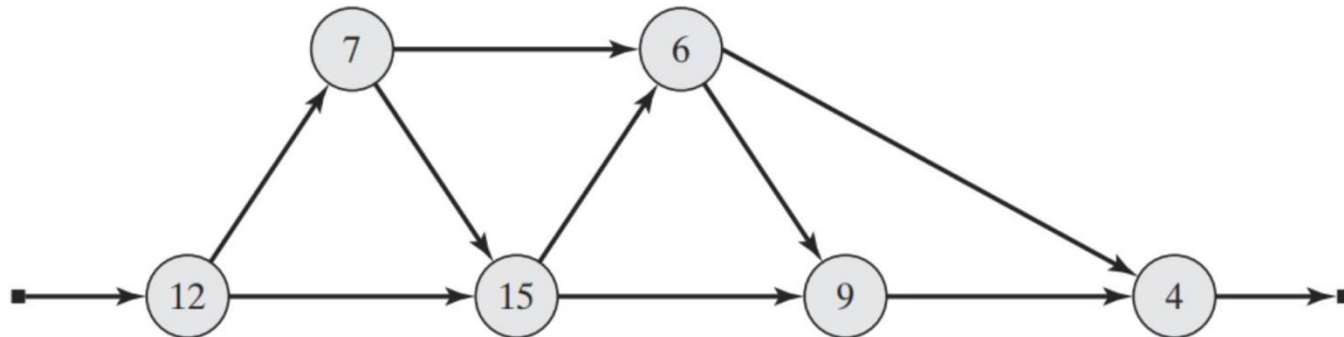
Pipeline Cutset Example

- ❖ The following data flow graph shows the propagation delay in each node
 - Create 2-3 different pipelined versions of this DFG and compute the maximum delay of each stage and minimum clock period for the pipelined computation
 - For simplicity, assume $t_{CO} = 0$



Pipeline Cutset Example

- ❖ The following data flow graph shows the propagation delay in each node
 - Create 2-3 different pipelined versions of this DFG and compute the maximum delay of each stage and minimum clock period for the pipelined computation
 - For simplicity, assume $t_{CO} = 0$

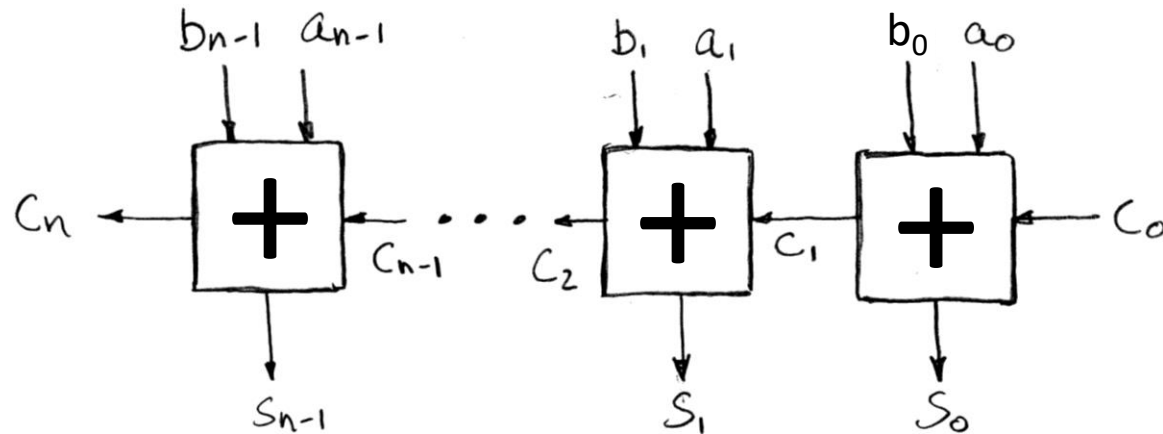


Pipeline Design Questions

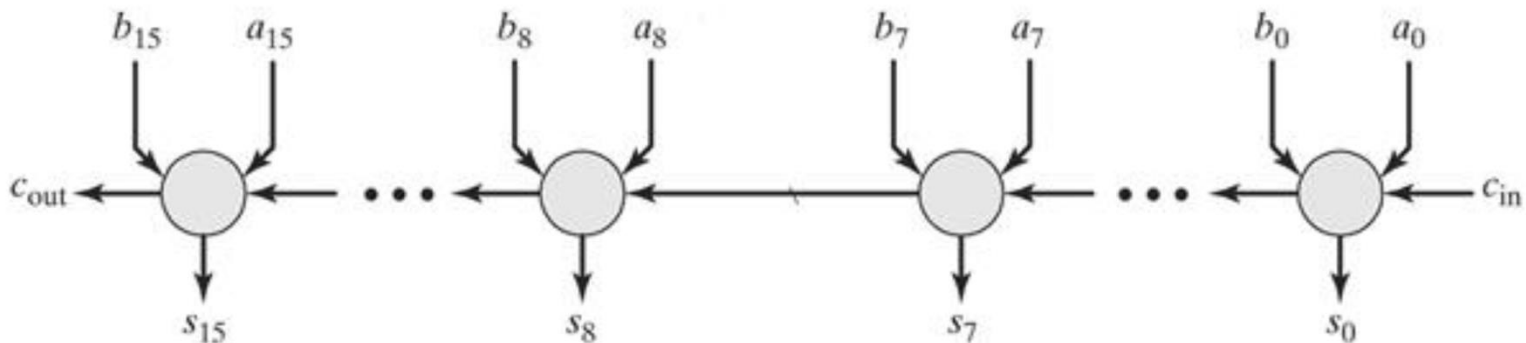
- ❖ When should I add pipelining?
 - Check if it is possible first (*i.e.*, a pipeline cutset must exist)
 - Want to reduce the **critical path** in your computation/system
 - Your system can afford the increase in latency and hardware
- ❖ Where do the pipeline registers go?
 - Must be placed at proper pipeline cutsets
 - Want to make pipeline stages as balanced as possible to maximize speedup

Design Example: 16-bit Ripple-Carry Adder

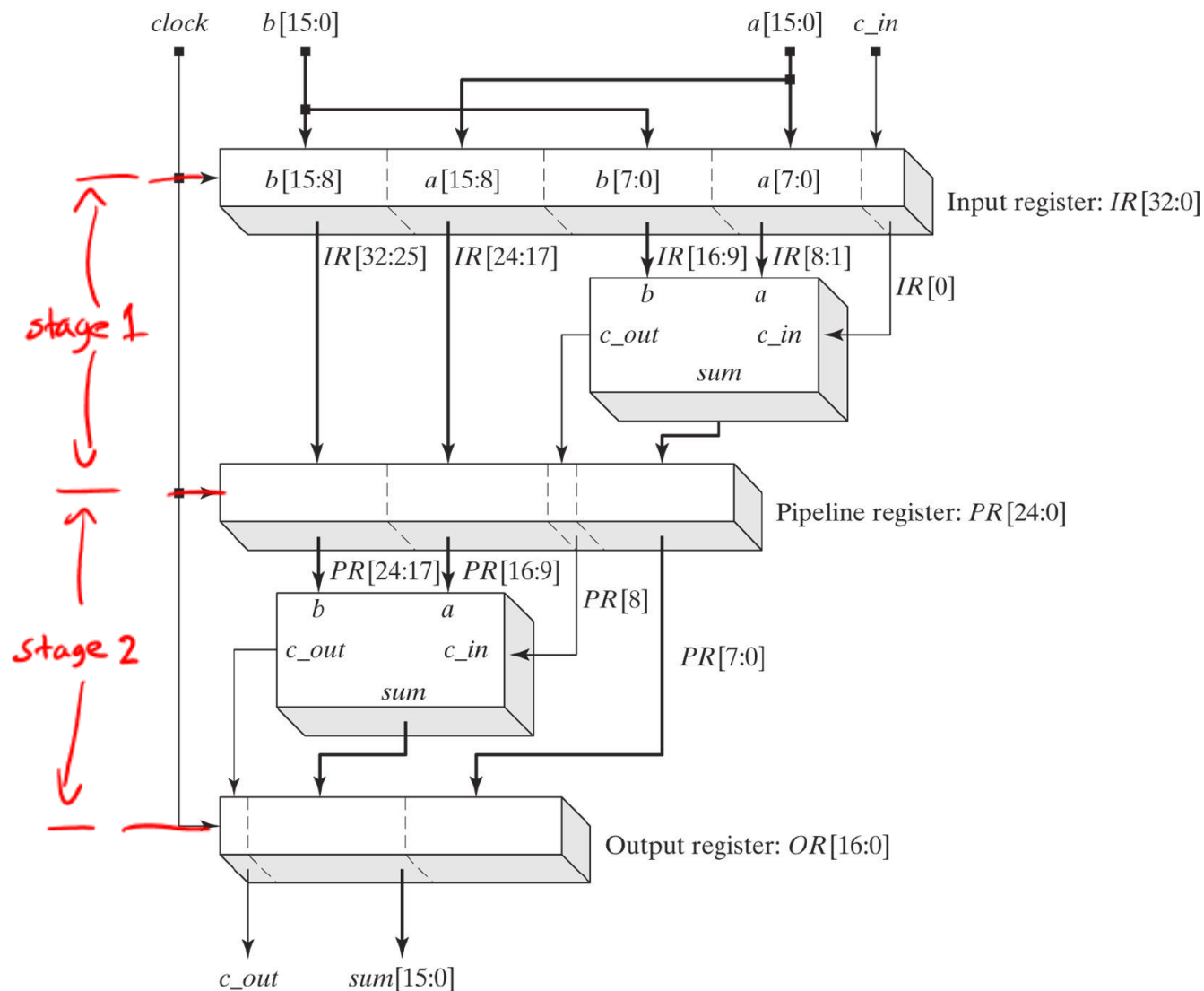
- ❖ Problem: C_n takes a long time to compute!



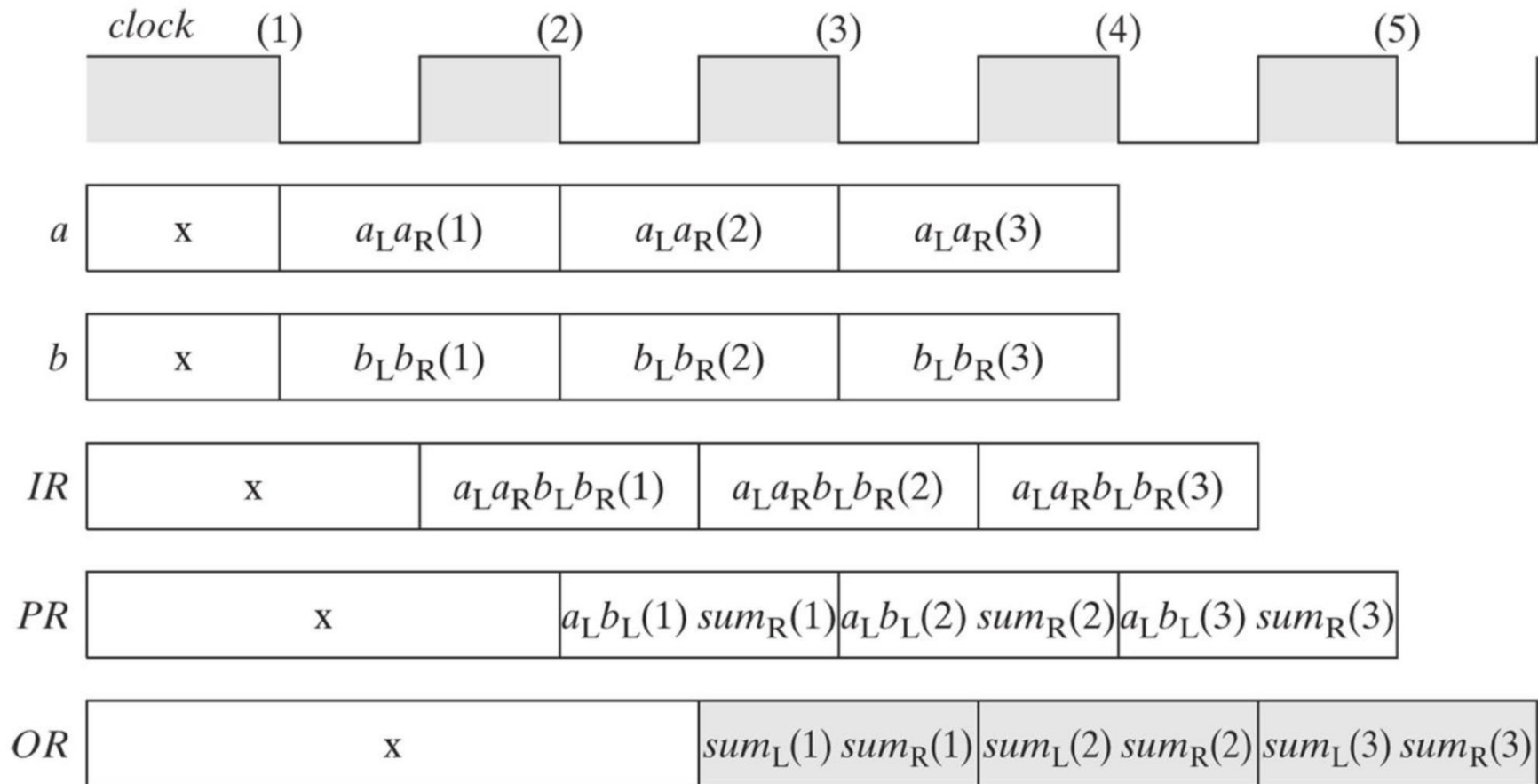
- ❖ 2-stage pipeline: which cutset to use?



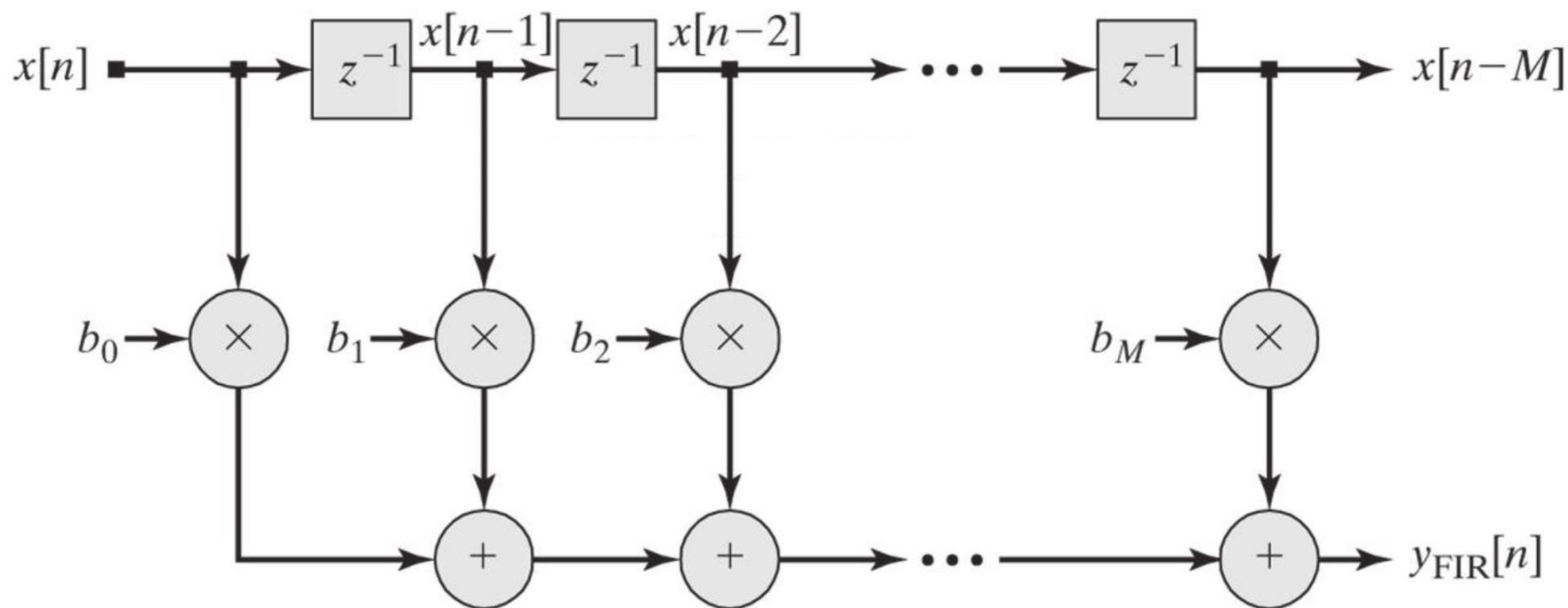
Design Example: 16-bit Pipelined Adder



Design Example: 16-bit Pipelined Adder



Design Example: FIR Filter



Design Example: Pipelined FIR Filter

