Design of Digital Circuits and Systems

Pipelining

Instructor: Vikram Iyer

NOTE: in class notes didn't save, but these should be very similar

Teaching Assistants:

Ariel Kao

Selim Saridede

Derek Thorp

Josh Wentzien

Jared Yoder

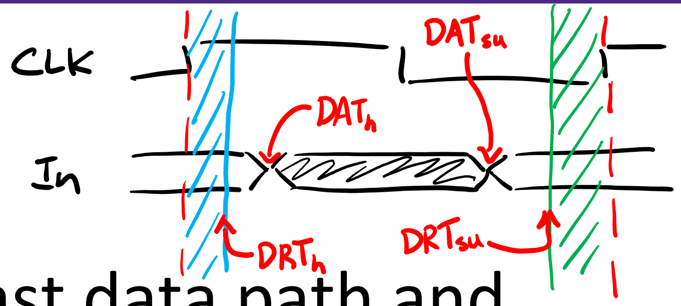
Adapted from material by Justin Hisa

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)

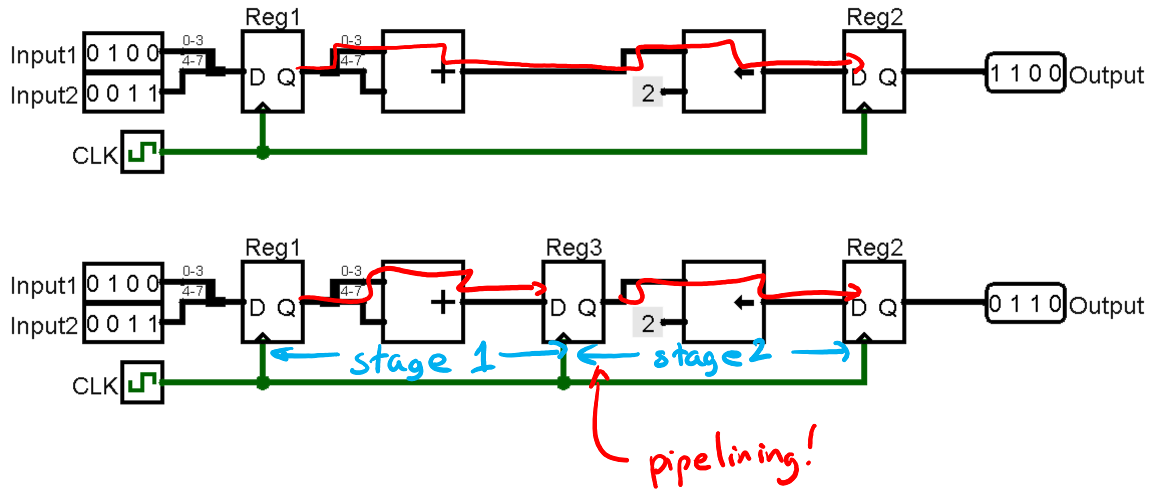


- Fixing **hold violations**: caused by fast data path and destination register's clock latency
  - Add delay in the data path with buffers or pairs of inverters (done automatically by Quartus)

- 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?

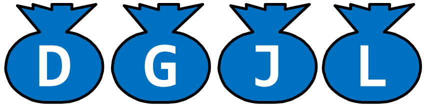
- Silly example: a Tesla vs. a school bus
  - 2015 Tesla Model S P90D
    - 5 passengers, 2.8 secs in quarter mile
  - 2011 Type D school bus
    - Up to 90 passengers, quarter mile time?

Measurements of Performance

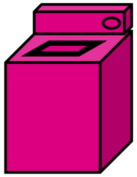
- *Latency* (or *response time* or *execution time*)
  - Time to complete one task
- *Throughput* (or *bandwidth*)
  - Tasks completed per unit time

Analogy: Doing Laundry

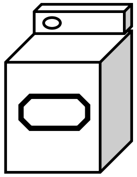
- **D**eepti, **G**ayathri, **J**ared, and **L**ancelot each have one load of clothes to ① wash, ② dry, ③ fold, and ④ put away



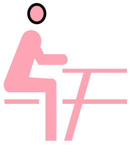
  - Washer takes 30 minutes



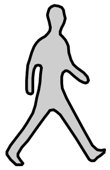
  - Dryer takes 30 minutes



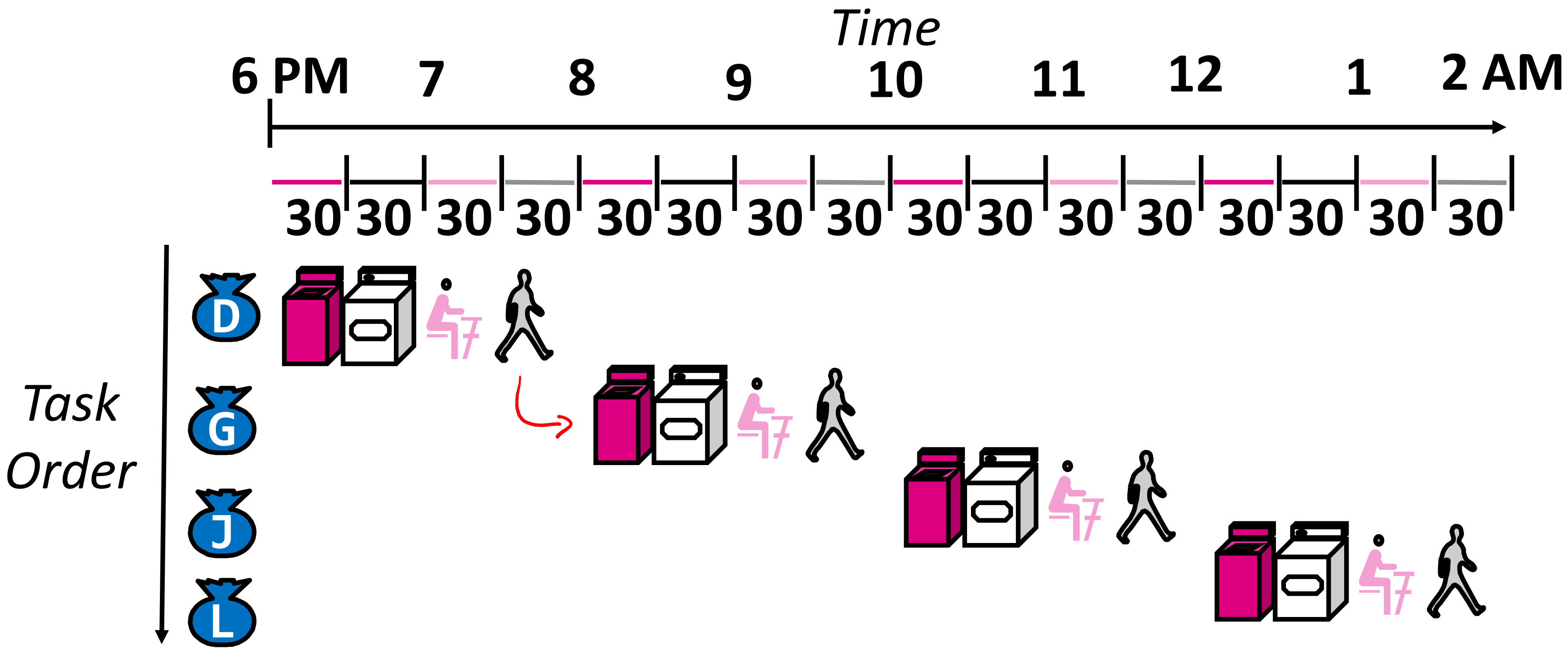
  - “Folder” takes 30 minutes



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



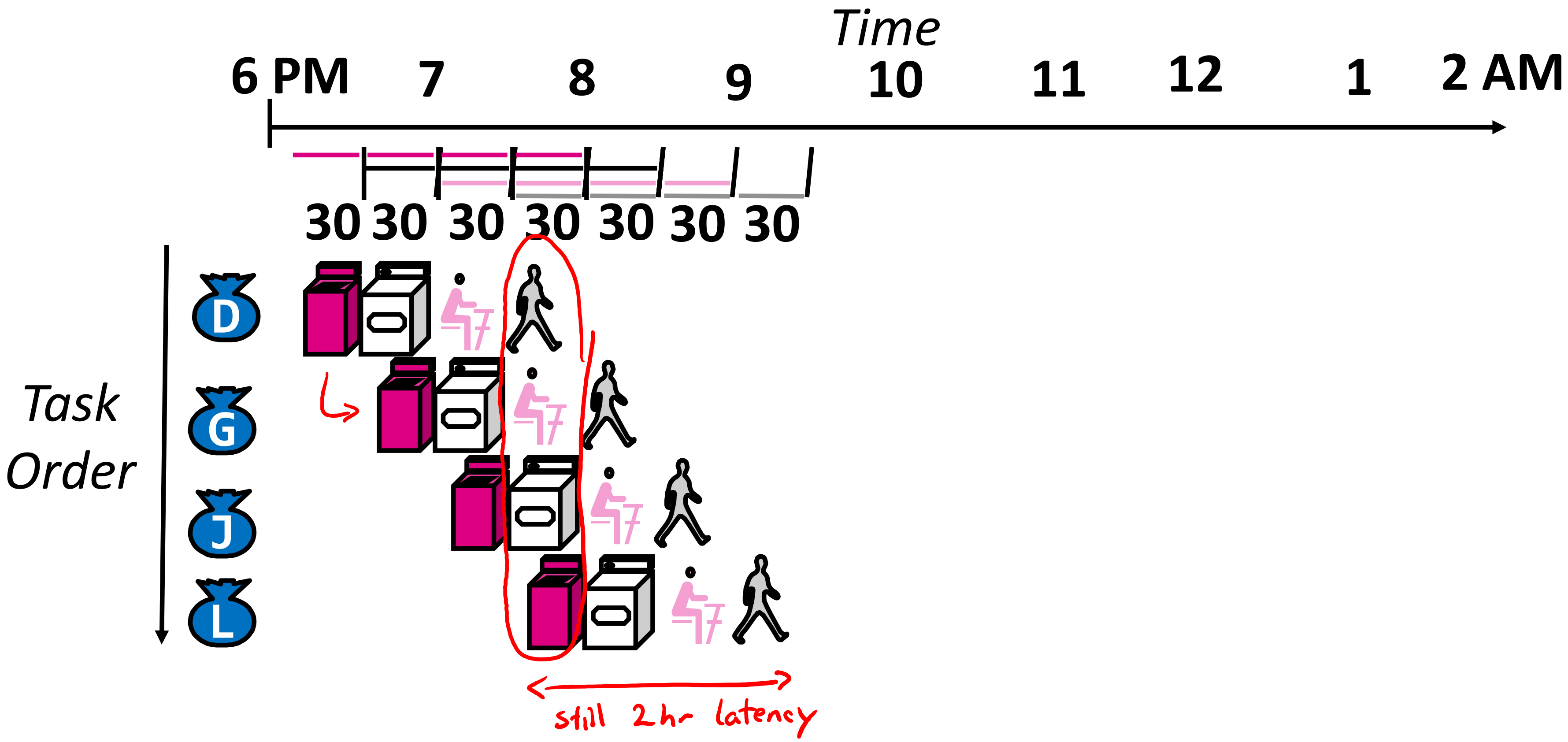
Sequential Laundry



- Sequential laundry takes 8 hours for 4 loads

only 1 person in laundry room at a time!

Pipelined Laundry



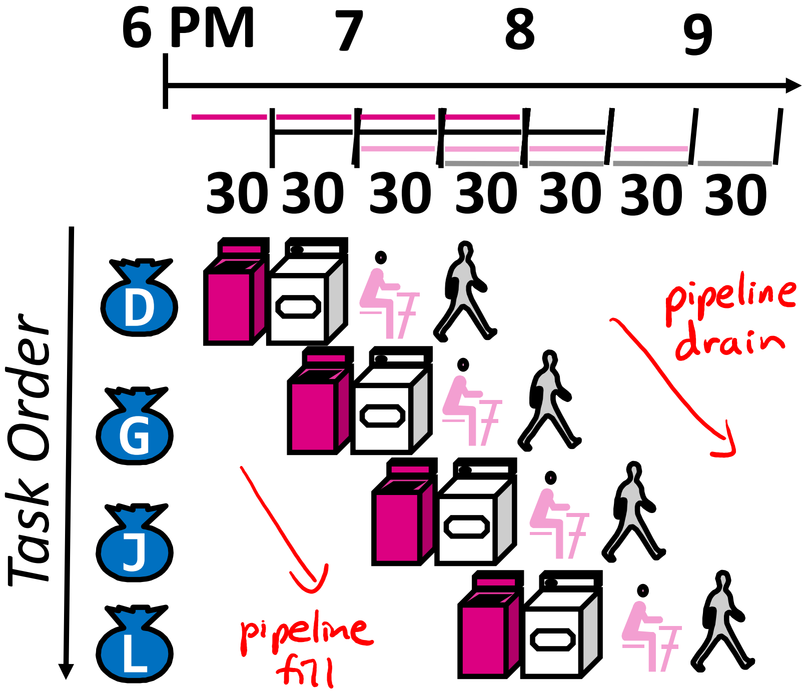
- Pipelined laundry takes 3.5 hours for 4 loads!

up to 4 people in the laundry room at a time!

at most 1 person using any station at a time

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

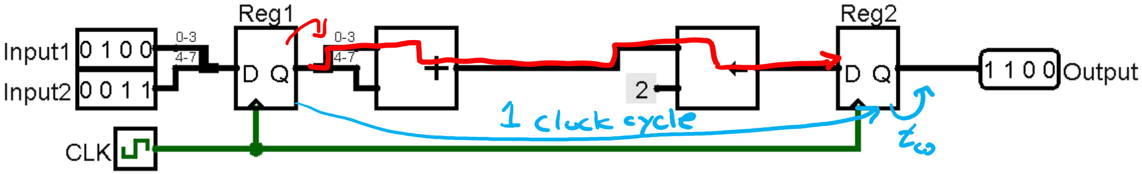
Pipelined Performance Example

- 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:

$T_{min} = 150\text{ns}$

throughput $= 1/T = 1/(150\text{ ns})$

latency $= T + t_{co} = 160\text{ns}$



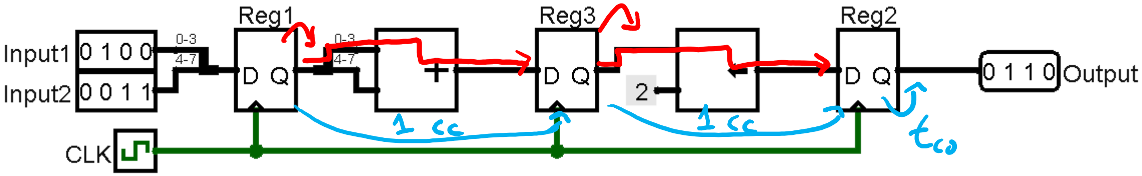
critical path $= t_{co} + t_{add} + t_{shl} \leq T - t_{su}$

  - Circuit 2:

$T_{min} = 100\text{ns}$

throughput $= 1/T = 1/(100\text{ ns})$

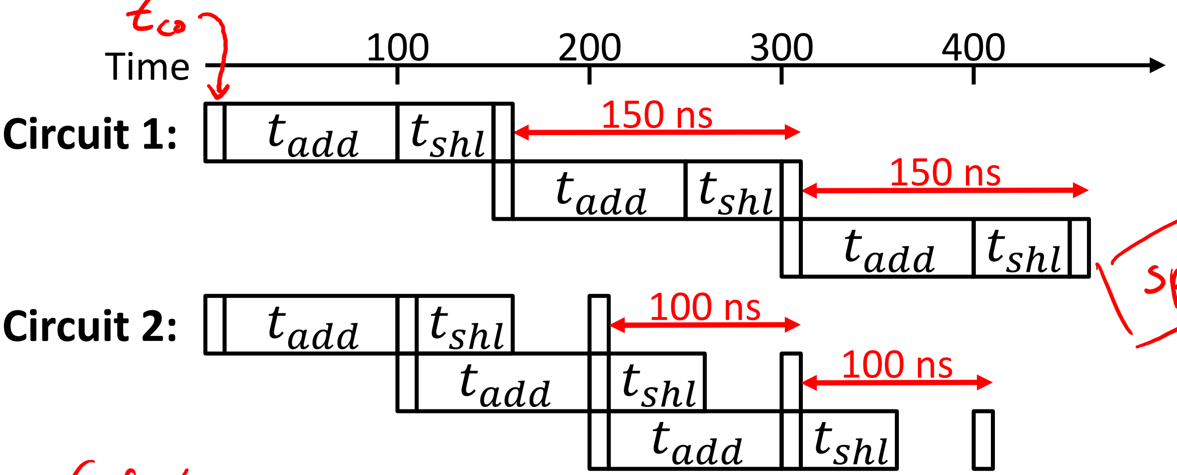
latency $= 2T + t_{co} = 210\text{ns}$



critical path $= \max(t_{co} + t_{add}, t_{co} + t_{shl}) \leq T - t_{su}$

Pipeline Performance

- In theory, can measure "speedup" as the ratio in time per completion (TC) of computations
  - $\text{speedup} = \frac{\text{TC}_{\text{original}}}{\text{TC}_{\text{pipelined}}}$
  - $\text{speedup}_{\text{max}}$ = # of pipeline stages
  - Speedup is reduced by *unbalanced* stages (and t_{CO}):



$\boxed{\text{Speedup} = \frac{150}{100} = 1.5 < 2}$

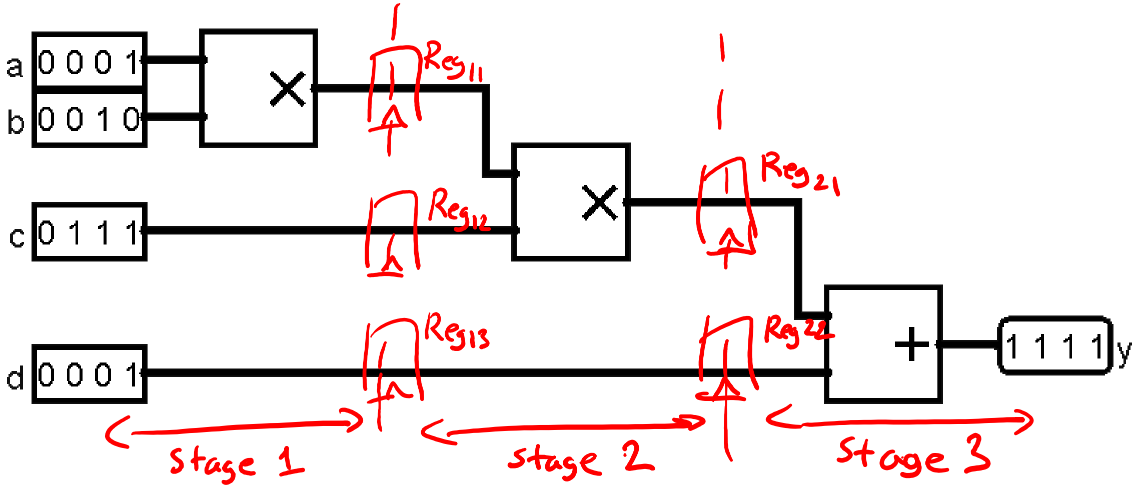
(if $t_{add} = t_{shl} = 70$ ns, could have used $T_{min} = 80$ns → speedup of 1.875)

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

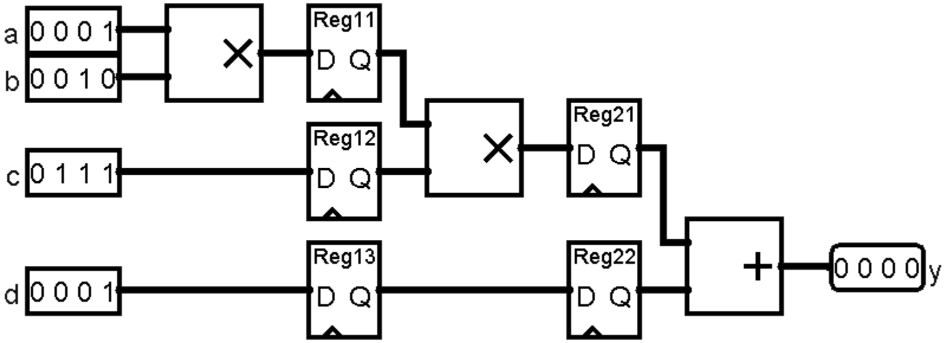
- Example: $y_i = (a_i \times b_i) \times c_i + d_i$



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

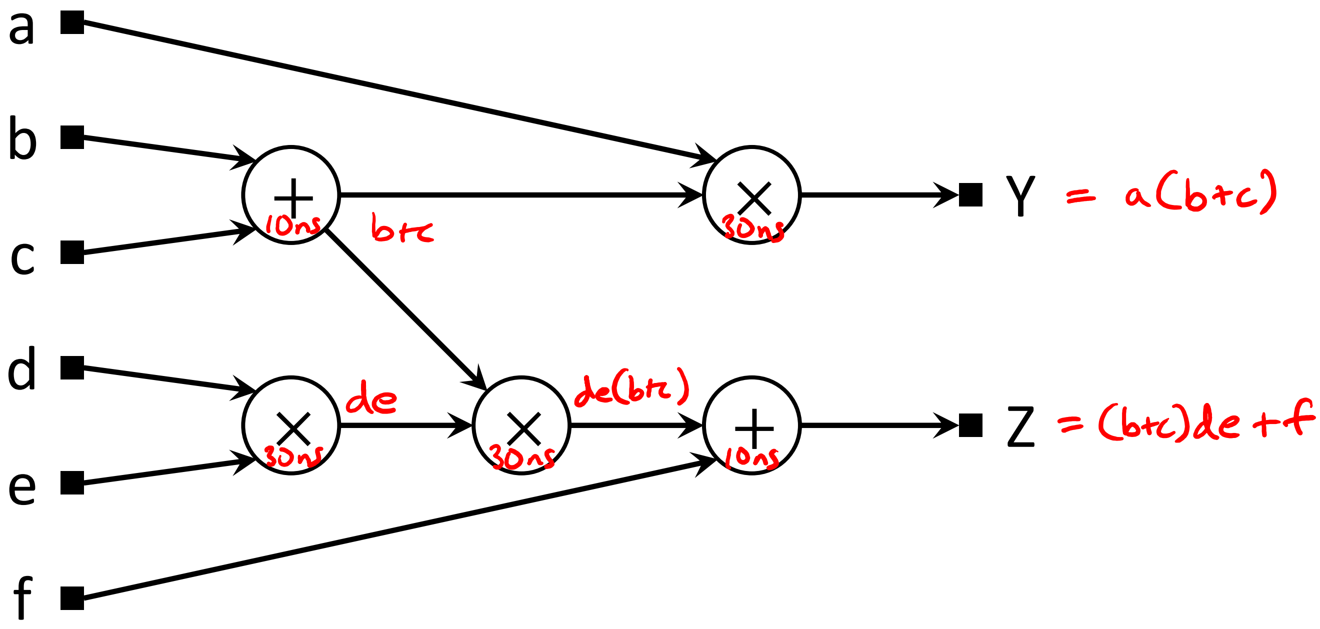
- Example: $y_i = (a_i \times b_i) \times c_i + d_i$
  - Signal flow:



Cycle	1	2	3	4
Reg11	X	$a_1 b_1$	$a_2 b_2$	$a_3 b_3$
Reg12	X	c_1	c_2	c_3
Reg13	X	d_1	d_2	d_3
Reg21	X	X	$a_1 b_1 c_1$	$a_2 b_2 c_2$
Reg22	X	X	d_1	d_2

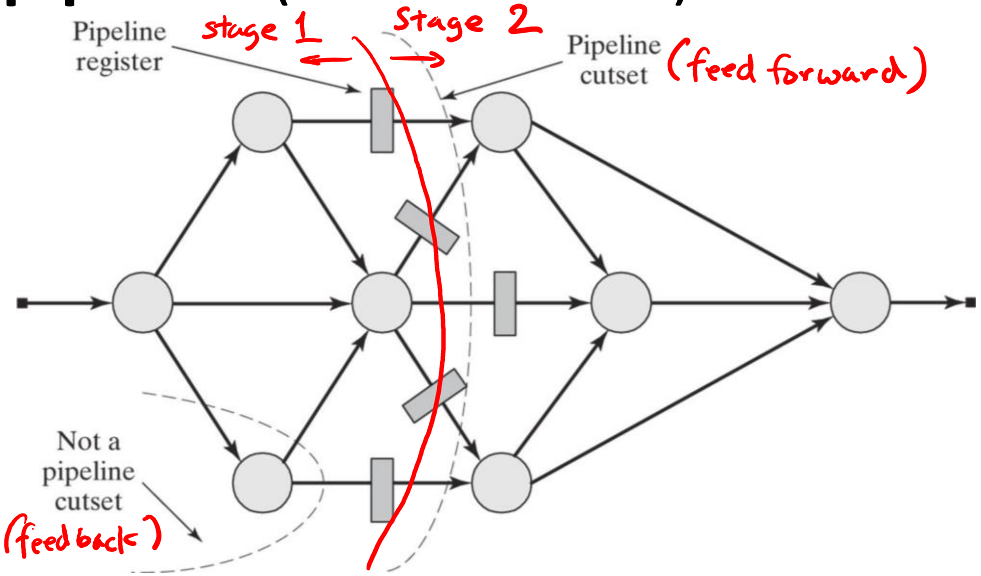
Data Flow Graph

- 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



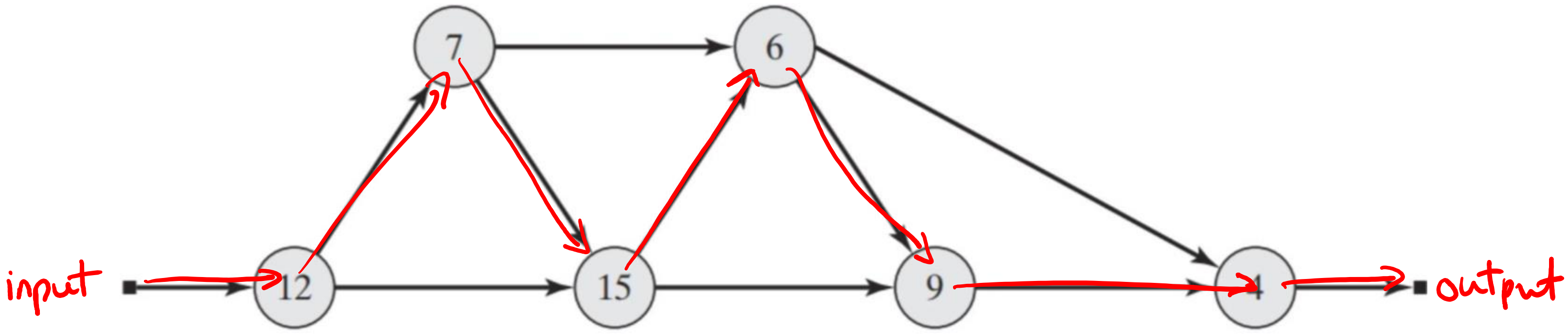
Pipeline Cutset

- 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:



critical path has delay of $12+7+15+6+9+4 = 53$

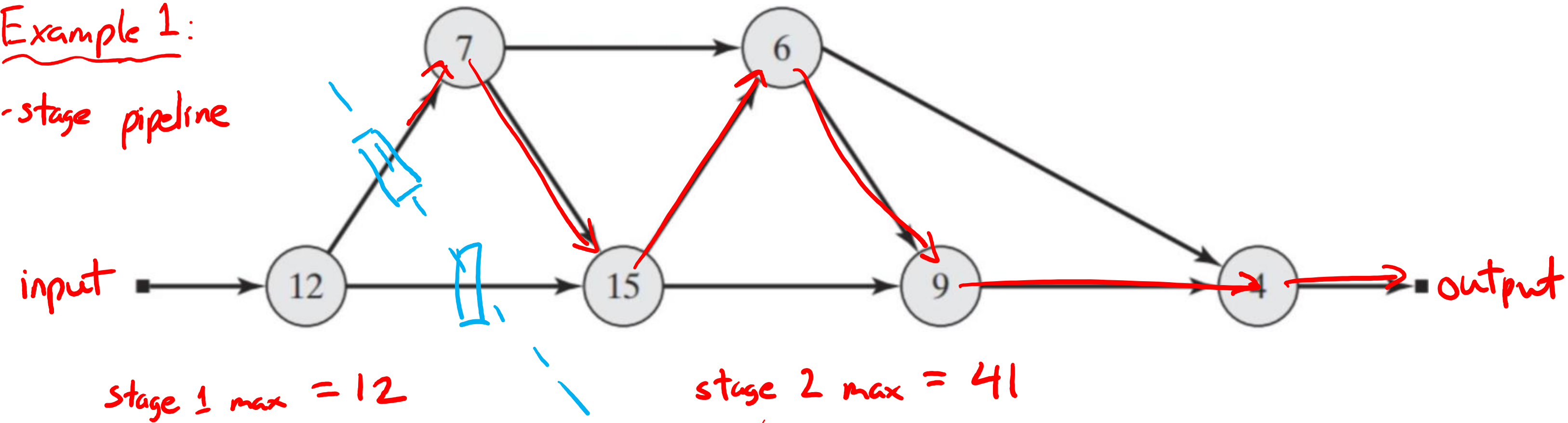
$\boxed{T_{min} = 53}$ $\xrightarrow{\times 1}$ latency > 53

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 $t_{CO} = 0$

Example 1:

2-stage pipeline



stage 1 max = 12

stage 2 max = 41

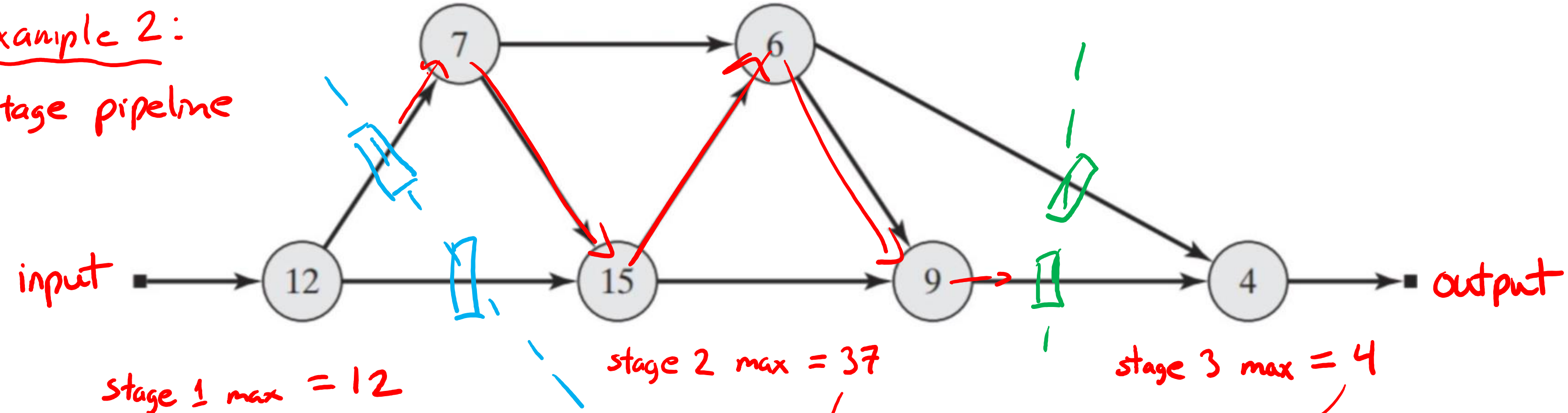
$\boxed{T_{min} = 41} \xrightarrow{\times 2}$ latency > 82

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 $t_{CO} = 0$

Example 2:

3-stage pipeline



stage 1 max = 12

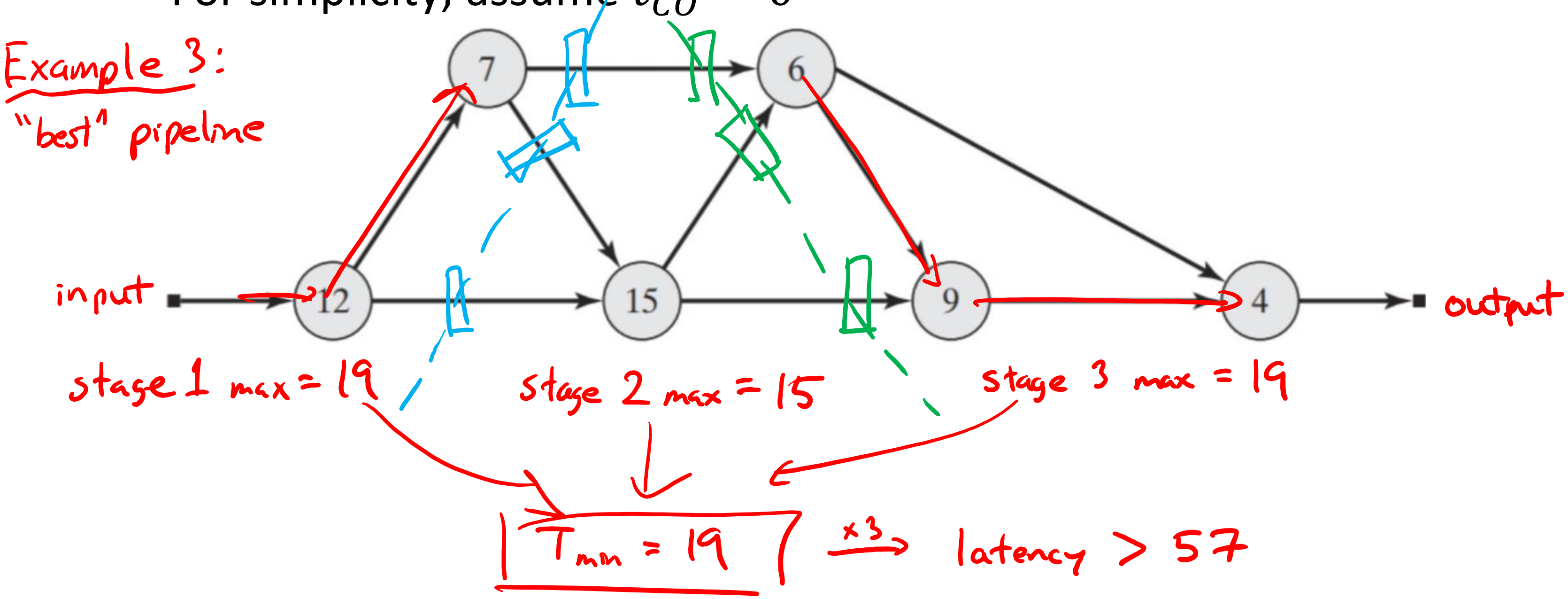
stage 2 max = 37

stage 3 max = 4

$\boxed{T_{min} = 37} \xrightarrow{x3}$ latency > 111

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 $t_{CO} = 0$



Example 3:
"best" pipeline

stage 1 max = 19
stage 2 max = 15
stage 3 max = 19

$\boxed{T_{min} = 19}$ $\xrightarrow{\times 3}$ latency > 57

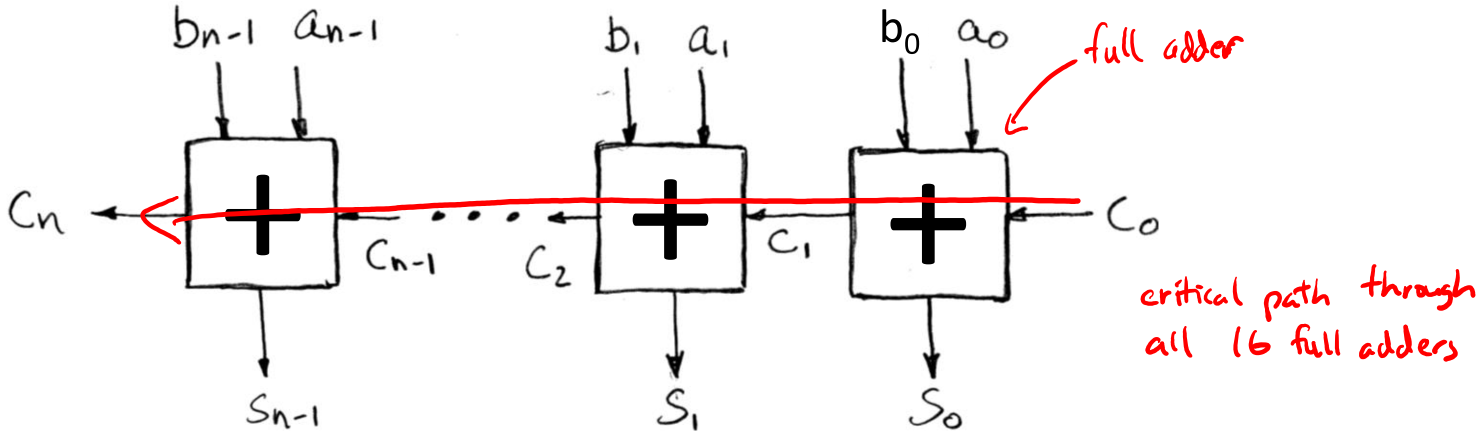
Pipeline Design Questions

- 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

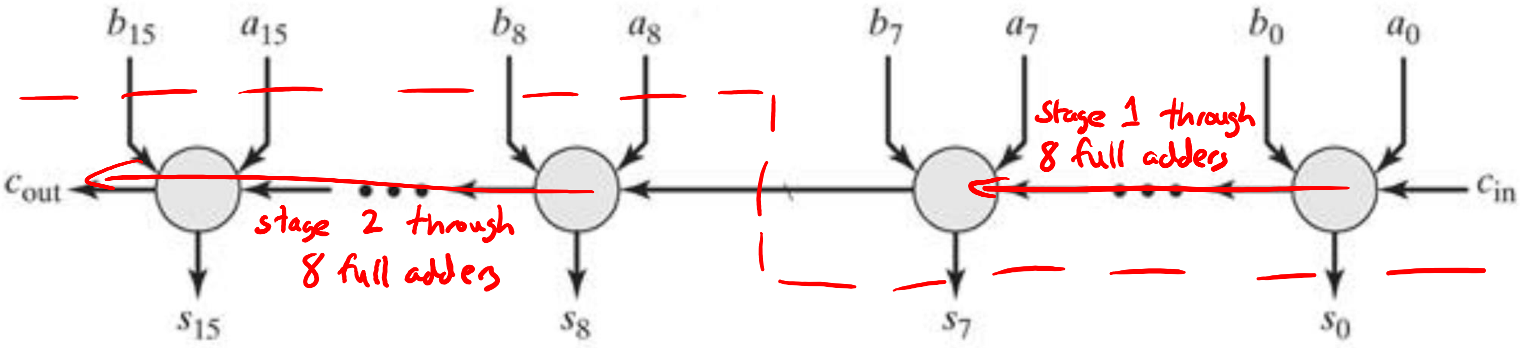
- Where do the pipeline registers go?
  - Must be placed at proper pipeline cutsets
  - Want to make pipeline stages as balanced as possible to maximize speedup

Design Example: 16-bit Ripple-Carry Adder

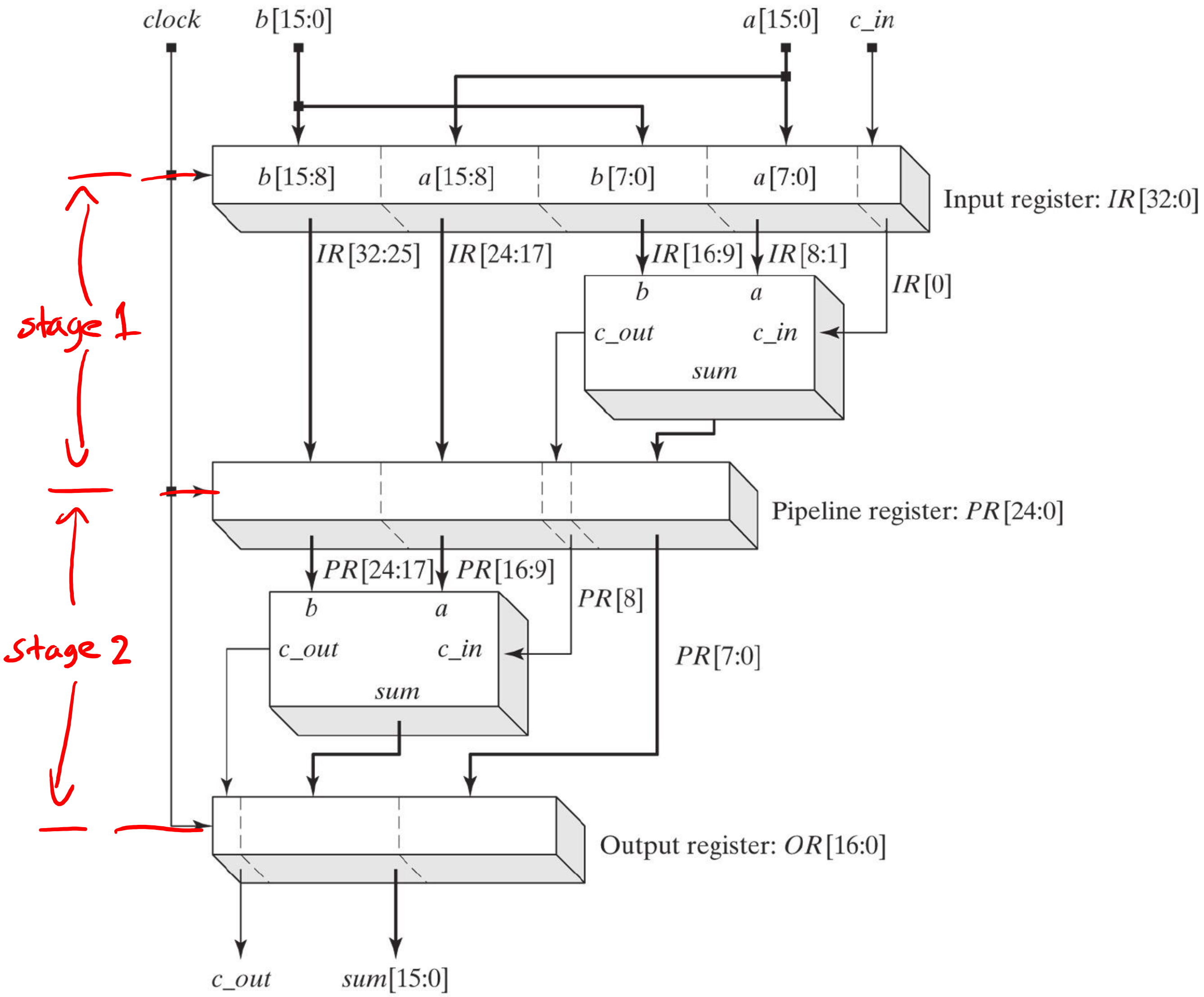
- Problem: C_n takes a long time to compute!



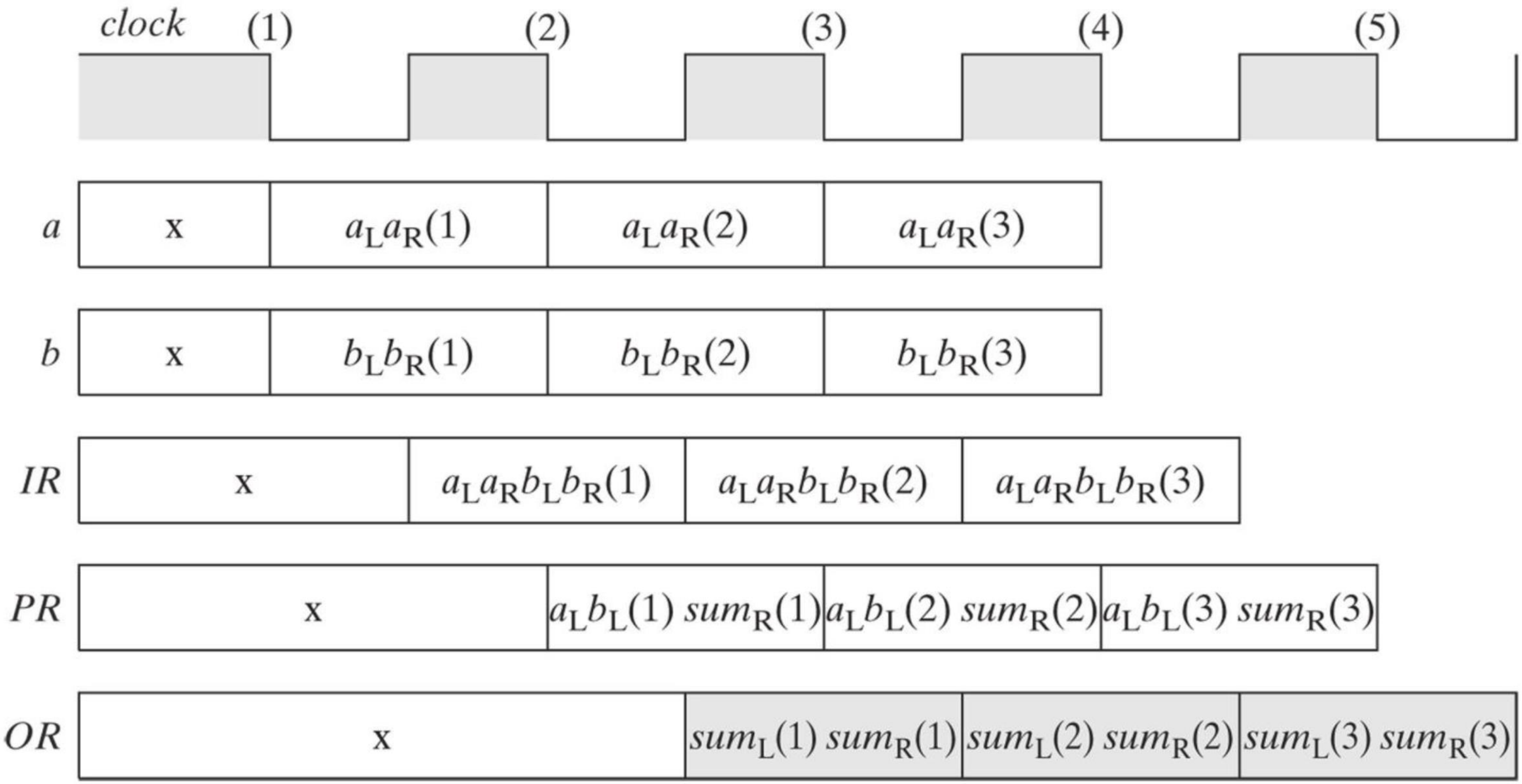
- 2-stage pipeline: which cutset to use? cut evenly in half



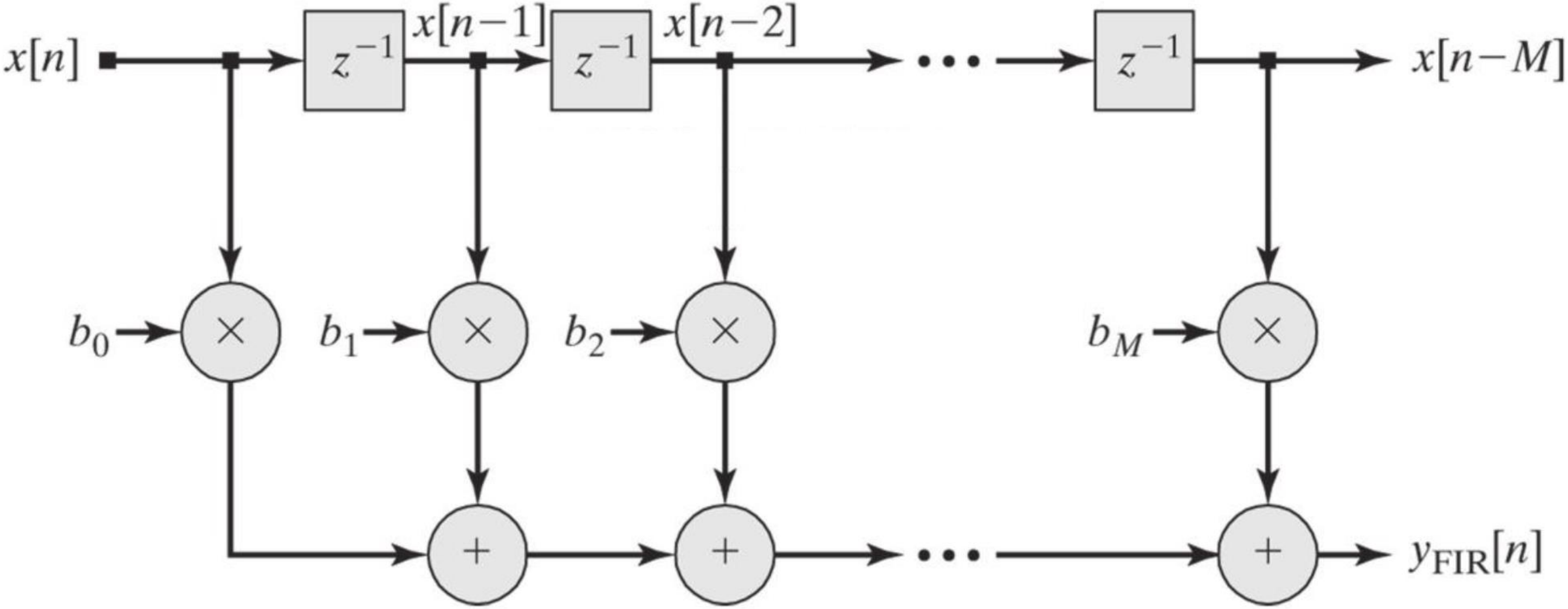
Design Example: 16-bit Pipelined Adder



Design Example: 16-bit Pipelined Adder

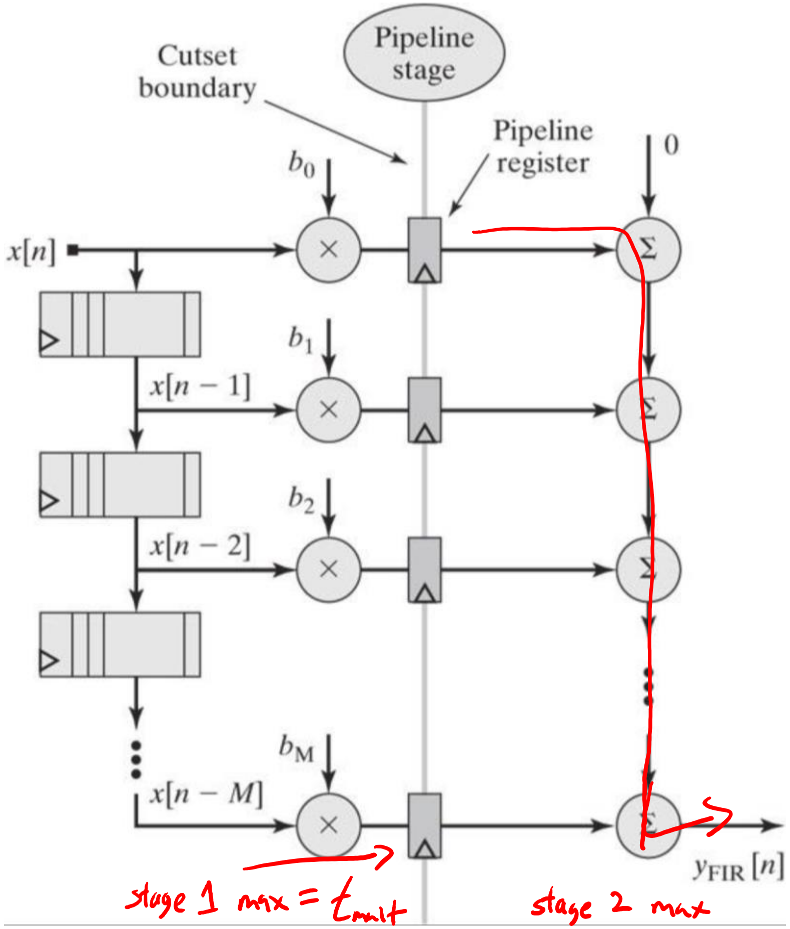


Design Example: FIR Filter



Design Example: Pipelined FIR Filter

2-stage pipeline

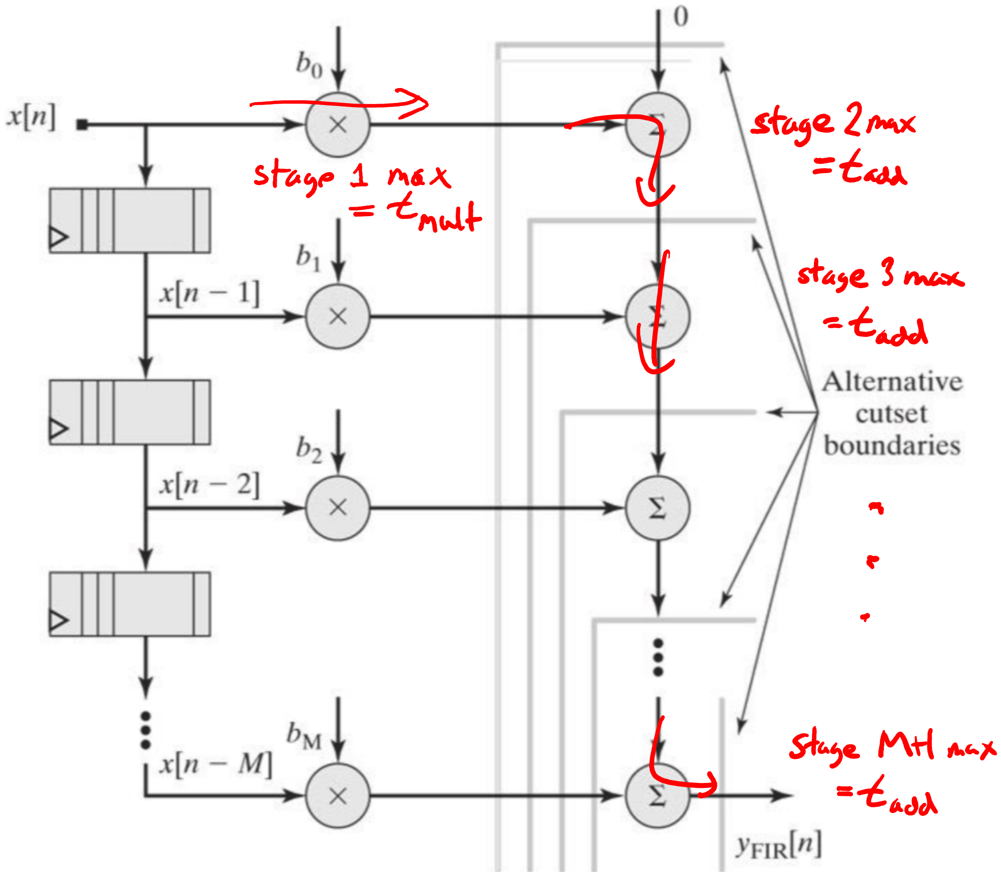


stage 1 max = t_{mult}

stage 2 max = $M \times t_{add}$

possibly unbalanced!

M+1-stage pipeline



stage 1 max = t_{mult}

stage 2 max = t_{add}

stage 3 max = t_{add}

⋮

stage M+1 max = t_{add}

maybe more balanced, but a lot more latency!