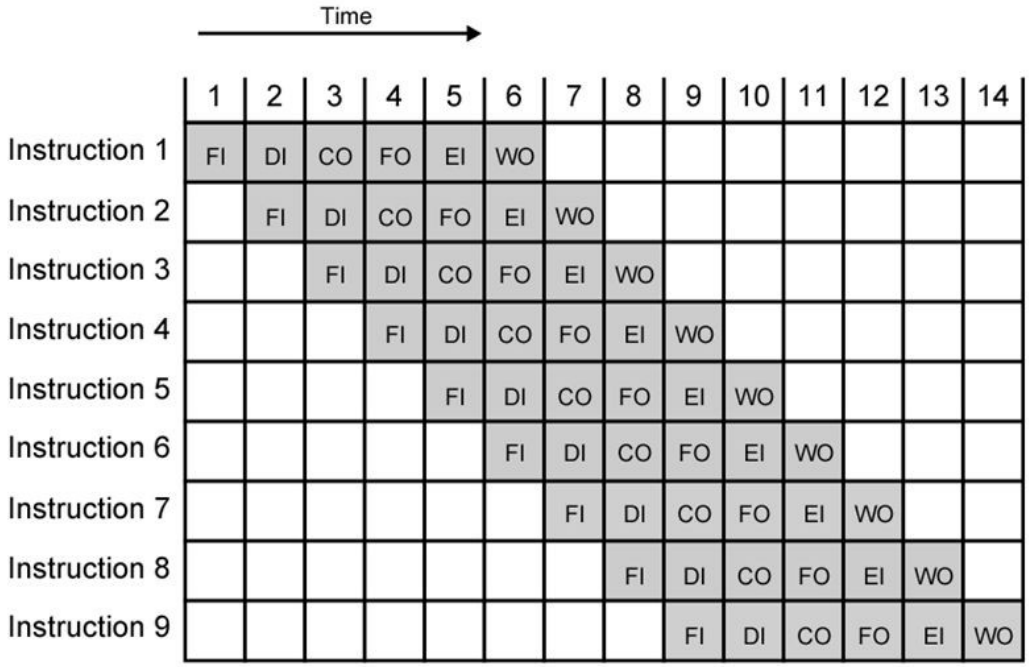
Timing Diagram for Instruction Pipeline Operation



Lecture 25-26 Pipeline Practice
(one question will be included in final exam)

Pipeline

- Pipelines used in many designs (DSP, image processing, communication, CPUs, encryption) when there is a continuous stream of data or information to be processed

- Split a complex logic function into multiple simpler stages
  - First pipeline stage starts processing new input while rest of pipeline is working on previous data

  - Advantage: provide for higher throughput and performance

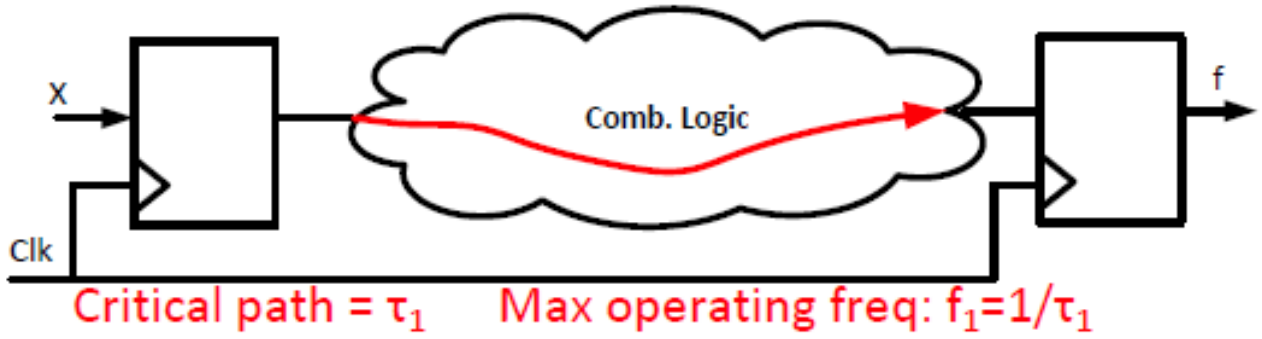
  - Disadvantage: more logic to implement and greater latency

  - Mix of number of stages and amount of combinational logic needs to be balanced

Pipeline Lessons

- Pipeline does not help **latency** of single task, it helps **throughput** of entire workload
- **Multiple** tasks operating simultaneously using different resources
- Pipeline speedup = **number pipe stages**
- Pipeline rate limited by **slowest** pipeline stage
- Unbalanced lengths of pipe stages reduces speedup

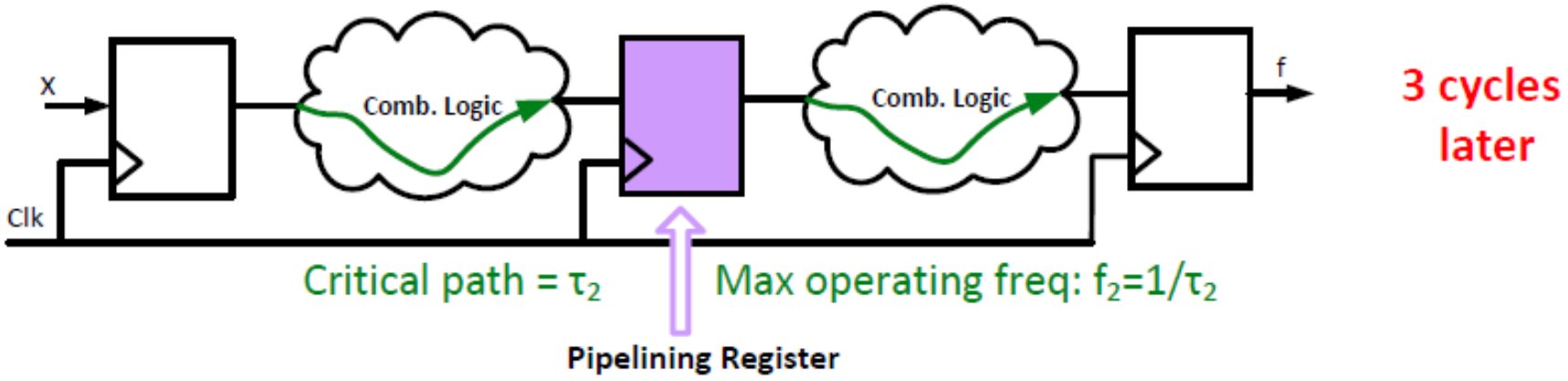
- Original System (critical path = τ_1, max operating frequency =$1/\tau_1$)



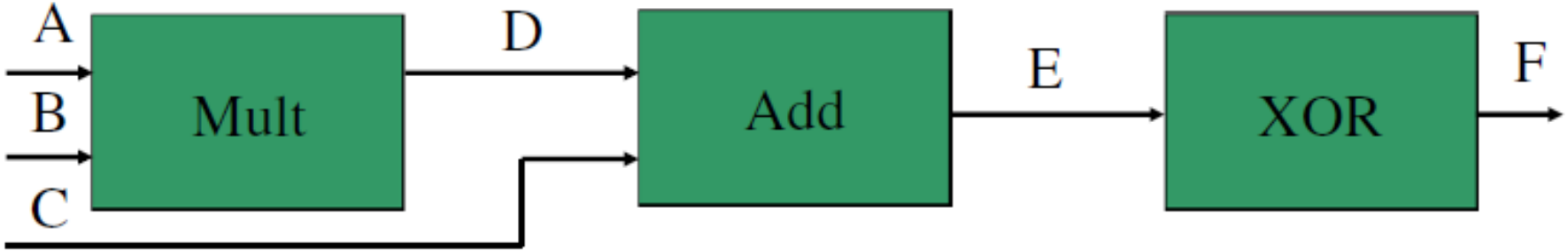
2 cycles later

Pipeline Lessons

- Pipelined version (critical path = τ_2, max operating frequency =$1/\tau_2$)
  - Smaller critical path -> higher throughput ($\tau_{2<}\ \tau_1$) -> ($f_1 > f_2$)
  - Longer latency



Exercise 1




```
module pipeline1 (a, b, c, f);
input [7:0] a, b, c;
output [15:0] f;
//assign f=((a*b)+c)^16`h1234

Assign d = a*b;
Assign e = d + c;
Assign f = e ^ 16`h1234;
endmodule
```

11.8ns combinational path

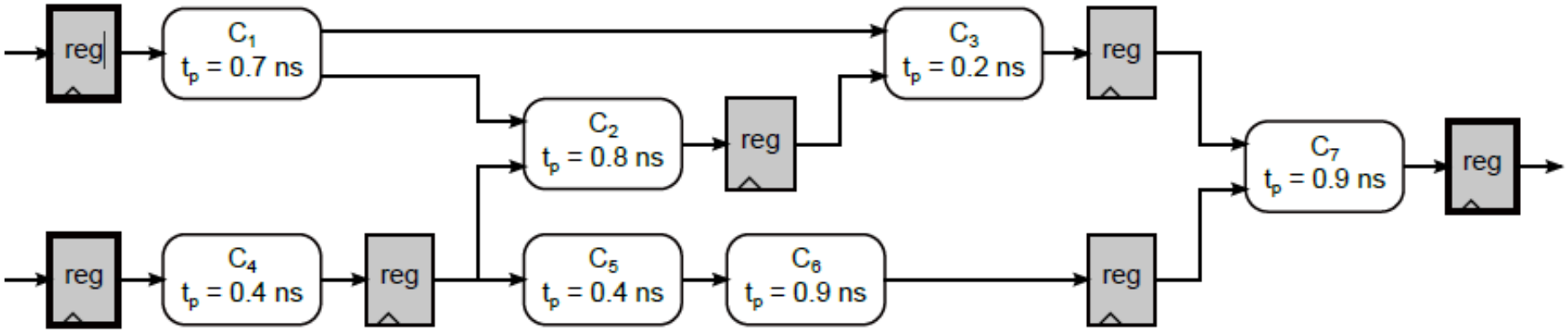
Exercise 1


```
module pipeline1 (clk, a, b, c, f);
input [7:0] a, b, c; input clk;
output reg [15:0] f;
reg [15:0] d, e; reg [7:0] c_d;
always @ (posedge clk)
        begin
         c_d <= c;
           d <= a * b;
           e <= d + c_d;
            f <= e ^ 16`h1234;
end
endmodule
```

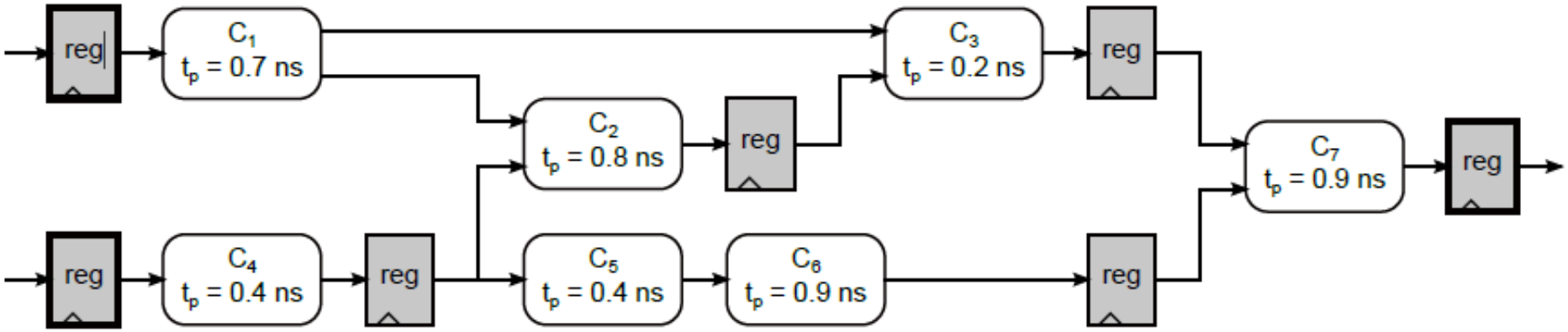
56 flip-flops
> 300MHz operation

Exercise 2

- Consider the digital circuit shown below. All registers are edge-triggered, and for simplicity they can be considered ideal (i.e., propagation delay, setup and hold times are all zero). The propagation delays of each combinational block are reported in the diagram.
- Question1: Calculate the minimum clock period at which the circuit operates correctly.

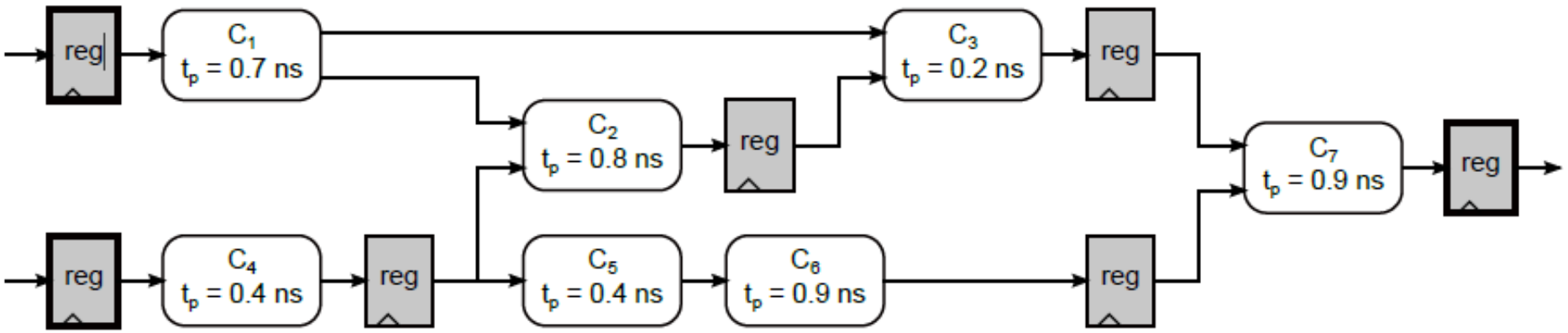


Exercise 2

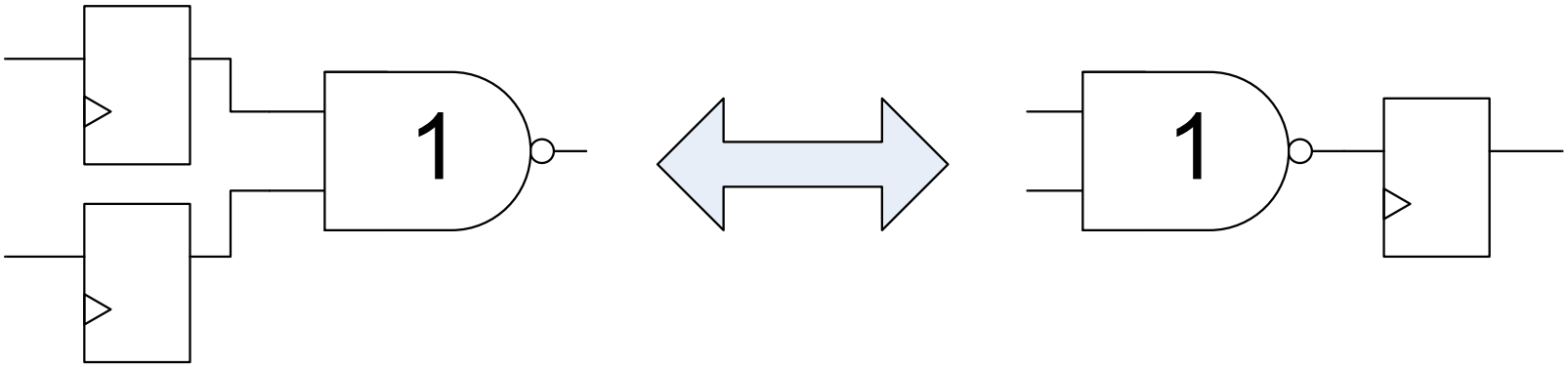


- Answer: minimum clock period determined by the maximum path delay: $C_1 + C_2 = 1.5$ns

Exercise 2

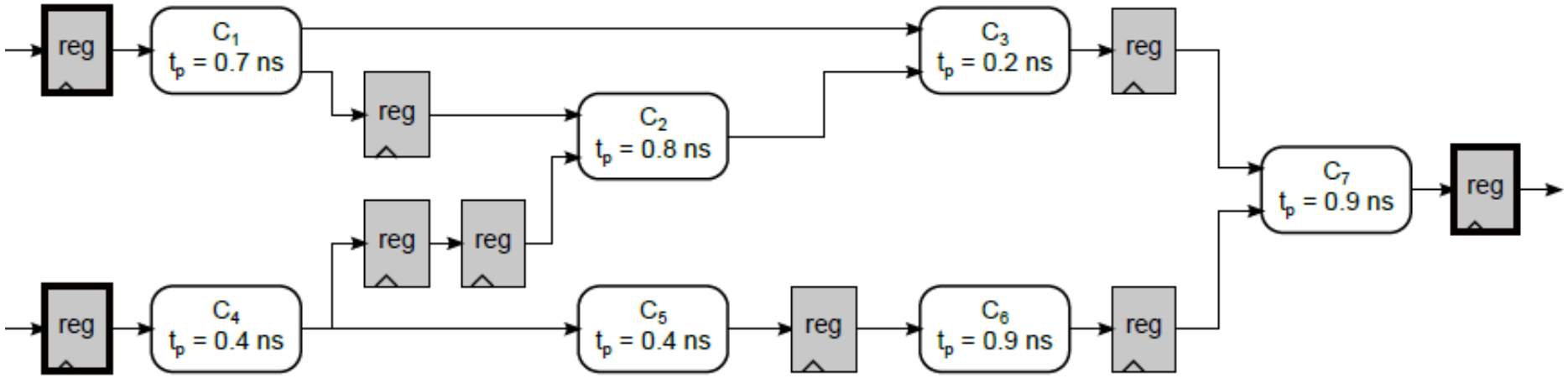


- Question2: apply the retiming method to the circuit, so that the clock period can be reduced to 1ns. The I/O registers (marked with bold edges) cannot be touched.
- Remember retiming is below



Exercise 2

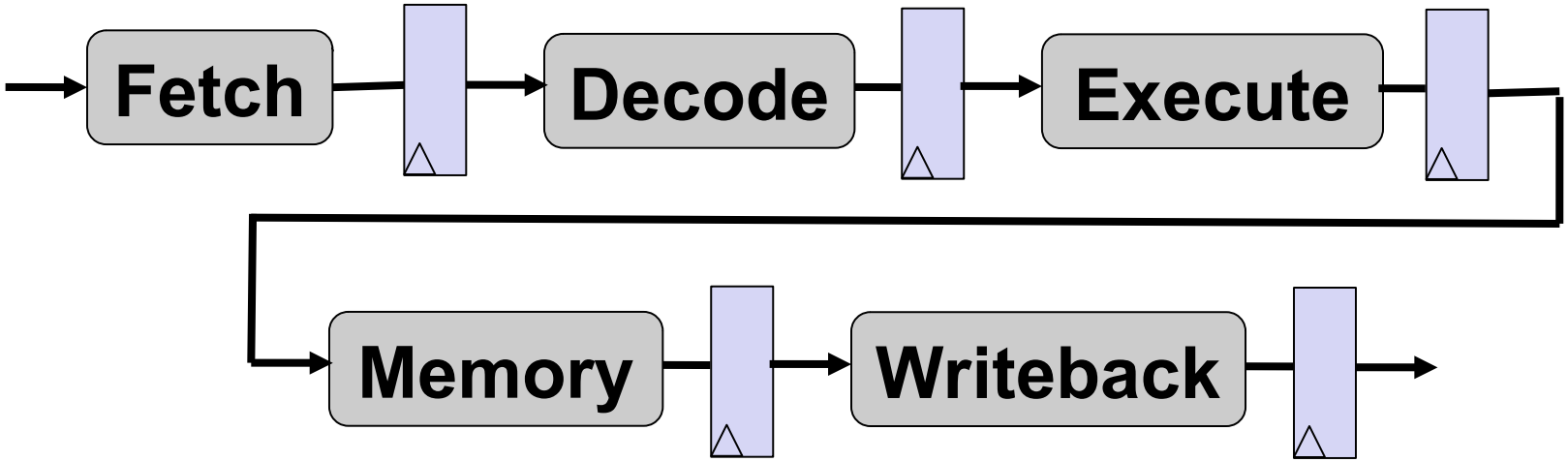
- Answer: reorganizing the circuit pipelines to balance the path delays and minimize required clock cycle, while sustaining the correct operation and timing constraints of the circuit.



Exercise 3

- A 5-stage processor has the following latencies. Assume that when pipelining, each pipeline stage costs 20ps extra for the registers between pipeline stages.

Fetch	Decode	Execute	Memory	Writeback
300ps	400ps	350ps	500ps	100ps



- Question1: what is the cycle time if no pipeline is used? What is the cycle time of pipeline is used?

Exercise 3

Fetch	Decode	Execute	Memory	Writeback
300ps	400ps	350ps	500ps	100ps

- Answer:

when there is no pipelining, so the cycle-time has to allow an instruction to go through all the stages each cycle. Therefore, CT = 300+400+350+500+100 = 1650ps.

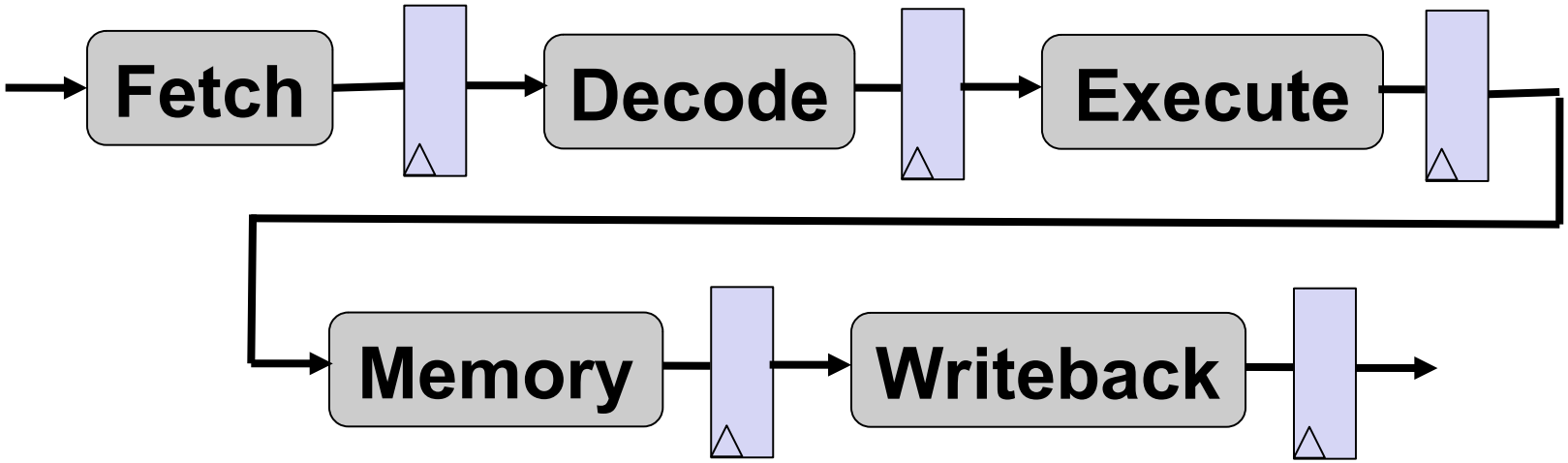
Pipeline to 5 stages reduces the cycle time to the length of the longest stage. Additionally, the cycle time needs to be slightly longer to accommodate the register at the end of the stage.

CT = 500 + 20 = 520ps

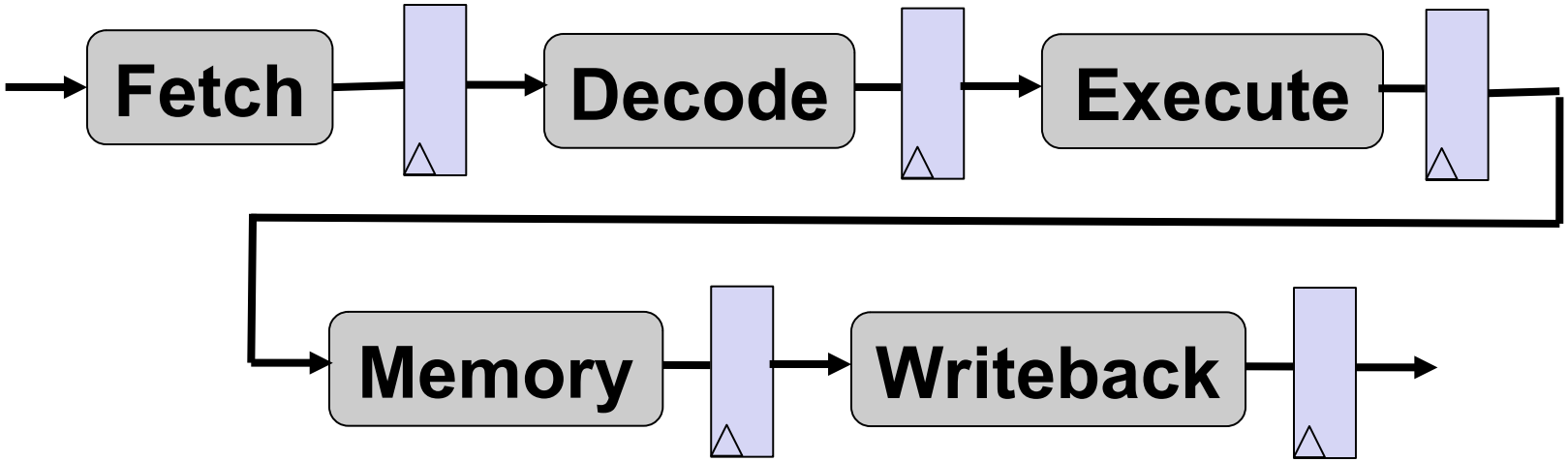
Exercise 3

Fetch	Decode	Execute	Memory	Writeback
300ps	400ps	350ps	500ps	100ps

- Question2: what is the latency of an instruction in non-pipeline case and pipeline case, respectively?



Exercise 3



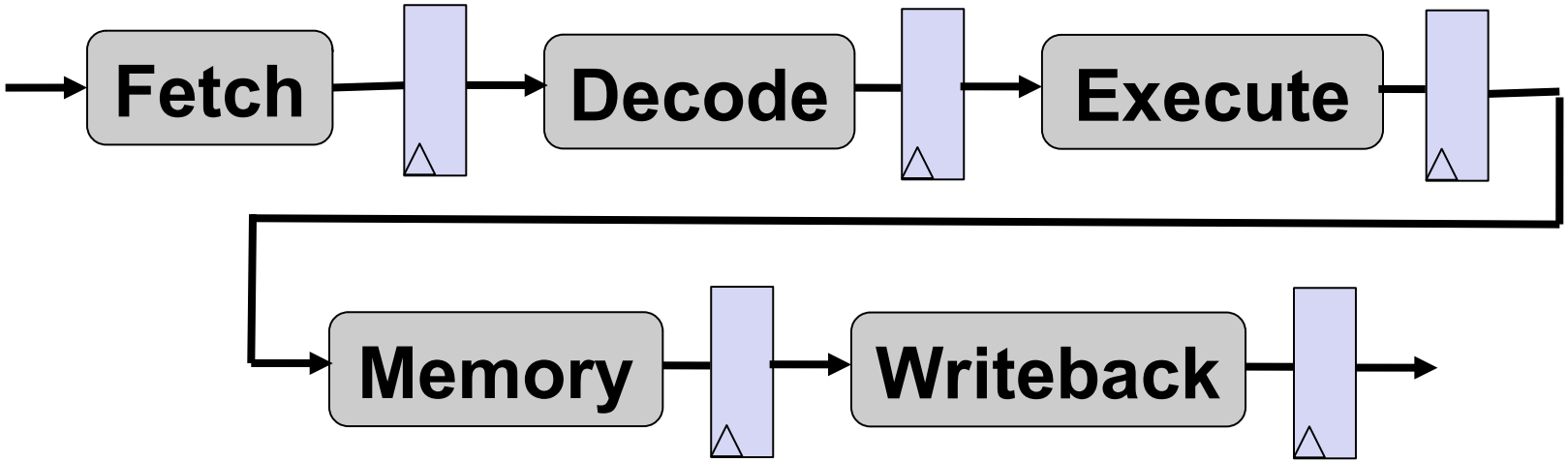
Fetch	Decode	Execute	Memory	Writeback
300ps	400ps	350ps	500ps	100ps

- Answer: the latency for an instruction in a non-pipelined design is the same as the cycle time (i.e., 1650ps), since each instruction takes an instruction to go from beginning fetch to the end of writeback.
- In a pipelined design, the latency is 5 * (cycle time), since an instruction needs to go through 5 pipeline stages, spending 1 cycle

Exercise 3

Fetch	Decode	Execute	Memory	Writeback
300ps	400ps	350ps	500ps	100ps

- Question3: If you could split one the pipeline stages into 2 equal halves, which one would you choose? What is the new cycle time? What is the new latency?



Exercise 3

Fetch	Decode	Execute	Memory	Writeback
300ps	400ps	350ps	500ps	100ps

- Answer: Splitting the longest stage is the only way to reduce the cycle time. After splitting it, the new cycle time is based on the new longest stage

Old longest stage is Memory. New cycle time = 400 + 20 = 420ps

The new latency is 6 * (cycle time), since an instruction needs to go through 6 pipeline stages now

Exercise 4

- Use the following delay parameters for this part of the question:

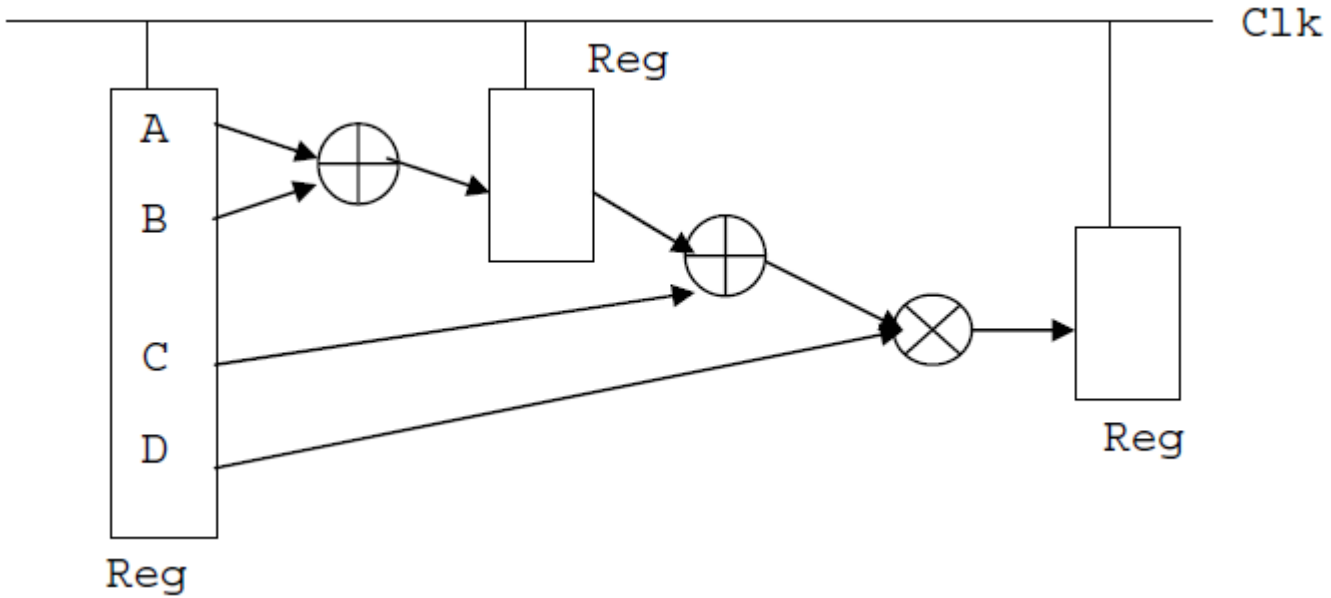
Critical path delay of adder: 3ns

Critical path delay of multiplier: 5ns

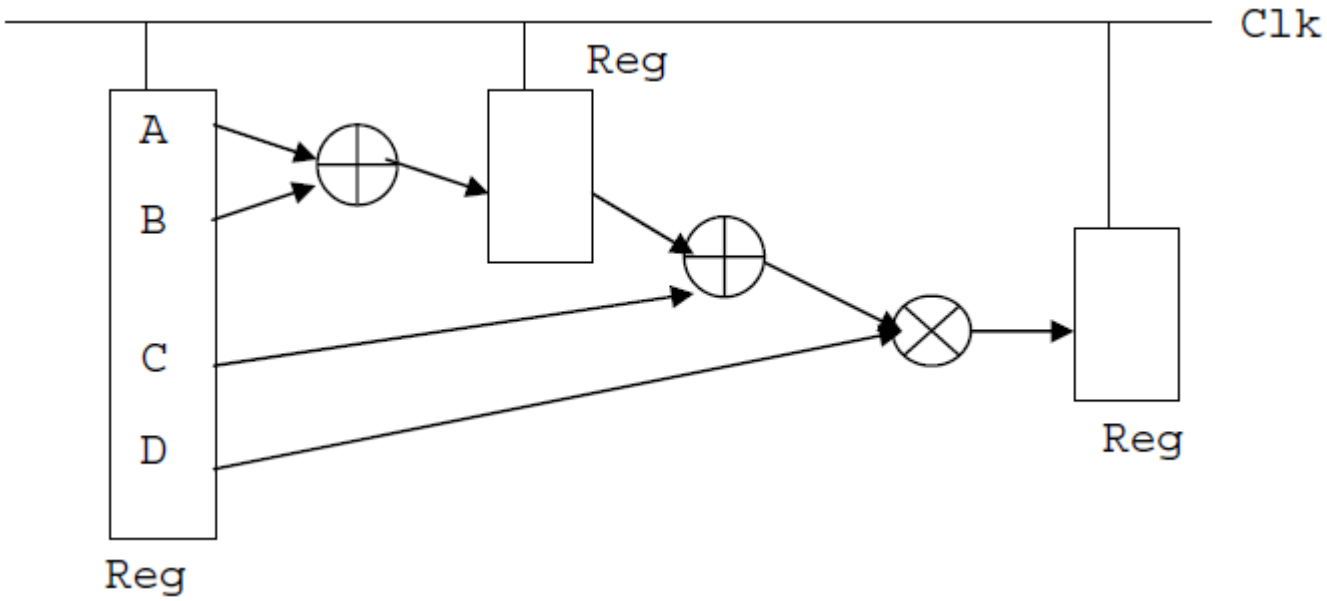
Clock to Q delay time of register: 0.5ns

Setup time of register: 0.5ns

- Question1: Explain why the pipeline strategy in the circuit is not optimized? Support your answer by calculating the minimum clock cycle time.



Exercise 4



- Answer: the pipeline strategy is not optimized because the delay between the registers is not balanced. The minimum clock cycle time is hence limited by the slowest block. For this example:

The minimum clock cycle time = 0.5 + 3 + 5 + 0.5 = 9ns

Latency = 2 * 9ns = 18ns

Exercise 4

- Use the following delay parameters for this part of the question:

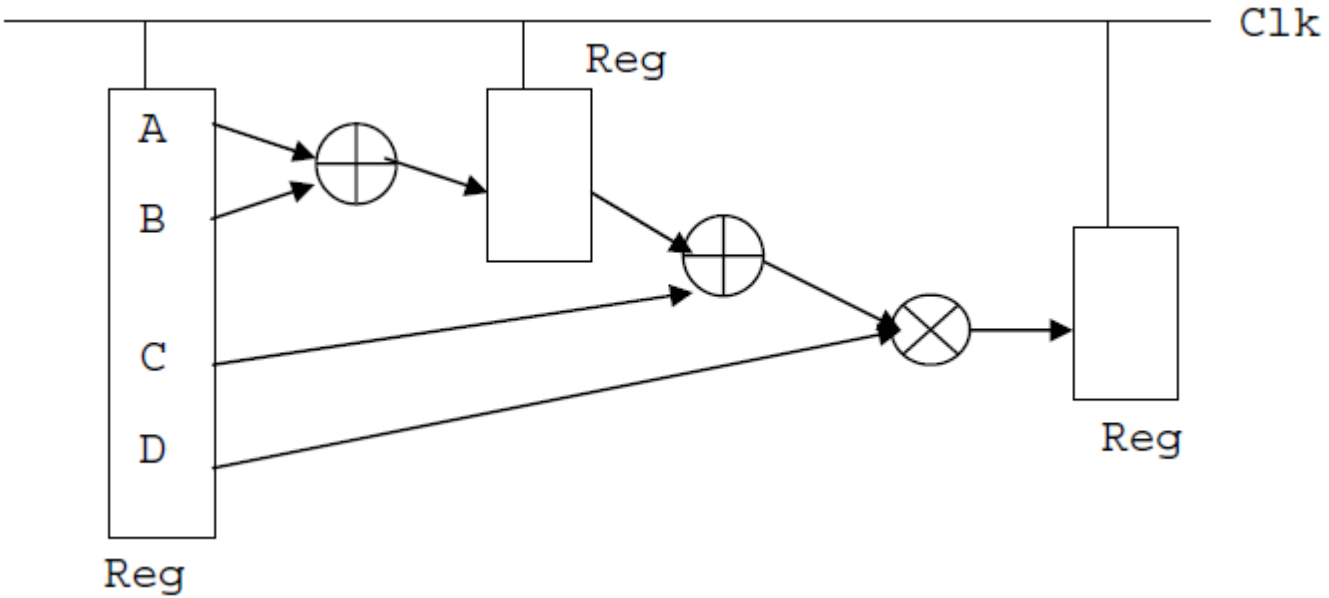
Critical path delay of adder: 3ns

Critical path delay of multiplier: 5ns

Clock to Q delay time of register: 0.5ns

Setup time of register: 0.5ns

- Question2: Propose an improvement for the circuit without adding any additional hardware. What is the minimum clock cycle time of your new circuit?



Exercise 4

- Use the following delay parameters for this part of the question:

Critical path delay of adder: 3ns

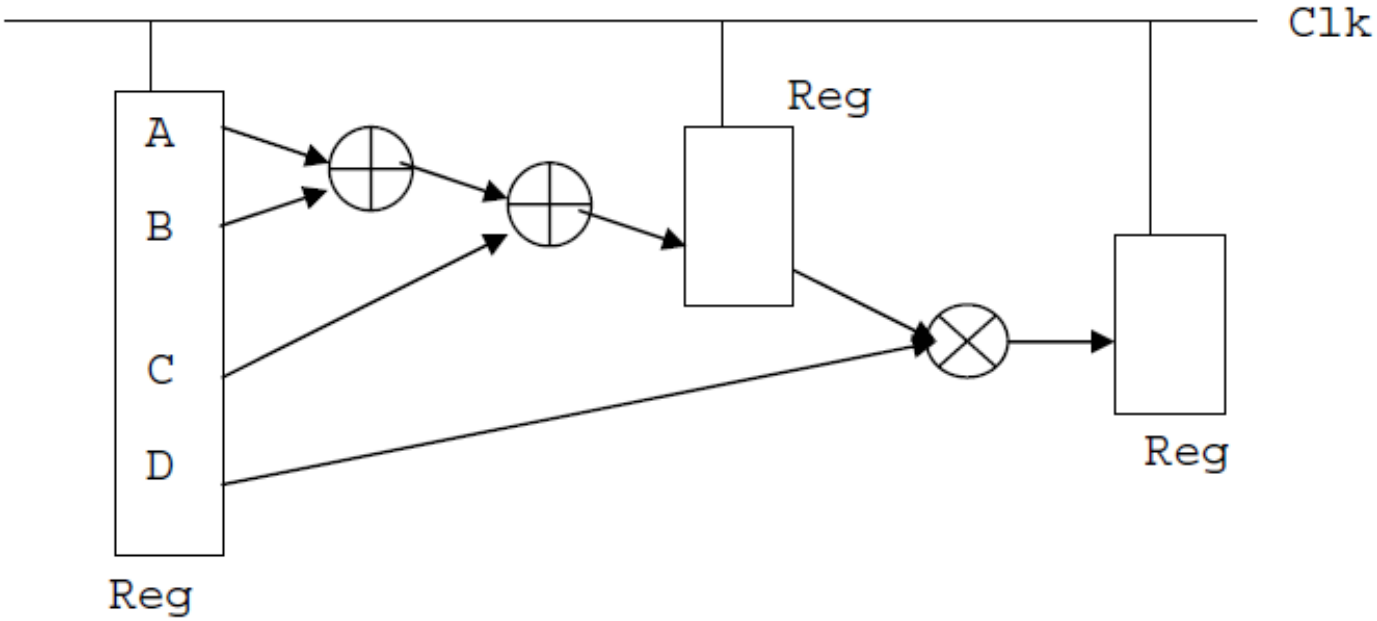
Critical path delay of multiplier: 5ns

Clock to Q delay time of register: 0.5ns

Setup time of register: 0.5ns

- Answer: The minimum clock cycle time = 0.5 + 3 + 3 + 0.5 = 7ns

Latency = 2 * 7ns = 14ns



Exercise 4

- Use the following delay parameters for this part of the question:

Critical path delay of adder: 3ns

Critical path delay of multiplier: 5ns

Clock to Q delay time of register: 0.5ns

Setup time of register: 0.5ns

- Question3: Design a non-pipeline data path to do the following calculation:

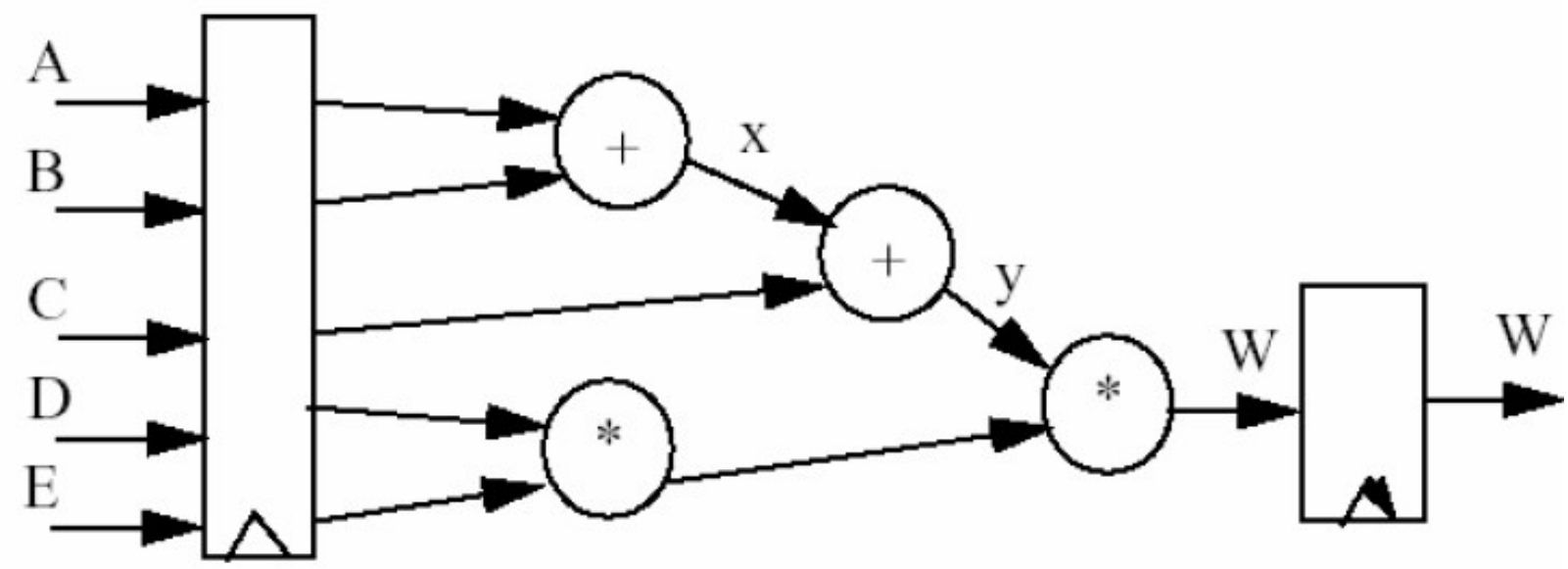
$$x = A + B \qquad y = x + C$$

$$z = y \times D \qquad w = z \times E$$

A, B, C, D, E are the primary inputs and are supposed to store in a register before sending to the data path. W is the primary output and it is stored to a register after the computation. Find the minimum clock cycle time, the throughput and latency of the data path.

Exercise 4

Answer:



The minimum clock cycle time = 0.5 + 3 + 3 + 5 + 0.5 = 12ns
Since there is no pipeline stage, the latency is 12ns

Exercise 4

- Use the following delay parameters for this part of the question:

Critical path delay of adder: 3ns

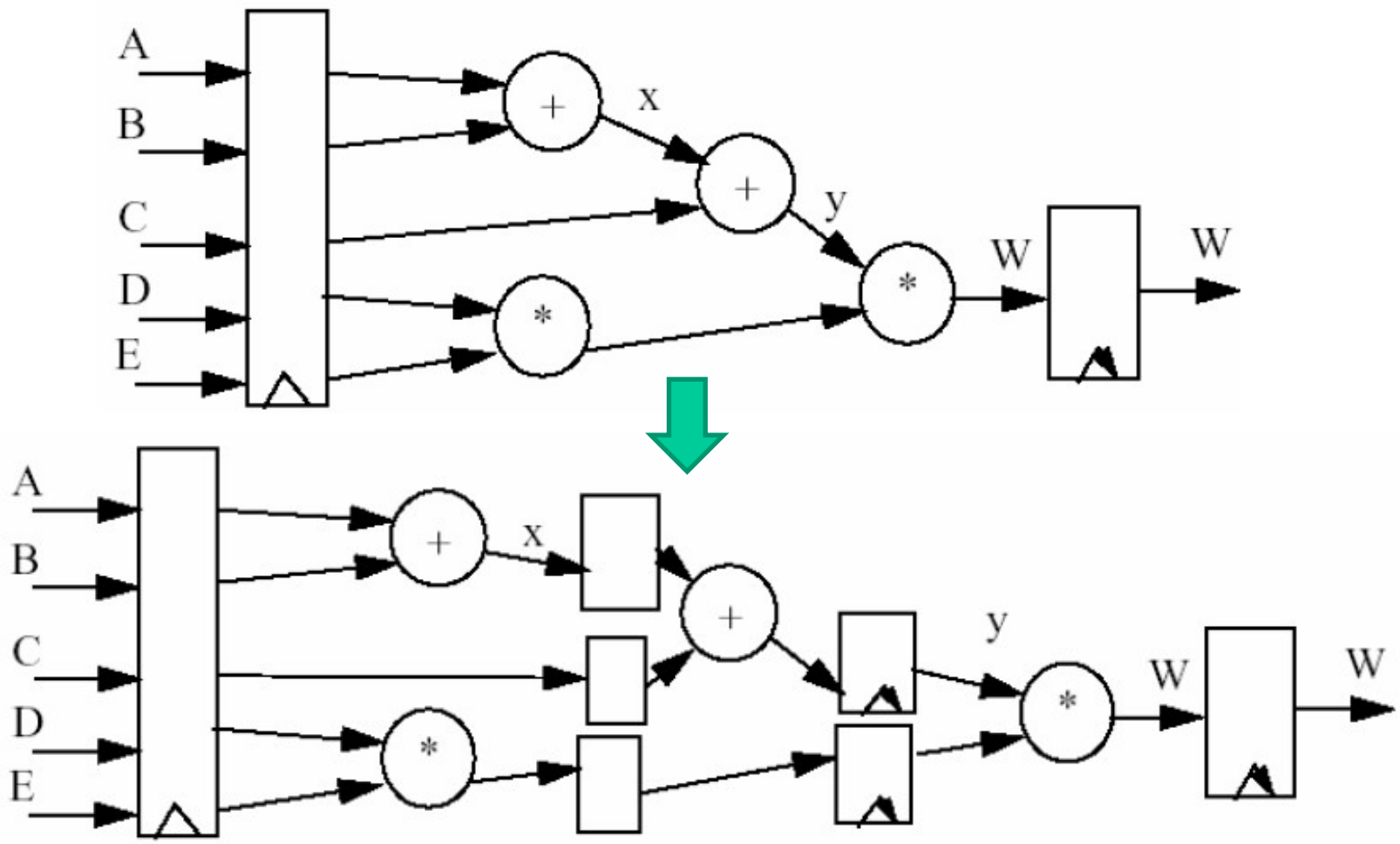
Critical path delay of multiplier: 5ns

Clock to Q delay time of register: 0.5ns

Setup time of register: 0.5ns

- Question4: Redesign the data path of question with pipeline technique to obtain the optimal performance of the system. What are the new clock cycle time and the latency of the system? How many extra registers are required?

Exercise 4

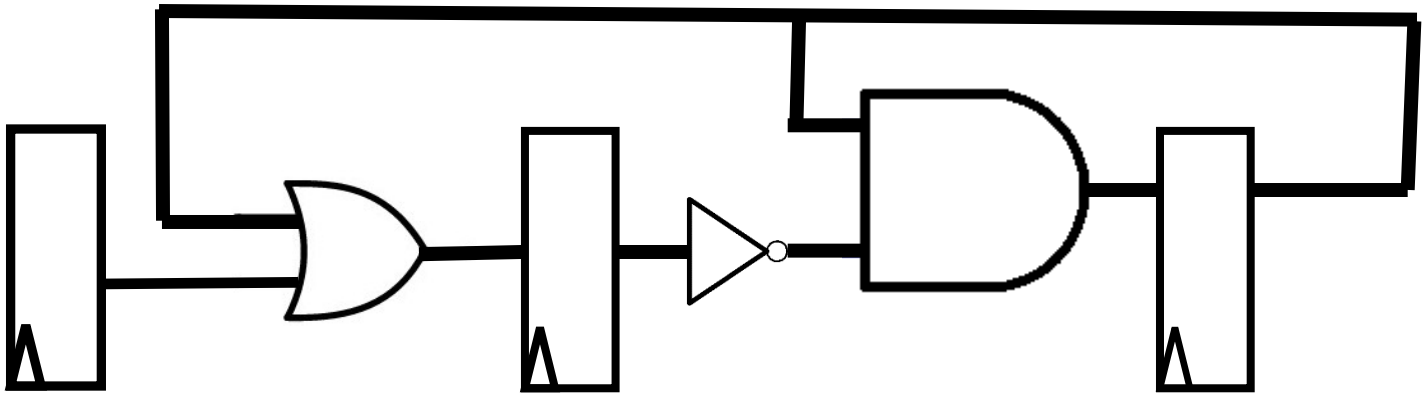


- The minimum clock cycle time = 0.5 + 5 + 0.5 = 6ns

Since there is 3 pipeline stages, the latency = 2 * 6 = 18ns and there are 5 extra registers.

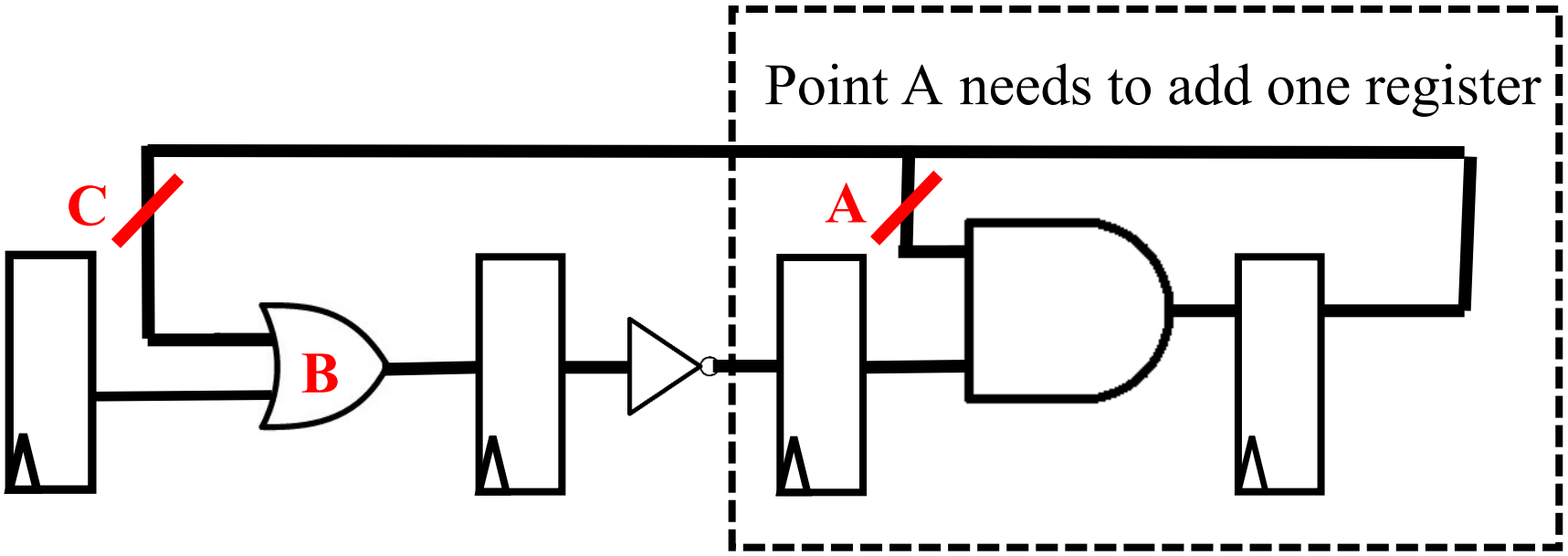
Exercise 5

- Ben and Allen are arguing over whether the performance of the following circuit can be improved with additional pipeline. Ben argues that it is possible to improve the performance by adding another register between the AND gate and inverter. Allen argues otherwise and claims that the structure of the circuit does not allow further pipelining. Who is correct? Briefly justify your answer.



Exercise 5

- If we put another register between the AND gate and inverter, what is the consequence?

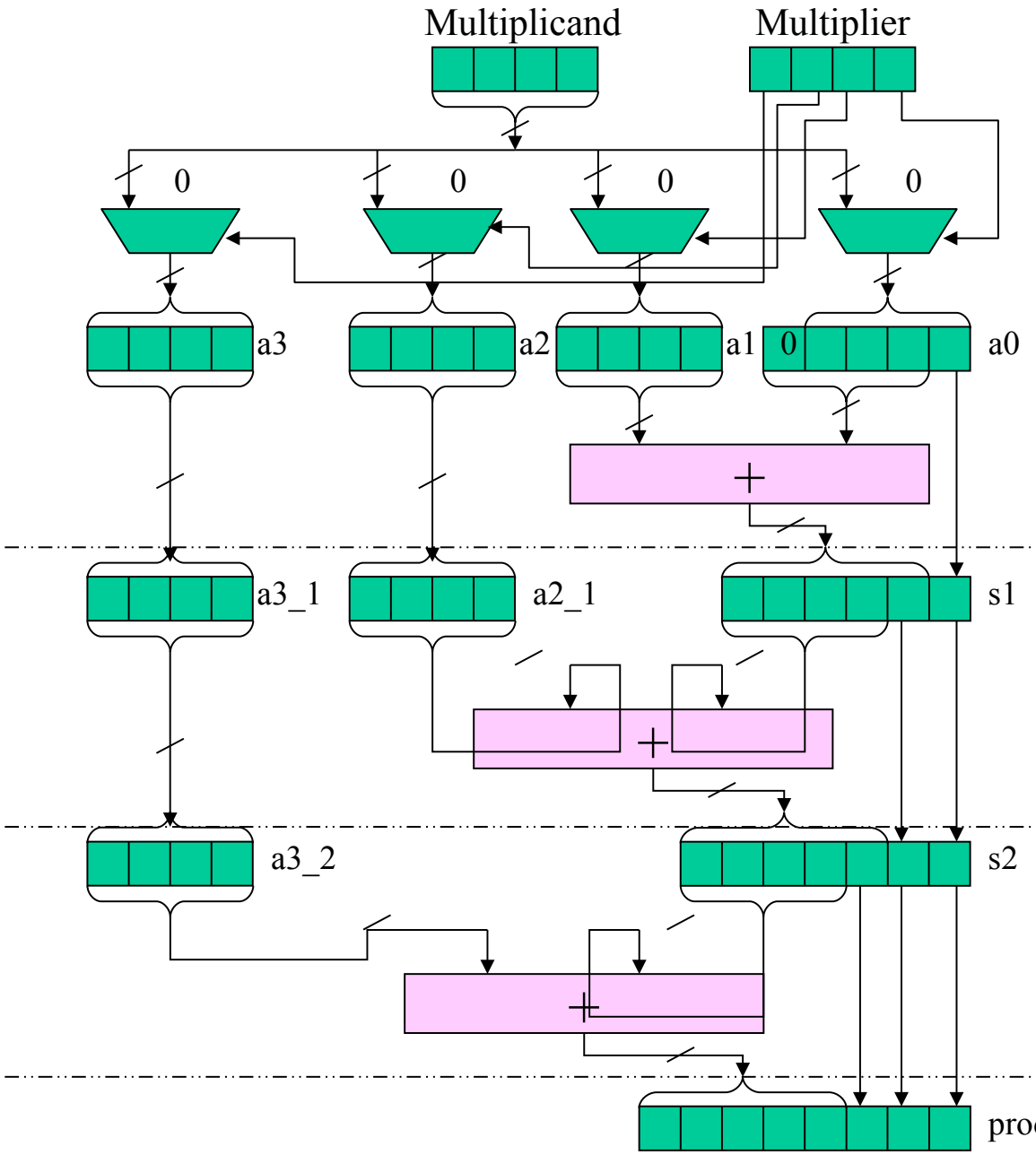


But function of B is wrong, even though register is placed at point C

So this circuit does not allow further pipelining

Exercise 6

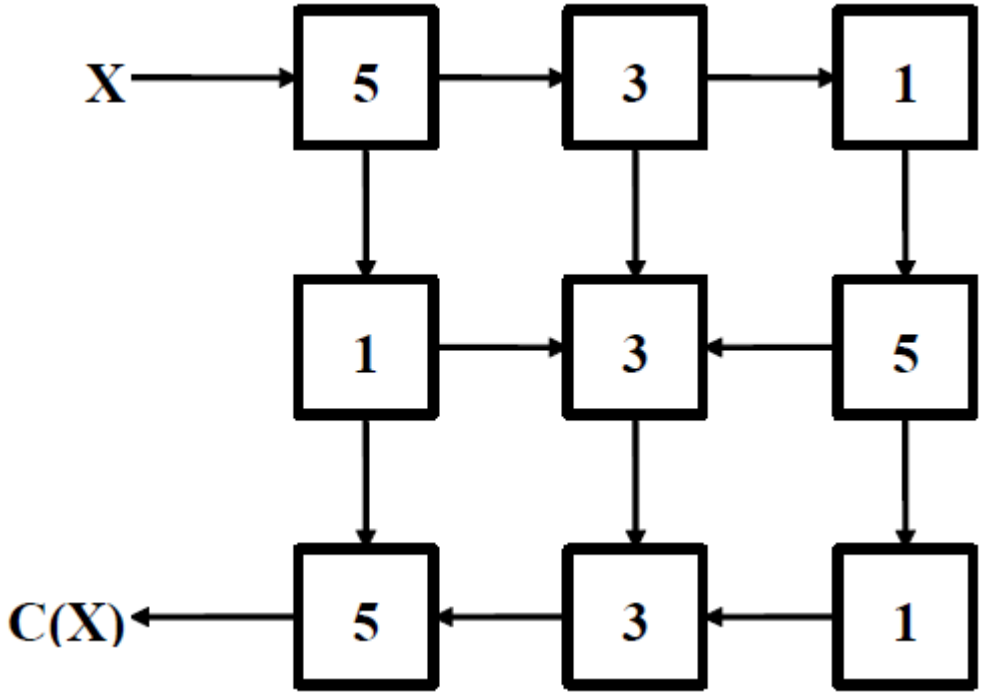
- Assume you have a full-adder component given to you. Suppose you want to pipeline a 4-bit multiplier, so it would be roughly 3X as fast. Draw the solution for your pipelined multiplier.




```
      1 0 1 1
    x 1 1 0 1
  -------------
      1 0 1 1
    0 0 0 0
  1 0 1 1
1 0 1 1
-------------------
1 0 0 0 1 1 1 1
```

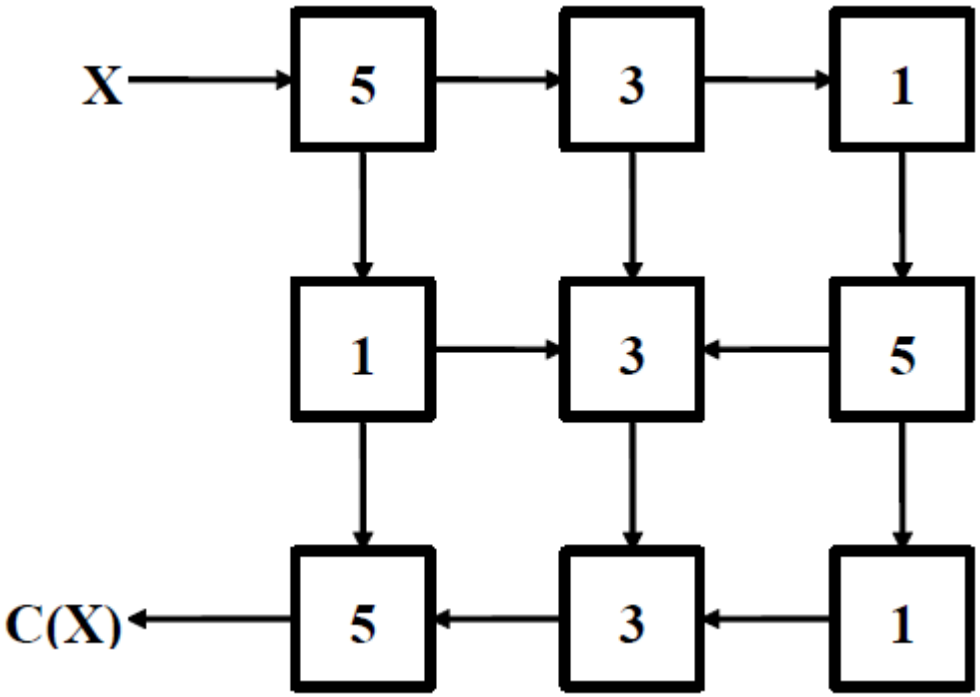
Exercise 7

- There is a combinational encryption device constructed of nine modules as follows. This device takes an integer value X and computes an encrypted version C(x). Each component is marked with its propagation delay in μs.



- Question1: what is the latency and throughput of this device?

Exercise 7

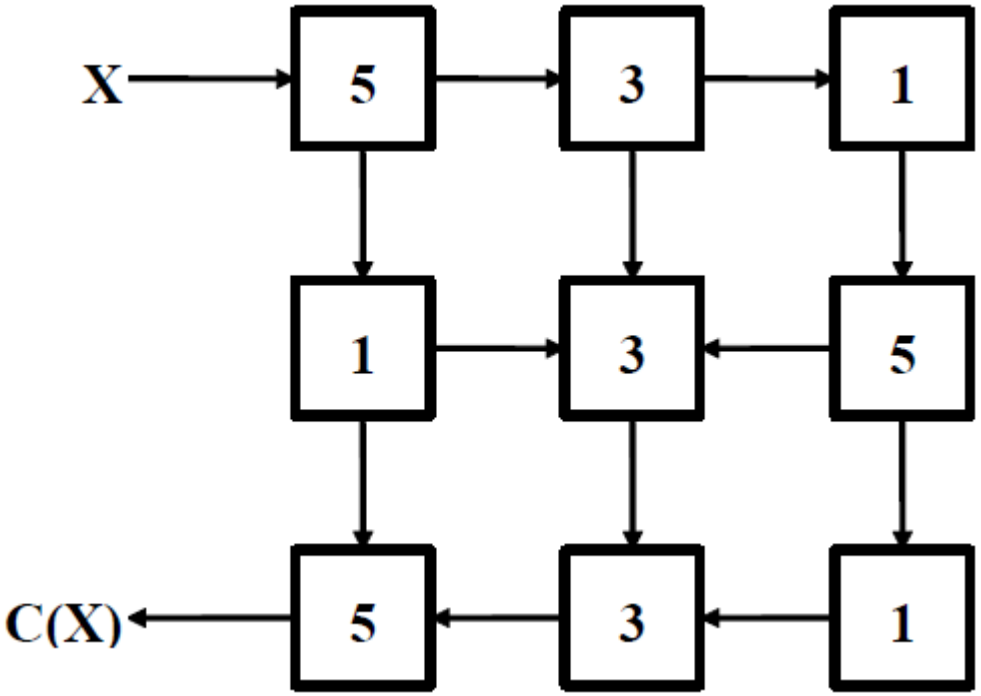


Latency = 5 + 3 +1 + 5 + 3 + 3 + 5 = 25μs

Throughput = 1/25μs

Exercise 7

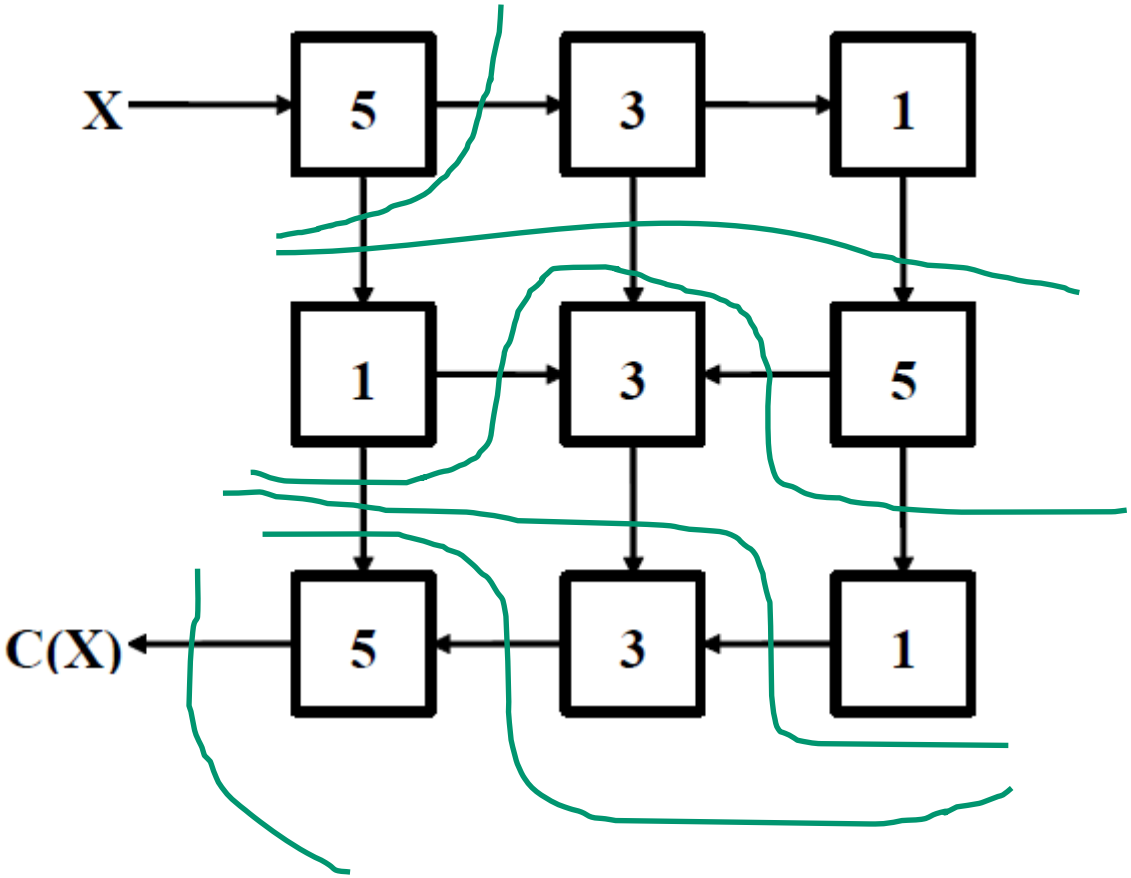
- Question2: how to pipeline the device to get maximum throughput?



Exercise 7

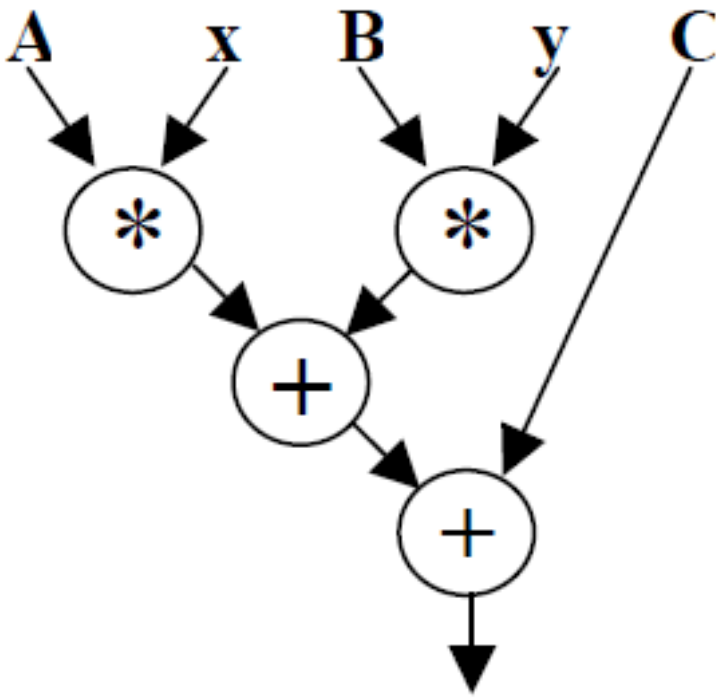
- Answer: the clock period is 5μs@6 stages, so latency = 30μs

  throughput = 1/5μs



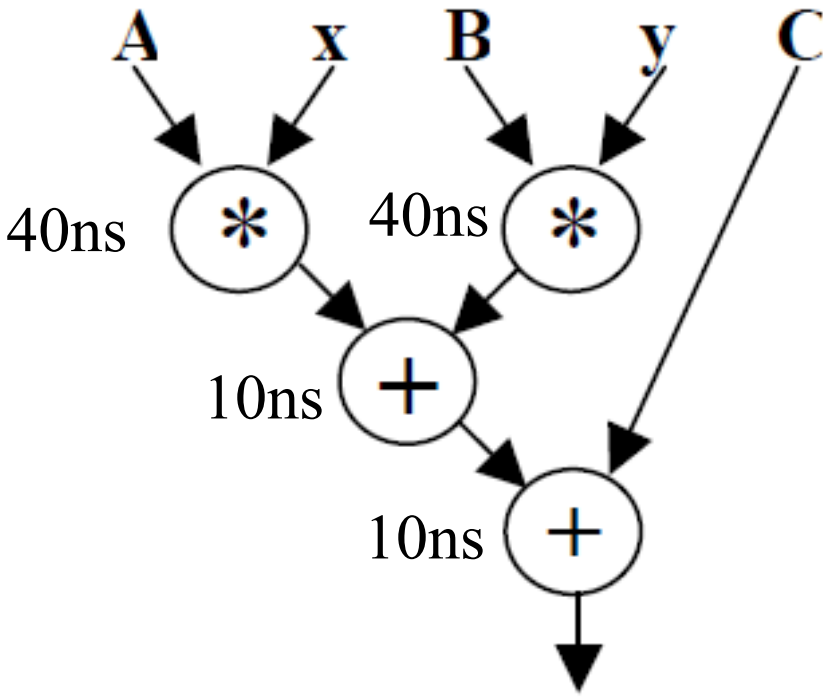
Exercise 8

- For the following implementation, assume the propagation delay of the multipliers is 40ns, and the propagation delay of the adders is 10ns.
- Question 1: what is the propagation delay and throughput of this design?



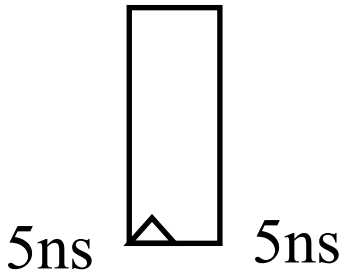
Exercise 8

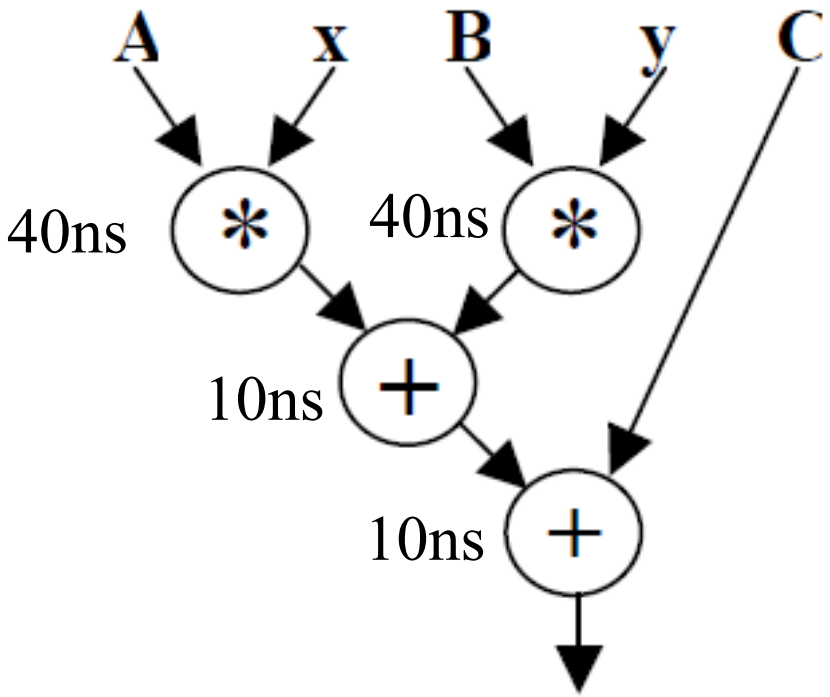
- Propagation delay is 40+10+10=60ns, and the throughput is 1/60ns



Exercise 8

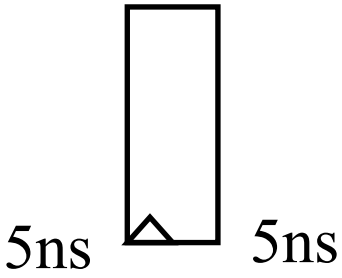
- Question 2: If registers are used to make a pipelined design, 5ns setup time and 5ns propagation delay for a register, what is the maximum throughput that can be achieved?

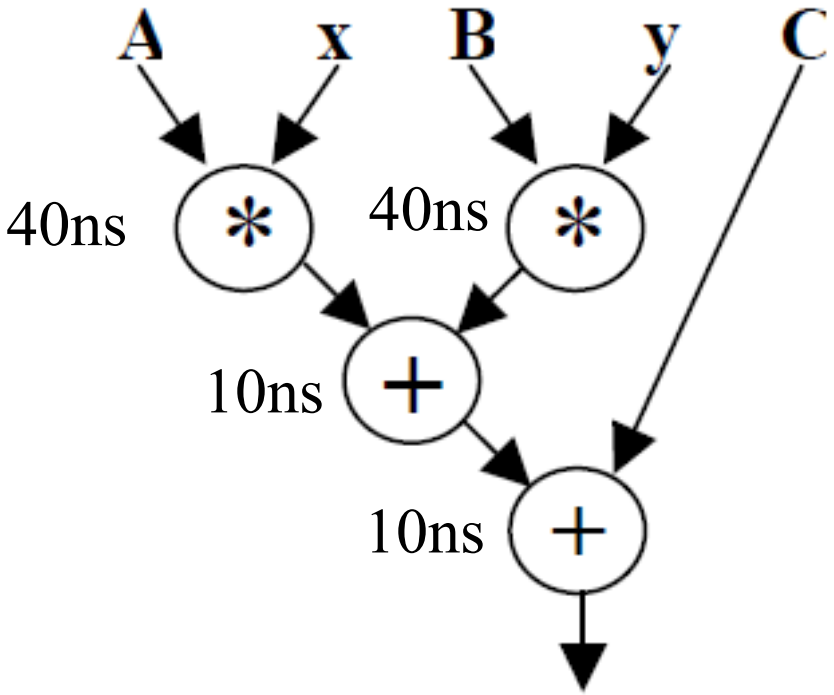




Exercise 8

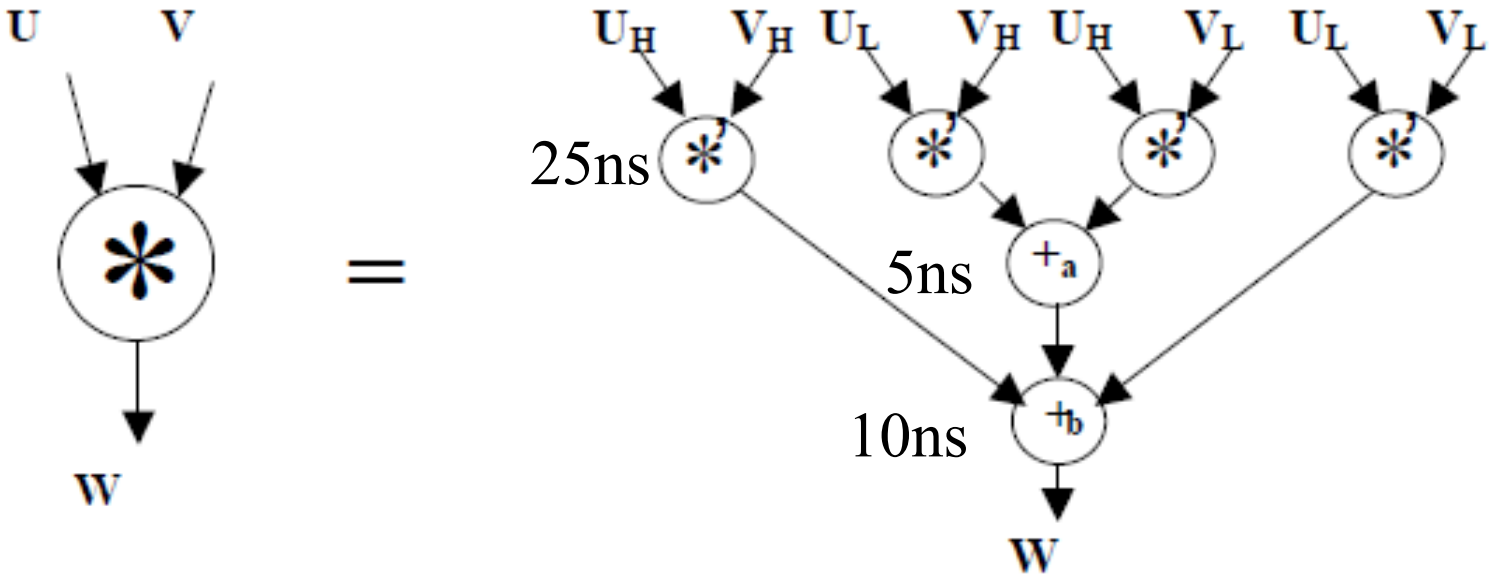
- Answer: The minimum clock period we can use is determined by the slowest component, which is a multiplier. Let us set our clock period to be 5+40+5=50ns. We can perform both multiplications in parallel. There is no need to put the two adders in separate clock stages, we can perform both additions in one clock period. Thus, it will take only two clock cycles to perform this computation. Maximum throughput = 1/50ns and latency is 100ns





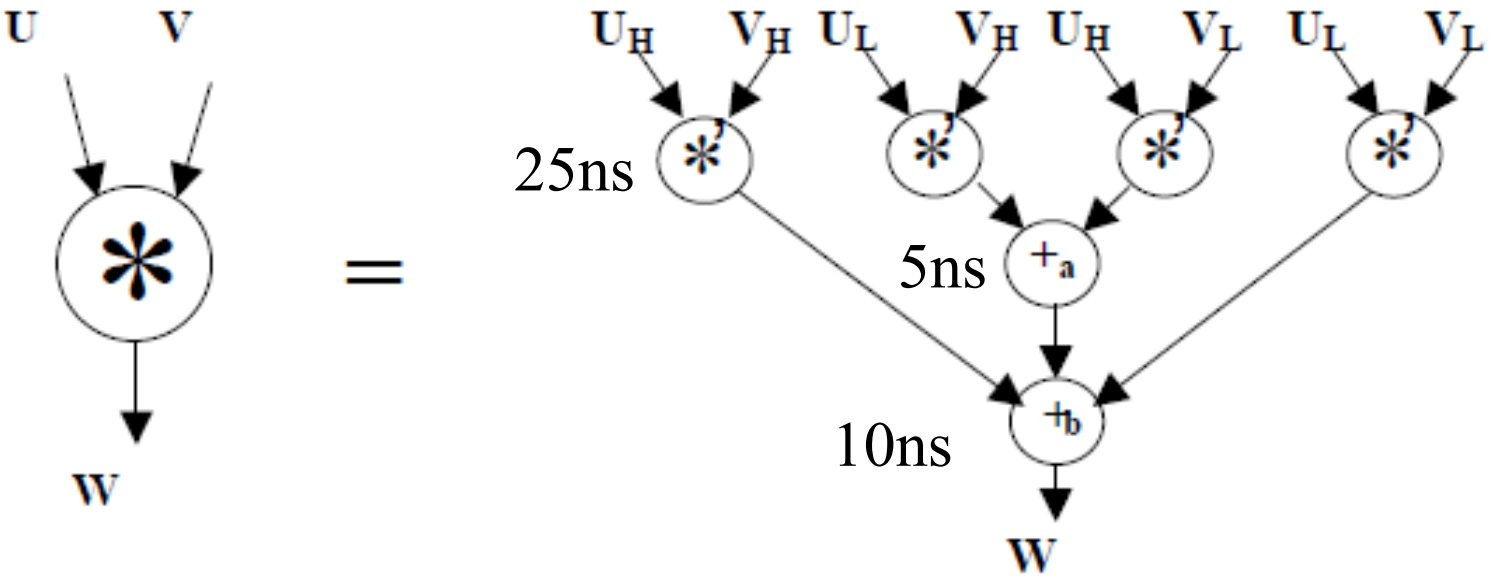
Exercise 8

- Question 3: assume a multiplier is actually implemented using four smaller multipliers as below. A smaller multiplier has a propagation delay of 25ns, the small adder (+a) has a propagation delay of 5ns, the large adder (+b) has a propagation delay of 10ns. What is the maximum throughput that can be achieved?



Exercise 8

- Answer: Again, the minimum clock period we can use is determined by the slowest component, which is the 25ns multiplier. Let us set our clock period to be 5+25+5 = 35ns. We can combine the two adders internally, since their combined delay is 15ns, less than 25ns. We will need one more pipeline stage to compute the final two additions. The maximum throughput is 1/35ns, and latency is 3*35ns =105ns



Exercise 9

- The following assign statement describes combinational logic that takes in a set of five 8-bit inputs (a, b, c, d, and e) and carries out the operations to produce a 16-bit output (f). Assume that the four operations take the same amount of time to execute.

```
assign f = ((a | b) + c ) * (d & e);
```

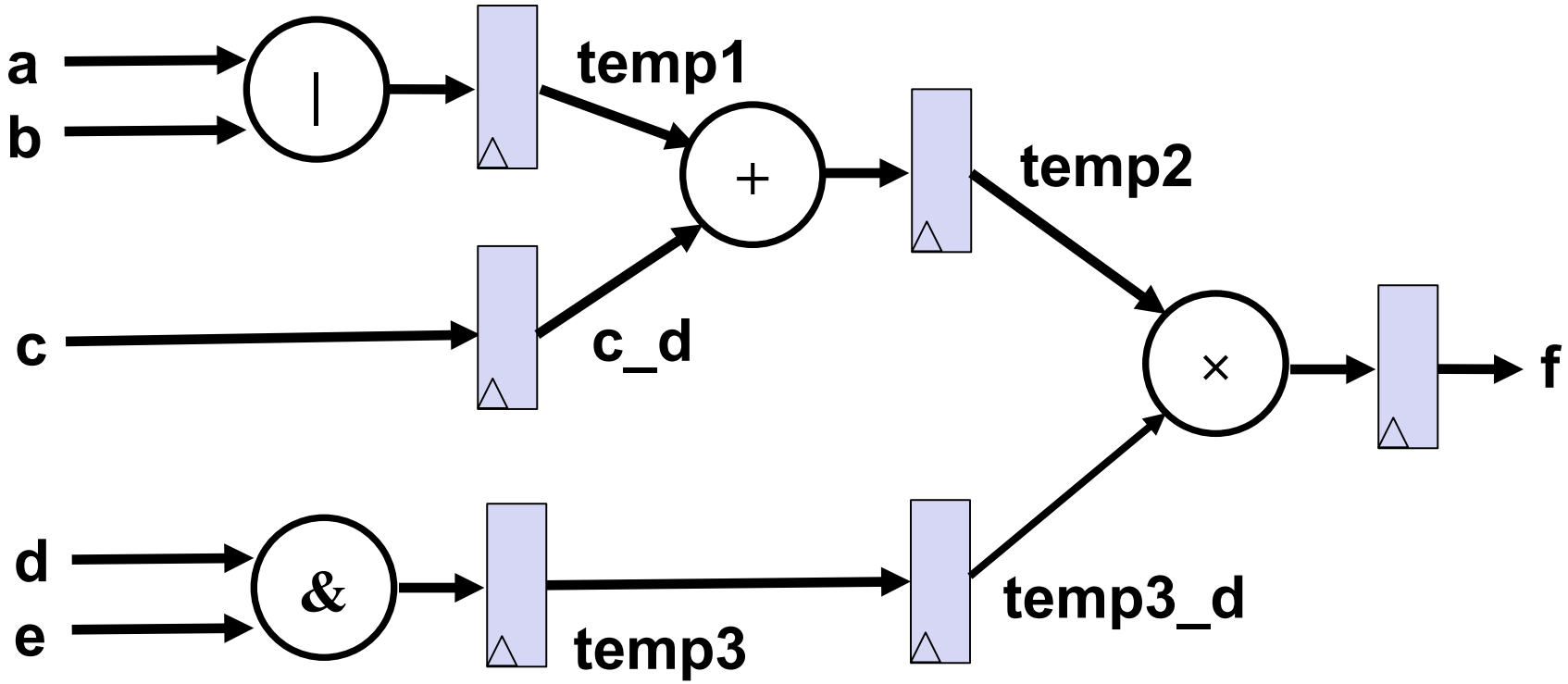
Question1: Write a synthesizable module description of a pipelined version to carry out the same operations so that new output (f) values can be generated fro each new set of input values (a, b, c, d, and e).

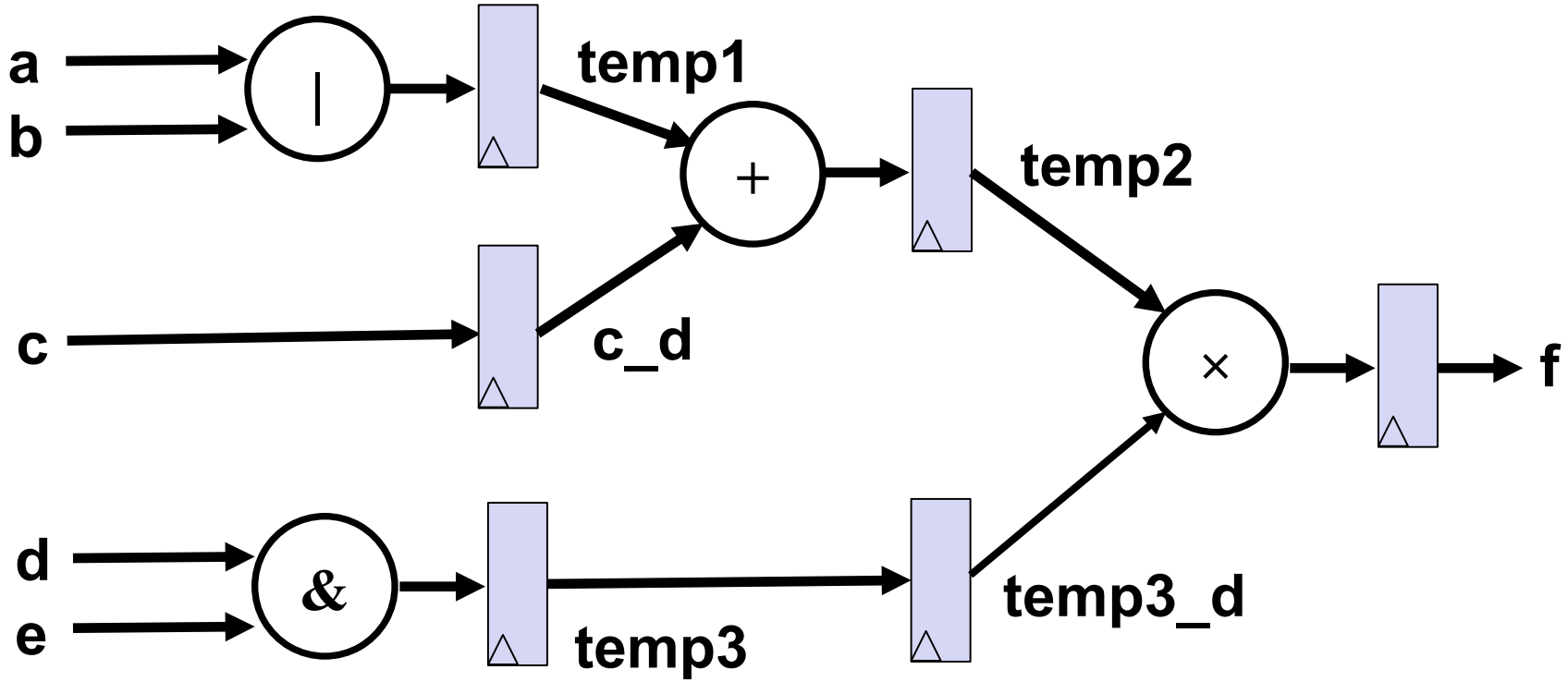
Question2: How many flip-flops will be required to implement this pipeline version?

Question 3: How many clock cycles will it take to produce a new output form a new set of inputs?

Exercise 9

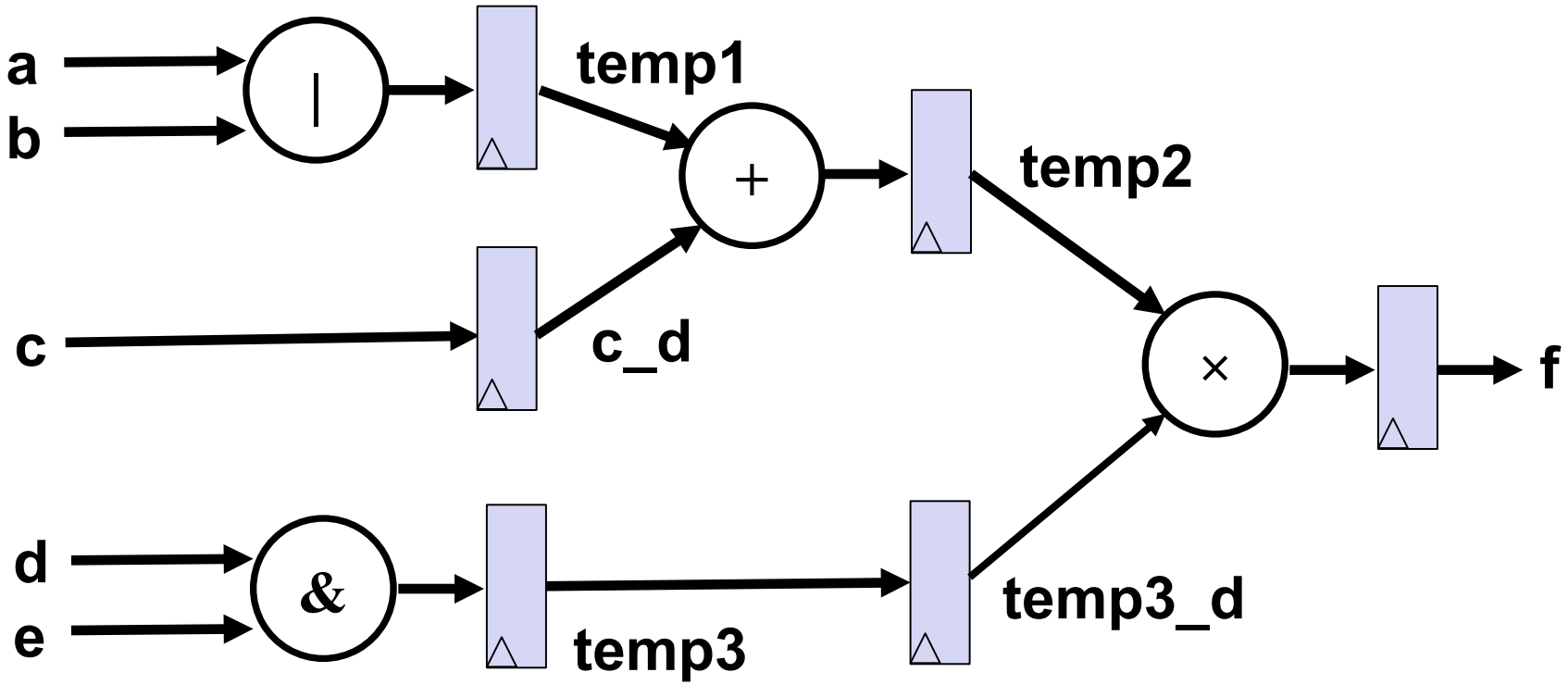
assign f = ((a | b) + c) * (d & e);






```
module pipeline2 (clk, a, b, c, d, e, f);
input clk;
input [7:0] a, b, c, d, e, f;
output reg [15:0] f;
reg [7:0] c_d, temp1, temp2, temp3, temp3_d;
always @ (posedge clk)
        begin
```

```
temp1 <= a | b;
c_d <= c;
temp2 <= temp1 + c_d;
temp3 <= d & e;
temp3_d <= temp3;
f <= temp2 * temp3_d;
        end
endmodule
```



flip-flops: 8+8+8+8+8+16 = 56
latency= 3 clock cycles