

Introduction to Pipelining

Welcome back to 6.004!

Reminders:

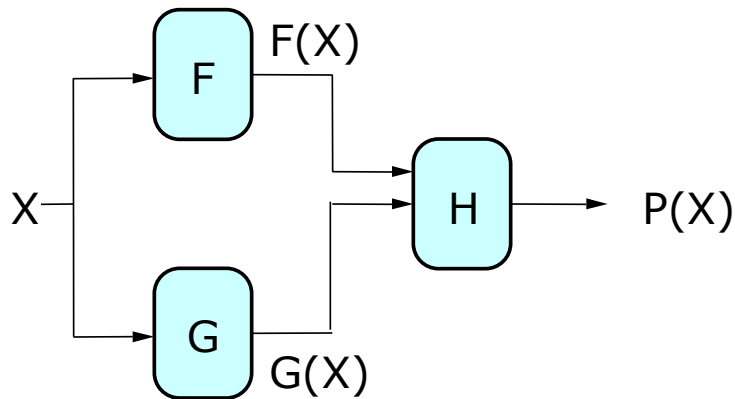
Lab 3 due Thursday, April 2nd

Lab 4 due Tuesday, April 7th

Performance Measures

- Two metrics of interest when designing a system:
 1. Latency: The *delay* from when an input enters the system until its associated output is produced
 2. Throughput: The *rate* at which inputs or outputs are processed
- The metric to prioritize depends on the application
 - *Airbag deployment system?* Latency
 - *General-purpose processor?* Throughput
(maximize instructions/second)

Performance of Combinational Logic

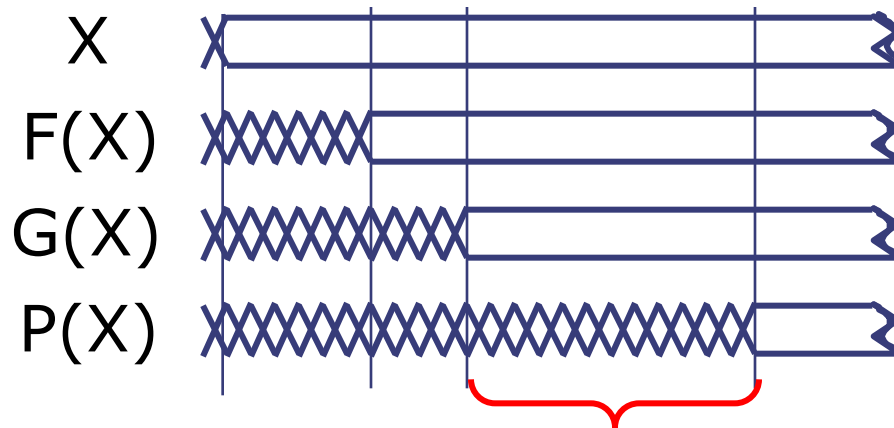


For combinational logic:

$$\text{latency} = t_{PD}$$

$$\text{throughput} = 1/t_{PD}$$

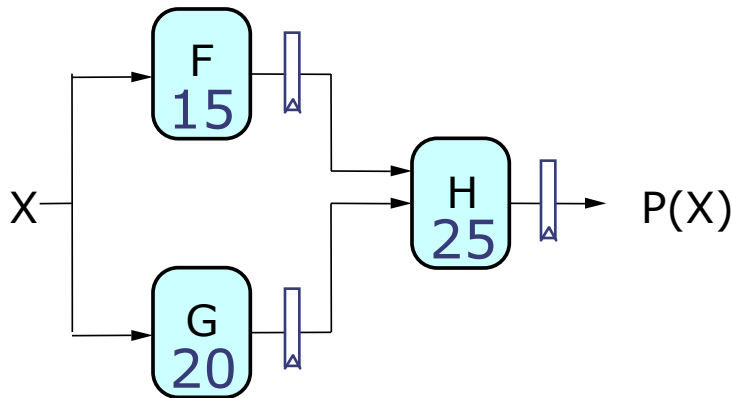
We can't get the answer any faster, but are we making effective use of our hardware at all times?



F & G are "idle", just holding their outputs stable while H performs its computation

Pipelined Circuits

Use registers to hold H's input stable!

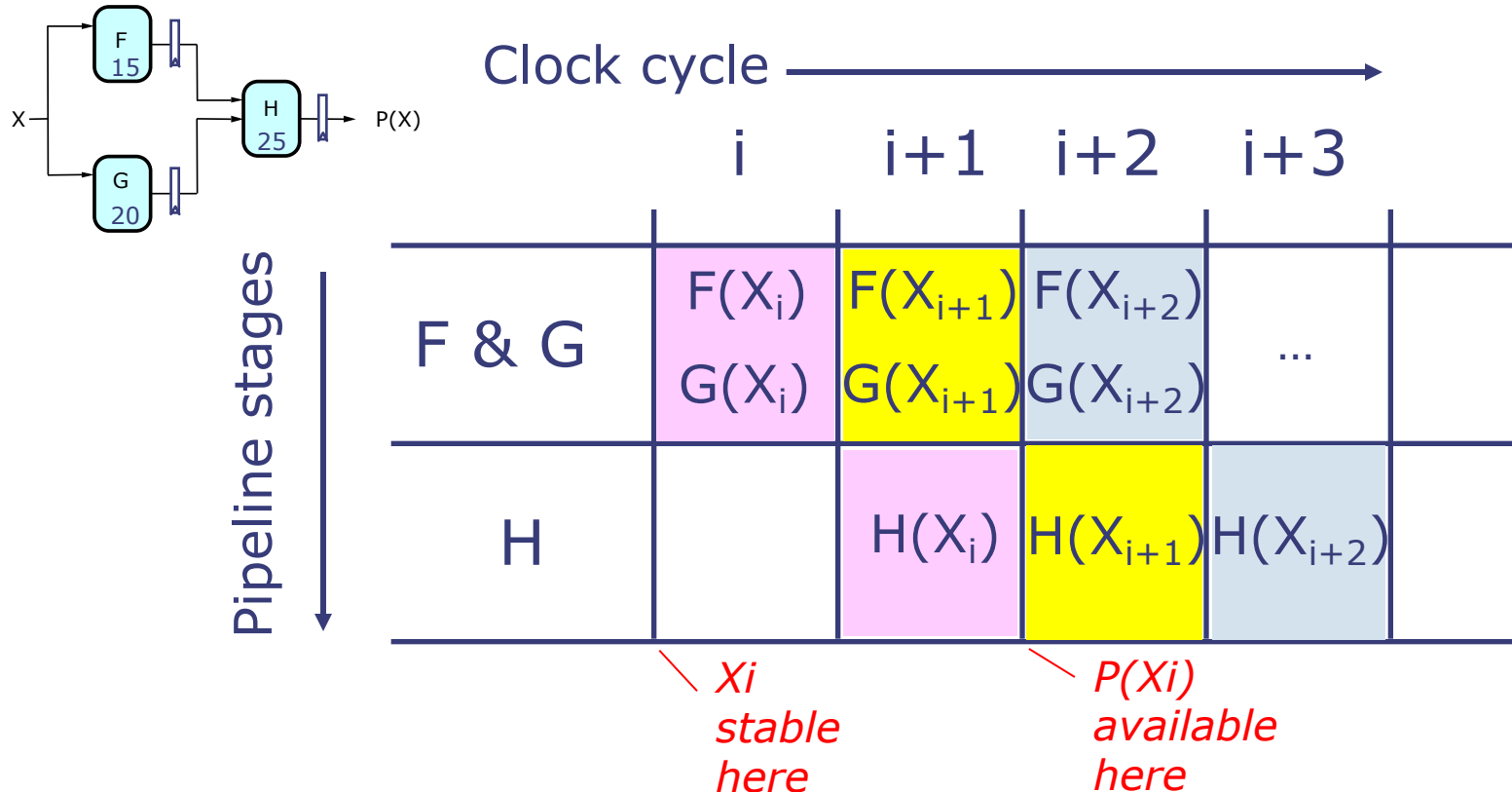


Now F & G can be working on input X_{i+1} while H is performing its computation on X_i . We've created a 2-stage *pipeline*: if we have a valid input X during clock cycle j, P(X) is valid during clock cycle j+2.

Suppose F, G, H have propagation delays of 15, 20, 25 ns and we are using ideal registers ($t_{PD} = 0$, $t_{SETUP} = 0$):

	<u>latency</u>	<u>throughput</u>
unpipelined	45	1/45
2-stage pipeline	<u>50</u>	<u>1/25</u>
	worse	better!

Pipeline Diagrams



The results associated with a particular set of input data moves *diagonally* through the diagram, progressing through one pipeline stage each clock cycle.

Pipeline Conventions

Definition:

A well-formed *K-Stage Pipeline* (“K-pipeline”) is an acyclic circuit having exactly K registers on *every* path from an input to an output.

A combinational circuit is thus a 0-stage pipeline.

Composition convention:

Every pipeline stage, hence every K-Stage pipeline, has a register on its *output* (not on its input).

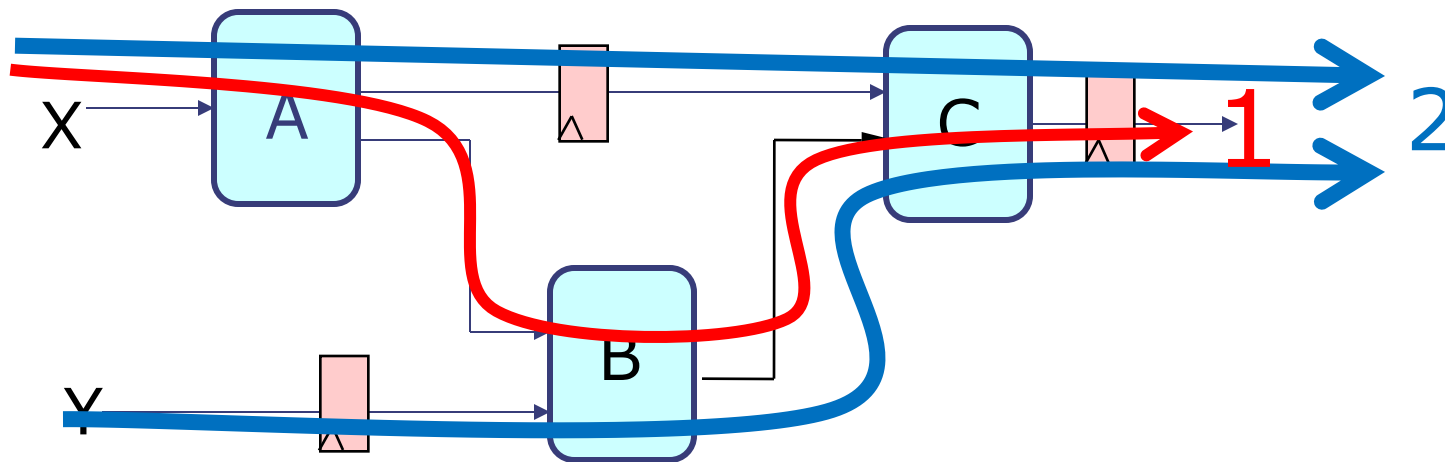
Clock period:

The clock must have a period t_{CLK} sufficient to cover the longest register to register propagation delay plus setup time.

$$\begin{aligned} \text{K-pipeline latency } L &= K * t_{\text{CLK}} \\ \text{K-pipeline throughput } T &= 1 / t_{\text{CLK}} \end{aligned}$$

Ill-Formed Pipelines

Consider a BAD job of pipelining:



For what value of K is the following circuit a K-Pipeline? *none*

Problem:

Successive inputs get mixed: e.g., $B(A(X_{i+1}), Y_i)$.
This happens because some paths from inputs to outputs have 2 registers, and some have only 1!

This can't happen in a well-formed K pipeline!

A Pipelining Methodology

Step 1:

Draw a line that crosses every output in the circuit, and mark the endpoints as terminal points.

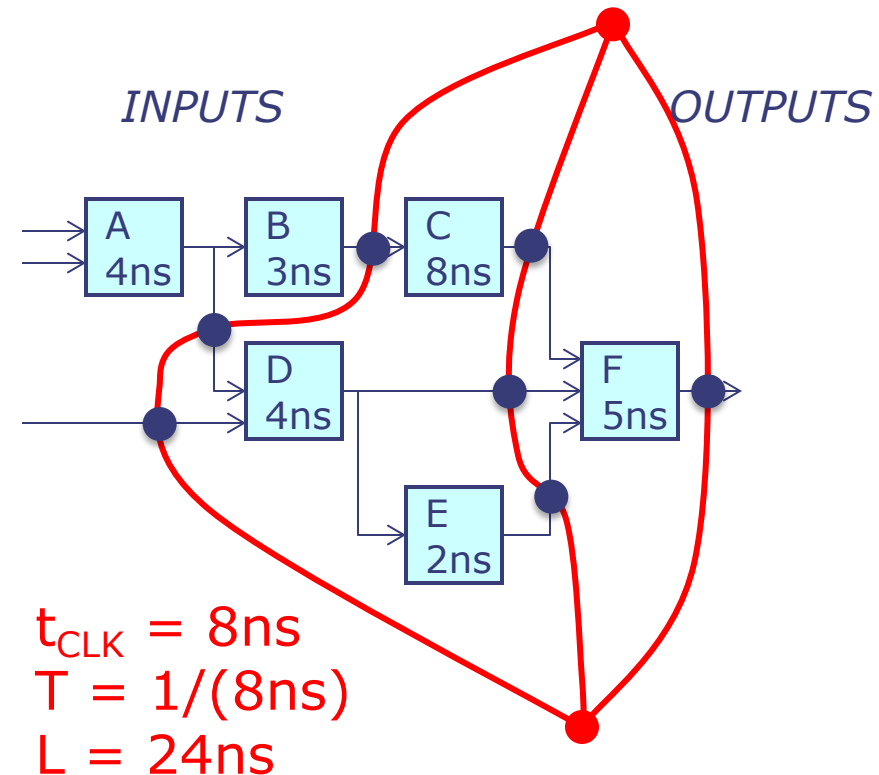
Step 2:

Continue to draw new lines between the terminal points across various circuit connections, ensuring that every connection crosses each line in the same direction. These lines demarcate *pipeline stages*.

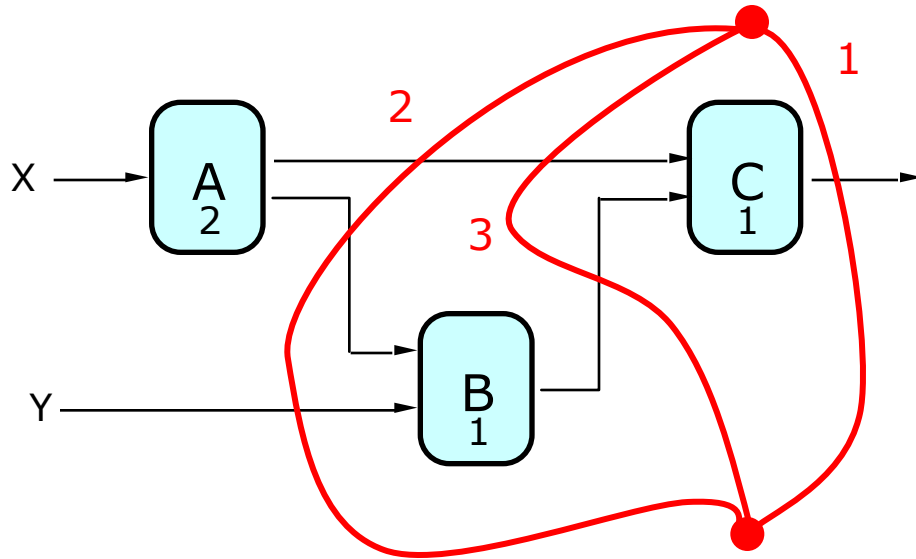
Adding a pipeline register at every point where a separating line crosses a connection will always generate a valid pipeline.

Strategy:

Focus your attention on placing pipelining registers around the slowest circuit elements (*bottlenecks*).



Pipeline Example

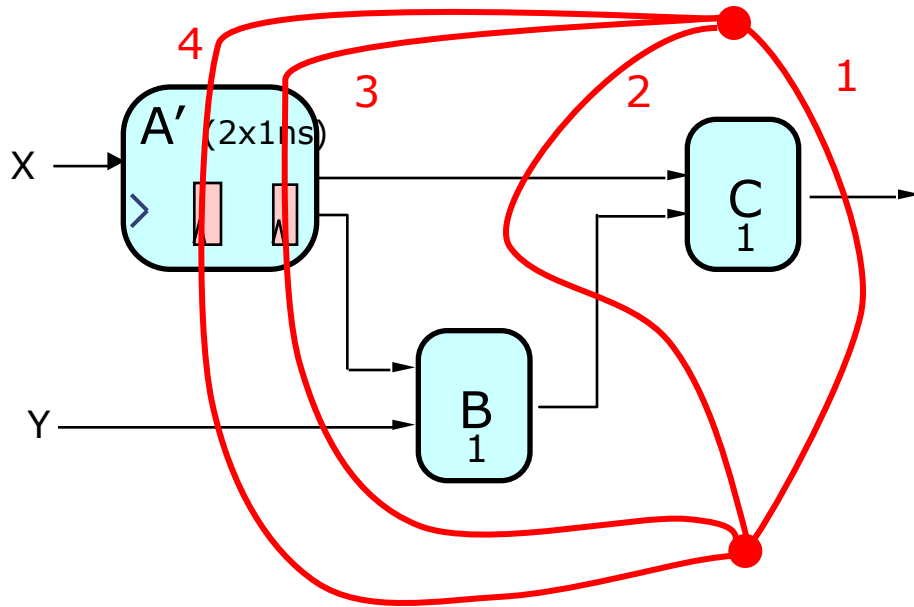


OBSERVATIONS:

- 1-pipeline improves neither L nor T.
- T improved by breaking long combinational paths, allowing faster clock.
- Too many stages cost L, don't improve T.
- Back-to-back registers are sometimes needed to keep pipeline well-formed.

	LATENCY	THROUGHPUT
0-pipe:	4	1/4
1-pipe:	4	1/4
2-pipe:	4	1/2
3-pipe:	6	1/2

Pipelined Components



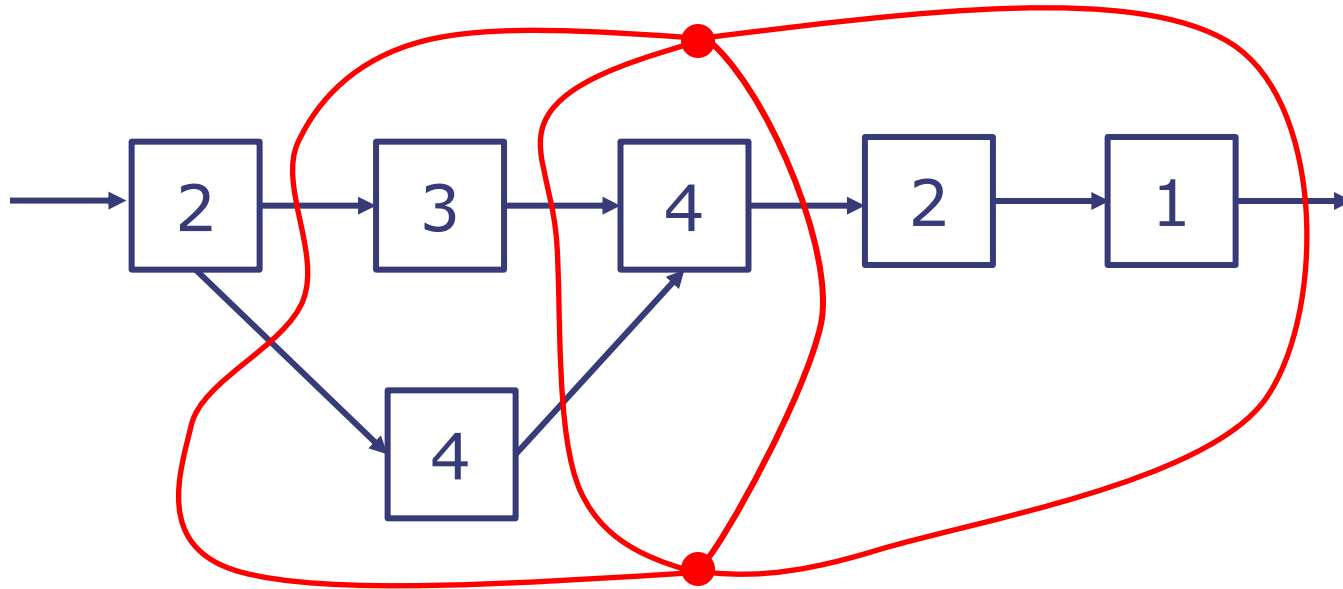
4-stage pipeline, throughput=1

Pipelined systems can be hierarchical:

- Replacing a slow combinational component with a k-pipe version may let us decrease the clock period
- Must account for new pipeline stages in our plan

Sample Pipelining Problem

- Pipeline the following circuit for maximum throughput while minimizing latency. The number in each module is the module's latency.



- What is the latency and throughput of your pipelined circuit?

$$\begin{aligned}t_{\text{CLK}} &= 4 \\T &= 1/(4) \\L &= 4*4 = 16\end{aligned}$$

Design Tradeoffs

Introduction:

Multiplier Case Study

Multiplication by repeated addition

b Multiplicand 1101 (13)
a Multiplier * 1011 (11)

tp
m0 + 0000
+ 1101
tp
m1 + 01101
+ 1101
tp
m2 + 100111
+ 0000
tp
m3 + 0100111
+ 1101
tp
10001111 (143)

At each step we add either 1101 or 0 to the result depending upon a bit in the multiplier

$$m_i = (a[i] == 0) ? 0 : b;$$

We also shift the result by one position at every step

Notice, the first addition is unnecessary because it simply yields m0

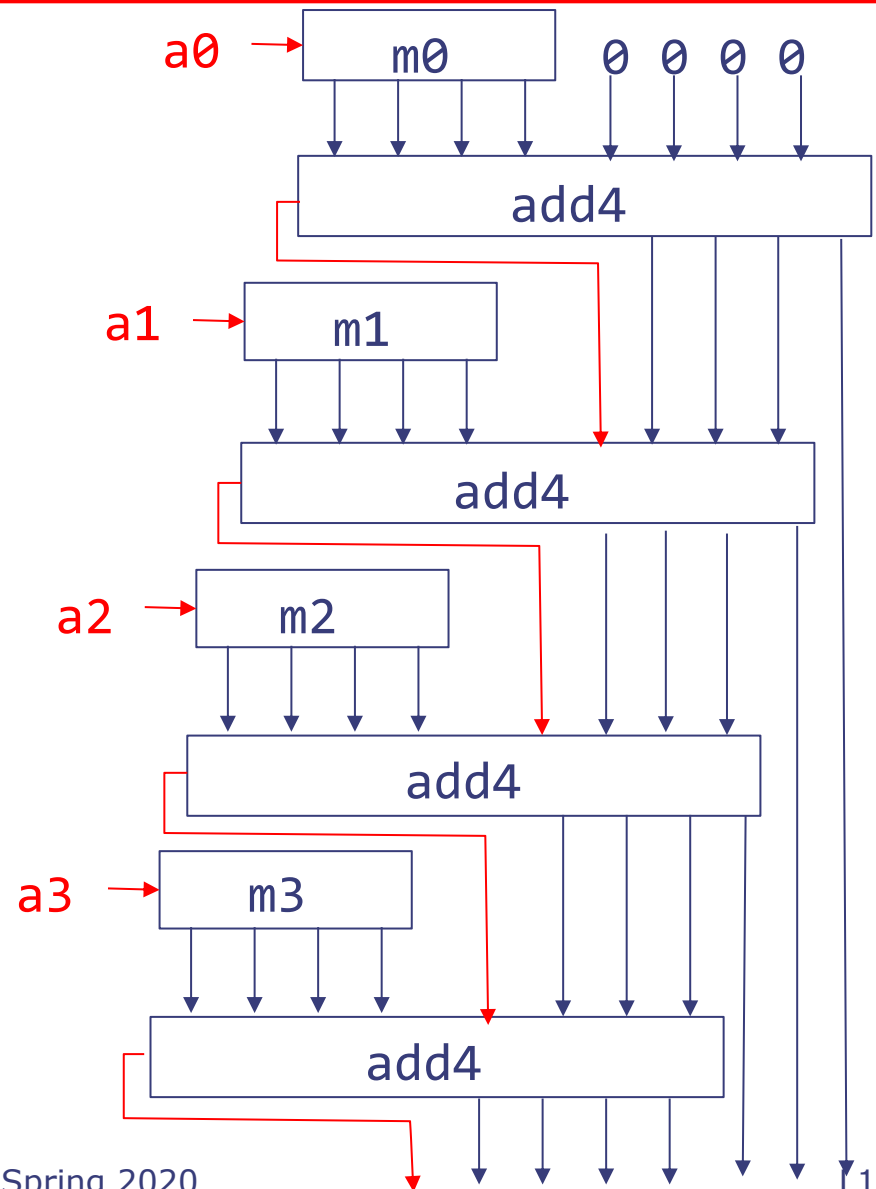
Also note that these are unsigned binary numbers.

Multiplication by repeated addition circuit

b Multiplicand 1101 (13)
a Multiplier * 1011 (11)

tp		0000	
m0	+	1101	
tp		01101	
m1	+	1101	
tp		100111	
m2	+	0000	
tp		0100111	
m3	+	1101	
tp		10001111	(143)

$m_i = (a[i] == 0) ? 0 : b;$



Implementation of m_i

The “Binary”
Multiplication
Table

*	0	1
0	0	0
1	0	1

= AND

$$m_i = (a[i] == 0) ? 0 : b;$$

$$\begin{array}{rcccc}
 & B_3 & B_2 & B_1 & B_0 \\
 \times & A_3 & A_2 & A_1 & A_0 \\
 \hline
 BA_i \text{ called a “partial product”} \longrightarrow & B_3A_0 & B_2A_0 & B_1A_0 & B_0A_0 & m0 \\
 & B_3A_1 & B_2A_1 & B_1A_1 & B_0A_1 & m1 \\
 & B_3A_2 & B_2A_2 & B_1A_2 & B_0A_2 & m2 \\
 + & B_3A_3 & B_2A_3 & B_1A_3 & B_0A_3 & m3 \\
 \hline
 \end{array}$$

Multiplying N-digit number by M-digit number gives (N+M)-digit result

Easy part: forming partial products (bunch of AND gates)

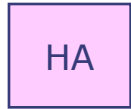
Hard part: adding M N-bit partial products

Combinational Multiplier Redrawn



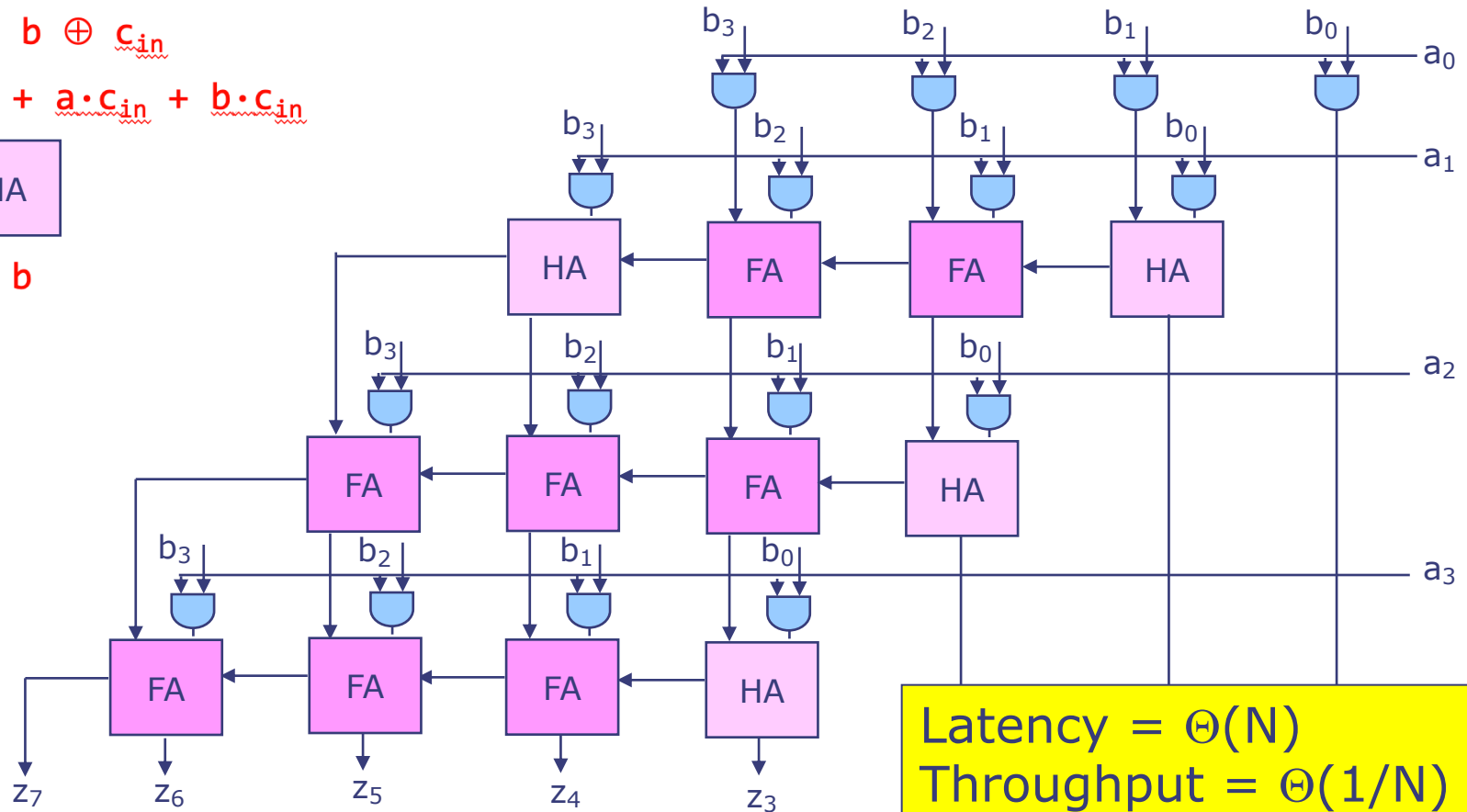
$$s = a \oplus b \oplus c_{in}$$

$$c_{out} = a \cdot b + a \cdot c_{in} + b \cdot c_{in}$$



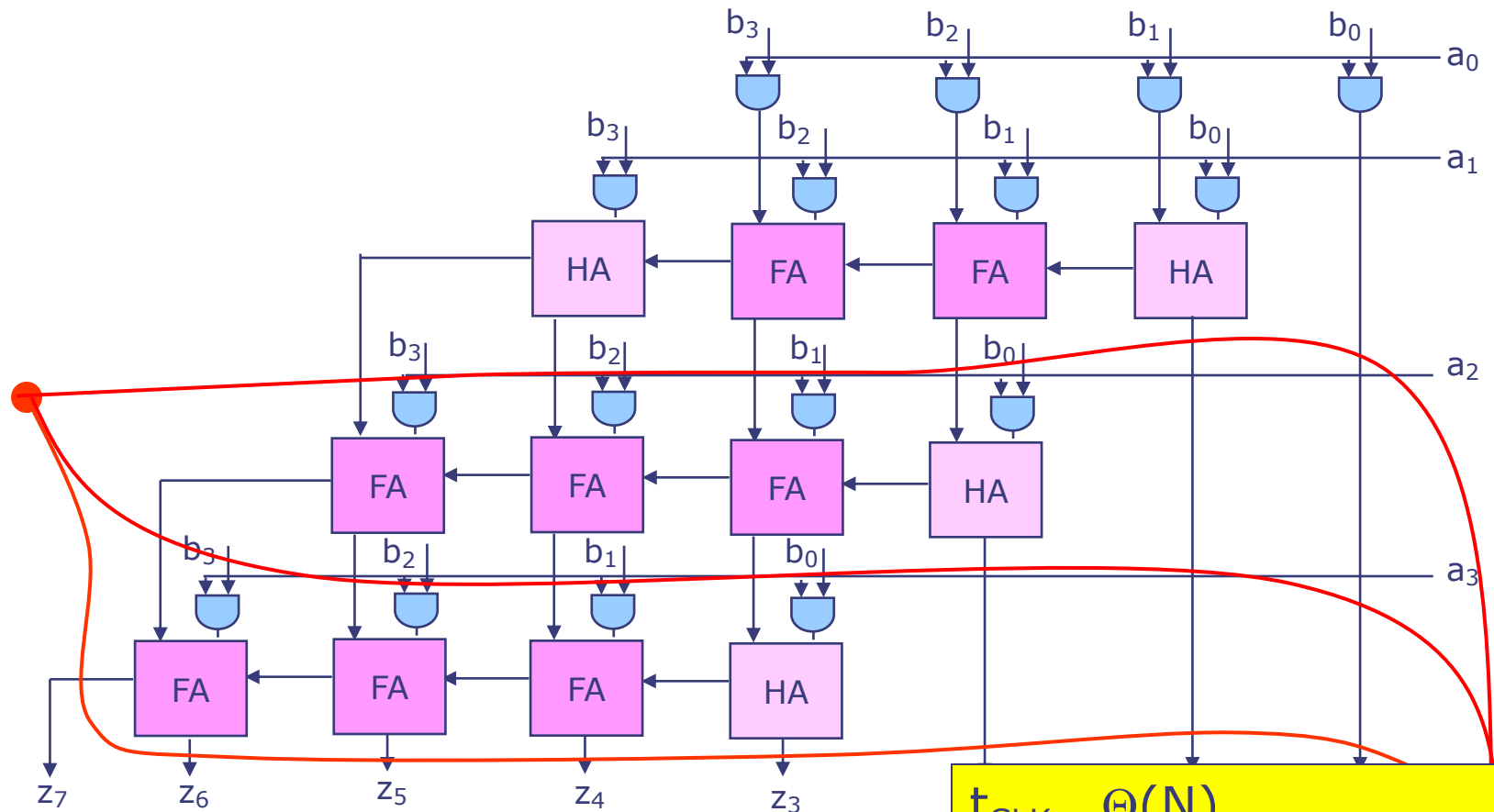
$$s = a \oplus b$$

$$c_{out} = a \cdot b$$



Latency = $\Theta(N)$
Throughput = $\Theta(1/N)$
Area = $\Theta(N^2)$

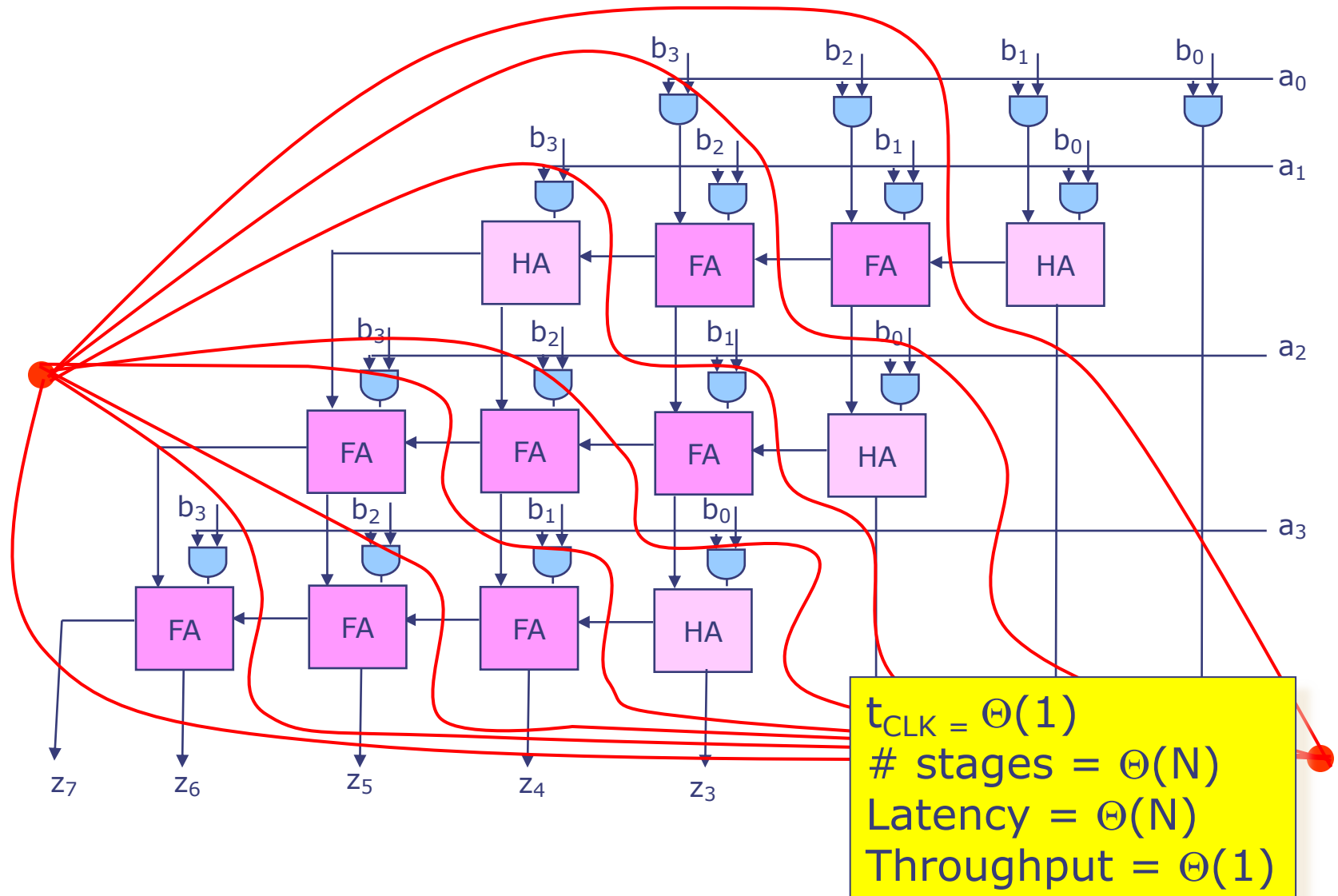
Increase Throughput with Pipelining - First Attempt



Need to break carry chain

$t_{\text{CLK}} = \Theta(N)$
stages = $\Theta(N)$
Latency = $\Theta(N^2)$
Throughput = $\Theta(1/N)$

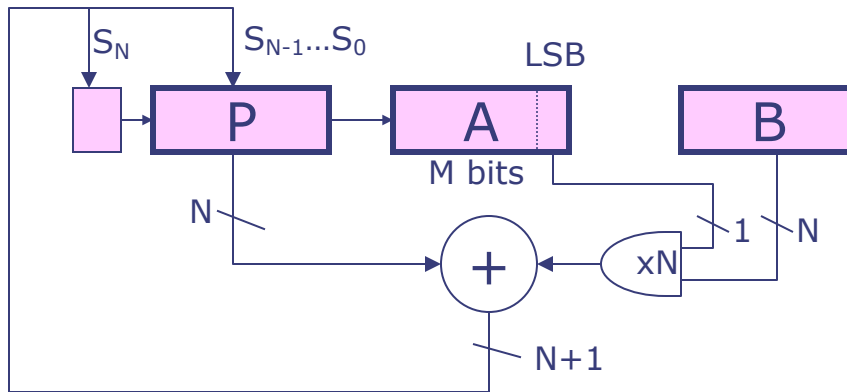
Increase Throughput with Pipelining



Folded Multiplier

Reduce Area With Sequential Logic

Assume the multiplicand (B) has N bits and the multiplier (A) has M bits. If we only want to invest in a single N-bit adder, we can process adds sequentially using the same adder M times. **Tradeoff increased latency for reduced area.**



Init: $P \leftarrow 0$, load A&B

Repeat M times {
 $P \leftarrow P + (A_{\text{LSB}} == 1 ? B : 0)$
 shift S_N, P, A right one bit
}

Done: (N+M)-bit result in P,A

Using Ripple Carry Adder

$t_{\text{CLK}} = \Theta(N)$

stages = $\Theta(N)$

Latency = $\Theta(N^2)$

Throughput = $\Theta(1/N^2)$

Area = $\Theta(N)$

Using Diagonal Partial Products

$t_{\text{CLK}} = \Theta(1)$

stages = $\Theta(N)$

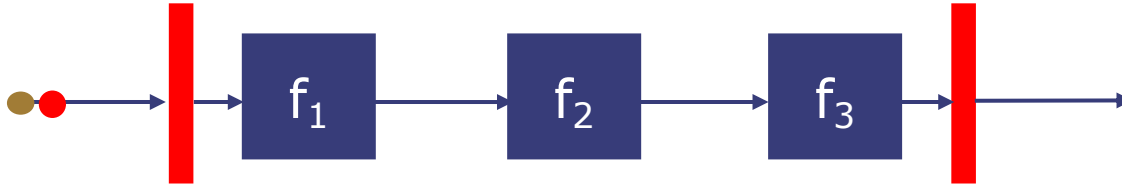
Latency = $\Theta(N)$

Throughput = $\Theta(1/N)$

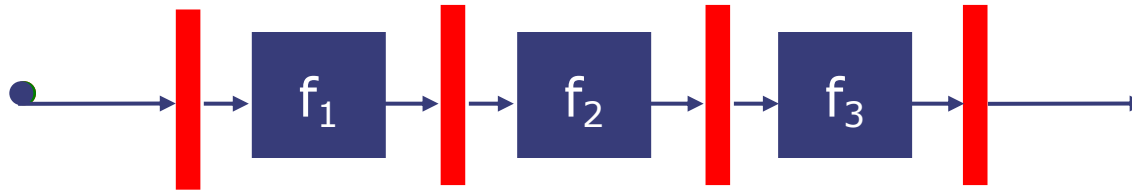
Area = $\Theta(N)$

Pipelining Design Alternatives

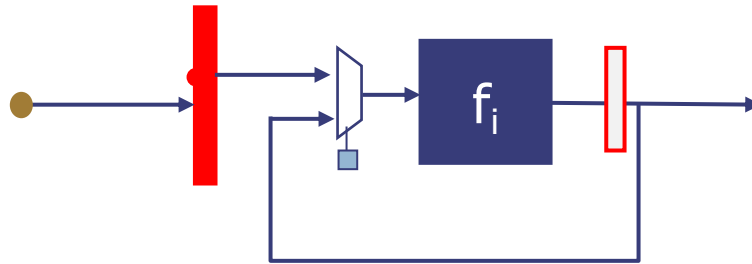
Several combinational modules in one pipeline stage (A)



One module per pipeline stage (B)



Folded reuse a block, multicycle (C)



Clock: $B \approx C < A$

Area: $C < A < B$

Throughput: $C < A < B$

Thank You!

Next Lecture:
Building a Single Cycle Processor