Digital Logic - Combinational

Synchronous Digital Systems (SDS)

Hardware of a processor, such as a RISC-V processor, is an example of a Synchronous Digital System

Synchronous:

- All operations coordinated by a central clock
  - "Heartbeat" of the system! (processor frequency)

Digital:

- Represent all values with two discrete values
- Electrical signals are treated as 1's and 0's
  - 1 and 0 are complements of each other
- High/Low voltage for True/False, 1/0

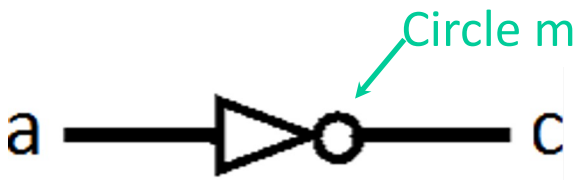
Type of Circuits

- *Digital Systems* consist of two basic types of circuits:
  - Combinational Logic (CL)
    - Output is a function of the inputs only, not the history of its execution
    - e.g. circuits to add A, B (ALUs)
  - Sequential Logic (SL)
    - Circuits that "remember" or store information
    - a.k.a. "State Elements"
    - e.g. memory and registers (Registers)

Logic Gates (1/2)

- Special names and symbols:

NOT



a	c
0	1
1	0

AND

a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

OR

a	b	c
0	0	0
0	1	1
1	0	1
1	1	1

Logic Gates (2/2)

Inverted versions are easier to implement in CMOS

NAND

a	b	c
0	0	1
0	1	1
1	0	1
1	1	0

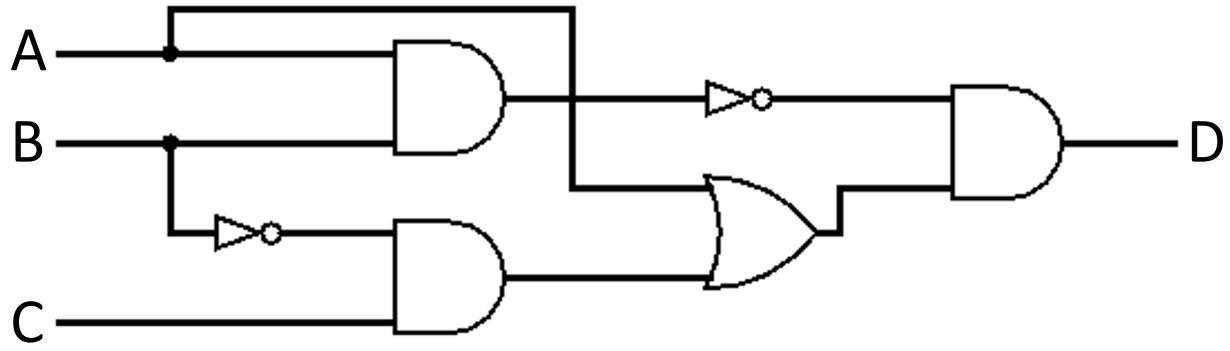
NOR

a	b	c
0	0	1
0	1	0
1	0	0
1	1	0

XOR

a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

Combining Multiple Logic Gates

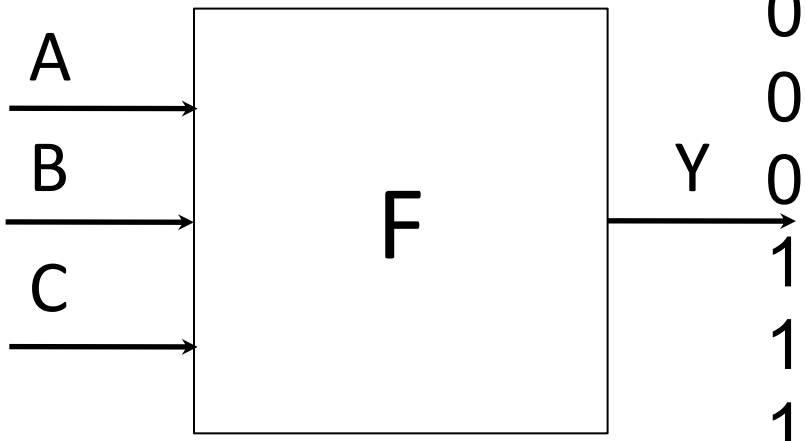


(NOT(**A** AND **B**)) AND (**A** OR (NOT **B** AND **C**))

Agenda

- **Combinational Logic**
  - Combinational Logic Gates
  - **Truth Tables**
  - Boolean Algebra
  - Circuit Simplification

General Form



A	B	C		Y
0	0	0	F(0,0,0)	0
0	0	1	F(0,0,1)	1
0	1	0	F(0,1,0)	0
0	1	1	F(0,1,1)	0
1	0	0	F(1,0,0)	0
1	0	1	F(1,0,1)	1
1	1	0	F(1,1,0)	1
1	1	1	F(1,1,1)	0

If N inputs, how many distinct functions F do we have?

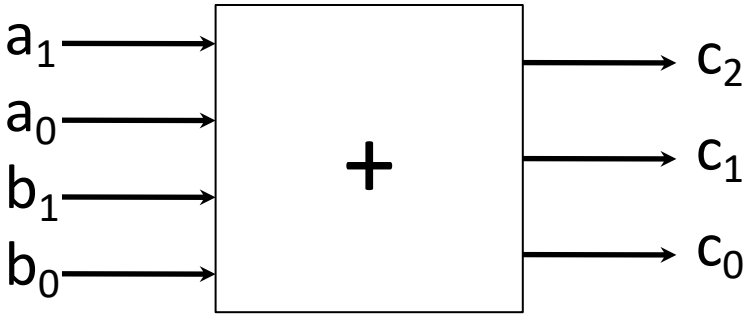
Function maps each row to 0 or 1, so 2^N possible functions

More Complicated Truth Tables

3-Input Majority

a	b	c	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

2-bit Adder



How many rows?

A		B		C		
a_1	a_0	b_1	b_0	c_2	c_1	c_0
			.			
			.			
			.			

Agenda

- **Combinational Logic**
  - Combinational Logic Gates
  - Truth Tables
  - **Boolean Algebra**
  - Circuit Simplification

Boolean Algebra

- Represent inputs and outputs as variables
  - Each variable can only take on the value 0 or 1
- Overbar or ¬ is NOT: "logical complement"
  - e.g. if A is 0, $\overline{A}$ is 1. If A is 1, then ¬A is 0

For slides, will use ¬A

- Plus (+) is 2-input OR: "logical sum"
- Product (·) is 2-input AND: "logical product"
  - All other gates and logical expressions can be built from combinations of these

    ¬**AB** + **A**¬**B** == (NOT(**A** AND **B**)) OR (**A** AND NOT **B**)

Laws of Boolean Algebra

These laws allow us to perform simplification:

$x \cdot \overline{x} = 0$	$x + \overline{x} = 1$	complementarity
$x \cdot 0 = 0$	$x + 1 = 1$	laws of 0's and 1's
$x \cdot 1 = x$	$x + 0 = x$	identities
$x \cdot x = x$	$x + x = x$	idempotent law
$x \cdot y = y \cdot x$	$x + y = y + x$	commutativity
$(xy)z = x(yz)$	$(x + y) + z = x + (y + z)$	associativity
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	distribution
$xy + x = x$	$(x + y)x = x$	uniting theorem
$\overline{x}y + x = x + y$	$(\overline{x} + y)x = xy$	uniting theorem v.2
$\overline{x \cdot y} = \overline{x} + \overline{y}$	$\overline{x + y} = \overline{x} \cdot \overline{y}$	DeMorgan's Law

Agenda

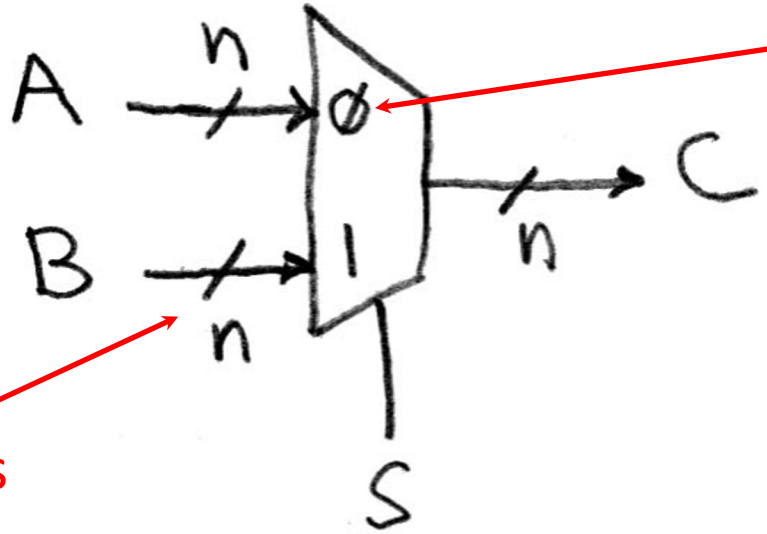
- **Muxes**
- Sequential Logic Timing
- Maximum Clock Frequency
- Finite State Machines
- Functional Units
- Summary

Bonus Slides

- Logisim Intro

Data Multiplexor

- Multiplexor ("MUX") is a *selector*
  - Place one of multiple inputs onto output (N-to-1)
- Shown below is an n-bit 2-to-1 MUX
  - Input S selects between two inputs of n bits each

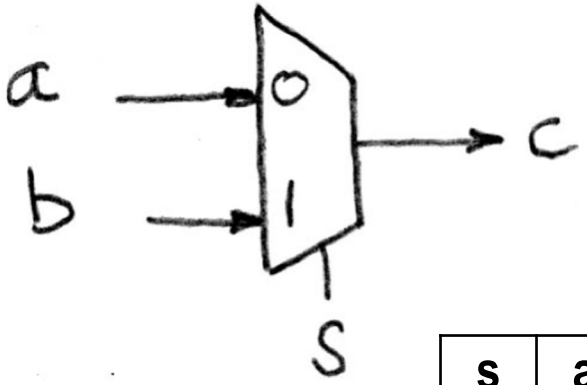


This input is passed to output if selector bits match shown value

Represents that this wire has n bits

Implementing a 1-bit 2-to-1 MUX

- **Schematic:**



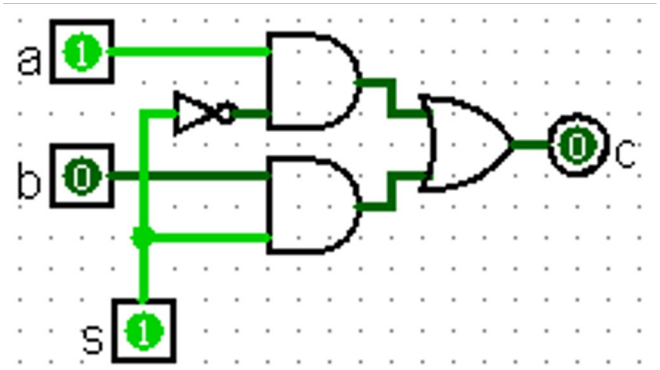
- **Truth Table:**

s	a	b	c
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

- **Boolean Algebra:**

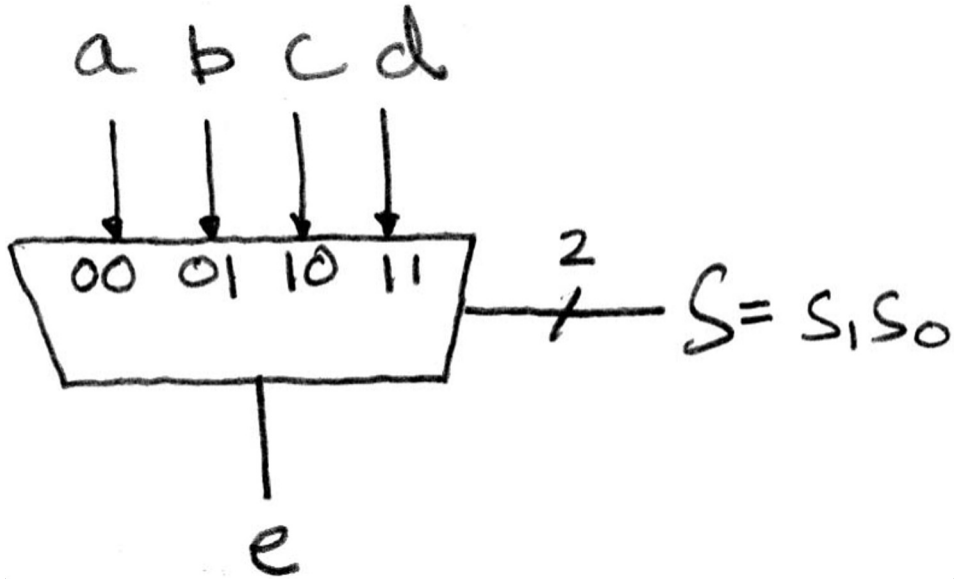
$$
\begin{aligned}
c &= \bar{s}a\bar{b} + \bar{s}ab + s\bar{a}b + sab \\
&= \bar{s}(a\bar{b} + ab) + s(\bar{a}b + ab) \\
&= \bar{s}(a(\bar{b} + b)) + s((\bar{a} + a)b) \\
&= \bar{s}(a(1) + s((1)b) \\
&= \bar{s}a + sb
\end{aligned}
$$

- **Circuit Diagram:**



1-bit 4-to-1 MUX (1/2)

- **Schematic:**

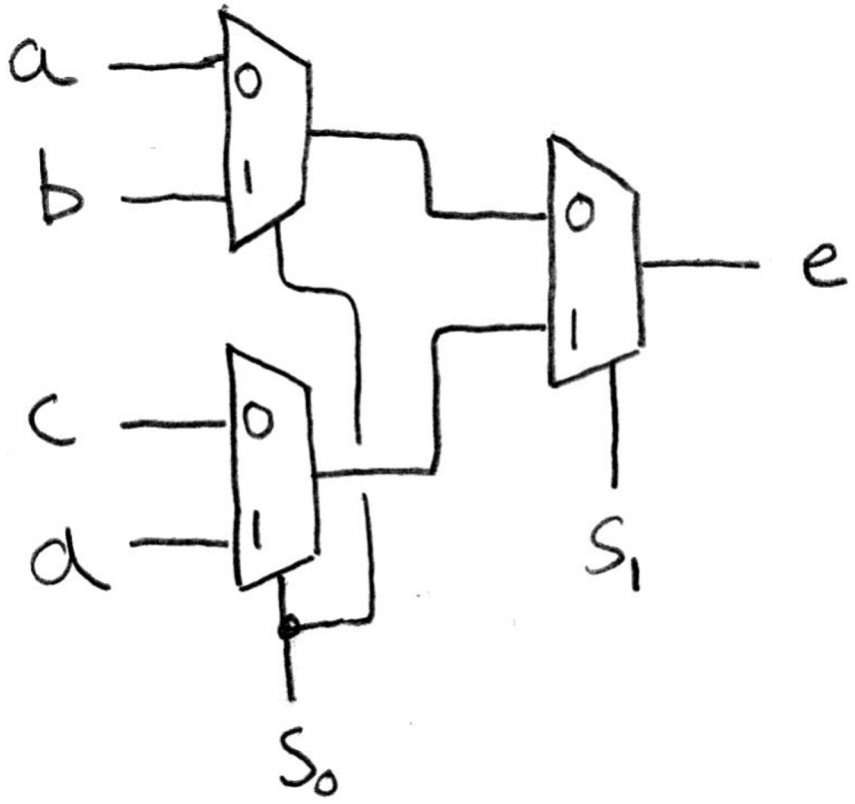


- **Truth Table:** How many rows? 2^6
- **Boolean Expression:**

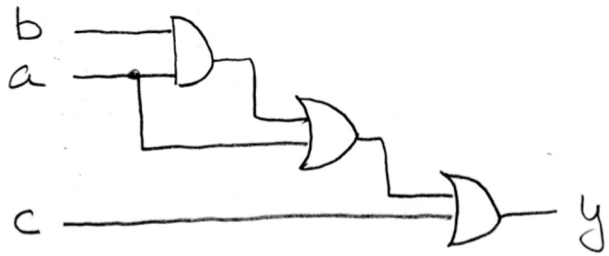
$$e = \neg s_1 \neg s_0 a + \neg s_1 s_0 b + s_1 \neg s_0 c + s_1 s_0 d$$

1-bit 4-to-1 MUX (2/2)

- Can we leverage what we've previously built?
  - Alternative hierarchical approach:



Circuit Simplification



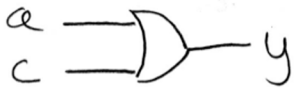
$$\downarrow$$

$$y = ((ab) + a) + c$$

$$\downarrow$$

$$= ab + a + c$$
$$= a(b + 1) + c$$
$$= a(1) + c$$
$$= a + c$$

$$\downarrow$$



1) original circuit (Transistors and/or Gates)

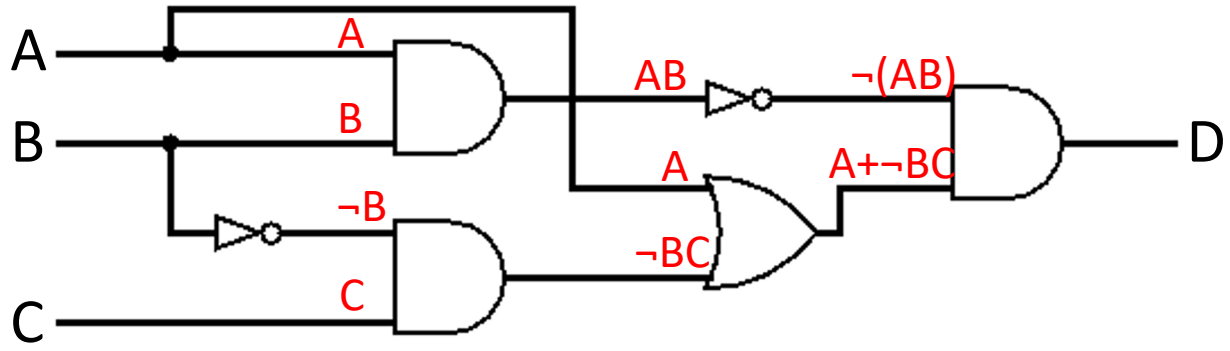
2) equation derived from original circuit

3) algebraic simplification

4) simplified circuit

Circuit Simplification Example (2/4)

- Simplify the following circuit:



- Start from left, propagate signals to the right

Arrive at D = ¬(AB)(A + ¬BC)

Circuit Simplification Example (3/4)

- Simplify Expression:

D = ¬(AB)(A + ¬BC)

= (¬A + ¬B)(A + ¬BC)

DeMorgan's

= ¬AA + ¬A¬BC + ¬BA + ¬B¬BC

Distribution

= 0 + ¬A¬BC + ¬BA + ¬B¬BC

Complementarity

= ¬A¬BC + ¬BA + ¬BC

Idempotent Law

= (¬A + 1)¬BC + ¬BA

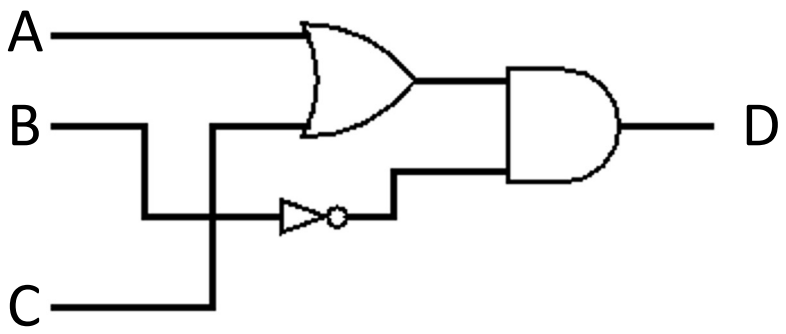
Which of these is "simpler"?

Circuit Simplification Example (4/4)

- Draw out final circuit:
  - D = ¬BC + ¬BA = ¬B(A + C)
    5 3

How many gates do we need for each?

- Simplified Circuit:

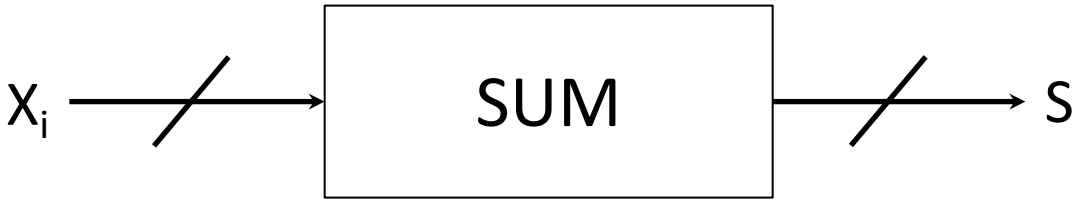


  - Reduction from 6 gates to 3!

Digital Logic - Sequential

Accumulator Example

An example of why we would need sequential logic



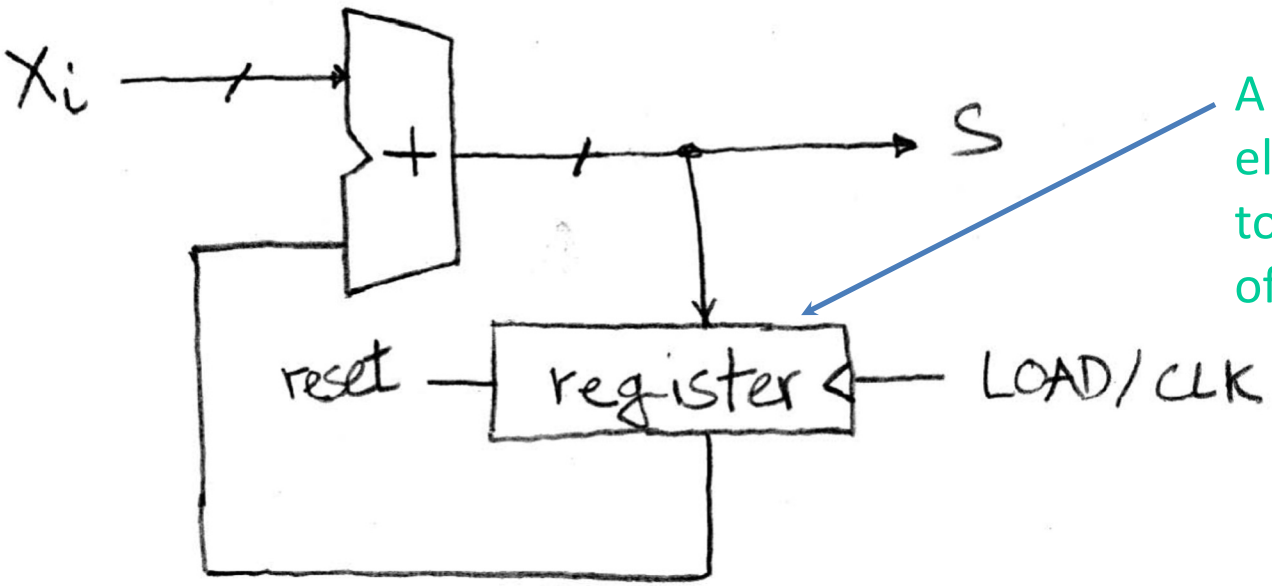
Want:

```
S=0;
for X1,X2,X3 over time...
    S = S + Xi
```

Assume:

- Each X value is applied in succession, one per cycle
- The sum since time 1 (cycle) is present on S

Second Try: How About This?



A *Register* is the state element that is used here to hold up the transfer of data to the adder

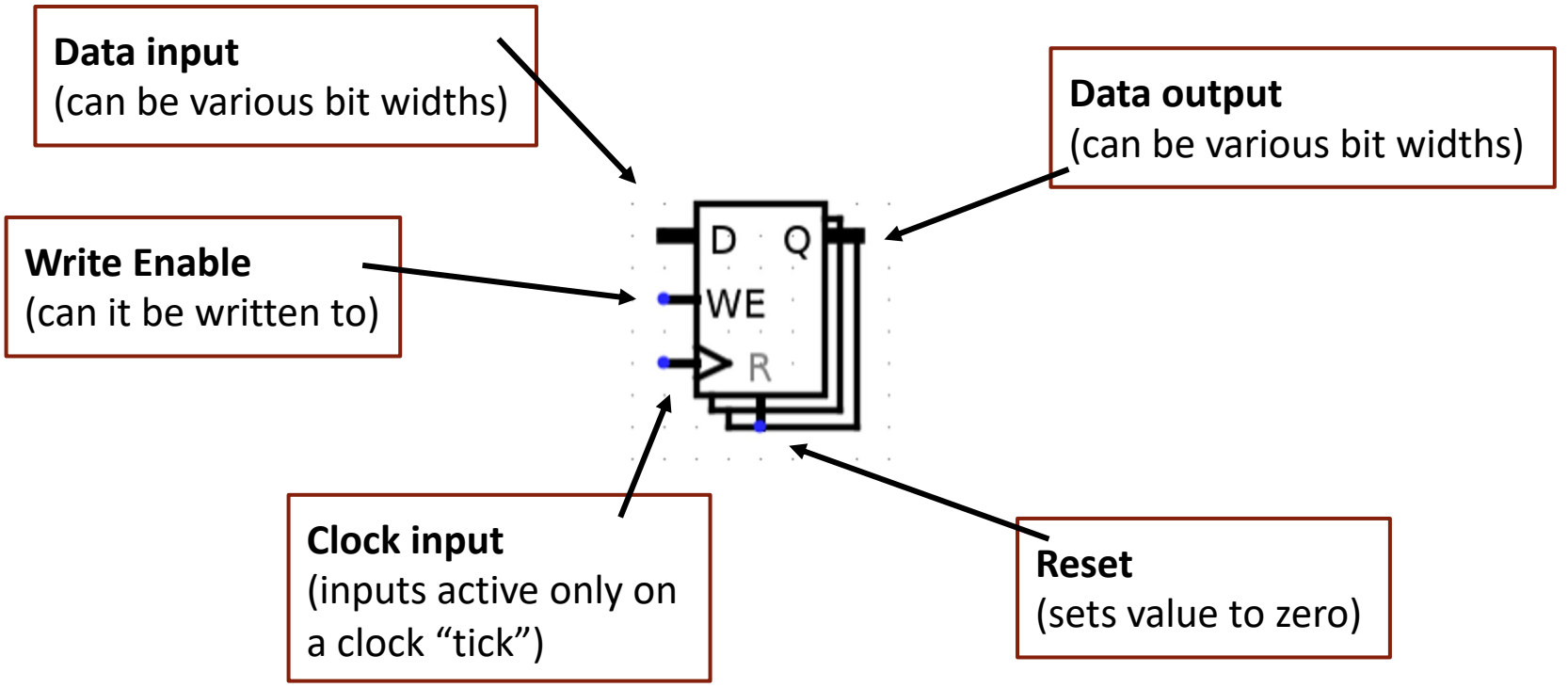
Uses for State Elements

- Place to store values for some amount of time:
  - Register files (like in RISCV)
  - Memory (caches and main memory)
- *Help control flow of information between combinational logic blocks*
  - State elements are used to hold up the movement of information at the inputs to combinational logic blocks and allow for orderly passage

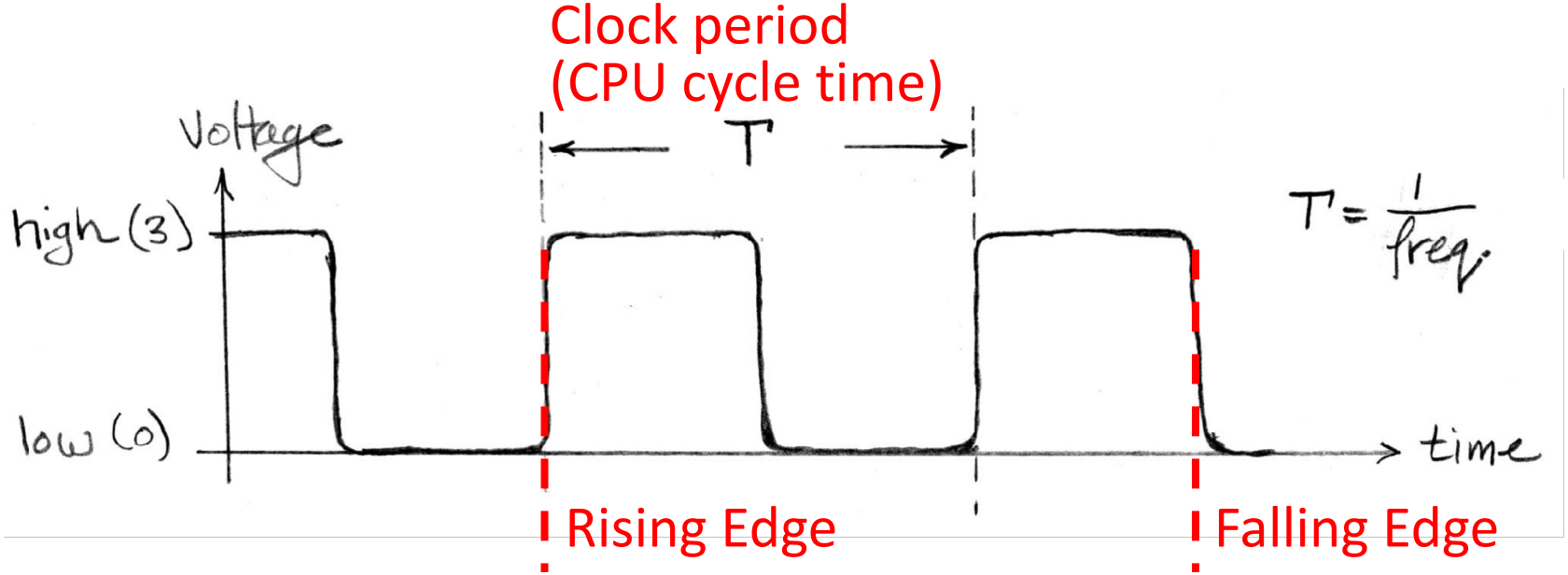
Registers

Same as registers in assembly:

- small memory storage locations



Signals and Waveforms: Clocks

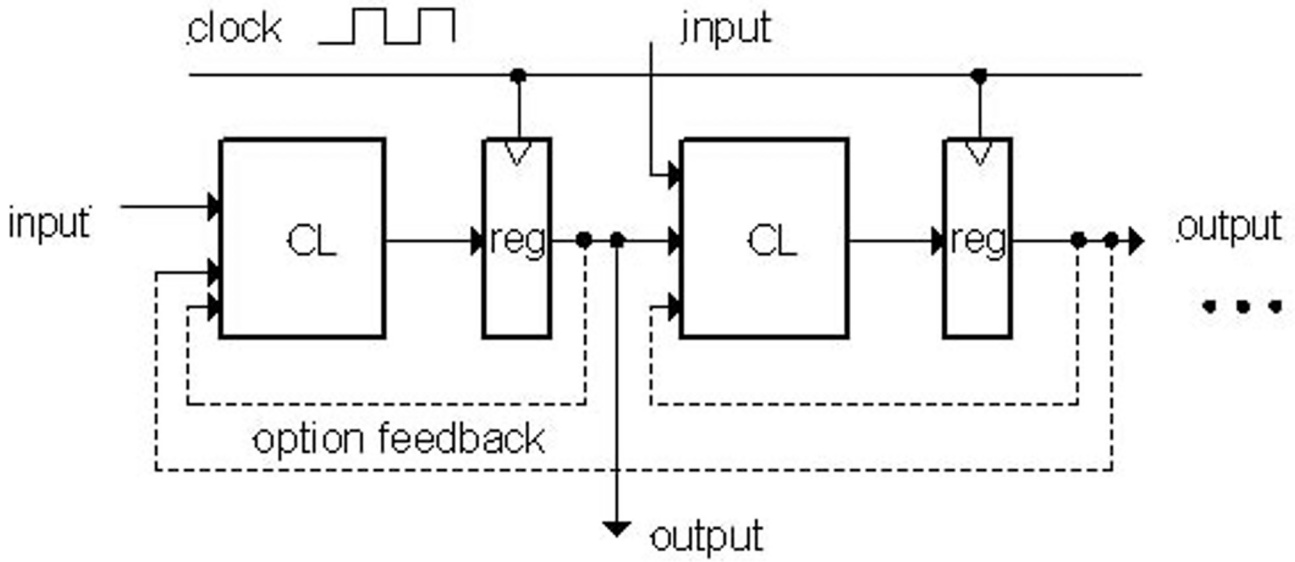


- Signals transmitted over wires continuously
- Transmission is effectively instantaneous
  - Implies that any wire only contains one value at any given time

Review of Timing Terms

- **Clock:** steady square wave that synchronizes system
- **Register:** several bits of state that samples on rising edge of Clock (positive edge-triggered); also has RESET
- **Setup Time:** when input must be stable *before* Clock trigger
- **Hold Time:** when input must be stable *after* Clock trigger
- **Clock-to-Q Delay:** how long it takes output to change from Clock trigger

Model for Synchronous Systems



- Combinational logic blocks separated by registers
  - Clock signal connects only to sequential logic elements
  - Feedback is optional depending on application
- How do we ensure proper behavior?
  - How fast can we run our clock?

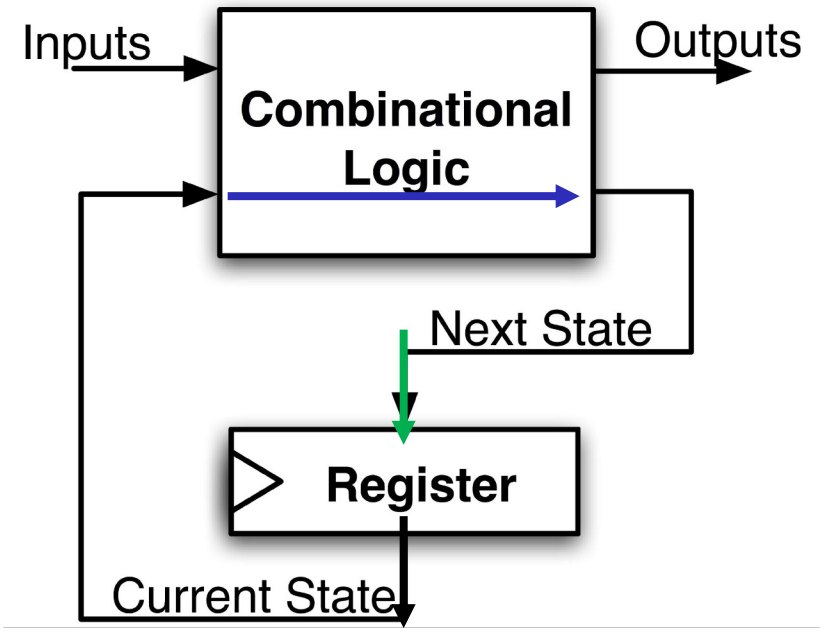
When can the input change?

- Needs to be stable for duration of setup time + hold time
- Often unstable until at least clock-to-q time has passed
  - Because register output isn't ready yet
- Needs to account for all combinational logic delay too

Maximum Clock Frequency

- What is the max frequency of this circuit?
  - Limited by how much time needed to get correct Next State to Register (t_{setup} constraint)

Assumes Max Delay > Hold Time



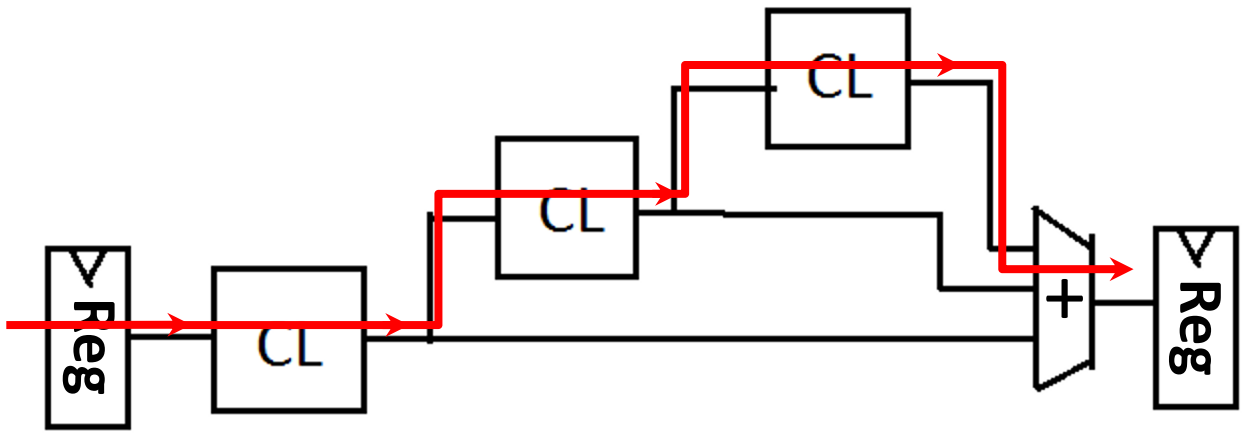
Max Delay = CLK-to-Q Delay
+ CL Delay
+ Setup Time

Min Period = Max Delay

Max Freq = 1/Min Period

The Critical Path

- The *critical path* is the longest delay between *any* two registers in a circuit
- The clock period must be *longer* than this critical path, or the signal will not propagate properly to that next register



Critical Path =

CLK-to-Q Delay

+ CL Delay 1

+ CL Delay 2

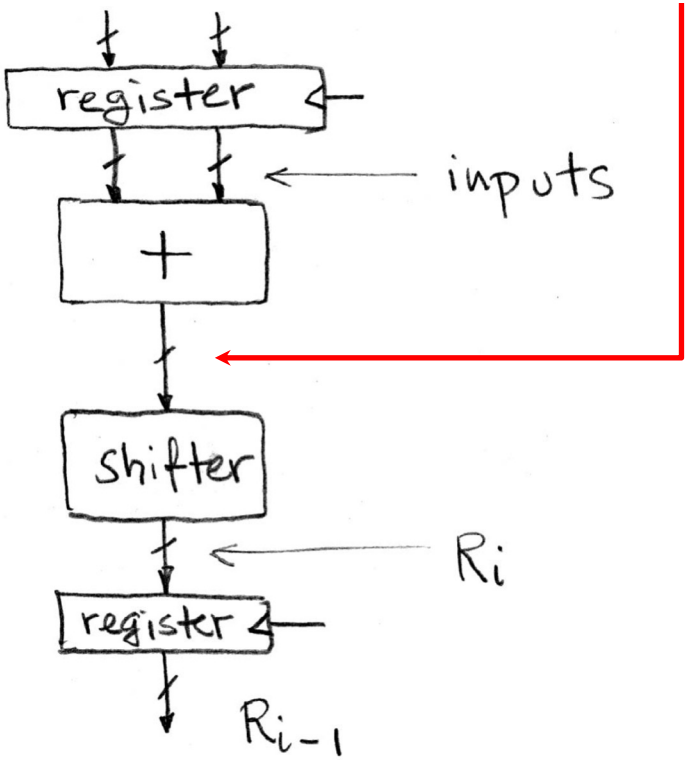
How do we go faster?

Pipelining!

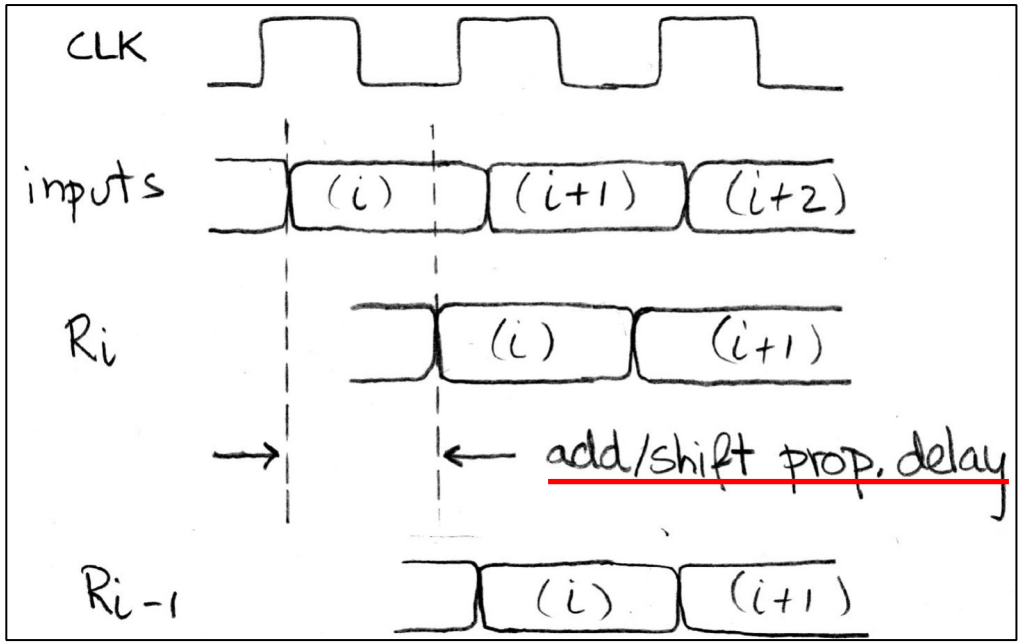
- Split operation into smaller parts and add a register between each one.

Pipelining and Clock Frequency (1/2)

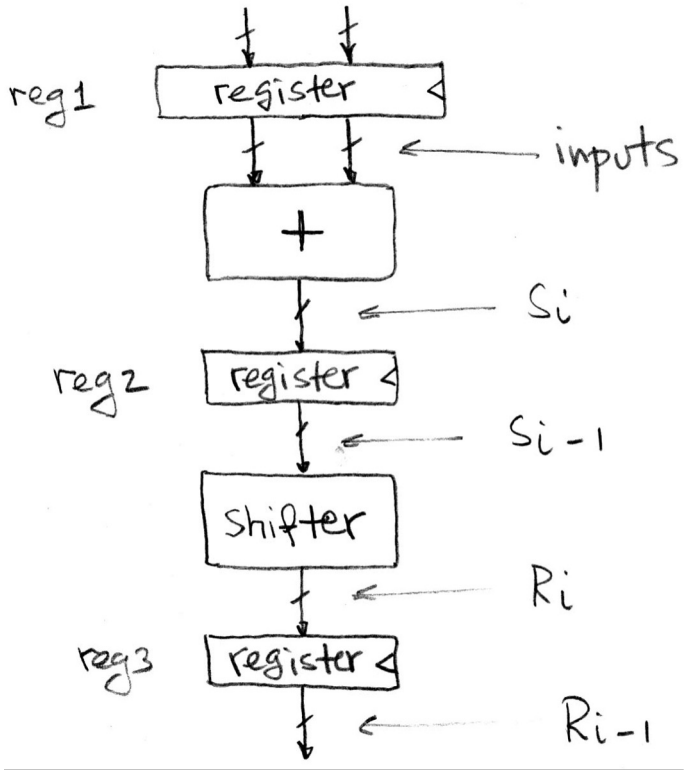
- Clock period limited by propagation delay of adder and shifter
  - Add an extra register to reduce the critical path!

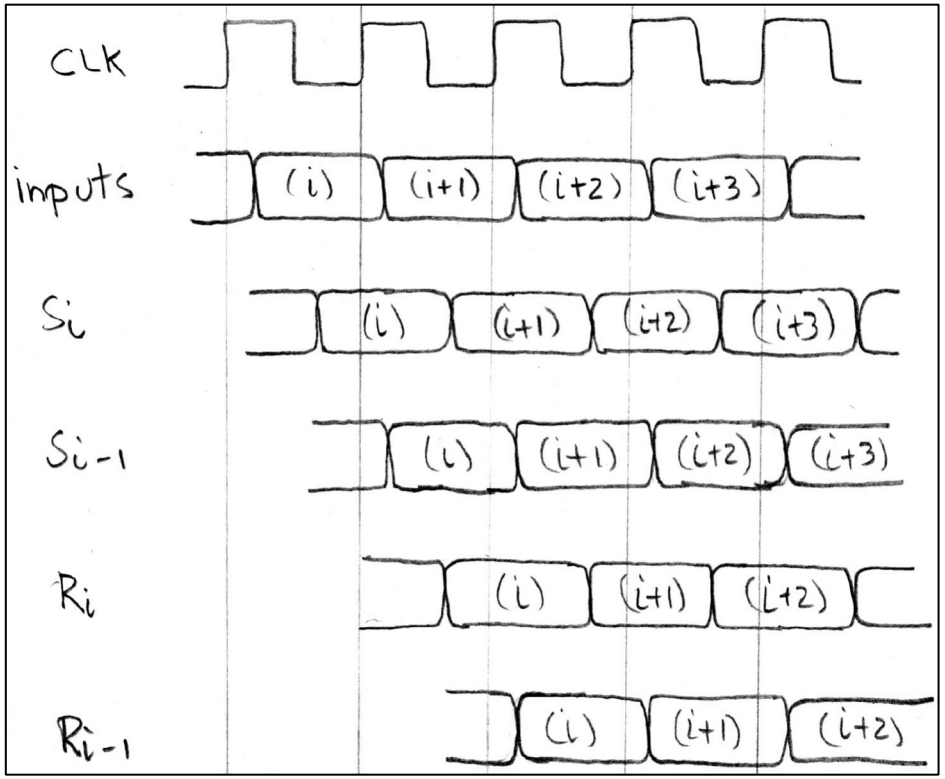


Timing:



Pipelining and Clock Frequency (2/2)





- Reduced critical path → allows higher clock freq.
- Extra register → extra (shorter) cycle to produce first output

Pipelining Basics

- By adding more registers, break path into shorter "stages"
  - Aim is to reduce critical path
  - Signals take an additional clock cycle to propagate through *each* stage
- New critical path must be calculated
  - Affected by placement of new pipelining registers
  - Faster clock rate → higher throughput (outputs)
  - More stages → higher startup latency
- Pipelining tends to improve performance
  - More on this (and application to CPUs) later