

We acknowledge and pay our respects to the Kaurna people,
the traditional custodians whose ancestral lands we gather on.

We acknowledge the deep feelings of attachment and relationship of the
Kaurna people to country and we respect and value their past, present
and ongoing connection to the land and cultural beliefs.



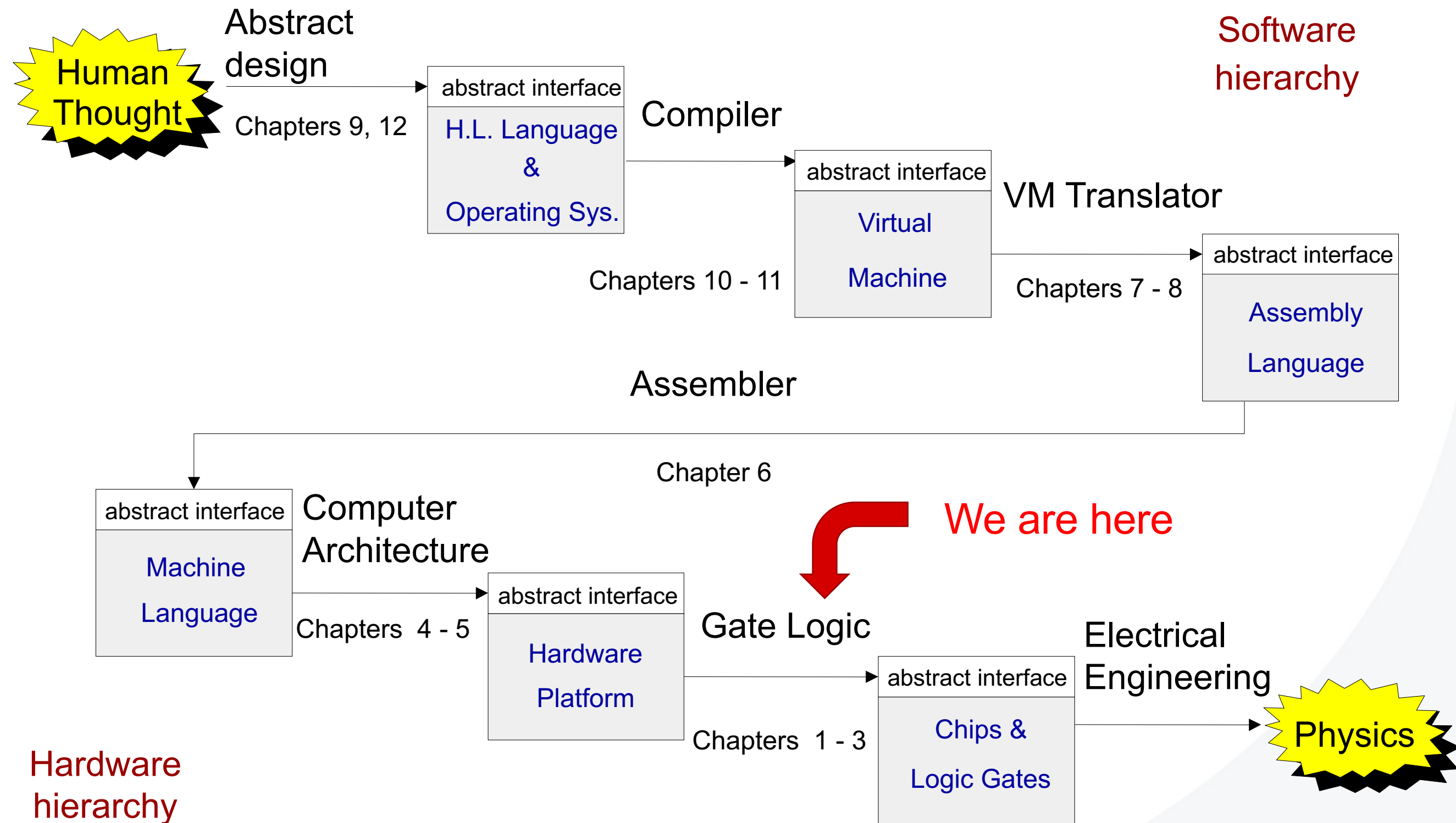
THE UNIVERSITY
of ADELAIDE

Computer Systems

Lecture 03: Boolean Arithmetic
& Sequential Logic



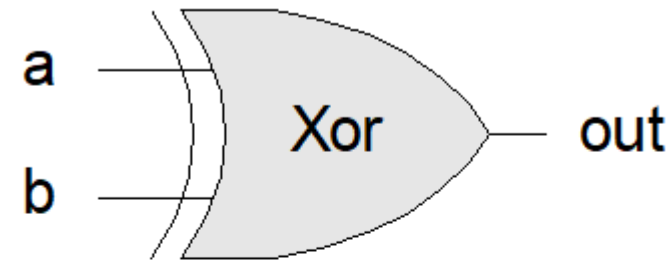
Our Journey



(Abstraction–implementation paradigm)

Review: Gate Logic

You will see this in exam!

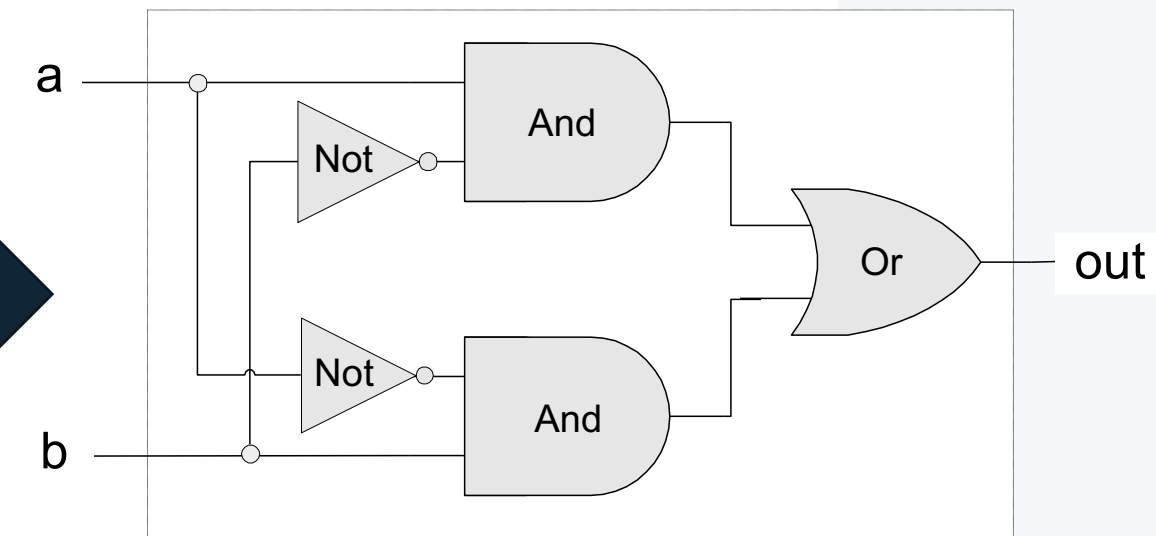


a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

Truth table



$$\text{out} = (\bar{a} \cdot b) + (a \cdot \bar{b})$$



Implementation using
And, Or, Not

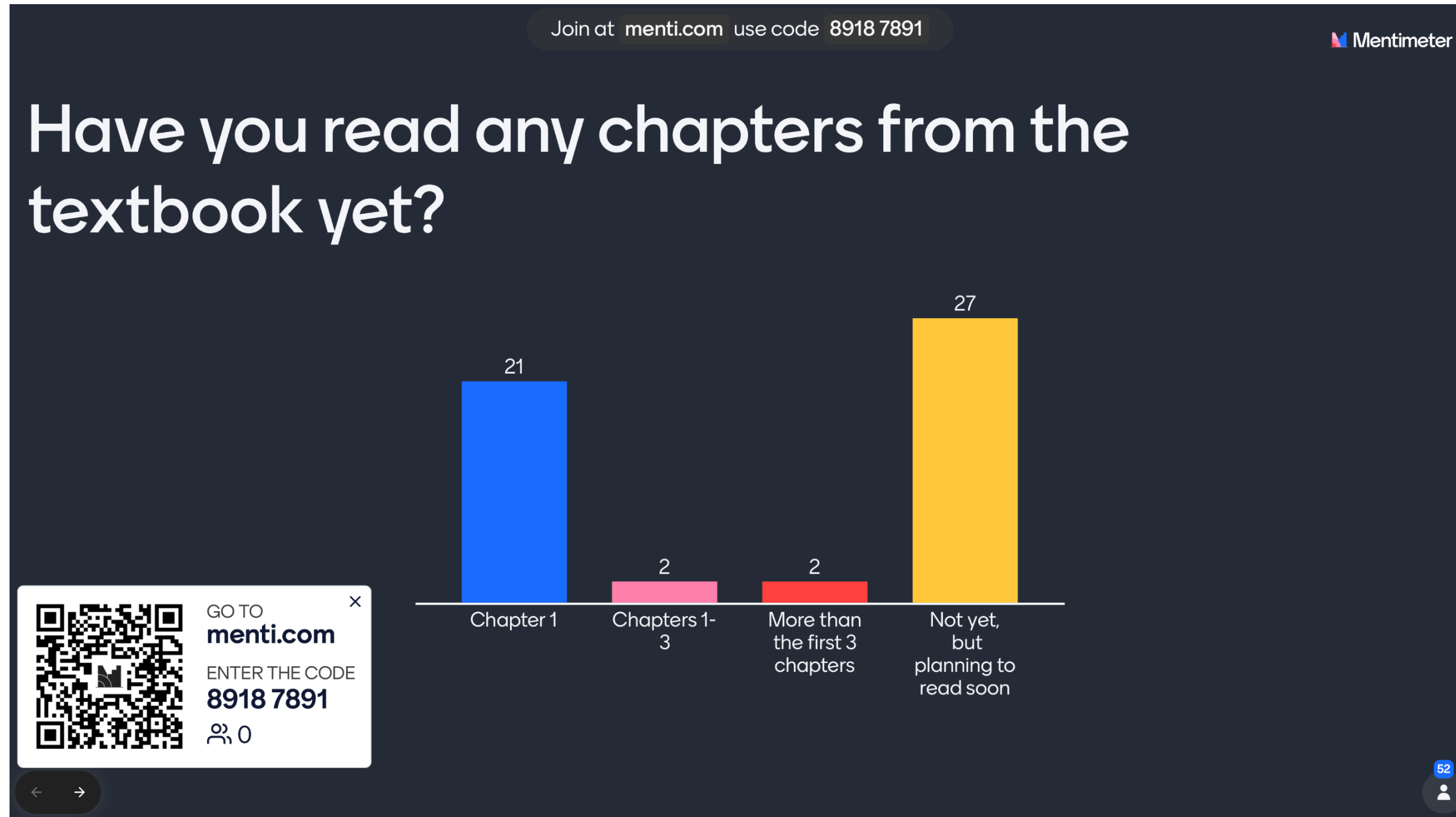
Boolean expression
In canonical form,
Our focus is SoP

Poll

<https://www.mentimeter.com/app/presentation/alvkpntxtc2c33naiezmn1wh7o2hvejr/pojbumnwpvo9>



Last week:



Binary Arithmetic



Number Representation

See more examples on Thursday

How do we represent numbers in the computer?

- Our logic gates can only handle 2 possible values; 0 or 1
- For example, a 4-bit binary number: 1 0 1 1



Number Representation

See more examples on Thursday

How do we represent numbers in the computer?

- Our logic gates can only handle 2 possible values; 0 or 1

- For example, a 4-bit binary number: $\begin{array}{cccc} & \nearrow & & \nwarrow \\ & 1 & 0 & 1 & 1 \\ \text{MSB} & & & & \text{LSB} \end{array}$

Weight: $2^3 \ 2^2 \ 2^1 \ 2^0$

$$1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 11$$



Number Representation: Unsigned Binary

Unsigned Binary	Decimal
1111	15
1110	14
1101	13
1100	12
1011	11
1010	10
1001	9
1000	8
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0

- An n-bit unsigned binary can represent 2^n numbers.
- From 0 to $2^n - 1$.
- Can not represent negative numbers.
- *How do we represent negative numbers when we don't have a +/- sign?*

Number Representation: One's Complement

One's Complete	Decimal
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	-0
1110	-1
1101	-2
1100	-3
1011	-4
1010	-5
1001	-6
1000	-7



Mirror image

- An n-bit one's complement can represent $2^n - 1$ numbers (consider $-0 = 0$).
- From $-(2^{n-1} - 1)$ to $(2^{n-1} - 1)$.
- Able to represent negative numbers.
- Redundant -0 .

Number Representation: Two's Complement

One's Complete	Decimal		Two's Complete	Decimal
0111	7		0111	7
0110	6		0110	6
0101	5		0101	5
0100	4		0100	4
0011	3		0011	3
0010	2		0010	2
0001	1		0001	1
0000	0		0000	0
1111	-0	✗	1111	-1
1110	-1	→	1110	-2
1101	-2	→	1101	-3
1100	-3	→	1100	-4
1011	-4	→	1011	-5
1010	-5	→	1010	-6
1001	-6	→	1001	-7
1000	-7	→	1000	-8

Shift up by one place

- An n-bit two's complement can represent 2^n numbers.
- From $-(2^{n-1})$ to $(2^{n-1} - 1)$.
- Able to represent negative numbers.
- One representation of 0.



Binary Addition

Assuming a 4-bit system:

$$\begin{array}{r} 0001 \\ + 0101 \\ \hline 00110 \end{array}$$

No overflow $1 + 5 = 6$

$$\begin{array}{r} 1011 \\ + 0111 \\ \hline 10010 \end{array}$$

Overflow $-5 + 7 = 2$

- Algorithm: the same as in decimal addition
- Overflow (MSB carry out) may need to be dealt with – we usually ignore it.

Binary Addition

Assuming a 4-bit system:

$$\begin{array}{r} a \rightarrow \boxed{0 \ 0 \ 0 \ 1} \\ + \quad \boxed{0 \ 1 \ 0 \ 1} \leftarrow b \\ \hline \text{carry} \rightarrow \boxed{0 \ 0 \ 0 \ 1} \\ \hline \text{sum} \rightarrow \boxed{0 \ 0 \ 1 \ 1 \ 0} \end{array}$$

No overflow $1 + 5 = 6$

$$\begin{array}{r} \quad \quad 1 \ 0 \ 1 \ 1 \\ + \quad 0 \ 1 \ 1 \ 1 \\ \hline 1 \ 0 \ 0 \ 1 \ 0 \end{array}$$

Overflow $-5 + 7 = 2$

- Algorithm: the same as in decimal addition
- Overflow (MSB carry out) may need to be dealt with – we usually ignore it.

Binary Addition

- How do we know if a two's complement number is negative?
 - The Most Significant Bit is 1
- To negate a number, flip all the bits and add 1
- If you flip all the bits in a number x, you get $-x - 1$
- Sometimes the result of an add operation is wrong!
 - When the MSB of a two's complement number changes.
 - Using subtract to compare numbers needs to account for this effect *Note this in your assignments.*

$$\begin{array}{r} 1\ 0\ 1\ 1 \\ + 1\ 0\ 1\ 1 \\ \hline 1\ 0\ 1\ 1\ 0 \end{array}$$

Bad overflow $-5 + -5 = 6$

$$\begin{array}{r} 0\ 1\ 0\ 1 \\ + 0\ 1\ 0\ 0 \\ \hline 0\ 1\ 0\ 0\ 1 \end{array}$$

Bad overflow $5 + 4 = -7$

Binary Addition

- How do we know if a two's complement number is negative?
 - The Most Significant Bit is 1
- To negate a number, flip all the bits and add 1
- If you flip all the bits in a number x, you get $-x - 1$
- Sometimes the result of an add operation is wrong!
 - When the MSB of a two's complement number changes.
 - Using subtract to compare numbers needs to account for this effect *Note this in your assignments.*

e.g., $5 = 0101$

$-6 = 1010$

$-5 = 1010 + 1 = 1011$

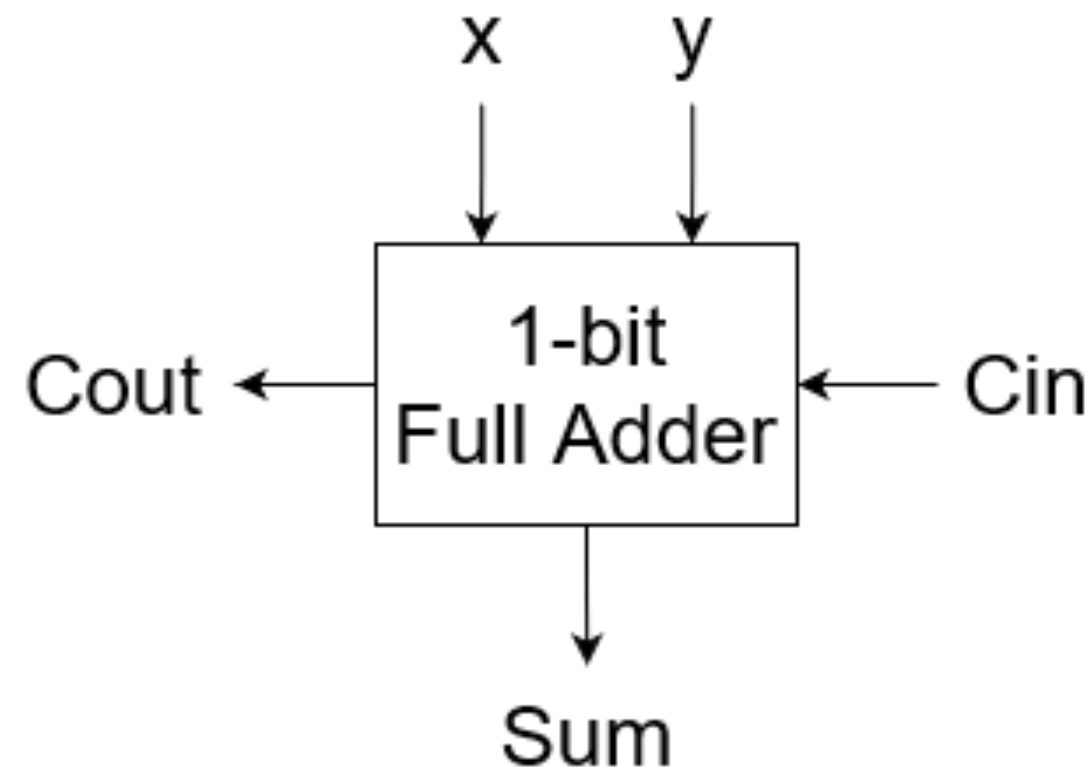
$$\begin{array}{r} 1\ 0\ 1\ 1 \\ + 1\ 0\ 1\ 1 \\ \hline 1\ 0\ 1\ 1\ 0 \end{array}$$

Bad overflow $-5 + -5 = 6$

$$\begin{array}{r} 0\ 1\ 0\ 1 \\ + 0\ 1\ 0\ 0 \\ \hline 0\ 1\ 0\ 0\ 1 \end{array}$$

Bad overflow $5 + 4 = -7$

Exercise: Canonical Form Boolean Expression for 1-bit Full Adder



x	y	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



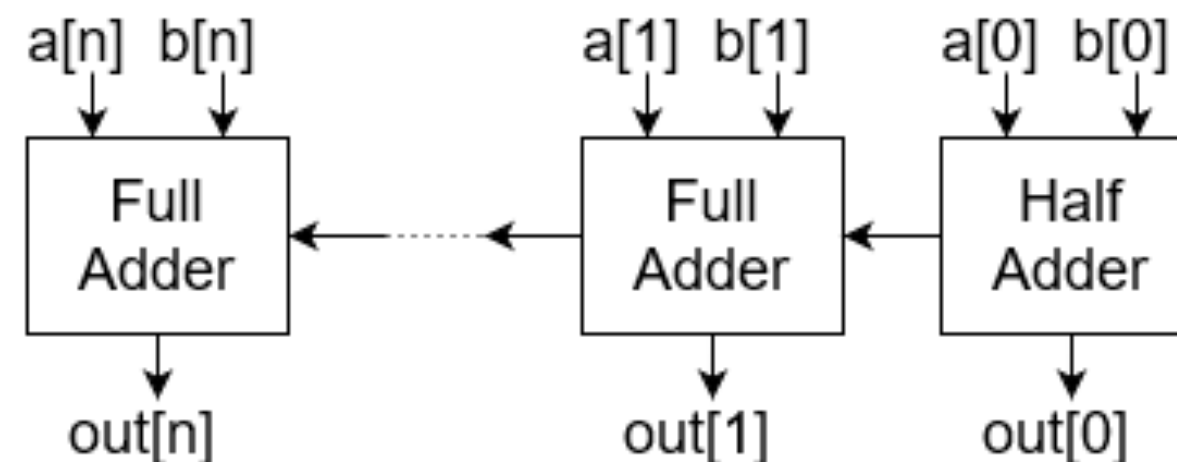
Building an Adder chip

You will build this in workshop!



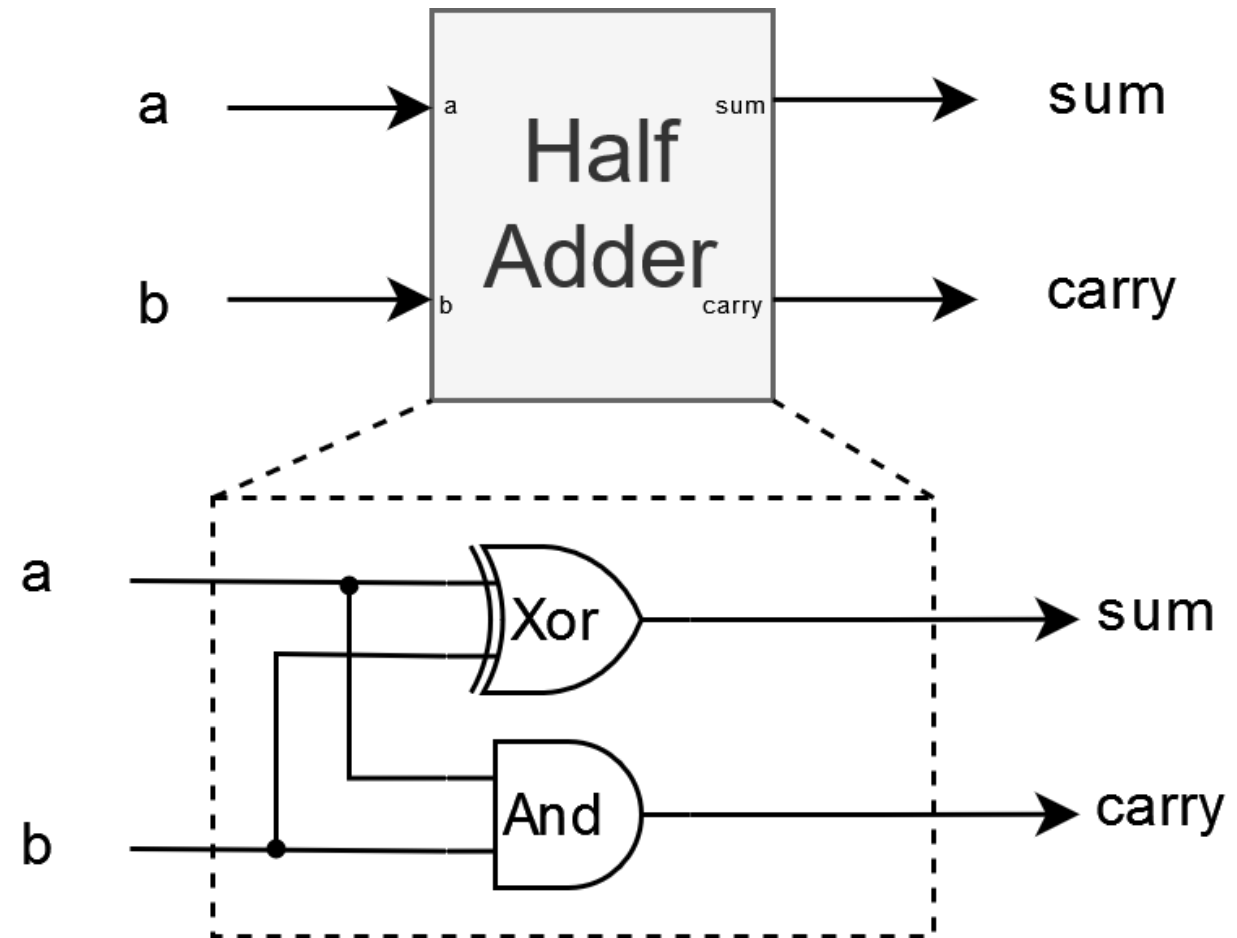
Adder: a chip designed to add two integers

- **n-bit adder:** add two n -bit numbers.
 - **1-bit half adder:** add 2 bits (we only need one at the LSB)
 - **1-bit full adder:** add 3 bits (we need $n-1$ of these)



Half adder (designed to add 2 bits)

a	b	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



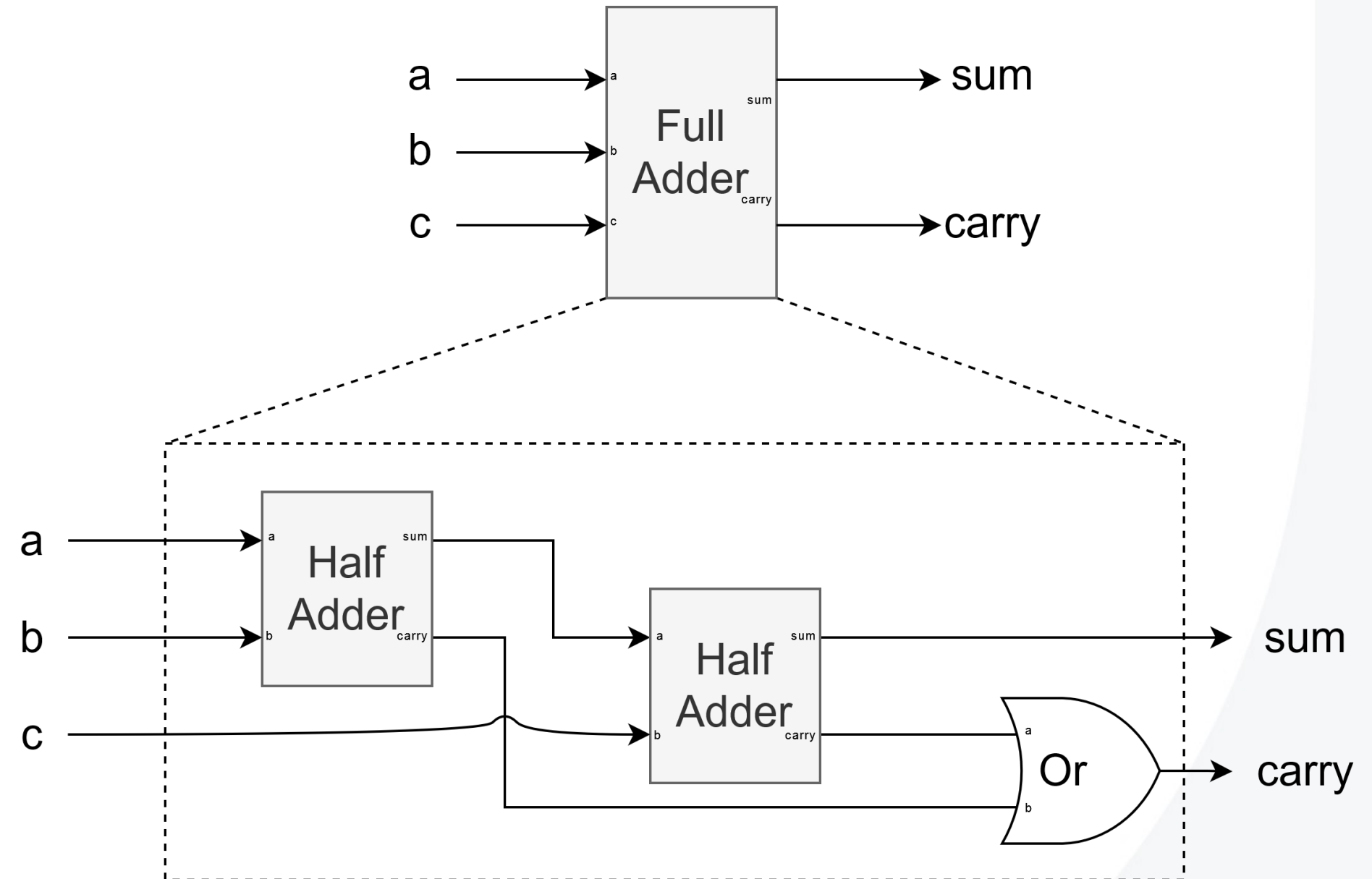
A half adder can be built from an Xor and an And

the sum column matches Xor

the carry columns matches And

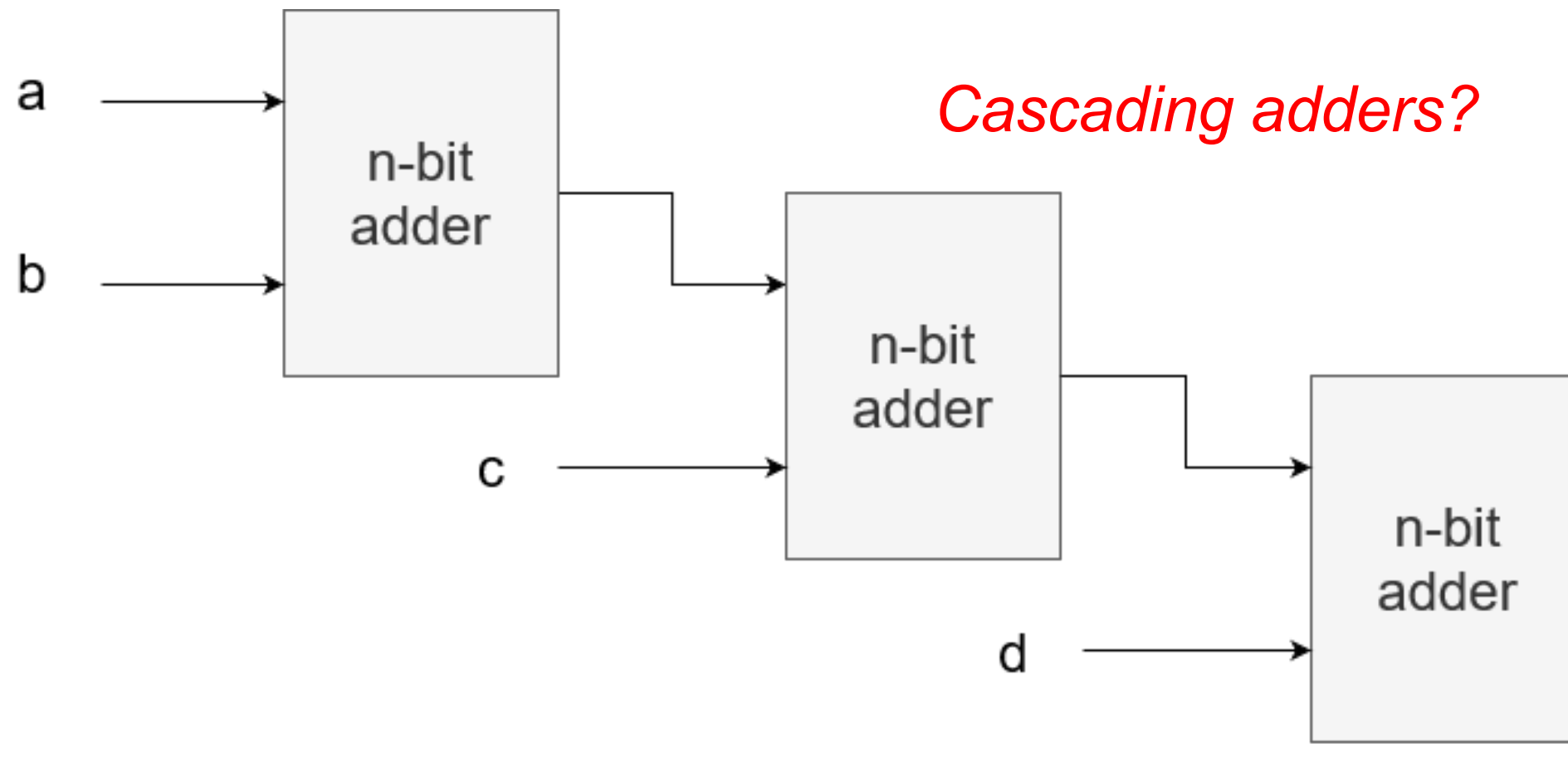
Full adder (designed to add 3 bits)

a	b	c	sum	carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Implementation: can be based on two half-adder and an Or gates.

What if we want to add more than 2 number?

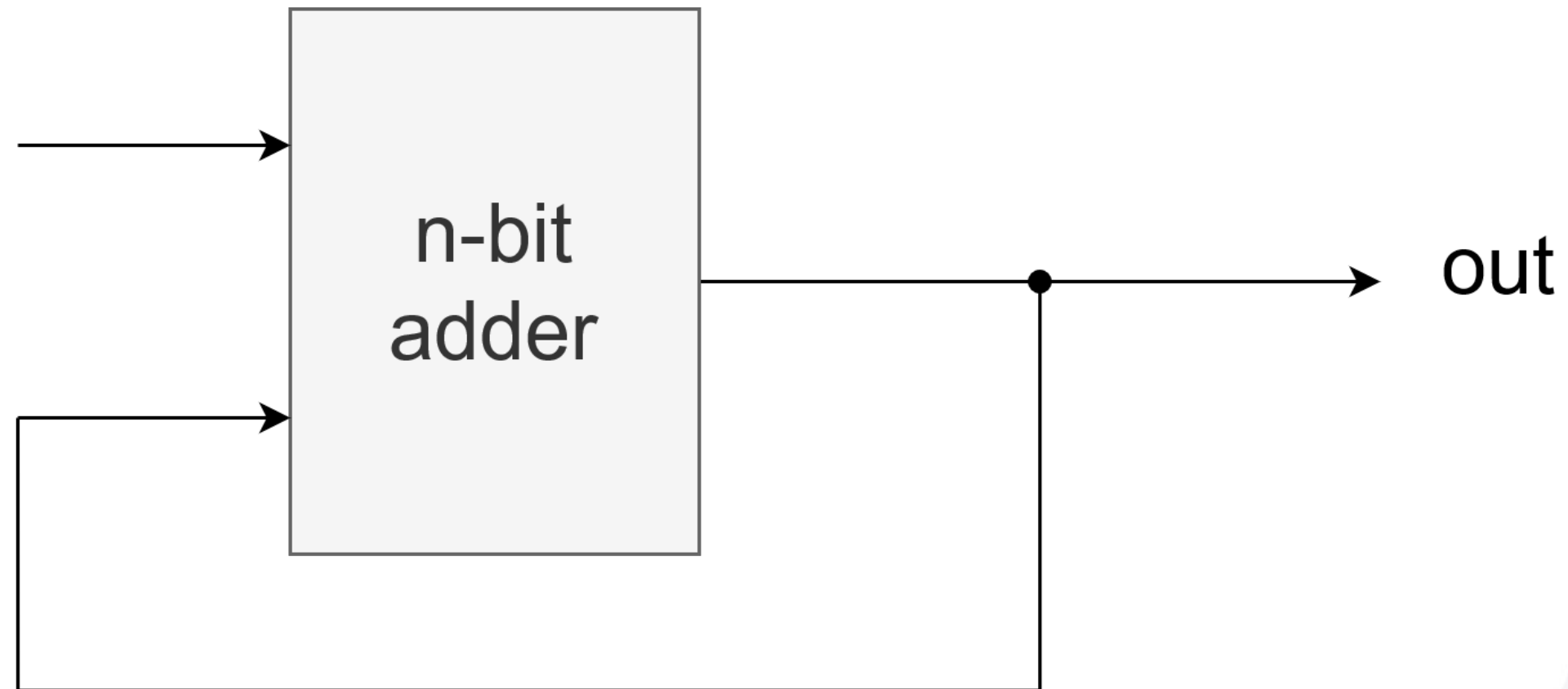


No!

- You can do this, but not recommended.
- Waste of chip area, power and design efforts only achieve a simple function.

Feed the output back and reuse the adder?

Sequentially
feed a, b, c, d...



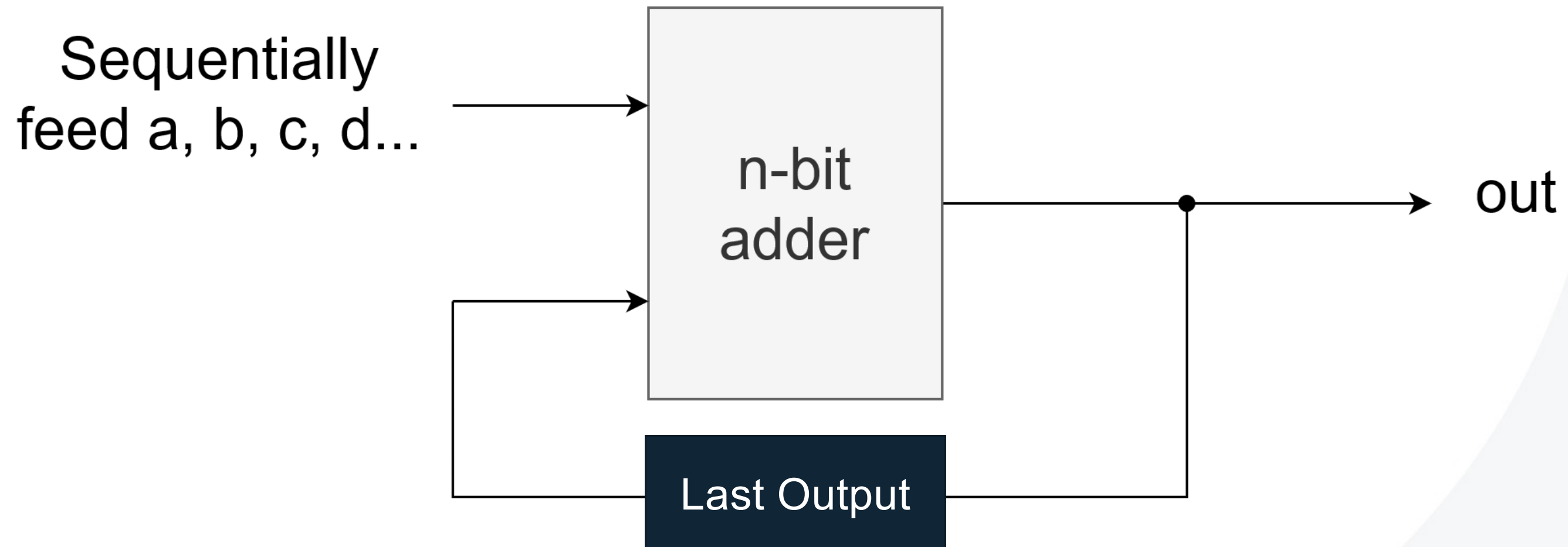
- The combinational logic (e.g., the adder) can not maintain its state.
- The output will start to change when the input is changed.
- We need some mechanism to memorise the current state.

Sequential Logic



Feed the output back and reuse the adder?

- Need to delay the output until we're ready to feed the next input
- Need to then store the new output until the next step.

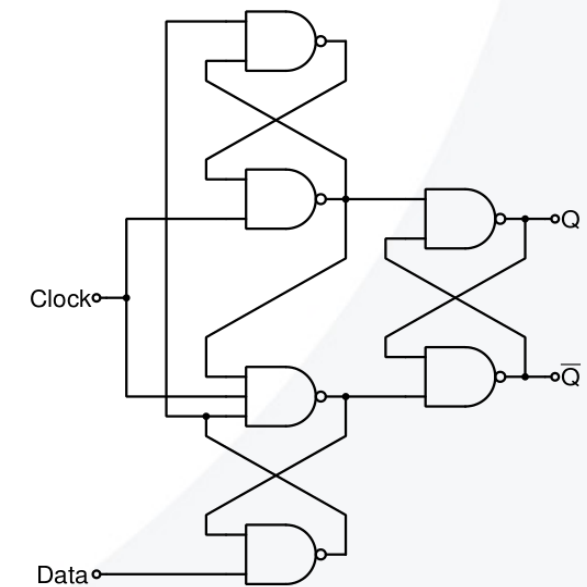
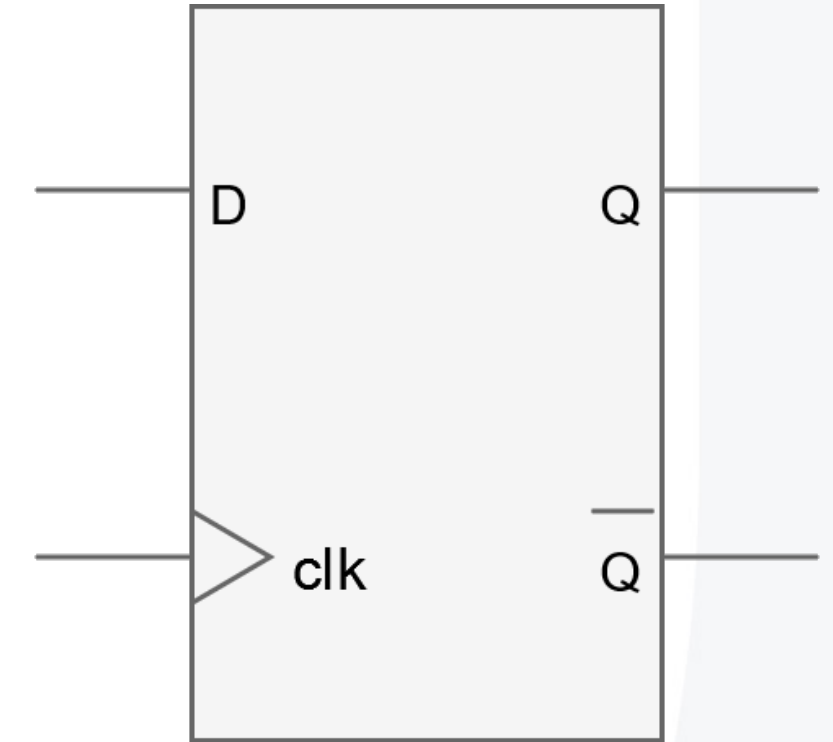


How can we store the previous output until we're ready?
How can we trigger the next step?

DFFs

All sequential chips can be based on one low-level sequential gate, called a 'data flip flop', or DFF.

- DFF takes an input and holds/delays that until the next clock 'tick'.
- The output Q of the DFF is whatever the input D was at the previous clock 'tick'.
- DFFs can be made from NAND gates using some clever feedback loops. (we don't expect you to memorise this).

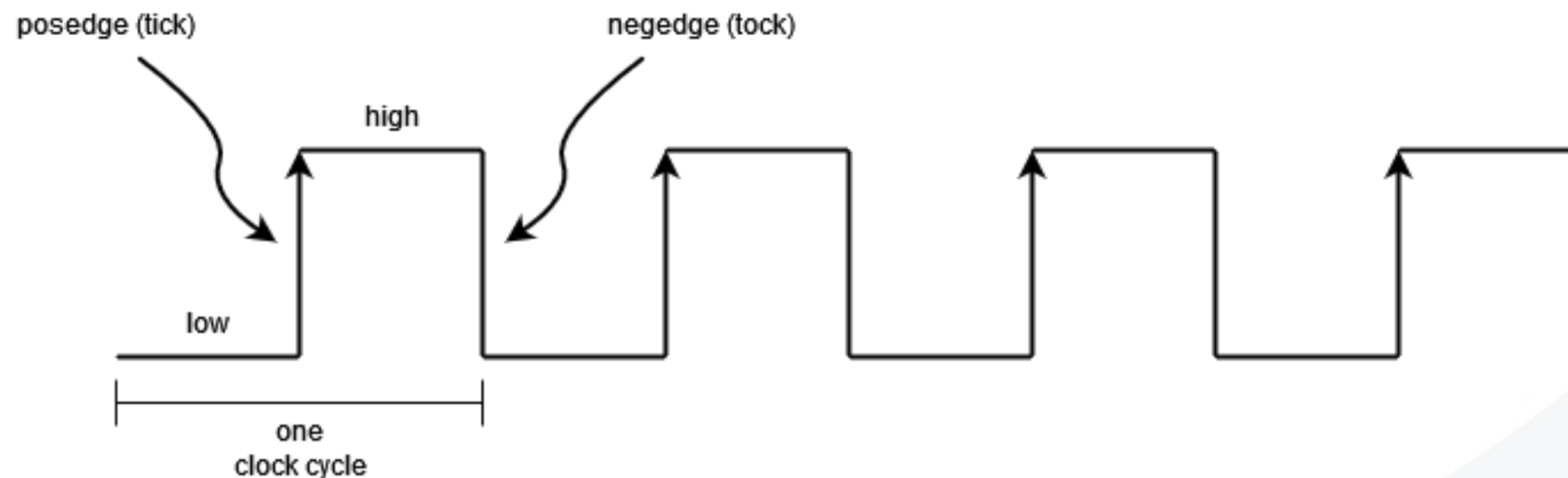


What's a clock in this context?

In computer systems, the clock *signal* is used to allow us to reflect what happens as the state of hardware changes over time.

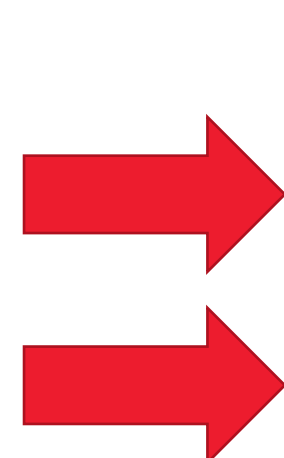
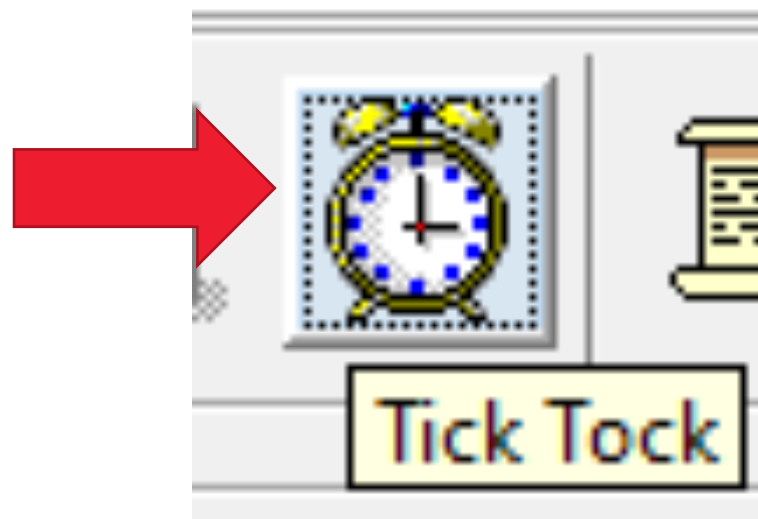
The hardware clock signal periodically oscillates between low and high states.

The transition from low to high is called posedge (or tick), the transition from high to low is called negedge (or tock).



Clocks

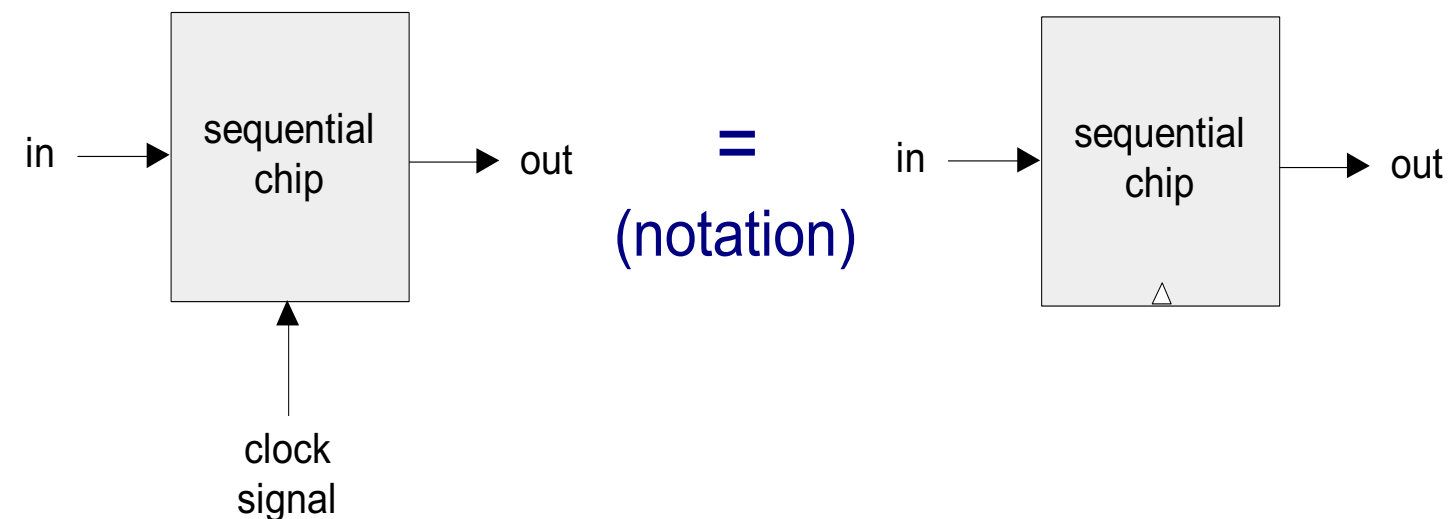
- The speed of the clock is going to affect the speed of everything in the computer as a clock cycle is the smallest unit of time in which anything can change.
- In actual hardware, we have specific components and circuits to generate and transmit clock signal.
- In the Nand2tetris Hardware Simulator, user can manually feed the clock signal, or use a test script contains tick/tock command.



```
set load 1,  
tick,  
output;  
tock,  
output;
```

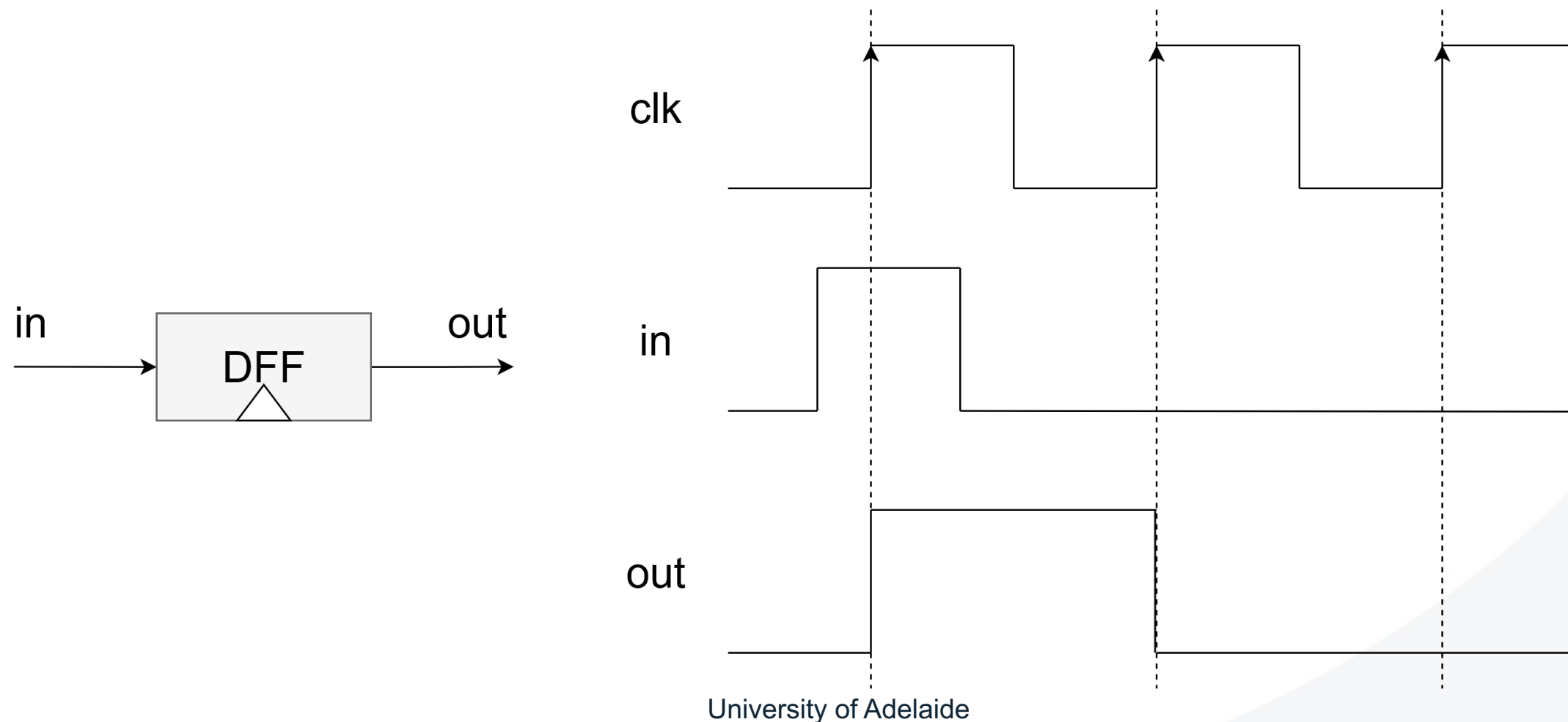
Using the clock

We're not worrying about the implementation of clock circuit in this course. You can assume all clock signals are commonly connected, running on the same master clock signal.



Problem of using DFF to maintain state

If we using a DFF alone its state can only maintain for one clock cycle.

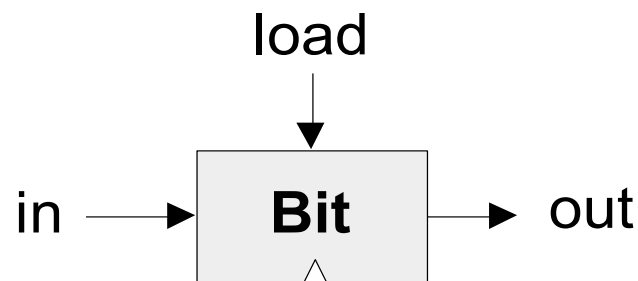


Let's build a bit! (1-bit register)

We want to be able to:

Change the state of the bit

Retain that state until we want to change it.



if load(t-1) then out(t)=in(t-1)
else out(t)=out(t-1)

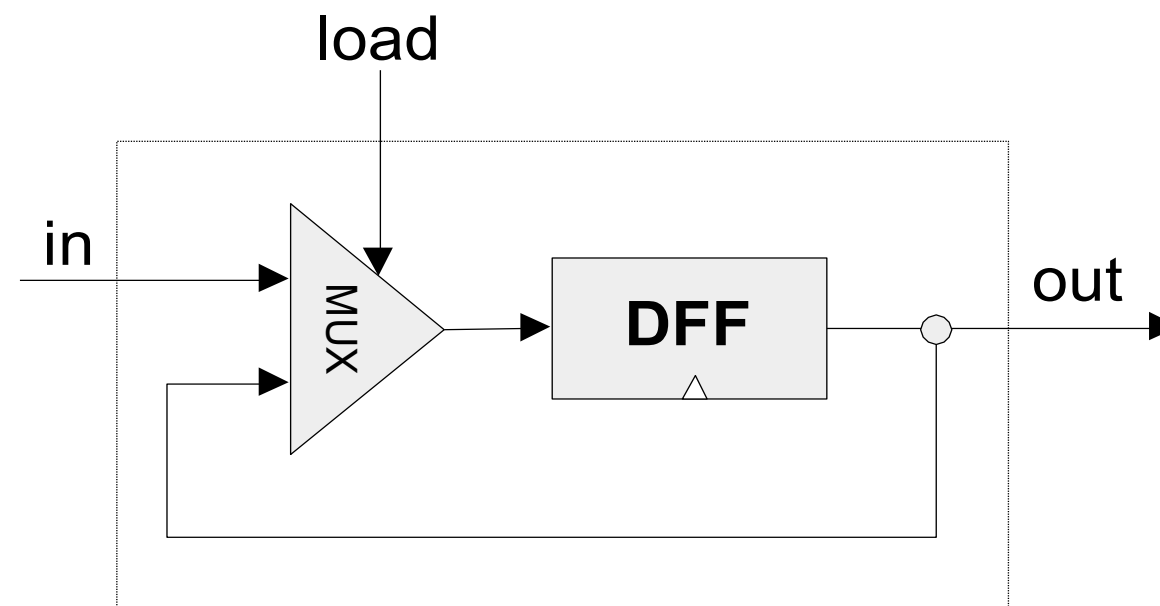
**Now we can tell the bit when to change, by controlling the load signal.
Can we build this from the DFF and gates we already know?**

Bit (1-bit register) implementation

Notice how we use the load signal.

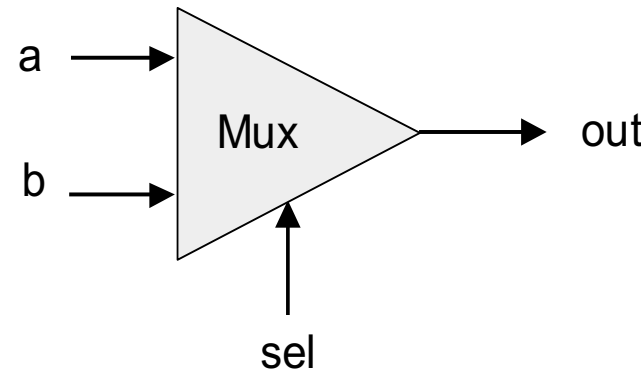
Now we have read logic and write logic.

Implementation



Canonical Form – Sum of Product (SoP)

a	b	sel	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

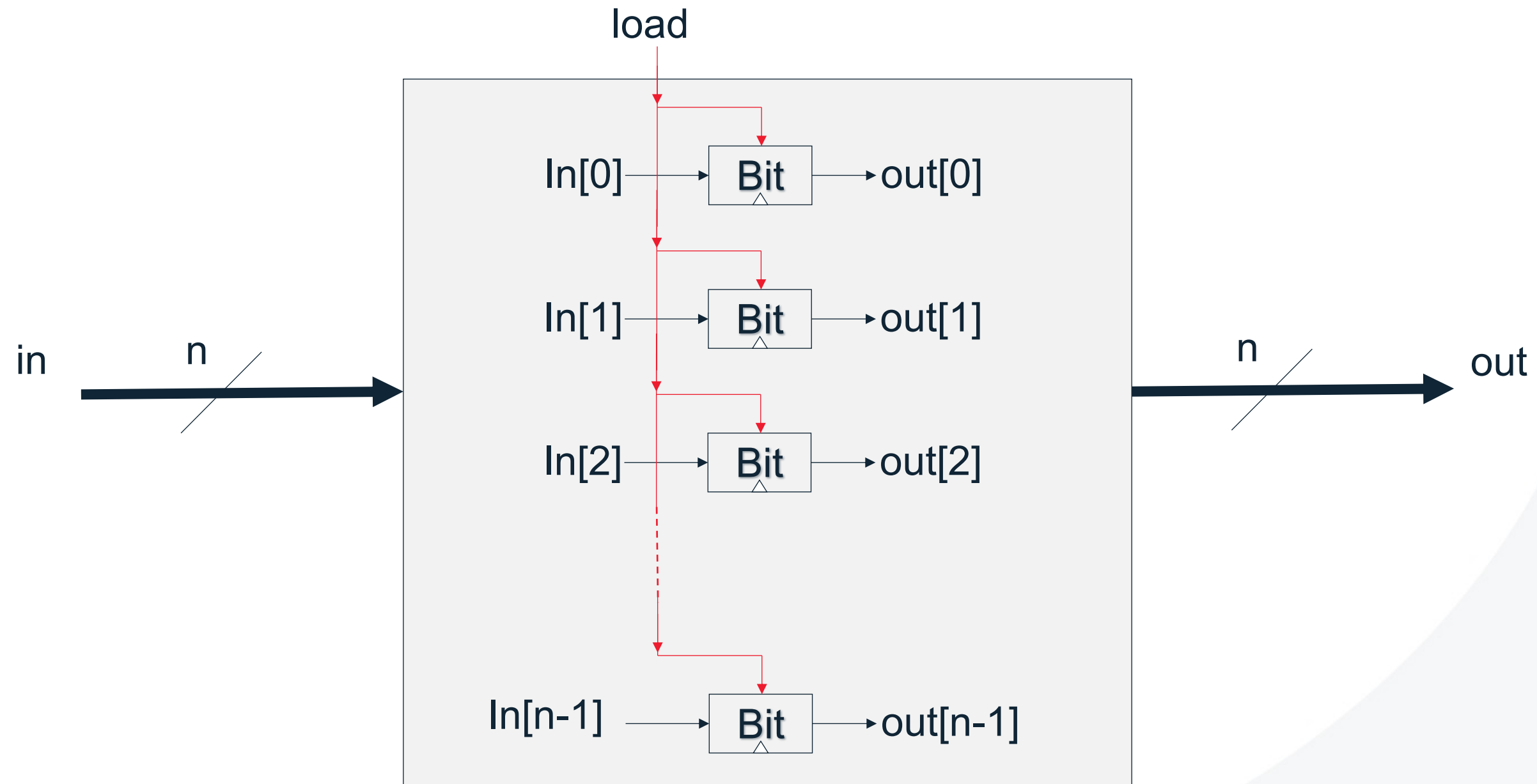


$$\text{out} = (\bar{a} \cdot b \cdot \text{sel}) + (a \cdot \bar{b} \cdot \overline{\text{sel}}) + (a \cdot b \cdot \overline{\text{sel}}) + (a \cdot b \cdot \text{sel})$$

1. Find all lines with out = 1, ignore all lines with out = 0.
2. Write inputs in product (And), if the input is 1 write as it is, otherwise, write in bar (Not) form.
3. Repeat 2. for each line with out = 1.
4. Use sum (Or) to connect each term.

Bigger registers

We can wire up n 1-bit registers in parallel to build a n -bit register.



Summary

- Boolean Logic can be used to perform basic binary arithmetic such as addition.
- We can combine basic 1-bit adder chips to add larger numbers.
- Sequential Logic allows us to perform operations that rely on previous results and state.
- DFFs allow us to store state by delaying output. This allows us to build registers and other data storage structures.



This Week

- Workshops start this week (week 2).
- Review Chapters 2 & 3 of the Textbook.
- Week 2 Quiz due before start of Thursday's Lecture.
We will be discussing quiz questions during Thursday's session
- Practical Assignment 1 due Sunday.



Questions?

<https://www.mentimeter.com/app/presentation/alvkpntxtc2c33naiezmn1wh7o2hvejr/sducq53b3hm9>

