COMP2300/6300

Computer Organisation and Prog tion

Digital Logic

Dr Charles Martin

Semester 1, 2022

Week 1: Boolean algebra

George Boole

It starts with a thought

An Investigation of the Laws of T[illegible]*Which are Founded the Mathematical Theories of Logic and Probabilities* by **Georg**[illegible]54

You can still buy it from Amazon

Boole's big idea: **true & false are all you need**

What is Boolean algebra?

Algebra is the study of mathema[illegible]ols and the rules for manipulating these symbols; it's about **variables** like *a* and *b* a[illegible]**ors** like ∧ (binary *and*), ∨ (binary *or*), ¬ (unary *not*, sometimes represented with an [illegible]g. $\overline{q}$).

Boolean means that all variables & expressions can take one of two values. We can call them **true** and **false**, **1** and **0**, or **Mary** and **Mengyuan**; it doesn't matter.

Boolean algebra builds expressions with these basic building blocks, e.g.

¬(a ∧ b) ∨ c

this is all revision from MATH1005, so we're gonna speed through

Truth tables

Truth tables are just a convenien[illegible]umerating all the of the possible values our variables can take. If you’ve got n [illegible], you need 2^n rows in your truth table (why?)

Here’s an example with 2 variables:

a	b	a∧b
T	T	T
T	F	F
F	T	F
F	F	F

Other handy Boolean operators

$a \longrightarrow b = (\neg a \lor b)$	a **implies** b
$(a \equiv b) = (a \land b) \lor (\neg a \land \neg b)$	a **equivalent to** b
$(a \oplus b) = (a \land \neg b) \lor (\neg a \land b)$	a **exclusive or**/**xor** b
$\neg(a \land b) = (\neg a \lor \neg b)$	a **not and**/**nand** b
$\neg(a \lor b) = (\neg a \land \neg b)$	a **not or**/**nor** b

You can reduce any Boolean expression to only NAND or only NOR operators (**try it and see!**).

Logic functions

You know about functions from [illegible] here's a two-argument function of $x, y \in \mathbf{R}$

$$f(x, y) = x^2 sin(y)$$

We can have functions of Boolean variables a and b as well:

$$g(a, b) = \ldots$$

talk

$f(x, y) = \ldots$

$g(a, b) = \ldots$

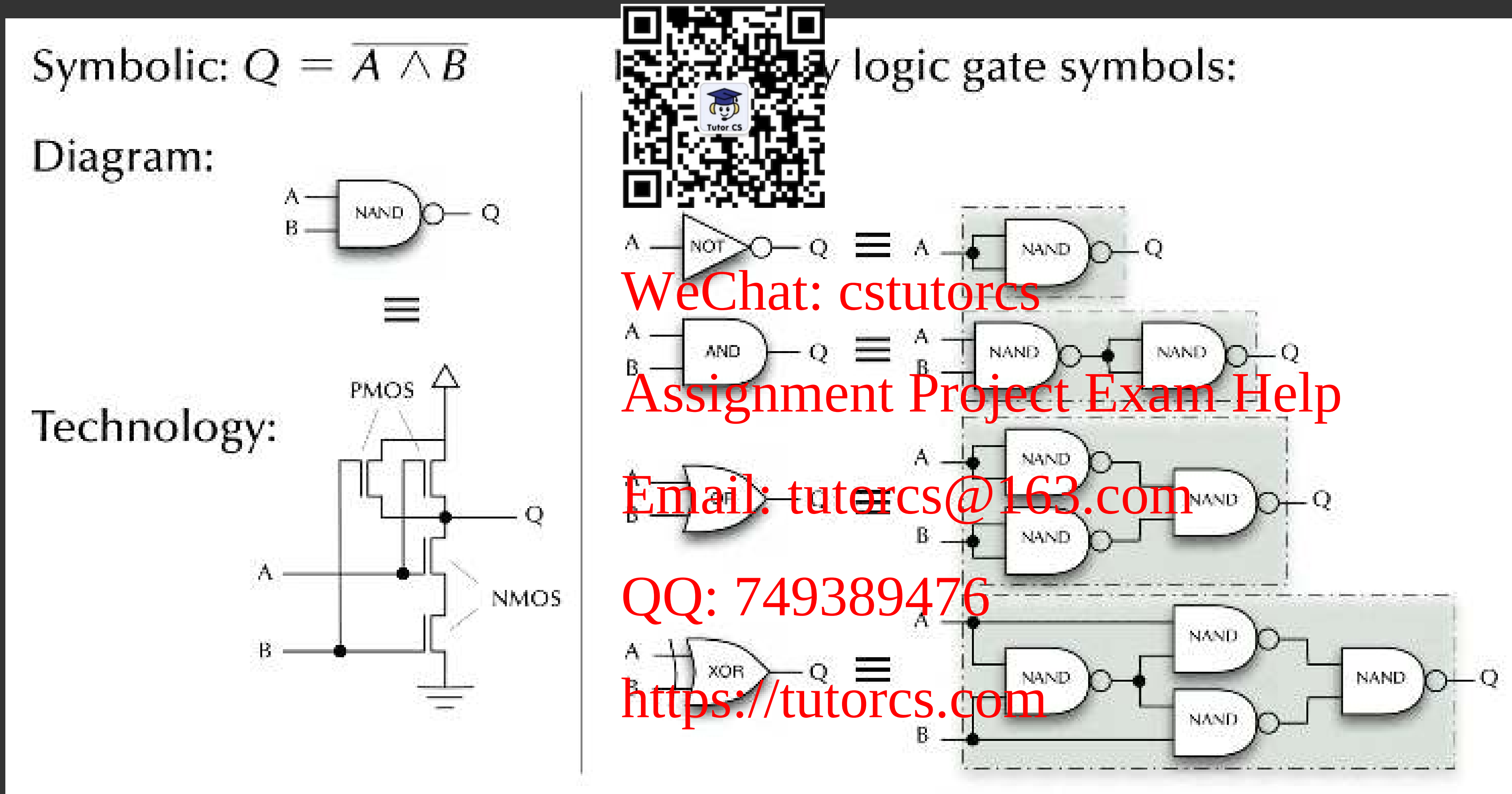
Can you think of anything we can do with the Boolean function $g(a, b)$ that we can't do with the real-valued function $f(x, y)$?

Full binary operator truth table

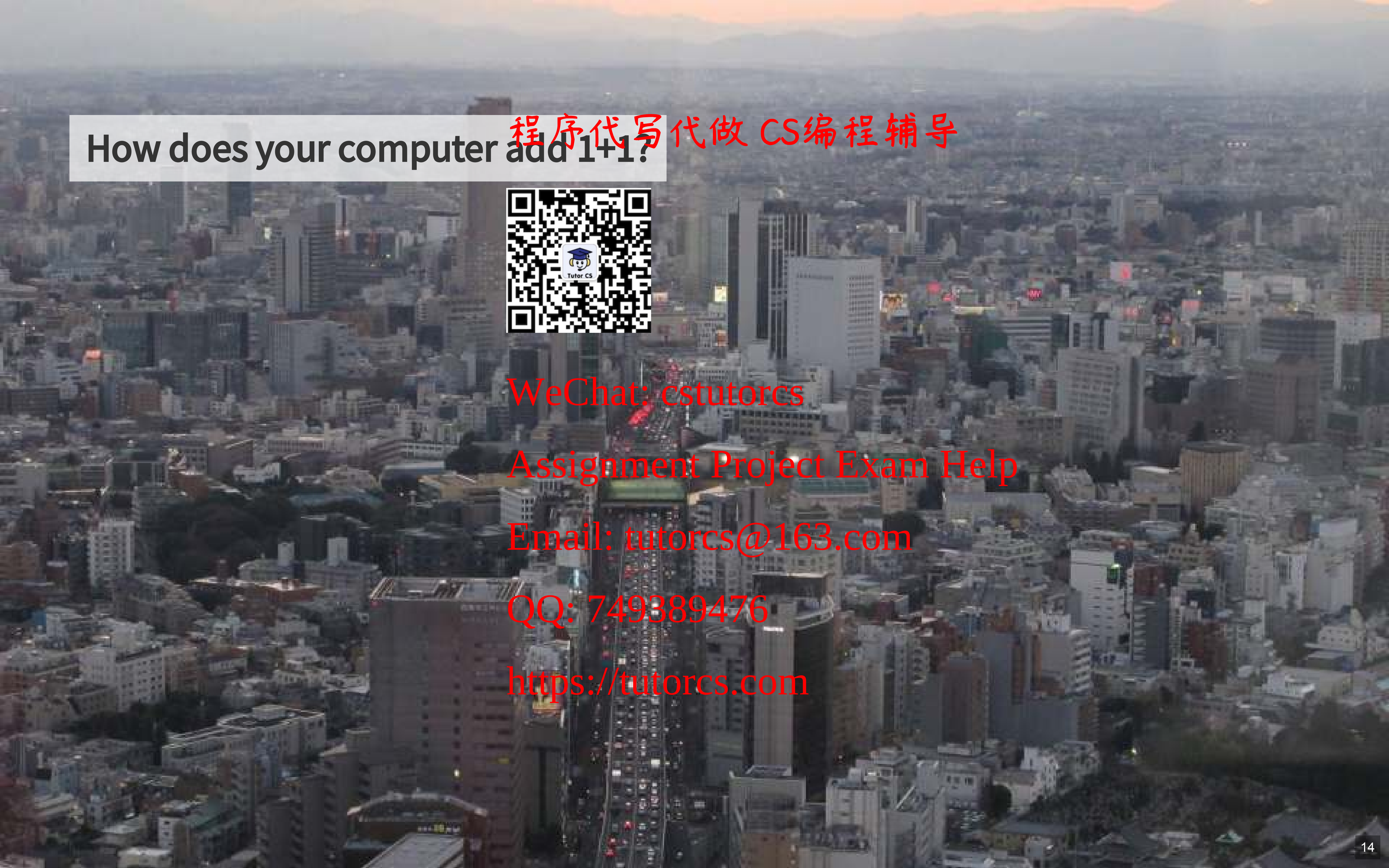
	Inputs a, b				Function	Name	Sum of products	NAND	Don't cares
a	F	F	T	T					
b	F	T	F	T					
q	F	F	F	F	False	Constant False			a, b
	F	F	F	T	$a \wedge b$	AND		$\overline{\overline{a \wedge b} \wedge \overline{a \wedge b}}$	
	F	F	T	F	$\overline{a \to b}$	NOT-IMPLICATION	$(a \wedge \bar{b})$		
	F	F	T	T	a	IDENTITY a			b
	F	T	F	F	$\bar{a} \wedge b$				
	F	T	F	T	b	IDENTITY b			a
	F	T	T	F	$a \oplus b$	EXCLUSIVE-OR, XOR	$(a \wedge \bar{b}) \vee (\bar{a} \wedge b)$		
	F	T	T	T	$a \vee b$	OR		$\overline{\overline{a \wedge a} \wedge \overline{b \wedge b}}$	
	T	F	F	F	$\overline{a \vee b}$	NOT-OR, NOR	$(\bar{a} \wedge \bar{b})$		
	T	F	F	T	$a = b$	EQUALITY, EQ	$(a \wedge b) \vee (\bar{a} \wedge \bar{b})$		
	T	F	T	F	$\bar{b}$	INVERSE b			a
	T	F	T	T	$a \vee \bar{b}$				
	T	T	F	F	$\bar{a}$	INVERSE a		$\overline{a \wedge a}$	b
	T	T	F	T	$a \to b$	IMPLICATION	$\bar{a} \vee b$		
	T	T	T	F	$\overline{a \wedge b}$	NOT-AND, NAND	$\bar{a} \vee \bar{b}$		
	T	T	T	T	True	Constant True			a, b
	Output q								

What does this have to do with my microbit?

Logic gates


Symbolic: $Q = \overline{A \wedge B}$
Diagram:
A
B
NAND
Q
≡
Technology:
PMOS
NMOS
Q
A
B
y logic gate symbols:
NOT
AND
OR
XOR
NAND

How does your computer add 1+1?



How would a 5yo do it?

Number representations

Remember that an integer can be represented in a different “base” (or “radix”), e.g. binary (base-2), octal (base-8), hexadecimal (base-16) or the familiar decimal (base-10).

Decimal	0
Hex	0x 0000 0000
Binary	0b 0000 0000 0000 0000 0000 0000 0000 0000

Note: hex & binary padded to 32-bit, negative numbers represented with 32-bit two’s complement

Combinational logic

Let’s start simple: 1+1

Consider the Boolean function $s = a + b$ (the s is short for *sum*). How would we put this in a (pseudo) truth table?

a	b	s
0	0	0
0	1	1
1	0	1
1	1	2

Not quite…

This doesn't really work because … have *three* distinct values (0, 1 and 2) in Boolean algebra. But what if we just cons… …olumn" of the addition?

a	b	s	
0	0	0	
0	1	1	
1	0	1	
1	1	0	(carry the 1)

Add a c (carry) column

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

bit == binary digit

But what is $s(a, b)$?

The truth table *is* a complete spe[illegible]nction $s(a, b)$ that we're interested in, but it doesn't tell us how to express s u[illegible]ules we looked at earlier.

a	b	s	c	s minterms	c minterms
0	0	0	0		
0	1	1	0	¬a ∧ b	
1	0	1	0	a ∧ ¬b	
1	1	0	1		a ∧ b

s = (a ∧ ¬b) ∨ (¬a ∧ b) = a ⊕ b

c = a ∧ b

So far...

Combinational Logic:

lets us make a Boolean expressions for any truth-table

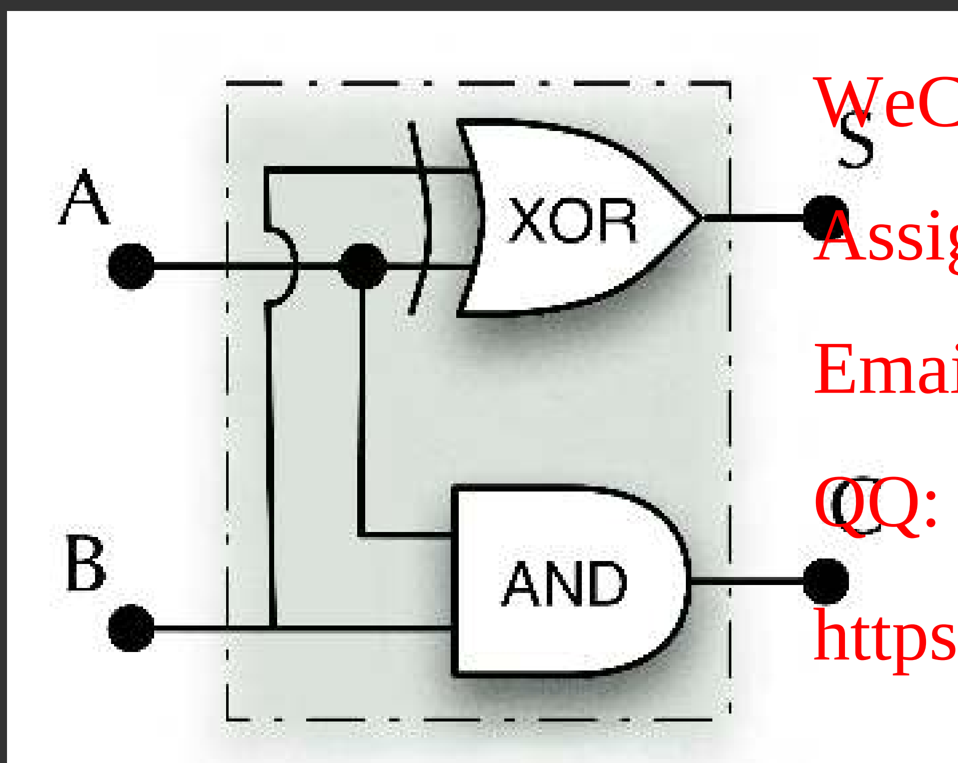
Boolean Algebra:

lets us simplify Boolean expressions to something manageable

Half-adder

$s = a \oplus b$

$c = a \wedge b$



DLS: Digital Logic Simulator

DLS is a time-driven event-based [illegible] ay 3-value digital logic simulator.

There's both **desktop** (cheap) & v[illegible] versions.

DLS isn't compulsory for the course, but it's a nice way for me to demo things.

Demo: Half-adder

Full-Adder

What about carry *in*?

What's missing with the half-add[illegible]*n* (ci) as well as carry out (co).

a	b	ci	s	co
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

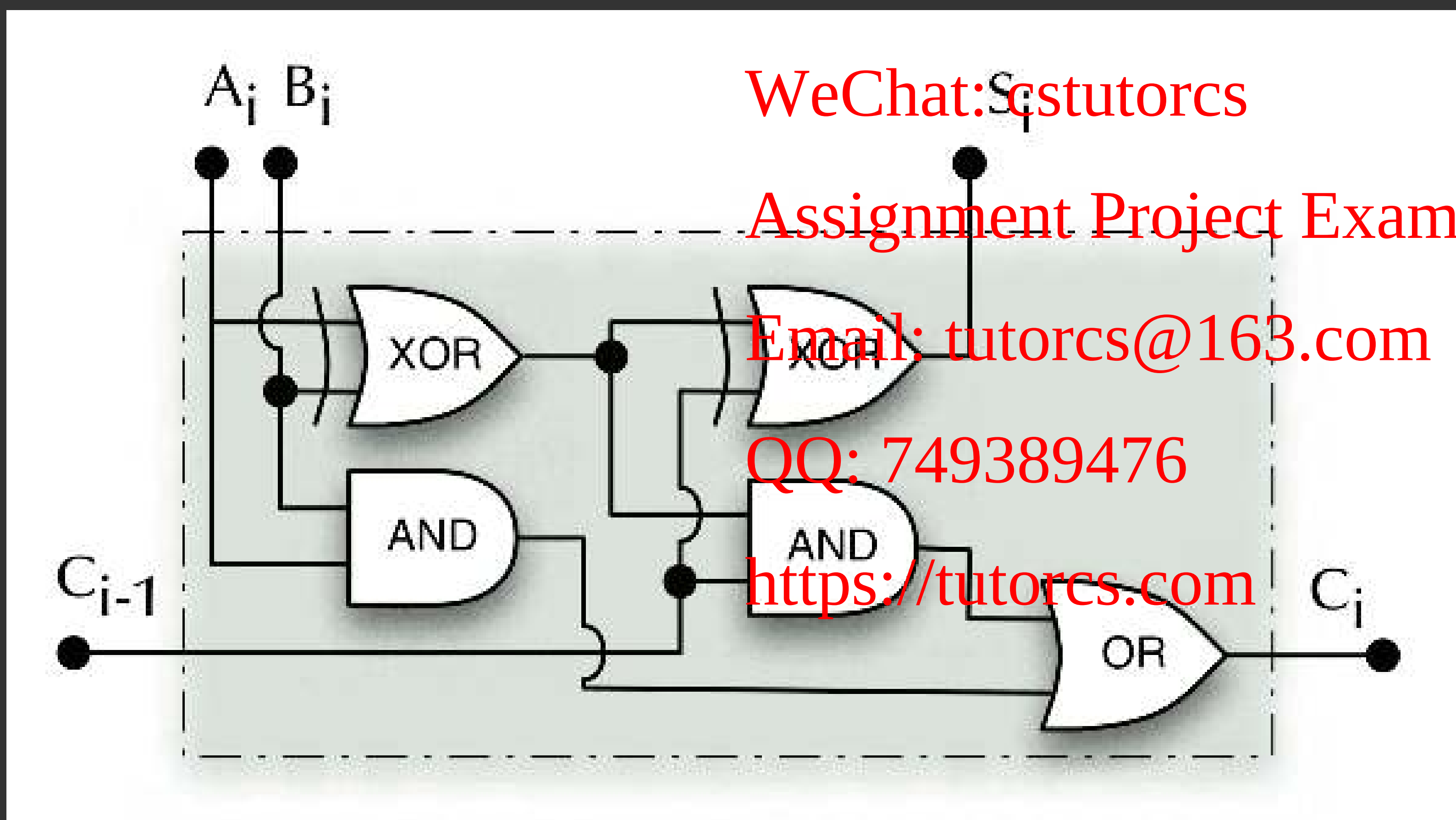
$$s = (a \land \neg b \land \neg ci) \lor (\neg a \land b \land \neg ci) \lor (\neg a \land \neg b \land ci) \lor (a \land b \land ci)$$

$$= (((a \land \neg b) \lor (\neg a \land b)) \land \neg ci) \lor (((\neg a \land \neg b) \lor (a \land b)) \land ci)$$

$$= ((a \oplus b) \land \neg ci) \lor ((a = b) \land ci)$$

Full-adder

cout = (a ∧ b) ∨ ((a ⊕ b) ∧ cin) *(trust me!)*



Demo: Full-adder

Ripple-carry adder

We can join them together like so

talk

1. how many bits can be added t

2. how long does it take?

3. where does the final carry bit go?

What about subtraction?

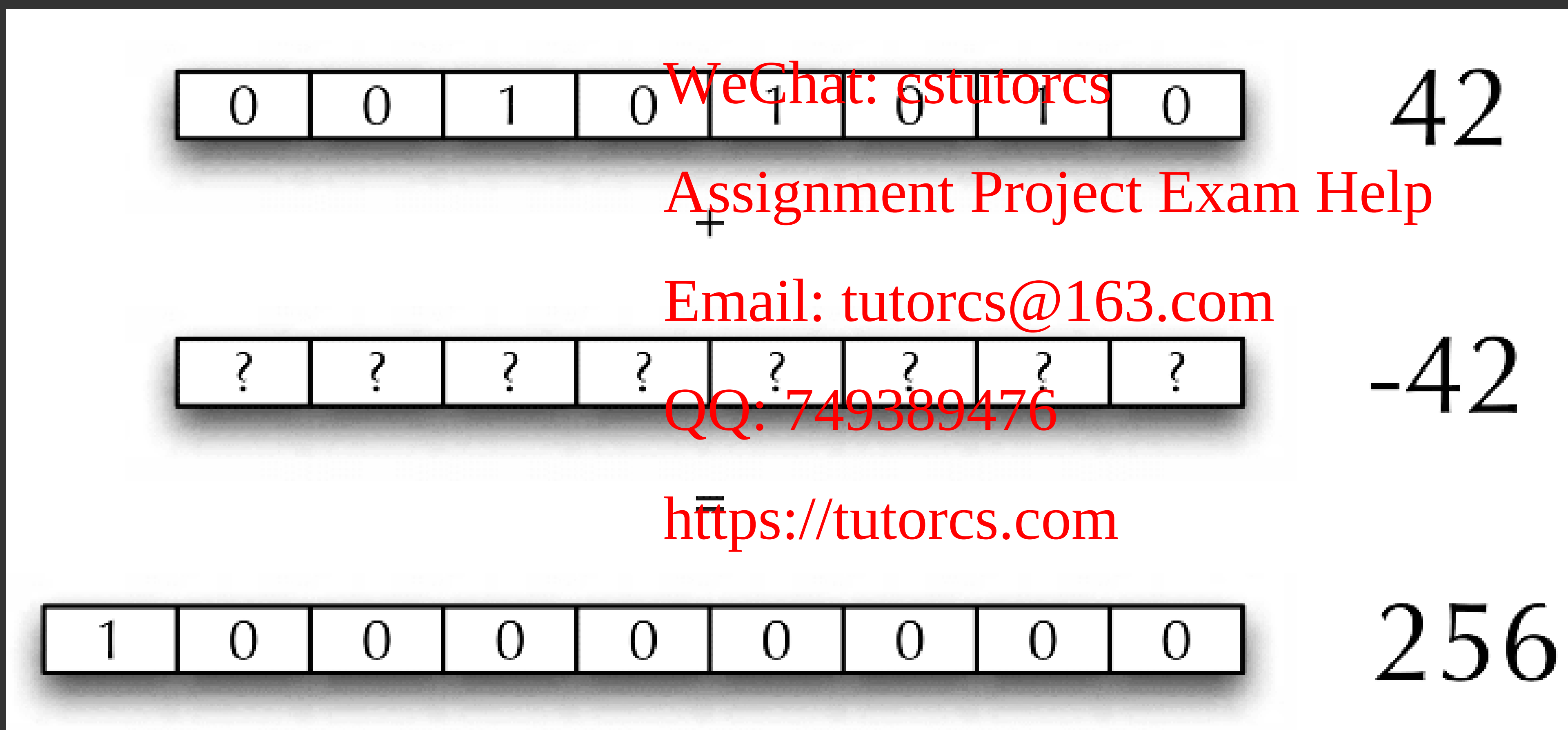
We've got two options:

1. do some more minterms stuff
2. trick the full *adder* into subtracting things instead

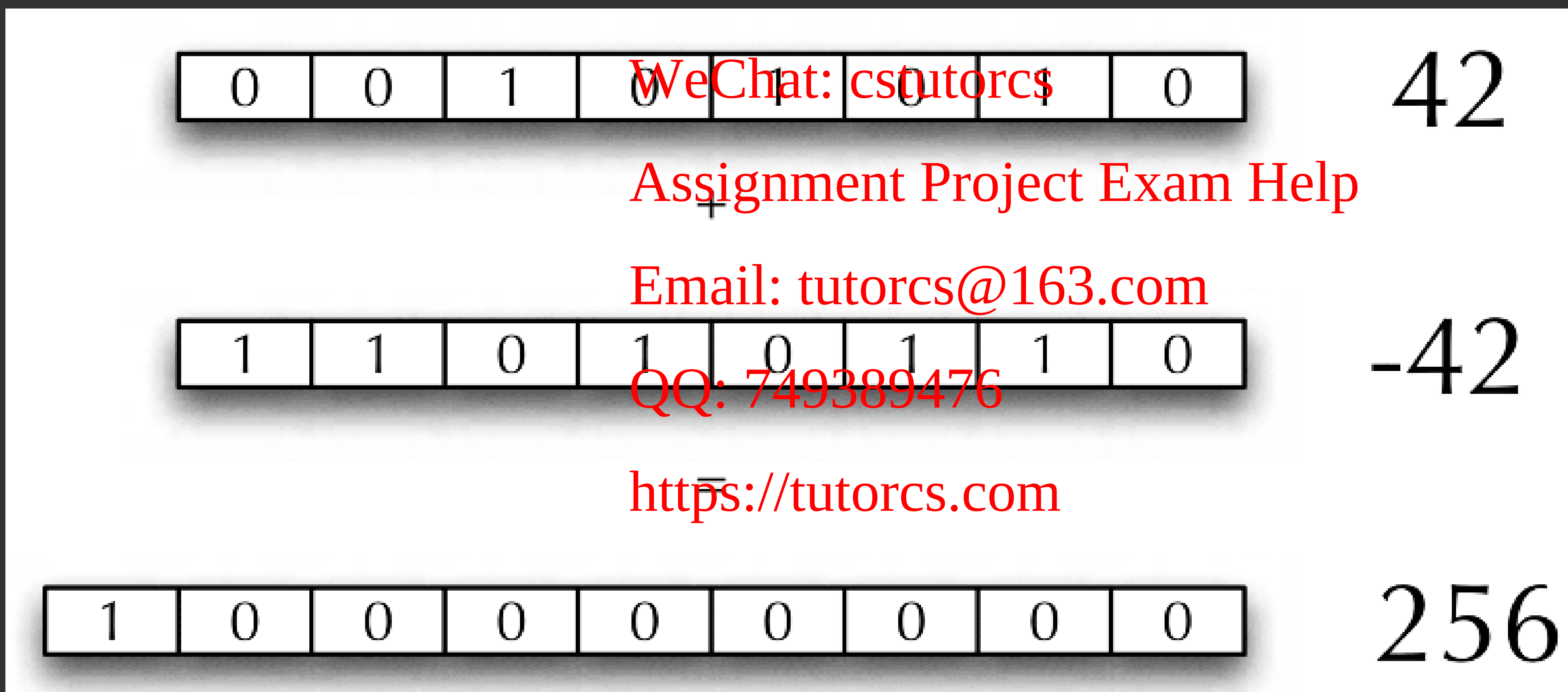
show of hands?

Twos complement representation

The basic idea: can we define (bi tive numbers such that our adder still works?



Twos complement representation



The twos complement "circle"

Quiz: negative or positive?

`0b10`

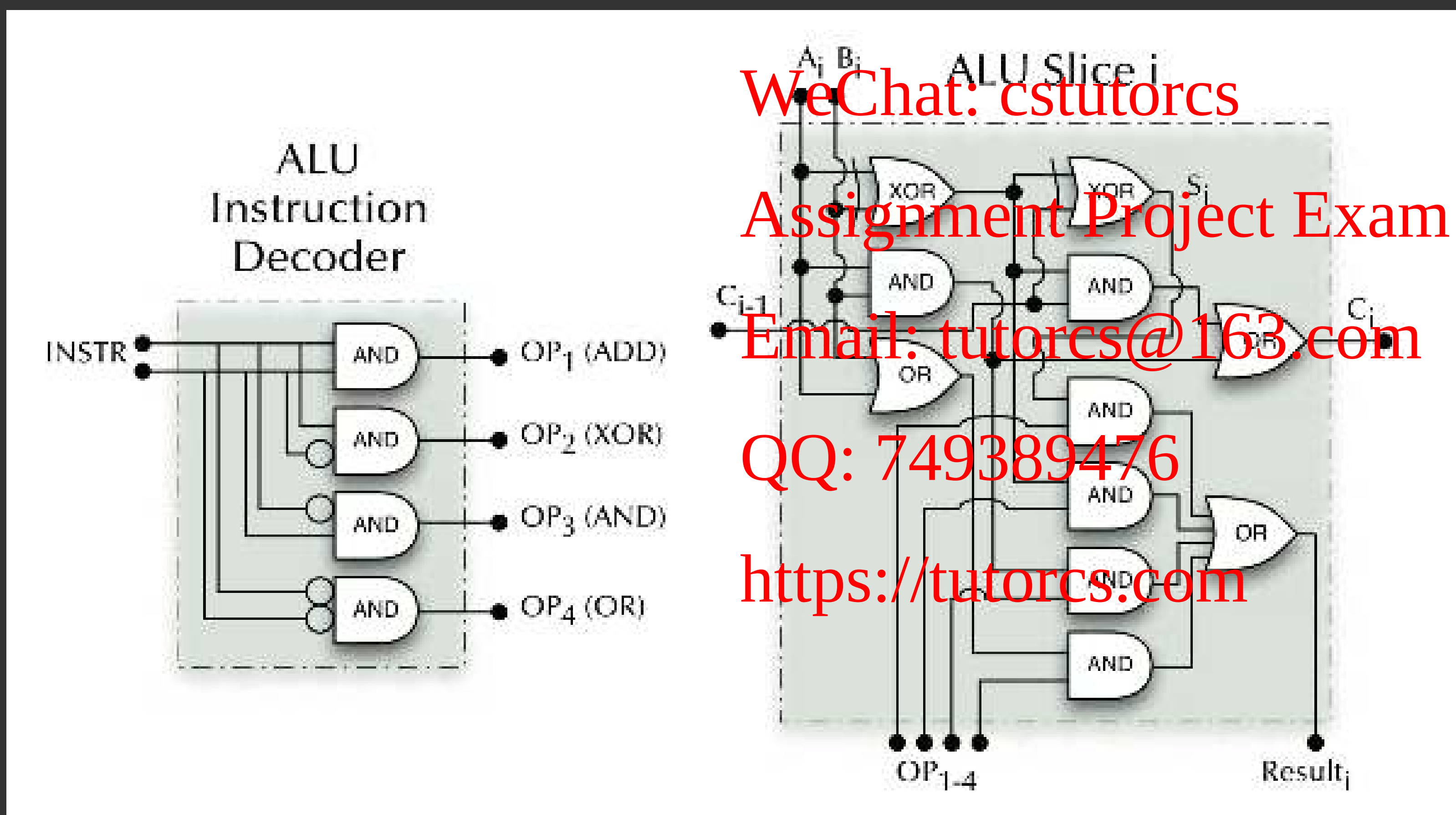
`0b011011`

`0b011011101110101010010`

`0b10110000000000011011111111111101010010000000001010101` it’s all in your mind

Simple ALU

A simple ALU (Arithmetic & Logic h can ADD, XOR, AND, OR two arguments.



Sequential logic

talk

What does it mean for a compute[illegible] **memory**? Can the combinational logic functions we’ve looked at so far *remembe*[illegible] no! we need feedback

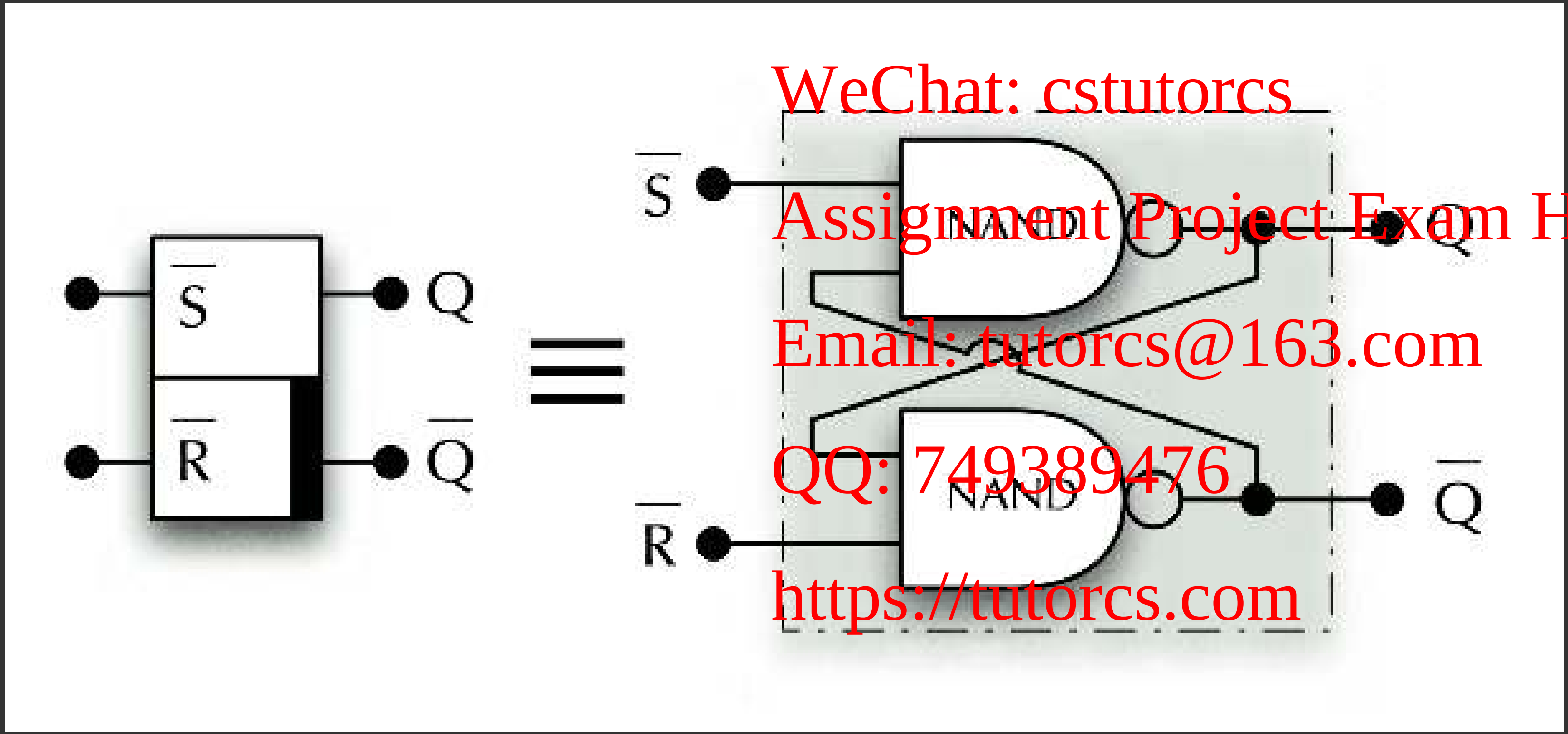
Sequential == state-oriented

This makes intuitive sense—the feedback loop allows the current output to be fed back in (as input).

Sequential logic circuits can no longer be treated as “pure” input-output black boxes—they carry “state” (i.e. the **sequence** of inputs matters).

SR latch

S (set), R (reset)



stackexchange asks: “Why is the output of stateful elements often named Q?”

SR input

There are four possible input co[illegible]:

¬s	¬r	*effect*
1	1	keep current state q
0	1	set q (to 1)
1	0	reset q (to 0)
0	0	**forbidden** (q and ¬q both set to 1)

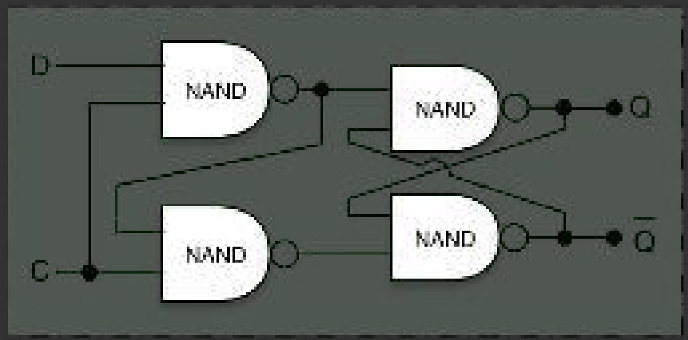
Demo: SR latch

Problems with the SR latch

1. the forbidden thing (this is obv)
2. there are some tricky timing is ause physics: remember, there are *real* electrons flying around)
3. have to keep changing inputs…

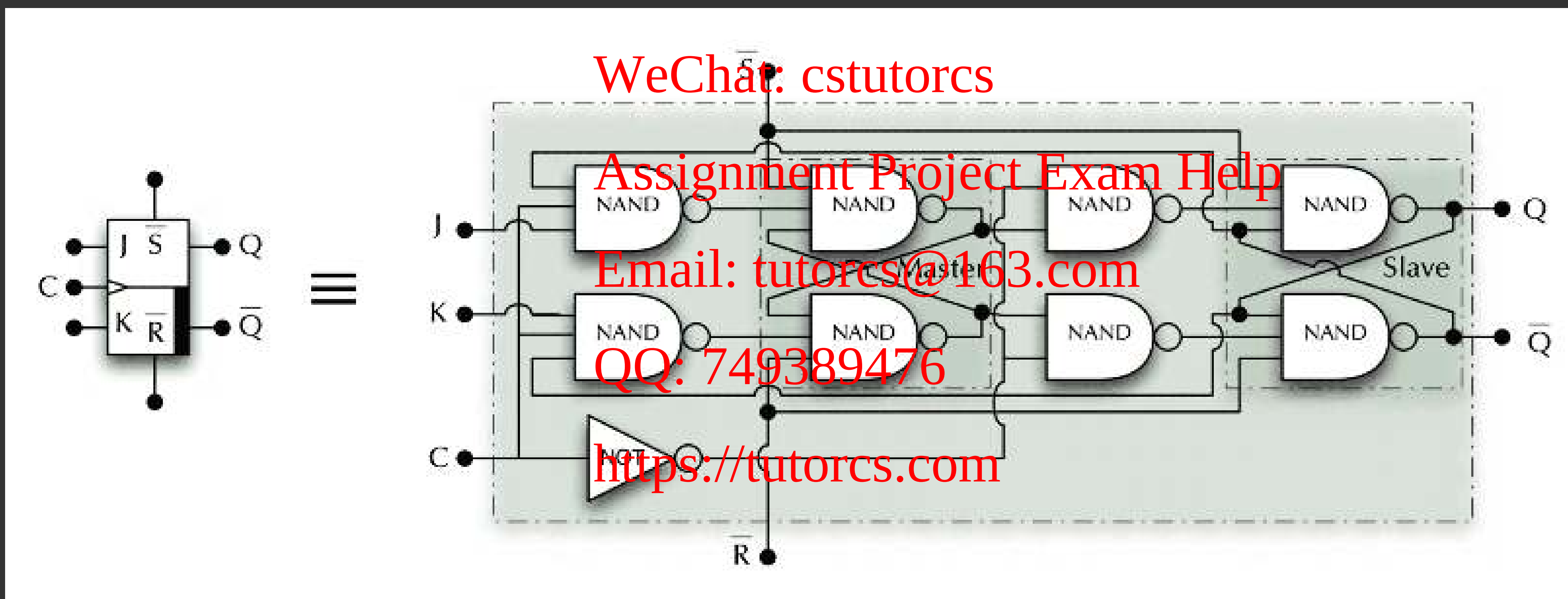
Gated D latch

The gated D latch uses an *enable* [illegible] that the latch is set only when you want it to be.



JK master-slave flip-flop

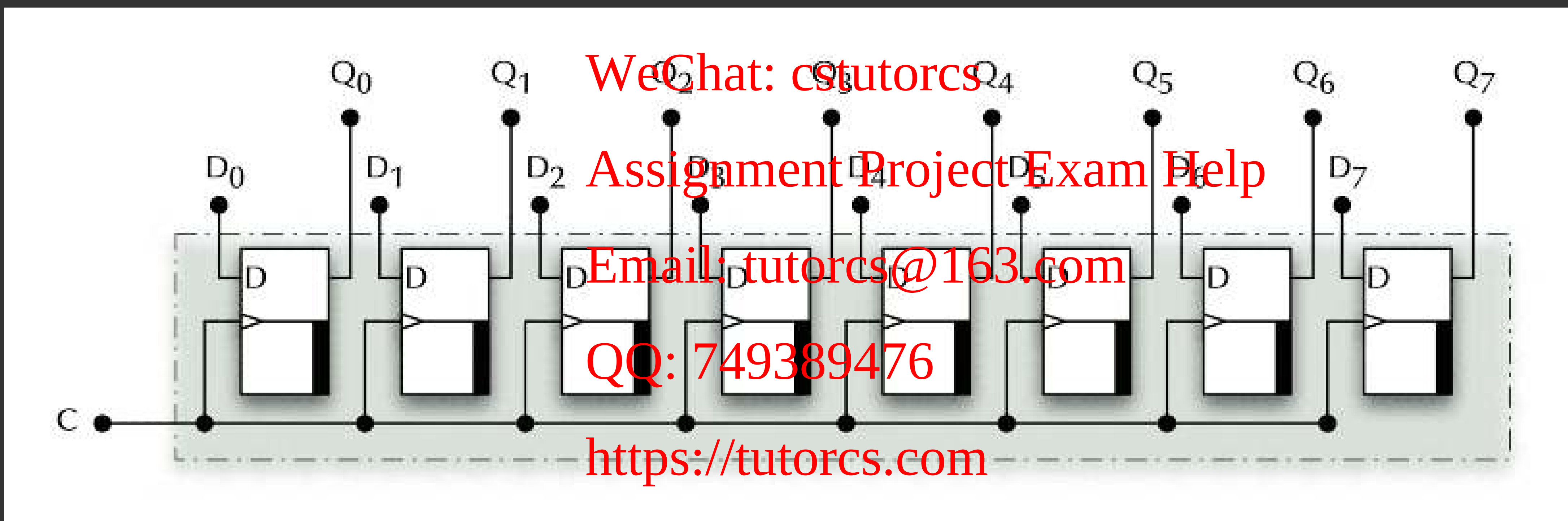
Better again, set/reset on a *clock* problems, no forbidden inputs, toggle operation (although it's a more complex ci



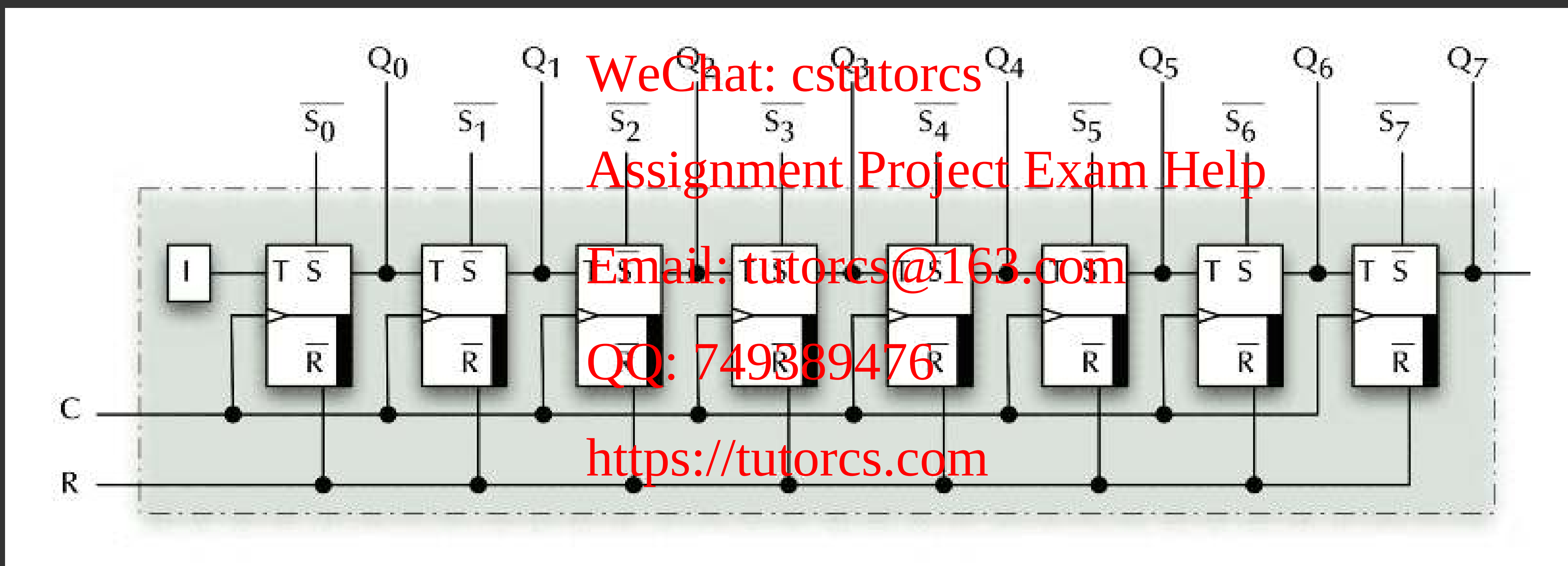
This video is private

Register

Store multiple bits, can serve as ...rpose *fast* on-CPU storage, or hold state for peripherals, etc. Note the shared...



Counter



Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
$\overline{S_0}$
$\overline{S_1}$
$\overline{S_2}$
$\overline{S_3}$
$\overline{S_4}$
$\overline{S_5}$
$\overline{S_6}$
$\overline{S_7}$
I
T $\overline{S}$
$\overline{R}$
C
R

Feeling lost?

These are all physical compone computer.

There's no magic (!), just these **billion** of these little mathematical machines.

Further reading

Plantz: Introduction to Computer Organization chapters 5-7

Patterson & Hennessy *Appendix A "The Basics of Logic Design"*

EEVblog **Intro to Digital Logic (YouTube)**

Ben Eater: **Logic Gates, SR Latch, D Latch, D Flip-Flop, JK Flip Flop**

Questions