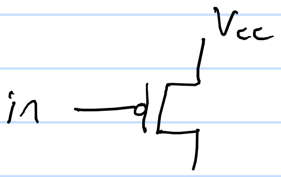
MIDTERM 2 TOPICS

We're going to start from building blocks. So,

Transistors

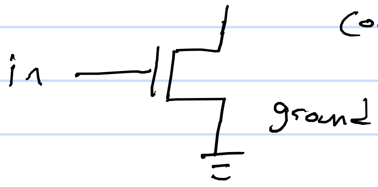
- A transistor is an electrical component that conducts charge based on its input.

p-type:



Conducts when in = ground

n-type:

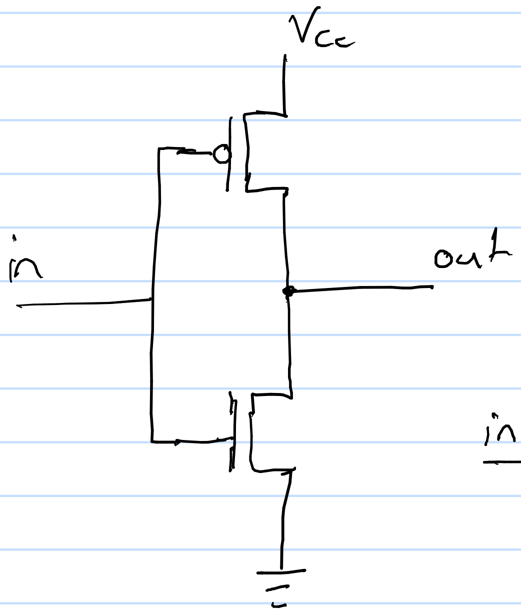


conducts when in = Vcc

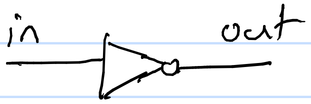
Aside: p-type transistors only conduct a Vcc source, and n-type transistors only conduct a ground source which has to do with transistor construction. You can check articles about these details on Wikipedia (which is pretty accurate) or the internet in general

Building more complicated things!

- Not Gate (Inverter)

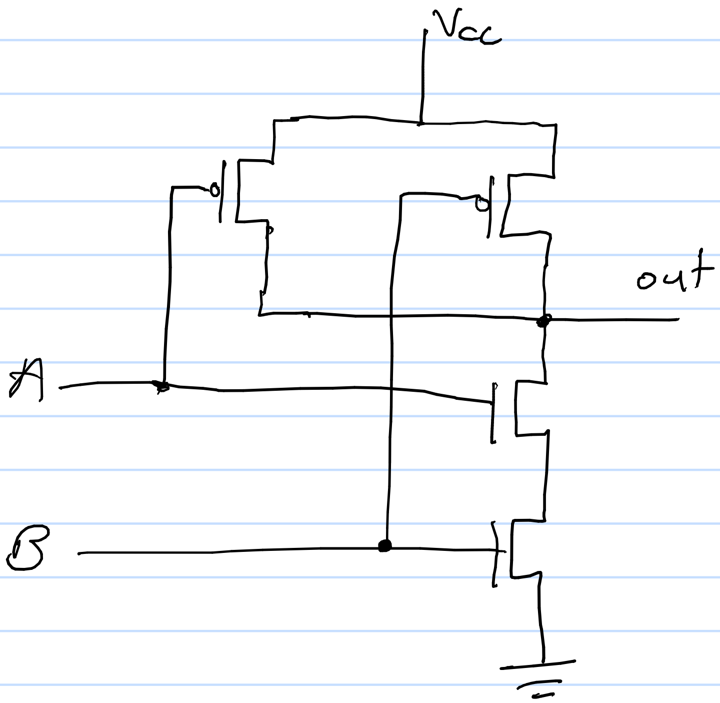


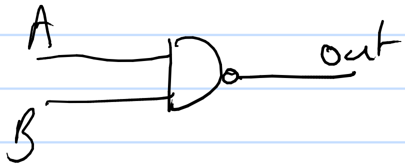
```
if input == 1
    output = 0
else
    output = 1
```



* Note that for the purposes of these gates we will consider Vcc to be 1 and ground to be 0.

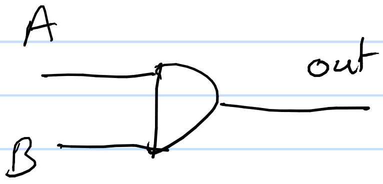
NAND Gate





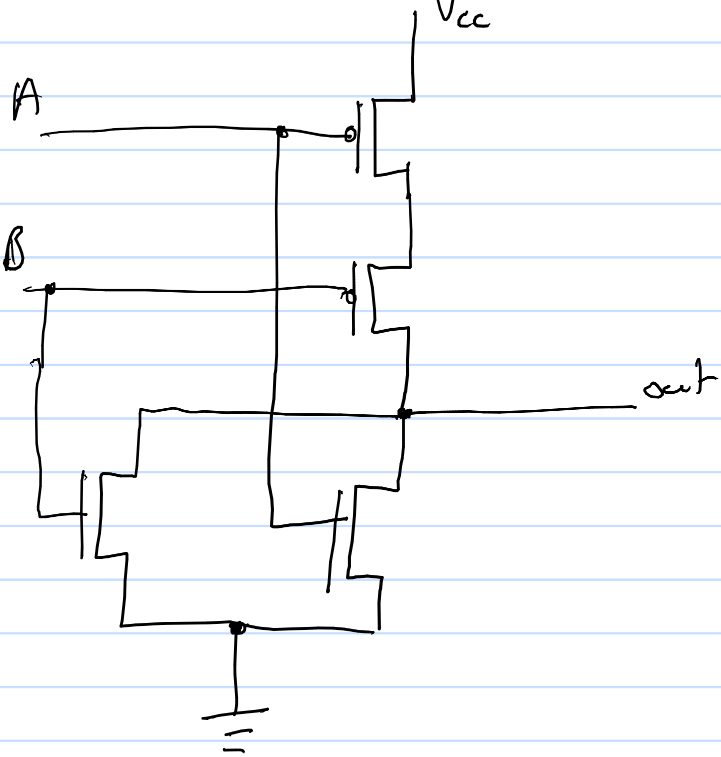
NAND

A	B	out
0	0	1
0	1	1
1	0	1
1	1	0

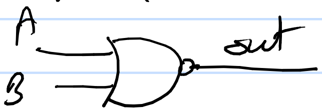


A	B	out
0	0	0
0	1	0
1	0	0
1	1	1

NOR GATE

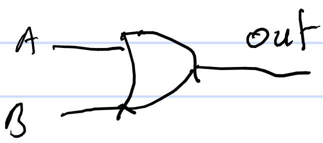


NOR



A	B	out
0	0	1
0	1	0
1	0	0
1	1	0

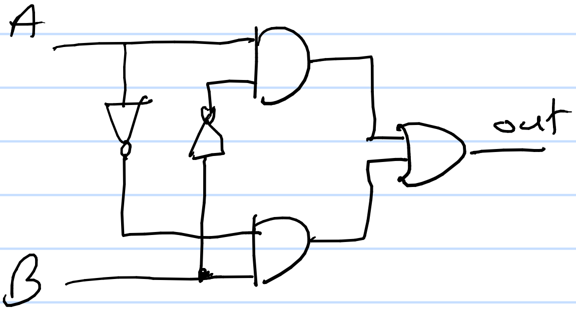
OR

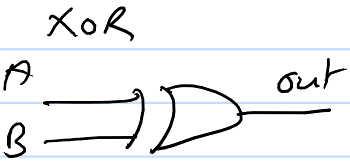


A	B	out
0	0	0
0	1	1
1	0	1
1	1	1

OR and AND gates are built with the combination of a NOR or NAND gate and a NOT gate. Therefore, their propagation delays are usually greater.

XOR



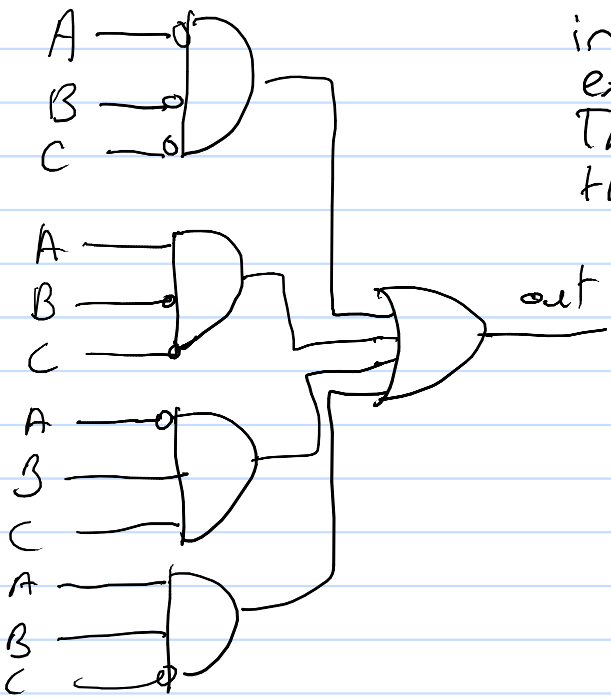


A	B	out
0	0	0
0	1	1
1	0	1
1	1	0

- Going from a truth table to a circuit:

a	b	c	f(a,b,c)
0	0	0	1
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	0

* Simple way: We look at the entries the circuit should return 1, then make AND gates corresponding to those entries (marked in red in the example). Then, we OR those.



* We will not cover minimization in this class. If interested, please look up Karnaugh Maps and other related topics.

* Circuit to truth table is simple: try every possible input on the circuit and fill the truth table in.

Boolean Algebra

- Defn: Algebra with boolean operators and binary operands

AND	$x \cdot y$	or xy
OR	$x + y$	
NOT	$\overline{x}$	

ORDER OF OPERATIONS

( ) first
NOT second (note a large $\overline{\quad}$ implies a parenthesis)
AND third
OR fourth

$a + bc = a + (bc)$

$\bar{a}\bar{b}$ = (not a) and (not b)

$\overline{ab}$ = not (a and b)

$a(b+c)$ = a and (b or c)

Laws

Identity Law : $A + 0 = A$ $\quad A \cdot 1 = A$

Zero/one : $A + 1 = 1$ $\quad A \cdot 0 = 0$

Inverse : $A + \bar{A} = 1$ $\quad A \cdot \bar{A} = 0$

Commutative : $A + B = B + A$ $\quad A \cdot B = B \cdot A$

Associative : $A + (B + C) = (A + B) + C$ $\quad A(BC) = (AB)C$

Distributive :

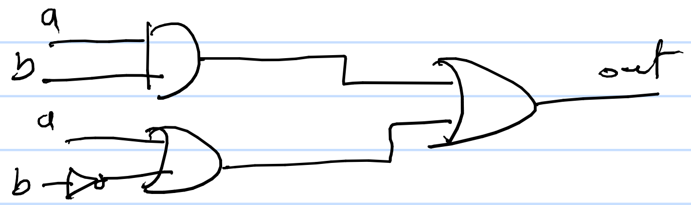
$$A(B + C) = AB + AC$$

$$A + (BC) = (A + B)(A + C)$$

Boolean Algebra → Circuits

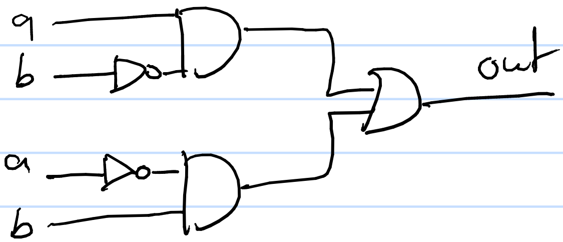
Each operator is a gate. Simply write the equation out in a circuit.

$$(ab) + (a + \bar{b})$$



Circuits to boolean algebra

- Write out the circuit gates as boolean operators!



$(a\bar{b}) + (\bar{a}b)$

* Go to truth tables from bool. alg. just as in circuits: plug in all possible inputs, find the correct output.

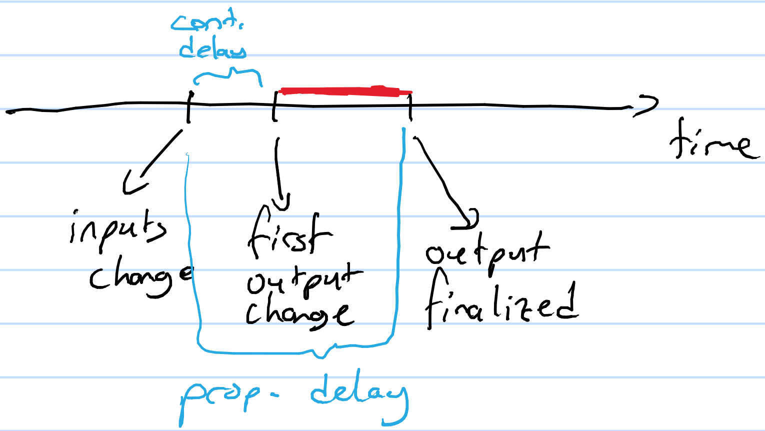
Propagation Delay

Defn: The delay from time an input signal changes to time the output of the circuit changes to its final " value.

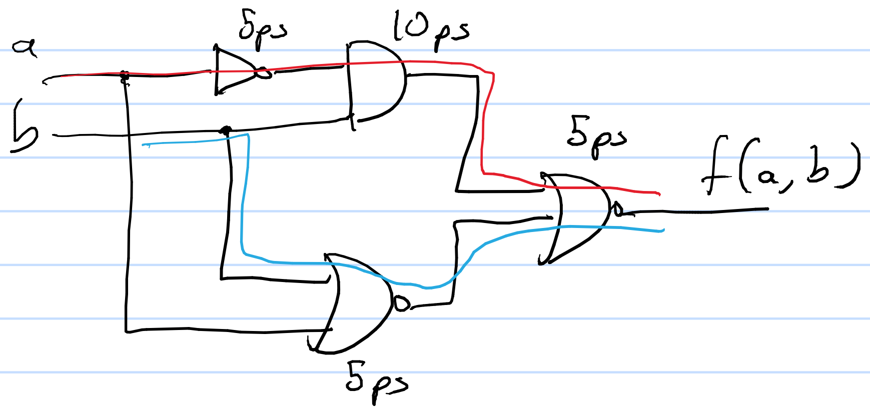
Contamination Delay

Defn: The delay from time an input signal changes to time an output signal changes (note that this refers to ANY change, not the final value).

A circuit may temporarily produce incorrect output in the red zone below:



- Gates and wires introduce delays. We will ignore wire delays in this lecture



prop. delay = 20ps
cont. delay = 10ps

* Since NAND and NOR gates are faster than using AND and OR gates, some simple substitutions may make a circuit faster. For this, we rely on this observation:

$$\overline{(a+b)} = \bar{a}\bar{b} \qquad \overline{(ab)} = \bar{a}+\bar{b}$$

The above circuit has:

$$\bar{a}b.$$

If we replace that with

$$\overline{(a+\bar{b})}$$

We lower our prop. delay to 15ps!

Combinational Circuits

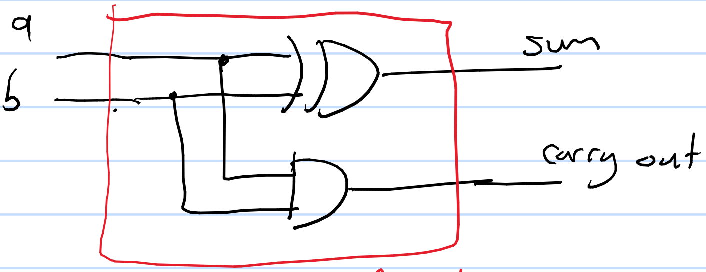
Defn: Circuits whose outputs depend ONLY on their current inputs.

* No concept of state in a combinational circuit.

ADDITION

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

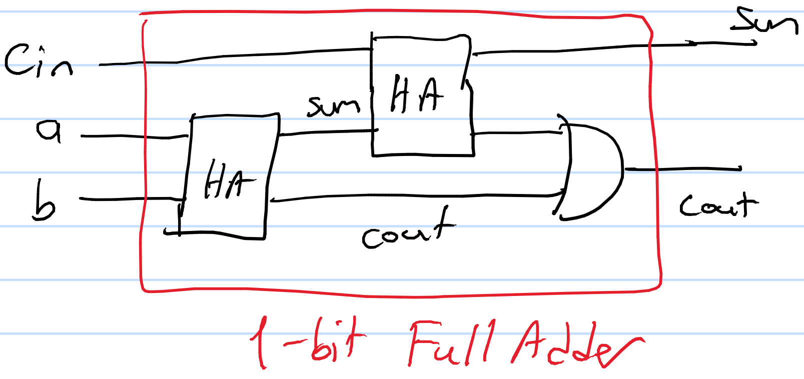
XOR AND



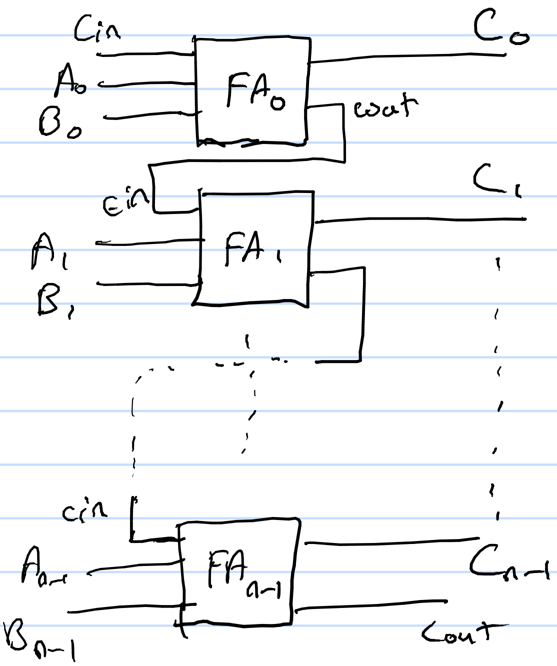
1 bit half adder

Full Adder

- Has additional cin input. Uses 2 Half adders.



n-bit Adder



* We can make an n-bit subtractor using a n-bit adder.

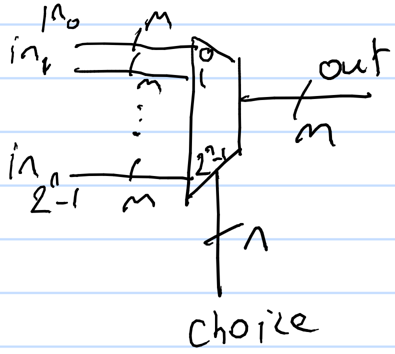
Let's observe that

$$A - B = A + (-B)$$

* Which requires negation and addition. Negation in 2's complement is NOT, +1. We can do the NOT with a gate, then use the Cin input of the final adder to do the +1 part. The adder will then do subtraction

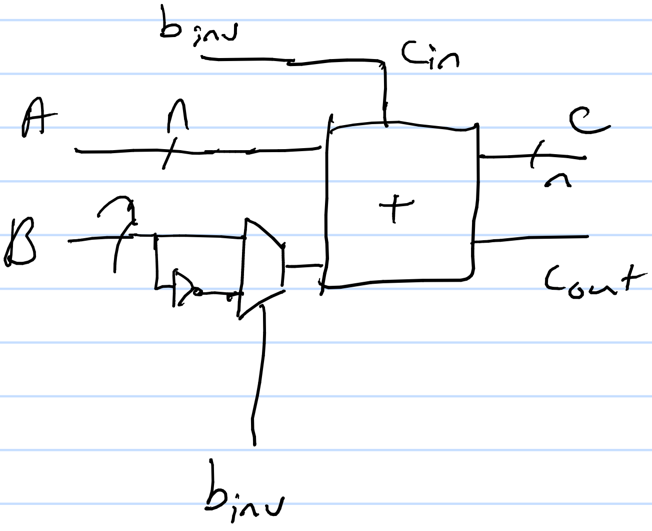
* To do both requires a selection. We select between multiple inputs with a multiplexer.

n-bit choice multiplexer with m-bit inputs:

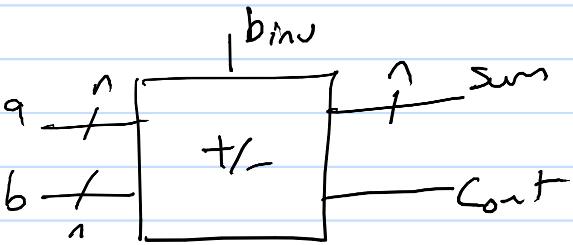


Based on the value of the choice inputs the MUX outputs ONLY one of its inputs.

Adder Subtractor

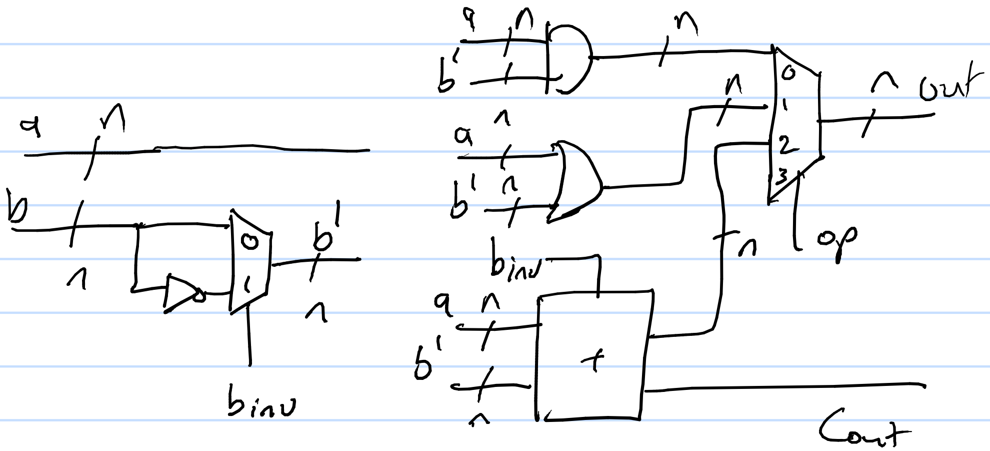


⇓



ALU with AND, OR, +, -

* Note that I'm using a regular adder here.



op₁	op₂	binv	output
0	0	0	a b
0	0	1	a $\overline{b}$
0	1	0	a+b
0	1	1	a+$\overline{b}$
1	0	0	a ADD b
1	0	1	a SUB b
1	1	X	illegal

Additional arithmetic operations

Unsigned overflow detection

- If $C_{out} == 1$, then there is overflow

2's comp. overflow detection

If most significant bit's $C_{in} \neq C_{out}$, there is overflow. Check with XOR.

Equality checking

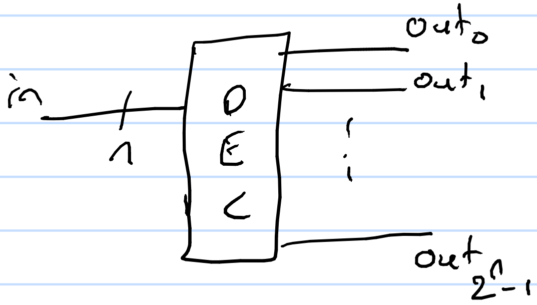
A) Do bitwise XOR, then NOR all bits. If 1, they're equal.

B) Subtract numbers, NOR all bits of result. If 1, they're equal.

Set on less than

- Subtract numbers, then the result's most sig. bit is 1 if $a < b$.

n-bit Decoder



* Based on in, only ONE of the output wires will be 1, the rest will be 0.

↳ There will always be a wire that is 1!!

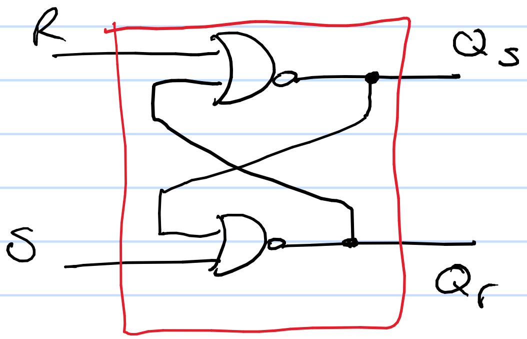
Aside: reverse of this is called an encoder

Sequential Circuits

Defn: Circuits whose outputs depend on their current inputs as well as their previous inputs

* These circuits maintain a state!

RS-Latch



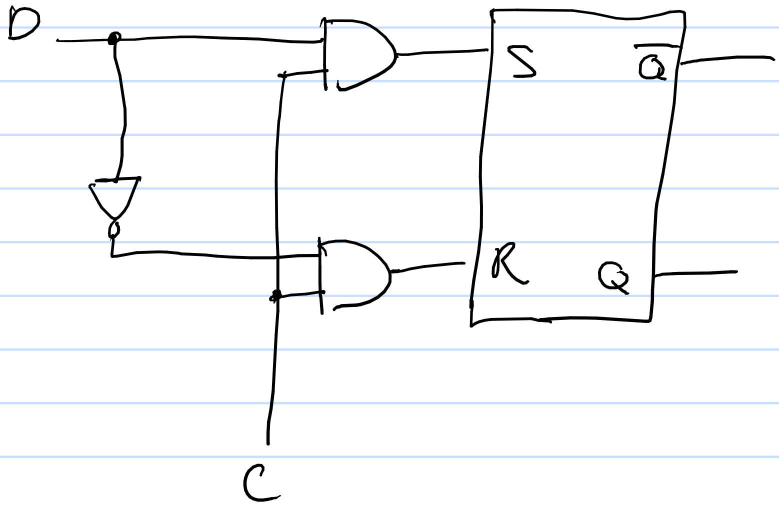
R	S	Q_s	Q_r
0	0	(memory)	
0	1	1	0
1	0	0	1
1	1	0	0

this entry can't be remembered!

* Set ONE of the inputs to write a value, then switch both inputs to 0 to remember.

* Mixes WHAT changes and WHEN it changes.

D-Latch



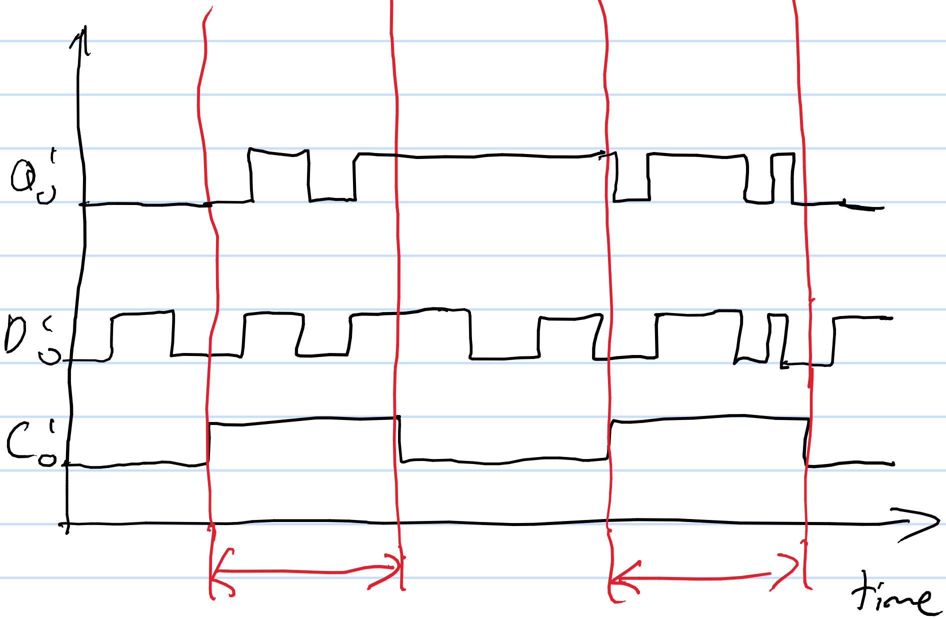
D	C	Q	$\overline{Q}$
0	0	(memory)	
0	1	0	1
1	0	(memory)	
1	1	1	0

* D controls WHAT the state should be

* C controls WHEN it should change.

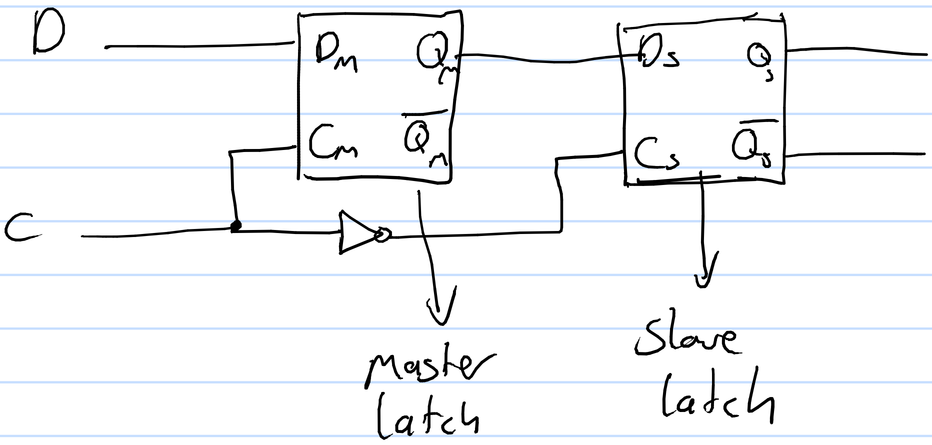
Clear distinction!

D-latch timeline

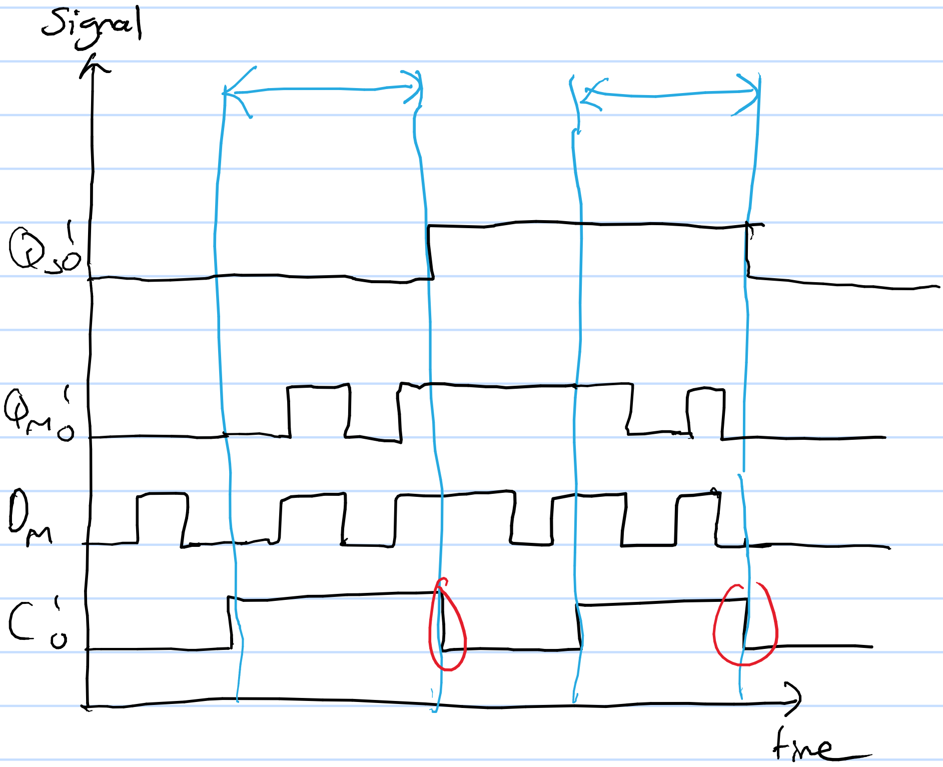


Q only changes when C is 1!

D-flip flop



D
C
D_M
Q_M
C_M
$\overline{Q_M}$
D_S
Q_S
C_S
$\overline{Q_S}$
master latch
slave latch

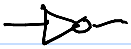


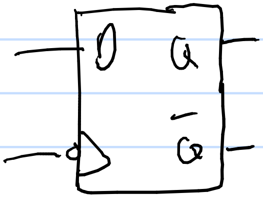
Signal
Q_S 1 0
Q_M 1 0
D_M
C 1 0
time

- Q_s only changes when the clock is falling!

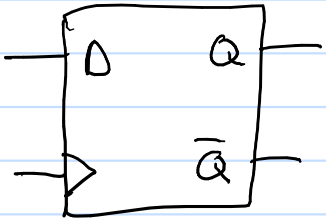
- Q_m changes when $C=1$

- Very brief window where change can happen.

- If the  is relocated to C_m's input, then Q_s will change when clock is rising and Q_m will change when $C=0$.

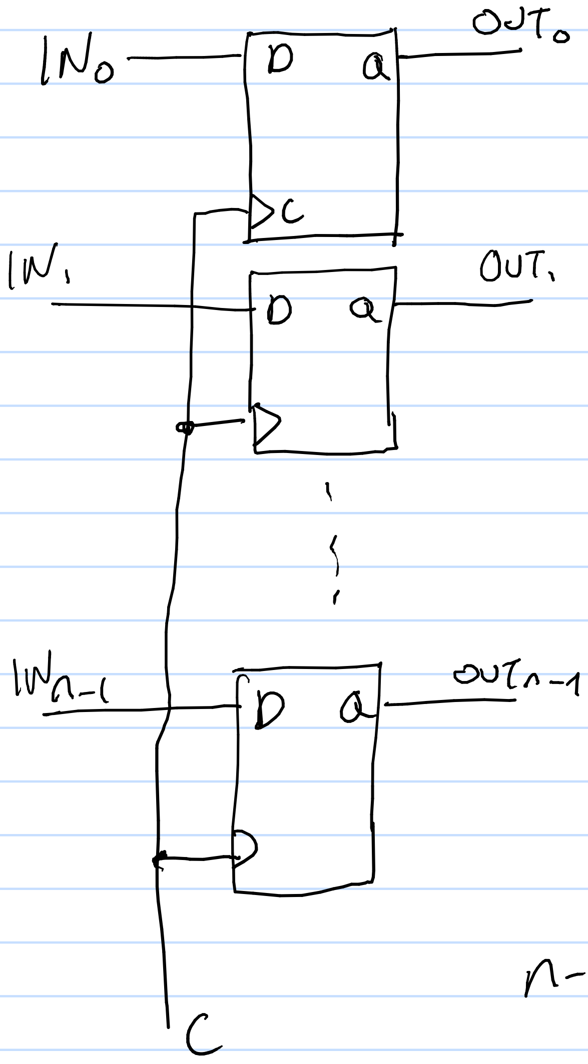


falling edge
D flip flop

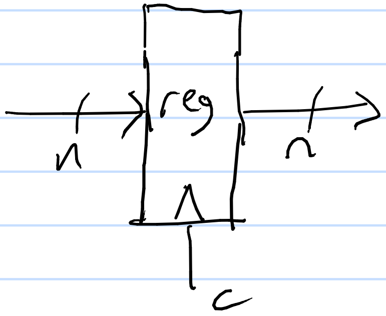


rising edge
D-flip flop

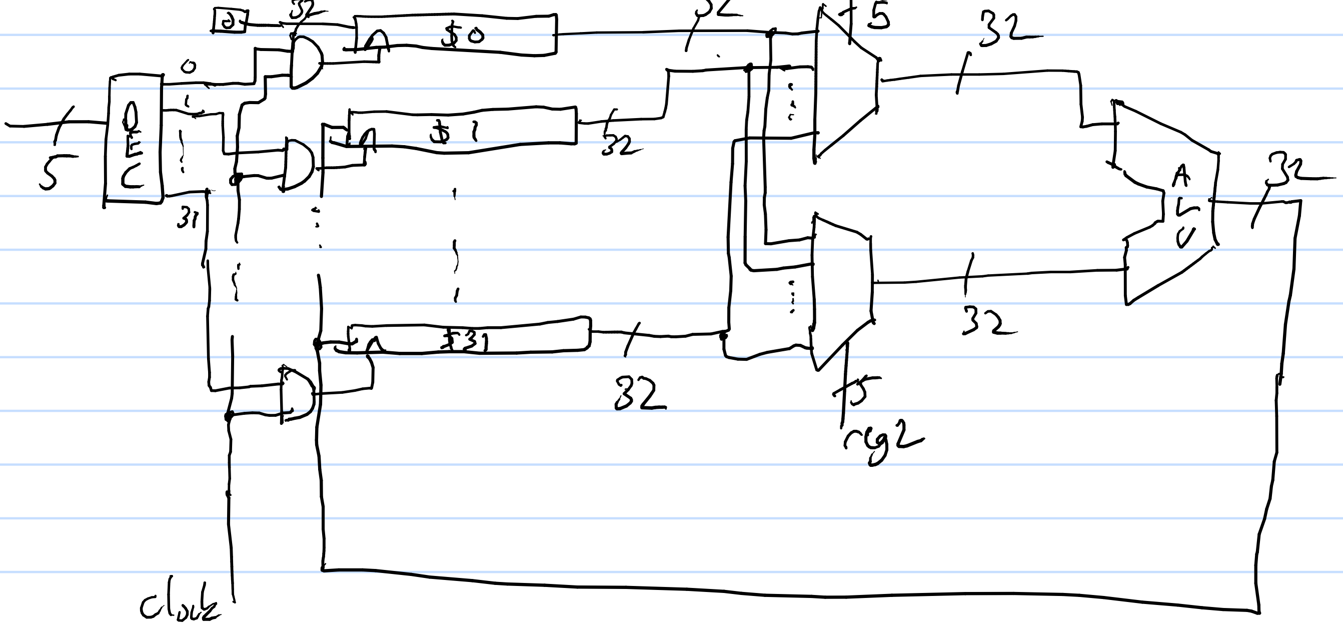
Registers



n-bit register

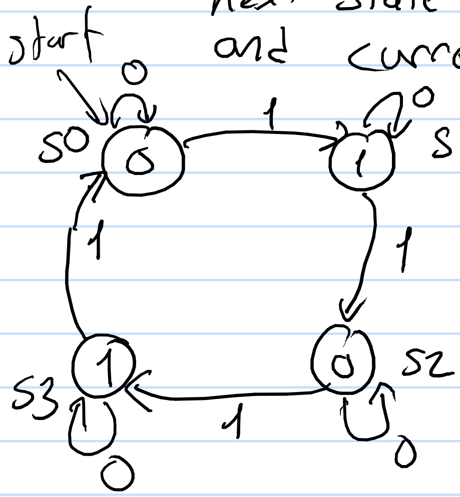


Sample MIPS Datapath



State Machines

* Consists of:
  1) A register to hold state
  2) A comb. circuit to determine output based on state
  3) A comb. circuit to determine next state based on input and current state.



States are represented with circles. Outputs are in states. Transitions are marked with input.

1) We need a 2 bit register to hold state.

2) our output function is:

q_1	q_0	out
0	0	0
0	1	1
1	0	0
1	1	1

3) Our next state function is

q_1	q_0	in	q_{1n}	q_{0n}
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

Memory

Random Access Memory

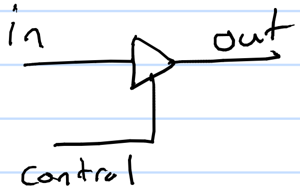
* Random means we can access any memory word we want in the same amount of time as any other word.

2 major types we will look at:

Static RAM (SRAM)
- ↳ Made from latches
- ↳ Remembers data as long as power is on

Dynamic RAM (DRAM)
- ↳ Stores data in capacitors
- ↳ Forgets data when read or after some time
- ↳ Much, much cheaper than SRAM.

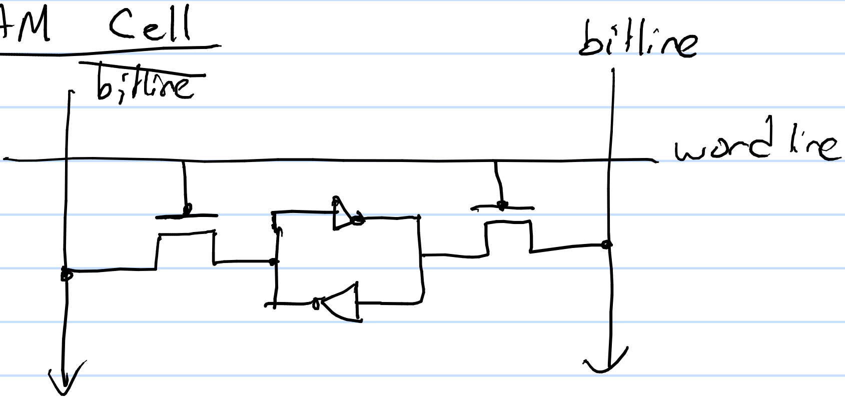
Tri-State Gate



control	in	out
0	0	hi-Z
0	1	hi-Z
1	0	0
1	1	1

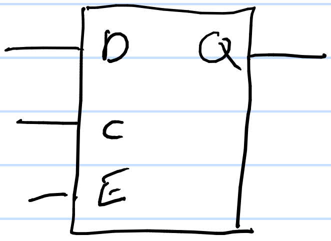
hi-Z is high impendance
↳ Means there's no charge at all on the wire.
↳ useful for connecting a bunch of outputs without using gates or risking short-circuits

SRAM Cell



To Read: wordline = 1, observe bitline
To Write: wordline = 1, charge and hold both bitline & $\overline{\text{bitline}}$

- The wordline serves as an enable switch.
- We can represent an SRAM cell like so:



C=1 to write
E=1 to read

n×M SRAM

n → # of words

M → word size → units in bits!!

4x2 SRAM means 4 words with 2 bits each

- Look into the handout for an example 4x2 SRAM! (also on Canvas)

WE = write enable
OE = output enable
CS = chip select (if CS = 0 then this chunk of RAM does nothing)

* If we want to build large SRAM's the circuits become thin rectangles.

↳ We want squares. Let's put multiple words on one line.

$8K \times 32 = 2^3 \cdot 2^{10} \cdot 2^5$
$= 2^{18}$ bits total

Square layout would have:

2^9 latches per side.

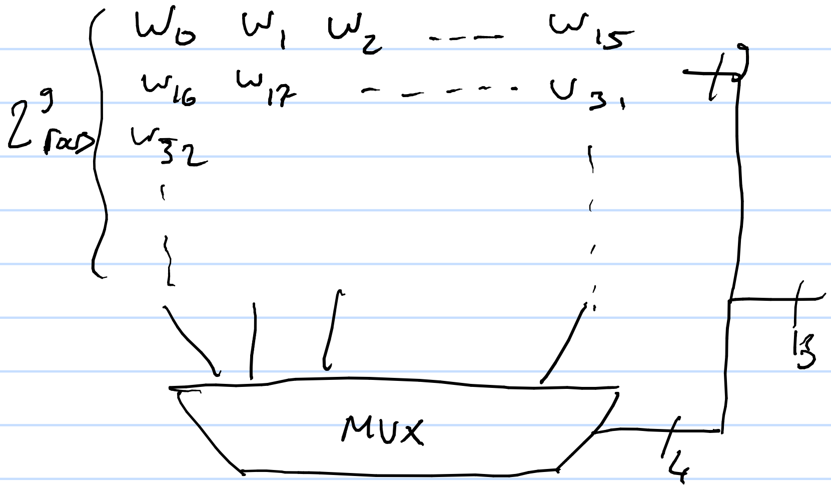
$2^9 / \underbrace{2^5}_{\text{word size}} = 2^4$ words in a line.

* We still want a single word out of this!

* We will use part of the address to select row, then select word with the rest of the bits.

* There are 8K words in the RAM.
* Therefore there are 2^{13} things to choose from. The address will be 13 bits!

* Since we have 2^9 rows, we will use 9 of those address bits to select the row. The remaining 4 will select the word.



* The <u>most significant bits</u> select the row!

Summary:

In an $n \times m$ SRAM:

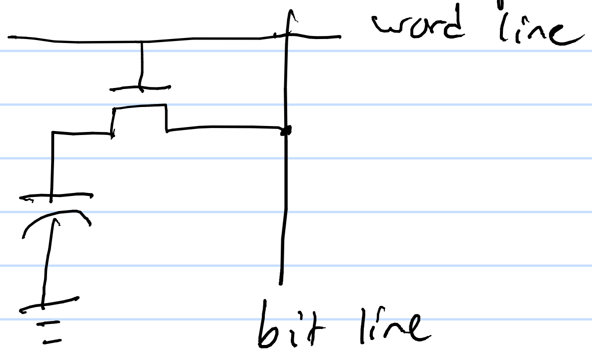
- $\log_2 n$ address bits
- Cell array is $\sqrt{n \times m}$ rows by $\sqrt{n \times m}$ columns
- $\log_2 \sqrt{n \times m}$ bits to select row
- $\log_2 n - \log_2 \sqrt{n \times m}$ input mux
- m data bits

So in a $32K \times 8$ SRAM

- $2^5 \cdot 2^{10} = 2^{15}$, 15 address bits
- 2^9 rows
- 2^9 colums
- 9 bits to select row
- $15 - 9 = 6$ bits to select word
- 8 data bits

DRAM Cell

- Stores value in capacitor



- Stores 1 bit! Uses fewer transistors than SRAM, thus cheaper.

To read:

1) wordline = 1
2) wait for capacitor to discharge
3) observe bitline.

* NOTE: Reading a DRAM cell destroys its value You need to write the value you observed back!

To write:

1) wordline = 1
2) charge and hold bitline until capacitor is charged or discharged
3) wordline = 0 to store

* The capacitor will leak in about 10^{-3} seconds even if not read. DRAM's periodically refresh themselves.

— END EXAM 2 —