CSC258 Notes

Max Xu

'25 Fall

Contents

§1 Day 1: Welcome, Transistors, and Electricity (Sept 2, 2025)

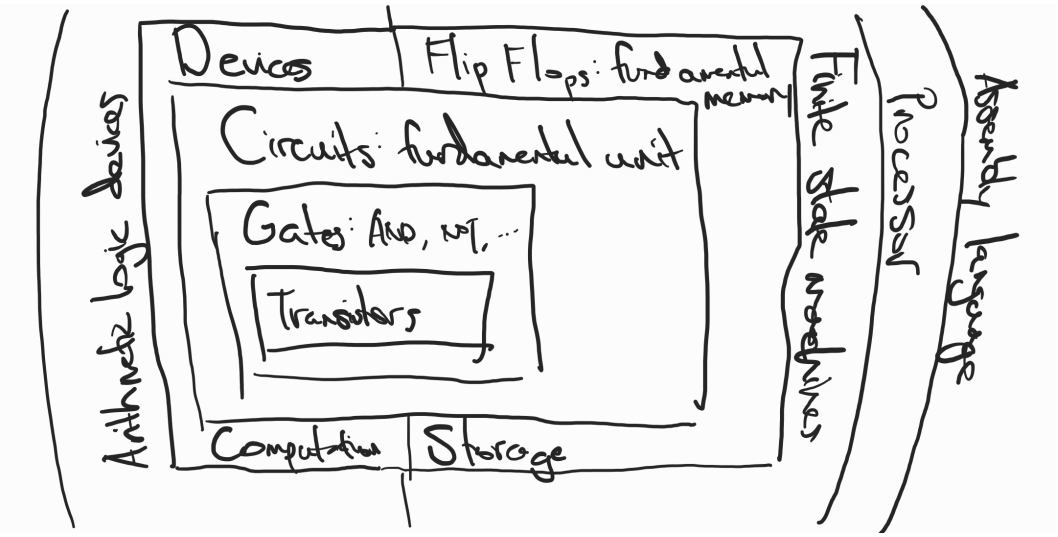
We are trying to:

- Understand computer software and hardware: Why is max int $2^{32} - 1$?
- How does Python evaluate `true` and `false`: Why are some values truthy? There's a hardware reason for that![1]

Why are we taking this course?

- Understanding the machine, integrating software with hardware (e.g. sensors): such hardware skills can be rare these days
- Working with devices (Internet of Things)
- Satisfying prerequisites: stepping stones, a strong foundation
- You don't get a choice :D

§1.1 The Course at a Glance



Remark 1.1. The tradeoff: we want a lot of transistors, because they speed up the processor, but they take up space, and we are reaching a physical limit as to how many we can stuff in a chip. Speeding up processors are not going to be as valuable, unless there is a significant breakthrough. Another way is to solve the problem using more *efficient* circuits, CS Theory enthusiasts rejoice! We could also restrict circuits from being general purpose to more specialized designs that excel at a particular task.[a]. All in all, there's a lot of active research, and utility for these thingamajigs.

[a] an example can be GPUs and matrix multiplication

Logic gates are the hardware equivalent of propositional operators previously encountered. Common ones include: AND, OR, NOT, XOR. We will learn about the mysterious buffer gate, which has the following behavior: $0 \to 0$, $1 \to 1$, in later weeks. To implement logic gates, we will need *transistors*, and we will later properly motivate why we need them. Logic gates can then be used to implement higher level behavior, like addition!

Unlike other CS courses, this course is about creating devices and machines, rather than programs and algorithms.

[1] None, False, and generally numeric zero values and empty sequences/collections evaluate to false

§1.2 Course Goals

- Be able to create and design circuits, implement Finite State Machines
- Understand microprocessor architecture
- Understand assembly language
- Learn about how lower-level languages and even hardware achieve higher-level behavior

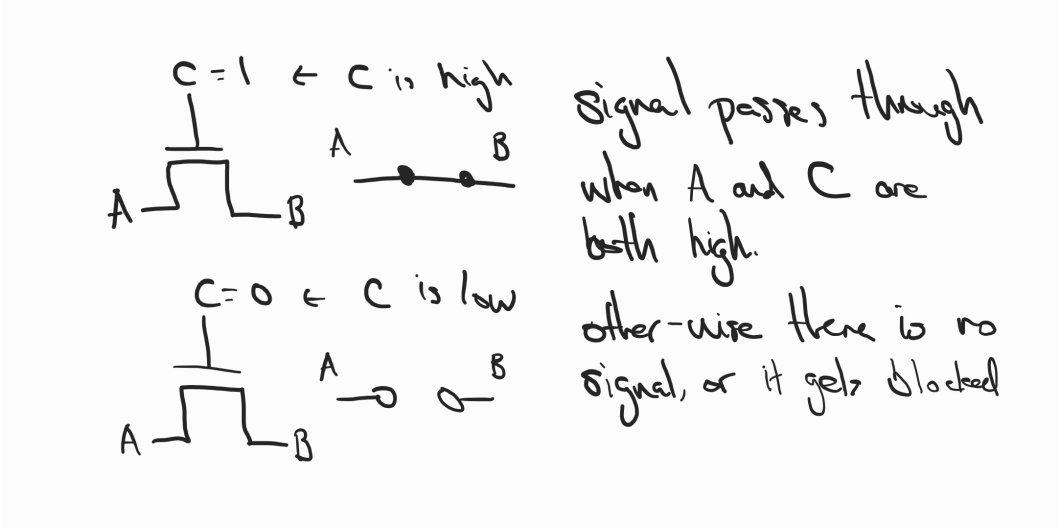
§1.3 Administrative Details

- 2 hours of lecture on Tuesday, with 1 hour of tutorial inbetween
  Half of this tutorial is explaining what the lab is, and the other is a TA going over review material
- 3 hour labs are more involved, note that you will be able to do most of the work at home. There are 7 total, 4% each, starting on week 2.
  - Pre-lab: exercises submitted on Quercus **before** the lab
  - Demo: performed for TAs, there may be follow up questions

  You **must** show up to the lab you're enrolled in, they are assigned according to your last name
- There is an Assembly Language Project worth 15%:
  - Create an interactive game in MIPS assembly
  - 5 milestones worth 3% each
  - Milestone demos take place in the lab rooms. Think about finishing early because labs may get pretty busy towards the end of the year
- Midterm worth 19% tentatively scheduled for Thurs Oct 16, 6pm-8pm
- Final worth 38%, need a 40% to pass the course, with 3 hours time limit

§1.4 Transistors

Transistors form the basic building blocks of all computer hardware, used for amplification, switching, and digital logic design. They made vacuum tubes obsolete, and made computing practical, since vacuum tubes keep breaking.



When the transistor is on, then the value at point B is determined by that of point A, in fact they are equal. Otherwise, the value at point B cannot be determined from the values of point A.

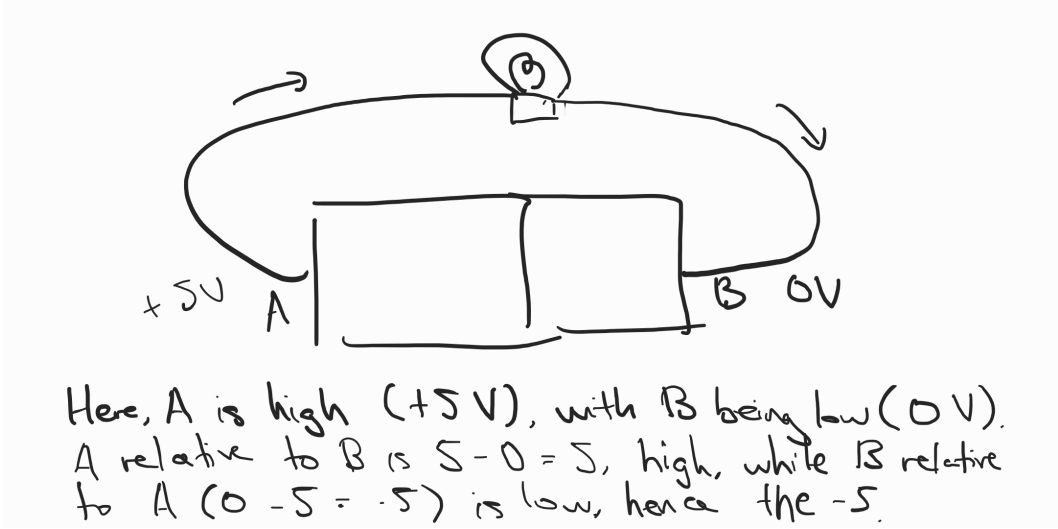
To represent 2 different states, we need connect our wire to a battery (for high), and to ground (for low). Otherwise we may pick up some noise in a wire not connected to ground, and interpret it as high which is undesirable.

> **Remark 1.2.** Logic gates are made from transistors, based on pn-junctions, which are made from semiconductors which conduct electricity. (We don't need to know what pn-junctions nor semiconductors are)

§1.5 Electricity

Definition 1.3 (High: Course Specific)**.** 5 Volts-ish

Definition 1.4 (Low: Course Specific)**.** 0 Volts-ish, or could be -5?



TODO: The reason -5 appears sometimes is because when you're measuring a low and compare it to a high, it will be -5, except when you measure 2 highs it'll show up as a low? Won't you always want to measure relative to ground or something idk.

Definition 1.5 (Electricity)**.** The flow of charged particles (usually electrons) through a material.

Charged particles can do *work*, and originate from atoms. Electrons flow from regions of high electrical potential (many electrons) to regions of low electrical potential (few electrons). This potential[2] is called **voltage**, and the rate of electron flow is called the **current**.

This means that voltage is almost always measured relative to something else. Current is influenced by voltage, as follows.

Example 1.6 (Water Tower Analogy)

Voltage is like the elevation of the water above the ground. **Current** is the rate at which the water flows. The more potential energy, the easier it is to unleash

Definition 1.7 (Resistance)**.** Denoted R, it is the relationship between voltage V (measured in volts) and current I (measured in amps).

$$R := \frac{V}{I}$$[3]

Electrical resistance indicates how well a material allows electricity to flow through it. **Insulators** have high resistance, do not conduct electricity well, with **conductors** having low resistance and conducting electricity well.

Remark 1.8. Even though current is caused by electrons flowing, the convention is to measure current as the **movement of positive charges**, despite them not actually moving. Electrons moving from A to B make A less negative and B more negative. But (rather incorrectly) people conclude that positive charges are moving from B to A, meaning A is getting less positive and B is getting more positive.

Static electricity demonstrates this imbalance, which is caused by materials 'stealing' electrons from each other. When an imbalanced object comes in contact or close enough with a balanced one (because electrons tend to not move through the air), extra electrons transfer over and there exists a *current*, since electrons are flowing!

Sources of electricity include:

- Batteries have a concentration of particles stored, which will run out eventually
- Outlets, which are constantly being supplied to avoid downtime

Current always flows toward the zero voltage point of a circuit, which is called **ground**, and takes the path of *least resistance*. Each circuit has an *source* (where the electrons come from) of electrical particles, some *path* between this source and the *ground* (destination), and some *resistance* along this path that dissipates these electrons.

Definition 1.9 (Semiconductor)**.** Exist somewhere inbetween conductors and insulators, which is a desirable property for transistors.

[2] the reason this is called a potential because it hasn't actually happened yet - prof

[3] also we say that resistance doesn't change a lot because "the material isn't changing" - prof

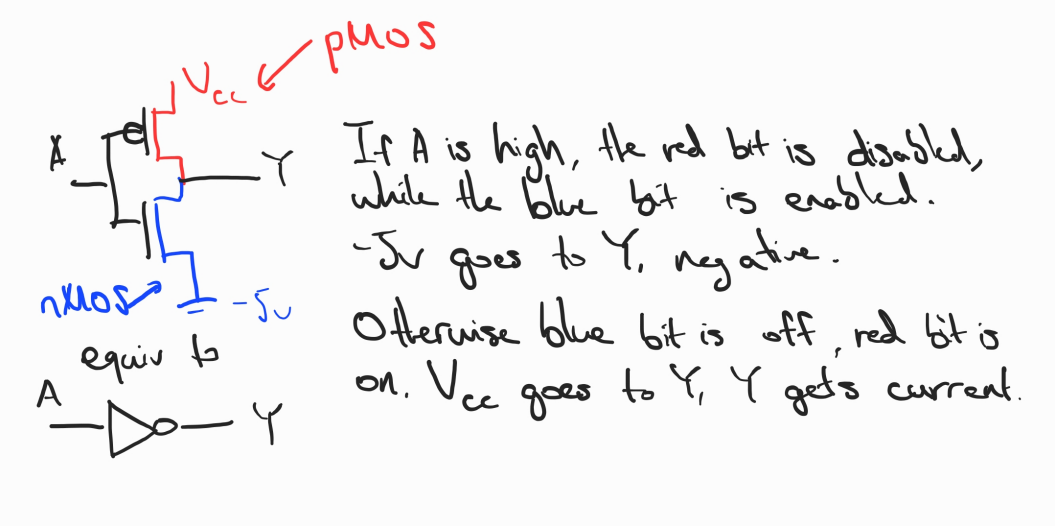
§2 Day 2: More Transistors and Circuit Creation (Sept 9, 2025)

The goal of today is to understand what's inside a gate, and for us to never think about the underlying transistors going on inside ever again. We wish to abstract away such details, and aim for more modular designs.

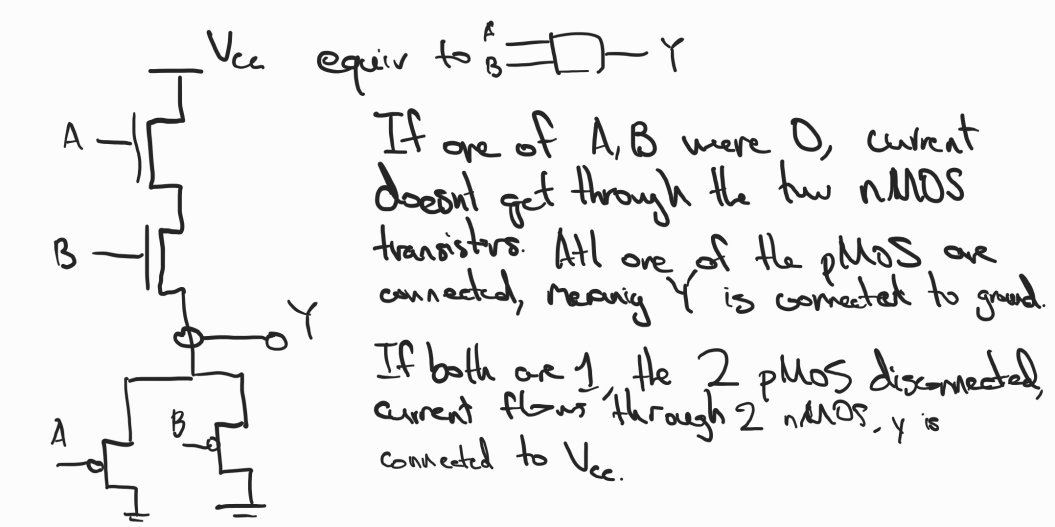
§2.1 Transistors

Problem 2.1. Why is NAND the most common gate?

nMOS transistors has its source and drain connected when the input (gate voltage) is high, and **pMOS** transistors connect when the input is low.



V_{cc} is "common collector voltage" which is high voltage ($5V$) in this course. You need to *explicitly* connect the output to one of V_{cc} and ground for it to be high and low respectively. Otherwise, the result could be high or low. This is what an AND gate looks like at the transistor-level:



We can build 3-input OR, AND gates and the like, but we will avoid making and using them in this course, as it's likely more convenient to use multiple OR and AND gates.

§2.2 Circuits

Now that we understand how transistors and gates work, we wish them to use them to build circuits. In circuits, there is no need to worry about grounding and V_{cc}, the transistors take care of it for you, it's abstracted away.

Problem 2.2. Given an input/output specification, we wish to connect circuit components to produce desired behavior. At the same time, we wish to save costs.

In lab 1 you will represent boolean expressions using logic gates. By convention AND takes precedence over OR. This means that $a\overline{b} + c$ actually means $(a \wedge \neg b) \vee c$.

Circuit Creation and Minterms/Maxterms

After this lecture, we should be able to build a truth table representing the desired behavior of the circuit you want to create, translate truth table rows into gates that implement said circuit, and finally use Karnaugh maps to reduce the circuit to the minimal number of gates.

1. Produce a truth table (or use one that's given)
2. Express truth table behavior as a boolean expression
3. Convert this expression into gates

Instead of repeatedly drawing NOT gates in this course, we say that $\neg A$ or A' or $\overline{A}$ is the **complemented form** of A.

Definition 2.3 (Minterm)**.** An AND expression with every input present in true or complemented form

Definition 2.4 (Maxterm)**.** An OR expression with every input present in true or complemented form.

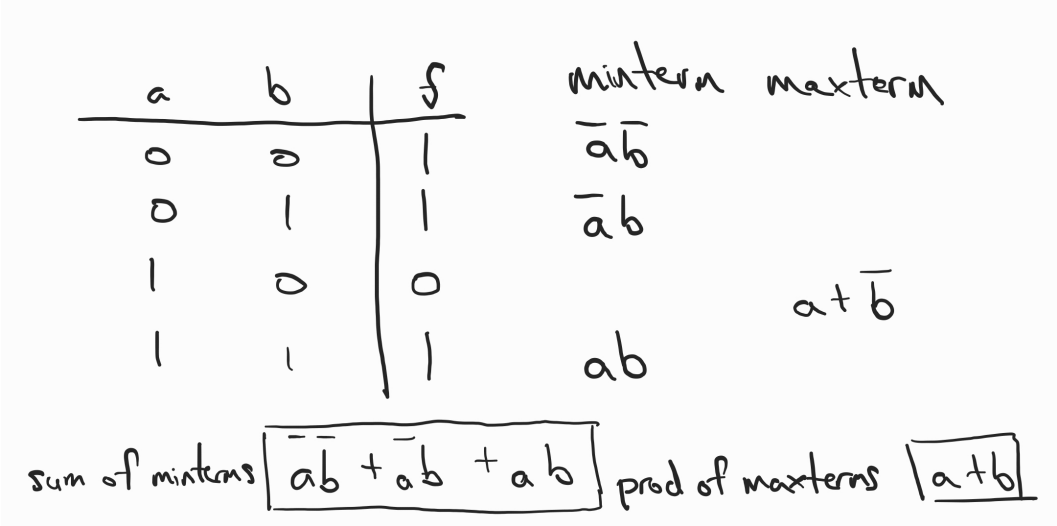
Given n variables, to be a minterm/maxterm, all n variables must be used. There exist exactly 2^n minterms and maxterms. Minterms and maxterms are also the **dual** of each other.[4]

Example 2.5 (Minterms and Maxterms)
Minterm: $A \cdot \overline{B} \cdot C \cdot D$
Maxterm: $A + \overline{B} + C + \overline{D}$

In a truth table, a row corresponds to both a minterm m_i and a maxterm M_i. In this course we mainly care about minterms, but maxterms are also testable. m_i corresponds to the row in *binary*. m_{15} would describe the case where the output is low at all times except when $A = B = C = D = 1$. Maxterms are capitalized, such as $M_0 = A + B + C + D$, which is always high except for the case where all four input values are low.

If row i corresponds to 1, then the minterm m_i should evaluate to 1, consisting of the product of all variables A_i, with A_i negated if the input is 0 for that row. If row i corresponds to 0, then the maxterm M_i should evaluate to 0, with the sum of all variables $\overline{A_i}$, with a variable not being negated when the input for it in that column is 0.

[4] reminds me of disjunctive and conjunctive normal forms



The goal is to join minterms and maxterms together to capture the truth table. The purpose of a minterm is to be true ONLY for that specific combination of input values. For a maxterm, it is to be false ONLY for that combination of input values. We can take sums (disjunctions) of minterms, to ensure that the expression is true iff it corresponds to a combination that makes it true. Can also take products (conjunctions) of maxterms, so that the expression is true iff everything is true, meaning the input values don't correspond to a false value. Both describe correct behavior.[5]

§2.3 Back to Gates

Once we have a sum-of-minterms expression, we can easily convert it to the equivalent combination of gates. Using DeMorgan's laws, we can convert everything to NAND gates, which are the cheapest to manufacture. Prof left proof as an exercise, which would solve problem 2.2.

To quantify how 'simple' an expression is, we look at the **gate cost** (G) or the **gate cost with NOTs** (GN). To make something simpler, we attempt to combine terms.

A tool to derive the *simplest possible circuit* is called the **Karnaugh map**. It's not always clear from looking at the truth table immediately, to draw a conclusion about which rows can be combined and simplified. Instead we can use a K-map to represent the same information, in a format that is easier to digest.

Definition 2.6 (Karnaugh Map)**.** K-maps are a grid of minterms/maxterms, arranged such that adjacent column/row labels in the grid differ by a single literal.[6]

With K-maps we wish to contain the 1s in boxes, satisfying some rules:

- Boxes must be rectangular, aligned with the map
- Number of values contained within each box must be a power of 2
- Boxes may overlap with each other
- Boxes may wrap around the edges of the map

You can also use K-maps for max-terms, where you group zeroes together instead. There are theorems that show that the order of choosing the row and column labels of the K-map doesn't affect the final result, but the prof decided not to show them in class.

[5] would clean this explanation up

[6] Rows and columns are ordered by Gray code instead of the usual binary, because this ordering has desirable properties.

§3 Day 3: Logical Devices (Sep 16, 2025)

To avoid having to deal with gates, we gain another level of abstraction through circuits, which are more complex structures. These include multiplexers (aka mux), decoders, adders (half and full), subtractors, and comparators.

Definition 3.1 (Combinational Circuit)**.** A circuit where the output strictly relies on the inputs.

Another category is *sequential circuits* which we will learn about later, which rely on memory. De-multiplexers may be testable, implement one in your own time.

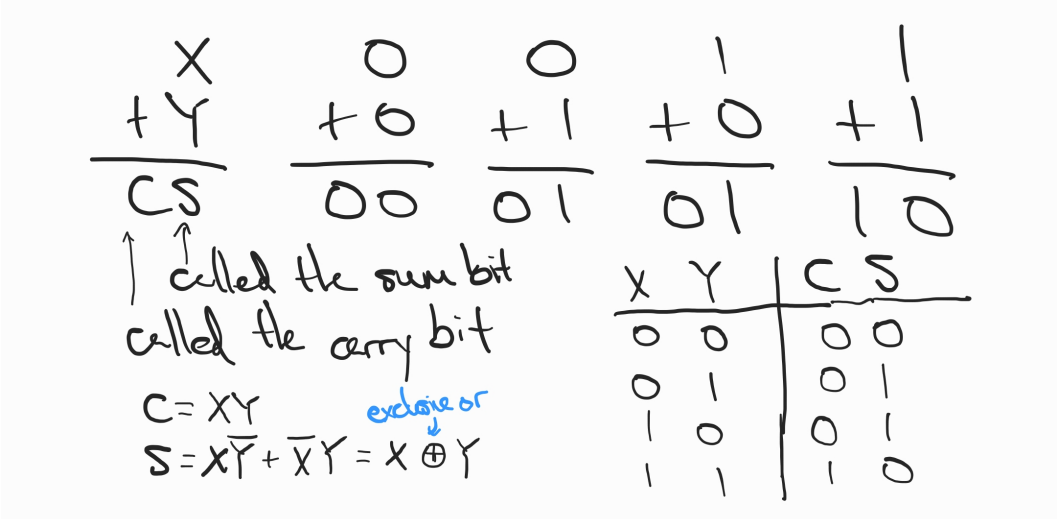
Decoders are translators, which translates the output of one circuit to the input of another. A 7 segment decoder, aka a 7 segment is active-high, meaning that a segment is on when its input is high, and off when its input is low. Active-low also exists, mainly used for safety purposes.

In a Karnaugh map, undefined inputs are represented by an `x`. We do not care about this value, we can make it 0 or 1, whichever to achieve a simpler expression (bigger rectangles = simpler expressions). Karnaugh maps are not a proof technique, they are a somewhat quick and dirty way to get a simple looking expression.

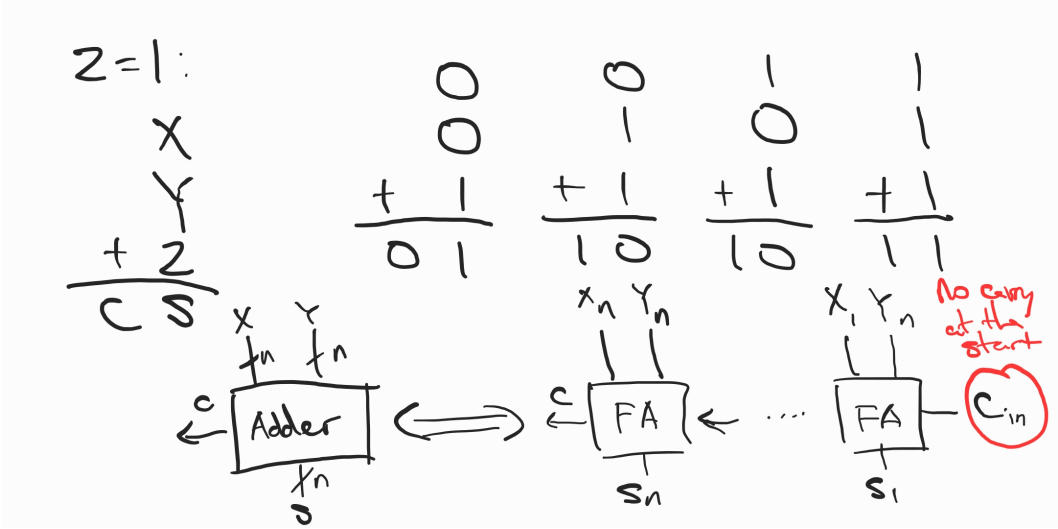
> **Remark 3.2.** If you are so *talented* that you can immediately identify the simplified expression, the steps are NOT for you. You wouldn't need them anyway. - Prof

§3.1 Adder Circuits

Aka binary adders, they are circuit devices that add two digits together. There exist half and full adders. Single digit unsigned binary addition is straightforward, which we capture using a half adder:



To account for the carry, we need potentially sum up 3 numbers! A full adder has inputs X, Y, Z and outputs C, S. When Z is 0, it behaves identically to a full adder. A full adder takes into consideration the carry of the sum of the previous two digits.



As shown in the diagram, to make use of the carry functionality provided by the full adder, we join them together to compute the sum of two n digit numbers, though you do also need to consider the carry.

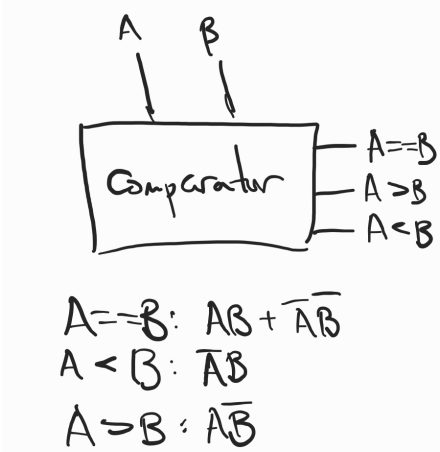
Now that we have an adder, we should look into subtraction. Recall that a negative integer is the additive inverse for a positive integer, meaning $(-a) + a = 0$.

When you flip all of the bits of a int, the number you obtain is called one's complement. When you add one to one's complement, you get *two's complement*. While using two's complement, to distinguish negative numbers from positive numbers we need to keep track of a sign-bit, typically the most significant bit. With a n-bit computer, you can only keep track of as big of a positive integer as a $(n-1)$-bit computer can.

Remember that in an adder circuit, c_{in} is unused, but we can reuse this feature in our subtractor circuit, where for two's complement need to flip all the bits for a number and add one, which c_{in} can do.

§3.2 Comparator

Comparators are crucial, they are used extremely often and are used to inform decisions. Comparators take two input vectors, and determines which one is bigger than or equal to the other. Recall that for any 2 integers A, B, exactly one of $A = B$, $A > B$, $A < B$ is true. We can implement a one bit comparator as follows:



A	B	Output
0	0	A==B
0	1	A<B
1	0	A>B
1	1	A==B

Exercise 3.3. Generalize to a n-bit comparator.

Exercise 3.4. Create a circuit that supports both addition and subtraction of n-digit binary numbers.

Lab Info

In the first part of the lab you will learn to make your designs modular, and create a 4-to-1 mux using 2-to-1 muxes.

In the second part of the lab you will be dealing with a field programmable gate array (FPGA), which is like a CPU, but you can re-wire it in the field. It gives you control over some gates. For our lab, the 7 segment display that we'll be working with is `HEX0`, which is the rightmost one. We're also concerned with the switches and LEDs, through which we handle input. **The DE1-SOC board is active-low**.

§4 Day 4: Sequential Circuits (Sep 23, 2025)

Recall that sequential circuits depend on more than just the input: they also depend on the memory. Suppose that you wanted to build a 'tickle-me elmo'. Your contraption also needs to keep track of how many times it's been tickled.

The sequential circuit will have some *feedback loop*. The stored information gets processed as 'input' again at some point. We also need some delay between when the information is read-in the first time, and when it is re-processed again.

Even in combinational circuits, outputs do not change instantly. There exists gate/propagation delay, which is the length of time it takes for an input to change to result in a corresponding output change. We call the time where the input changes T, and the time where the output changes/finishes processing $T + 1$.

Some gates possess useful properties when outputs are fed back in on inputs.

AND, OR Gates

NAND Gates

If we leave A unchanged, we can store the value of in Q unchanged indefinitely. There exists waveform behavior, where whenever A is high, Q (as a whole) is oscillating. Due to propagation delay, this process isn't instant in practice, although it should be in theory. This behavior is not quite right for storing memory though.

§4.1 $\overline{SR}$ Latches

The reason we negate S and R here is so that set retains its original meaning. If we didn't, Q would actually be storing the negation of the value we are setting, which complicates things.

We claim that inputs of 11 retain the previous input state,

If you want to store 1, set $\overline{S}$ to 1.

Due to physical limitations, we cannot oscillate the values of set and reset too quickly, because of gate-propagation related delays.

§4.2 Clock Signals

Clocks are 'regular' pulse signals, the high value indicates when to update the output of the latch. The frequency is the number of pulses per second, measured in hertz (Hz), which is s^{-1}.

Clocked SR latches gives us a control input signal C.

Exists a forbidden state, S, R, C being all high at the same time. Then we get $Q = \overline{Q}$, which is undesired. Introduce D-latches (aka gated D latch), which prevent this very thing from happening.

As a somewhat unfortunate side effect, we cannot have 00 anymore, but the pros outweigh the cons.

We then encounter timing issues, in each clock cycle, we may set something a bajillion times.

Definition 4.1 (Transparent)**.** Any changes to the inputs of a D-latch are visible to the output when the control signal (clock) is 1.

Latches are transparent, the output values of this clock cycle will also affect the computations of this cycle.

§4.3 Devices

We have a device that implements flip-flops, but how do we use them to actually remember things? A **shift register** is a series of D flip-flops, which can store a multi-bit value. Data can be shifted into this register, over n clock cycles total for a n bit integer.

With this implementation, we cannot write instantaneously. It would take n clock cycles to write into this register. Thus a faster clock cycle (higher Hz) is desirable, making operations take less time.

However, we could've done this all in 1 clock cycle with another implementation, called a n-bit **load register**, where we can individually load the values into the flip-flops in parallel. But then we need to control when