Logic Gates and Circuits

CSCB58: Computer Organization

Lecture 1: January 7th, 2025

Computer Science
UNIVERSITY OF TORONTO

A little bit about your instructor

Nandita Vijaykumar

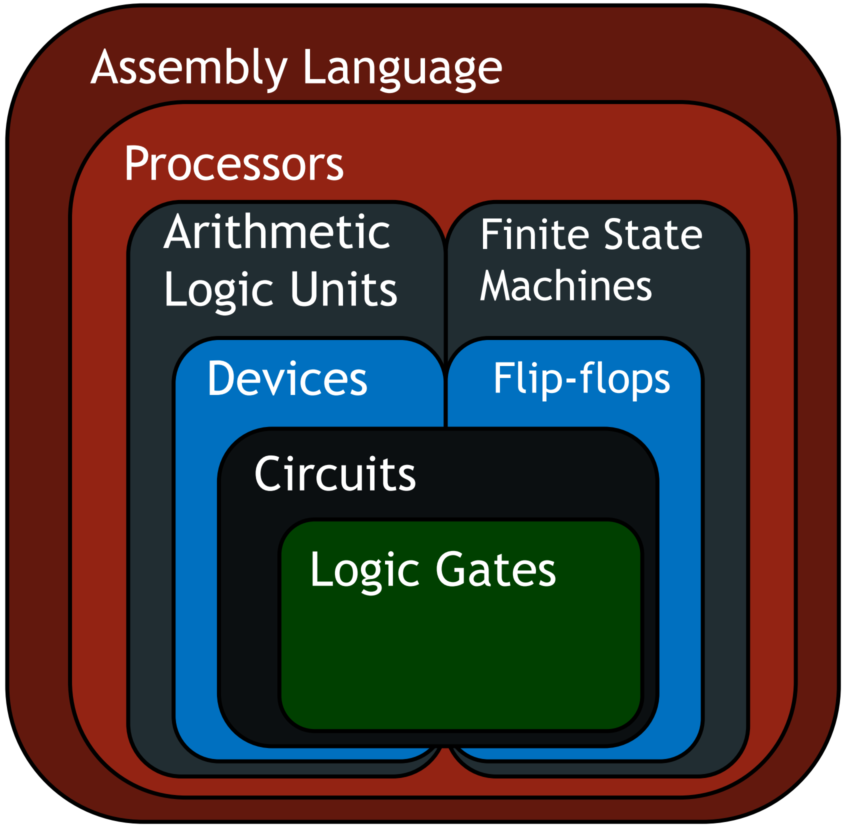
- Assistant Professor in the Computer Science Department at UofT and UTSC, since 2020
- PhD from Carnegie Mellon University
- Worked at Intel, AMD, Microsoft, and Nvidia
- Best way to reach me: nandita@cs.toronto.edu
- My research group: embARC Lab (www.embarclab.com)
- Our research: computer systems + architecture
  - Cross-layer spanning applications, systems, and hardware
  - Application-specific: E.g., machine learning, graphics, HPC, robotics

My research group: embARC

Today!

- Why take B58?
- What's in B58?
- How does the course work?
- Logic gates and circuits

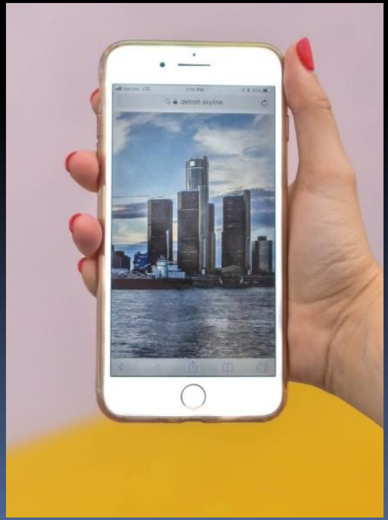
The course at a glance…



CSCB58 Is Intense

- There is a lot to learn, practice, and do.
- You have tasks every week.
- It is historically one of the busiest courses for students.
- But there is good news:
  - Material not hard to understand, but it does require practice and effort.
  - All this practice makes you really good.
  - Course grade tend to skew high

What is a computer?



do you need electricity for a computer?

CSCB58 Asks the Big Questions

- What is a computer?
- What is memory? How does a computer store information
- Why do computers work in binary?
- how does a computer actually… compute?
- How does the code we write in Python/Java/C++/etc actually translate into things happening?

Why take CSCB58?

- To better understand computers!
- See what's going on "under the hood"
- Open the black box, get rid of the mystery
- Understand the whole pipeline, from atoms to assembly
  - Everything above assembly is virtualization and abstraction
  - Everything else is an **illusion!**
- Build a cool software-based project!

CSCB58 Course Goals

- Learn to build computers…
  - Understand and design the underlying architecture of computer systems.
- … and how they work.
  - How programs use digital structures to do computation.
- Learn engineering.
  - Build systems from components, understand abstractions, make tradeoffs.
- Learn **there is no magic!**
  - There is nothing you cannot understand.

CSCB58 Course Goals

- By the end of this course, you will be able to build a computer from the atomic level upwards*
  - Build circuits from logic gates
  - Build memory/computational units from circuits
  - Build a computer from memory/computational units

(* Given an infinite amount of time, and the ability to manipulate individual atoms.)

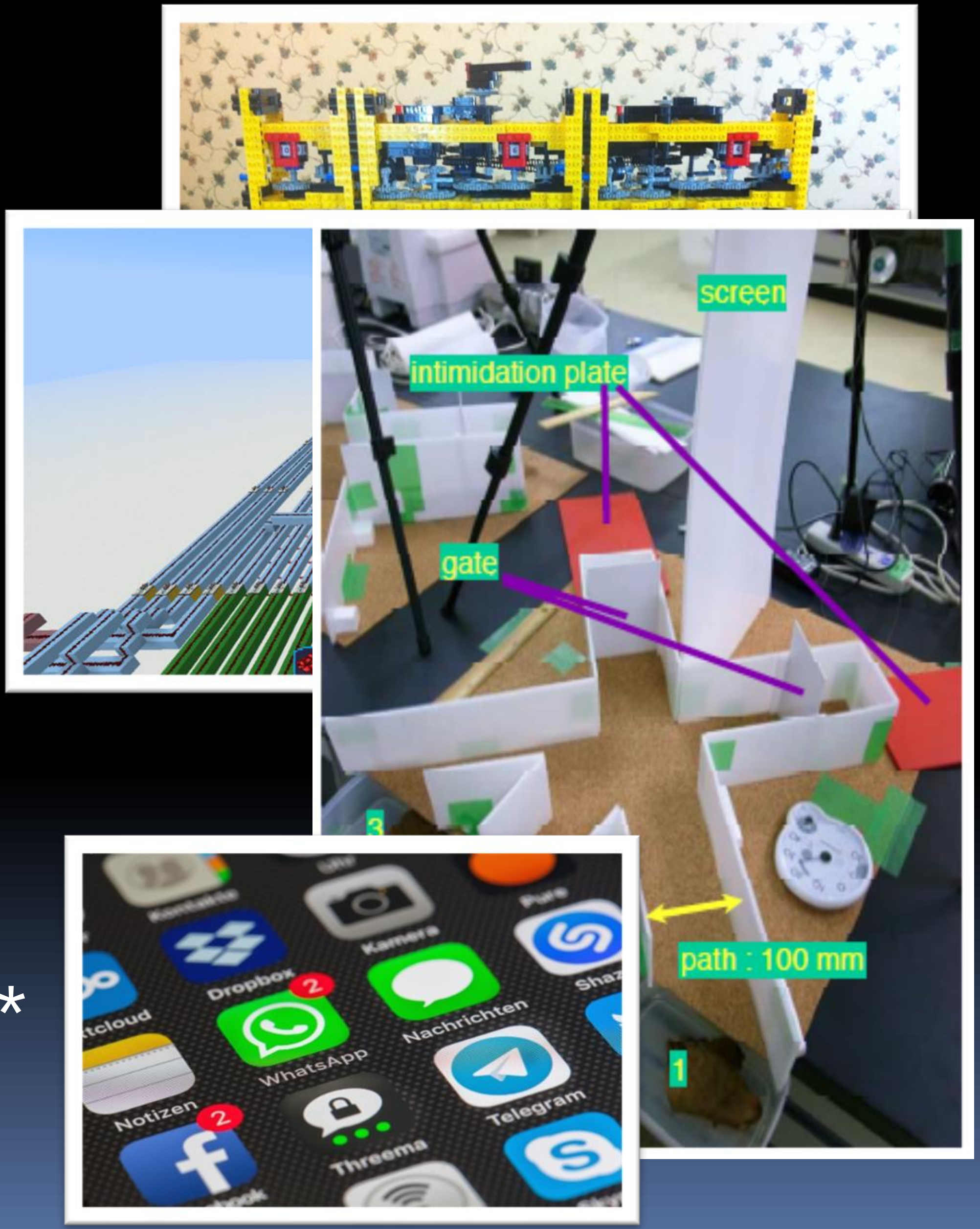
Build a Computer!

- …from LEGO
- …in Minecraft
- …from living crabs*

* Do not build a computer from live crabs. It is unethical.

- …inside an iMessage to break into iPhones**

** also unethical.



Some Admin

https://villains.fandom.com/wiki/Vogons

Administrivia

- This course is on Quercus: q.utoronto.ca
  - Check it regularly!
- Get help on Piazza
  - Use for all assignment or technical questions.
  - Read pinned posts (first posts at the top) before asking questions!
  - Private posts for questions relating to you only
- E-mail (use official email only!)
  - Only for emergencies and confidential personal matters
- Office hours
  - Dedicated lab office hours by TAs (for lecture material, labs, and assignments)
  - Instructor office hours (for lecture material and personal situations)

Course Components

- Lectures
  - 2 hour in person lectures every week on Tuesday
  - No TUT sessions
- Quizzes 10%
  - 11 short weekly quizzes on Mondays (starting next week)
- Labs 40%
  - 5 labs starting this week (first is very easy, don’t worry)
- Project 17%
  - 3-week assembly project
- Final exam 33%
  - Final exam - must get 30% to pass the course

Administrivia

- Some general principles for how we will operate:
- No exceptions other than AccessAbility and for medical emergencies
- We want consistent and fair policies for all!
- Course requirements: All requirements and prerequisites must be met

Labs Marking

- Submit each lab solution to Quercus at each due date (TBD)
  - Usually a Logisim file, sometimes also a PDF.
  - Late? You will lose marks.
- Each students has bi-weekly 10-minute interview slot with your TA to demonstrate:
  - … your solution.
  - … basic understanding of the material.
    (by answering oral questions from the TA).
  - Your mark is based on both!

Missed Term Work

- Life happens. We understand.
- We give you justification-free tokens.
- Can miss/do badly on 2 quizzes and 1 lab.
  - For any reason. We don't need to know.
- Works automatically, no need to tell us.
  - We drop lowest 2 quizzes and lowest 1 lab
  - You just focus on getting better.
- Do not waste or abuse them!
  - We will not give extra "tokens", no matter what.
  - Best strategy: show up and do your best.
- No make up for late work. Late = missed.

How to Succeed in Labs?

- The labs are not hard, but they require work.
- Labs are your chance to practice.
  Do the work yourself.
  - Submit something meaningful and on time.
- Don’t plagiarize
  - You won’t be able to answer TA questions…
  - … or answer the next quiz …
  - … or the final exam.

Lab Software

- Lab 1-3 use Logisim-evolution for hardware.
  - Lab 1 will tell you all you need to know.
  - Do not use the original Logisim or its other variations of it. Use the in the lab PDF file or the link above.
- Labs 4-5 will use MARS simulator for MIPS assembly.
  - Same idea: link will be included.

Project

- Multi-week assembly programming project.
- Worth 17%
- At the end of the course.
- Done individually.
- Marking based on:
  - Short project report and video.
  - Submitted code
- Details will follow later.

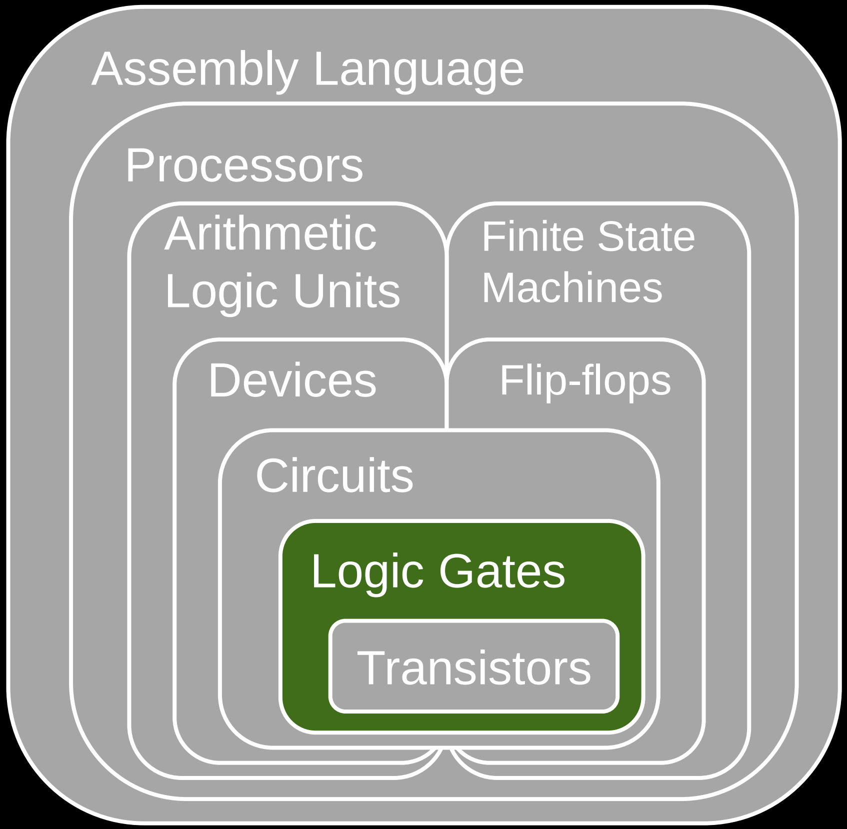
Final Exam

- Worth 33%
- Closed book.
- In-person
- Details will follow.

How to do well in CSCB58

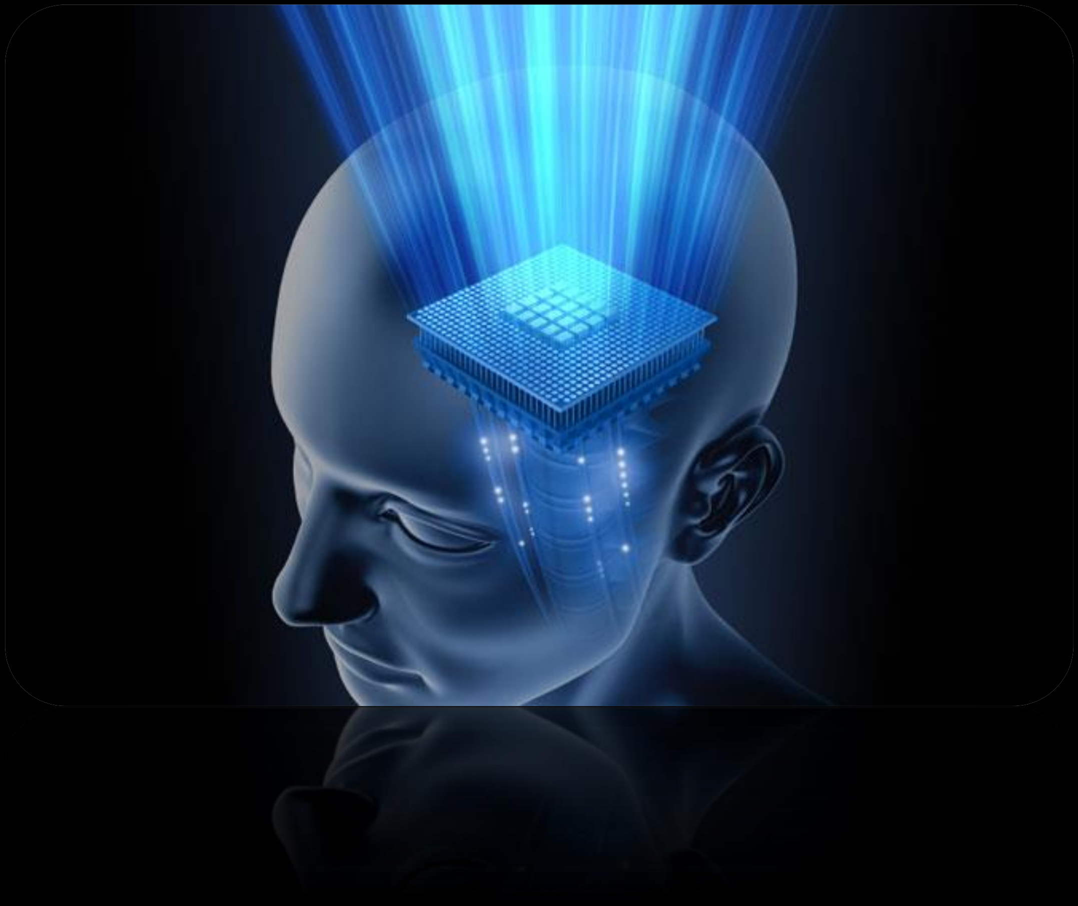
- Be Interested!
- Put in the effort:
  - Do labs!
  - Practice solving lecture questions on your own.
  - Study.
  - “In theory there’s no difference between practice and theory, but in practice there usually is”
- Interact
  - lectures, labs (TAs), Piazza
  - Ask questions in Piazza or review sessions.

This week



Assembly Language
Processors
Arithmetic Logic Units
Finite State Machines
Devices
Flip-flops
Circuits
Logic Gates
Transistors

You already know some important concepts…

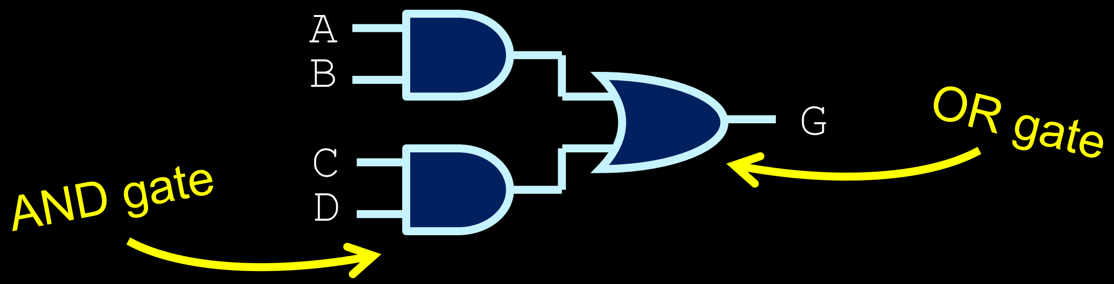


Boolean Logic from CSCA67

- <u>Example:</u> Create an expression that is true if both variables `A` and `B` are true, or if both `C` and `D` are true.

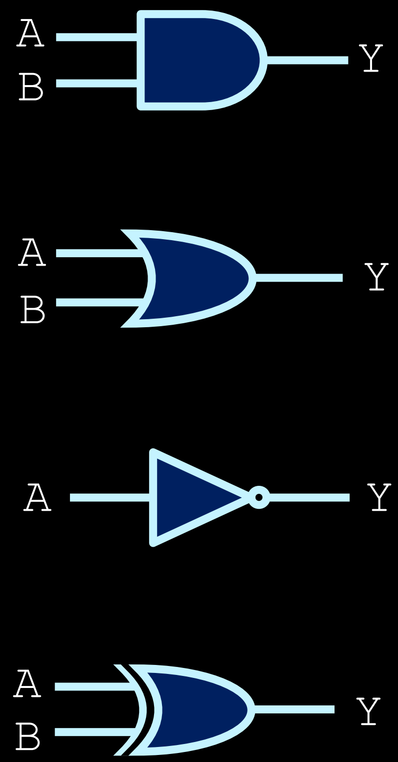
```
G = A & B | C & D
```

- Now create a circuit that does the same thing:

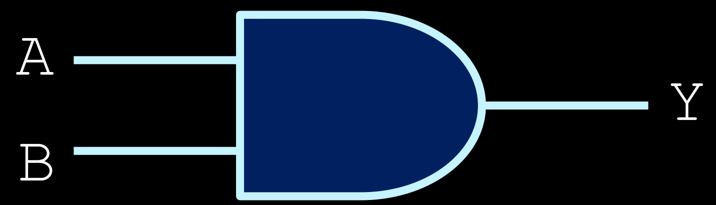


Logic Gates

- If you know how to create simple logical expressions, you already know the basics of putting logic gates together to form simple circuits.
- Just need to know which logic operations are represented by which gate!



AND Gate



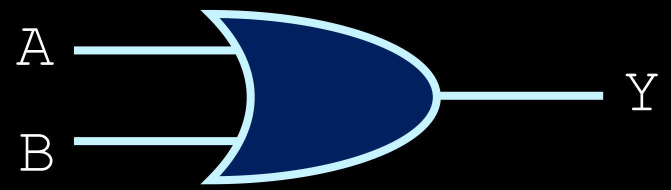
inputs outputs

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

truth table
For every combination of **inputs**, what are the **outputs**?

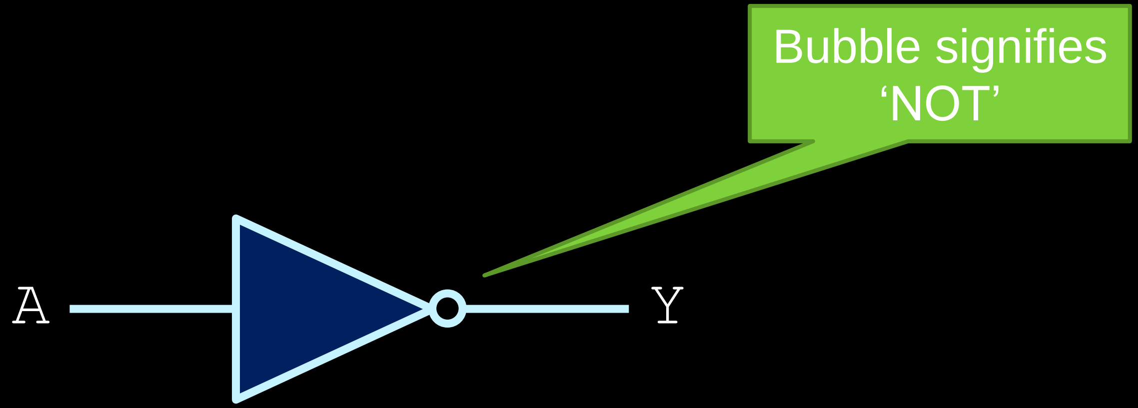
If the truth table for two circuits is the same, they are equivalent

OR Gate



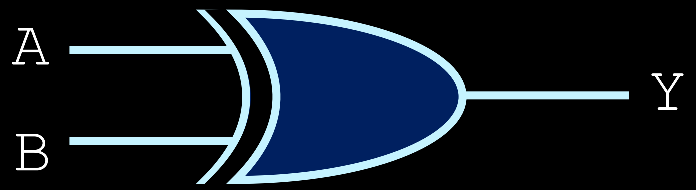
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gates



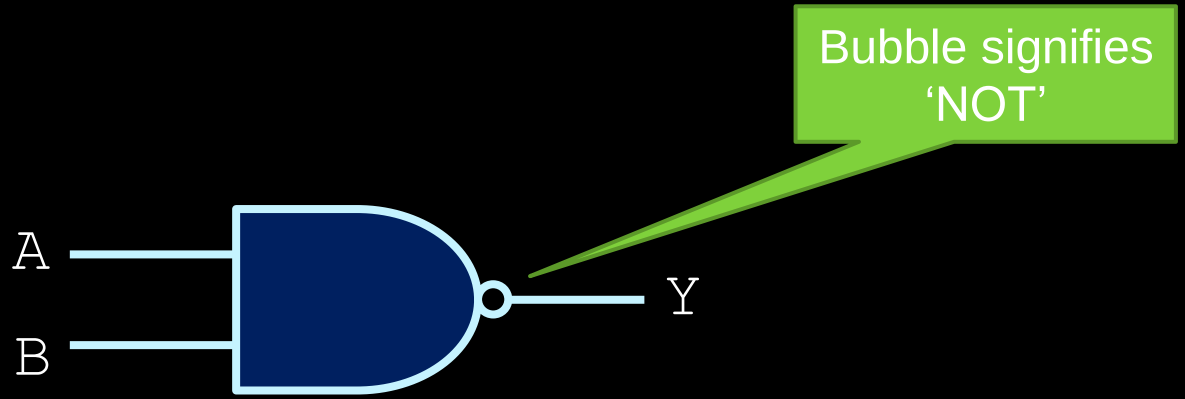
A	Y
0	1
1	0

XOR Gates



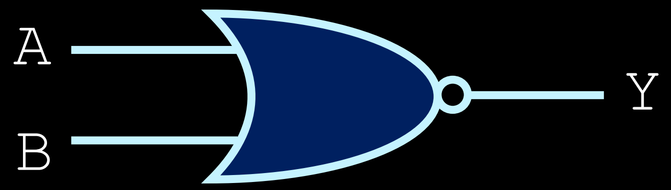
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

NAND Gates



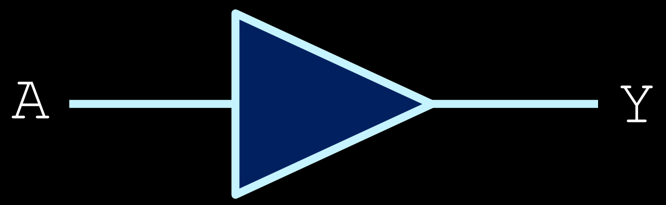
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gates



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Buffer



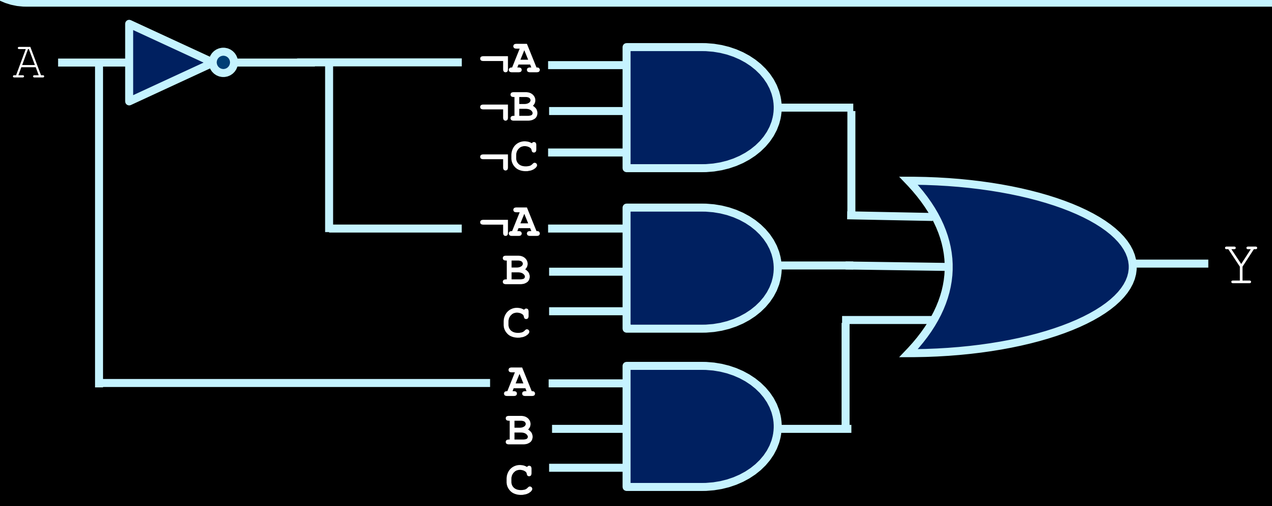
This is not as silly as you might think now, as we'll see later…

A	Y
0	0
1	1

Circuit Design?

- Creating circuit logic can be similar to creating Boolean logic in Python, C or Java:

```
Y = (!A and !B and !C) or
    (!A and B and C) or
    (A and B and C)
```

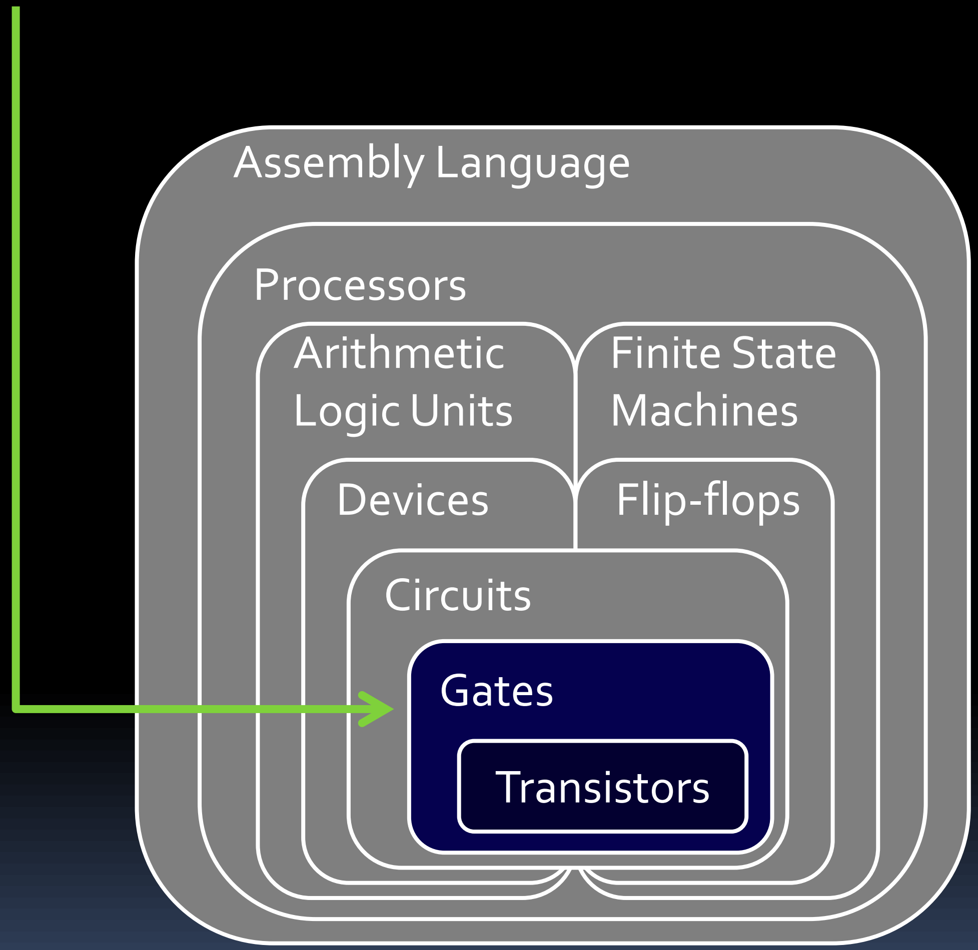


The real challenge in circuit design…

- Given a truth table or description…
- …find a circuit that implements it.
- Many ways of tackling the problem

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

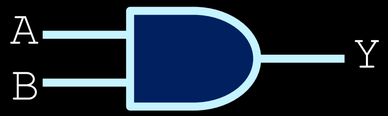
You are here


Assembly Language
Processors
Arithmetic Logic Units
Finite State Machines
Devices
Flip-flops
Circuits
Gates
Transistors

Making logic with gates

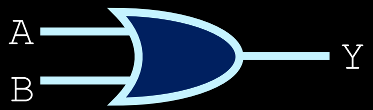
- Logic gates create an output value, based on one or more input values.
- These correspond to Boolean logic that we've seen before in CSCA08/A48/A67:

AND



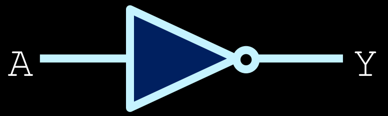
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

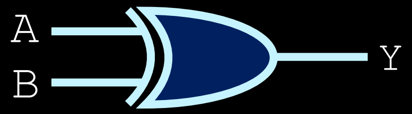
NOT



A	Y
0	1
1	0

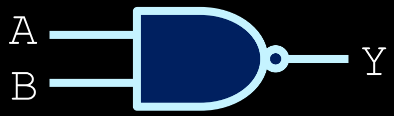
Other gates

XOR



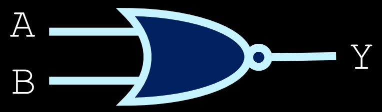
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

NAND



A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NOR



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Aside: notation

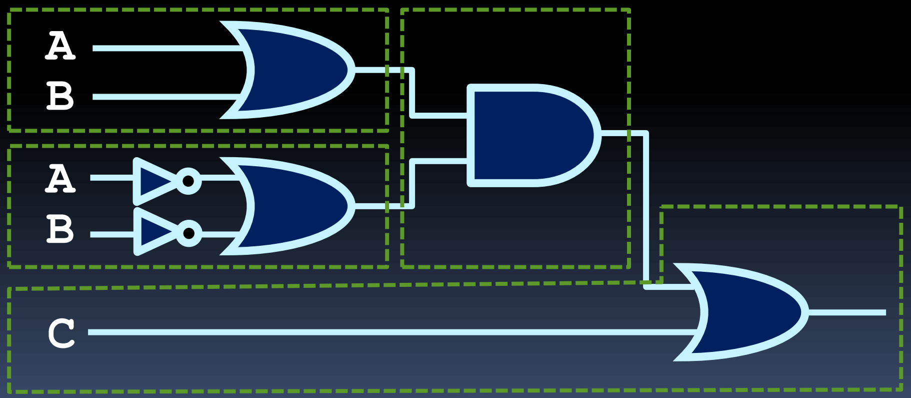
- While we're talking about notation…
  - AND operations are denoted in these expressions by the multiplication symbol.
    - e.g. $A \cdot B \cdot C$ or ABC or $A*B*C$ $\approx$ $A \wedge B \wedge C$
  - OR operations are denoted by the addition symbol.
    - e.g. $A+B+C$ $\approx$ $A \vee B \vee C$
  - NOT is denoted by multiple symbols.
    - e.g. $!A$ or $\sim A$ or $\overline{A}$ or A' or $\neg A$
  - XOR occurs rarely in circuit expressions.
    - e.g. $A \oplus B$

Making Boolean expressions

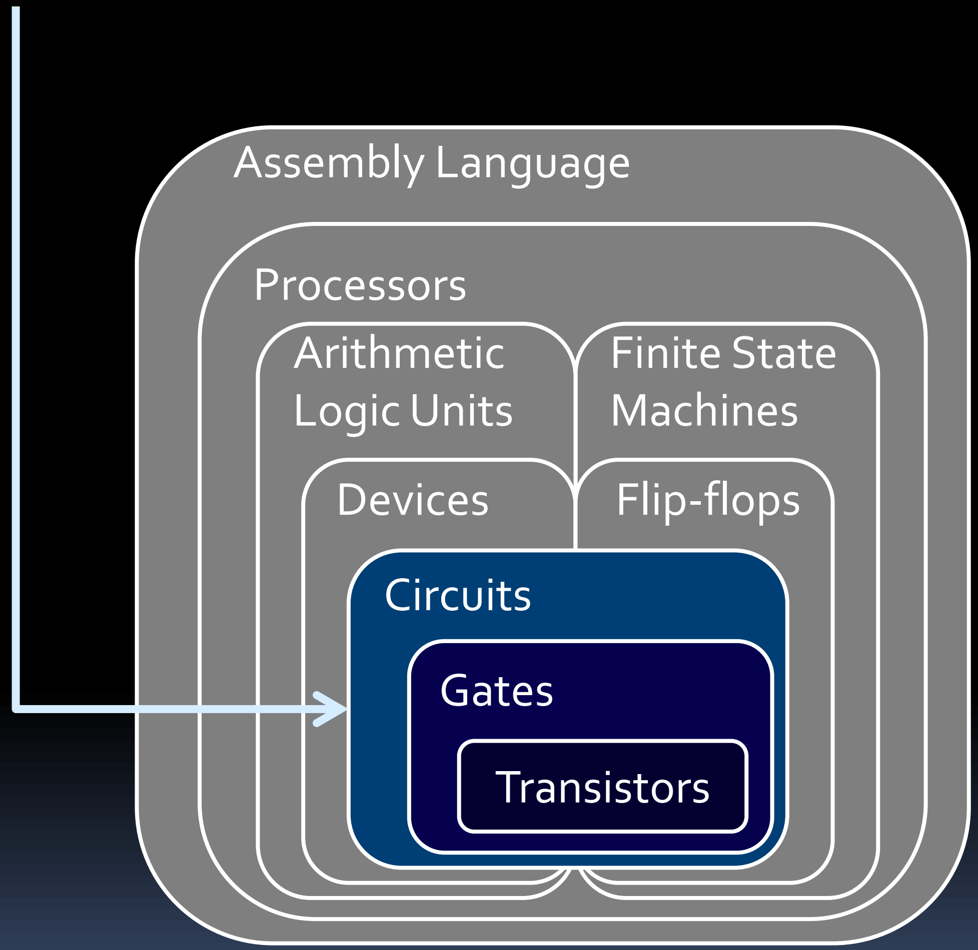
- So how would you represent boolean expressions using logic gates?

```
Y = (A or B) and (not A or not B) or C
```

- Like so:

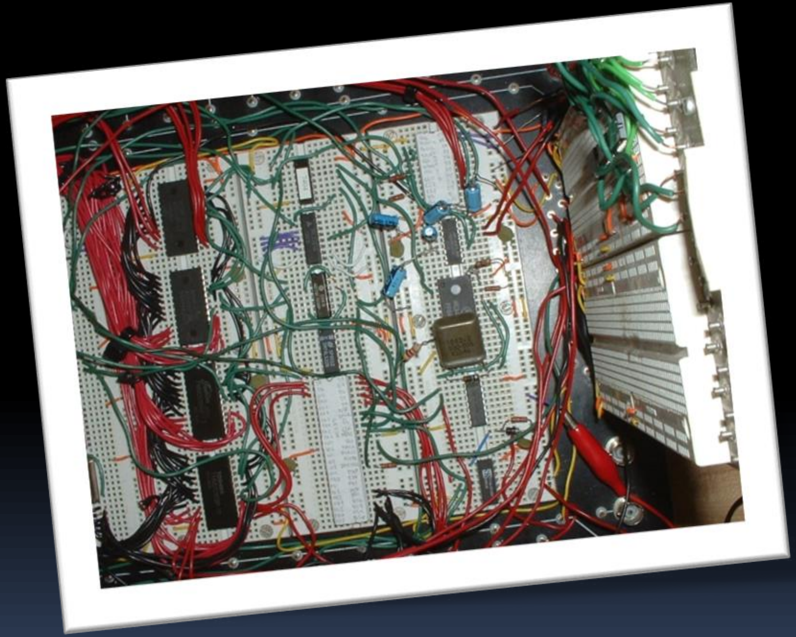


Now you are here



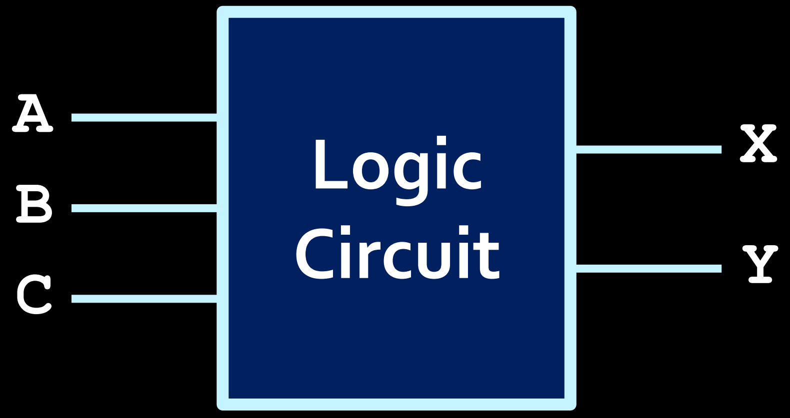
Creating complex circuits

- What do we do in the case of more complex circuits, with several inputs and more than one output?
  - If you're lucky, a truth table is provided to express the circuit.
  - Usually the behaviour of the circuit is expressed in words, and the first step involves creating a truth table that represents the described behaviour.



Circuit example

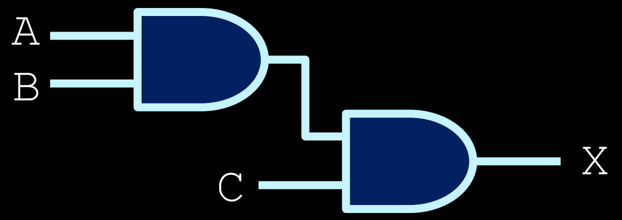
- The circuit on the right has three inputs (A, B and C) and two outputs (X and Y).

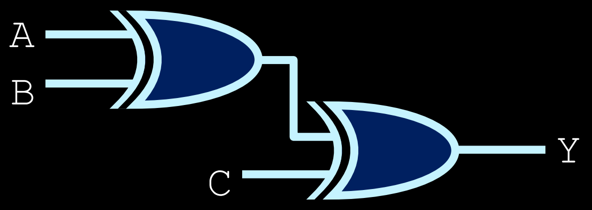


- What logic is needed to set X high when all three inputs are high?
- What logic is needed to set Y high when the number of high inputs is odd?

Combinational circuits

- Small problems can be solved easily.

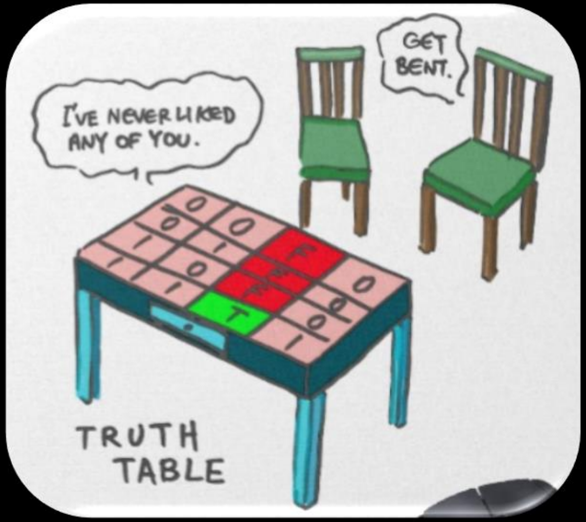




- Larger problems require a more systematic approach.
  - <u>Example:</u> Given three inputs A, B, and C, make output Y high in the case where all of the inputs are low, or when A and B are low and C is high, or when A and C are low but B is high, or when A is low and B and C are high.

Creating complex logic

- How do we approach problems like these (and circuit problems in general)?
- Basic steps:
  1. Create truth tables.
  2. Express as Boolean expression.
  3. Convert to gates.
- The key to an efficient design?
  - Spending extra time on Step #2.



Example truth table

- Consider the following example:
  - *"Given three inputs A, B, and C, make output Y high wherever any of the inputs are low, except when all three are low or when A and C are high."*
  - This leads to the truth table on the right.
  - Is there a **more compact way** to describe this function?

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Warm-Up Exercise

For each of the following logic expressions, what are the A, B, C values that make the expression evaluate to **1** ?

A'B'C'

ABC

A'BC

ABC'

Answers to Warm-Up Exercise

For each of the following logic expressions, what are the A, B, C values that make the expression evaluate to **1** ?

A'B'C'

- A=0, B=0, C=0 , and only this!

ABC

- 111 and only this!

A'BC

- 011 and only this!

ABC'

- 110 and only this!

Maxterms, informally

- Assume a standard truth table format.
  - Sort rows as if input ABC is binary number.
- Maxterms tell us which rows have low output.
  - These rows are referred to as maxterms.
- Express circuit behaviour by listing those rows.
  - In this example, we only need maxterms M_0 M_5 M_7

Row index	A	B	C	Y
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0

Maxterm	Y
M_0	0
M_1	1
M_2	1
M_3	1
M_4	1
M_5	0
M_6	1
M_7	0

Minterms, informally

- A more popular alternative:
  list which input rows cause high output.
  - These rows are referred to as minterms.
  - In this case we have the minterms m_1 m_2 m_3 m_4 m_6

Row index	A	B	C	Y
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0

Minterm	Y
m_0	0
m_1	1
m_2	1
m_3	1
m_4	1
m_5	0
m_6	1
m_7	0

Minterms and maxterms

- A more formal description:
  - minterm = an AND expression with every input present in true or complemented form.
  - maxterm = an OR expression with every input present in true or complemented form.
- For example, given four inputs (A, B, C, D):
  - Valid minterms:
    - $A \cdot B \cdot \overline{C} \cdot D$, $A \cdot \overline{B} \cdot C \cdot \overline{D}$, $A \cdot B \cdot C \cdot D$
  - Valid maxterms:
    - $A+B+\overline{C}+D$, $A+\overline{B}+C+\overline{D}$, $A+B+C+D$
  - Neither minterm nor maxterm:
    - $A \cdot B+C \cdot D$, $A \cdot B \cdot D$, $A+B$

Naming

- Given n inputs, there are 2^n minterms and maxterms possible (same as rows in a truth table).
- minterms are labeled as m_x
  maxterms are labeled as M_x
  - The x subscript indicates the row in the truth table.
  - x starts at 0 and ends with $n-1$.
- Minterms are about when output is **1**:
  - From m_0 for $\overline{A} \cdot \overline{B} \cdot \overline{C}$ to m_7 for $A \cdot B \cdot C$
- Maxterms are about when output is **0**:
  - M_0 $(A+B+C)$ to M_7 $(\overline{A}+\overline{B}+\overline{C})$

$n=3$

m_0 vs M_0

- Minterm m_0 is $\bar{A}$ and $\bar{B}$ and $\bar{C}$
  - $m_0 = 1$ if and only if: $A = B = C = 0$ (row 0)
- Maxterm M_0 is A or B or C
  - $M_0 = 0$ if and only if: $A = B = C = 0$ (row 0)
- Minterms tell us when the output is 1
- Maxterms tell us when the input is 0

Examples

For example, given four inputs (A, B, C, D):

- Valid minterms:
  - $A \cdot B \cdot \overline{C} \cdot D$ — m_{13} (because 1101 = 13)
  - $A \cdot \overline{B} \cdot C \cdot \overline{D}$ — m_{10} (because 1010 = 10)
  - $A \cdot B \cdot C \cdot D$ — m_{15} (because 1111 = 15)
- Valid maxterms:
  - $A+B+\overline{C}+D$ — M_2 (because 0010 = 2)
  - $A+\overline{B}+C+\overline{D}$ — M_5 (because 0101 = 5)
  - $A+B+C+D$ — M_0 (because 0000 = 0)
- Neither minterm nor maxterm:
  - $A \cdot B+C \cdot D$ — mixes AND and OR
  - $A \cdot B \cdot D$
  - $A+B$ — missing some of the inputs

Quick Exercises

- Given 4 inputs A, B, C and D write:
  - m_9 =>
  - m_{15} =>
  - m_{16} =>
  - M_2 =>
- Which minterm is this?
  - $\overline{A} \cdot B \cdot \overline{C} \cdot \overline{D}$
- Which maxterm is this?
  - $A+B+C+\overline{D}$

What is This For?

- Recap:
  - Minterms and maxterms are a shorthand to refer to rows of the truth table.
  - minterms describe rows where output is high.
  - maxterms describe rows where output is low.
- Use minterms and maxterms to go from truth table to logic expression
  - Define output by OR-ing minterms or AND-ing maxterms.
  - Don’t mix them both

Using minterms and maxterms

- What are minterms used for?
  - A single minterm indicates a set of inputs that will make the output go high.
  - Example: m_2
  - Output only goes high in third row of truth table.

Row index	A	B	C	D	m_2
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

Using minterms and maxterms

- What happens when you OR two minterms?
- Result is output that goes high in both minterm cases.
- For m_2+m_8, both third and ninth rows of truth table result in high output.

	A	B	C	D	m_2	m_8	m_2+m_8
0	0	0	0	0	0	0	
1	0	0	0	1	0	0	
2	0	0	1	0	1	0	
3	0	0	1	1	0	0	
4	0	1	0	0	0	0	
5	0	1	0	1	0	0	
6	0	1	1	0	0	0	
7	0	1	1	1	0	0	
8	1	0	0	0	0	1	
9	1	0	0	1	0	0	
10	1	0	1	0	0	0	
11	1	0	1	1	0	0	
12	1	1	0	0	0	0	
13	1	1	0	1	0	0	
14	1	1	1	0	0	0	
15	1	1	1	1	0	0	

Creating Boolean expressions

- Two canonical forms of Boolean expressions:
- Sum-of-Minterms (SOM):
  - Since each minterm corresponds to a single high output in the truth table, the combined high outputs are a union of these minterm expressions.
  - Also known as: Sum-of-Products.
- Product-of-Maxterms (POM):
  - Since each maxterm only produces a single low output in the truth table, the combined low outputs are an intersection of these maxterm expressions.
  - Also known as Product-of-Sums.

$$Y = m_2 + m_6 + m_7 + m_{10} \quad \text{(SOM)}$$

	A	B	C	D	m_2	m_6	m_7	m_{10}	Y
0	0	0	0	0					
1	0	0	0	1					
2	0	0	1	0					
3	0	0	1	1					
4	0	1	0	0					
5	0	1	0	1					
6	0	1	1	0					
7	0	1	1	1					
8	1	0	0	0					
9	1	0	0	1					
10	1	0	1	0					
11	1	0	1	1					
12	1	1	0	0					
13	1	1	0	1					
14	1	1	1	0					
15	1	1	1	1					

$$Y = m_2 + m_6 + m_7 + m_{10} \quad (SOM)$$

	A	B	C	D	m_2	m_6	m_7	m_{10}	Y
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0
2	0	0	1	0	1	0	0	0	1
3	0	0	1	1	0	0	0	0	0
4	0	1	0	0	0	0	0	0	0
5	0	1	0	1	0	0	0	0	0
6	0	1	1	0	0	1	0	0	1
7	0	1	1	1	0	0	1	0	1
8	1	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	0
10	1	0	1	0	0	0	0	1	1
11	1	0	1	1	0	0	0	0	0
12	1	1	0	0	0	0	0	0	0
13	1	1	0	1	0	0	0	0	0
14	1	1	1	0	0	0	0	0	0
15	1	1	1	1	0	0	0	0	0

$$Y = M_3 \cdot M_5 \cdot M_7 \cdot M_{10} \cdot M_{14} \text{ (POM)}$$

A	B	C	D	M_3	M_5	M_7	M_{10}	M_{14}	Y
0	0	0	0						
0	0	0	1						
0	0	1	0						
0	0	1	1						
0	1	0	0						
0	1	0	1						
0	1	1	0						
0	1	1	1						
1	0	0	0						
1	0	0	1						
1	0	1	0						
1	0	1	1						
1	1	0	0						
1	1	0	1						
1	1	1	0						
1	1	1	1						

$$Y = M_3 \cdot M_5 \cdot M_7 \cdot M_{10} \cdot M_{14} \text{ (POM)}$$

A	B	C	D	M_3	M_5	M_7	M_{10}	M_{14}	Y
0	0	0	0	1	1	1	1	1	1
0	0	0	1	1	1	1	1	1	1
0	0	1	0	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	0
0	1	0	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	0
0	1	1	0	1	1	1	1	1	1
0	1	1	1	1	1	0	1	1	0
1	0	0	0	1	1	1	1	1	1
1	0	0	1	1	1	1	1	1	1
1	0	1	0	1	1	1	0	1	0
1	0	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1	1

Using Sum-of-Minterms

- Sum-of-Minterms is a way of expressing which inputs cause the output to go high. Product-of-Maxterms is a way of expression which inputs cause the output to go low.
  - Assumes that the truth table columns list the inputs according to some logical or natural order.
- Minterm and maxterm expressions are used for efficiency reasons:
  - More compact that displaying entire truth tables.
  - Sum-of-minterms (SOM) are useful when very few input combinations that produce high output.
  - Product-of-maxterms (POM) useful when expressing truth tables that have very few low output cases.

Converting SOM to gates

- Once you have a Sum-of-Minterms expression, it is easy to convert this to the equivalent combination of gates:

$$m_0 + m_1 + m_2 + m_3 =$$

$$\bar{A} \cdot \bar{B} \cdot \bar{C} + \bar{A} \cdot \bar{B} \cdot C + \bar{A} \cdot B \cdot \bar{C} + \bar{A} \cdot B \cdot C =$$