Great Ideas in Computer Architecture

Combinational Logic

Instructor: Jenny Song

Review

- Compiler converts a single HLL file into a single assembly file `.c` → `.s`
- Assembler removes pseudo-instructions, converts what it can to machine language, and creates a checklist for linker (relocation table) `.s` → `.o`
  - Resolves addresses by making 2 passes (for internal forward references)
- Linker combines several object files and resolves absolute addresses `.o` → `.out`
  - Enable separate compilation and use of libraries
- Loader loads the executable into memory and begins execution `.out` → `[running]`

Agenda

- CALL Review
- **Hardware Design Overview**
- Switches and Transistors
- CMOS Networks
- Combinational Logic
  - Combinational Logic Gates
  - Truth Tables
  - Boolean Algebra
  - Circuit Simplification

Overview

Higher-Level Language Program (e.g. C)

↓ *Compiler*

Assembly Language Program (e.g. RISC-V)

↓ *Assembler*

Machine Language Program (RISC-V)

↓ *Machine Interpretation*

Hardware Architecture Description (e.g. block diagrams)

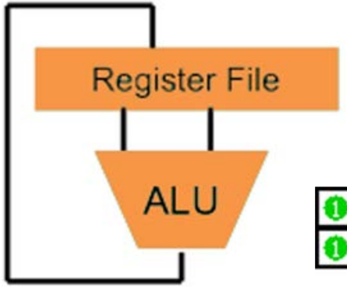
↓ *Architecture Implementation*

Logic Circuit Description (Circuit Schematic Diagrams)

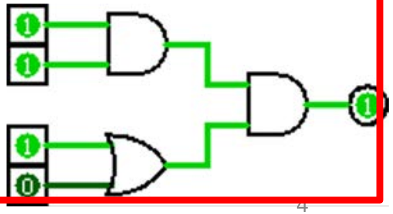
```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw    $t0, 0($2)
lw    $t1, 4($2)
sw    $t1, 0($2)
sw    $t0, 4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```



We are here



Why study hardware design?

- Be able to answer to the question “How does a computer work?”
- We need some digital systems knowledge to build our own processor
- Understand how code is actually executed on a computer
- Understand capabilities and limitations of HW in general and processors in particular
- What processors can do fast and what they can’t do fast (avoid slow things if you want your code to run fast!)
- Background for other more in-depth classes EECS151, CS152
- There is just so much you can do with standard processors: you may need to design own custom hardware for extra performance.

Synchronous Digital Systems (SDS)

Hardware of a processor, such as a RISC-V processor, is an example of a Synchronous Digital System

Synchronous:

- All operations coordinated by a central clock
  - “Heartbeat” of the system! (processor frequency)

Digital:

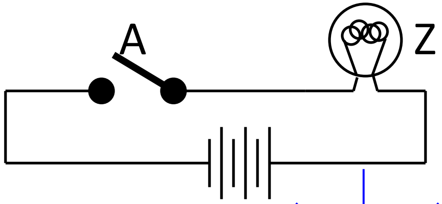
- Represent all values with two discrete values
- Electrical signals are treated as 1’s and 0’s
  - 1 and 0 are complements of each other
- High/Low voltage for True/False, 1/0

Agenda

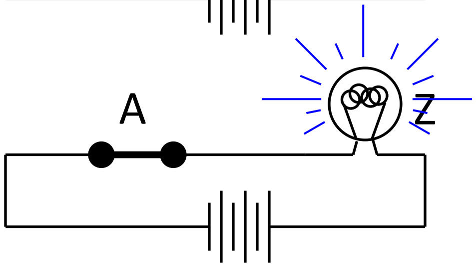
- CALL Review
- Hardware Design Overview
- **Switches and Transistors**
- CMOS Networks
- Combinational Logic
  - Combinational Logic Gates
  - Truth Tables
  - Boolean Algebra
  - Circuit Simplification

Switches

- The basic element of physical implementations
- Convention: if input is a "1," the switch is *asserted*



Open switch if A is "0" (unasserted) and turn OFF light bulb (Z)



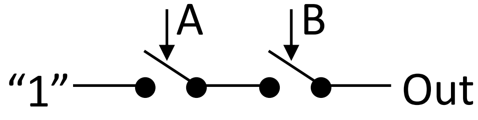
Close switch if A is "1" (asserted) and turn ON light bulb (Z)

In this example, $Z \equiv A$.

Switch Logic

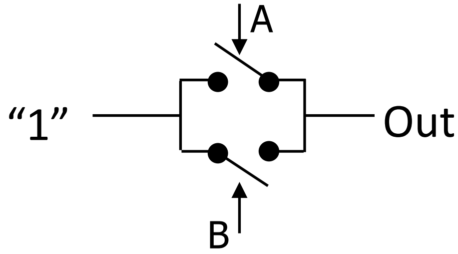
- Can compose switches into more complex ones (Boolean functions)
  - Arrows show action upon assertion (1 = close)

AND:



Z ≡ A and B

OR:



Z ≡ A or B

Historical Note

- Early computer designers built ad hoc circuits from switches
- Began to notice common patterns in their work: ANDs, ORs,
- Master's thesis (by Claude Shannon, 1940) made link between work and 19th Century Mathematician George Boole
- Called it "Boolean" in his honor
- Could apply math to give theory to hardware design, minimization…

Computers need switches

To create an electric computer:

- We need the ability to control switches with electricity
- Switches controlling groups of other switches
  - So electricity should flow through the switch

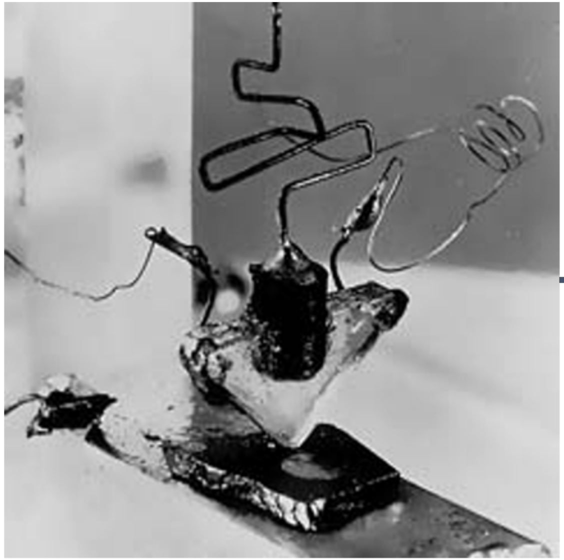
Transistors are hardware switches!

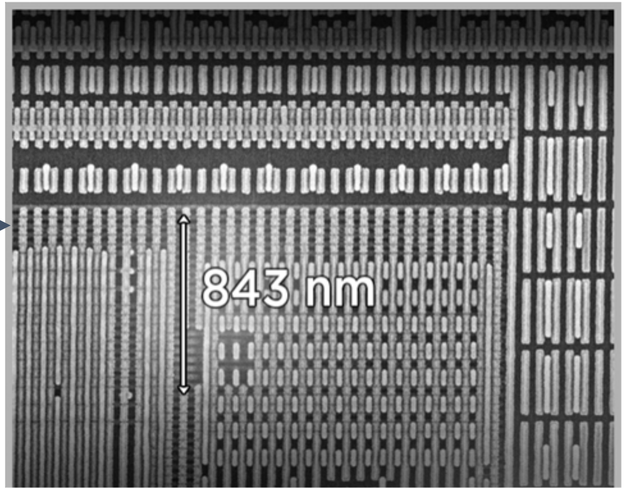
- Modern digital systems designed in CMOS
  - MOS: Metal-Oxide on Semiconductor
  - C for complementary: normally-open and normally-closed switches
- MOS transistors act as voltage-controlled switches

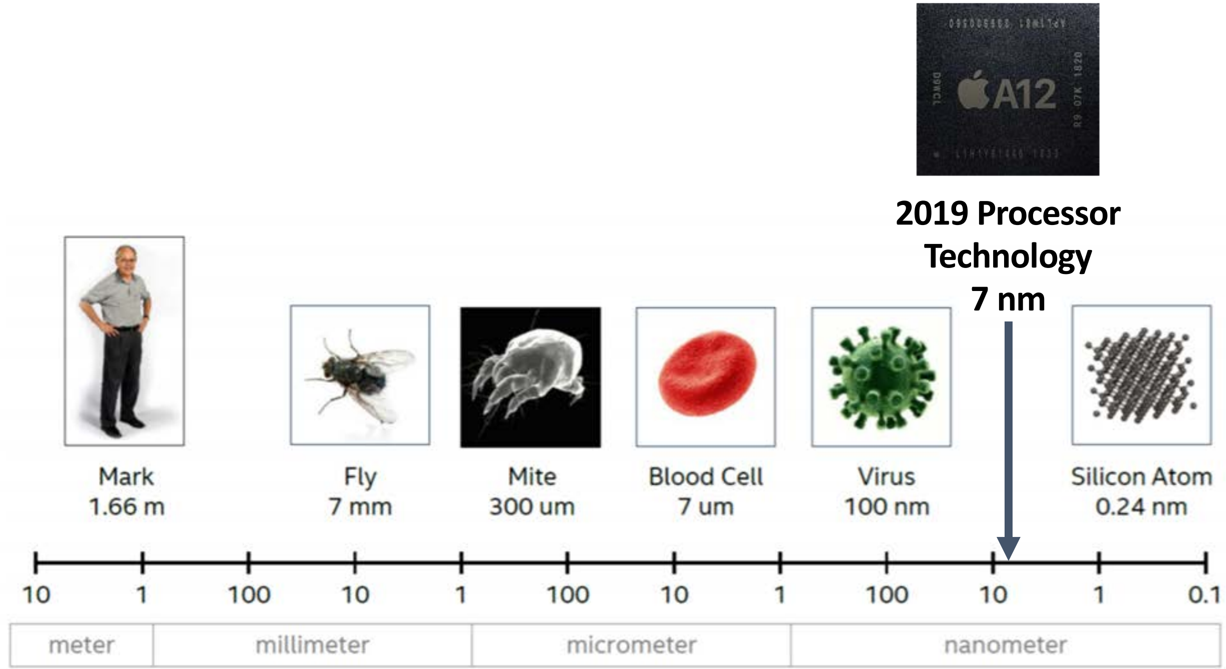
Transistors and CS61C

- The internals of transistors are important, but won't be covered in this class
  - Physical limitations relating to speed and power consumption
  - Can take EE16A/B, EE105, and EE140
- We will proceed with the abstraction of *Digital Logic* (0/1)
- It's also important to understand hardware **trends**

1947 → Today





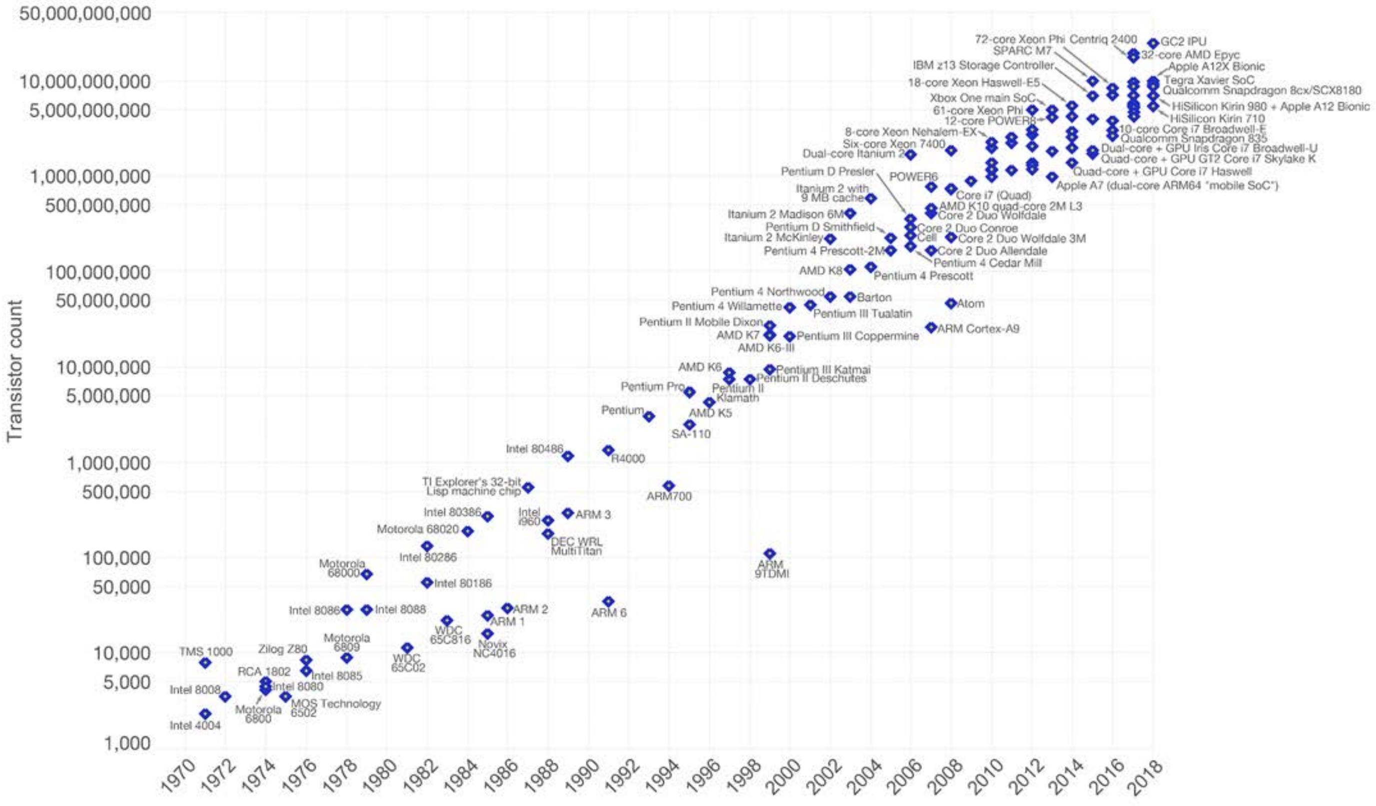


Source: Mark Bohr, IDF14

Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

Our World in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.



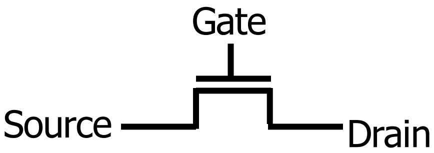
Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

Agenda

- CALL Review
- Hardware Design Overview
- Switches and Transistors
- **CMOS Networks**
- Combinational Logic
  - Combinational Logic Gates
  - Truth Tables
  - Boolean Algebra
  - Circuit Simplification

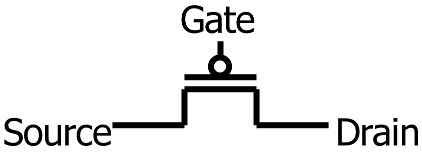
Basics

N-channel



$V_{GS} > V_{TH}$ Switch CLOSED
$V_{GS} < V_{TH}$ Switch OPEN

P-channel

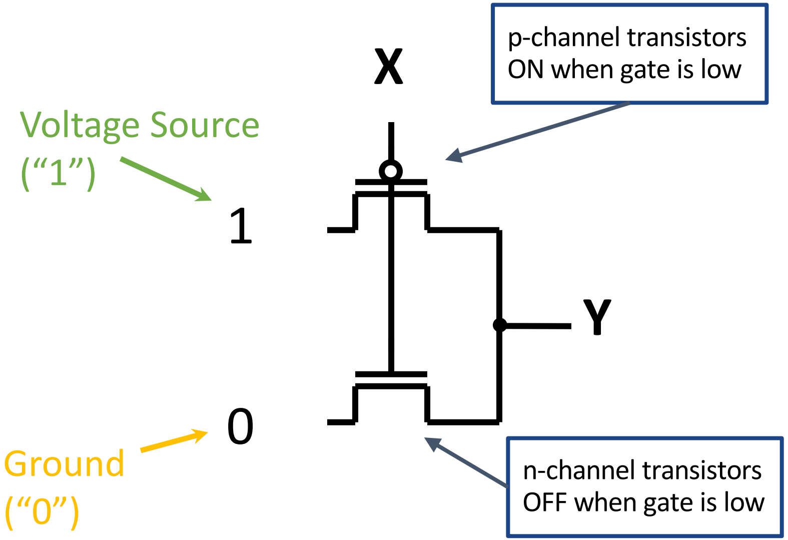


$V_{SG} > V_{TH}$ Switch CLOSED
$V_{SG} < V_{TH}$ Switch OPEN

- Three terminals: **Source**, **Gate**, and **Drain**
- Switch action: if voltage on gate terminal is (some amount) higher/lower than source terminal then conducting path established between drain and source terminals
- $V_{GS} = V_{Gate} - V_{Source}$; $V_{SG} = V_{Source} - V_{Gate}$; $V_{TH} = V_{Threshold}$

MOS Networks

What is the relationship between X and Y?

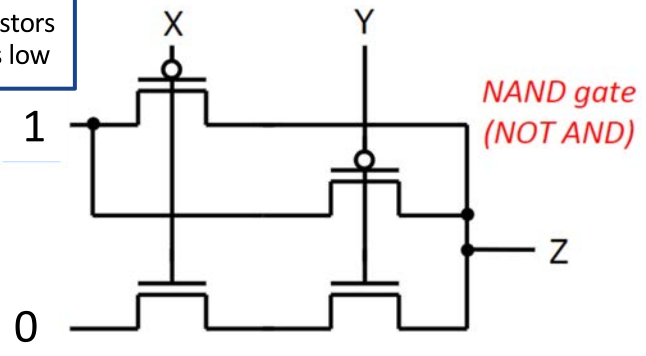


X	Y
0	1
1	0

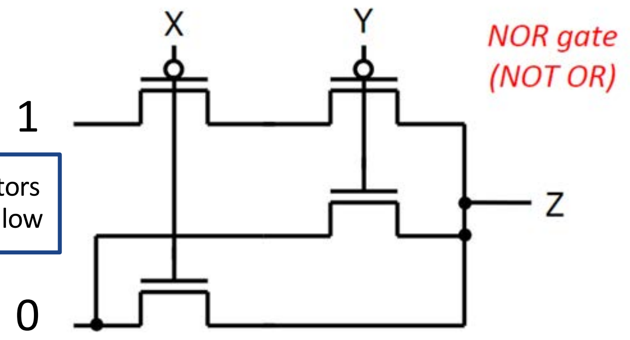
Called an inverter or NOT gate

Two Input Networks

p-channel transistors
ON when gate is low



X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

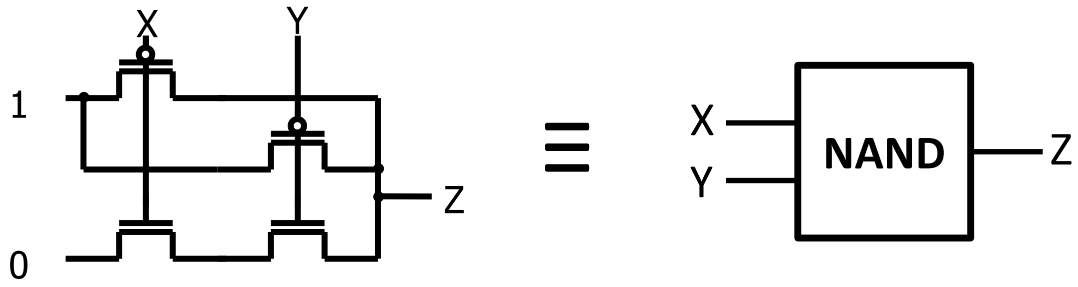


n-channel transistors
OFF when gate is low

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

Abstraction: Block Diagrams

- In reality, chips composed of just transistors and wires
  - Small groups of transistors form useful building blocks, which we show as *blocks*



- Can combine to build higher-level blocks
  - You can build AND, OR, and NOT out of NAND!

Agenda

- CALL Review
- Hardware Design Overview
- Switches and Transistors
- CMOS Networks
- **Combinational Logic**
  - **Combinational Logic Gates**
  - Truth Tables
  - Boolean Algebra
  - Circuit Simplification

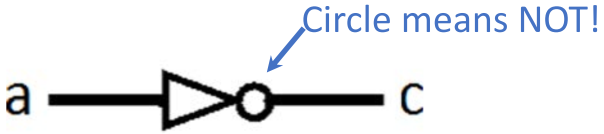
Type of Circuits

- *Digital Systems* consist of two basic types of circuits:
  - Combinational Logic (CL)
    - Output is a function of the inputs only, not the history of its execution
    - e.g. circuits to add A, B (ALUs)
  - Sequential Logic (SL)
    - Circuits that “remember” or store information
    - a.k.a. “State Elements”
    - e.g. memory and registers (Registers)

Logic Gates (1/2)

- Special names and symbols:

NOT

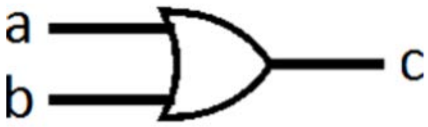


a	c
0	1
1	0

AND

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

OR



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

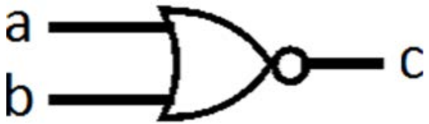
Logic Gates (2/2)

Inverted versions are easier to implement in CMOS

NAND

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

NOR

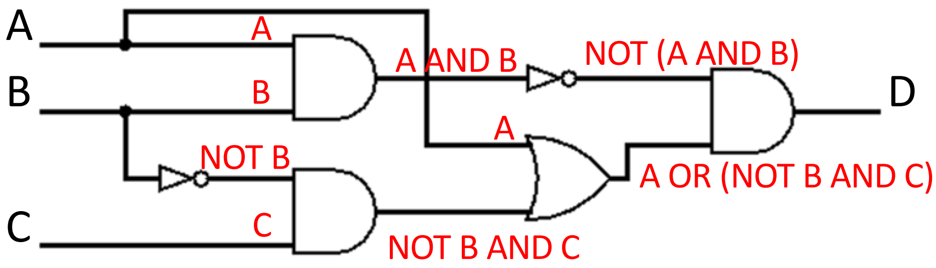


a	b	c
0	0	1
0	1	0
1	0	0
1	1	0

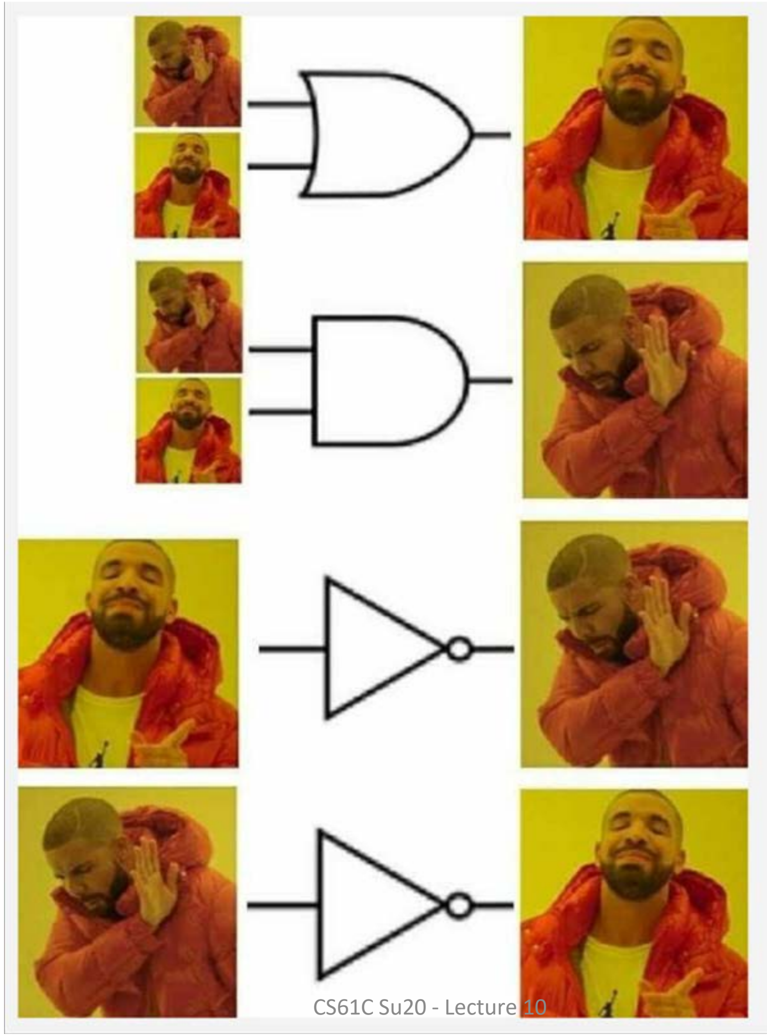
XOR

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

Combining Multiple Logic Gates



(NOT(**A** AND **B**)) AND (**A** OR (NOT **B** AND **C**))



Agenda

- CALL Review
- Hardware Design Overview
- Switches and Transistors
- CMOS Networks
- **Combinational Logic**
  - Combinational Logic Gates
  - **Truth Tables**
  - Boolean Algebra
  - Circuit Simplification

Representations of Combinational Logic

✓ Text Description

✓ Circuit Diagram

- Transistors and wires
- Logic Gates

✓ Truth Table

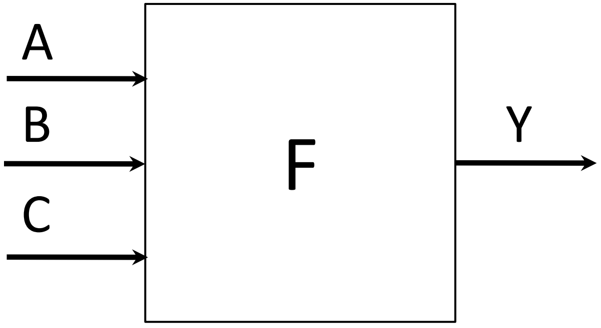
✓ Boolean Expression

✓ *All are equivalent*

Truth Tables

- Table that relates the inputs to a combinational logic circuit to its output
  - Output *only* depends on current inputs
  - Use abstraction of 0/1 instead of high/low V
  - Shows output for *every* possible combination of inputs
- How big?
  - 0 or 1 for each of N inputs, so 2^N rows

General Form



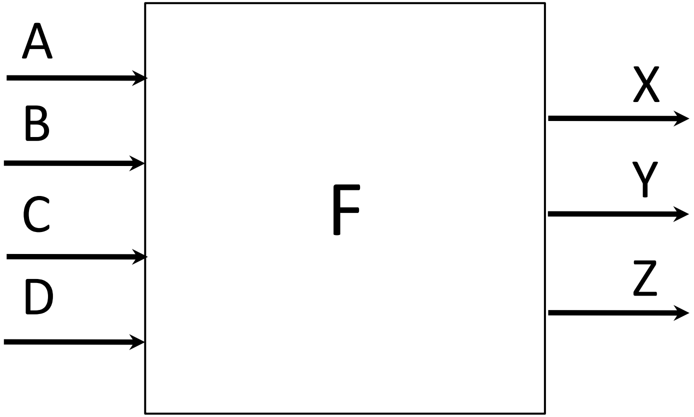
A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

R
Rows

If N inputs, how many distinct functions F do we have?

Function maps each row to 0 or 1, so 2^R possible functions

Multiple Outputs



- For 3 outputs, just three indep. functions: X(A,B,C,D), Y(A,B,C,D), and Z(A,B,C,D)
  - Can show functions in separate columns without adding any rows!

Question: Convert the following statements into a Truth Table for: (x XOR y) OR (NOT z)

X	Y	Z	(A)	(B)	(C)	(D)
0	0	0	1	1	1	1
0	0	1	0	0	0	0
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	0	1	1	1	0	1
1	1	0	1	1	1	0
1	1	1	1	0	1	1

Question: Convert the following statements into a Truth Table for: (x XOR y) OR (NOT z)

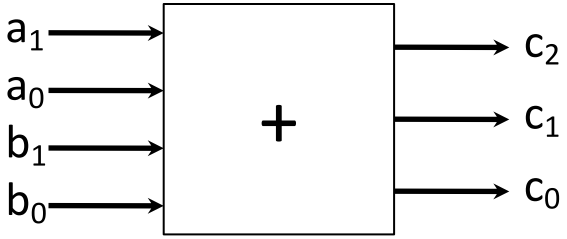
X	Y	Z	(A)	(B)	(C)	(D)
0	0	0	1	1	1	1
0	0	1	0	0	0	0
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	0	1	1	1	0	1
1	1	0	1	1	1	0
1	1	1	1	0	1	1

More Complicated Truth Tables

3-Input Majority

a	b	c	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

2-bit Adder



A		B		C		
a_1	a_0	b_1	b_0	c_2	c_1	c_0
			.		How many rows?	
			.			
			.			

Agenda

- CALL Review
- Hardware Design Overview
- Switches and Transistors
- CMOS Networks
- **Combinational Logic**
  - Combinational Logic Gates
  - Truth Tables
  - **Boolean Algebra**
  - Circuit Simplification

My Hand Hurts…

- Truth tables are huge
  - Write out EVERY combination of inputs and outputs (thorough, but inefficient)
  - Finding a particular combination of inputs involves scanning a large portion of the table
- There must be a shorter way to represent combinational logic
  - Boolean Algebra to the rescue!

a	b	c
00..0	00..0	00..0
00..0	00..1	00..1
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
11..1	11..1	11..1

Boolean Algebra

- Represent inputs and outputs as variables
  - Each variable can only take on the value 0 or 1
- Overbar or ¬ is NOT: "logical complement"
  - e.g. if A is 0, $\overline{A}$ is 1. If A is 1, then ¬A is 0
- Plus (+) is 2-input OR: "logical sum"
- Product (·) is 2-input AND: "logical product"
  - All other gates and logical expressions can be built from combinations of these
    ¬AB + **A¬B** == (NOT(**A** AND **B**)) OR (**A** AND NOT **B**)

For slides, will use ¬A

Truth Table to Boolean Expression

- Read off of table
  - For 1, write variable name
  - For 0, write complement of variable
- *Sum of Products (SoP)*
  - Take rows with 1's in output column, sum products of inputs
  - c= ¬a b + a ¬b
- *Product of Sums (PoS)*
  - Take rows with 0's in output column, product the sum of the *complements of the inputs*
  - c = (a + b) · (¬a + ¬b)

= ¬(¬a ¬b + a b)

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

We can show that these are equivalent!

Laws of Boolean Algebra

These laws allow us to perform simplification:

$x \cdot \overline{x} = 0$	$x + \overline{x} = 1$	complementarity
$x \cdot 0 = 0$	$x + 1 = 1$	laws of 0's and 1's
$x \cdot 1 = x$	$x + 0 = x$	identities
$x \cdot x = x$	$x + x = x$	idempotent law
$x \cdot y = y \cdot x$	$x + y = y + x$	commutativity
$(xy)z = x(yz)$	$(x + y) + z = x + (y + z)$	associativity
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	distribution
$xy + x = x$	$(x + y)x = x$	uniting theorem
$\overline{x}y + x = x + y$	$(\overline{x} + y)x = xy$	uniting theorem v.2
$\overline{x \cdot y} = \overline{x} + \overline{y}$	$\overline{x + y} = \overline{x} \cdot \overline{y}$	DeMorgan's Law

Agenda

- CALL Review
- Hardware Design Overview
- Switches and Transistors
- CMOS Networks
- **Combinational Logic**
  - Combinational Logic Gates
  - Truth Tables
  - Boolean Algebra
  - **Circuit Simplification**

Manipulating Boolean Algebra

- SoP and PoS expressions can still be long
  - We wanted to have shorter representation than a truth table!
- Boolean algebra follows a set of rules that allow for simplification
  - Goal will be to arrive at the simplest equivalent expression
  - Allows us to build simpler (and faster) hardware

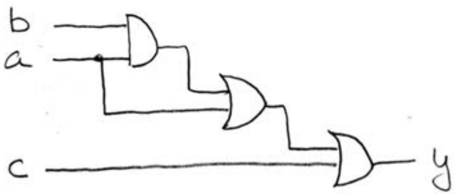
Faster Hardware?

- **Recall:** Everything we are dealing with is just an abstraction of transistors and wires
  - Inputs propagating to the outputs are voltage signals passing through transistor networks
  - There is always some *delay* before a CL's output updates to reflect the inputs
- Simpler Boolean expressions ↔ smaller transistor networks ↔ smaller circuit delays ↔ faster hardware

Boolean Algebraic Simplification Example

$$y = ab + a + c$$

Circuit Simplification



1) original circuit (Transistors and/or Gates)

$$\downarrow$$

$$y = ((ab) + a) + c$$

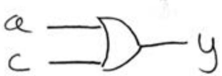
2) equation derived from original circuit

$$\downarrow$$

$$\begin{aligned} &= ab + a + c \\ &= a(b+1) + c \\ &= a(1) + c \\ &= a + c \end{aligned}$$

3) algebraic simplification

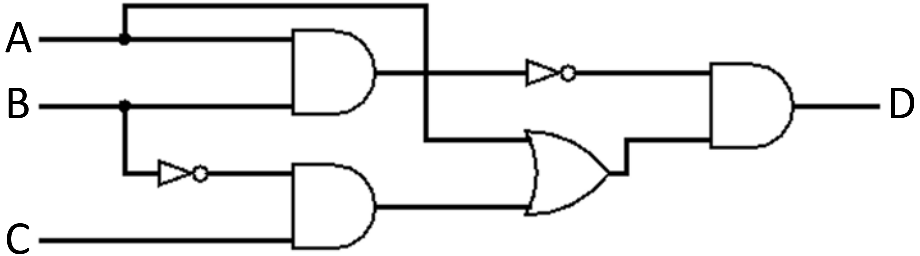
$$\downarrow$$



4) simplified circuit

Circuit Simplification Example (1/4)

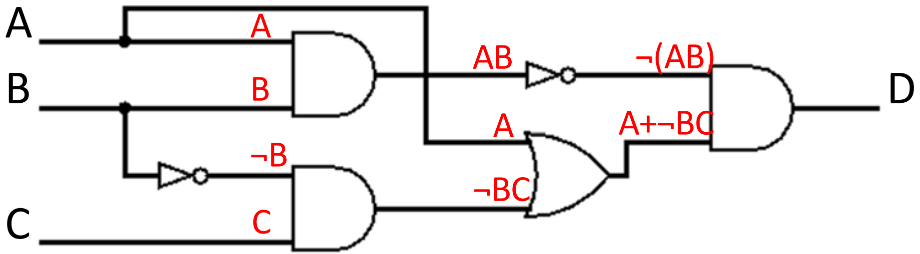
- Simplify the following circuit:



- Options:
  1) Test all combinations of the inputs and build the Truth Table, then use SoP or PoS
  2) Write out expressions for signals based on gates
     - Will show this method here

Circuit Simplification Example (2/4)

- Simplify the following circuit:



- Start from left, propagate signals to the right

Arrive at $D = \neg(AB)(A + \neg BC)$

Circuit Simplification Example (3/4)

- Simplify Expression:

$D = \neg(AB)(A + \neg BC)$	
$= (\neg A + \neg B)(A + \neg BC)$	DeMorgan's
$= \neg AA + \neg A\neg BC + \neg BA + \neg B\neg BC$	Distribution
$= 0 + \neg A\neg BC + \neg BA + \neg B\neg BC$	Complementarity
$= \neg A\neg BC + \neg BA + \neg BC$	Idempotent Law
$= (\neg A + 1)\neg BC + \neg BA$	Distribution
$= \neg BC + \neg BA$	Law of 1's
$= \neg B(A + C)$	Distribution

Which of these is "simpler"?

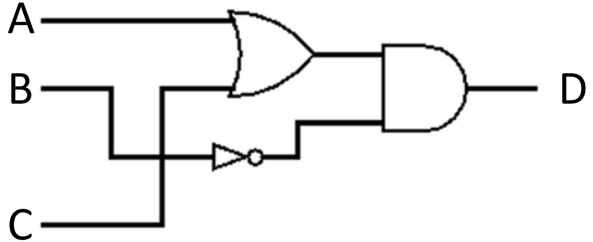
Circuit Simplification Example (4/4)

- Draw out final circuit:
  - D = ¬BC + ¬BA = ¬B(A + C)

5 3

How many gates do we need for each?

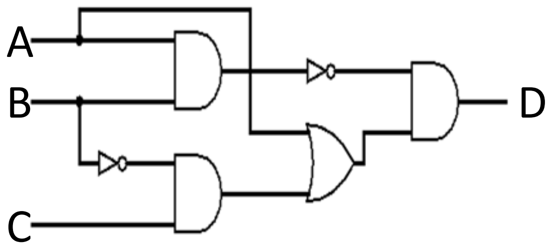
- Simplified Circuit:



  - Reduction from 6 gates to 3!
  - Beore: ¬(AB)(A + ¬BC)

Summary

- Beginnings of hardware design and layered abstractions
- Transistors -> CMOS Networks -> Combinational Logic



✓ Text Description
✓ Circuit Diagram
  – Transistors and wires
  – Logic Gates
✓ Truth Table
✓ Boolean Expression

✓ All are equivalent

Converting Combinational Logic