UNIVERSITY of WASHINGTON

Digital Logic - Combinational

UNIVERSITY of WASHINGTON

Agenda

- Combinational Logic
  - Combinational Logic Gates
  - Truth Tables
  - Boolean Algebra
  - Circuit Simplification

UNIVERSITY of WASHINGTON

Roadmap

C:

```
car *c = malloc(sizeof(car));
c->miles = 100;
c->gals = 17;
float mpg = get_mpg(c);
free(c);
```

Java:

```
Car c = new Car();
c.setMiles(100);
c.setGals(17);
float mpg =
    c.getMPG();
```

Assembly language:

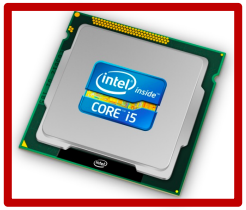
```
get_mpg(car*):
        lw      a5,0(a0)
        lw      a4,4(a0)
        divw    a5,a5,a4
        fcvt.s.w        fa0,a5
        ret
```

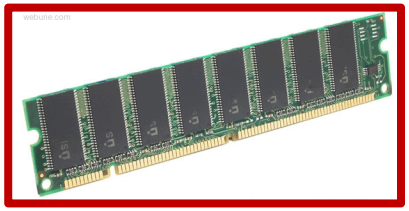
Machine code:

```
01110100000110000
10001101000001000000010
1000100111000010
11000001111110100011111
```

OS:

Computer system:





Memory & data
Arrays & structs
Integers & floats
RISC V assembly
Procedures & stacks
Executables
Memory & caches
Processor Pipeline
Performance
Parallelism

UNIVERSITY of WASHINGTON

Why is this important?

- Why study hardware design?
  - Background for other more in-depth classes
  - Only course in CMPT that deals with hardware
  - We need some digital systems knowledge to build our own processor

UNIVERSITY of WASHINGTON

Why is this important?

- We need some digital systems knowledge to build our own processor

- Why build our own processor?
  - Because we believe computer scientists should have an answer to the question “How does a computer work?”
  - So you can understand how code is actually executed on a computer

UNIVERSITY of WASHINGTON

Why is this important?

- So you can understand how code is actually executed on a computer

- Why do we care about how code executes?
  - Reliability
  - Performance
  - Security

UNIVERSITY of WASHINGTON

Hardware Design

Why study hardware design and processors and computers if you only care about high-level software?

Example of the power of layered abstractions

- Transistors -> Combinational Logic
- Combinational Logic -> Sequential Logic
- Sequential Logic -> Processors
- Processors -> Machine Language
- Machine Language -> Assembly
- Assembly -> High-level Programming Languages
- High-level Programming Languages -> Word,Minecraft,Twitter

At each step we can “abstract away” the lower steps and (mostly) forget they exist

UNIVERSITY of WASHINGTON

Synchronous Digital Systems (SDS)

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

Synchronous:

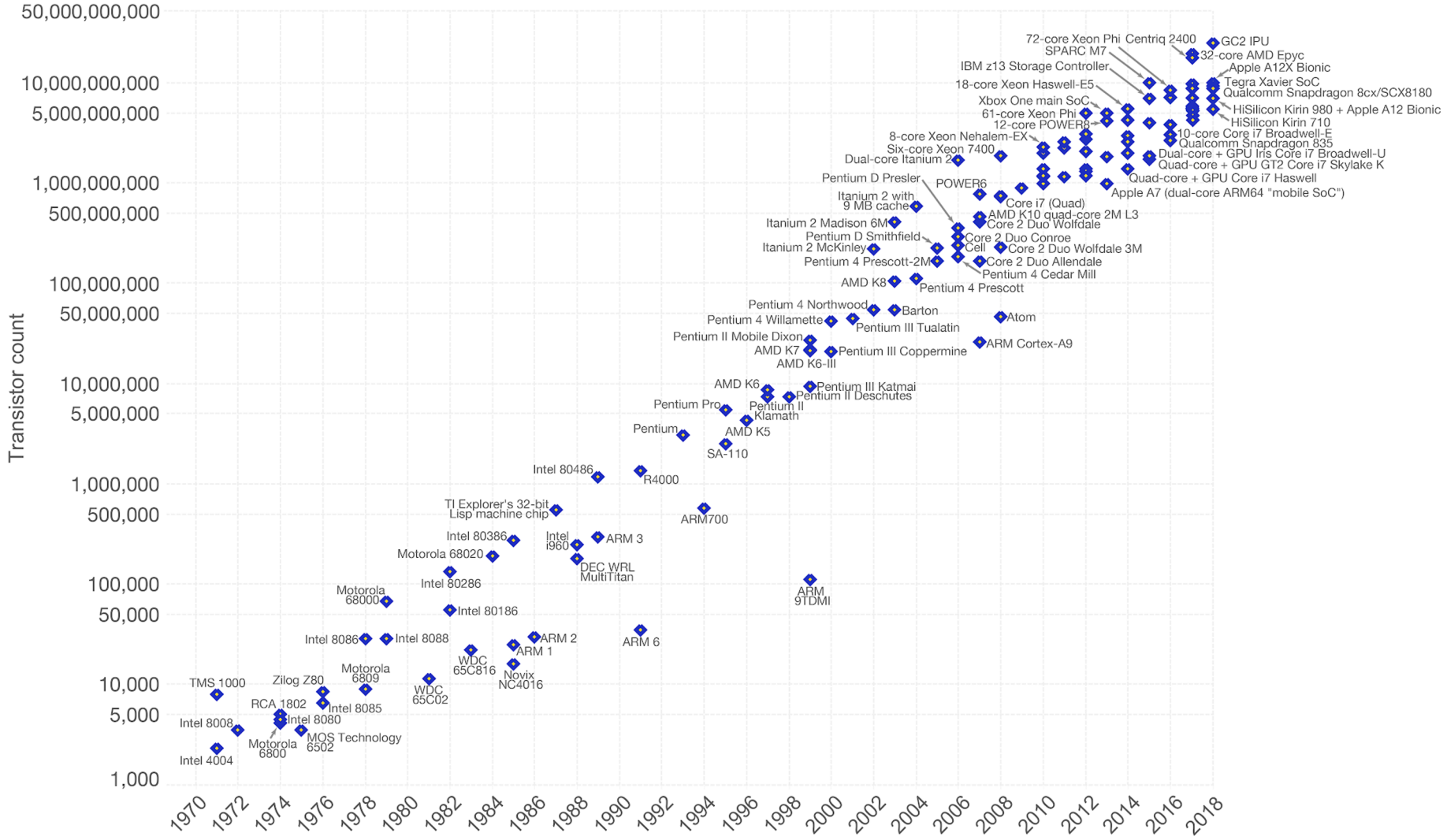
- All operations coordinated by a central clock
  - "Heartbeat" of the system! (processor frequency)

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

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

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.

Licensed under CC-BY-SA by the author Max Roser.

UNIVERSITY of WASHINGTON

Historical Trend

Trend:

- Hardware gets more powerful every year.
  - (Really due to hard work of thousands of engineers)

Therefore:

- Software gets more resources and faster compute!
  - (And has to keep up with ever-changing hardware)

UNIVERSITY of WASHINGTON

Agenda

- **Combinational Logic**
  - **Combinational Logic Gates**
  - Truth Tables
  - Boolean Algebra
  - Circuit Simplification

UNIVERSITY of WASHINGTON

Type of Circuits

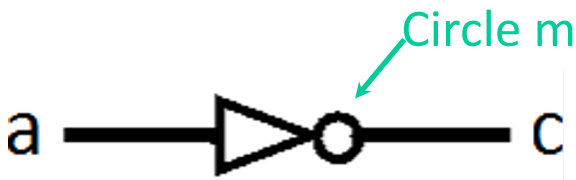
- *Digital Systems* consist of two basic types of circuits:
  - Combinational Logic (CL)
    - Output is a function of the inputs only, not the history of its execution
    - e.g. circuits to add A, B (ALUs)
  - Sequential Logic (SL)
    - Circuits that "remember" or store information
    - a.k.a. "State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

UNIVERSITY of WASHINGTON

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

UNIVERSITY of WASHINGTON

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

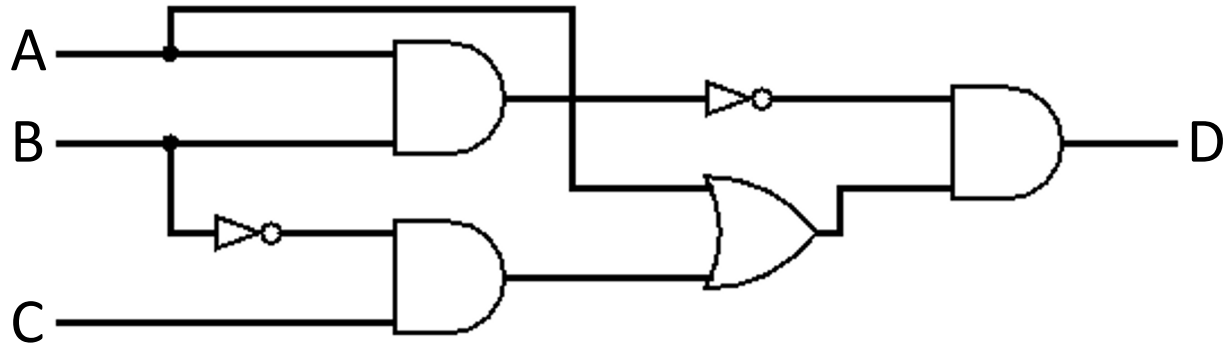
a	b	c
0	0	1
0	1	0
1	0	0
1	1	0

XOR

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

UNIVERSITY of WASHINGTON

Combining Multiple Logic Gates



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

UNIVERSITY of WASHINGTON

Agenda

- **Combinational Logic**
  - Combinational Logic Gates
  - **Truth Tables**
  - Boolean Algebra
  - Circuit Simplification

UNIVERSITY of WASHINGTON

Representations of Combinational Logic

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

✓ *All are equivalent*

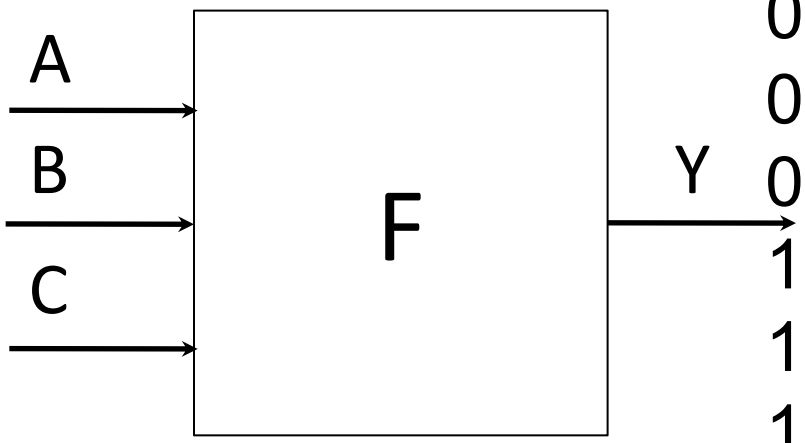
UNIVERSITY of WASHINGTON

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

UNIVERSITY of WASHINGTON

General Form



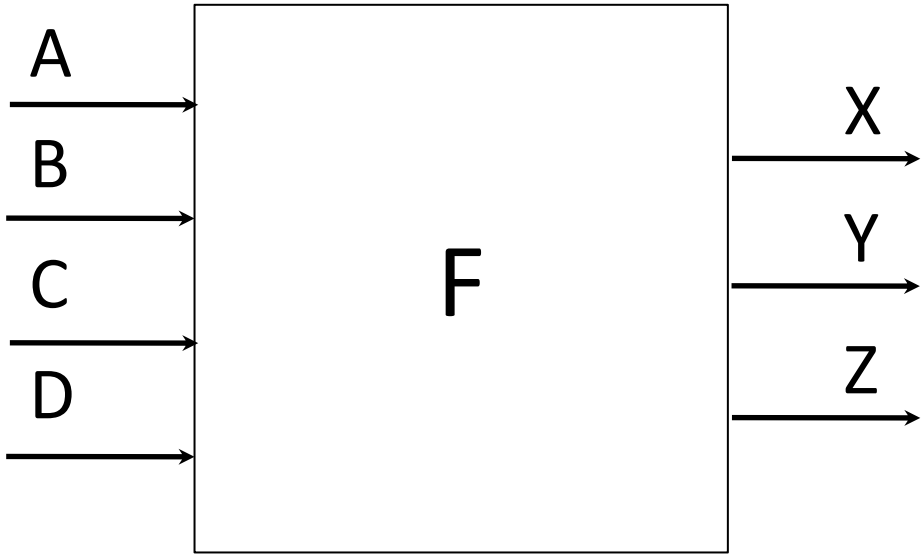
A	B	C		Y
0	0	0	F(0,0,0)	0
0	0	1	F(0,0,1)	1
0	1	0	F(0,1,0)	0
0	1	1	F(0,1,1)	0
1	0	0	F(1,0,0)	0
1	0	1	F(1,0,1)	1
1	1	0	F(1,1,0)	1
1	1	1	F(1,1,1)	0

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

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

UNIVERSITY of WASHINGTON

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!

UNIVERSITY of WASHINGTON

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

UNIVERSITY of WASHINGTON

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

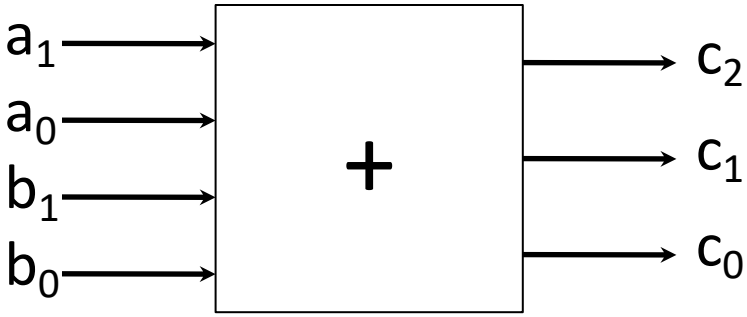
UNIVERSITY of WASHINGTON

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



How many rows?

A		B		C		
a_1	a_0	b_1	b_0	c_2	c_1	c_0
			.			
			.			
			.			

UNIVERSITY of WASHINGTON

Agenda

- **Combinational Logic**
  - Combinational Logic Gates
  - Truth Tables
  - **Boolean Algebra**
  - Circuit Simplification

UNIVERSITY of WASHINGTON

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!

UNIVERSITY of WASHINGTON

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

For slides, will use ¬A

- 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**)

UNIVERSITY of WASHINGTON

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

UNIVERSITY of WASHINGTON

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	c
0	0	0
0	1	1
1	0	1
1	1	0

We can show that these are equivalent!

UNIVERSITY of WASHINGTON

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

UNIVERSITY of WASHINGTON

Agenda

- **Muxes**
- Sequential Logic Timing
- Maximum Clock Frequency
- Finite State Machines
- Functional Units
- Summary

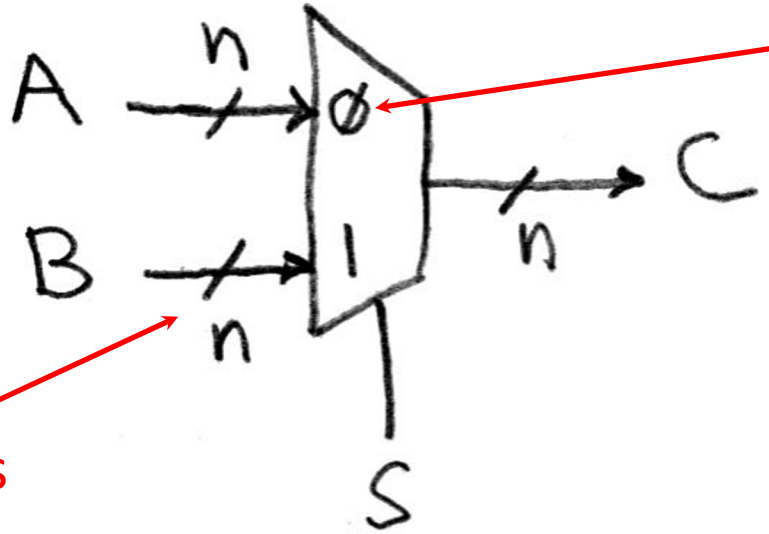
Bonus Slides

- Logisim Intro

UNIVERSITY of WASHINGTON

Data Multiplexor

- Multiplexor ("MUX") is a *selector*
  - Place one of multiple inputs onto output (N-to-1)
- Shown below is an n-bit 2-to-1 MUX
  - Input S selects between two inputs of n bits each



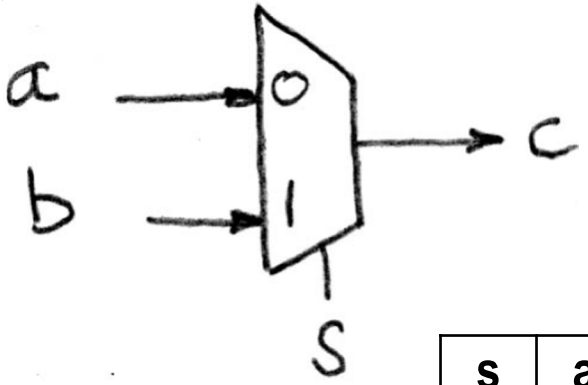
This input is passed to output if selector bits match shown value

Represents that this wire has n bits

UNIVERSITY of WASHINGTON

Implementing a 1-bit 2-to-1 MUX

- **Schematic:**



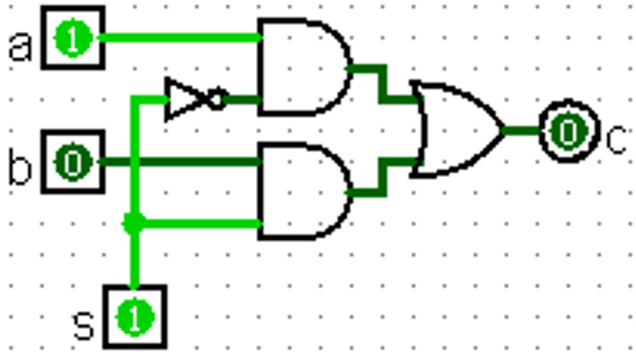
- **Truth Table:**

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

- **Boolean Algebra:**

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

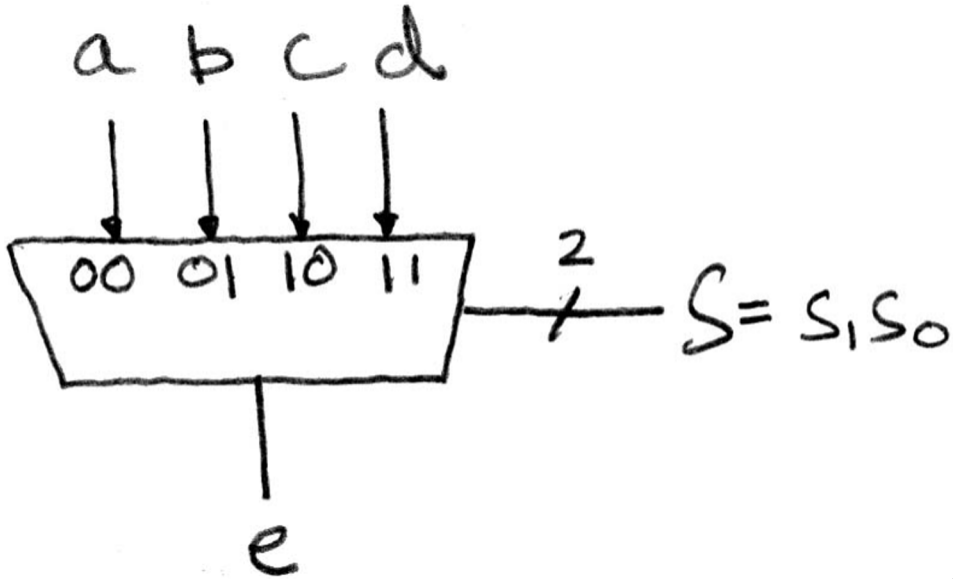
- **Circuit Diagram:**



UNIVERSITY of WASHINGTON

1-bit 4-to-1 MUX (1/2)

- **Schematic:**



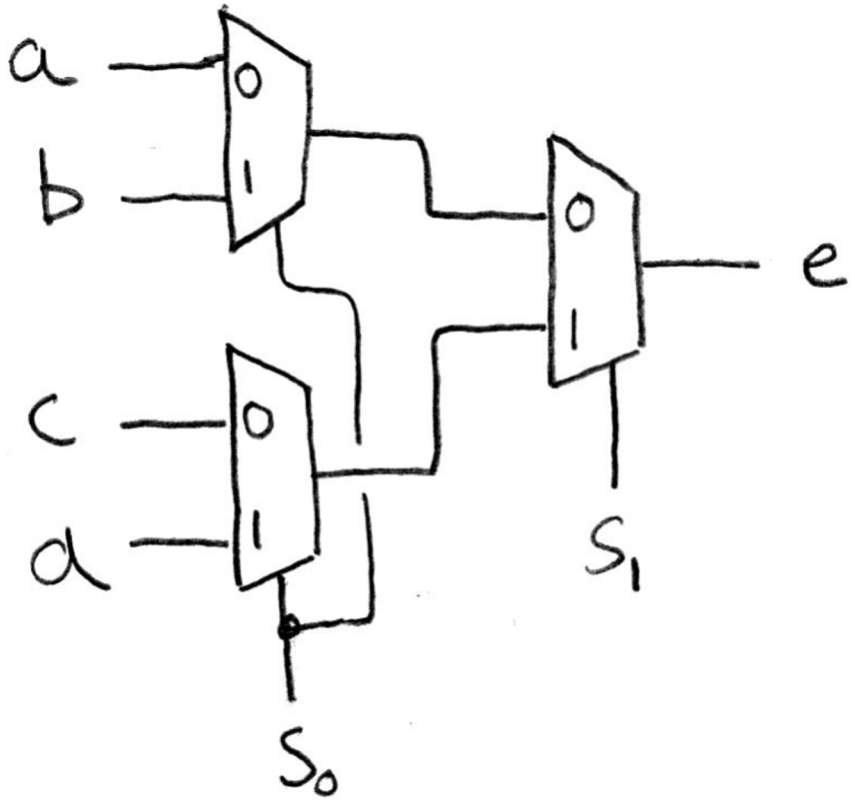
- **Truth Table:** How many rows? 2^6
- **Boolean Expression:**

$$e = \neg s_1 \neg s_0 a + \neg s_1 s_0 b + s_1 \neg s_0 c + s_1 s_0 d$$

UNIVERSITY of WASHINGTON

1-bit 4-to-1 MUX (2/2)

- Can we leverage what we've previously built?
  - Alternative hierarchical approach:



UNIVERSITY of WASHINGTON

Bonus slides (not testable)

UNIVERSITY of WASHINGTON

Agenda

- **Combinational Logic**
  - Combinational Logic Gates
  - Truth Tables
  - Boolean Algebra
  - **Circuit Simplification**

UNIVERSITY of WASHINGTON

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

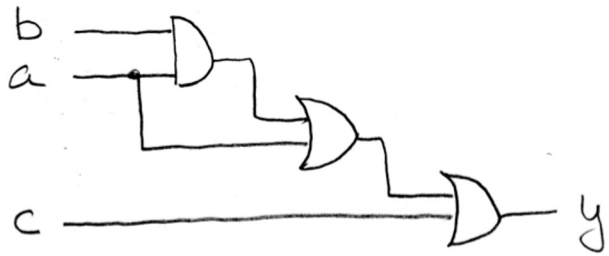
UNIVERSITY of WASHINGTON

Boolean Algebraic Simplification Example

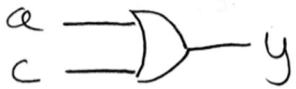
$$y = ab + a + c$$

UNIVERSITY of WASHINGTON

Circuit Simplification



$$y = ((ab) + a) + c$$
$$= ab + a + c$$
$$= a(b + 1) + c$$
$$= a(1) + c$$
$$= a + c$$



1) original circuit (Transistors and/or Gates)

2) equation derived from original circuit

3) algebraic simplification

4) simplified circuit

UNIVERSITY of WASHINGTON

Representations of Combinational Logic

✓ Text Description

✓ Circuit Diagram

- Transistors and wires
- Logic Gates

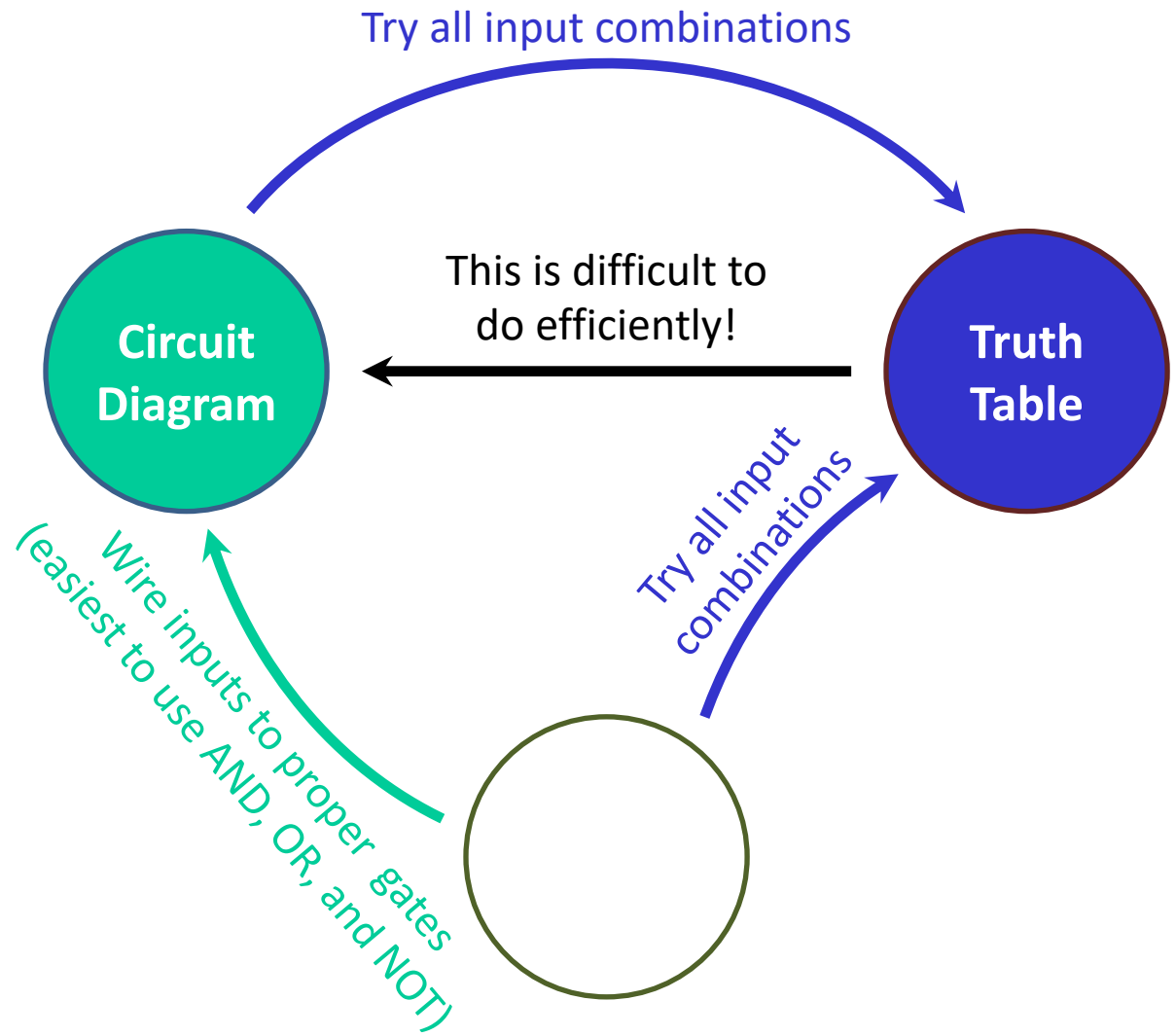
✓ Truth Table

✓ Boolean Expression

✓ *All are equivalent*

UNIVERSITY of WASHINGTON

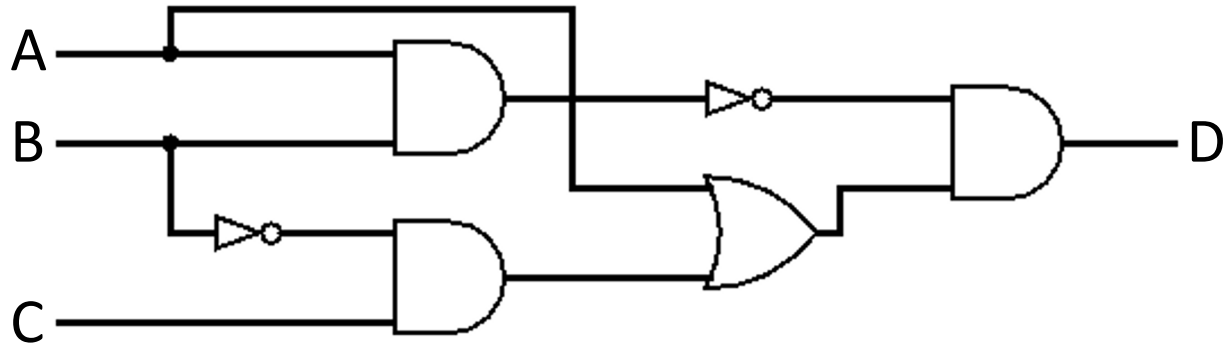
Converting Combinational Logic



UNIVERSITY of WASHINGTON

Circuit Simplification Example (1/4)

- Simplify the following circuit:

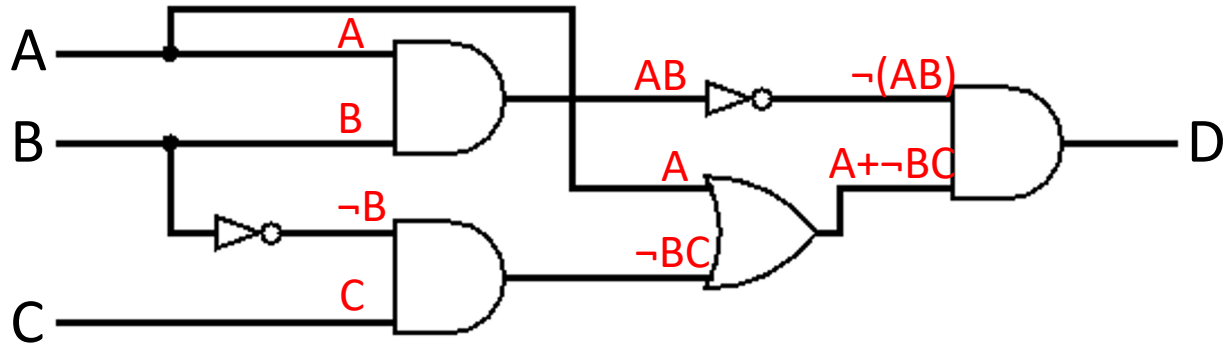


- Options:
  1) Test all combinations of the inputs and build the Truth Table, then use SoP or PoS
  2) Write out expressions for signals based on gates
     - Will show this method here

UNIVERSITY of WASHINGTON

Circuit Simplification Example (2/4)

- Simplify the following circuit:



- Start from left, propagate signals to the right

Arrive at D = ¬(AB)(A + ¬BC)

UNIVERSITY of WASHINGTON

Circuit Simplification Example (3/4)

- Simplify Expression:

D = ¬(AB)(A + ¬BC)

= (¬A + ¬B)(A + ¬BC)

DeMorgan's

= ¬AA + ¬A¬BC + ¬BA + ¬B¬BC

Distribution

= 0 + ¬A¬BC + ¬BA + ¬B¬BC

Complementarity

= ¬A¬BC + ¬BA + ¬BC

Idempotent Law

= (¬A + 1)¬BC + ¬BA

Which of these is "simpler"?

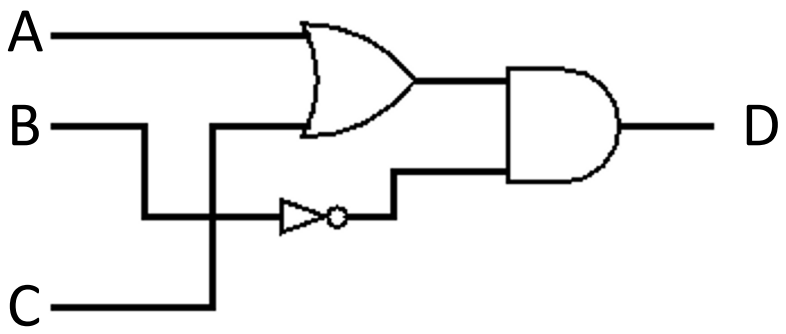
UNIVERSITY of WASHINGTON

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!