

ENGN2219/COMP6719

Computer Systems & Organization

Convener: Shoaib Akram

shoaib.akram@anu.edu.au



Australian
National
University




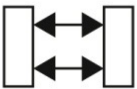
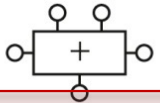

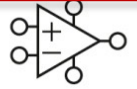


Plan: Week 2

Week 1: Digital abstraction and binary digits




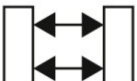
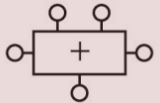

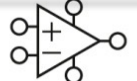


Week 2: Number systems for binary variables, Logic gates

This Week: Boolean logic & Logic gates (contd)

This Week: Combinational logic (more than just gates)

Application Software		Programs
Operating Systems		Device Drivers
Architecture		Instructions Registers
Micro-architecture		Datapaths Controllers
Logic		Adders Memories
Digital Circuits		AND Gates NOT Gates
Analog Circuits		Amplifiers Filters
Devices		Transistors Diodes
Physics		Electrons

We were here

Application Software		Programs
Operating Systems		Device Drivers
Architecture		Instructions Registers
Micro-architecture		Datapaths Controllers
Logic		Adders Memories
Digital Circuits		AND Gates NOT Gates
Analog Circuits		Amplifiers Filters
Devices		Transistors Diodes
Physics		Electrons

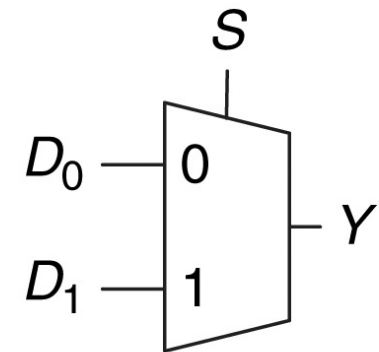
**Broadening our horizon
"one layer at a time"**

2:1 Multiplexer (Mux)

We have seen a 2:1 multiplexer (mux)

- Two data inputs (D_0 and D_1)
- Another input called the **select** signal
- Choose D_0 or D_1 based on the value of the select signal

We will use the high-level schematic for 2:1 mux and ignore the gate-level implementation details

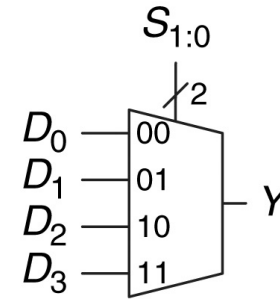


High-level Schematic

Wider (4:1) Multiplexer

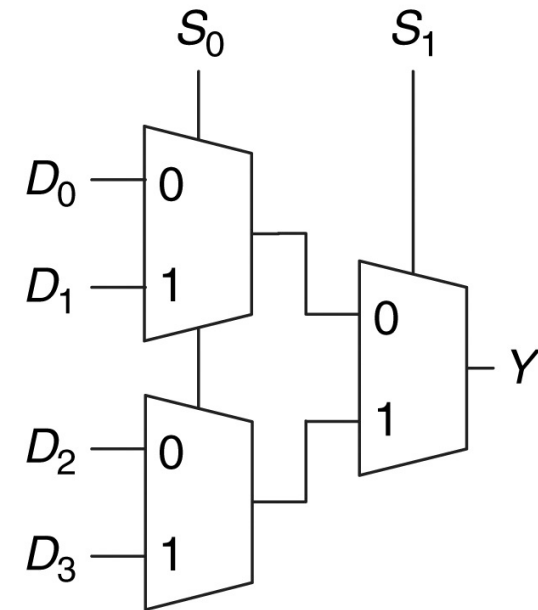
We want to build a 4:1 mux

- How many **select** signals?
 - Call them S_0 and S_1
 - *A / and 2 implies a bus (of width 2) and not a 1-bit wire or input*
- One option is to construct the truth table and derive the Boolean equations. How many rows will there be in the table? (**tedious!**)
- We will use **intuition** to build a 4:1 mux from two 2:1 multiplexers



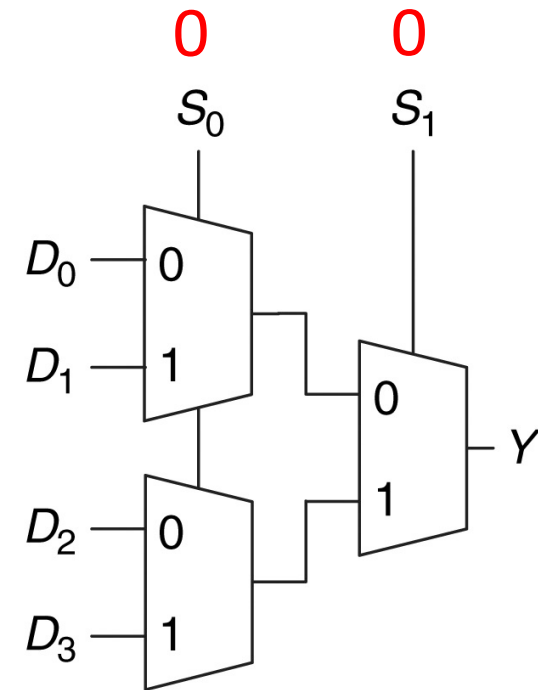
Wider (4:1) Multiplexer

S_0	S_1	Y
0	0	D_0
1	0	D_1
0	1	D_2
1	1	D_3



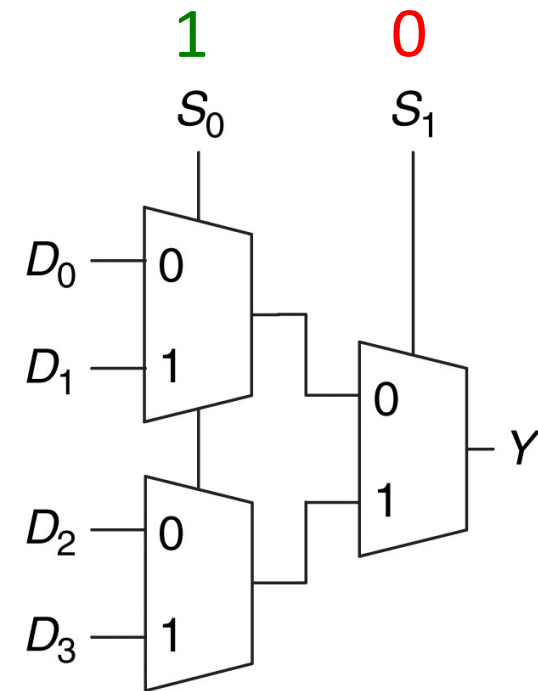
Wider (4:1) Multiplexer

S_0	S_1	Y
0	0	D_0
1	0	D_1
0	1	D_2
1	1	D_3



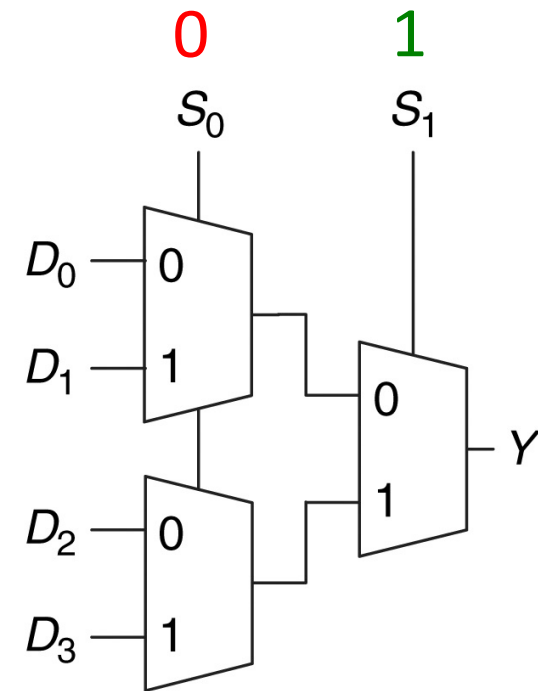
Wider (4:1) Multiplexer

S_0	S_1	Y
0	0	D_0
1	0	D_1
0	1	D_2
1	1	D_3



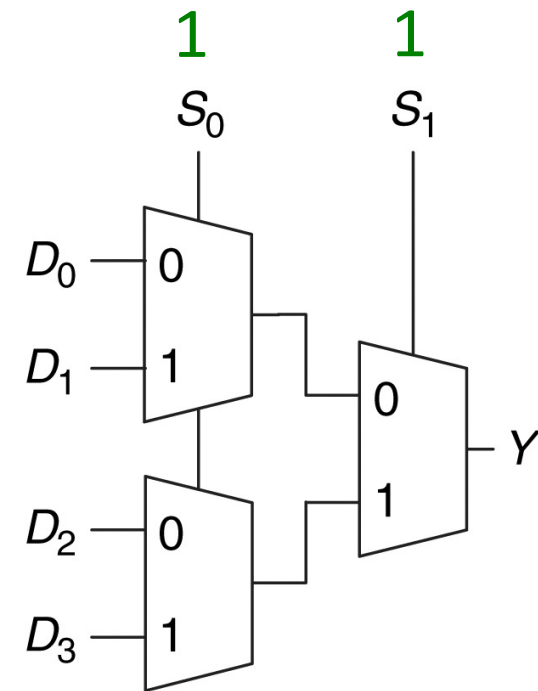
Wider (4:1) Multiplexer

S_0	S_1	Y
0	0	D_0
1	0	D_1
0	1	D_2
1	1	D_3



Wider (4:1) Multiplexer

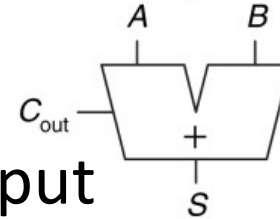
S_0	S_1	Y
0	0	D_0
1	0	D_1
0	1	D_2
1	1	D_3



Ripple Carry Adder

We have seen the half adder

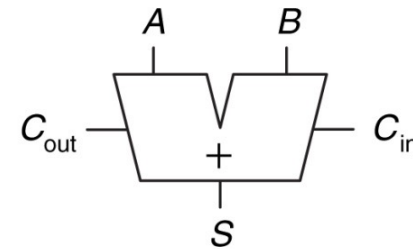
- Limitation of half adder: No carry input
- **Problem:** Adding multiple bits requires the need to add carry out from the previous column to the next column



$$\begin{array}{r} 1 \\ 1001 \\ + 0101 \\ \hline 1110 \end{array}$$

Full adder *so/ves* the problem

- Accepts three inputs including a carry in
- Signals flow from right to left to reflect the carry propagation in arithmetic circuits

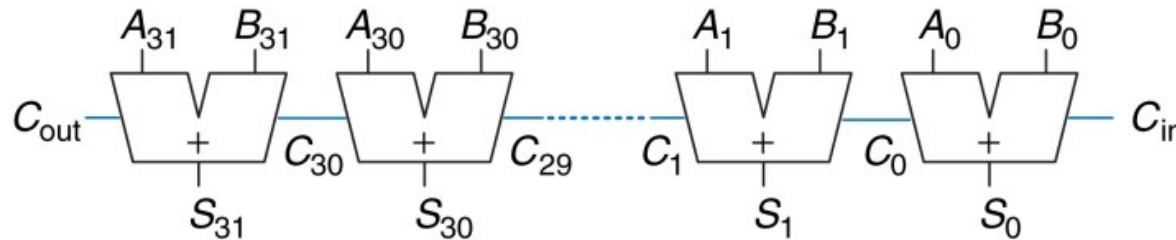


What if we want to add two N-bit numbers?

Ripple Carry Adder

What if we want to add two N-bit numbers?

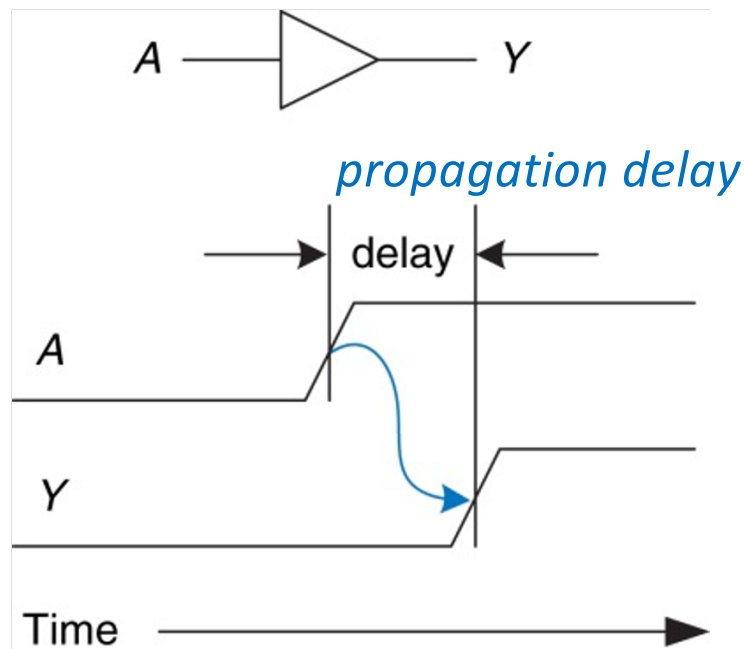
- *Connect a chain of full adders from right to left*



Ripple carry adder has a critical drawback!

Timing

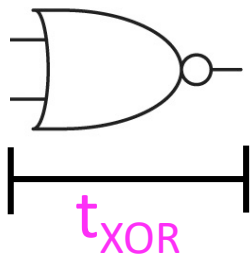
- Every combinational circuit has a *delay (seconds)*
 - *The time it takes for the output to reach a final stable value when the input changes (nanoseconds or picoseconds)*



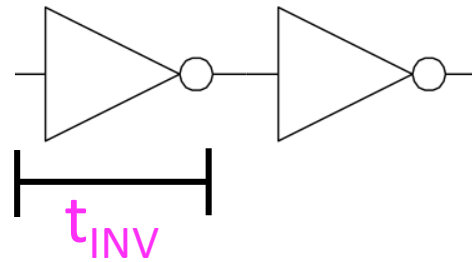
Examples

- **Example:** When the two inputs of the AND gate change from (0,0) to (1,1), how long does it take to reliably measure the output of the AND gate change from 0 to 1?
- **Another example:** When A, B, and C_{in} are supplied to a full adder, how long does it take to observe the final (and stable) S and C_{out} ?

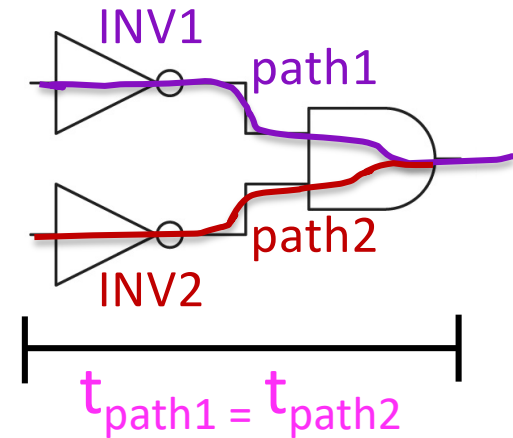
Our Notion of Timing/Delay



Each gate has
a delay



Chain of gates:
Sum the delay of
each gate in the
chain $2 \times t_{\text{INV}}$



Multiple paths from
input to output

$$t_{\text{path1}} = t_{\text{INV1}} + t_{\text{AND}}$$

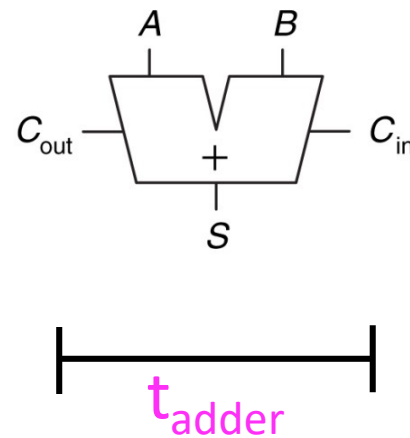
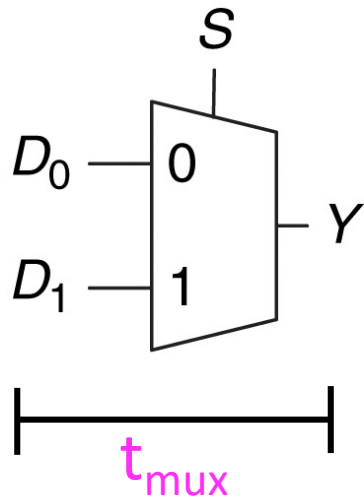
$$t_{\text{path2}} = t_{\text{INV2}} + t_{\text{AND}}$$

Critical and Shortest Path

- Many combinational circuits have multiple paths from input to the output
 - The slowest path (*longest delay*) is called the **critical path**
 - Critical path limits the speed at which the circuit operates
 - In contrast, the shortest path is the fastest
- For simplification, we will ignore the delay of nodes (wires)
 - Although the delay is non-trivial, it is studied best at lower levels of abstraction (e.g., analog circuits)

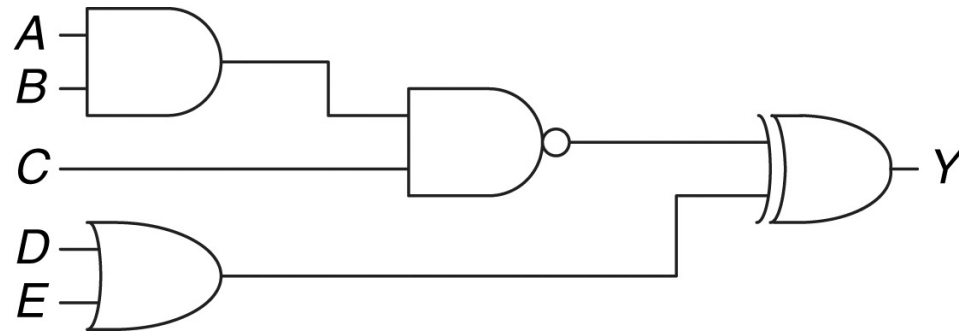
Our Notion of Timing/Delay

We will use a similar notion of delay for combinational circuits such as multiplexers and adders



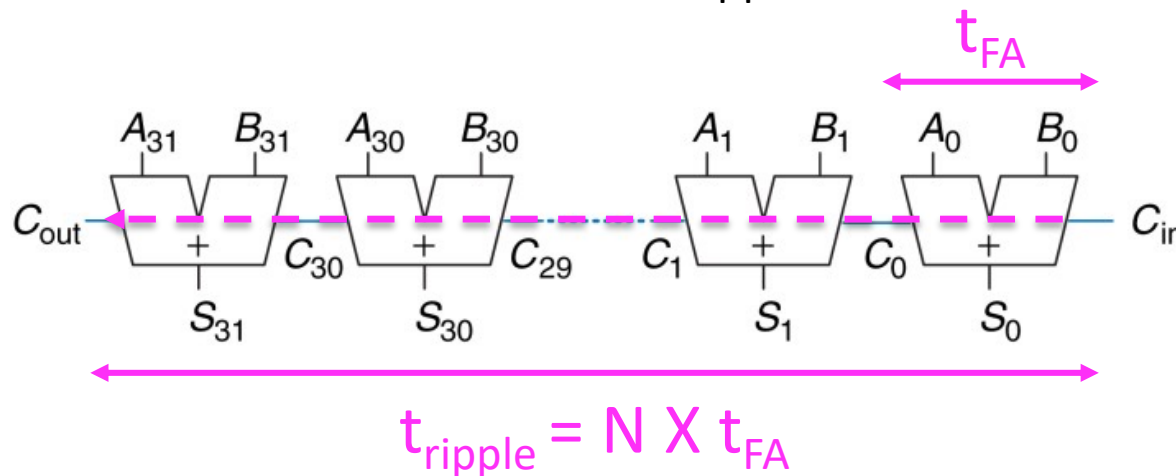
Exercise

Each gate has a propagation delay of 50 picoseconds. Find the shortest path. Find the critical path.



Drawback: Ripple Carry Adder

- If we abstract the delay of full adder as t_{FA} , then what is the delay of the ripple carry adder, t_{ripple} ?



- *The critical path runs through the chain of full adders*
- *Every full adder is on the critical path*
- *The critical path consists of N full adders (slow when N is large)*

Carry-Lookahead Adder

- Another one in the class of *carry propagate adders* that accelerates the computation of carry signals
- Pre-compute (lookahead) carry signals
- Use additional logic (area) to speed up addition
 - Details in Section 5.2.1 (**Optional self-study**)

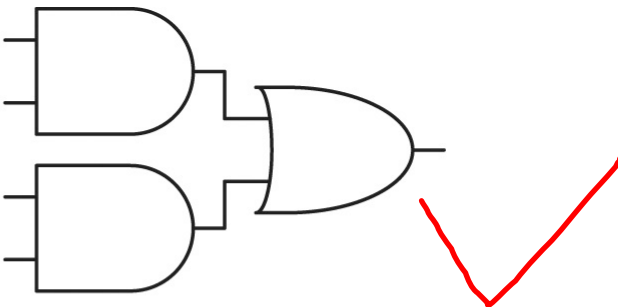
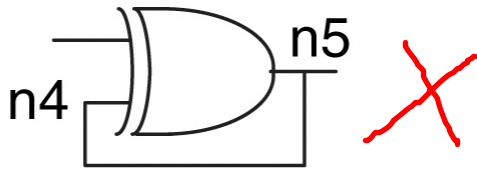
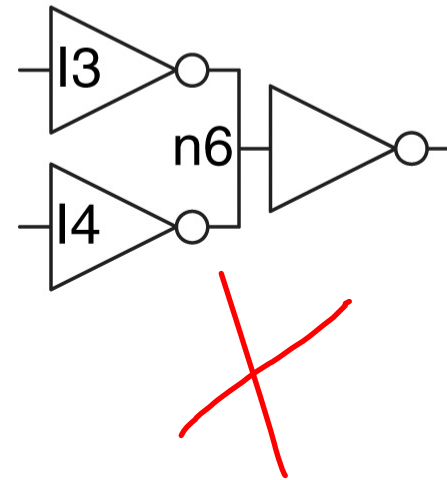
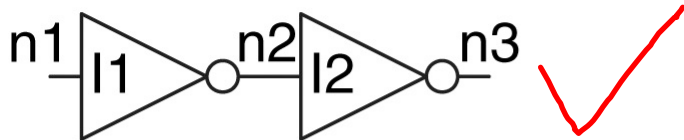
In digital systems, there is very often a tradeoff in performance (speed) and hardware cost (area/power)

Combinational Composition

Rules

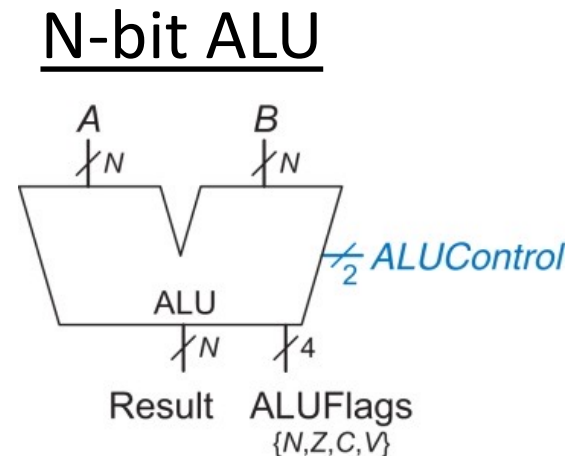
1. Every circuit element is itself combinational
2. Each node is either an input to the circuit or connects to exactly one output terminal of a circuit element
3. The circuit contains no cyclic paths; every path through the circuit visits each circuit node at most once

Which circuits are combinational?



Arithmetic and Logic Unit (ALU)

- The circuits we have looked so far can do one useful thing
 - XNOR gate *performs* equality testing
 - Adder *performs* addition
 - Multiplexer *performs* selection
- ALU is our first *general purpose* circuit
 - Can do a variety of mathematical and logical operations
 - Add, subtract, AND, OR
 - Yet we abstract its inner details as a monolithic unit



ALU Interface/Instructions

- N-bit *data inputs* and *outputs*
- 2-bit *control* input (ALUControl)
 - Pick one of four functions
 - A 2-bit signal specifies the function
 - Think of setting ALUControl to **00**, **01**, **10**, and **11** as giving “*instructions*” to the ALU
- The assignment of binary codes to ALU functions is not arbitrary
 - It is clever (**01** for Subtract in particular) as we will reveal

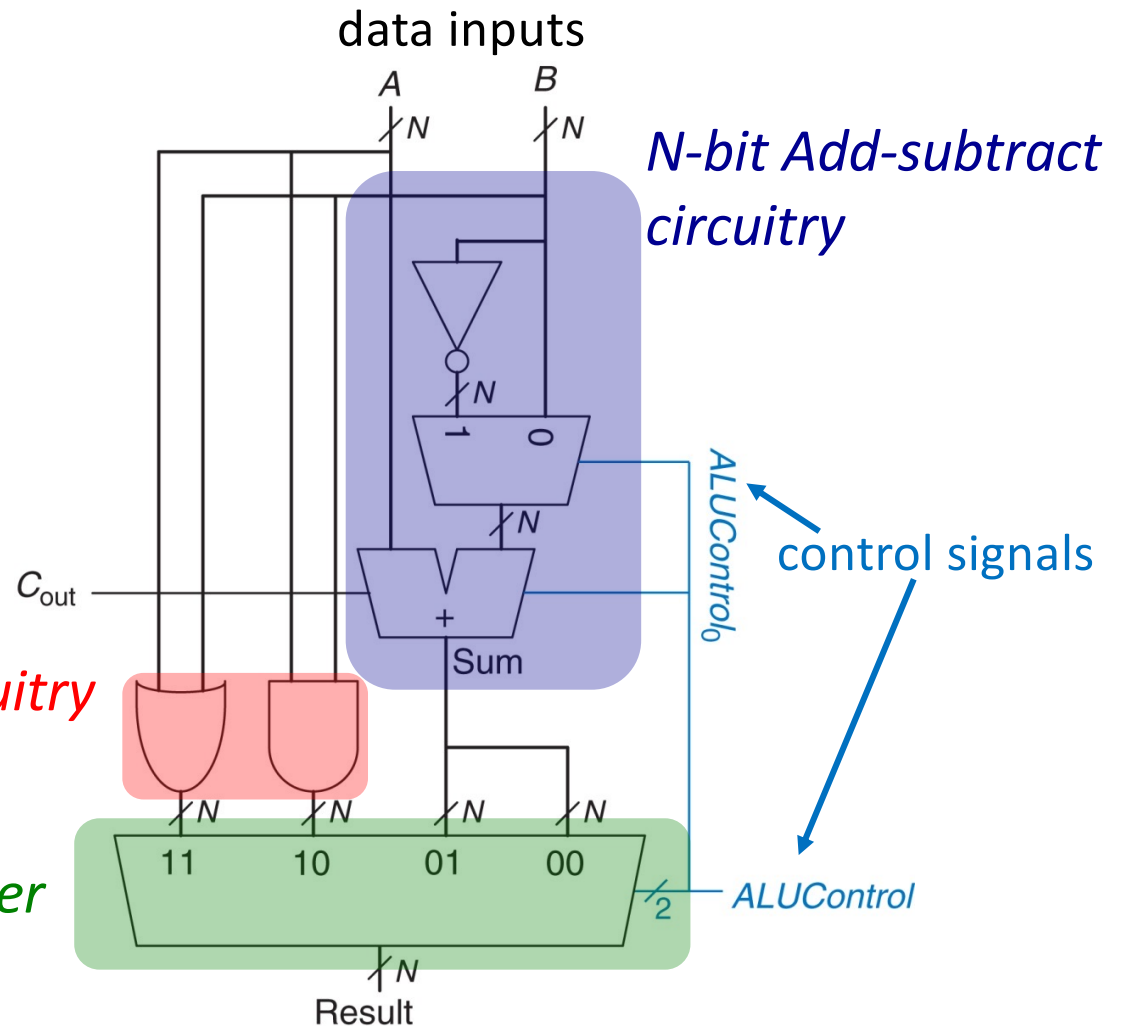
ALUControl _{1:0}	Function
00	Add
01	Subtract
10	AND
11	OR

ALU Implementation

ALUControl _{1:0}	Function
00	Add
01	Subtract
10	AND
11	OR

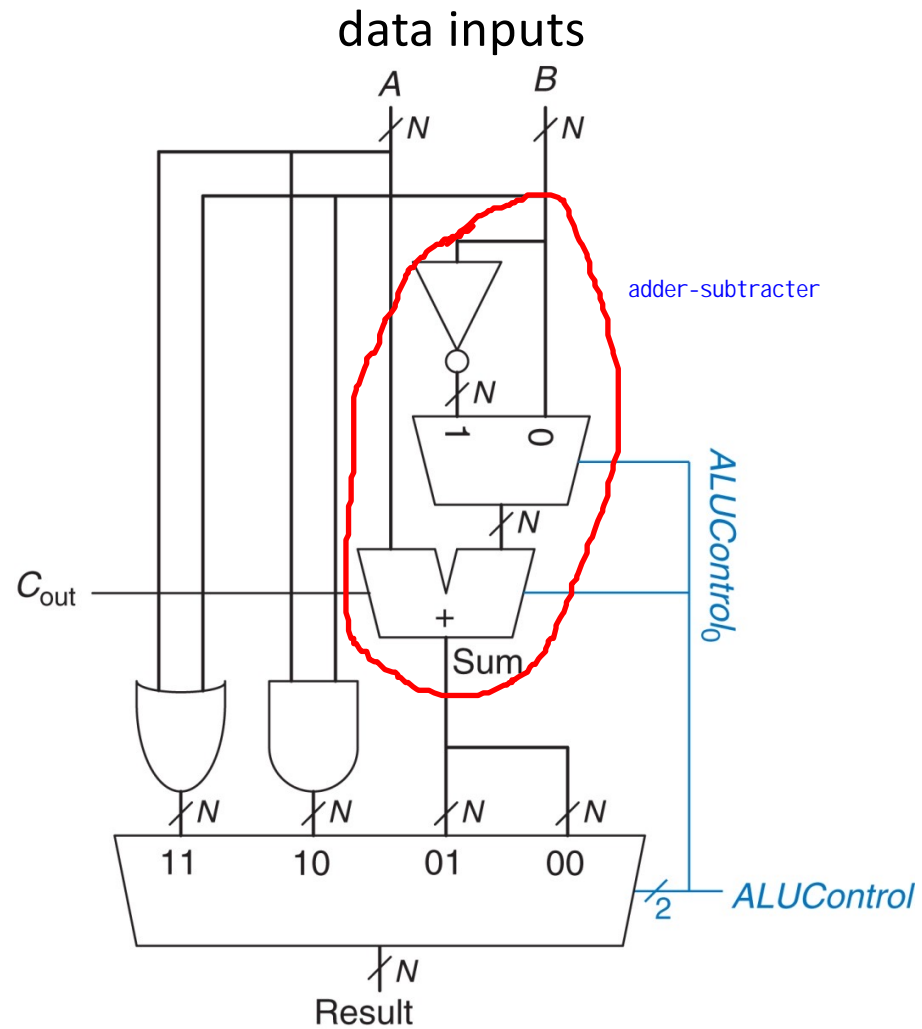
*N-bit Logic circuitry
AND, OR*

4:1 multiplexer



ALU Implementation

ALUControl _{1:0}	Function
00	Add
01	Subtract
10	AND
11	OR



Add-Subtract Circuitry

- $A + B$
 - Normal addition
- $A - B$
 - $A + (-B)$
 - In 2's complement, $-B = B' + 1$
 - An inverter performs B'
 - We send $ALUControl_0$ as the carry input of the adder
 - $ALUControl_0$ is 1 when the ALU function is Subtract

Big Ideas

- **(Parallelism) Hardware is inherently parallel**
 - All logic gates in the ALU work in parallel when the circuit is presented with valid input
- **(Redundancy) Generality leads to redundancy**
 - ALU is a general-purpose circuit that can perform a variety of operations. Some work/effort is wasted
 - The output of OR/AND is wasted when ALUControl is 01
- **(Control) Control circuitry comes with a cost**
 - ALU consumes more area than the individual functional units it combines (4:1 multiplexer is for controlling output)

ALUFLAGS

- We need information about the ALU output
 - Is the result negative (**N**)?
 - Is the result zero (**Z**)?
 - Is there a carry out (**C**)?
 - Is there an overflow (**V**)?
- Many scientific algorithms rely on flags for the “next steps”
 - If overflow, discard result, and redo
 - Carry out is the carry in for another operation
 - If the result is negative: do {...}; else do {...}

Flags are only relevant for arithmetic operations ($ALUControl_1 = 0$)

ALUFLAGS

- Negative
 - Check the MSB of result
- Zero
 - NOR all bits of the result (same as invert then AND)
- Carry
 - AND ALUControl_1 with C_{out} from the adder
- Overflow
 - **Option # 1:** Use A and B to compute overflow
 - **Option # 2:** Use A and the output of 2:1 multiplexer to compute overflow

Option # 1 for Overflow

The following scenarios generate overflow, i.e., the overflow flag needs to be asserted (**1**)

	ALControl ₀	A ₃₁	B ₃₁	S ₃₁
Scenario # 1	0 (Add)	0	0	1
Scenario # 2	0 (Add)	1	1	0
Scenario # 3	1 (Subtract)	0	1	1
Scenario # 4	1 (Subtract)	1	0	0

Case # 1 in plain English: When doing **A + B** , if A and B are +ve, and the sum is -ve

Case # 2: **A + B**, if A and B are -ve, and the sum is +ve

Case # 3: **A - B**, if A is +ve and and B is -ve, and the sum is -ve

Case # 4: **A - B**, if A is -ve and and B is +ve, and the sum is +ve

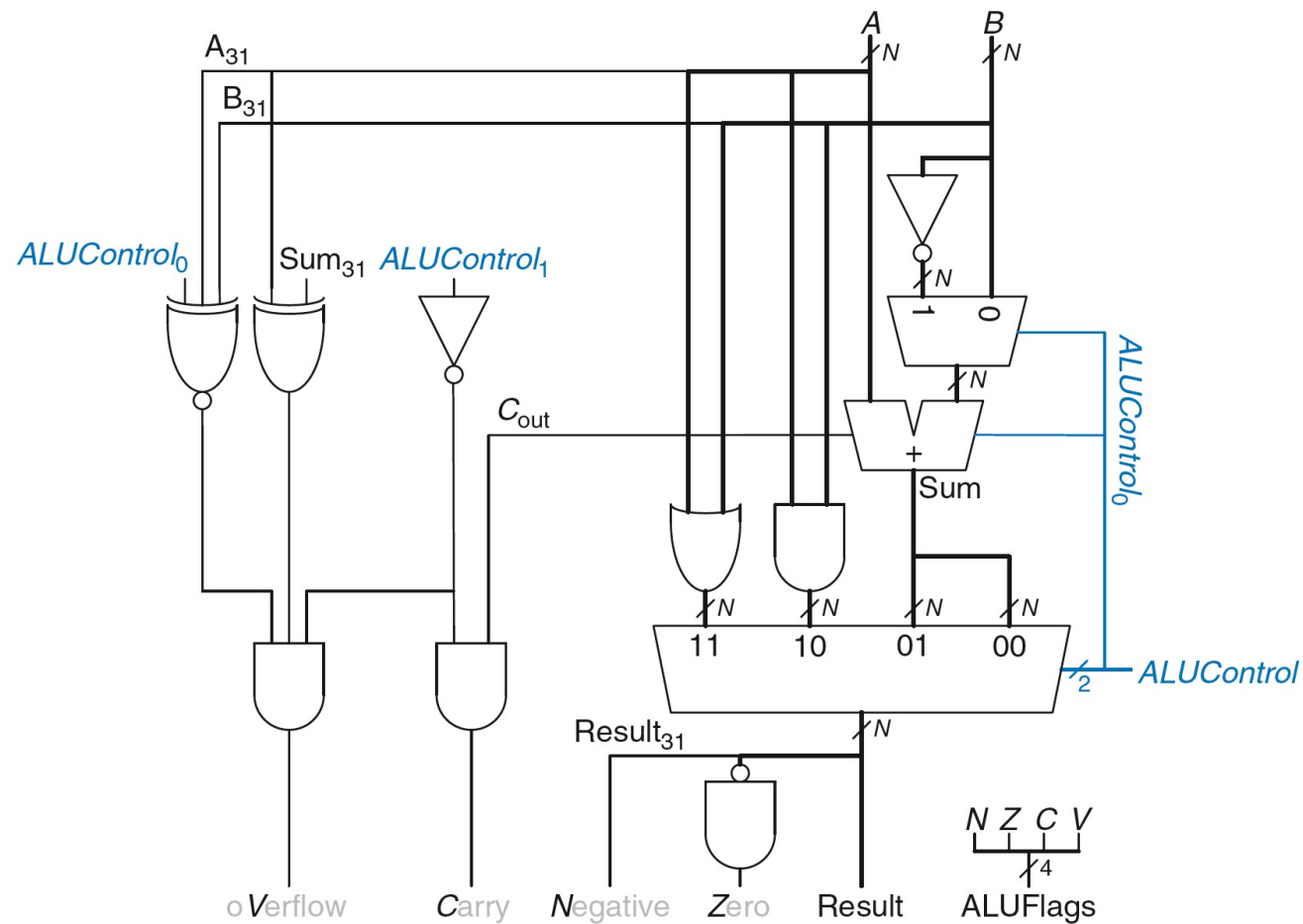
Option # 1 for Overflow

The following scenarios generate overflow, i.e., the overflow flag needs to be asserted (**1**)

	ALUControl ₀	A ₃₁	B ₃₁	S ₃₁
Scenario # 1	0 (Add)	0	0	1
Scenario # 2	0 (Add)	1	1	0
Scenario # 3	1 (Subtract)	0	1	1
Scenario # 4	1 (Subtract)	1	0	0

- Overflow is **1** whenever there is an even number of **1**'s among ALUControl₀, A₃₁, and B₃₁
 - XNOR ALUControl₀, A₃₁, and B₃₁
- Overflow is **1** whenever A₃₁ and S₃₁ are different
 - XOR A₃₁ and S₃₁

Option # 1 for Overflow



Option # 2

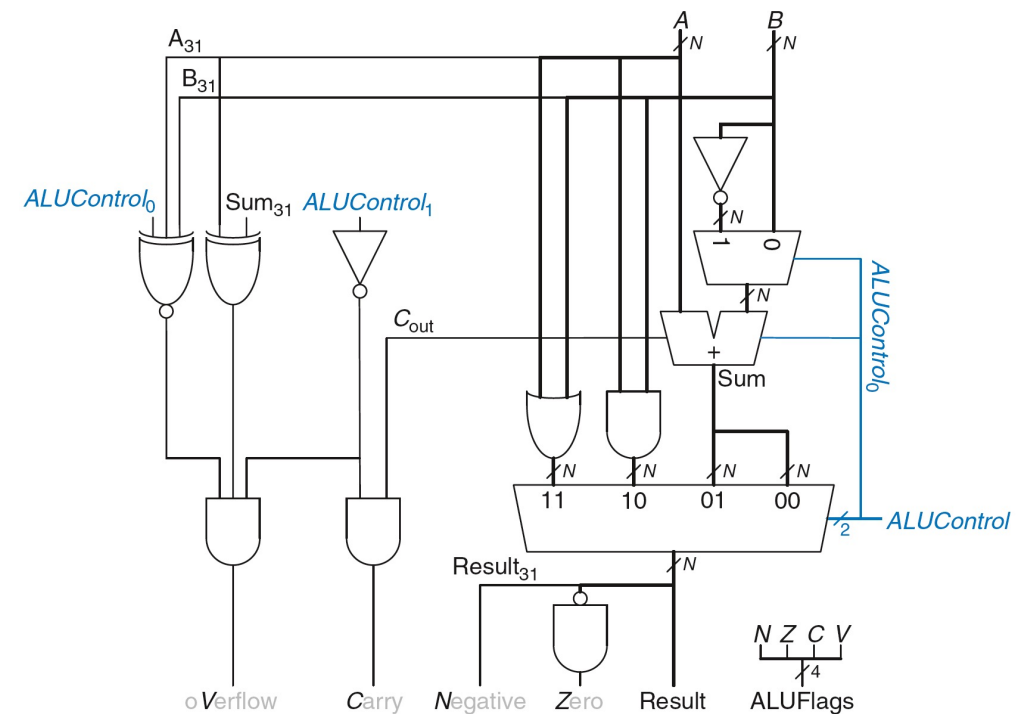
- Use A and the output of 2:1 mux
 - B if the instruction is an Add and $-B$ if the instruction is a subtract
- Easy to reason conceptually
 - If $A - B$ is the same as $A + (-B)$ then everything is an add
 - There is no need to consider subtract separately when reasoning about overflow generation
- The circuitry is also much simpler
 - Homework assignment: Figure out the circuitry for overflow generation with option # 2

ALU Timing Analysis

Homework

picoseconds (10^{-12} seconds) = ps

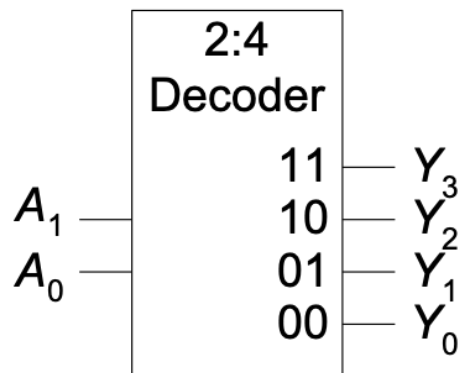
Element	Delay
Inverter	$t_{INV} = 1 \text{ ps}$
2:1 Mux	$t_{mux2} = 5 \text{ ps}$
4:1 Mux	$t_{mux4} = 8 \text{ ps}$
Adder	$t_{adder} = 14 \text{ ps}$
AND	$t_{AND} = 2 \text{ ps}$
OR	$t_{OR} = 2 \text{ ps}$



- Find t_{Result} in ps for the four ALU instructions/functions. Ignore overflow generation
 - Which function takes the longest time (and is the critical path)? Ignore wire delay
- Express t_{Result} in the form of an equation for Add and Subtract. What is the difference?

Decoders

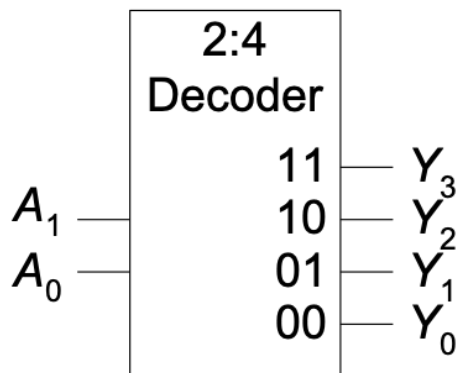
- N **inputs** and 2^N **outputs**
- For each **input** combination, only one of the **outputs** is **1**
 - The **outputs** are affectionately called *one-hot*



Decoders

- N **inputs** and 2^N **outputs**
- For each **input** combination, only one of the **outputs** is **1**
 - The **outputs** are affectionately called *one-hot*

2:4 Decoder Truth Table

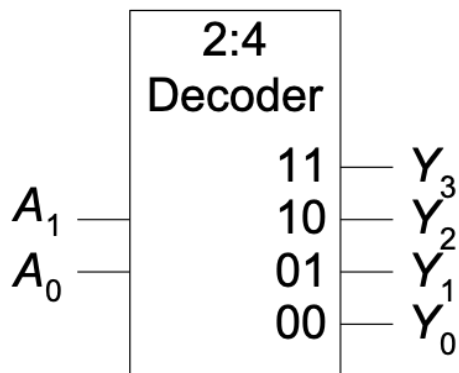


A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Decoders

- N **inputs** and 2^N **outputs**
- For each **input** combination, only one of the **outputs** is 1
 - The **outputs** are affectionately called *one-hot*

2:4 Decoder Truth Table and Boolean Equations



A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

$$Y_0 = A_1' A_0'$$

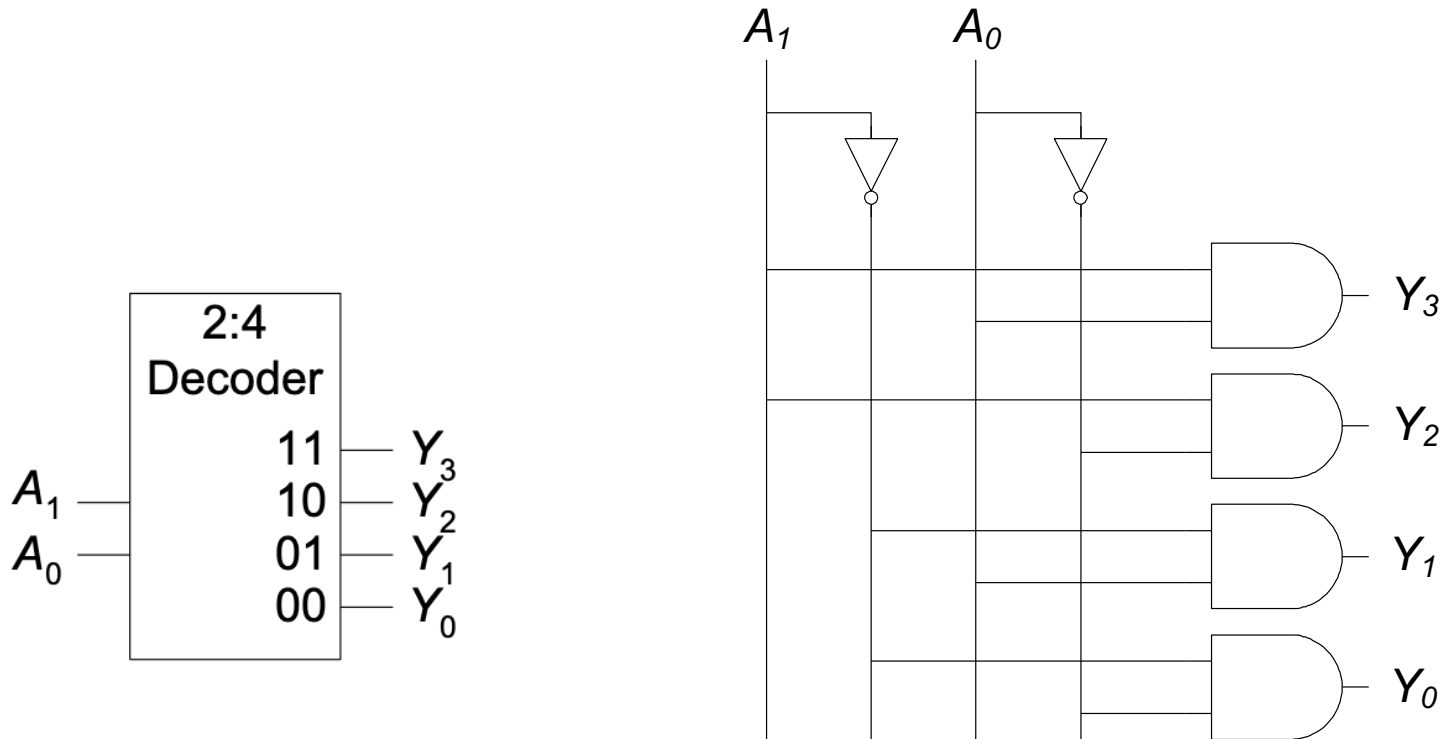
$$Y_1 = A_1' A_0$$

$$Y_2 = A_1 A_0'$$

$$Y_3 = A_1 A_0$$

Decoders

- N **inputs** and 2^N **outputs**
- For each **input** combination, only one of the **outputs** is **1**



$$Y_0 = A_1' A_0'$$

$$Y_1 = A_1' A_0$$

$$Y_2 = A_1 A_0'$$

$$Y_3 = A_1 A_0$$

Decoder Implementation

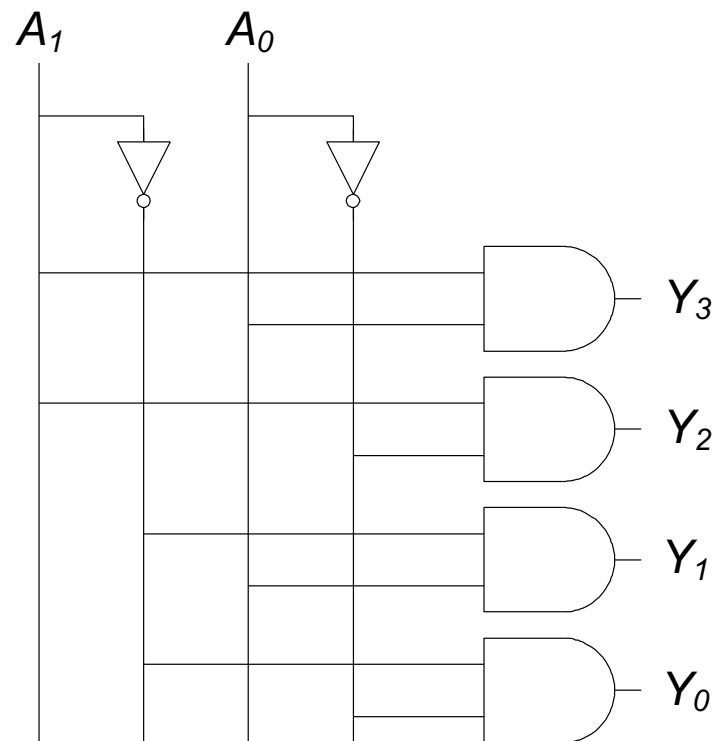
For each **input** combination, only one of the **outputs** is **1**

$$Y_0 = A_1' A_0'$$

$$Y_1 = A_1' A_0$$

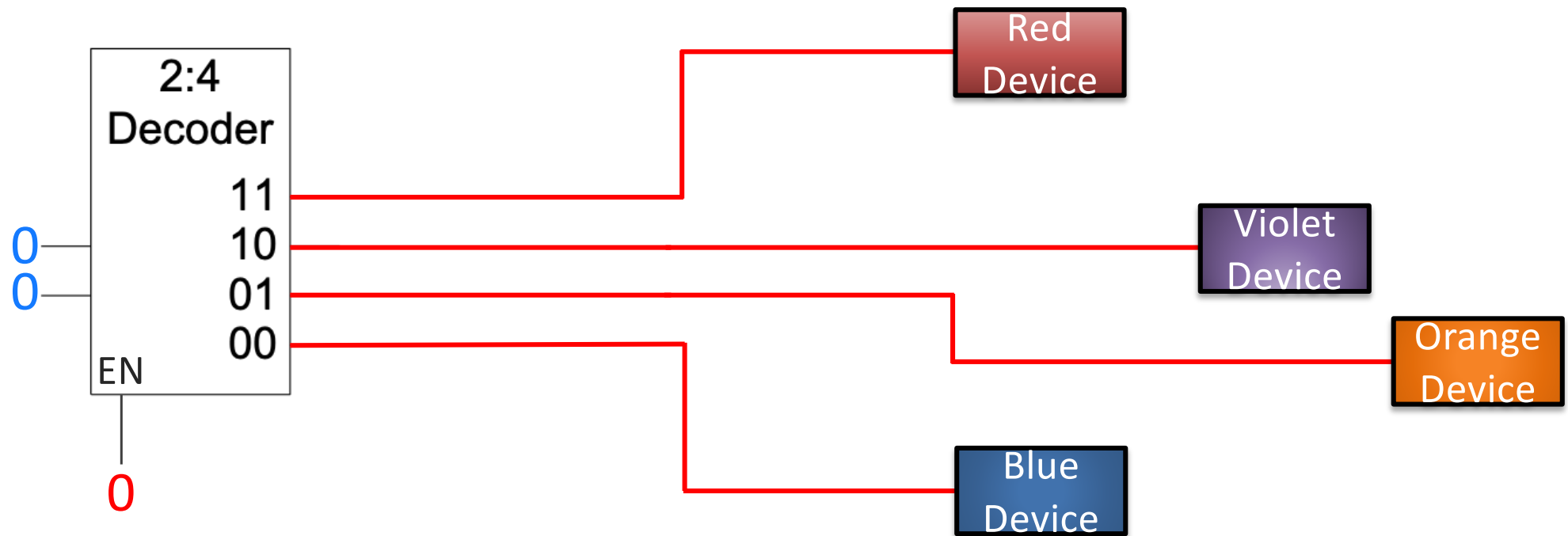
$$Y_2 = A_1 A_0'$$

$$Y_3 = A_1 A_0$$



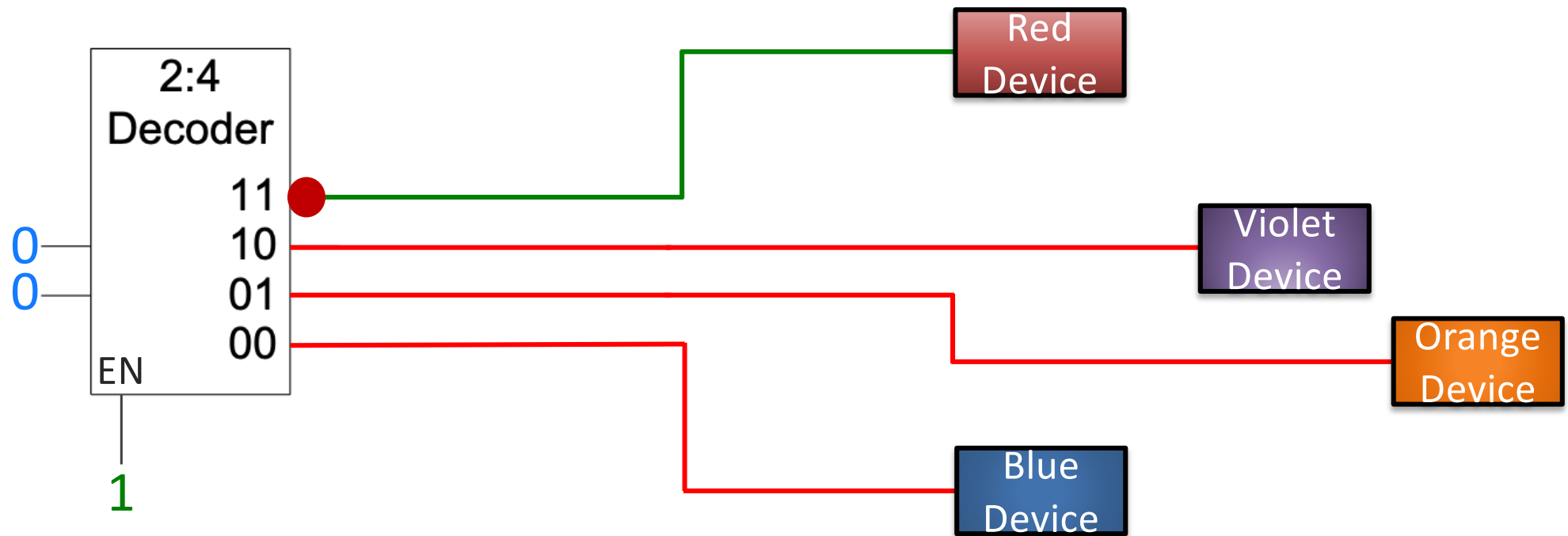
Uses of Decoders

For each **input** combination, only one of the **outputs** is **1**



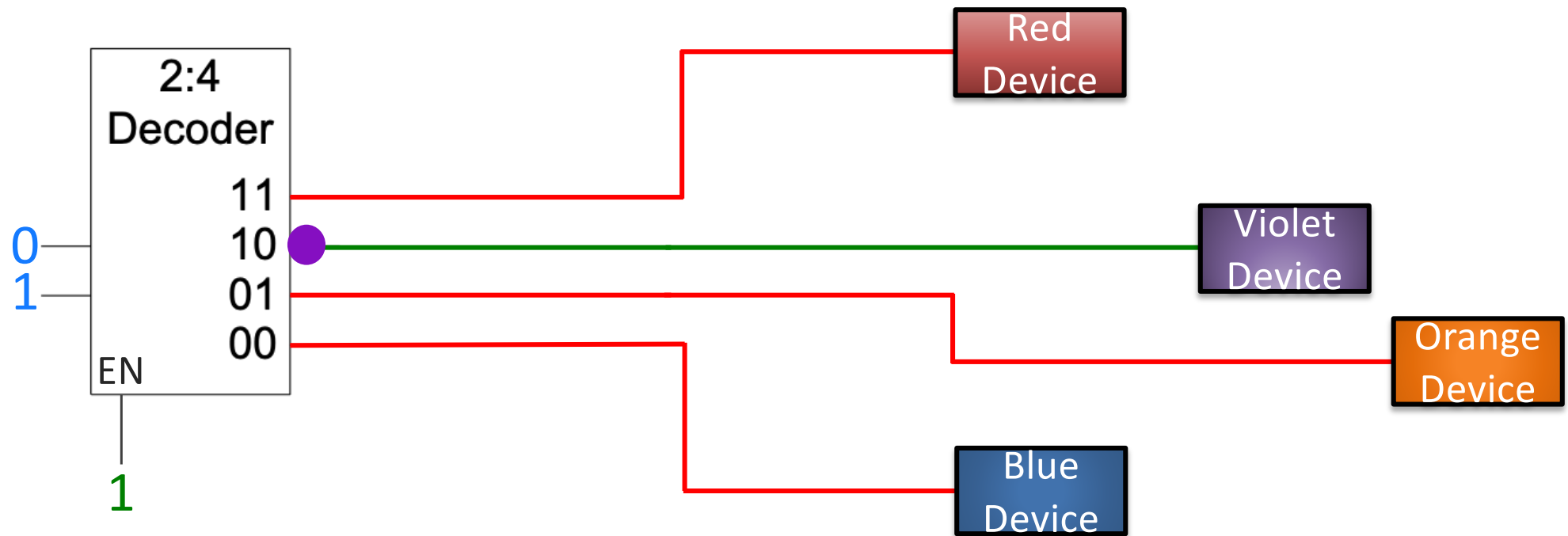
Uses of Decoders

For each **input** combination, only one of the **outputs** is **1**



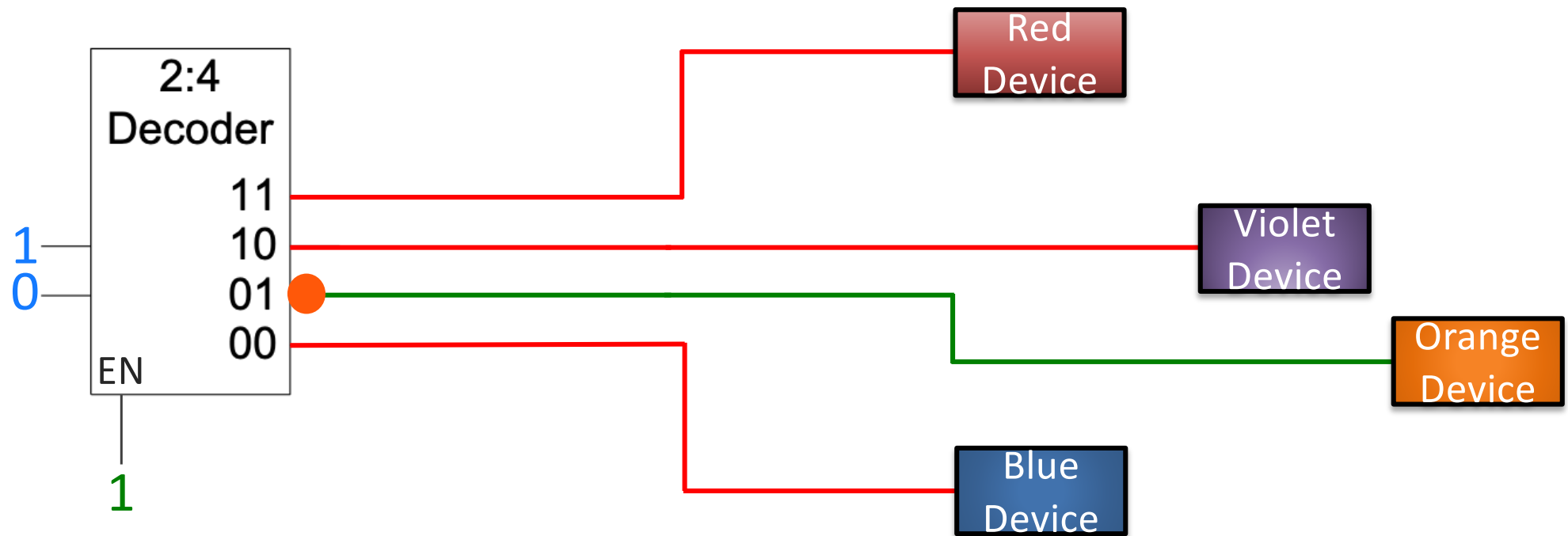
Uses of Decoders

For each **input** combination, only one of the **outputs** is **1**



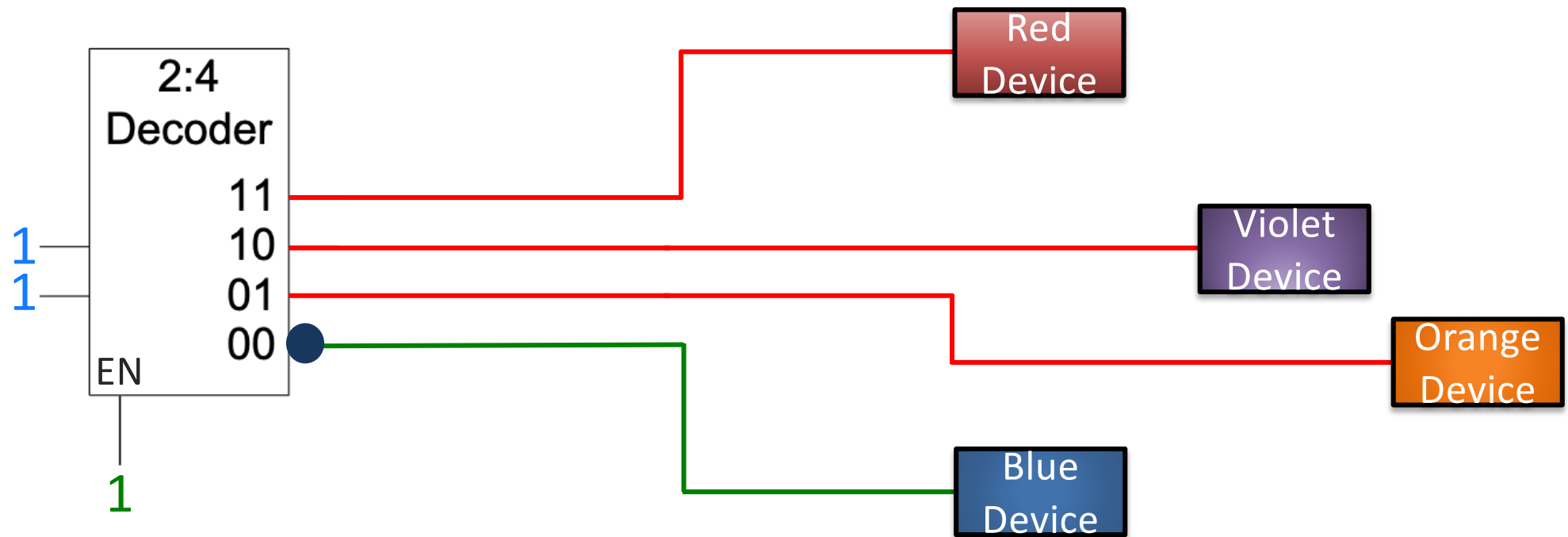
Uses of Decoders

For each **input** combination, only one of the **outputs** is **1**



Uses of Decoders

For each **input** combination, only one of the **outputs** is **1**



Multiplexer Logic

- We have seen how to build multiplexers and other combinational circuits with logic gates
- Question: Can we use a multiplexer to implement a logic gate?
 - If yes, build a 2-input AND gate from a 2:1 multiplexer
 - *Note: You can connect any multiplexer input to **0** (**zero/ground**) or **1** (**high**)*
 - *Interestingly, the answer is yes!*

A 2^N -input multiplexer can be programmed to perform any N-input logic function by applying **0's** and **1's** to the appropriate data inputs

Multiplexer Logic

- Any truth table can be seen as a lookup table
 - If we lookup **00** in a 2-input AND truth table, we see **0**
 - If we lookup **10** in a 2-input OR truth table, we see **1**
- Multiplexers can be used as lookup tables
 - Connect the data inputs to **0** or **1**
 - Use inputs (A and B) as select lines/signals

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

$Y = AB$

