CMSC389E:

Digital Logic Design Through Minecraft

Adders, Encoders, Decoders

Fall 2020
Akilesh Praveen

Administrivia

- Project Submission via ELMS
  - CS Department Bureaucracy :(

- Keep checking the course website! (will be updated tonight)
  (http://www.cs.umd.edu/class/fall2020/cmsc389E/)

Announcements

- Project 2
  - Conceptual knowledge in today’s lecture + online textbook
  - Project spec will be released on ELMS and course website **tonight / Saturday**

Numbers and Computers

Numbers & Computers

- Let’s talk about numbers and computers
- How do they deal with them?

Numbers & Computers

- Let’s talk about numbers and computers
- How do they deal with them?
    - In binary

Numbers & Computers

- Let’s talk about numbers and computers
- How do they deal with them?
  - In binary
- How do **we** deal with them?

Numbers & Computers

- Let's talk about numbers and computers
- How do they deal with them?
  - In binary
- How do **we** deal with them?
  - In decimal

Numbers & Computers

- We’re going to work through creating circuits that let us
  - **Manipulate** binary numbers on a computer
  - **Convert** binary numbers to decimal (for us to work with) and vice versa

- *READ: Digital Logic & Comp Arch in Minecraft, C3*

Adders

Adders

- We all know how to add numbers

Adders

- We all know how to add numbers
  - Hopefully

Adders

- We all know how to add numbers
    - Hopefully
- How do computers add them?

Adders

- We all know how to add numbers
  - Hopefully
- How do computers add them?
  - First, think about number representation in **digital logic circuits**

Adders

- We all know how to add numbers
  - Hopefully
- How do computers add them?
  - First, think about number representation in **digital logic circuits**
- Let’s work through an example

Adders

- How would you, **as a human**, add the following numbers
  - 0b11101
  - 0b11011

Adders

- How would you, **as a human**, add the following numbers
  - `0b11101`
  - `0b11011`
- Let's take it back to elementary school

Adders

- Remember, we're working with binary

$$\begin{array}{r} 11101 \\ +11011 \\ \hline \end{array}$$

Adders

- Let’s add the first ‘column’ here

$$
\begin{array}{r}
11101 \\
+11011 \\
\hline
0
\end{array}
$$

Adders

- We seem to have a 'carry-over' happening here

$$\begin{array}{r} {\color{red}1} \\ 11101 \\ +11011 \\ \hline 0 \end{array}$$

Adders

- This happens when we exceed the a number system's limit with an 'add'
- Where else during this computation would this occur?
    - Keep this one in mind

$$
\begin{array}{r}
{\color{red}1} \\
11101 \\
+11011 \\
\hline
0
\end{array}
$$

Half Adders

Half Adders

- Now, think about how a computer would do this operation

$$
\begin{array}{r}
{\color{red}1} \\
11101 \\
+\,11011 \\
\hline
0
\end{array}
$$

Half Adders

- Now, think about how a computer would do this operation
  - Let's make it into a smaller problem

$$\begin{array}{r} 1 \\ 11101 \\ +\,11011 \\ \hline 0 \end{array}$$

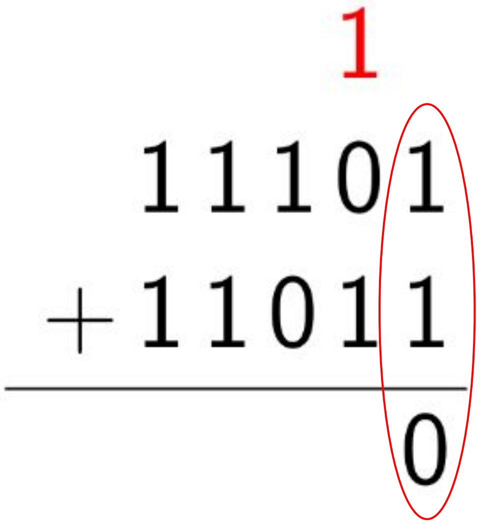
Half Adders

- Now, think about how a computer would do this operation
  - Let's make it into a smaller problem
  - Adding two **one bit binary numbers**

$$
\begin{array}{r}
1 \\
11101 \\
+11011 \\
\hline
0
\end{array}
$$

Half Adders

- In particular, **this** operation right here is what we want to do
- Consider the possible inputs and outputs for an operation like this
  - How many bits in?
  - How many bits out?



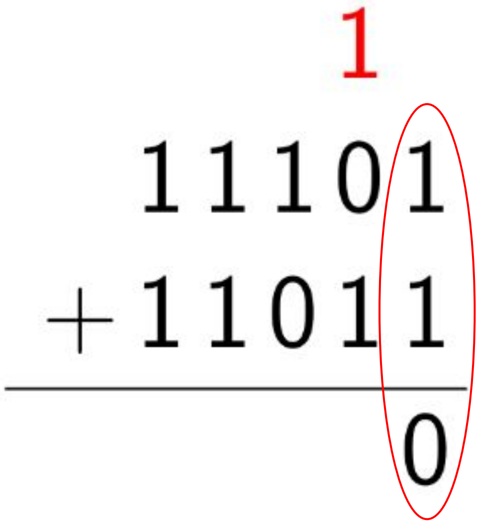
Half Adders

- We want **two** bits input
- We want **two** bits output

$$
\begin{array}{r}
\color{red}{1} \\
11101 \\
+11011 \\
\hline
0
\end{array}
$$

Half Adders

- We want **two** bits input
  - Input number 1 (one digit)
  - Input number 2 (one digit)
- We want **two** bits output
  - One for the sum
  - One for the ‘carry’



Half Adders

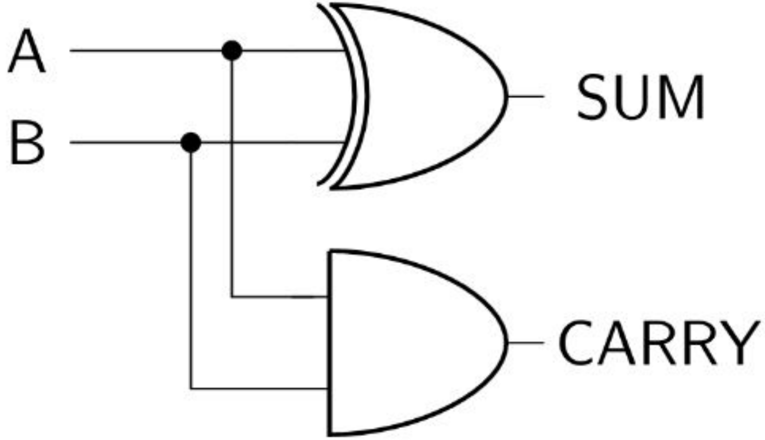
- Turns out, **generating** those two output bits given our two inputs is pretty easy!

- *READ: Digital Logic & Comp Arch in Minecraft, C3.1*

Half Adders

- Here’s our truth table and logic circuit

Half Adder		
In	Sum	Carry
0 0	0	0
0 1	1	0
1 0	1	0
1 1	0	1



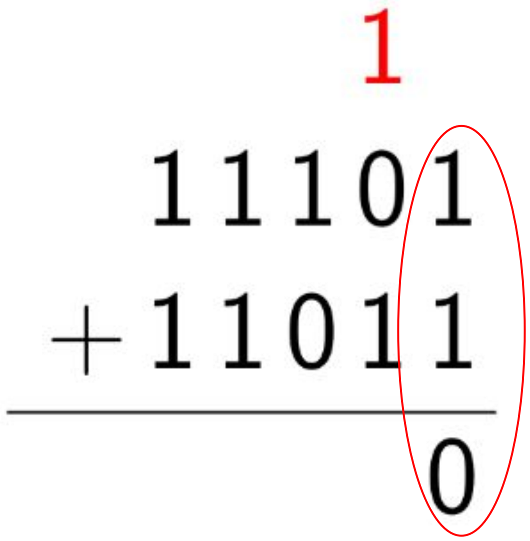
Full Adder

Full Adders

- The Half Adder is great, but limited in functionality
- Let’s work on scaling it up
  - From adding **two one digit numbers**
  - To adding **two *n* digit numbers**
- How can we accomplish this?
  - Think back to our addition example

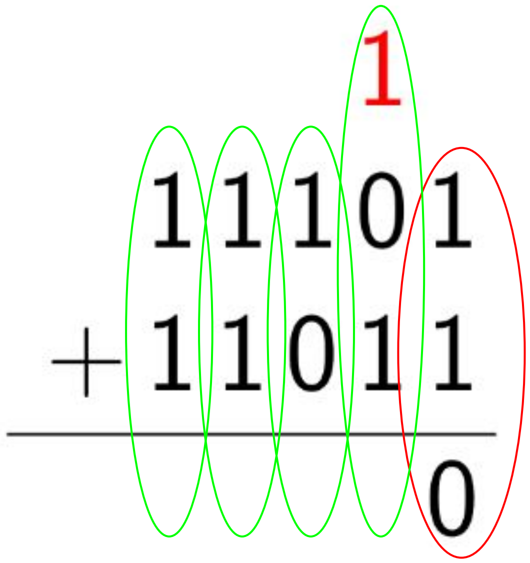
Full Adders

- In other words, our **half-adder** takes care of this one...



Full Adders

- But our new circuit needs to handle all of these as well

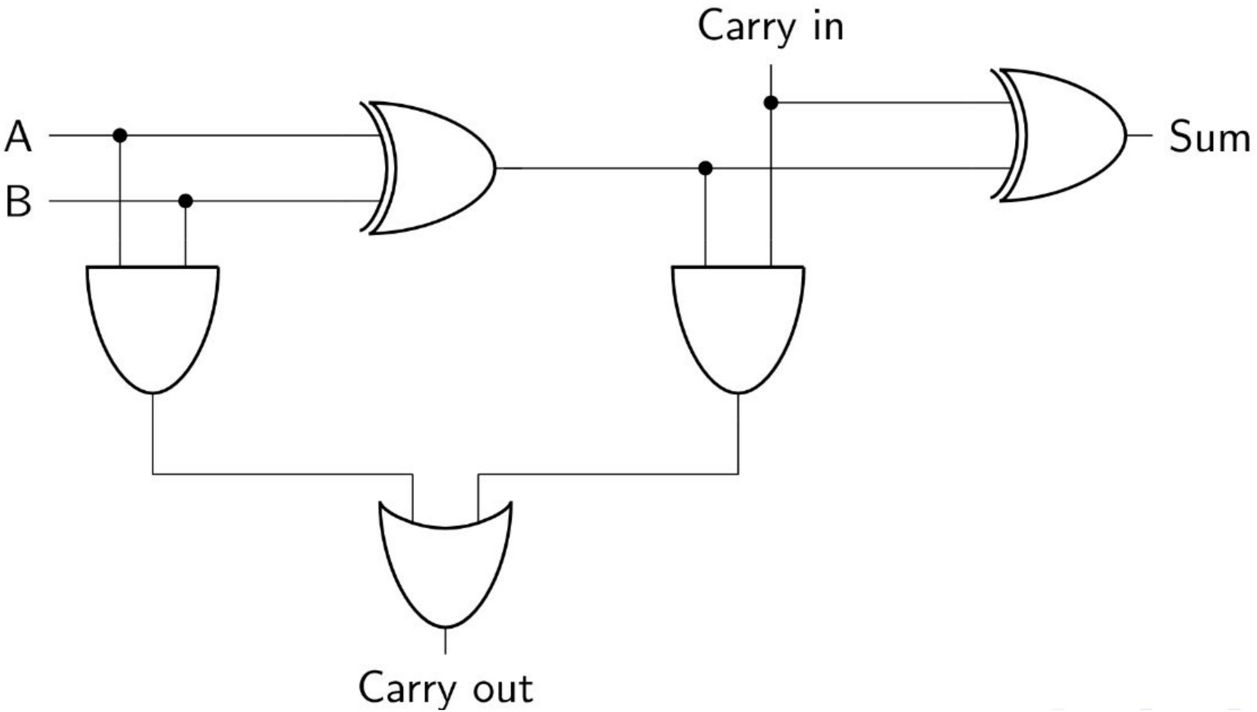


Full Adders

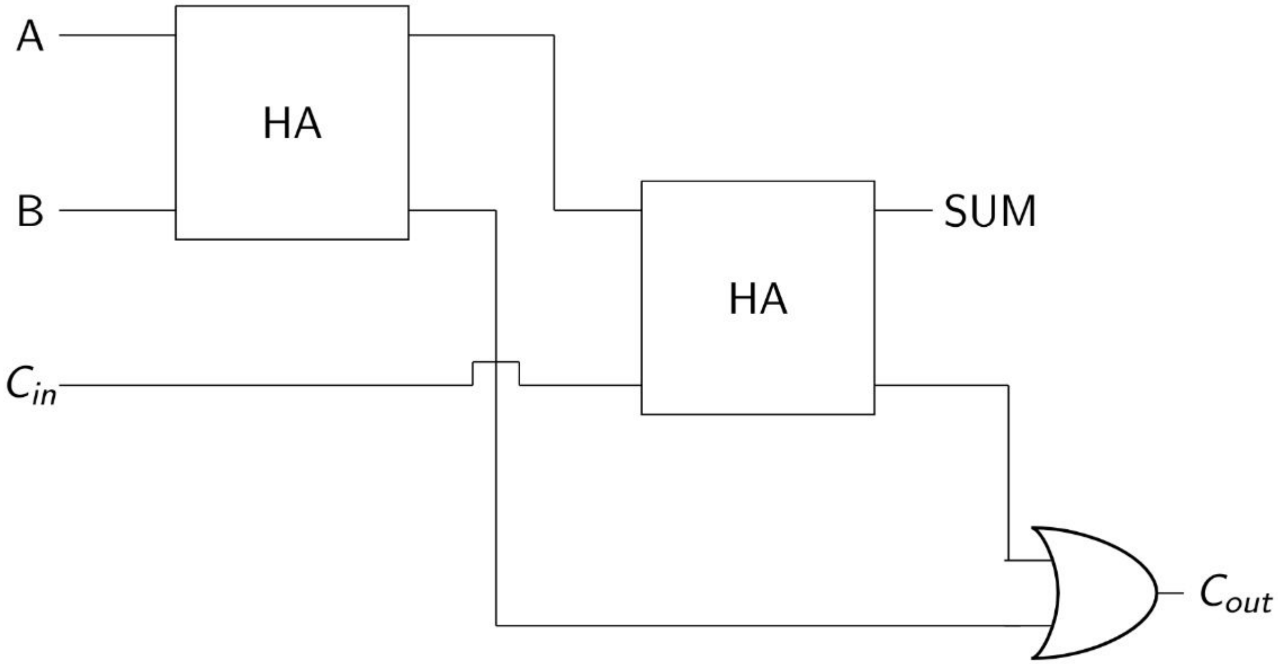
- Our solution?
  - Just handle a ‘carry in’ bit!
  - It turns out, that’s all we need

- *READ: Digital Logic & Comp Arch in Minecraft, C3.2*

Full Adders

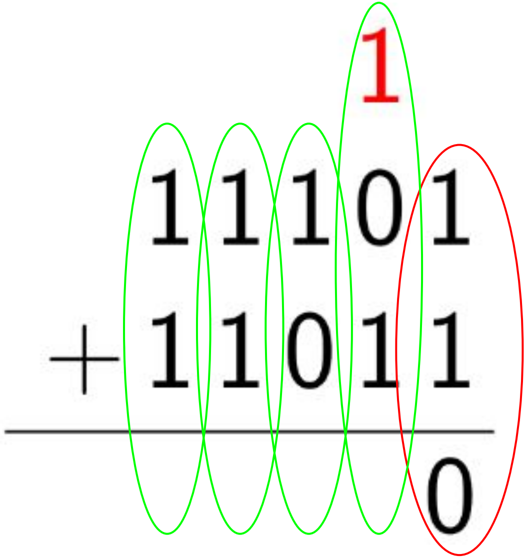


Full Adders



Full Adders

- Q: How would doing all this sequential math actually look in terms of a circuit?



Full Adders

- A: Like this
    - Note that the adder determining S_0 can also be a HA

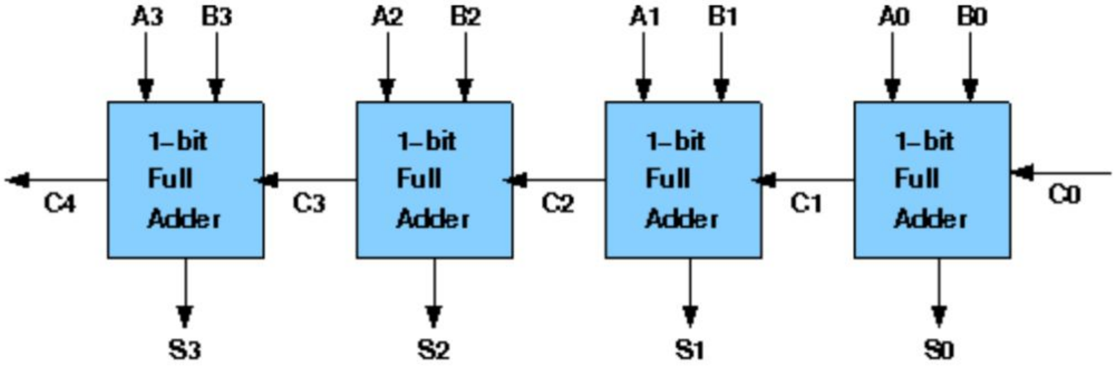


Image courtesy of IIT

Full Adders

- In terms of Minecraft, we really only **need** to know how to make a Full adder
  - A half adder is the same thing, without a carry bit
- Luckily, there's a pretty simple way to put together a full adder leveraging Minecraft's odd nuances
  - Full adder demo by Ashwath

Decoders & Encoders

Decoders & Encoders

- How will we get this data out of our computer and into an interpretable form?
- I.e. how can we convert this binary representation into decimal representation easily?

Decoders & Encoders

- This is an essential problem in digital logic
    - Pertaining to computer design
- Engineers have designed two logical circuits help us interface with different number systems
    - Decoder
    - Encoder

Decoders & Encoders

- **Decoders** convert binary signals into decimal representations of those signals
- **Encoders** convert decimal representations of signals back into binary
- These circuits can be generalized to different number systems as well

Decoders & Encoders

- **Decoders:** Binary to decimal
- **Encoders:** Decimal to binary

Decoders & Encoders

- Let's first talk about decoders
- What would you do (as a human) if I asked you to convert the following binary number to decimal form?
  - `0b110`

Decoders & Encoders

- Let's first talk about decoders
- What would you do (as a human) if I asked you to convert the following binary number to decimal form?
  - `0b110`
- Now think in terms of three wires coming in, each representing a digit

Decoders & Encoders

- A solution: assign a wire for each possible output
  - If the binary input is 0, turn only a certain wire on
  - If the binary input is 1, turn only another wire on
  - If the binary input is 2, turn only another wire on
  - etc.

Decoders & Encoders

- We will need more output wires than input wires

Decoders & Encoders

- We will need more output wires than input wires
- Given 3 inputs

Decoders & Encoders

- We will need more output wires than input wires
- Given 3 inputs
  - We will need 8 outputs

Decoders & Encoders

- We will need more output wires than input wires
- Given *3* inputs
  - We will need 8 outputs
- Given *n* inputs?

Decoders & Encoders

- We will need more output wires than input wires
- Given *3* inputs
  - We will need 8 outputs
- Given *n* inputs
  - We will need 2^n outputs

Decoders & Encoders

- Let's figure out our smaller example first
  - How will we take 3 input binary values and select one of 8 outputs for them?
  - It turns out, the answer is fairly straightforward once you create a truth table

Decoders & Encoders

- Let's figure out our smaller example first
  - How will we take 3 input binary values and select one of 8 outputs for them?
  - It turns out, the answer is fairly straightforward once you create a truth table

Decoders & Encoders

Table I Truth table for 3 to 8 decoder

Enable pin	Input lines			Output lines							
E	I_2	I_1	I_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

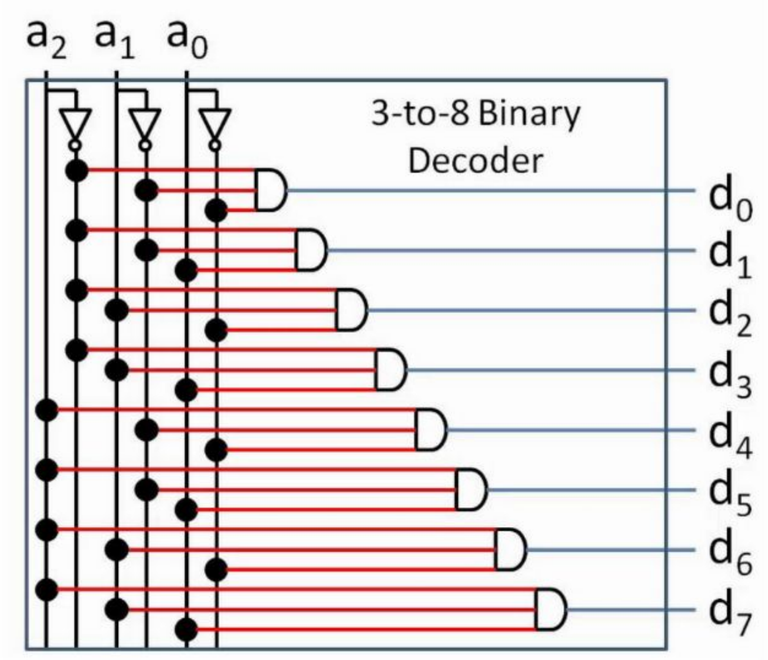
Note: 'x' denotes don't care condition

Decoders & Encoders

- Once we've made our truth table, it becomes much easier to create a logical circuit
- Notice the clever use of input buses (we will put up another video about those in the near future)

READ: Digital Logic & Comp Arch in Minecraft, C3.3

Decoders & Encoders



Decoders & Encoders

- Suppose we wanted to go the other way around
- From decimal to binary
  - Why?

- *READ: Digital Logic & Comp Arch in Minecraft, C3.5*

Decoders & Encoders

- Encoders are essentially the complement to decoders
  - Suppose this time that we had 8 input wires, each signifying numbers from 0-7
  - Our goal now is to decide binary output based on which wire out of the 8 is ‘on’

Decoders & Encoders

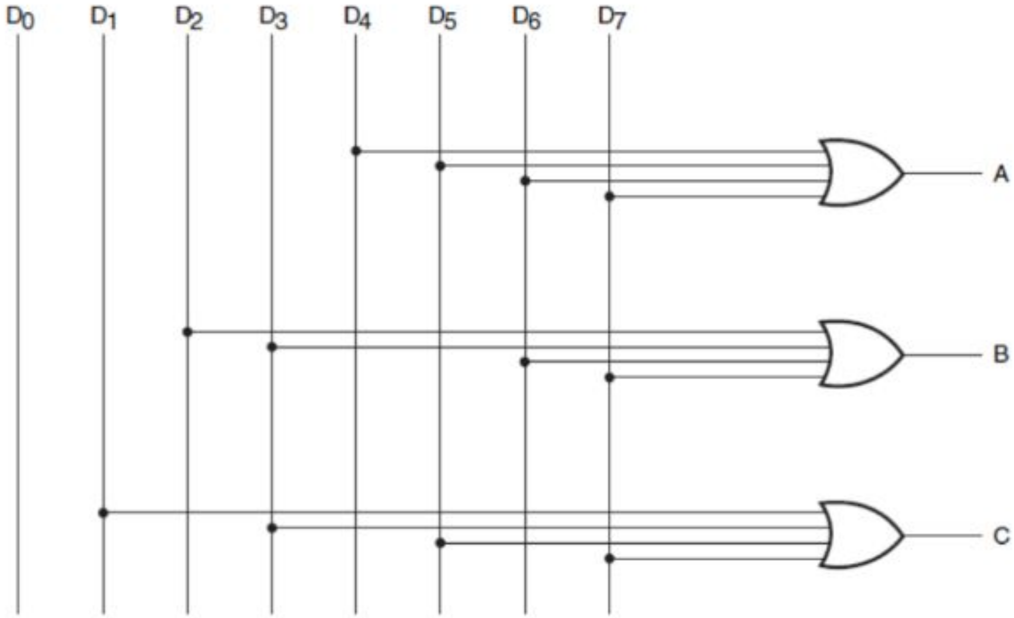
- The logical circuit is strikingly similar to the decoder
- First, let's look at the truth table

D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Decoders & Encoders

- Now, the circuit

READ: Digital Logic & Comp Arch in Minecraft, C3.4



Buses Demo

Project 2