

INTRODUCTION TO DIGITAL LOGIC

Robert P. Webber, Longwood University

Digital logic is at the “bottom” of the computer. It models what happens when we strip away all the programming languages and prewritten packages, and try to understand how the computer actually performs operations. For example, how does a computer actually add two binary numbers? To understand that, we have to understand digital logic.

Digital logic is based on the fundamental operations from symbolic logic: **and**, **or**, and **not**. These are called **logical operations** or **Boolean operations**, after George Boole, who was the first person to analyze them.

Boolean variables have precisely two values, `true` and `false` (often represented by 1 and 0). A Boolean value can be represented by a bit, or a binary digit.

We can represent Boolean operations by tables of 0s and 1s, showing what happens in every combination of bits. These tables are like the **truth tables** that are used in symbolic logic.

Here are the definitions of the basic functions.

Not This function just reverses the value of the variable: **not** `true` = `false`, and **not** `false` = `true`. Let *A* be the variable. Possible values of *A* are 1 and 0 (`true` and `false`).

<i>A</i>	not <i>A</i>
1	0
0	1

Or The **or** function takes two operands. It is `true` if at least one operand is `true` and `false` if both are `false`. Let *A* and *B* be the operands. Each can be `true` or `false`. This means there are four possible combinations.

<i>A</i>	<i>B</i>	<i>A or B</i>
0	0	0
0	1	1
1	0	1
1	1	1

The order of the rows doesn't matter.

<i>A</i>	<i>B</i>	<i>A or B</i>
1	1	1
1	0	1
0	1	1
0	0	0

defines the same operation.

And This function takes two operands. It is `true` if both operands are `true` and `false` otherwise.

<i>A</i>	<i>B</i>	<i>A and B</i>
0	0	0
0	1	0
1	0	0
1	1	1

These functions have special graphics symbols.



The functions are called **circuits** when we represent them graphically.

Of course, we could define other logical functions by means of tables, too. Here's such a function *F*

<i>A</i>	<i>B</i>	<i>F</i>
0	0	1
0	1	1
1	0	1
1	1	0

That is, $0 F 0 = 1$, $0 F 1 = 1$, and so on; or, as mathematicians prefer to write, $F(0,0) = 1$, $F(0,1) = 1$, $F(1,0) = 1$, and $F(1,1) = 0$.

Indeed, this function turns out to be very useful, and it also has a name: **nand**. Notice that the column for *F* is the opposite of the column for **and**.

A	B	$A \text{ and } B$	$A \text{ nand } B$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

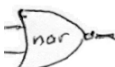
That is, **nand** is the same thing as **not and**, which means **and** followed by **not**. It even has a special graphics symbol:



Two other very useful functions are **nor** and **xor**.

nor (**not or**, which means **or** followed by **not**)


A	B	$A \text{ nor } B$
0	0	1
0	1	0
1	0	0
1	1	0

The graphics symbol for **nor** is 

xor (exclusive **or**)

A	B	$A \text{ xor } B$
0	0	0
0	1	1
1	0	1
1	1	0

Notice that **xor** is true if exactly one of the operands is true and false otherwise. It isn't obvious how to write this function using **and**, **or**, and **not**, but we'll figure out how to do it shortly.

The graphics symbol for **xor** is 

Other logical functions could be defined (see exercises 1 and 2, for instance), but they haven't been found to be as useful as these. Also, we can define more complicated logical functions by combining the basic ones. For instance, consider the function

(not A) and (not B)

Derive its truth table as follows.

A	B	not A	not B	(not A) and (not B)
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

Notice the final column is the same as the **nor** table! We have just shown that these circuits are **equivalent**; that is, that they have the same truth values.

$$A \text{ nor } B \Leftrightarrow (\text{not } A) \text{ and } (\text{not } B)$$

Recall that we defined above that

$$A \text{ nor } B \Leftrightarrow \text{not } (A \text{ or } B)$$

Therefore, we have shown that

$$\text{not } (A \text{ or } B) \Leftrightarrow (\text{not } A) \text{ and } (\text{not } B)$$

We can determine whether any two logical functions are equivalent by comparing their truth tables. The functions are equivalent if and only if the truth values are the same.

Electrical engineers can physically construct these six basic circuits from wire, silicon, and transistors. They are called **gates**, and they, or at least a subset of them, form the basis for modern computers. A computer is, at its most basic level, a bunch of connected gates.

It isn't necessary to build all six gates. Some of them can be written as combinations of others. For instance, we have already seen that **nor** can be written as a combination of **not** and **and** gates:

$$A \text{ nor } B \Leftrightarrow (\text{not } A) \text{ and } (\text{not } B)$$

If we have two **not** gates and one **and** gate, we can build a circuit that will perform the **nor** function.

Similarly, we can write **nand** and **xor** as combinations of **and**, **or**, and **not** gates (Exercises 9 and 11). This means that the set of gates **{and, or, not}** is **functionally complete**, because the other basic gates (and indeed, any logical circuit) can be built from only these three gates.

Are there other functionally complete sets of gates? The answer is yes. Indeed, **and** can be written as a combination of **or** and **not**:

$$A \text{ and } B \Leftrightarrow \text{not } ((\text{not } A) \text{ or } (\text{not } B)) ,$$

which you can verify using a truth table. So **{not, or}** is functionally complete.

Similarly, **or** can be written as a combination of **and** and **not**.

$$A \text{ or } B \Leftrightarrow \text{not } ((\text{not } A) \text{ and } (\text{not } B)) ,$$

so **{not, and}** is functionally complete.

Surprisingly, **{nand}** by itself is functionally complete!

$$\text{not } A \Leftrightarrow A \text{ nand } A$$

$$A \text{ and } B \Leftrightarrow (A \text{ nand } B) \text{ nand } (A \text{ nand } B) \quad (\text{Exercise 13})$$

$$A \text{ or } B \Leftrightarrow (A \text{ nand } A) \text{ nand } (B \text{ nand } B) \quad (\text{Exercise 14})$$

Similarly, **{nor}** is a functionally complete set. This means that a computer engineer with a big supply of **nand** or **nor** gates could theoretically build the most powerful computer in the world.

EXERCISES

1. Here is a truth table for a logical function F .
What is the value of F for inputs

I_2	I_1	F
0	0	1
0	1	0
1	0	1
1	1	1

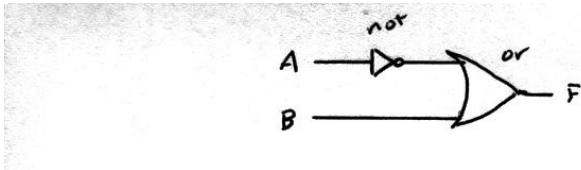
- a. $I_2 = 1, I_1 = 0$?
b. $I_2 = 0, I_1 = 1$?
c. $I_2 = 1, I_1 = 1$?

2. Here is a truth table for a logical function F .
What is the value of F for inputs

I_2	I_1	F
0	0	1
0	1	1
1	0	0
1	1	1

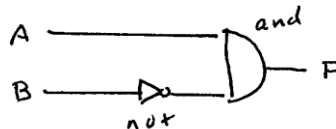
- a. $I_2 = 0, I_1 = 1$?
b. $I_2 = 1, I_1 = 0$?
c. $I_2 = 0, I_1 = 0$?

3. Here is the diagram for a circuit. Complete the truth table. The first line has been done for you as an example.



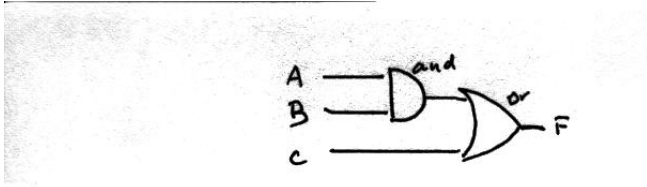
A	B	F
0	0	1
0	1	
1	0	
1	1	

4. Here is a diagram for a circuit. Complete the truth table. The first line has been done for you as an example.

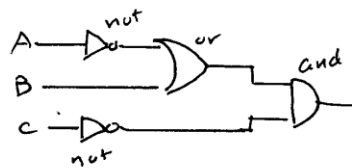


A	B	F
0	0	0
0	1	
1	0	
1	1	

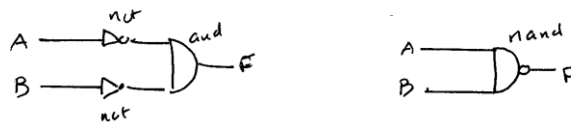
5. Here is a diagram for a circuit. Construct the truth table.



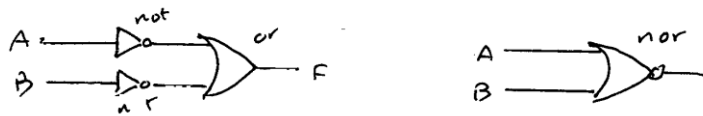
6. Here is the diagram for a circuit. Construct the truth table.



7. Are the following circuits functionally equivalent?



8. Are the following circuits functionally equivalent?



In exercises 9 through 14, write the truth table for the function.

9. **not (not x)**
10. **not (x or y)**
11. **A and (not B)**

12. $A \text{ or } (A \text{ and } B)$
13. $X \text{ and } (X \text{ or } Y)$
14. $x \text{ and } (x \text{ and } y)$
15. Are the circuits $x \text{ and } x$ and x functionally equivalent?
16. Are the circuits $x \text{ or } (x \text{ and } y)$ and x functionally equivalent?
17. Are the circuits $\text{not } (A \text{ or } B)$ and $(\text{not } A) \text{ or } (\text{not } B)$ functionally equivalent?
18. Are the circuits $\text{not } (A \text{ and } B)$ and $(\text{not } A) \text{ or } (\text{not } B)$ functionally equivalent?
19. Design a circuit that is functionally equivalent to a *NAND* gate using only *AND*, *OR*, and *NOT* gates.
20. Design a circuit that is functionally equivalent to a *NOR* gate using only *AND*, *OR*, and *NOT* gates.
21. Design a circuit that is functionally equivalent to a *XOR* gate using only *AND*, *OR*, and *NOT* gates.
22. Here is the truth table for a logical function \Rightarrow . Design a functionally equivalent circuit using only *AND*, *OR*, and *NOT* gates.

A	B	\Rightarrow
0	0	1
0	1	1
1	0	0
1	1	1

23. Design a circuit that is functionally equivalent to an *AND* gate using only *NAND* gates.
24. Design a circuit that is functionally equivalent to an *OR* gate using only *NAND* gates.