CHAPTER 2

Circuits

Chapter 1 describes how numbers are represented and manipulated by computers, but how are these representations physically realized, and how are these manipulations actually effected? At a high level, computer components (such as *central processing units* or *CPUs*) are constructed from *digital circuits* which are constructed from *logic gates* which are in turn ultimately constructed from *transistors*. In this chapter, we examine digital circuits and how they are constructed from logic gates (and ultimately transistors), and in the next chapter we will examine the mathematics which underpins these components at a logical level, *Boolean algebra*.

2.1 Transistors and Switches

A transistor is effectively a digital switch which is used to either establish or break an electrical connection, in much the same way that a light switch can either connect or disconnect a light bulb to or from household current. Diagrammatically,[1] switches are shown below. Note that the

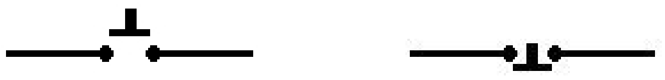


Figure 2.1: Diagrammatic representation of a digital switches. The left switch is "normally open" while the right switch is "normally closed."

[1]The switching, logic gate, and circuit diagrams in this chapter are courtesy of the Wikipedia: `http://en.wikipedia.org/wiki/Logic_gate` and `http://en.wikipedia.org/wiki/Adder_(electronics)`.

left switch is “normally open” and “pushing” the switch establishes the electrical connection, while the right switch is “normally closed” and “pushing” the switch breaks the electrical connection.

2.2 Basic Logic Gates: AND, OR, NOT

Switches can be wired together to form basic *logic gates* which are used to construct circuits which can manipulate numbers. The basic logic gates are the AND, OR, and NOT gates.

2.2.1 AND Gate

An AND gate takes two inputs (switches) and is “on” (switched) so long as *both* switches have been “pushed”. In terms of switches, an AND gate is represented diagrammatically as follows. In this diagram, A and B represent the two input switches, and a connection is established

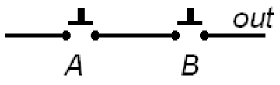


Figure 2.2: Switch diagram of an AND gate.

only if both switches are “pushed.” Logic gates arise so frequently that they have their own diagrammatic representations; the diagram corresponding to an AND gate is given below. Actual CPUs constructed from circuits, logic gates, and ultimately transistors do not function

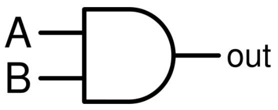


Figure 2.3: Logic diagram of an AND gate.

physically like switches in that no transistor is actually ever “pushed.” Instead, a “high” voltage (typically +5V) given as input to a transistor causes it to “close” and supply a “high” voltage to its output; similarly, a “low” voltage (typically 0V) given as input to a transistor causes it to remain “open” and supply no voltage (i.e., 0V) to its output. Physically and logically, binary 1s and 0s are represented by these “high” and “low” voltages, respectively. Given this representation, we can describe the action of an AND gate using a *truth table*. For example, the truth table corresponding to the possible actions of an AND gate are given below. Given two inputs (A and B) which can each take on two values (0 or 1), there are four possible input

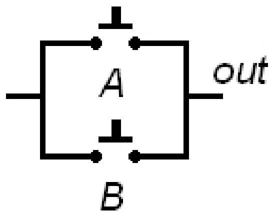
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

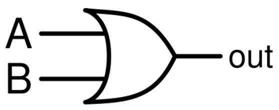
Figure 2.4: Truth table corresponding to an AND gate.

pairs to the AND gate. Each row in the truth table corresponds to one such input pair, and the corresponding output of the AND gate is also given. Note that the "A AND B" is 1 if and only if both A and B are 1; this corresponds to the logical idea that for a connection to be established, both switches must be "pushed."

2.2.2 OR Gate

An OR gate takes two inputs and is "on" so long as at least one of the inputs is "on." The switch diagram, logic gate representation, and truth table for an OR gate is given below. Note





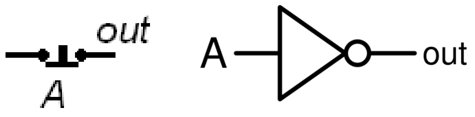
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Figure 2.5: OR: switch diagram, logic gate, and truth table.

that an OR gate is 1 ("on") if and only if at least one of its inputs is 1, and note how this is realized physically with switches.

2.2.3 NOT Gate

The final basic logic gate is the NOT gate. Unlike the AND and OR gates, the NOT gate has only one input, and its output is simply the opposite of its input. The switch diagram, logic gate representation, and truth table for a NOT gate is given below. Note that in the switch diagram, the switch is of the "normally closed" variety; pushing the switch breaks the connection in this case.



A	NOT A
0	1
1	0

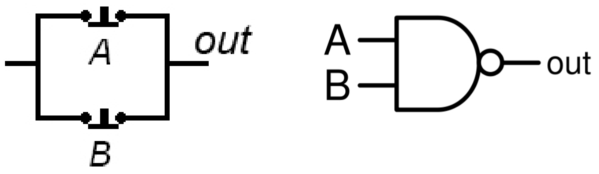
Figure 2.6: NOT: switch diagram, logic gate, and truth table.

2.3 Other Logic Gates: NAND, NOR, XOR, XNOR

As we shall learn in the next chapter, every conceivable truth table and its corresponding logic gate can be realized using combinations of AND, OR, and NOT gates. However, some truth tables are so common that they have their own dedicated logic gate representations; four such logic gates are described below.

2.3.1 NAND Gate

The NAND gate is the opposite of an AND gate: it is 1 (on) if and only if it is *not* the case that both of its inputs are 1. A NAND gate can be constructed from an AND gate whose output is attached to a NOT gate. The switch diagram, logic gate representation, and truth table for a NAND gate is given below. The NAND gate has two interesting properties: (1) It is



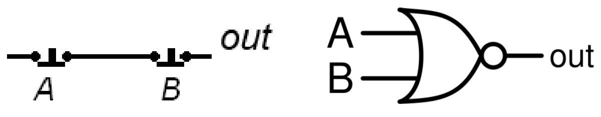
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

Figure 2.7: NAND: switch diagram, logic gate, and truth table. Note the use of normally closed switches.

the simplest logic gate to construct from common electrical components (transistors, resistors, wires, etc.) or to fabricate as part of an integrated circuit. (2) The NAND gate is "logically complete" in that every conceivable truth table, logic gate, or circuit can be constructed solely from NAND gates.

2.3.2 NOR Gate

The NOR gate is the opposite of an OR gate: it is 1 (on) if and only if it is *not* the case that at least one of its inputs 1. A NOR gate can be constructed from an OR gate whose output is attached to a NOT gate. The switch diagram, logic gate representation, and truth table for a NOR gate is given below.

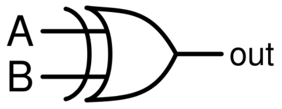


A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

Figure 2.8: NOR: switch diagram, logic gate, and truth table. Note the use of normally closed switches.

2.3.3 XOR Gate

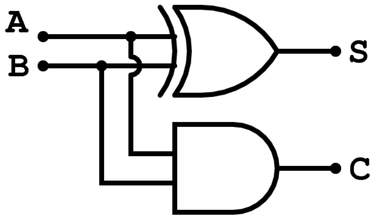
The XOR gate is the "exclusive OR" gate; it is 1 (on) if and only if one input is 1, but not both. The logic gate representation and truth table for a XOR gate is given below. The XOR gate



A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Figure 2.9: XOR: logic gate and truth table.

is very useful in implementing binary arithmetic. Consider adding two binary digits: if both bits are 0, the sum is 0; if one of the bits is 0 and the other bit is 1, the sum is 1; and if both bits are 1, the sum is 2, or in binary, 10. Note that the XOR gate gives the proper output of the *least significant bit* in adding two bits, and further note that an AND gate gives the proper output of the *most significant bit* (or carry) in adding two bits. Such a simple circuit is called a *half adder*; see the figure below. In later sections, we will see how logic gates can be used to



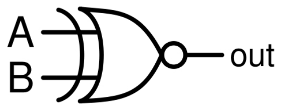
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Figure 2.10: Half adder circuit and truth table.

perform arithmetic on arbitrarily long binary numbers.

2.3.4 XNOR Gate

The XNOR gate is the "exclusive NOR" gate; it is the opposite of the XOR gate, and can be constructed by an XOR gate whose output is attached to a NOT gate. The XNOR gate is 1 (on) if and only if both of its inputs are identical (i.e., both 1 or both 0). The XNOR gate is used to test if its inputs are identical, and as a consequence, it is often referred to as the "equivalence gate."



A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

Figure 2.11: XNOR: logic gate and truth table.

2.4 Binary Arithmetic: Ripple Carry Adders

As we saw in the previous chapter, in order to perform the addition of two binary numbers, one must in each column sum the corresponding bits from each input number together with any input carry bit, producing an output bit and possibly a carry bit. Letting A, B, and C_i denote the first and second input bits and the input carry bit, and letting S and C_o denote the output sum and carry bit, the following truth table shown in Figure 2.12 represents the required action for a circuit dealing with one column of binary addition; a circuit implementing this truth table is shown in Figure 2.13

Stringing together a series of full adders, one for each column binary addition, yields a *ripple carry adder* as shown in Figure 2.14.

A	B	C_i	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 2.12: Full truth table.

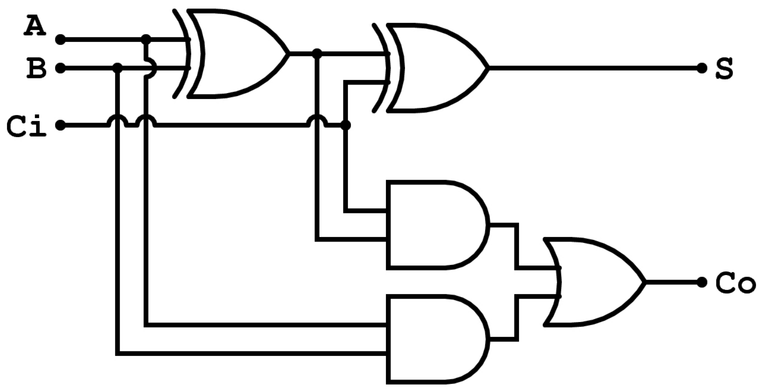


Figure 2.13: Circuit implementing a full adder

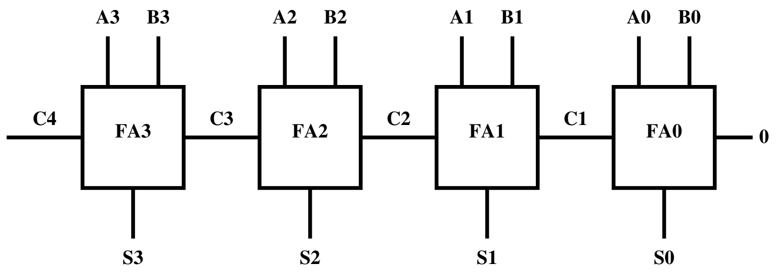


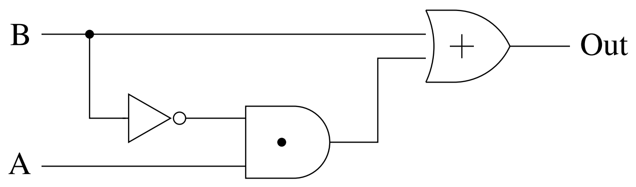
Figure 2.14: Ripple carry adder for four-bit addition. Here, FA$_i$ represents a full adder for column i, and note the use of a 0 for the initial “carry in” to column 0.

Exercises

Exercise 2.1

Converting circuits to truth tables.

a. Convert the following circuit to its equivalent truth table.



b. What logical operation does this circuit compute?

Exercise 2.2

Convert the following truth table to an equivalent circuit.

A	B	C	Out
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	0
1	1	1	1