CS-160 | Lab #2: Gates, Circuits and Binary Representations

General

Read through the "Background" material below, then download the Lab #2 question set and answer the questions. Turn in the questions using the instructions posted on the class web site.

For ALL Word processing documents, you must submit your documents in one of the following formats: MS-Word (NOT Works), RTF (most word processors can save in this format), or Open Document (used by the freely available Open Office suite). They will be returned ungraded if submitted in any other format.

Concepts

The purpose of this lab is to give you an introduction to some of the most fundamental operations of computational systems: the basic electronics of logic gates and combinatorial circuits and an introduction to binary representations.

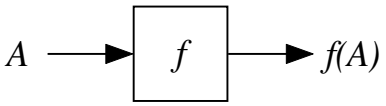
Background

Note: Original source of the background contained below is from the "CS160 Worksheets" by Daniel Balls of the CS department at Oregon State University; updated and revised by Mitch Fry (CS, Chemeketa Community College).

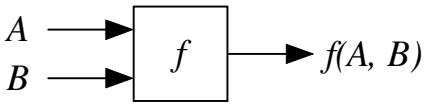
Logic Gates

In order to add, subtract, and compare numbers, computers manipulate 0s and 1s. Operations that manipulate individual 0s and 1s are called Boolean operations. A *logic gate* is a mechanism that performs a Boolean operation by taking one or more 0s and 1s as inputs and producing a 0 or 1 as output.

The Boolean operations we will study are either unary or binary. A unary operation has a single input, and a binary operation has two inputs. Finding the output of a logic gate is simple, once a few principles are understood.

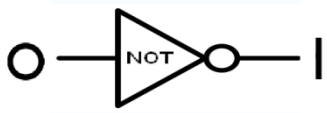


*A **unary** operation (one input).*



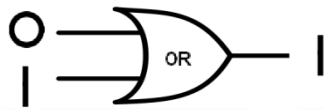
*A **binary** operation (two inputs).*

The only unary Boolean operation is the NOT function. The NOT gate returns the complement of the operand. An input of 1 results in an output of 0; conversely, an input of 0 results in an output of 1. The latter of the two possibilities for the NOT operation is shown below.



*The **NOT** gate returns the complement of the input.*

The most common binary Boolean operations include AND, OR, and XOR (short for 'exclusive OR'). If *at least* one of the two operands is 1, the OR operation gives an output of 1. The output of the XOR operation is 1 when *exactly* one of the operands is 1. The AND operation returns 1 only if both inputs are 1. There are four possible combinations of input pairs for *each* of these three gates. One of these combinations for each gate is shown below.

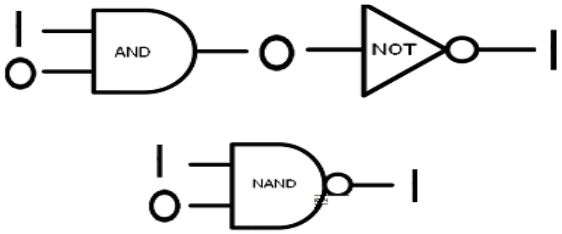


*The **OR** gate returns 1 when at least one of its inputs is 1.*

*The **XOR** gate returns 0 when both operands are 1 or both operands are 0.*

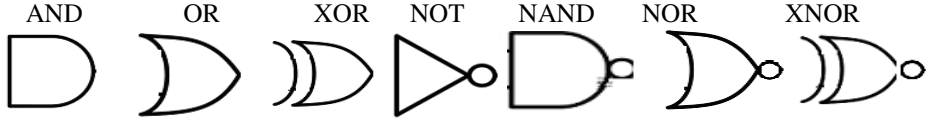
*When at least one of the operands is 0, the output of the **AND** gate is 0.*

Other logical gates can be formed by appending the NOT gate to any of the binary gates. These gates are denoted by NAND (NOT AND), NOR (NOT OR), and XNOR (NOT XOR). The output of these gates can be found by first determining the output of the binary gate and then using this output as the input of the NOT gate, as shown below.



*The **NAND** gate is equivalent to an **AND** gate followed by a **NOT** gate.*

The symbols that we have used above for the different logic gates are easily distinguishable and widely used, with one difference: the name of the gate is not written inside the symbol. Henceforth we will follow that practice. The symbols for the seven logic gates are as follows:



These are the symbols for the seven logic gates presented in this worksheet.

One way to represent a Boolean function is through the use of a truth table. The first column(s) give all possible input combinations. The values in the last column(s) represent the output of the function given the inputs that precede it. Such tables are called truth tables. An example of the truth tables for the NOT and OR operations are shown below (I_1 stands for the first input, I_2 stands for the second input and **O** represents the output):

I_0	O
0	1
1	0

*Truth table for the **NOT** gate.*

I_0	I_1	O
0	0	0
0	1	1
1	0	1
1	1	1

*Truth table for the **OR** gate.*

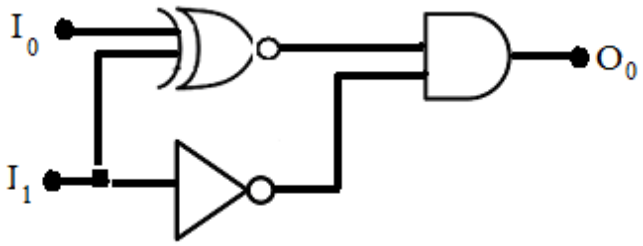
An applet that will help you become more familiar with the basic logic gates can be accessed at

http://newterra.chemeketa.edu/faculty/mfry5/cs160/applets/LogicGates.html.

1. Make sure the option *1 Gate* is selected.
2. Double-click on the rectangle and choose one of the logic gates available.
3. Next change the values of the inputs by clicking on the values.
4. Repeat steps 2 and 3 for all the logic gates.
5. Notice than when gate is selected, a table appears to the right. This is a truth table for the selected gate. Once inputs have been selected, the row that corresponds to these inputs is highlighted.

Combinatorial Circuits

An arrangement of more than one logic gate is called a combinatorial circuit. The gates in a circuit are linked. Often it is the case that one gate has an output that serves as the input for one or more gates within the circuit. The use of logical gates and combinatorial circuits play a fundamental role in how computers operate. An example of a combinatorial circuit is shown below. Notice that in this circuit the lower input serves as the input for the NOT gate as well as the second input of the XNOR gate. Note that we normally label Inputs and Outputs from 0,1,2.. etc. from top to bottom; and generally try to show inputs on the left, outputs on the right.



An example of a combinatorial circuit.

Use the applet at:

http://newterra.chemeketa.edu/faculty/mfry5/cs160/applets/LogicGates.html

to help you understand more effectively the ideas behind combinatorial circuits.

1. Select the option labeled *2 Gates*.
2. Double click on each rectangle and select a logic gate for each one.
3. Change the inputs to investigate the nature of the circuit.
4. Create a different circuit by selecting different gates and investigate its nature. Notice how the truth table changes as the logic gates are changed.
5. Select the options for *3 Gates (2 inputs)*, *3 Gates (4 inputs)*, and *4 Gates*, and repeat steps 2 – 4.

Binary Number Systems

Digital computers store information in the form of 0s and 1s. An important use of digital computers is performing computations. That means we must be able to store numbers as patterns of 0s and 1s. The *binary number system* is a way of representing integers using only 0s and 1s.

Before we explore the binary system, it will be helpful to review the number system we are more familiar with—the decimal system. The decimal, or base-ten, number system uses ten digits—0 through 9. The position of each digit in a number is associated with a specific power of ten. The rightmost position in a decimal representation has a value of 10^0 (or 1), followed on the left by 10^1 (or 10), then 10^2 (or 100), then 10^3 (or 1000), and so on. For the number represented in decimal form by 247, the digit 7 is associated with 10^0, the 4 with 10^1, and the 2 with 10^2. The value of the representation (e.g., 247) can be found by multiplying each digit by its position's value: the representation 247 has the value $(2 \cdot 10^2 + 7 \cdot 10^1 + 4 \cdot 10^0)$.

$$\underset{10^2}{\mathbf{2}} \quad \underset{10^1}{\mathbf{4}} \quad \underset{10^0}{\mathbf{7}} = \mathbf{2}*100 + \mathbf{4}*10 + \mathbf{7}*1$$

The value of the decimal representation 247.

The binary number system has a structure similar to the decimal system. In the binary, or base-two, system there are only 2 digits—0 and 1. The position of each bit (binary digit) is associated with a specific power of 2—2^0, 2^1, 2^2, 2^3, etc. (moving from right to left). For example, the decimal equivalent of the binary representation 1101 can be found by multiplying each digit by its position value: $1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 13$. Therefore, the binary number 1011 is equivalent to the decimal number 13.

binary *decimal*

$$\mathbf{1101} = \underset{2^3}{\mathbf{1}} \quad \underset{2^2}{\mathbf{1}} \quad \underset{2^1}{0} \quad \underset{2^0}{\mathbf{1}} = \mathbf{1}*8 + \mathbf{1}*4 + \mathbf{0}*2 + \mathbf{1}*1 = \mathbf{13}$$

The binary number 1101 is equivalent to the decimal number 13.

When we express a binary number verbally, it is helpful to say the digits in order from left to right. For example, as opposed to "one thousand one hundred and one," the binary number 1101 would be pronounced "one-one-zero-one."

As a computer science major, you need to become adept with the binary number system. The following activity will help you achieve this goal.

Practice Exercise (for practice): Fill in the following blanks with the binary equivalent of the decimal number given. To make sure you are on track, some of the binary representations have been filled in for you .

Binary	Dec.	Binary	Dec.	Binary	Dec.	Binary	Dec.
	1		26		51		76
	2		27		52		77
	3	**11100**	28		53		78
	4		29		54		79
	5		30		55		80
	6		31		56		81
	7		32		57		82
	8		33		58		83
	9		34		59		84
	10		35		60	**1010101**	85
	11		36		61		86
	12		37		62		87
1101	13		38		63		88
	14		39		64		89
	15		40		65		90
	16		41		66		91
	17		42		67		92
	18		43		68		93
	19		44		69		94
	20		45		70		95
	21		46	**1000111**	71		96
	22		47		72		97
	23		48		73		98
	24		49		74		99
	25		50		75		100

Critical Thinking: (Practice questions, this will help with the assignment)

1) What is the binary representation of the decimal value 17?
2) What is the binary representation of the decimal value 34?
3) What is the binary representation of the decimal value 68?
4) What is the binary representation of the decimal value 22?
5) What is the binary representation of the decimal value 44?
6) What is the binary representation of the decimal value 88?

Converting Between Binary and Decimal Representation

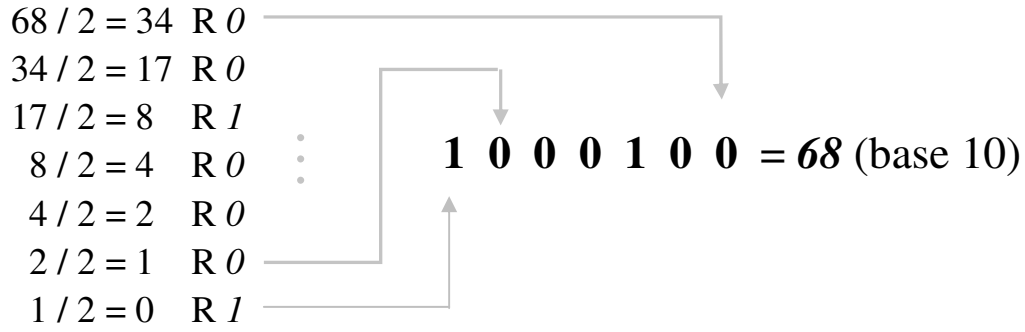
There are two common methods for converting numbers from their decimal to binary representations.

One method of finding the binary equivalent of a decimal number makes use of one's knowledge of the powers of 2—2, 4, 8, 16, 32, 64, 128, etc. The procedure begins by determining the greatest power of 2 less than or equal to the decimal number being converted. For example, when converting 68 into its binary equivalent, we notice that $64 = 2^6$ is the largest power of 2 less than 68. We subtract this power of 2 from the original number, and repeat the process: 68 – 64 = 4, so what is the largest power of 2 less than or equal to 4? The answer is $2^2 = 4$. We stop when the difference between the number and the power of 2 is zero. So we have determined that 68 consists of one 64 and one 4. That means the binary representation of 68 is found by creating a string of bits that has a 1 in the places representing 64 and 4, and a 0 everywhere else: 1000100.

Converting a binary number into its decimal equivalent is related to the method just described. We simply add up the powers of 2 represented by the 1s in the binary representation. For example, the number 1011001 can be converted to a decimal representation as follows:

$$1\cdot2^6 + 0\cdot2^5 + 1\cdot2^4 + 1\cdot2^3 + 0\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 = 64 + 16 + 8 + 1 = 89.$$

An alternative method involves successively dividing by 2. The quotient at each step will always be a whole number with a remainder of 0 or 1 (think about why this is true). If the remainder is recorded but not included in the next step of the division, the process will continue until the quotient is 0. The string of remainders (going from last to first) is the binary representation (written left to right) of the decimal number. An example is shown in the next figure.



The division method for converting decimal numbers into their binary representation is shown here. The remainders are recorded and become the digits of the binary number. This example shows 68 base ten is 1000100 base 2.

Use the applet at:

http://newterra.chemeketa.edu/faculty/mfry5/cs160/applets/NumberFormats.html

to help you clarify your understanding Binary, Octal, Decimal and Hexadecimal number systems and conversions.