Name: MARQUESES, LORENZ BERNARD B. Section:

1.0 Supplementary Activity – Integrating Logic Gates with Programming

Hello Class, during this Supplementary Activity, we would be investigating the different digital logic gates which are fundamental components in a digital electronic system as well as its relationship with our programming language. During the lecture class, we have discussed Boolean Data Types as well as Logic Operations, forming the basis of Logic Gates, which are circuits that serve as basic building blocks of the CPU.

For this Supplementary Activity, we would use these Logic Operations in action, through the use of programming, and verify their output in relation to what we know as the Truth Table. This would enable us to determine if the Logic Operations are valid or work as expected based on what we learned from the lectures. To start, we would need a way to compile C or interpret Python programs through the use of programs or IDEs such as Dev-C++ or Python IDLE. If you do not have your compiler or interpreter installed, you may also opt to use an online IDE such as REPL (https://replit.com/) You may choose to use either of the programming languages, based on your preference, no need to use both.

1.1 Procedure

**** README ****

Note that since we are dealing with Boolean Data Types, for all intents and purposes, we would be following with the idea that the value of **True = 1** and the value of **False = 0**.

In addition, note that the provided lines of code for either programming language would contain the value ***<Input>*** which is *meant to be replaced with the value* of the **Input** under the **Truth Table**. This would mean that if the value of **Input** is 0, then replace ***<Input>*** in the program with the value of 0 before running the program, then record the result in the corresponding **Output** (same goes any other ***<Input>*** or Output).

1. Open the compiler/interpreter/IDE of your preferred programming language then insert the following lines of code below. Replace the value of ***<Input>*** in the program to the value under the **Input** of the **Truth Table**, then run the program. Complete the **Truth Table** below and determine the **Output**.

Program in C

```c
#include <stdio.h>

int main(void) {
  int A = <Input>;
  if(A)
    printf("1");
  else
    printf("0");
  return 0;
}
```

Program in Python 3

```python
A = <Input>

if A:
  print("1")
else:
  print("0")
```

Truth Table

Input	Output
0	0
1	1

The condition in the `if else` statement that is used in this expression is to just provide the input variable. This Logic Operations represent the **Buffer Gate** or also known as the Delay Gate, which has one input and its output follows the same Logic State or value as the input. In Digital Electronics, the Buffer Gate is typically used to introduce a delay element or provide a current boost-up, but is generally not used as much in programming.

2. Let's try another Logic Operation, insert the following lines of code below and remember to replace the value of *<Input>* in the program to the value under the **Input** of the **Truth Table,** before running the program. Complete the **Truth Table** below and determine the **Output**.

Program in C

```c
#include <stdio.h>

int main(void) {
  int A = <Input>;
  if(!A)
    printf("1");
  else
    printf("0");
  return 0;
}
```

Program in Python 3

```python
A = <Input>

if not A:
  print("1")
else:
  print("0")
```

Logical NOT Truth Table

Input	Output
0	1
1	0

The condition in the `if else` statement that is used in this expression is the **Logical NOT** which is also known as **Negation** and is represented by the exclamation symbol (`!`) or the `not` expression. This Logic Operation represents the **NOT Gate**, which implements the logical negation returning the value of **True** (**1**) if the input is **False** (**0**) then **False** (**0**) if the input is **True** (**1**).

3. This time, let's try to use two (2) input Logic Operations, insert the following lines of code below and once again, remember to replace the value of *<Input A>* in the program to the value under **Input A** of the **Truth Table**, then *<Input B>* in the program to the value under **Input B** of the **Truth Table**, before running the program. Complete the **Truth Table** below and determine the **Output**.

Program in C

```c
#include <stdio.h>

int main(void) {
  int A = <Input A>;
  int B = <Input B>;
  if(A && B)
    printf("1");
  else
    printf("0");
  return 0;
}
```

Program in Python 3

```python
A = <Input A>
B = <Input B>

if A and B:
  print("1")
else:
  print("0")
```

Logical AND Truth Table

Input A	Input B	Output
0	0	0
0	1	0
1	0	0
1	1	1

The condition in the `if else` statement that is used in this expression is the **Logical AND** which is represented by the double ampersand symbol (`&&`) or the `and` expression. This Logic Operation represents the **AND Gate**, which implements the Logical AND**,** returning the value of **True** (**1**) only if <u>all of its inputs</u> are **True** (**1**) then **False** (**0**) if <u>at least one of its inputs</u> are **False** (**0**).

4. Let's try another two (2) input Logic Operation, insert the following lines of code below and remember to replace the value of *<Input A>* and *<Input B>* to the respective **Input** on the **Truth Table**, before running the program. Complete the **Truth Table** below and determine the **Output**.

Program in C

```c
#include <stdio.h>

int main(void) {
  int A = <Input A>;
  int B = <Input B>;
  if(A || B)
    printf("1");
  else
    printf("0");
  return 0;
}
```

Program in Python 3

```python
A = <Input A>
B = <Input B>

if A or B:
  print("1")
else:
  print("0")
```

Logical OR Truth Table

Input A	Input B	Output
0	0	0
0	1	1
1	0	1
1	1	1

The condition in the `if else` statement that is used in this expression is the **Logical OR** which is represented by the double pipe symbol (`||`) or the `or` expression. If you don't know where the pipe symbol (|) is, it can be found above the return or enter key in the keyboard, for most keyboard layouts. This Logic Operation represents the **OR Gate**, which implements the Logical OR**,** returning the value of **True** (**1**) if <u>at least one of its inputs</u> are **True** (**1**) then **False** (**0**) if <u>all of its inputs</u> are **False** (**0**).

5. You probably got the hang of the instructions already, so once again, let's try another two (2) input Logic Operation, insert the following lines of code below and remember to replace the value of *<Input A>* and *<Input B>* to the respective **Input** on the **Truth Table**, before running the program. Complete the **Truth Table** below and determine the **Output**.

Program in C

```c
#include <stdio.h>

int main(void) {
  int A = <Input A>;
  int B = <Input B>;
  if(A ^ B)
    printf("1");
  else
    printf("0");
  return 0;
}
```

Program in Python 3

```python
A = <Input A>
B = <Input B>

if A ^ B:
  print("1")
else:
  print("0")
```

Logical XOR Truth Table

Input A	Input B	Output
0	0	0
0	1	1
1	0	1
1	1	0

The condition in the `if else` statement that is used in this expression is the **Logical XOR** which is also known as **Exclusive OR** and is represented by the caret or circumflex symbol (`^`) or the ⊕ mathematical expression (a circle with a plus sign inside). If you don't know where the caret symbol (^) is, it is generally typed using **shift+6** in the keyboard, for most keyboard layouts. This Logic Operation represents the **XOR Gate**, which implements the Logical XOR or Logical Inequality (NEQ)**,** returning the value of **True** (**1**) if <u>one and only one of its inputs</u> are **True** (**1**) then **False** (**0**) if otherwise (can be interpreted from the IEEE Standard 91-1984).

6. How about another we try another two (2) input Logic Operation, insert the following lines of code below and remember to replace the value of ***<Input A>*** and ***<Input B>*** to the respective **Input** on the **Truth Table**, before running the program. Complete the **Truth Table** below and determine the **Output**.

Program in C

```c
#include <stdio.h>

int main(void) {
  int A = <Input A>;
  int B = <Input B>;
  if(!(A && B))
    printf("1");
  else
    printf("0");
  return 0;
}
```

Program in Python 3

```python
A = <Input A>
B = <Input B>

if not (A and B):
  print("1")
else:
  print("0")
```

Logical NAND Truth Table

Input A	Input B	Output
0	0	1
0	1	1
1	0	1
1	1	0

The condition in the `if else` statement that is used in this expression is the **Logical NAND** which is represented by a combination of the exclamation symbol (`!`) or the `not` symbol before the double ampersand (`&&`) or the `and` expression. This Logic Operation represents the **NAND Gate**, which implements the Logical NAND**,** returning the value of **True** (**1**) if at least one of its inputs are **False** (**0**) then **False** (**0**) only if all of its inputs are **True** (**1**).

7. But wait, there's more! Actually there are still other Logic Operations such as **NOR** and **XNOR**, which are a combination of a NOT and OR as well as a NOT and XOR Gates, respectively. However, that much may already be too simple for you, so we won't be programming them. Instead, how about lets see some practical applications for our Logic Operations.

You know the drill, insert the following lines of code below and remember to replace the value of ***<Input A>*** and ***<Input B>*** to the respective **Input** on the **Truth Table**, before running the program. Complete the **Truth Table** below and determine the **Output C** and **Output S**. Note that the output is a 2-digit combination of **Output C** and **Output S**, where the digit on the left would be **Output C** and the digit on the right is **Output S**.

Program in C

```c
#include <stdio.h>

int main(void) {
  int A = <Input A>;
  int B = <Input B>;
  // Output C
  if(A && B)
    printf("1");
  else
    printf("0");

  // Output S
  if(A ^ B)
    printf("1");
  else
    printf("0");
  return 0;
}
```

Program in Python 3

```python
A = <Input A>
B = <Input B>

# Output C
if A and B:
  print("1", end='')
else:
  print("0", end='')

# Output S
if A ^ B:
  print("1", end='')
else:
  print("0", end='')
```

Truth Table

Input A	Input B	Output C	Output S
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

*Note that the `end=''` argument of print in the Python 3 Program just removes the newline when using the `print()` statement

What did you think was the output of the previous program?

Output S is the bitwise binary sum of A and B, with the carry being expressed as the output C. This is known as a half-adder.

If it wasn't too obvious, you actually were able to create a **Half Adder** using Logic Operations. **Input A** is the **Augend** and **Input B** is the **Addend**, while the **Output S** is your **Sum** and **Output C** is your **Carry**. This means that you were able to do Arithmetic through the use of Logic! Now isn't that something?

Just imagine that computers actually use combinations of multiple different kinds of Logic Gates and Logic Operations and in different sequences, having varying inputs and outputs, to do the complex functions that our modern computers perform today.

1.2 Synthesis

With all the tasks given to you for this activity, can you summarize your learnings and findings by providing a Conclusion. The Conclusion may discuss some realizations on the relationships of Logic Operations and Logic Gates in both Hardware and Software as well as the role of Logic Operations in a computer. Use the space provided below:

Logic operations are the foundation on which all computer processes rest. In hardware, it enables the construction of the more complex systems that one would recognize as a modern computer. This is thanks to the fundamental concepts of logic gates being both incredibly simple and, by the nature of the binary number system, correspondent with physical electrical signals through states of "true" and "false." In software, on the other hand, logic remains a key component of algorithm design that dictates how programs should handle specific instructions. For instance, operators such as && or || emulate logic gates in how they take inputs and return an output.

After completing the activity, don't forget to submit the completed manual in the respective assignment in AnimoSpace.

Have fun!