**DIGITAL ELECTRONICS 2 LAB ASSIGNMENT 1**

Name: Demirkan Korbey Baglamac

1. **My Digital Electronics 2 repository:** <https://github.com/dkorbey/Digital-electronics-2>
3. **| :** This symbol represents the OR operator.

|  |  |  |
| --- | --- | --- |
| A | B | A|B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

As you can see from the truth table, it is enough if one of the entries are 1, for OR operator to return 1.

**Example:**

#include <stdio.h>

int main() {

    int a = 5; // Binary Representation: 0000 0101

    int b = 3; // Binary Representation: 0000 0011

    printf("%d", a | b);

    return 0;

}

When you run this example code you will see that the output is 7. Let’s prove it,

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Bit8** | **Bit7** | **Bit6** | **Bit5** | **Bit4** | **Bit3** | **Bit2** | **Bit1** | **Result** |
| **a** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 5 |
| **b** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 3 |
| **a|b** | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 7 |

Note: **int** is normaly 32bit in C, but for an easy representation I assume 8bits.

1. **& :** This symbol represents the AND operator.

|  |  |  |
| --- | --- | --- |
| A | B | A|B |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

As you can see from the truth table, for AND operator to return 1 all two input must be 1.

**Example:**

#include <stdio.h>

int main() {

    int a = 5; // Binary Representation: 0000 0101

    int b = 3; // Binary Representation: 0000 0011

    printf("%d", a & b);

    return 0;

}

When you run this example code you will see that the output is 7. Let’s prove it,

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Bit8** | **Bit7** | **Bit6** | **Bit5** | **Bit4** | **Bit3** | **Bit2** | **Bit1** | **Result** |
| **a** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 5 |
| **b** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 3 |
| **a&b** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

1. **^ :** This symbol represents the XOR operator.

|  |  |  |
| --- | --- | --- |
| A | B | A^B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

XOR operator got his name from the expression Exclusive OR and exclusive means that we exclude the possibility of both inputs are 1.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Bit8** | **Bit7** | **Bit6** | **Bit5** | **Bit4** | **Bit3** | **Bit2** | **Bit1** | **Result** |
| **a** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 5 |
| **b** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 3 |
| **a^b** | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 6 |

So if we run the same code we use in the OR operation with XOR we get,

1. **~ :** This symbol represents the NOT operator.

|  |  |
| --- | --- |
| A | ~A |
| 0 | 1 |
| 1 | 0 |

NOT operator needs only one input and as you can understand from the name, it inverts the input value.

**Example:** If we have a 8-bit binary number C = 1001 0011, then ~C = 0110 1100.

1. **<< :** This symbol represents the Left Shift operator.

This operator’s syntax is, **X << N** where X is the number which we will apply the shifting and N is the number for how many bits we will shift the binary representation of the X.

And when we shift the binary number the emptied bits will be automatically 0.

**Example:** Lets assume that A = 0001 1101 = 29(Base-10);

**A<<3 :** 1110 1000 = 232(Base-10)

**29<<1 :** 58(Base-10) = 0011 1010