DIGITAL ELECTRONICS 2 LAB ASSIGNMENT 1

Name: Demirkan Korbey Baglamac

1) **My Digital Electronics 2 repository:** https://github.com/dkorbey/Digital-electronics-2

2)

a) **|** : 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:

```c
#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.

b) & : 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:

```c
#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 1. 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

c) ^ : 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.

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

	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

d) **~ :** 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.

e) **<< :** 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

3) The main.c code for the morse code representation of DE2,

```
/*
 * DE2_MorseCode.c
 *
 * Created: 28.9.2020 15:14:47
 * Author : dkorb
 */

/* Defines -----------------------------------------------------------
--*/
#define LED_GREEN   PB5     // AVR pin where green LED is connected
#define SHORT_DELAY 500      // Delay in ms
#define LONG_DELAY  1000


#ifndef F_CPU
#define F_CPU 16000000      // CPU frequency in Hz required for dela
y func
#endif
```

```c
/* Includes ------------------------------------------------------------
--*/
#include <util/delay.h>     // Functions for busy-wait delay loops
#include <avr/io.h>         // AVR device-specific IO definitions

/* Variables -----------------------------------------------------------
--*/

/* Function prototypes -------------------------------------------------
--*/

void shortDot();
void longDot();

int main(void)
{

    // Set pin as output in Data Direction Register
    // DDRB = DDRB or 0010 0000
    DDRB = DDRB | (1<<LED_GREEN);

    // Set pin LOW in Data Register (LED off)
    // PORTB = PORTB and 1101 1111
    PORTB = PORTB & ~(1<<LED_GREEN);

    /* Replace with your application code */
    while (1)
    {

        // Start delay ms
        _delay_ms(LONG_DELAY);

        //D in morse code
        longDot();
        shortDot();
        shortDot();

        //Interval delay
        _delay_ms(SHORT_DELAY);

        //E
        shortDot();

        //Interval Delay
        _delay_ms(SHORT_DELAY);

        //2
        shortDot();
```

```
        shortDot();
        longDot();
        longDot();
        longDot();

    }

    return 0;
}

/* Functions ---------------------------------------------------------
--*/

void shortDot() {

    // Start delay ms
    _delay_ms(SHORT_DELAY);

    // Invert LED in Data Register
    // PORTB = PORTB xor 0010 0000
    PORTB = PORTB ^ (1<<LED_GREEN);

    _delay_ms(SHORT_DELAY);

    PORTB = PORTB ^ (1<<LED_GREEN);
}

void longDot(){

    // Start delay ms
    _delay_ms(SHORT_DELAY);

    // Invert LED in Data Register
    // PORTB = PORTB xor 0010 0000
    PORTB = PORTB ^ (1<<LED_GREEN);

    // Start delay ms
    _delay_ms(LONG_DELAY);

    PORTB = PORTB ^ (1<<LED_GREEN);
}
```