

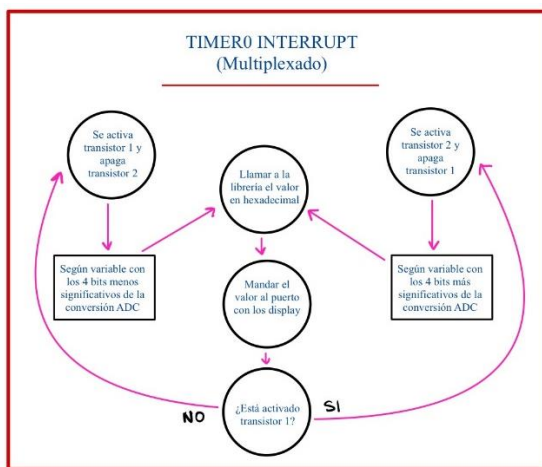
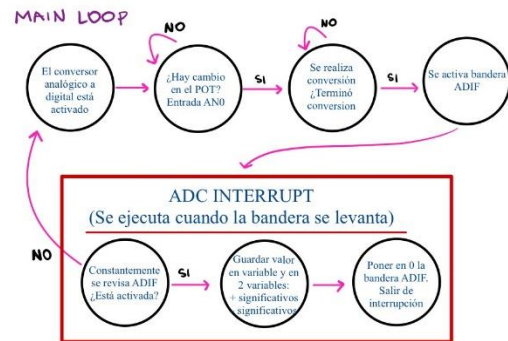
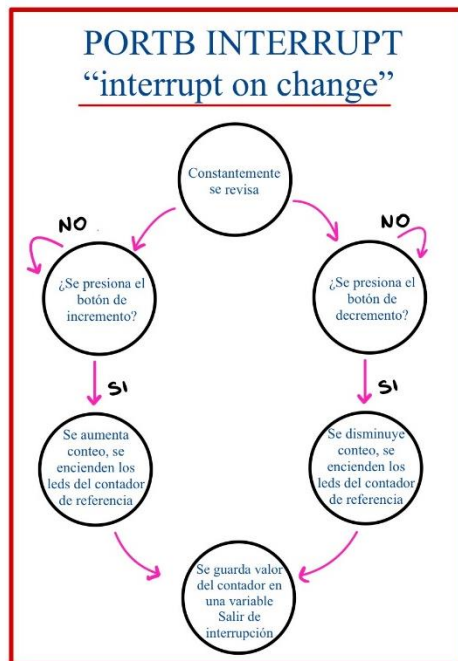
Laboratorio # 2

Interrupciones y uso de librerías

Link del video:

https://youtu.be/caB_ga2O2kl

Pseudocódigo



Código

```
/*
 * Laboratorio # 2
 * Author: Natalia de León Bercián
 * carné: 18193
 * Digital 2
 *
 * Created on 29 de enero de 2021
 */

//*****//
//IMPORTAR LIBRERIAS                                //
//*****//

#include <xc.h>
#include <stdint.h>
#include "Oscilador.h"
#include "Config_ADC.h"
#include "Display.h"

//*****//
//CONFIGURACION BITS                                //
//*****//

// CONFIG1

#pragma config FOSC = INTRC_NOCLKOUT// Oscillator Selection bits (INTOSCIO oscillator: I/O function on RA6/OSC2/CLKOUT pin, I/O function on RA7/OSC1/CLKIN)

#pragma config WDTE = OFF    // Watchdog Timer Enable bit (WDT disabled and can be enabled by SWDTEN bit of the WDTCON register)

#pragma config PWRTE = OFF    // Power-up Timer Enable bit (PWRT disabled)

#pragma config MCLRE = OFF    // RE3/MCLR pin function select bit (RE3/MCLR pin function is digital input, MCLR internally tied to VDD)
```

```
#pragma config CP = OFF    // Code Protection bit (Program memory code protection is disabled)
#pragma config CPD = OFF   // Data Code Protection bit (Data memory code protection is disabled)
#pragma config BOREN = OFF  // Brown Out Reset Selection bits (BOR disabled)
#pragma config IESO = OFF   // Internal External Switchover bit (Internal/External Switchover mode is disabled)
#pragma config FCMEN = OFF  // Fail-Safe Clock Monitor Enabled bit (Fail-Safe Clock Monitor is disabled)
#pragma config LVP = OFF    // Low Voltage Programming Enable bit (RB3 pin has digital I/O, HV on MCLR must be used for programming)
```

```
// CONFIG2
```

```
#pragma config BOR4V = BOR40V // Brown-out Reset Selection bit (Brown-out Reset set to 4.0V)
#pragma config WRT = OFF      // Flash Program Memory Self Write Enable bits (Write protection off)
```

```
//**************************************************************************
```

```
//DEFINE                                     //
```

```
//**************************************************************************
```

```
#define _XTAL_FREQ 4000000
```

```
//**************************************************************************
```

```
//VARIABLES                                   //
```

```
//**************************************************************************
```

```
uint8_t contador = 0; //Variable de incremento para contador
```

```
uint8_t debouncing1 = 0; //Variable que controla debouncing de un push
```

```
uint8_t debouncing2 = 0; //Variable que controla debouncing del otro push
```

```
uint8_t toggle = 0;
```

```
int ADC_NIBBLE1;
```

```
int ADC_NIBBLE2;
```

```
int ADC_VALOR;
```

```
int ADC_SWAP;
```

```

//*****//

//PROTOTOPOS DE FUNCIONES                                     //

//*****//

void setup(void);

void Config_INTERRUPT(void);

void CONVERSION_ADC(void);

void TOGGLE_1(void);

void DisplayADC(void);

void Revision(void);


//*****//

//INTERRUPCIONES                                             //

//*****//

void __interrupt() ISR(void) {

    // ---- Interrupción del PUERTO B ----

    if (INTCONbits.RBIF == 1){ // Interrupcion on change

        if (PORTBbits.RB0 == 1){ //debouncing

            debouncing1 = 1;

            contador = contador;

        }

        if (PORTBbits.RB1 == 1){ //debouncing

            debouncing2 = 1;

            contador = contador;

        }

        if(PORTBbits.RB0 == 0 && debouncing1 == 1){ //hasta revisar bandera...

            contador++;           // de deboucing y que el boton no...

            PORTC = contador;     //este presionado, se aumenta o...

            debouncing1 = 0;      //decrementa.

        }

    }
}

```

```

        if(PORTBbits.RB1 == 0 && debouncing2 == 1){

            contador--;

            PORTC = contador;

            debouncing2 = 0;

        }

        INTCONbits.RBIF = 0; //limpiar bandera

    }

// ---- Interrupción del ADC ----
if (PIR1bits.ADIF) {

    PIR1bits.ADIF = 0;

    __delay_ms(2); //Inicio de conversion ADC

    ADCON0bits.GO = 1;

    while (ADCON0bits.GO != 0) { //Mientras no se haya terminado una convers.

        ADC_VALOR = ADC(ADRESL, ADRESH);

        Revision(); //revisar si el valor del ADC es mayor al contador

        DisplayADC();

    }

}

// ---- Interrupción del TMR0 ----
if (INTCONbits.TMR0IF == 1) {

    INTCONbits.TMR0IF = 0;

    TMR0 = 6; //valor que se le agrega para que ocurra cada 1000 ns

    Revision();

    TOGGLE_1(); //toggle de variables cada 1000 ns, para encender transistores

}

}

//*****//

//PROGRAMACION PRINCIPAL                                     //
```

```

//*****//
void main(void) {
    PORTEbits.RE0 = 0;

    setup(); //Configuracion de puertos de entrada y salida
    Config_INTERRUPT(); //Configuracion de la interrupcion del puerto B

    //*****//

    //LOOP PRINCIPAL                                //

    //*****//

    while (1) {
        CONVERSION_ADC(); //Separacion de Nibbles
    }

    return;
}

//*****//

//FUNCIONES                                //

//*****//

void TOGGLE_1(void) { //toggle para luego encender los transistores

    if (toggle == 1) {
        toggle = 0;
    } else if (toggle == 0) {
        toggle = 1;
    }
}

void DisplayADC(void) { //segun la variable de toggle se enciende cada transistor

    PORTE = 0;

    if (toggle == 0) {
        PORTEbits.RE1 = 1;
    }
}

```

```

        PORTD = display(ADC_NIBBLE2); //se busca en la libreria el valor en HEX
    } else if (toggle == 1) {
        PORTEbits.RE2 = 1;
        PORTD = display(ADC_NIBBLE1);
    }
}

void CONVERSION_ADC(void) {
    ADC_SWAP = SWAP_ADC(ADC_VALOR); // swap de los nibbles para tener los valores
                                     // de los 'msb'
    ADC_NIBBLE1 = NIBBLE1_ADC(ADC_VALOR); // AND con 0b00001111 para dejar
    ADC_NIBBLE2 = NIBBLE2_ADC(ADC_SWAP); //solo el los 'lsb' en ambas variables
}

void Revision(void){
    if (ADC_VALOR > contador) {
        PORTEbits.RE0 = 1;

    }
    else if (ADC_VALOR < contador){
        PORTEbits.RE0 = 0;
    }
}

//***** CONFIGURACION PRINCIPAL *****//

void setup(void) { //Configuración de puertos de entrada y salida
    initOsc(0b00000110);
    ANSEL = 0b00000001; //RA0 como analogico
    ANSELH = 0;

```

```

TRISA = 0b00000001; //potenciometro, como entrada

PORTA = 0;

TRISB = 0b00000011; // push, como entradas

PORTB = 0;

PORTC = 0;

TRISC = 0;

TRISD = 0;

PORTD = 0;

TRISE = 0;

PORTE = 0;

}

//***** CONFIGURACION INTERRUPTIONES *****/

void Config_INTERRUPT(void) {

    TMR0 = 6; // Valor que se le agrega al TMR0 para que ocurra cada 1000ns

    OPTION_REG = 0b10001000;

    INTCON = 0b10101001;

    IOCB = 0b00000011;

    PIE1bits.ADIE = 1; // enables ADC interrupt

    PIR1bits.ADIF = 1;

    ADCON1 = 0b00000000;

    ADCON0 = 0b01000001;

}

```

Link de Github