**UNIVERSIDAD NACIONAL DE INGENIERIA (UNI)**

**AREA DE CONOCIMIENTO DE TECNOLOGIA DE LA INFORMACION Y COMUNICACIÓN**

**ARQUITECTURA DE MAQUINAS DE COMPUTADORAS**

**Integrantes: Raymar Atahualpa Cruz Orozco // 2018-0302U**

**Jamie Sicely Rodríguez Sanchez// 2021-0772I**

**Juridia Del Rosario Mayorga Salinas //2022-0365U**

**Jesly Isayana Moraga Pastran //2022-0485U**

**Docente: Juan Carlos Aguirre**

**Grupo: 4E1-COM-S**

**Fecha: 17/05/2025**

ELABORE UNA INFOGRAFIA DE LOS SIGUIENTES PUNTOS.

Fundamentos del Lenguaje Ensamblador.

• Expresiones, operaciones y operadores

• Directivas

• Macroinstrucciones

• Subrutinas

• Organización de un programa en Lenguaje Ensamblador

• Los sistemas informáticos MPLAB y PROTEUS Interrupciones del Lenguaje Ensamblador.

• Interrupciones fijas versus sectorizadas

• Estructura de una subrutina de atención a interrupción



**Organización de un Programa en Ensamblador para MPLAB (Microcontroladores PIC)**

**Ejercicios: Compruebe y logre simular los siguientes puntos.**

1: Encender y apagar un LED cada vez que se presione un botón conectado a RB0. Pista: Usa interrupciones o consulta el estado del pin con BTFSS PORTB, 0.

#include <xc.h>

// CONFIG

#pragma config FOSC = XT // Oscillator Selection bits

#pragma config WDTE = OFF // Watchdog Timer Enable bit

#pragma config PWRTE = ON // Power-up Timer Enable bit

#pragma config CP = OFF // Code Protection bit

#define \_XTAL\_FREQ 4000000 // Frecuencia del oscilador (ajústala si es diferente)

void main(void) {

// Configurar RB0 como entrada, RB1 como salida

TRISB0 = 1; // Botón como entrada

TRISB1 = 0; // LED como salida

// Estado inicial del LED apagado

PORTBbits.RB1 = 0;

// Variable para guardar el estado del LED

char ledEncendido = 0;

while(1) {

// Verifica si el botón está presionado (RB0 en alto)

if (PORTBbits.RB0 == 1) {

\_\_delay\_ms(20); // Pequeño retardo para evitar rebote

if (PORTBbits.RB0 == 1) { // Confirmar que sigue presionado

ledEncendido = !ledEncendido; // Cambia el estado del LED

PORTBbits.RB1 = ledEncendido; // Aplica el nuevo estado

while (PORTBbits.RB0 == 1); // Esperar que se suelte el botón

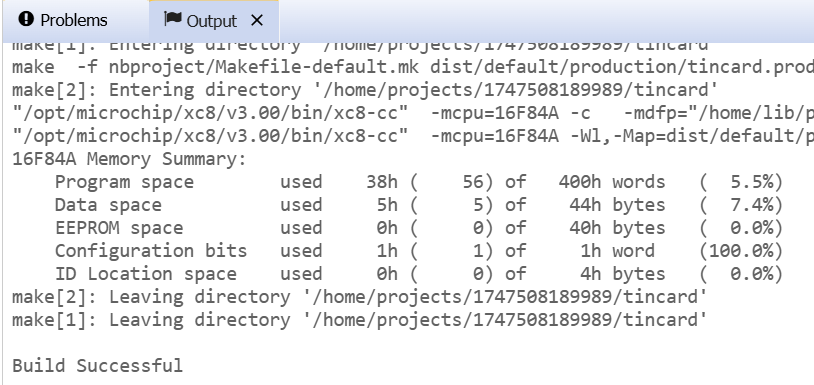
\_\_delay\_ms(20); // Debounce adicional

}

}

}

}



**2: Incrementar un contador en PORTA cada vez que se presione un botón. Objetivo: Aprender a usar registros y operaciones aritméticas.**

#include <xc.h>

// CONFIG

#pragma config FOSC = XT // Oscilador XT

#pragma config WDTE = OFF // Watchdog Timer OFF

#pragma config PWRTE = ON // Power-up Timer ON

#pragma config BOREN = OFF // Brown-out Reset OFF

#pragma config LVP = OFF // Low-Voltage Programming OFF

#pragma config CPD = OFF // Data EEPROM Memory Code Protection OFF

#pragma config WRT = OFF // Flash Program Memory Write Enable OFF

#pragma config CP = OFF // Code Protection OFF

#define \_XTAL\_FREQ 4000000

void main(void) {

// Configuración de TRIS (1 = entrada, 0 = salida)

TRISA = 0x00; // PORTA como salida

TRISB0 = 1; // RB0 como entrada

PORTA = 0x00; // Limpiar PORTA al inicio

unsigned char contador = 0; // Contador de tipo byte (0–255)

while(1) {

if (PORTBbits.RB0 == 1) { // Si el botón está presionado

\_\_delay\_ms(20); // Esperar para evitar rebote

if (PORTBbits.RB0 == 1) { // Confirmar que sigue presionado

contador++; // Incrementar contador

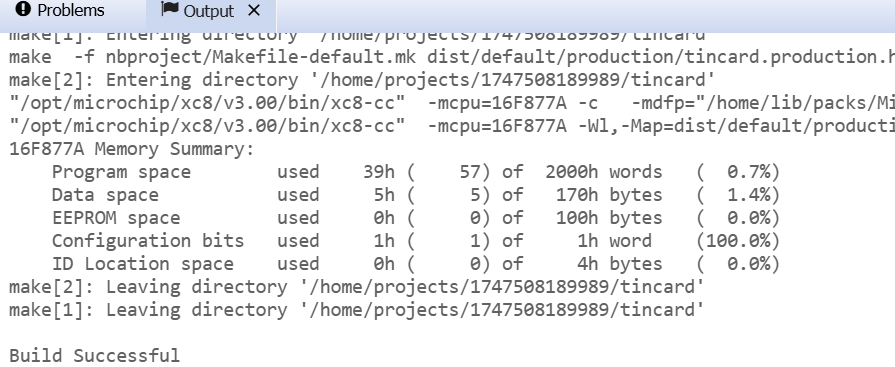
PORTA = contador; // Mostrar valor en LEDs del PORTA

while (PORTBbits.RB0 == 1); // Esperar que se libere

\_\_delay\_ms(20); // Debounce adicional

}

}

 }

}

**3: Enviar un carácter por el puerto serie (USART) cada segundo. Pista: Requiere configurar el módulo USART del PIC (usa un PIC con USART como el 16F877A).**  #include <xc.h>

// CONFIG

#pragma config FOSC = XT // Oscilador XT

#pragma config WDTE = OFF // Watchdog Timer deshabilitado

#pragma config PWRTE = ON // Power-up Timer habilitado

#pragma config BOREN = OFF

#pragma config LVP = OFF

#pragma config CPD = OFF

#pragma config WRT = OFF

#pragma config CP = OFF

#define \_XTAL\_FREQ 4000000 // Frecuencia del oscilador 4 MHz

// --- Función para inicializar el USART ---

void USART\_Init(void) {

// Configurar Baud Rate (9600 bps @ 4MHz)

SPBRG = 25; // Fórmula: SPBRG = (\_XTAL\_FREQ / (64 \* BaudRate)) - 1

// Configuración del transmisor

TXSTAbits.SYNC = 0; // Modo asíncrono

TXSTAbits.BRGH = 0; // Baud rate bajo

RCSTAbits.SPEN = 1; // Activar puerto serie (TX/ RX en RC6 / RC7)

TXSTAbits.TXEN = 1; // Habilitar transmisión

}

// --- Función para enviar un carácter ---

void USART\_Enviar(char dato) {

while (!TXSTAbits.TRMT); // Esperar hasta que el buffer esté libre

TXREG = dato; // Enviar el carácter

}

void main(void) {

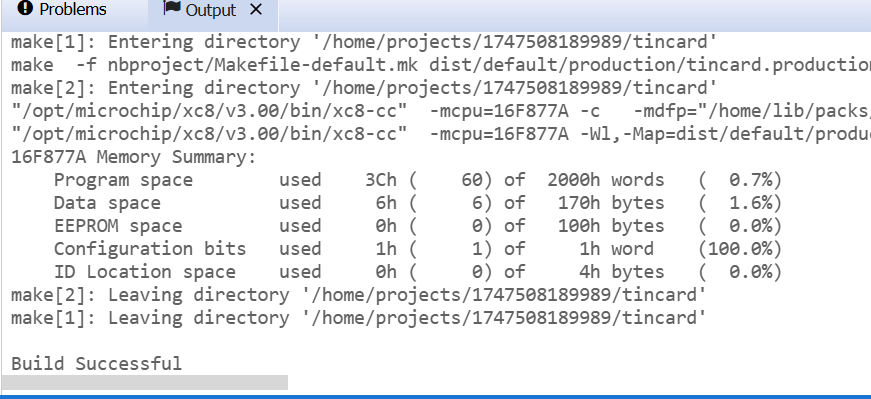
USART\_Init(); // Inicializar USART

while (1) {

USART\_Enviar('A'); // Enviar carácter 'A'

\_\_delay\_ms(1000); // Esperar 1 segundo

}

****}

**CONFIRME SI EL PROGRAMA FUNCIONA (SIMULAR).**

**Encender y apagar un LED conectado a RA0 cada vez que se presiona un botón conectado a RB0 utilizando interrupciones externas**

**Objetivo del Programa**

**CONFIRME SI EL PROGRAMA FUNCIONA (SIMULAR). Objetivo del Programa Microcontrolador utilizado**

**• PIC16F84A (compatible con MPLAB y sencillo para comenzar)**

**• Si tienes otro PIC en mente, dime y ajustamos el código.**

#se emigro a lenguaje C.

#include <xc.h>

// CONFIG

#pragma config FOSC = XT // Oscilador XT

#pragma config WDTE = OFF // Watchdog Timer

#pragma config PWRTE = ON // Power-up Timer

#pragma config CP = OFF // Código no protegido

#define \_XTAL\_FREQ 4000000

void \_\_interrupt() ISR(void) {

if (INTF) {

PORTA ^= 0x01; // Cambiar el bit 0 (toggle RA0)

INTF = 0; // Limpiar bandera de interrupción

}

}

void main(void) {

TRISA = 0x00; // RA0 como salida

PORTA = 0x00; // Apagar todo al inicio

TRISB = 0xFF; // Todo PORTB como entrada

OPTION\_REGbits.INTEDG = 0; // Interrupción en bajada (botón presionado)

INTE = 1; // Habilitar interrupción externa (RB0)

GIE = 1; // Habilitar interrupciones globales

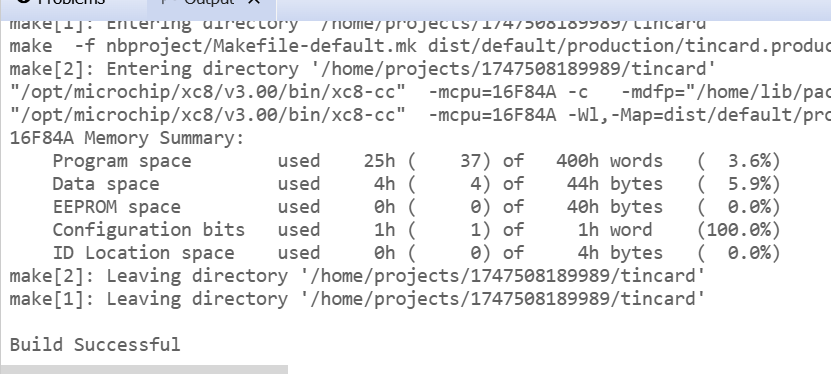
while (1) {

// Esperar la interrupción

}

}

#Salida MPLAB



**Encender y apagar un LED conectado a RA0 cada vez que se presiona un botón conectado a RB0 utilizando interrupciones externas**

**Programa**

const int ledPin = 8; // Pin donde está conectado el LED (equivalente a RA0)

const int buttonPin = 2; // Pin donde está conectado el botón (INT0 en Arduino UNO)

volatile bool ledState = false;

void setup() {

pinMode(ledPin, OUTPUT);

pinMode(buttonPin, INPUT\_PULLUP); // Botón con resistencia interna de pull-up

attachInterrupt(digitalPinToInterrupt(buttonPin), toggleLED, FALLING);

// Interrupción en flanco de bajada cuando se presiona el botón

}

void loop() {

// No es necesario poner nada aquí, ya que todo lo maneja la interrupción

}

void toggleLED() {

ledState = !ledState; // Cambia el estado

digitalWrite(ledPin, ledState);

}

**Salida Arduino**

