Practica 1

Punto 1

Configuración del PIC16F84

Se inicia el código con la configuración del microcontrolador PIC16F84. Se establecen las configuraciones de oscilador, deshabilitando el Watchdog Timer y otras opciones mediante la instrucción `__CONFIG`. La frecuencia del oscilador se fija en 4MHz.

```
__CONFIG(FOSC_HS & WDTE_OFF & PWRTE_OFF & CP_OFF);
#define _XTAL_FREQ 4000000  // Frecuencia del oscilador (4MHz en este ejemplo)
```

Configuración de Puertos

Se configuran los pines de los puertos A y B según las necesidades del programa. RA0 y RA1 se definen como entradas, mientras que RB4 y RB5 se definen como salidas.

```
TRISA0 = 1;  // RA0 como entrada
TRISA1 = 1;  // RA1 como entrada
TRISB4 = 0;  // RB4 como salida
TRISB5 = 0;  // RB5 como salida
```

Inicialización de LEDs

Se inicia el sistema con ambos LEDs encendidos.

```
RB4 = 1;
RB5 = 1;
```

Bucle Principal

El código entra en un bucle principal infinito que verifica constantemente si alguno de los pulsadores (RA0 o RA1) está presionado. En caso de que ambos pulsadores estén en estado alto (no presionados), el programa entra en un bucle donde alterna el estado de los LEDs RB4 y RB5 con un retardo de 250 milisegundos entre cada transición.

```
while (1) {
    // Verificar si alguno de los pulsadores está presionado
    while (RA0 == 1 && RA1 == 1) {

    }

    // Encender y apagar los LEDs de manera alternada
    RB4 = 1;
```

```
    RB5 = 0;
    __delay_ms(250);  // Retardo de 250 ms

    RB4 = 0;
    RB5 = 1;
    __delay_ms(250);  // Retardo de 250 ms
}
```

Punto 2

Configuración del PIC16F84

El código comienza con la configuración del microcontrolador PIC16F84 mediante la instrucción `__CONFIG`. Se establecen las configuraciones del oscilador, desactivando el Watchdog Timer y otras opciones.

```
__CONFIG(FOSC_HS & WDTE_OFF & PWRTE_OFF & CP_OFF);
#define _XTAL_FREQ 4000000 // Frecuencia del oscilador (4MHz en este ejemplo)
```

Interrupción del Timer0

El código implementa una rutina de interrupción (`INT_ISR`) para el Timer0. Esta rutina se ejecuta cada vez que el Timer0 desborda, generando así una interrupción. Dentro de la rutina, se verifica si la interrupción es del Timer0 y, en caso afirmativo, alterna el estado de los LEDs RB4 y RB5 cada 250 ms.

```
void interrupt INT_ISR(void)
{
    if (T0IF && T0IE)
    {
        // Alternar el estado de los LEDs RB4 y RB5
        if (alternador == 0)
        {
            RB4 = 1;
            RB5 = 0;
            alternador = 1;
        }
        else
        {
            RB4 = 0;
            RB5 = 1;
            alternador = 0;
        }

        T0IF = 0;  // Limpia la bandera de la interrupción
        TMR0 = 6;  // Reinicia el valor del Timer0 para que genere una interrupción cada 250 ms
    }
}
```

Configuración del Timer0

El Timer0 se configura con un preescalador de 1:256, utilizando la fuente de reloj interna. Además, se establece el valor inicial del Timer0 para generar una interrupción cada 250 ms.

```
GIE = 1; // Habilitar las interrupciones globales
PSA = 0;                                        // Asignar el preescalador al Timer0
OPTION_REG = (OPTION_REG & 0b111111110) | 0b111; // Configuración del Timer0 con preescalador 1:256
T0CS = 0;                                       // Fuente de reloj interna para el Timer0
INTCON = 0b10000000;                            // Habilitar las interrupciones globales y la interrupción externa INT
TMR0 = 6;                                       // Con esta configuración, el Timer0 generará una interrupción cada 250 ms
```

Configuración de Puertos

Se configuran los pines de los puertos A y B para su uso en el programa.

```
TRISA = 0xFF; // Puerto A como entradas
TRISB = 0x00; // Puerto B como salidas
RB4 = 1;      // Inicialización: LED RB4 encendido
RB5 = 1;      // Inicialización: LED RB5 encendido
```

Bucle Principal

En el bucle principal el programa entra en un estado de espera hasta que uno de los pulsadores (RA0 o RA1) se presiona. En ese momento, se inicia el Timer0 para activar el control temporal de los LEDs.

```
while (1)
{
    if (RA0 == 0 || RA1 == 0)
    {
        // Iniciar el Timer0
        INTCON = 0b10100000;
    }
}
```

Punto 3

Configuración de pines

En la primera sección del código, se realiza la configuración de los pines del microcontrolador. Los pines del puerto A se establecen como entradas para la lectura del valor analógico, mientras que los pines de los puertos B y D se configuran como salidas para controlar el display de 8 segmentos.

```
TRISA = 0xFF; // Configurar todos los pines de PORTA como entradas
TRISB = 0x00; // Configurar todos los pines de PORTB como salidas
TRISD = 0x00; // Configurar todos los pines de PORTD como salidas
```

Configuración del ADC

La segunda sección del código se encarga de configurar el Convertidor Analógico-Digital (ADC). Se establece que todos los pines de entrada son analógicos, y se habilita la referencia de voltaje (VREF+) en el pin RA3.

```
ADCON1 = 0x81; // ADFM=1, todos los pines de entrada analógicos, VREF+ habilitado con entrada en RA3
ADCON0 = 0xC1; // Encender el ADC y seleccionar canal (0xC5 para comenzar a convertir en AN0)
```

Bucle Principal y Conversión ADC

El bucle principal del programa inicia la conversión analógico a digital en el canal AN0 y espera a que la conversión se complete antes de continuar.

```
while (1)
{
    ADCON0 = 0xC5; // Iniciar la conversión en AN0
    while (ADCON0 & 0b00000100)
        ; // Esperar a que la conversión se complete
```

Lectura del Valor Analógico y Visualización

Una vez completada la conversión, se combinan los resultados del ADC alto y bajo para obtener el valor analógico completo. Este valor se muestra en un display de 8 segmentos mediante la función `disp_adc`. Se activa y desactiva un bit en el puerto D para indicar el progreso del proceso. Luego se hace un retraso de 64ms.

```
unsigned short adc_value = (ADRESH << 8) | ADRESL; // Combinar los resultados del ADC alto y bajo

// Mostrar el valor capturado
PORTD = 0x01; // Mostrar que el ISR está en progreso
disp_adc(adc_value);
PORTD = 0x00; // Quitar el bit de progreso

__delay_ms(64); // Retraso de 64ms
```

Función disp_adc

El proceso se lleva a cabo en dos etapas: la configuración y presentación del byte inferior y del byte superior del valor analógico. Primero, se toma el byte inferior del valor (`value`) y se coloca en el puerto B, asegurándose de que solo se tomen los 8 bits menos significativos. A continuación, se activa y desactiva el Latch (Latch Enable - LE), un componente necesario para transferir el byte inferior al display de 8 segmentos. Después de un breve periodo de espera, se repite el proceso para el byte superior, utilizando el desplazamiento a la derecha (`>>`) por 8 bits para obtener estos bits de la parte alta del valor analógico. Nuevamente, se activa y desactiva el Latch para transferir el byte superior al display.

```
void disp_adc(int value)
{
    PORTB = value & 0xFF; // Escribir el byte inferior en PORTB

    PORTD |= 0x40;  // Activar/desactivar el latch
    __delay_ms(1);  // Esperar
    PORTD &= ~0x40; // Activar/desactivar el latch

    PORTB = (value >> 8) & 0xFF; // Escribir el byte superior en PORTB

    PORTD |= 0x80;  // Activar/desactivar el latch
    __delay_ms(1);  // Esperar
    PORTD &= ~0x80; // Activar/desactivar el latch
}
```