Hernández Diaz Sebastian

Laboratorio de Microcomputadoras

Práctica No. 8

Puerto serie y programación en C

Objetivo. Realización de programas a través de programación en C y empleo del puerto serie para visualización y control.

Desarrollo. Realizar los siguientes ejercicios.

1.- Escribir, comentar, compilar y ejecutar el siguiente programa usando el ambiente del PIC C Compiler.

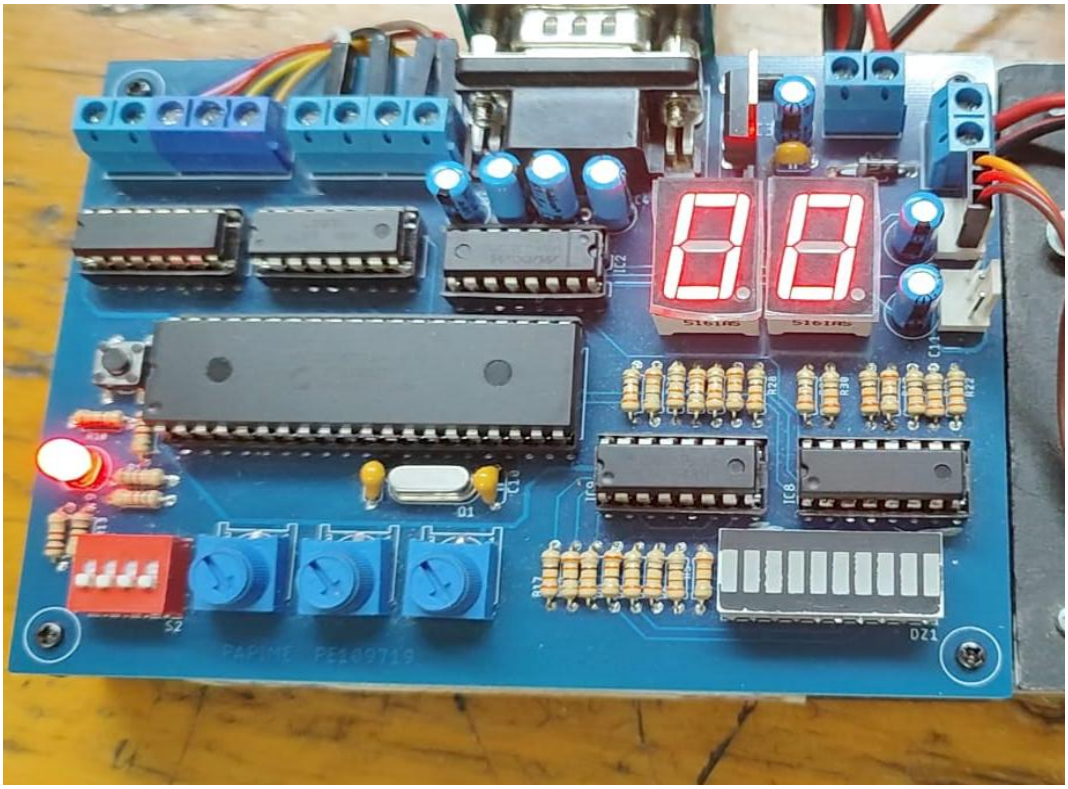
El código usado para esta actividad fue el siguiente:

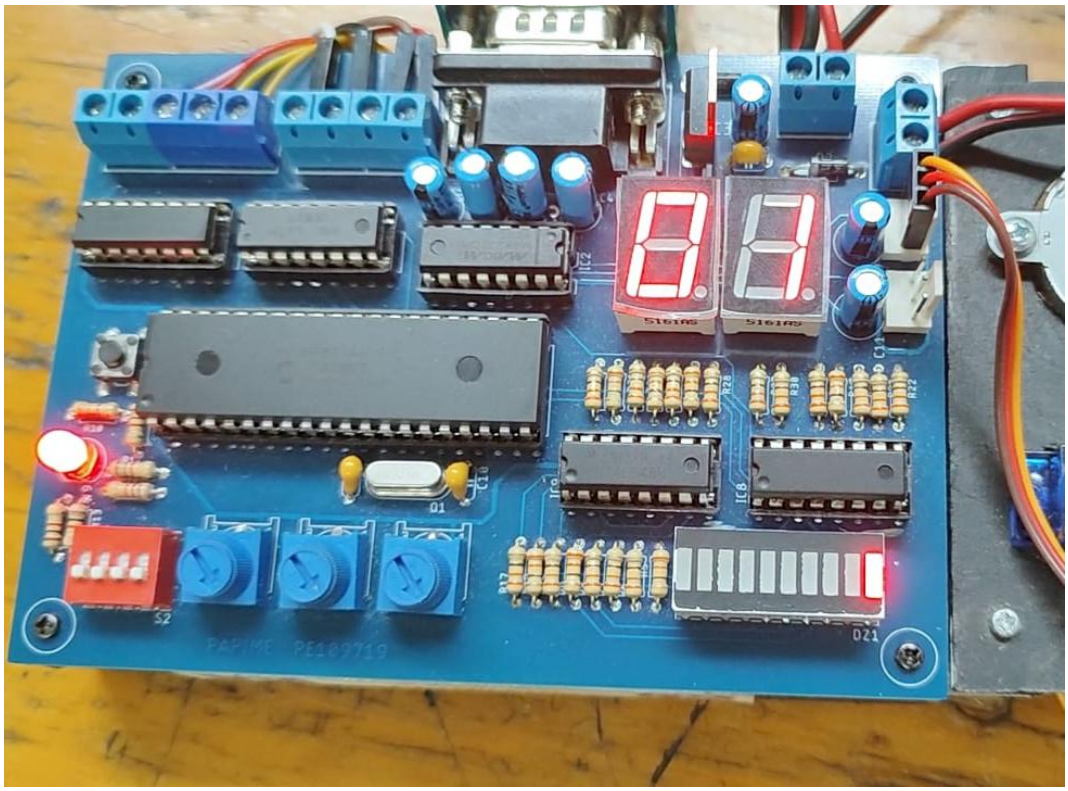
```
#include <16f877.h>                          // Cabecera del PIC16F877
#fuses HS, NOPROTECT                         // Oscilador alta velocidad, sin protección de código
#use delay(clock = 20000000)                 // Frecuencia de reloj de 20 MHz
#org 0x1F00, 0x1FFF void loader16F877(void){} // Reservar espacio para bootloader

void main()
{ // Inicio del programa principal
    while (1)
    {                   // Bucle infinito
        output_d(0x01); // Encender RD0
        delay_ms(1000); // Esperar 1 segundo
        output_d(0x00); // Apagar puerto D
        delay_ms(1000); // Esperar 1 segundo
    }
}
```

Este programa hace parpadear un LED conectado al pin RD0 del PIC16F877 cada segundo (encendido 1 segundo, apagado 1 segundo). Usa un reloj de 20 MHz, tiene reservada la memoria para un posible bootloader, y está pensado para ejecutarse indefinidamente.

La ejecución del código es la siguiente:

Hernández Diaz Sebastian





Por último, para poder utilizar este código en lenguaje ensamblador se tiene que tener en cuenta lo siguiente:

1. Configurar el puerto D como salida (TRISD = 0x00).
2. Encender RD0 (PORTD = 0x01).
3. Implementar una rutina de retardo de 1 segundo.
4. Apagar PORTD (PORTD = 0x00).
5. Repetir el ciclo infinitamente.

TRISD se encuentra en banco 1 → se debe seleccionar el banco adecuado.

PORTD está en banco 0.

También se necesitan usar instrucciones como:

- bsf, bcf, movlw, movwf, goto, call, return

Para el retardo: usar bucles anidados que decrecen registros y verifican con decfsz.

2.- Modificar el programa para que active y desactive todos los bits del puerto B.

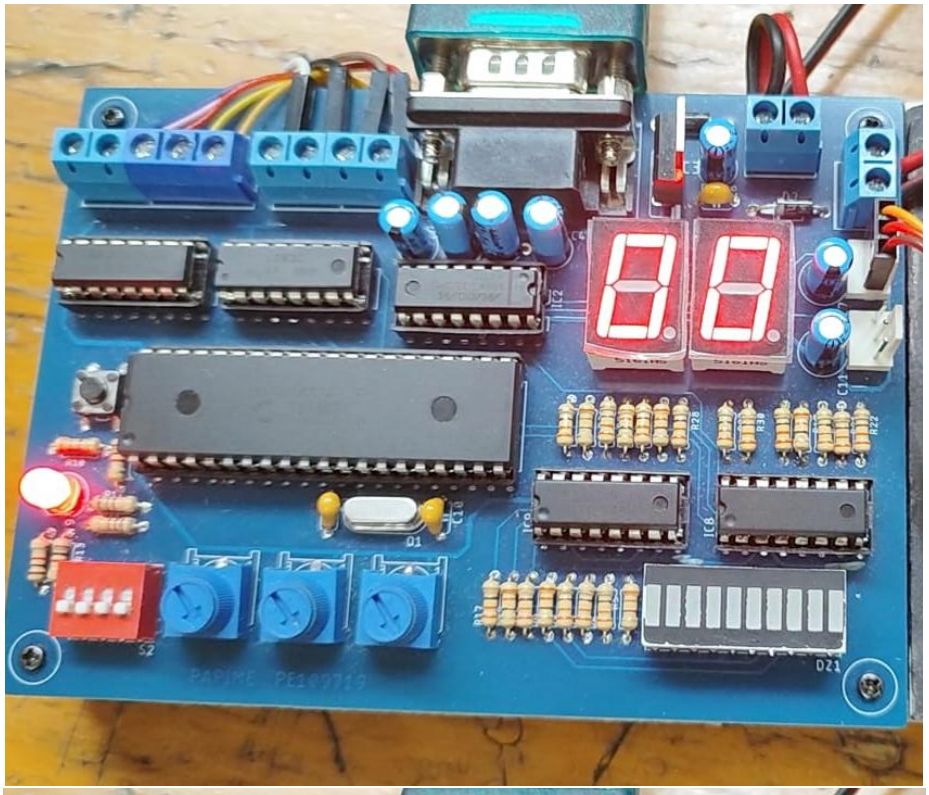
El código usado para esta actividad fue el siguiente:

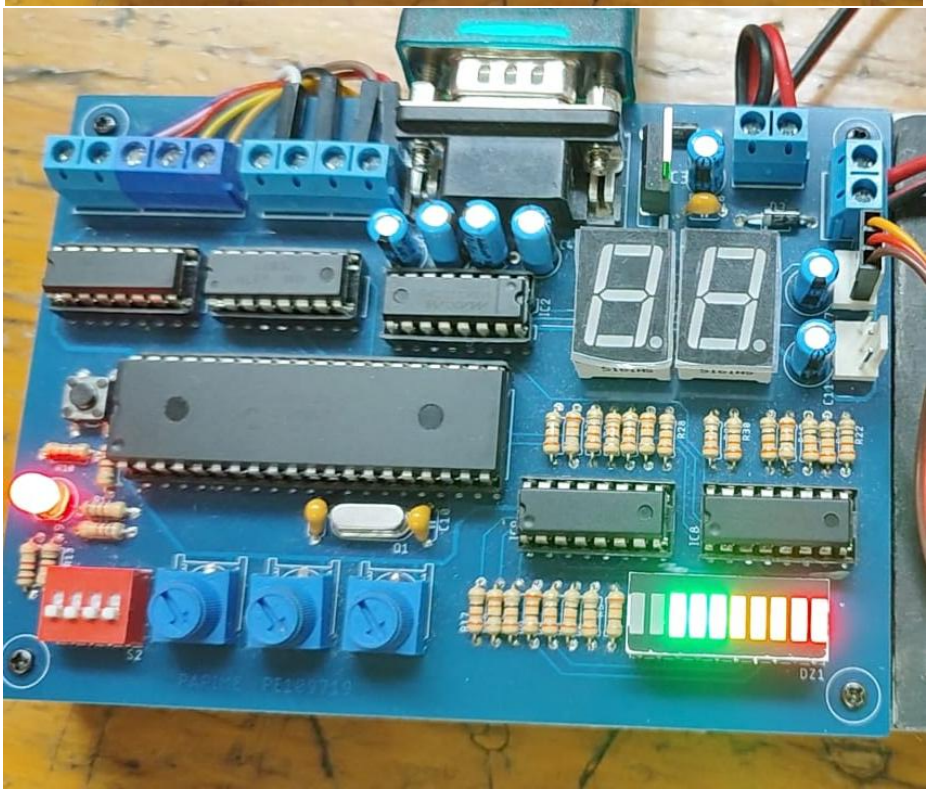
```
#include <16f877.h>                          // Cabecera del PIC16F877
#fuses HS, NOPROTECT                         // Oscilador alta velocidad, sin protección de código
#use delay(clock = 20000000)                 // Frecuencia de reloj de 20 MHz
#org 0x1F00, 0x1FFF void loader16F877(void){} // Reservar espacio para bootloader

void main()
{ // Inicio del programa principal
    while (1)
    {                   // Bucle infinito
        output_d(0xFF); // Encender todos los pines del puerto D
        delay_ms(1000); // Esperar 1 segundo
        output_d(0x00); // Apagar todos los pines del puerto D
        delay_ms(1000); // Esperar 1 segundo
    }
}
```

El programa principal realiza una acción muy simple: parpadeo de todos los pines del puerto D cada 1 segundo. Esto se logra alternando entre escribir 0xFF (todos los bits en 1 → encender todos los LEDs conectados al puerto D) y 0x00 (todos los bits en 0 → apagar todos los LEDs).

La estructura while(1) genera un bucle infinito, donde el código debe ejecutarse constantemente mientras el sistema esté encendido.

La ejecución del código es la siguiente:





Por último, para poder utilizar este código en lenguaje ensamblador se tiene que tener en cuenta lo siguiente:

Seleccionar bancos de memoria para acceder a TRISD y PORTD.

Usar las instrucciones:

- bsf y bcf para manipular bits de STATUS (RP0 y RP1).
- movlw, movwf para cargar y almacenar datos.
- goto, call, return para control de flujo.
- decfsz para retardos por software.

Crear una rutina de retardo aproximado para simular delay_ms(1000).

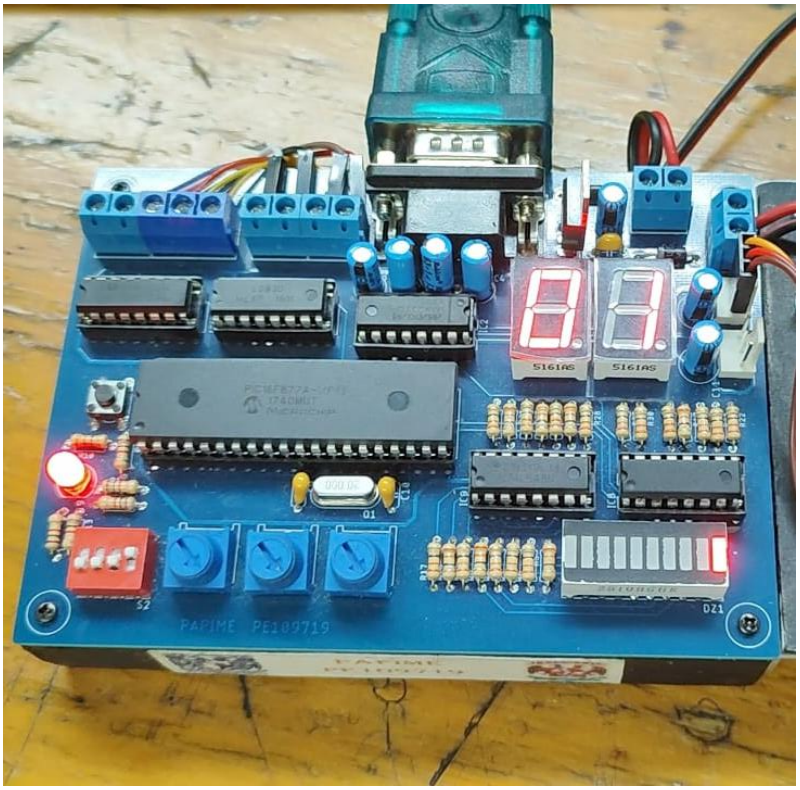
3.- Escribir, comentar, compilar y ejecutar el siguiente programa usando el ambiente del PIC C Compiler.

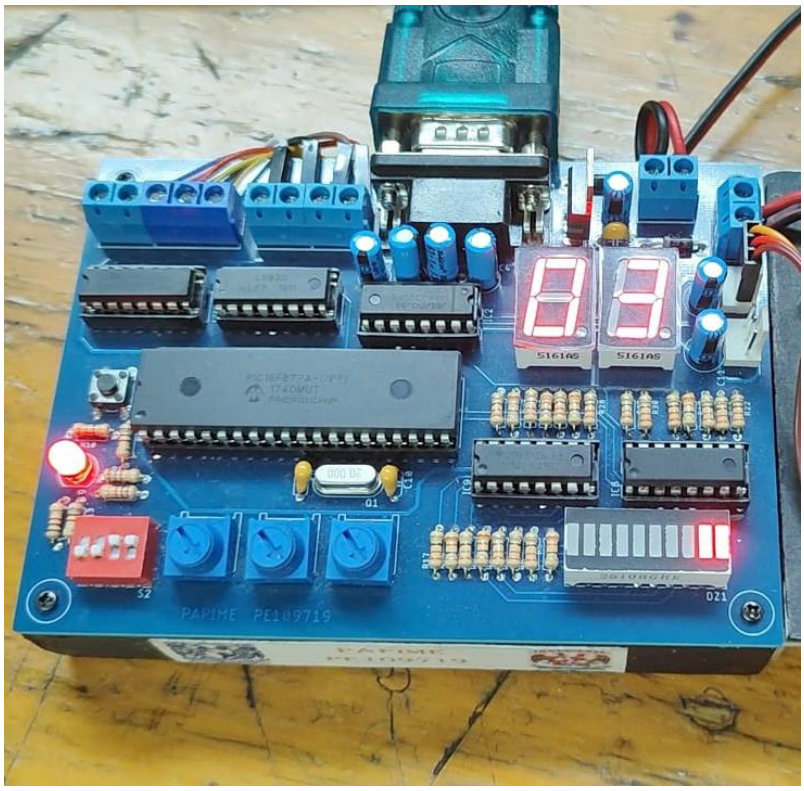
El código usado para esta actividad fue el siguiente:

```
#include <16f877.h>                          // Cabecera para el microcontrolador PIC16F877
#fuses HS, NOPROTECT                         // Oscilador de alta velocidad, sin protección de código
#use delay(clock = 20000000)                 // Frecuencia de reloj: 20 MHz
#org 0x1F00, 0x1FFF void loader16F877(void){} // Espacio reservado para el bootloader

int var1; // Variable entera para almacenar el valor de PORTA

void main()
{ // Función principal
    while (1)
    {                    // Bucle infinito
        var1 = input_a(); // Leer el valor del puerto A
        output_d(var1);   // Enviar ese valor al puerto D
    } // Fin del bucle
} // Fin del programa
```

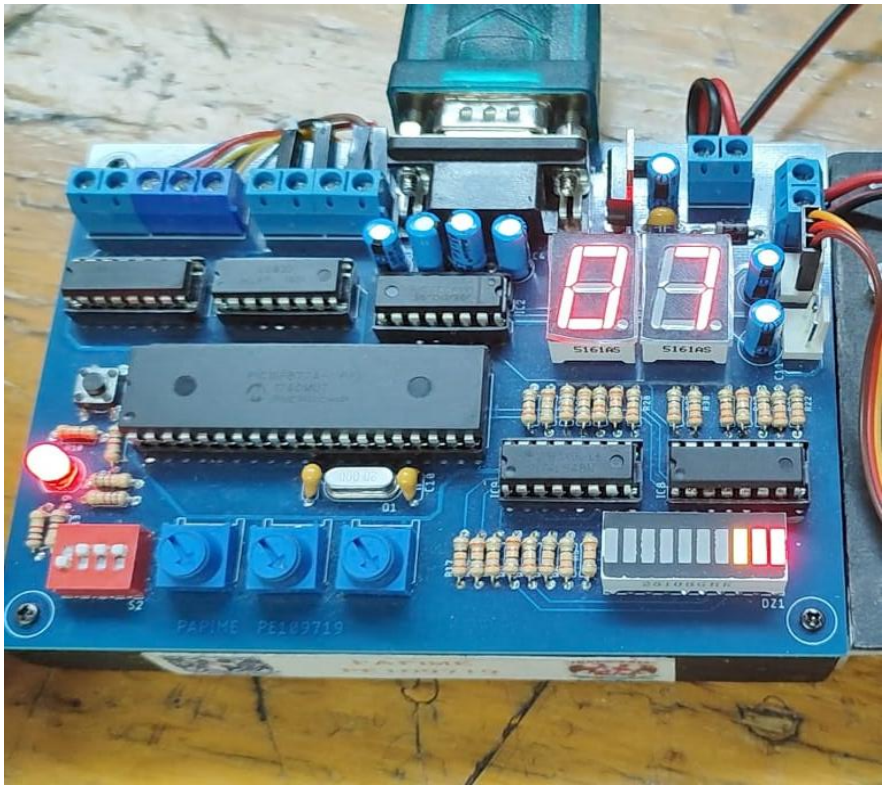
Este programa tiene como función principal leer continuamente el estado de las entradas del puerto A del microcontrolador PIC16F877 y copiar ese mismo valor al puerto B como salida. Es decir, actúa como un espejo digital donde cada bit que esté en alto (1) en el puerto A se reflejará encendido en el puerto D, y cada bit en bajo (0) se reflejará apagado.

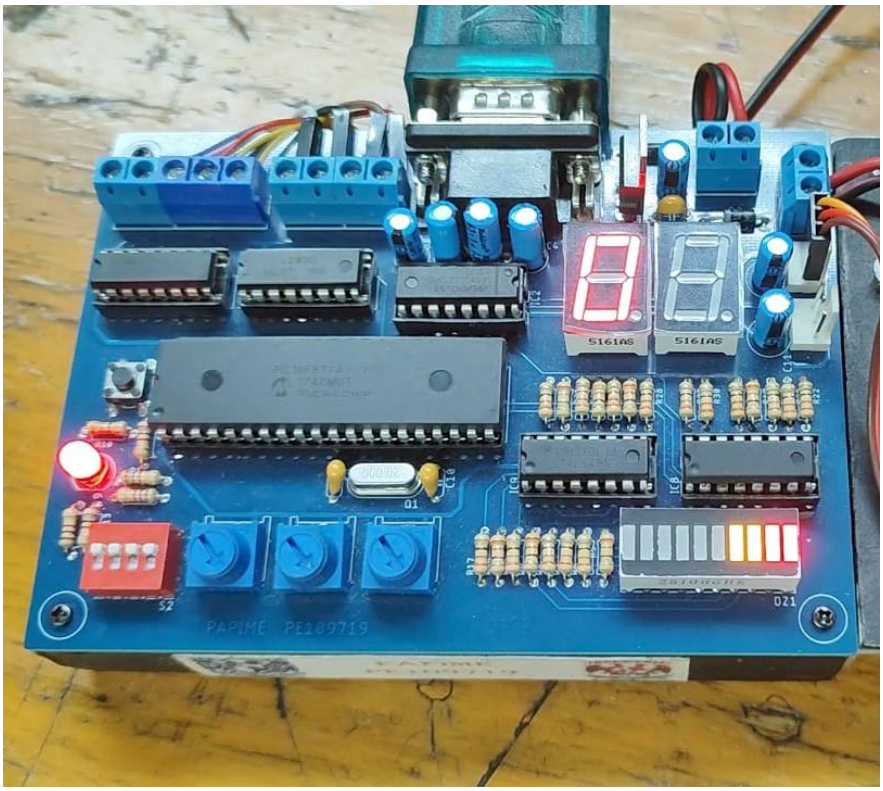
La ejecución del código es la siguiente:

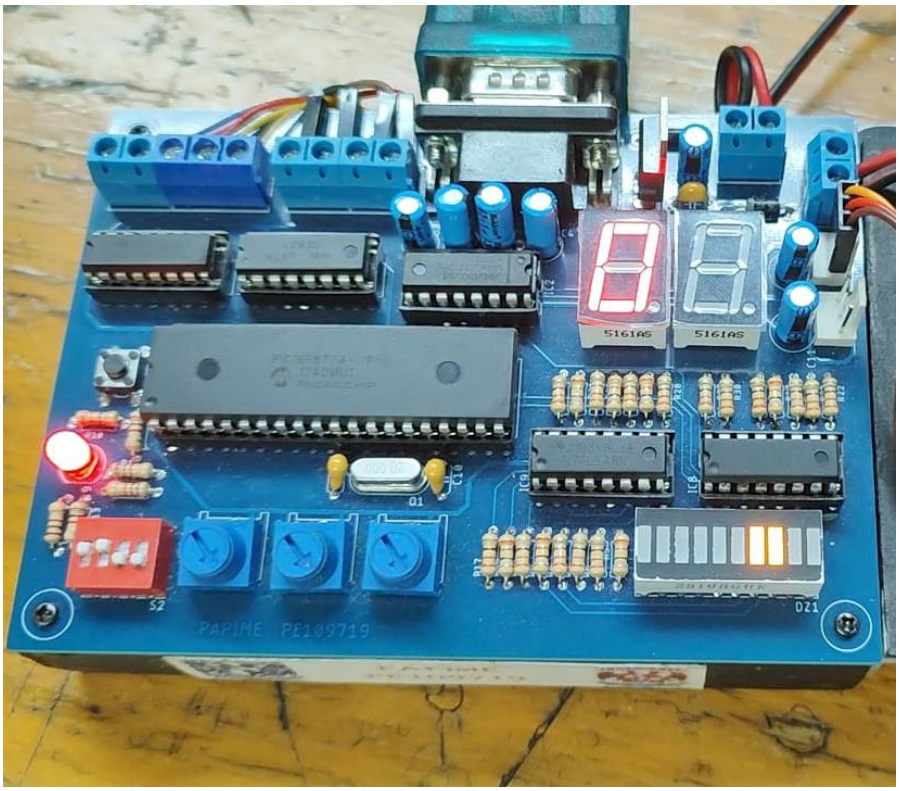




Hernández Diaz Sebastian







Por último, para poder utilizar este código en lenguaje ensamblador se tiene que tener en cuenta lo siguiente:

TRISA debe configurarse como entrada (0xFF).

TRISD debe configurarse como salida (0x00).

Como los registros TRISA y TRISD están en banco 1, deberás cambiar a ese banco con los bits RP0 y RP1 del registro STATUS.

Usar la instrucción movf PORTA, W para mover el contenido de PORTA al registro W.

Asegurarse de estar en el **banco 0**, ya que PORTA y PORTD están ahí.

Después de cargar el valor de PORTA en W, usar movwf PORTD para copiarlo a PORTD.

4.- Escribir, comentar, compilar y ejecutar el siguiente programa usando el ambiente del PIC C Compiler.

El código usado para esta actividad fue el siguiente:

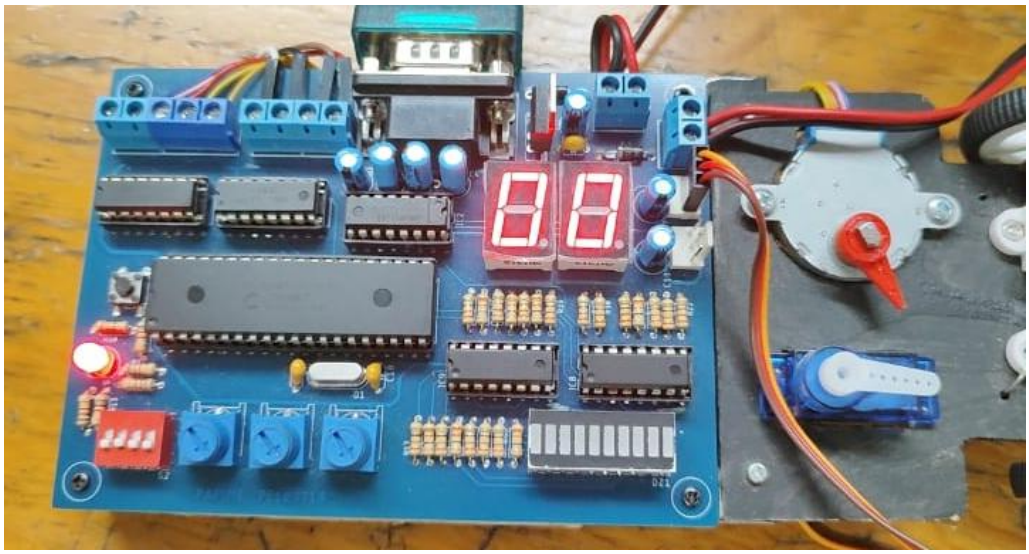
```
#include <16f877.h>                                  // Cabecera para el microcontrolador PIC16F877
#fuses HS, NOPROTECT                                 // Oscilador de alta velocidad, sin protección de código
#use delay(clock = 20000000)                         // Frecuencia de reloj: 20 MHz
#use rs232(baud = 38400, xmit = PIN_C6, rcv = PIN_C7) // Configura la comunicación serial RS232
#org 0x1F00, 0x1FFF void loader16F877(void){}         // Espacio reservado para el bootloader

void main()
{ // Función principal
    while (1)
    {                                                // Bucle infinito
        output_d(0xff);                              // Enciende todos los pines del puerto D (todos en alto)
        printf(" Todos los bits encendidos \n\r");   // Envía mensaje por puerto serial
        delay_ms(1000);                              // Espera 1 segundo

        output_d(0x00);                             // Apaga todos los pines del puerto D (todos en bajo)
        printf(" Todos los leds apagados \n\r");     // Envía mensaje por puerto serial
        delay_ms(1000);                             // Espera 1 segundo
    } // Fin del bucle
} // Fin del programa
```

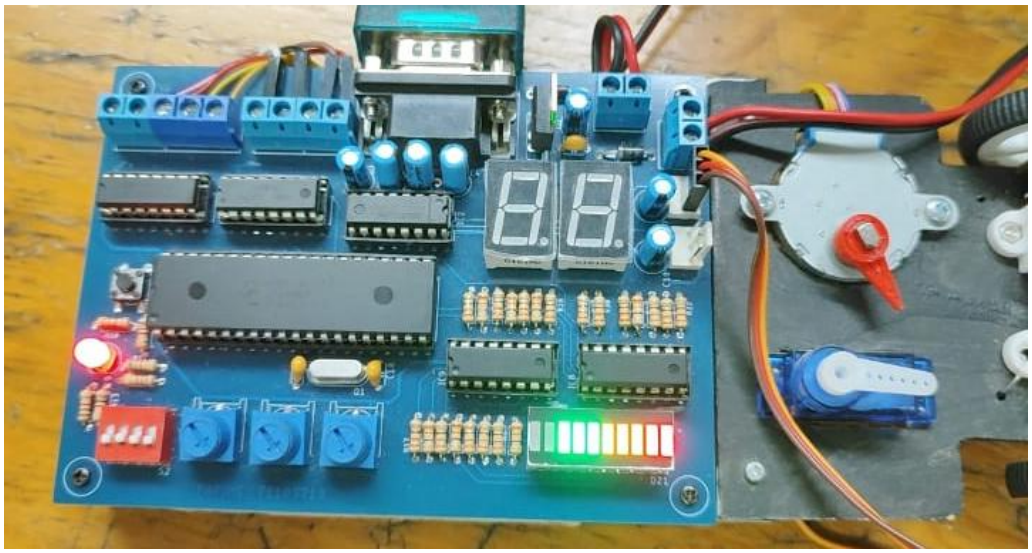
Este programa está diseñado para que el microcontrolador PIC16F877 realice dos acciones principales en un bucle infinito:

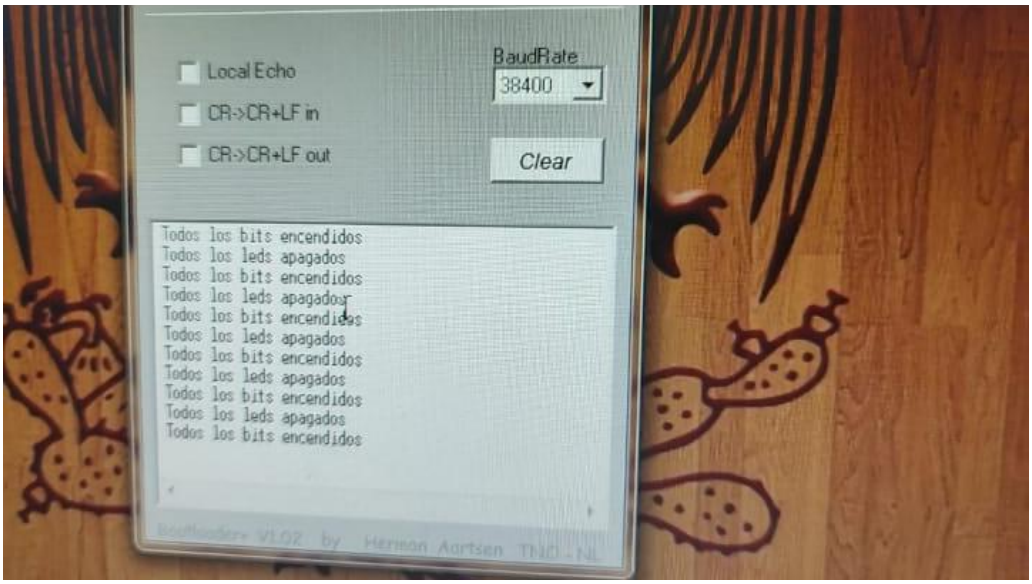
1. Enciende y apaga todos los pines del puerto B alternativamente cada segundo, lo cual enciende y apaga 8 LEDs conectados a dicho puerto.
2. Envía un mensaje por el puerto serial RS232 cada vez que realiza el cambio de estado, indicando si los pines (o LEDs) están encendidos o apagados. Este mensaje puede visualizarse en un monitor serial en la computadora.

En resumen, el programa controla una secuencia de encendido/apagado de LEDs y comunica su estado por el puerto serial, siendo útil tanto para pruebas visuales como para monitoreo remoto del estado del sistema.

La ejecución del código es la siguiente:







Por último, para poder utilizar este código en lenguaje ensamblador se tiene que tener en cuenta lo siguiente:

El programa usa el puerto D como salida, así que necesitas:

- Cambiar a banco 1.
- Configurar TRISD = 0x00 (todos los pines como salida).
- Volver al banco 0 para manipular PORTD

En C, se usa printf(). En ensamblador, seria:

- Escribir manualmente cada carácter ASCII en el registro TXREG
- Esperar a que TXIF esté en 1 antes de enviar cada byte
- Esto se hace en un bucle, carácter por carácter

Finalmente, movlw 0xFF y movwf PORTD para encender y movlw 0x00, movwf PORTD para apagar.

5.- Realizar las modificaciones necesarias al ejercicio 2 de la práctica tres para que ahora el comando que selecciona la acción sea a través del puerto serie, usar retardos de ½ segundos, usando programación en C.

DATO	ACCION Puerto B	*Ejecución*
0	Todos los bits apagados	*00000000*
1	Todos los bits encendidos	*11111111*
2	Corrimiento del bit más significativo hacia la derecha	*10000000* *00000001*
3	Corrimiento del bit menos significativo hacia la izquierda	*00000001* *10000000*
4	Corrimiento del bit más significativo hacia la derecha y a la izquierda	*10000000* ..... *00000001* *10000000*
5	Apagar y encender todos los bits.	*00000000* *11111111*

El código usado para esta actividad fue el siguiente:

```
#include <16f877.h> // Incluye el archivo de cabecera específico para el PIC16F877

// Configura los fusibles: oscilador de alta velocidad, desactiva watchdog, sin protección de código, sin programación en bajo voltaje
#fuses HS, NOWDT, NOPROTECT, NOLVP

#use delay(clock = 20000000) // Define la frecuencia de reloj del sistema en 20 MHz

// Configura la comunicación serial RS232 a 38400 baudios con 8 bits, sin paridad y 1 bit de parada
#use rs232(baud = 38400, xmit = PIN_C6, rcv = PIN_C7, bits = 8, parity = N, stop = 1)

void encender()
{
   output_d(0xFF); // Enciende todos los bits del puerto D (todos los LEDs conectados se encienden)
}

void apagar()
{
   output_d(0x00); // Apaga todos los bits del puerto D (todos los LEDs conectados se apagan)
}
```

Hernández Diaz Sebastian

```
void derecha()
{
   int i;
   int valor = 0x80; // Inicia con el bit más significativo encendido (10000000)
   for (i = 0; i < 8; i++)
   {
      output_d(valor); // Muestra el valor en el puerto D
      delay_ms(500);   // Espera medio segundo
      valor >>= 1;     // Desplaza el bit hacia la derecha
   }
}

void izquierda()
{
   int i;
   int valor = 0x01; // Inicia con el bit menos significativo encendido (00000001)
   for (i = 0; i < 8; i++)
   {
      output_d(valor); // Muestra el valor en el puerto D
      delay_ms(500);   // Espera medio segundo
      valor <<= 1;     // Desplaza el bit hacia la izquierda
   }
}

void ambos()
{
   derecha();   // Ejecuta la función de desplazamiento a la derecha
   izquierda(); // Luego ejecuta el desplazamiento a la izquierda
}
```

```
void parpadeo()
{
   int i;
   for (i = 0; i < 5; i++)
   {                    // Repite el encendido/apagado 5 veces
      output_d(0xFF); // Enciende todos los bits del puerto D
      delay_ms(500);  // Espera medio segundo
      output_d(0x00); // Apaga todos los bits
      delay_ms(500);  // Espera medio segundo
   }
}
```

```c
void main()
{
   char dato;
   set_tris_d(0x00); // Configura el puerto D como salida
   output_d(0x00);   // Inicialmente todos los pines en bajo

   while (TRUE)
   {
      if (kbhit())
      {                 // Verifica si hay un dato recibido por el puerto serial
         dato = getc(); // Lee el carácter recibido
         switch (dato)
         {
         case '0':
            apagar(); // Si el dato es '0', apaga los LEDs
            break;
         case '1':
            encender(); // Si el dato es '1', enciende todos los LEDs
            break;
         case '2':
            derecha(); // Si el dato es '2', realiza desplazamiento hacia la derecha
            break;
         case '3':
            izquierda(); // Si el dato es '3', realiza desplazamiento hacia la izquierda
            break;
         case '4':
            ambos(); // Si el dato es '4', realiza desplazamiento derecha e izquierda
            break;
         case '5':
            parpadeo(); // Si el dato es '5', enciende y apaga todos los LEDs 5 veces
            break;
         default:
            apagar(); // Si el dato no es válido, apaga por seguridad
            break;
         }
      }
   }
}
```

Este código implementa un controlador de LEDs a través de comandos enviados por comunicación serial (UART) utilizando un microcontrolador PIC. Al recibir un carácter por el puerto serial, el programa ejecuta una acción específica sobre los LEDs conectados al puerto D del microcontrolador.

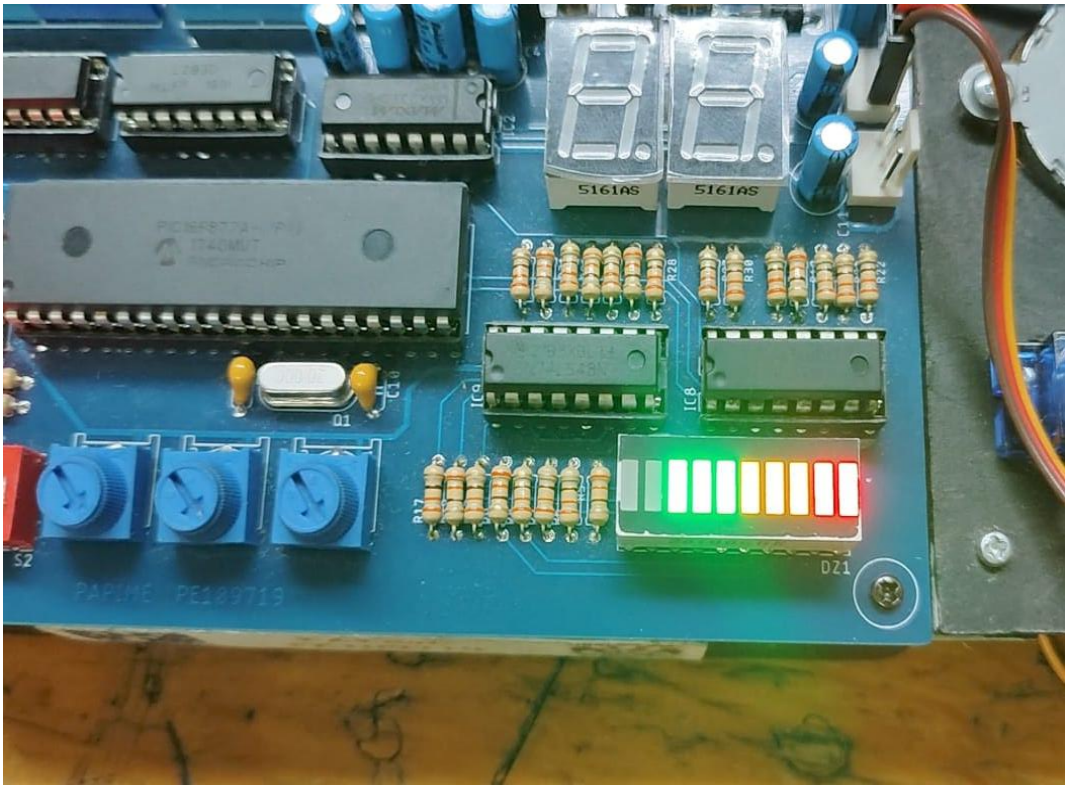
Las funciones disponibles permiten:

- Encender todos los LEDs.
- Apagar todos los LEDs.
- Encender los LEDs en una secuencia de derecha a izquierda o de izquierda a derecha (simulando movimiento).

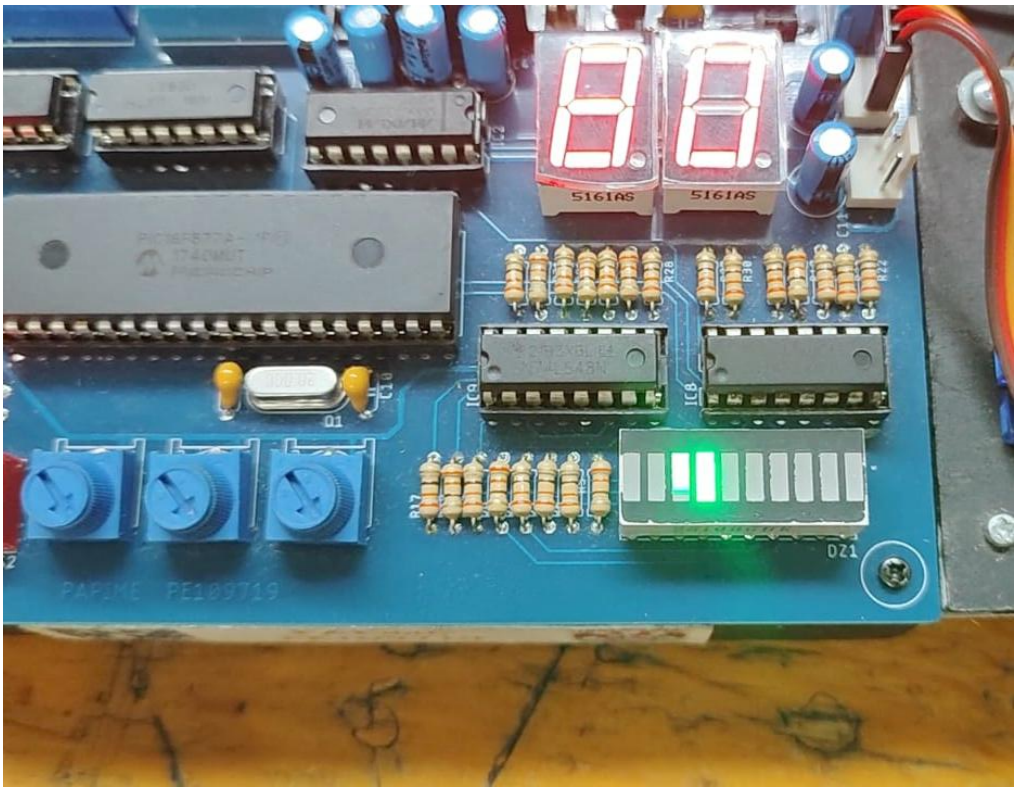
- Ejecutar ambas secuencias consecutivamente.
- Hacer que los LEDs parpadeen.

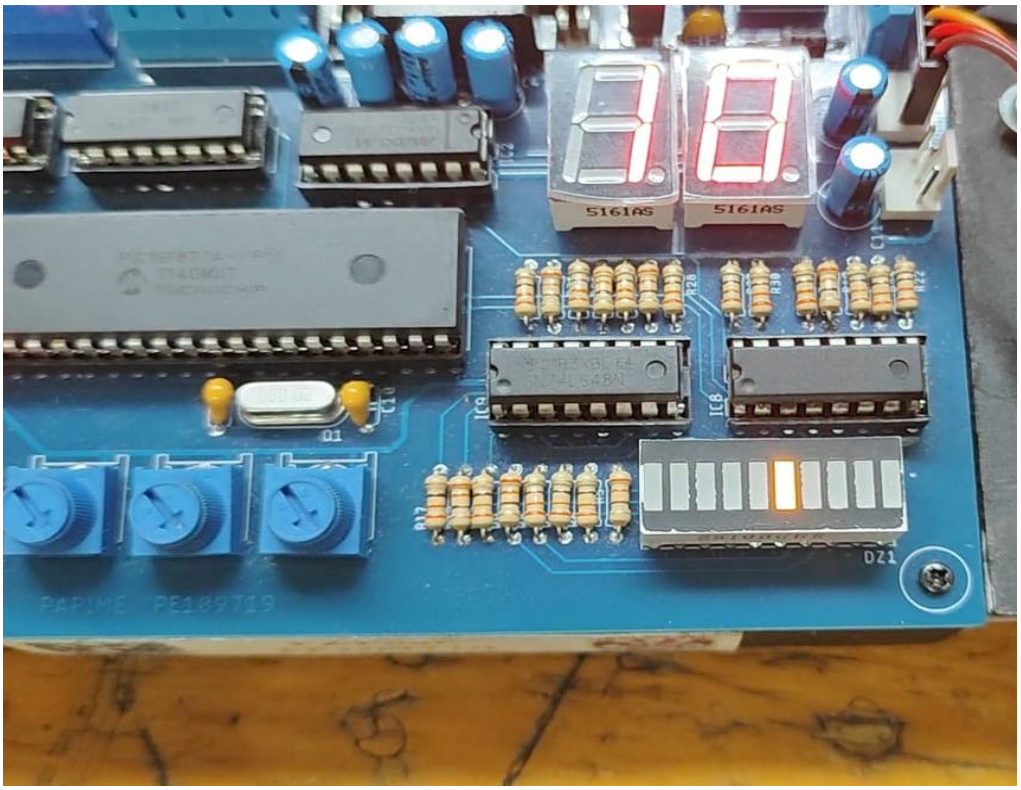
Todo esto está gestionado mediante un bucle principal que escucha continuamente datos seriales y responde según el carácter recibido.

La ejecución del código es la siguiente:

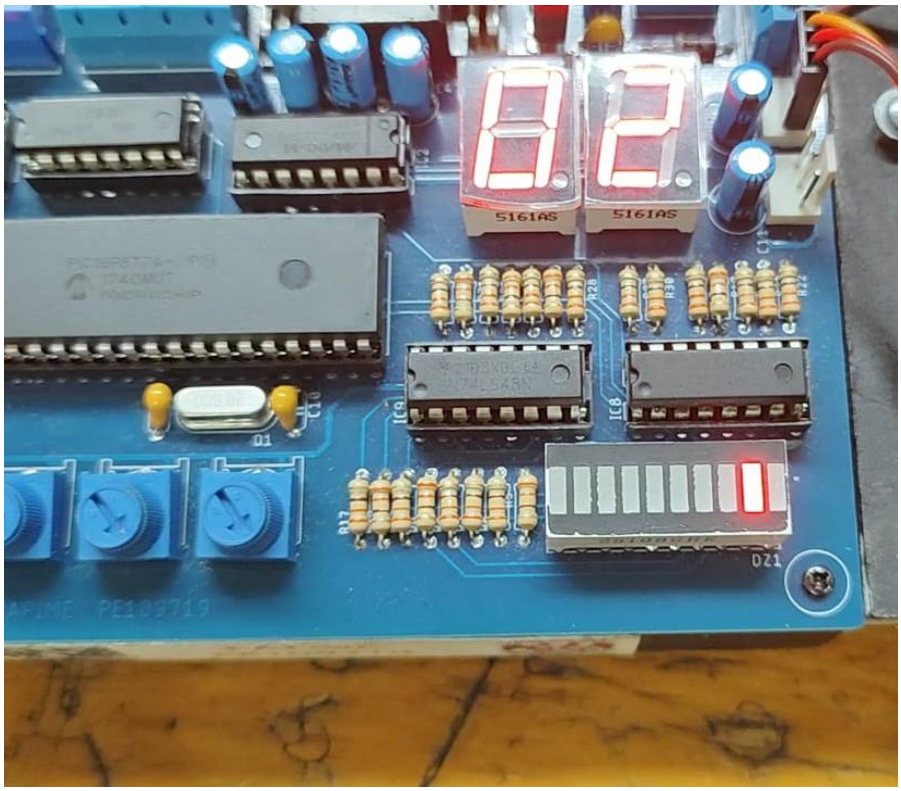


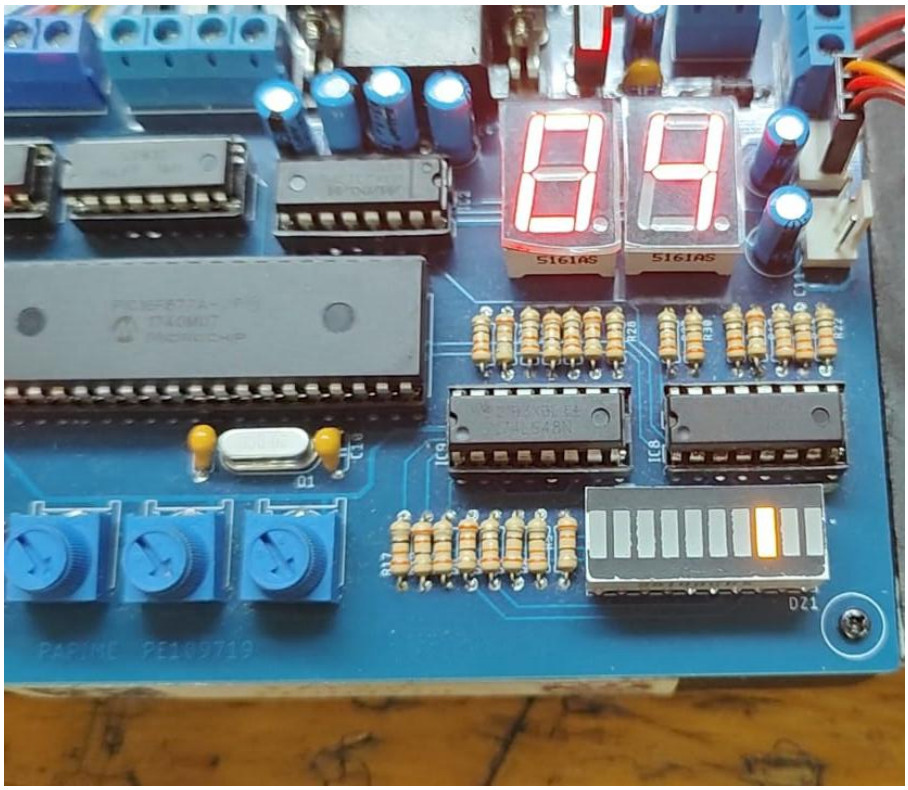
Hernández Diaz Sebastian



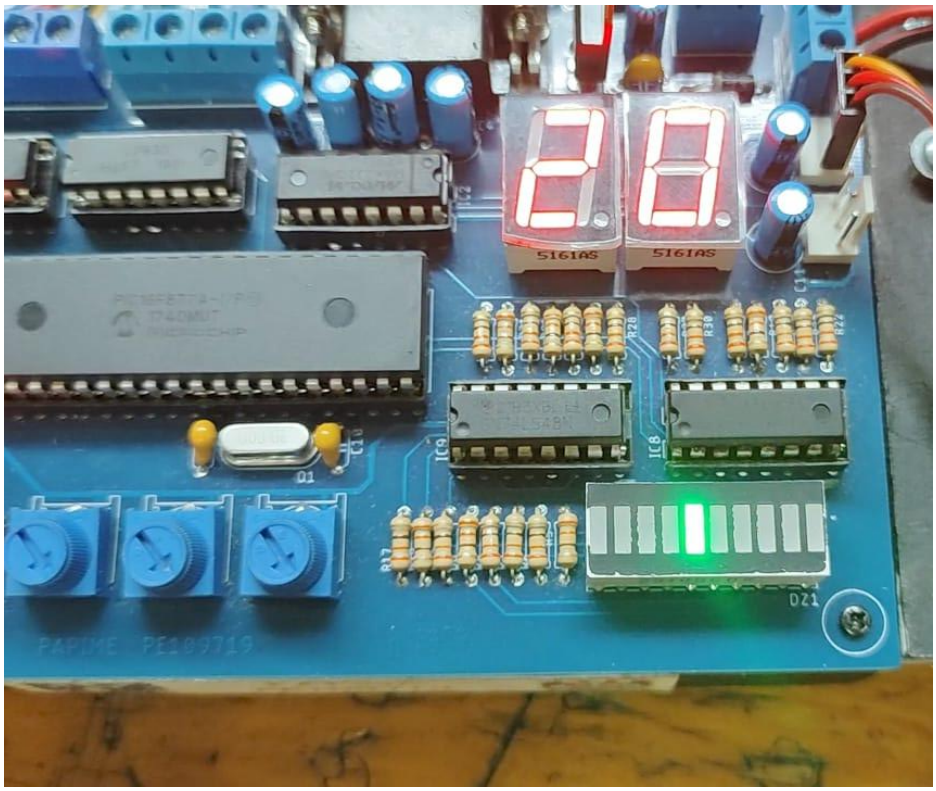


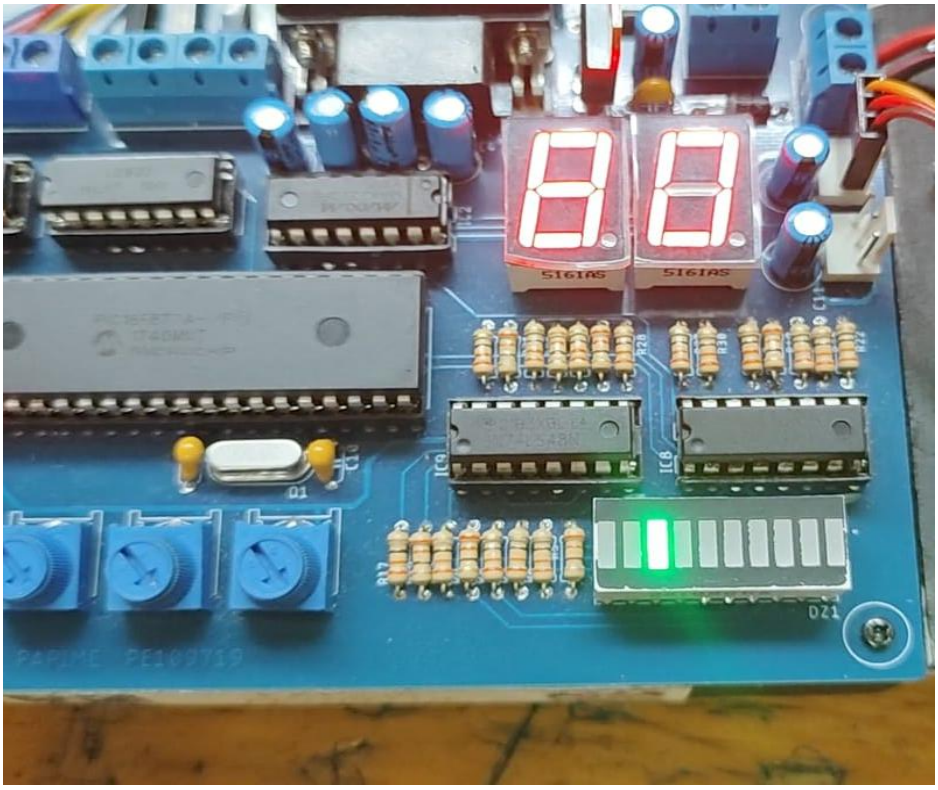
Hernández Diaz Sebastian



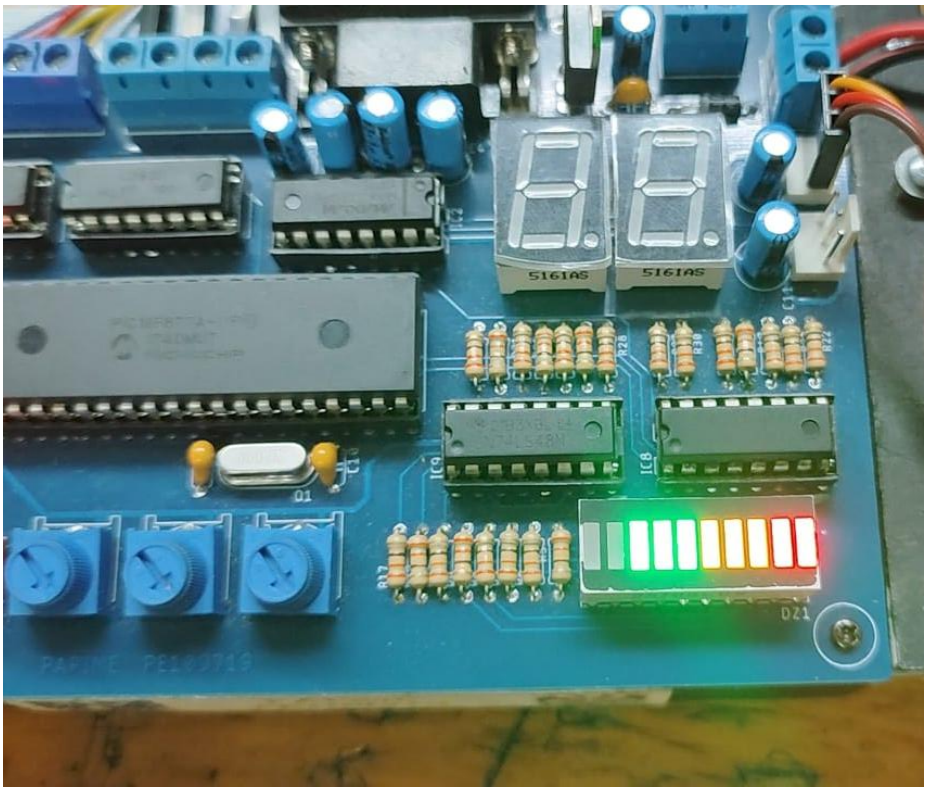


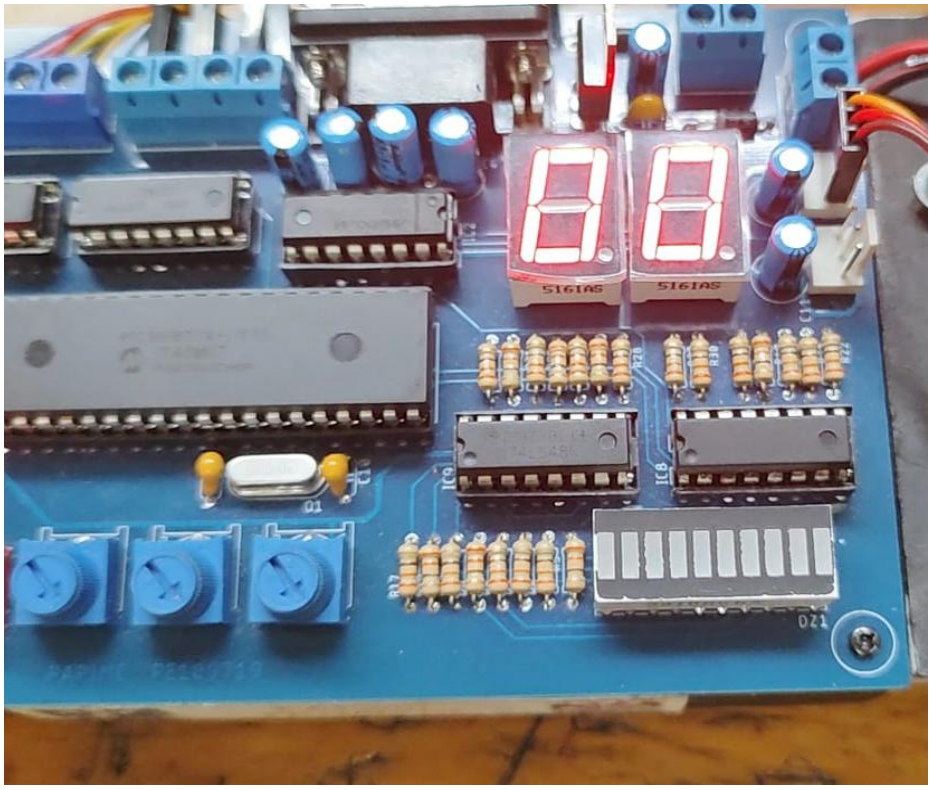
Hernández Diaz Sebastian

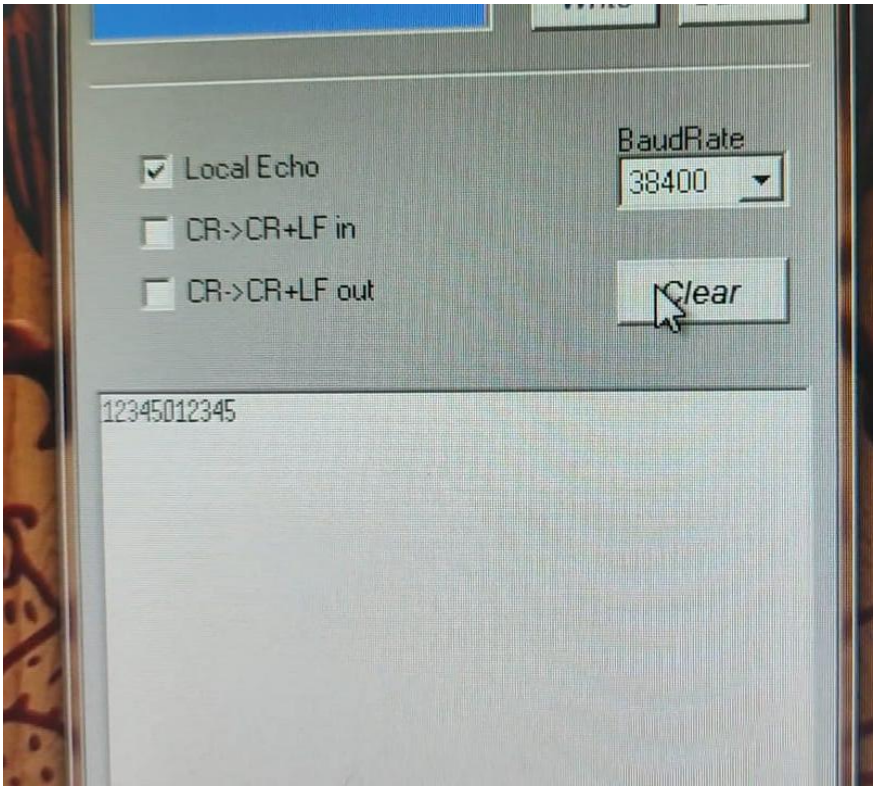




Hernández Diaz Sebastian







Por último, para poder utilizar este código en lenguaje ensamblador se tiene que tener en cuenta lo siguiente:

Función encender()

- Acción: Escribir 0xFF en PORTD para encender todos los pines.
- En ensamblador:
    - Asegurarse de estar en banco 0.
    - Usar movlw 0xFF y movwf PORTD.

Función apagar()

- Acción: Escribir 0x00 en PORTD para apagar todos los pines.
- En ensamblador:
    - movlw 0x00 y movwf PORTD.

Función derecha()

- Acción: Enciende un bit a la vez, desde el más significativo (bit 7) hacia el menos significativo (bit 0), con retardo entre cada paso.
- En ensamblador:
    - Inicializar un registro (por ejemplo, REG) con 0x80.

    - Mostrar su valor en PORTD.
    - Llamar a una rutina de retardo.
    - Usar rrf para desplazar el bit hacia la derecha.
    - Repetir 8 veces usando un contador y decfsz.

Función izquierda()

- Acción: Enciende un bit a la vez, de izquierda a derecha (de bit 0 a bit 7), con retardo.
- En ensamblador:
    - Inicializar el registro con 0x01.
    - Mostrarlo en PORTD.
    - Llamar al retardo.
    - Usar rlf para desplazar a la izquierda.
    - Controlar el número de repeticiones con un contador.

Función ambos()

- Acción: Ejecuta primero derecha() y luego izquierda().
- En ensamblador:
    - Simplemente llamar secuencialmente a las subrutinas derecha y luego izquierda.

Función parpadeo()

- Acción: Enciende y apaga todos los bits del puerto D repetidamente (5 veces).
- En ensamblador:
    - Bucle que se repite 5 veces (con un contador).
    - En cada iteración:
        - PORTD = 0xFF, luego retardo.
        - PORTD = 0x00, luego retardo.

Hernández Diaz Sebastian

Conclusiones

El desarrollo de estos ejercicios demuestra de manera práctica la importancia de la comunicación serial, especialmente cuando se requiere una interfaz sencilla pero eficiente entre un microcontrolador y un dispositivo externo, como una computadora. Al sustituir la entrada tradicional basada en puertos (por ejemplo, PORTA) por comandos enviados a través del puerto serial (USART), se logra una interacción mucho más flexible, escalable y adecuada para proyectos reales donde se necesita control remoto o automatización.

En el primer ejercicio, se aprendió el uso del puerto D para generar una secuencia simple de parpadeo de un solo bit con retardos controlados mediante la instrucción delay_ms(). Este ejercicio permitió familiarizarse con funciones básicas como output_b() y el concepto de bucle infinito while(1), el cual es común en sistemas embebidos.

En el segundo ejercicio, se amplió la funcionalidad del código inicial al controlar todos los bits del puerto D de forma simultánea, lo que introdujo la importancia del manejo de valores binarios o hexadecimales para activar o desactivar múltiples salidas digitales. Se entendió cómo modificar directamente registros del puerto con valores como 0xFF (todos los bits en alto) y 0x00 (todos los bits en bajo).

El tercer ejercicio permitió implementar una lectura digital del puerto A, utilizando la instrucción input_a(), y transmitir directamente el valor leído al puerto D con output_b(var1). Esto sirvió como introducción al concepto de lectura de entradas digitales, mapeando el estado físico de interruptores, sensores o cualquier dispositivo conectado a ese puerto.

En el cuarto ejercicio se integró el uso de la comunicación serial asíncrona RS232 mediante la directiva #use rs232(). Se configuró el puerto serial con los pines RC6 y RC7 para transmitir mensajes con printf(), proporcionando una salida útil para monitoreo desde una terminal en el PC. Este ejercicio facilitó el entendimiento de cómo se realiza la interacción entre el microcontrolador y un sistema externo, como un computador.

Finalmente, el ejercicio cinco integró todos los conocimientos anteriores para crear un sistema de control más complejo donde el microcontrolador recibe comandos por UART y ejecuta acciones sobre el puerto B según una tabla de referencia. Se implementaron varias funciones que involucran desplazamientos de bits (corrimientos a izquierda, derecha y ambos sentidos), encendido/apagado de LEDs, y parpadeo secuencial. Esto permitió practicar el uso de estructuras switch-case, la comparación de caracteres recibidos y la ejecución de rutinas basadas en condiciones. Asimismo, se reforzó el uso de retardos, manejo de puertos, y modularización del código con funciones.

Bibliografía

Ibrahim, D. (2011). *Designing Embedded Systems with PIC Microcontrollers: Principles and Applications*. Newnes.

Barnett, R. H., Cox, S., & O'Cull, L. (2012). *Embedded C Programming and the Atmel AVR*. Cengage Learning.

Microchip Technology Inc. (2022). *PIC16F877A Datasheet*. Retrieved from https://www.microchip.com

Valvano, J. W. (2012). *Embedded Systems: Introduction to Microcontrollers*. CreateSpace Independent Publishing Platform.

Pont, M. J. (2002). *Embedded C*. Addison-Wesley.