



UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO  
FACULTAD DE INGENIERIA  
LABORATORIO DE MICROCOMPUTADORAS

## PRÁCTICA 8: PROGRAMACIÓN EN C

**Alumnos:**

**Alcantar Correa Vianey.**

Grupo Teoria (3)

**Sánchez Rosas Alexis Alejandro.**

Grupo Teoria (1)

**Velazquez Martínez Karla Andrea.**

Grupo Teoria (4)

**Profesor: Moisés Melendez Reyes**

**Grupo: 01**

FECHA DE ENTREGA: 13/05/2024

## Objetivo.

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

## Introducción

La programación de microcontroladores PIC utilizando el lenguaje C y el PIC C Compiler de CCS representa un enfoque poderoso y flexible para el desarrollo de sistemas embebidos. Este compilador, diseñado específicamente para la familia de microcontroladores PIC de Microchip, permite a los desarrolladores aprovechar las ventajas de un lenguaje de alto nivel como C para implementar funcionalidades complejas de manera eficiente y con código más legible. La capacidad de C para manejar operaciones a nivel de hardware a través de estructuras de alto nivel, combinado con las optimizaciones específicas del compilador de CCS, facilita la creación de aplicaciones robustas que pueden incluir desde la manipulación de entradas/salidas digitales hasta la comunicación avanzada por puerto serie.

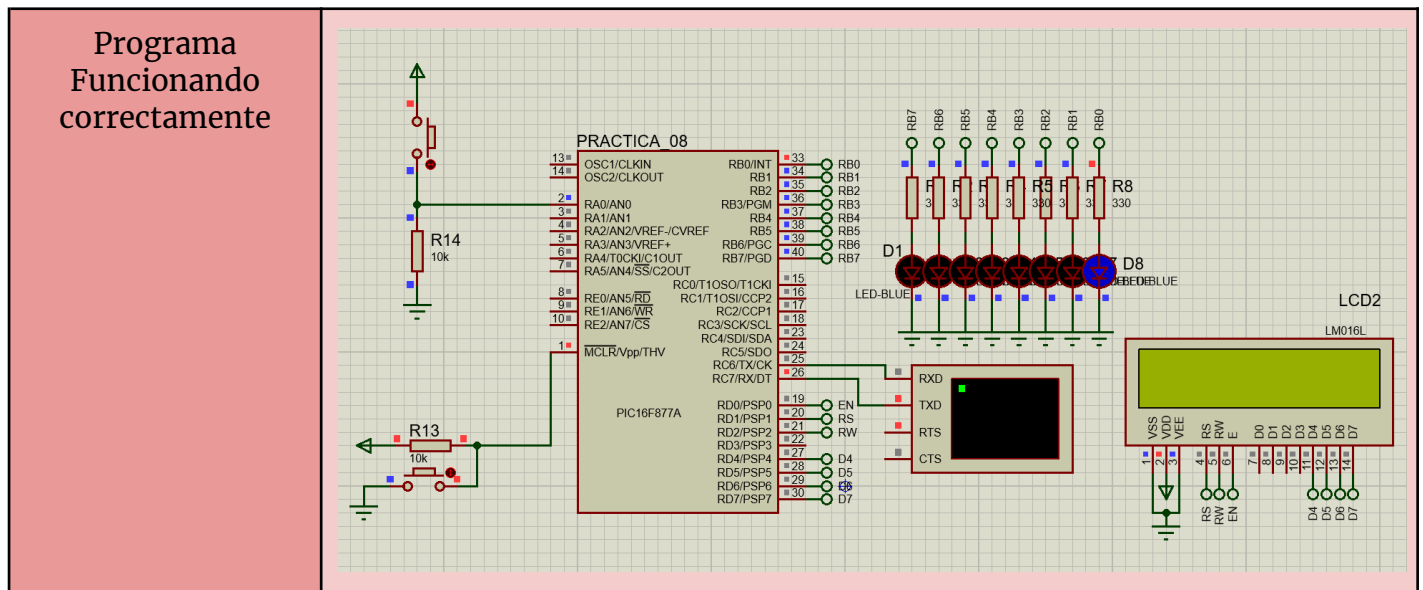
## Desarrollo de la práctica

### Ejercicio 1

Escribir, comentar, compilar el siguiente programa usando el ambiente del PIC C Compiler y comprobar el funcionamiento.

Comentado el código

```
1 #include <16f877.h>
2 #fuses HS,NOPROTECT, //Alta velocidad, no queremos proteger el código
3 #use delay(clock=20000000) //Reloj de 20 MHz
4 #org 0x1F00, 0xFFFF void loader16F877(void) {} //A partir de la 1f00 y hasta la 1fff vamos a reservar una función vacía
5
6 void main(){
7     while(1){
8         output_b(0x01); //Mandamos un 0b00000001 al portb
9         delay_ms(1000); //retraso de 1 segundo
10        output_b(0x00); //Mandamos un 0b00000000 al portb
11        delay_ms(1000); //retraso de 1 segundo
12    } //while
13 } //main
```

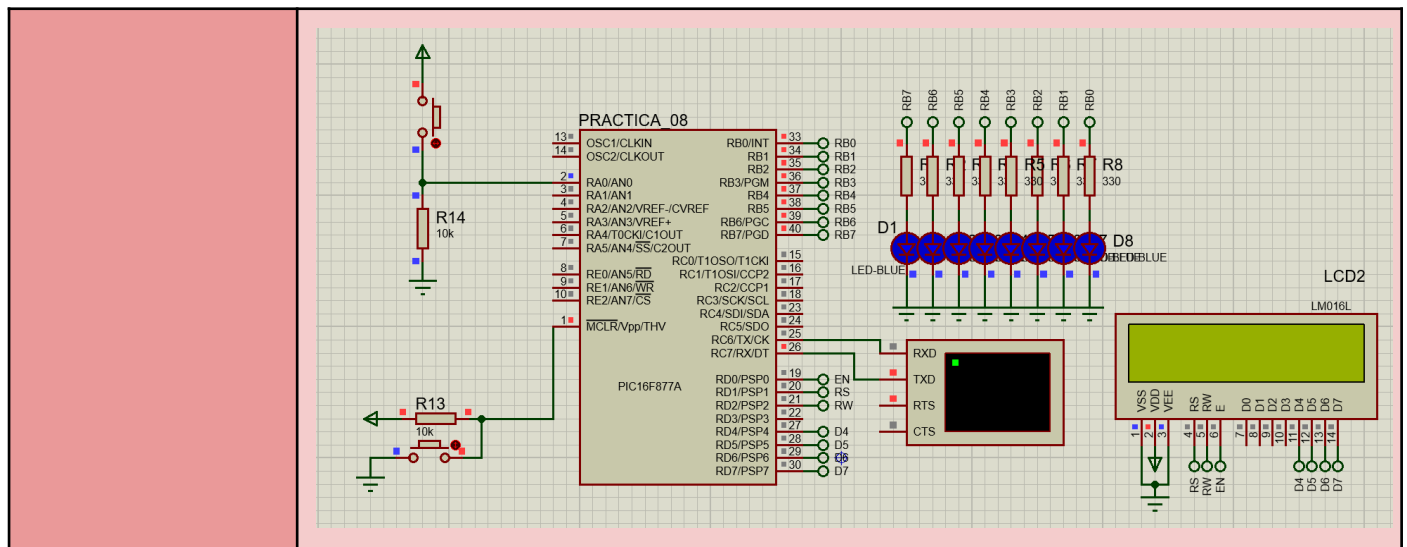


Este código es la implementación en el lenguaje C para encender y apagar un led con un retardo de 1 segundo entre cada cambio de estado.

## Ejercicio 2

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

<p style="text-align: center;">Propuesta de solución</p>	<pre> 1  #include &lt;16f877.h&gt; 2 3  #fuses HS,NOPROTECT, //Alta velocidad, no queremos proteger el código 4  #use delay(clock=20000000) //Reloj de 20 Mhz 5  #org 0x1f00, 0x1fff void loader16f877(void) {} //A partir de la 1f00 y hasta la 1fff vamos a reservar una función vacía 6 7  void main(){ 8      while(1){ 9          output_b(0xFF); 10         delay_ms(1000); 11         output_b(0x00); 12         delay_ms(1000); 13     } //while 14 } //main                     </pre>
<p style="text-align: center;">Funcionando correctamente</p>	



El cambio en el código es bastante sencillo, solo basta con indicar el puerto B que encienda todos los leds en lugar de solo el primero.

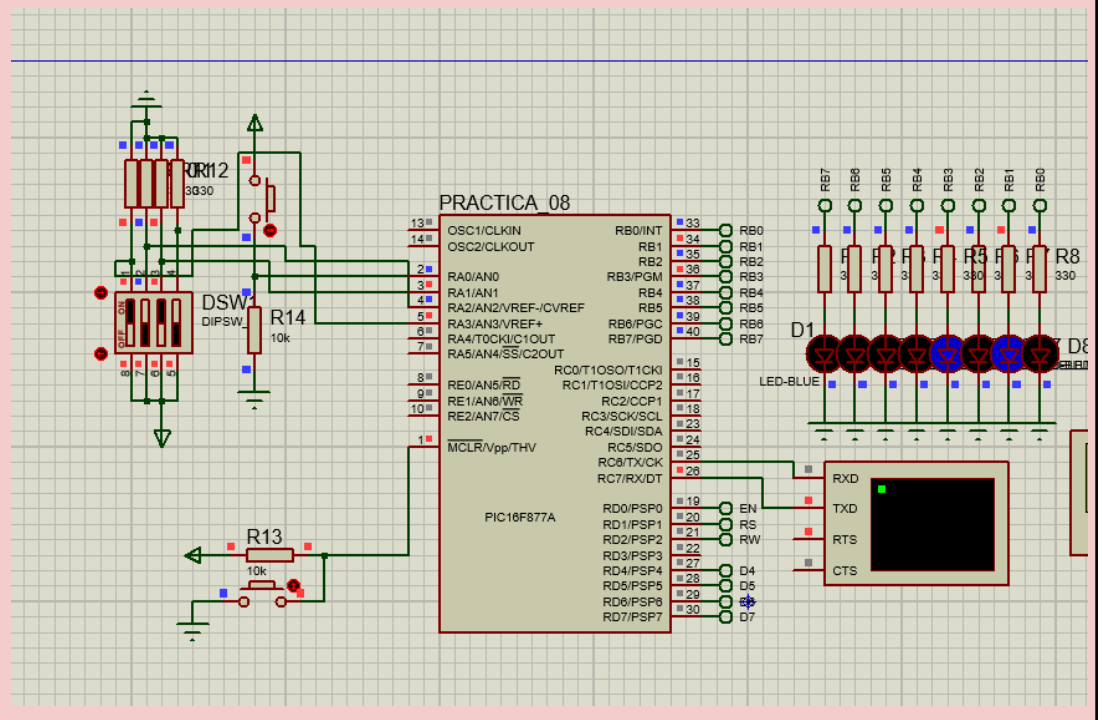
### Ejercicio 3

Escribir, comentar, compilar el siguiente programa usando el ambiente del PIC C Compiler y comprobar el funcionamiento.

#### Propuesta de solución

```
#include <16f877.h>
#fuses HS,NOPROTECT,
#use delay(clock=20000000)
#org 0x1F00, 0x1FFF void loader16F877(void) {}
int var1;
void main(){
    while(1){
        var1=input_a(); //Guardamos los 8 bits del puerto portA en una variable de tipo entero
        output_b(var1); //Mandamos el valor de la variable como salida en el puerto B
    }
}
```

Funcionando correctamente



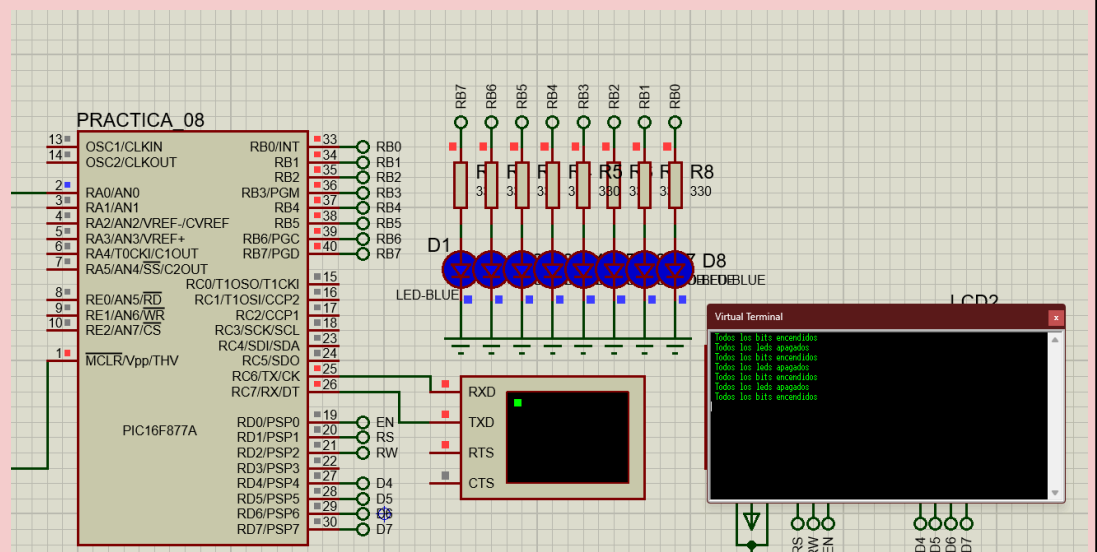
Este sencillo ejercicio nos ayuda a familiarizarnos con la sintaxis para trabajar con los puertos paralelos tanto en entrada como en salida.

#### Ejercicio 4

Escribir, comentar, compilar, el siguiente programa usando el ambiente del PIC C Compiler y comprobar el funcionamiento.

Propuesta de  
solución

```
#include <16f877.h>
#fuses HS,NOPROTECT,
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
#org 0x1F00, 0x1FFF void loader16F877(void) {}
void main(){
    while(1){
        output_b(0xff); //
        printf(" Todos los bits encendidos \n\r");
        delay_ms(1000);
        output_b(0x00);
        printf(" Todos los leds apagados \n\r");
        delay_ms(1000);
    } //while
} //main
```

Funcionando  
correctamente


Trabajar con dispositivos de E/S en ensamblador resultó ser muy difícil y representó diversas dificultades, fueron ejercicios que nos ayudaron a entender mejor la estructura y funcionamiento de nuestro microcontrolador. La inclusión de librerías en C nos permite configurar y controlar los dispositivos de manera sencilla y eficaz sin necesidad de escribir demasiado código.

## Ejercicio 5

Escribir, comentar, compilar, el siguiente programa usando el ambiente del PIC C Compiler y comprobar el funcionamiento.

Nota: La biblioteca lcd.c asigna las terminales para uso del LCD, en la plataforma usada se ha conectado al puerto D; también permite usar el puerto B para las señales de control; en este caso agregar previo a incluir la librería #define use\_portb\_lcd true.

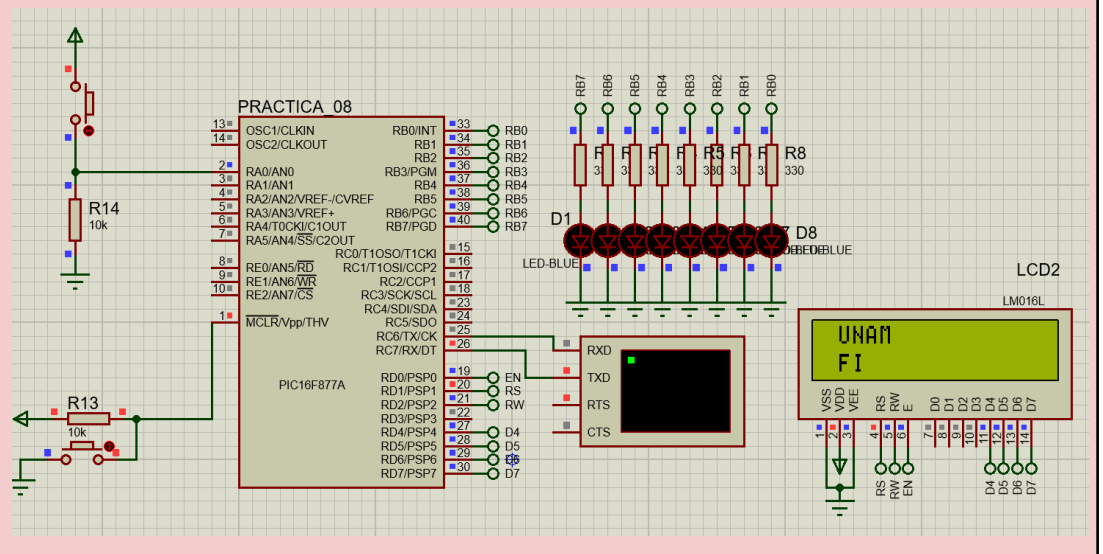
Propuesta de  
solución

```
#include <16F877.h>
#fuses HS, NOWDT, NOPROTECT, NOLVP
#use delay(clock=20000000)
#include <lcd.c>
void main() {

    lcd_init();

    while( TRUE ) {
        lcd_gotoxy(1,1);
        printf(lcd_putc, " UNAM \n ");
        lcd_gotoxy(1,2);
        printf(lcd_putc, " FI \n ");
        delay_ms(300);
    }
}
```

Ensamblado correctamente



Esté código nos permite utilizar las dos filas de la pantalla LCD y nos muestra la forma tan sencilla que tenemos para mandar caracteres sin la necesidad de considerar los ascii.

### Ejercicio 6

Realizar un programa empleando el compilador de C, para ejecutar las acciones mostradas en la siguiente tabla, estas son controladas a través del puerto serie; usar retardos de ½ segundos.

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





## Propuesta de solución

```
#include <16f877.h> //Librería del controlador PIC16f877
#fuses HS,NOPROTECT, //Directivas de ADCON1
#use delay(clock=2000000) //Velocidad del reloj de la tarjeta
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7) //Directivas para poder trabajar con el puerto serie, baud rate de 38,400, PIN 6 para la transmisión de datos y PIN 7 para la recepción de datos
#org 0x1F00, 0x1FFF void loader16f877(void) {} //for the 8k 16F876/7

void main() {
    char c; //variable para leer el carácter de entrada
    int dato,i; //variable con el dato de salida y para el contador de los ciclos para los retardos
    //delay_ms(500); // Retardo inicial de 500 ms

    while(1) {
        printf("Indica el dato\n"); //Pedimos el dato al usuario para que este seleccione la acción a realizar
        c=getch(); //Se obtiene el carácter del teclado con la función getch()
        putc(c); //Se envía el carácter al flujo de datos
        switch(c) { //switch case con el valor de c
            case '0': //c = 0
                printf("\n\tLEDs apagados"); //Se apagan los LEDs del puerto de salida
                output_b(0x00); // Todos los bits apagados
                delay_ms(500); //retardo de medio segundo
                break;
            case '1': //c = 1
                printf("\n\tLEDs encendidos"); //Se prenden los LEDs del puerto de salida
                output_b(0xff); /// Todos los bits encendidos
                delay_ms(500); //retardo de medio segundo
                break;

            case '2': //c = 2
                printf("\n\tCorrimiento a la derecha"); //Se indica en pantalla que se hace un corrimiento a la derecha
                dato=0x80; //dato=80h
                for(int i=0;i<=7;i++)
                {
                    output_b(dato); //se manda a la salida dato
                    delay_ms(500); //retardo de medio segundo
                    dato=dato>>1; //corrimiento a la derecha de un bit
                }
                break;
            case '3': //c = 3
                printf("\n\tCorrimiento a la izquierda"); //Se indica en pantalla que se hace un corrimiento a la izquierda
                dato=0x01; //dato=01h
                for(i=0;i<=7;i++)
                {
                    output_b(dato); //se manda a la salida dato
                    delay_ms(500); //retardo de medio segundo
                    dato=dato<<1; //corrimiento a la izquierda de un bit
                }
                break;

            case '4': //c = 4
                printf("\n\tCorrimiento a la derecha"); //Se indica en pantalla que se hace un corrimiento a la derecha
                dato=0x80;
                for(i=0;i<=7;i++)
                {
                    output_b(dato); //se manda a la salida dato
                    delay_ms(500); //retardo de medio segundo
                    dato=dato>>1; //corrimiento a la derecha de un bit
                }
                printf("\n\tCorrimiento a la izquierda"); //Se indica en pantalla que se hace un corrimiento a la izquierda
                dato=0x01;
                for(i=0;i<=7;i++)
                {
                    output_b(dato); //se manda a la salida dato
                    delay_ms(500); //retardo de medio segundo
                    dato=dato<<1; //corrimiento a la izquierda de un bit
                }
                break;
            case '5': //c = 5
                output_b(0x00); // Apagar todos los bits
                delay_ms(500); // Retardo de 500 ms
                output_b(0xff); // Encender todos los bits
                delay_ms(500); // Retardo de 500 ms
                output_b(0x00);
                break;
            default:
                // Accion por defecto en caso de comando no reconocido
                break;
        }
    }
}
```

## Pseudocódigo

Iniciar:

Configurar microcontrolador con:

- Oscilador HS
- Sin protección de escritura
- Sin protección contra bajos voltajes
- Velocidad del reloj de 20 MHz
- Comunicación serial a 9600 baudios
- PIN C6 para transmisión de datos
- PIN C7 para recepción de datos

Inicializar LCD



Esperar 500 ms (Retardo inicial)

Mientras verdadero:

Mostrar en pantalla "Indica el dato"

Leer un carácter 'c' de la entrada serial

Enviar el carácter 'c' de vuelta al flujo de datos

Evaluar 'c':

Si 'c' es '0':

Mostrar "Leds apagados"

Apagar todos los LEDs

Esperar 500 ms

Si 'c' es '1':

Mostrar "Leds encendidos"

Encender todos los LEDs

Esperar 500 ms

Si 'c' es '2':

Mostrar "Corrimiento a la derecha"

Iniciar con el bit más alto encendido

Repetir 8 veces:

Mostrar el estado actual de los LEDs

Desplazar los LEDs a la derecha

Esperar 500 ms

Si 'c' es '3':

Mostrar "Corrimiento a la izquierda"

Iniciar con el bit más bajo encendido

Repetir 8 veces:

Mostrar el estado actual de los LEDs

Desplazar los LEDs a la izquierda

Esperar 500 ms

Si 'c' es '4':

Realizar un ciclo de desplazamiento:

Corrimiento a la derecha desde el bit más alto

Corrimiento a la izquierda desde el bit más bajo

Mostrar cada estado intermedio y esperar 500 ms entre cambios

Si 'c' es '5':

Mostrar "Apagado y encendido de LEDs"

Apagar todos los LEDs

Esperar 500 ms

Encender todos los LEDs

Esperar 500 ms

Apagar todos los LEDs

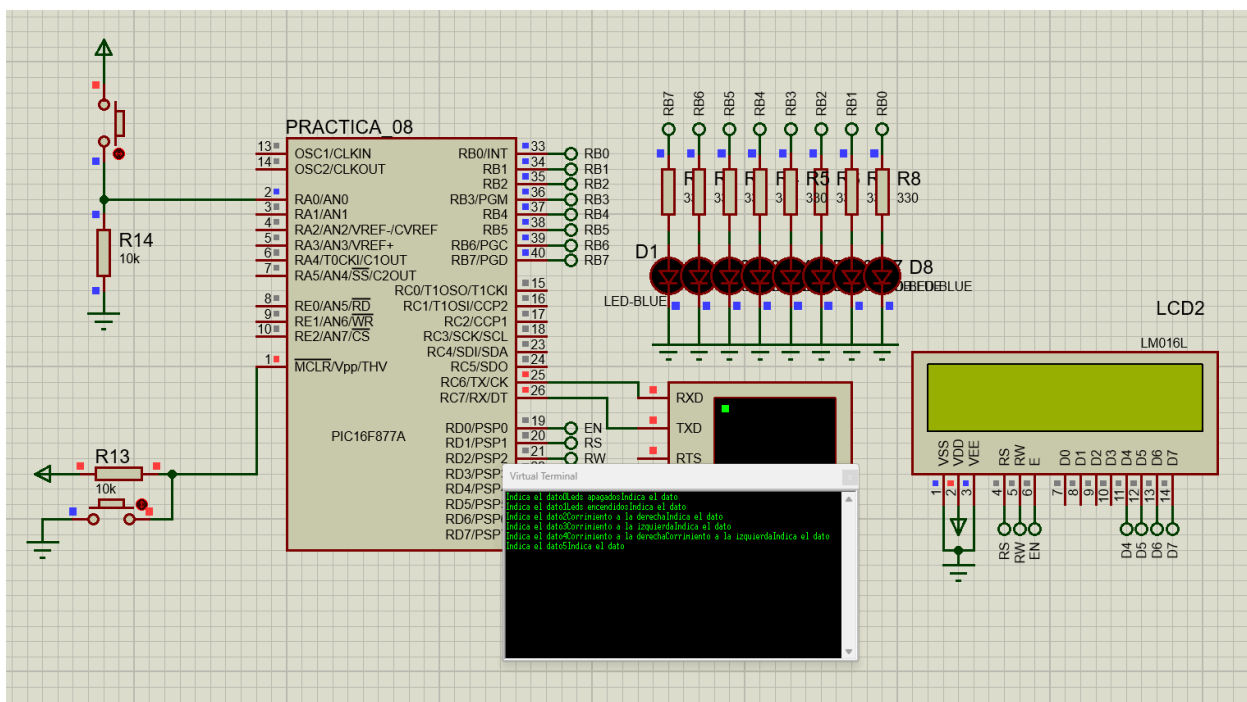
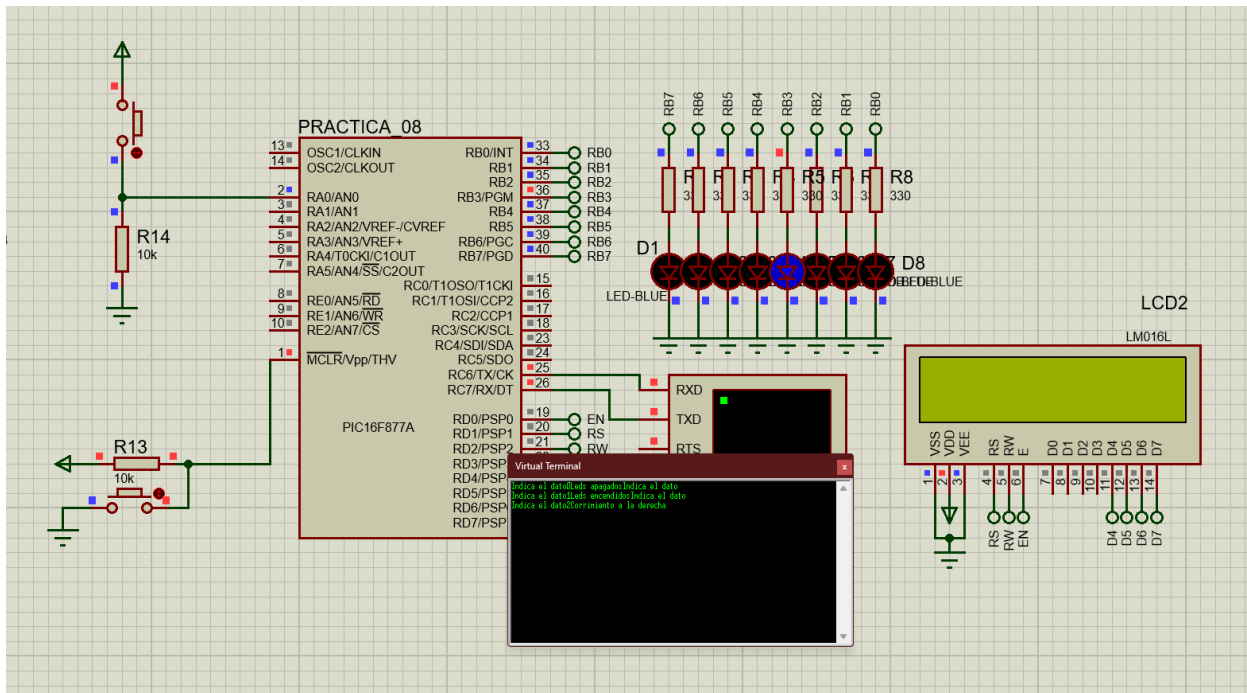
Si 'c' es otro carácter:

Ignorar y esperar la próxima entrada

Fin Mientras

Fin

### Resultados:



## Ejercicio 7

Realizar un programa que muestre en un Display de Cristal Líquido, la cantidad de veces que se ha presionado un interruptor, el cual está conectado a la terminal A0. El despliegue a mostrar es:

- Primer línea y 5 columna; la cuenta en decimal
- Segunda línea y 5 columna; la cuenta en hexadecimal

### Pseudocódigo

```
Iniciar:
    Inicializar el LCD

Mientras Verdadero:
    Leer estado del puerto A en la variable 'var'

    Si 'var' es diferente de 0:
        Incrementar 'counter'
        Mientras 'var' sea diferente de 0:
            Leer nuevamente el estado del puerto A en 'var'
            Esperar 10 ms para evitar rebotes del botón o ruido

    Posicionar cursor en LCD en columna 5, fila 1
    Mostrar 'counter' en formato decimal en el LCD

    Posicionar cursor en LCD en columna 5, fila 2
    Mostrar 'counter' en formato hexadecimal en el LCD

    Esperar 200 ms antes de la siguiente lectura

Fin Mientras
Fin
```

### Implementación:

```
#include <16F877.h>
#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=20000000)
#include <lcd.c>
int counter;

void main() {

    lcd_init();          //Inicializamos el LCD

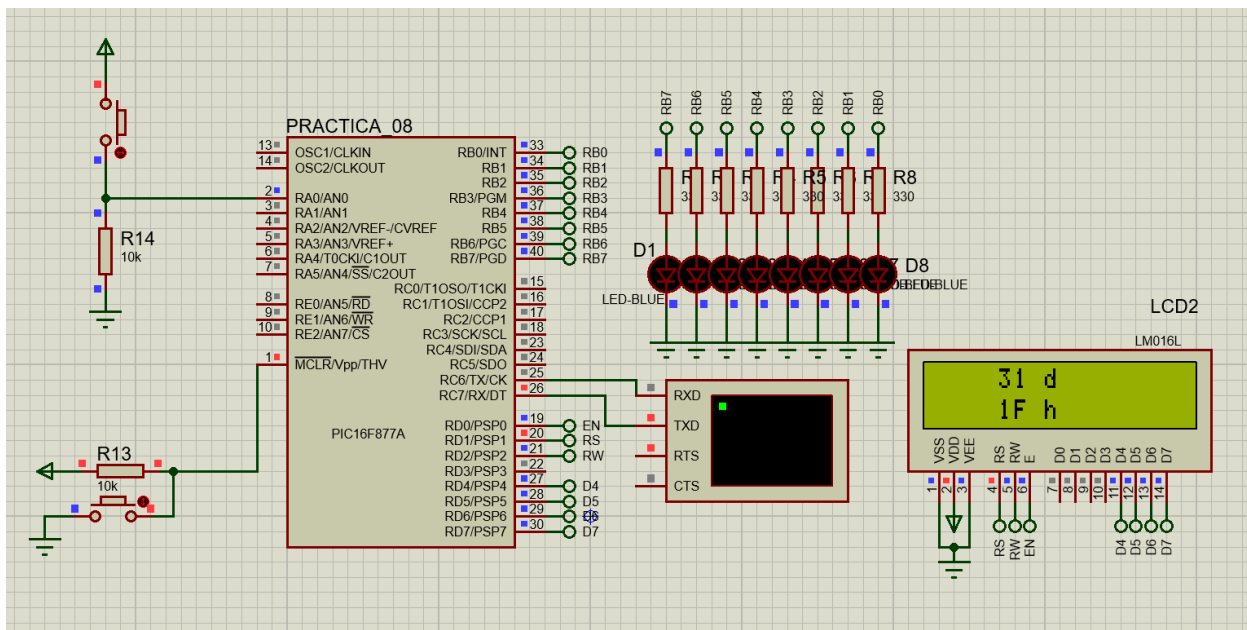
    while( TRUE ) {
        int var = input_a();
        if(var != 0){
            counter++;
            while(var != 0){
                var = input_a();
                delay_ms(10);
            }
        }
    }
}
```

```

    }
}
lcd_gotoxy(5,1);
printf lcd_putc,"%d d  ", counter); //Mostramos el contador en decimal
lcd_gotoxy(5,2);
printf lcd_putc,"%X h  ", counter); //Mostramos el contador en hexadecimal
delay_ms(200);
}
}

```

## Resultados:



## Análisis técnico

1. ¿Se puede comprobar que la solución producida funciona?

Gracias a la implementación de circuitos físicos y las simulaciones pudimos observar el correcto funcionamiento de las actividades realizadas durante la clase.

2. ¿Se alcanzó el objetivo?

Sí, logramos implementar una solución efectiva para cada actividad. Además, pudimos verificar el correcto desempeño de nuestros códigos gracias a los circuitos físicos como simulados.

## Ejercicio 6

## ¿Cuál es el flujo interno de los datos?

### 1. Inicialización y Configuración Inicial:

- Configuraciones de fusibles, reloj del sistema y comunicación serial se establecen al inicio.
- Se espera un retardo inicial de 500 ms, aunque esta línea está comentada en el código.

### 2. Entrada de Datos:

- Se solicita al usuario que ingrese un dato a través de la interfaz serial utilizando ``printf("Indica el dato\n");``.
- El carácter ingresado se captura mediante ``c = getch();``, que lee un carácter desde el puerto serial.

### 3. Proceso y Respuesta:

- El carácter capturado se reenvía de vuelta al puerto serial para confirmación usando ``putc(c);``.
- Dependiendo del carácter ingresado (``c``), se ejecutan diferentes acciones controladas por una estructura ``switch``:
  - '0': Apaga todos los LEDs conectados al puerto B.
  - '1': Enciende todos los LEDs.
  - '2' y '4': Realizan corrimientos de bits a la derecha de un dato inicial.
  - '3' y '4': Realizan corrimientos de bits a la izquierda del mismo dato tras corrimiento a la derecha.
  - '5': Apaga y enciende todos los LEDs en secuencia.
- Cada comando que afecta a los LEDs espera 500 ms tras su ejecución para proporcionar un retardo visual.

### 4. Repetición Continua:

- Este proceso se repite indefinidamente en un bucle ``while(1)``, permitiendo entradas continuas y acciones repetitivas basadas en las entradas.

## ¿Cuáles fueron los modos de direccionamiento utilizados?

### 1. Inmediato:

- Uso de constantes literales como ``0x00``, ``0xff``, ``0x80``, ``0x01`` para manipular directamente los datos en instrucciones de operación y asignación. Estos valores son utilizados directamente en las funciones, como en ``output_b(0xff);``.

### 2. Directo:

- Acceso directo a puertos de hardware específicos, como en ``output_b(dato);``, donde ``dato`` es manipulado y luego enviado directamente al puerto B del microcontrolador para controlar los LEDs.

### 3. Registro:

- Manipulación de variables locales como ``dato`` y ``c`` dentro de funciones y estructuras de control. Aunque técnicamente en C estas son consideradas accesos a memoria directos o a

variables, en el contexto de un microcontrolador, cada variable requiere una dirección de memoria que es gestionada por el compilador.

## Ejercicio 7

### ¿Cuál es el flujo interno de los datos?

#### 1. Inicialización:

- El LCD se inicializa mediante ``lcd_init()``. Esto prepara el display para mostrar datos.

#### 2. Entrada de Datos:

- Se lee el estado de un puerto de entrada usando ``input_a()``. Esta función verifica si hay una señal (presumiblemente de un botón o sensor) en el puerto A del microcontrolador.

#### 3. Procesamiento de Datos:

- Si el puerto de entrada devuelve un valor no nulo (``var != 0``), indica que hay una entrada activa. En respuesta, se incrementa el valor de ``counter``.
- Se entra en un bucle interno mientras ``var`` sea diferente de cero, leyendo repetidamente el estado del puerto para asegurarse de que la entrada se ha desactivado. Durante este bucle, se introduce un pequeño retardo (``delay_ms(10)``) para manejar el rebote del botón o para evitar leer continuamente el puerto de entrada.

#### 4. Salida de Datos:

- El valor de ``counter`` se muestra en el LCD. Se presenta en dos formatos:
- Decimal: utilizando ``printf(lcd_putc, "%d d ", counter)``.
- Hexadecimal: utilizando ``printf(lcd_putc, "%X h ", counter)``.
- Estos datos se posicionan en diferentes lugares del LCD a través de ``lcd_gotoxy()``, que configura la posición del cursor en el LCD antes de imprimir el valor.

#### 5. Repetición:

- Después de mostrar los datos, el programa espera 200 milisegundos antes de comenzar la próxima iteración del bucle ``while``, permitiendo una actualización periódica de la pantalla y la lectura de nuevos datos de entrada.

### ¿Cuáles fueron los modos de direccionamiento utilizados?

#### 1. Directo:

- Cuando se usa `input_a()`, se está accediendo directamente al valor del puerto A del microcontrolador. Este es un ejemplo de direccionamiento directo, donde se accede directamente a un registro de hardware específico.

#### 2. Inmediato:

- Los valores literales como los utilizados en ``delay_ms(10)`` y ``delay_ms(200)`` son ejemplos de direccionamiento inmediato, donde se pasa un valor constante directamente a la función.

### 3. (Indirectamente) Registro Directo:

- El uso de ``counter`` en ``printf(lcd_putc, "%d d ", counter)`` y en incrementos (``counter++``) implica el acceso a una variable almacenada en una ubicación de memoria específica. Aunque el modo de acceso es directo a la variable, en un nivel más bajo, podría considerarse un acceso indirecto dado que se manipula una dirección de memoria donde se almacena ``counter``

## Conclusiones

### **Alcantar Correa Vianey:**

Aprender a programar en C después de haber utilizado ensamblador durante el semestre ha sido tremendamente beneficioso para mí. Con C, he logrado reducir significativamente el número de líneas de código y he encontrado que su sintaxis es más clara. Además, al ser un lenguaje de un nivel más alto, facilita enormemente la declaración de variables y la configuración de puertos de entrada y salida. Aunque el ensamblador sigue siendo crucial para entender cómo funcionan los programas en C, ahora me resulta más sencillo convertir programas de C a ensamblador utilizando diversas directivas e instrucciones. También he descubierto funciones útiles en la biblioteca `.h` y directivas para la comunicación serie-asíncrona. A través de esta experiencia, he aprendido a utilizar el PIC C Compiler, una herramienta eficaz que traduce programas de C a ensamblador, lo que ha sido una adición valiosa a mi conjunto de herramientas de programación.

### **Sanchez Rosas Alexis Alejandro:**

El cambio en el lenguaje de programación fue sin lugar a dudas un aprendizaje refrescante y al cual conseguimos adaptarnos con facilidad gracias al conocimiento adquirido previamente en el lenguaje ensamblador. Este conocimiento nos ayudó a entender cómo funcionaba el microcontrolador y todos los conceptos y herramientas necesarias para poder trabajar con él con cualquier tipo de herramienta. La programación en C se encuentra varios niveles por encima de la programación en ensamblador y es interesante ver como todo el código puede ser convertido en un archivo HEX e interpretado por el microcontrolador, gracias al lenguaje somos capaces de desarrollar soluciones más óptimas y eficientes en varios sentidos; el uso de bibliotecas y la posibilidad de poder realizar operaciones más complejas que la suma y la resta sin lugar a dudas nos facilitó bastante el trabajo al momento de diseñar e implementar los códigos de las actividades de la práctica.



**Velazquez Martinez Karla Andrea:**

En esta práctica, hemos explorado y profundizado en la programación en C para microcontroladores, específicamente el PIC16F877, enfocándonos en la utilización efectiva del puerto serie para la visualización y el control de dispositivos externos como LEDs. A través del desarrollo de varios programas, se ha demostrado cómo el uso del puerto serie no solo facilita la interacción con el usuario, sino que también permite un control preciso y visualización en tiempo real del estado del hardware. La capacidad de programar en C, un lenguaje de alto nivel, y aplicarlo en un contexto de microcontrolador ha simplificado significativamente la gestión de la entrada/salida, mostrando su eficacia en la reducción de la complejidad y en la mejora de la claridad del código, lo que es crucial para aplicaciones de control en tiempo real.

**Bibliografía**

1. Microside Technology. (n.d.). LCD con PIC16F887 o PIC16F877A - PIC C Compiler CCS. Recuperado de <https://docs.microside.com/practicas/pic-c-compiler-ccs/pic16f887-or-pic16f877a/10-lcd>
2. Microside Technology. (n.d.). PIC C Compiler CCS - Manuales de usuario. Recuperado de <https://docs.microside.com/manuales-de-usuario/ides/pic-c-compiler-ccs>
3. Control Automático Educación. (n.d.). PIC C Compiler. Recuperado de <https://controlautomaticoeducacion.com/microcontroladores-pic/pic-c-compiler/>
4. Microchip Technology Inc. 2003. PIC16F87XA Data Sheet. Recuperado de <https://ww1.microchip.com/downloads/en/devicedoc/39582b.pdf>