



## OBJETIVO:

Diseñar e implementar sistemas microcontrolados.

## INDICE

1. [Introducción.](#)
2. [Manejo de Puertos. \(Prácticas A\)](#)
3. [LCD \(Prácticas B\).](#)
4. [Convertidor ADC \(Prácticas C\).](#)
5. [PWM \(Prácticas D\).](#)
6. [Convertidor DAC \(Prácticas E\).](#)
7. [Comunicación Serial \(Prácticas F\).](#)
8. [Comunicación Inalámbrica. \(Prácticas G\)](#)
9. [Aplicaciones \(Prácticas H\).](#)
10. [Guía de Programas Demo.](#)
11. [Guía de Actividades.](#)
12. [Apéndice.](#)





## CONTENIDO TEMÁTICO

### **1. Introducción.**

- 1.1 Definiciones Globales.**
- 1.2 Declaración de Subrutinas o métodos.**
- 1.3 Funciones definidas para el microcontrolador.**
- 1.4 Operadores.**
- 1.5 Sentencias y ciclos.**

### **2. Manejo de Puertos. (Prácticas A)**

### **3. LCD (Prácticas B).**

### **4. Convertidor ADC (Prácticas C).**

### **5. PWM (Prácticas D).**

### **6. Convertidor DAC (Prácticas E).**

### **7. Comunicación Serial (Prácticas F).**

### **8. Comunicación Inalámbrica. (Prácticas G)**

### **9. Aplicaciones (Prácticas H).**

### **10. Guía de Programas Demo.**

### **11. Guía de Actividades.**

### **12. Apéndice.**





## CONTENIDO TEMÁTICO

1. Introducción.
2. **Manejo de Puertos. (Prácticas A)**
  - 2.1 Configuración de Puertos.
  - 2.2 Modo Bit, Entrada y Salida.
    - 2.2.1. Consideraciones en Práctica.
  - 2.3 Modo Byte, Entrada y Salida.
    - 2.3.1. Asignación Directa.
    - 2.3.2. Asignación Indirecta.
    - 2.3.3. Ejercicios prácticos.
  - 2.4 Uso de Arreglos.
  - 2.5 Uso de Display 7 segmentos.
3. LCD (Prácticas B).
4. Convertidor ADC (Prácticas C).
5. PWM (Prácticas D).
6. Convertidor DAC (Prácticas E).
7. Comunicación Serial (Prácticas F).
8. Comunicación Inalámbrica. (Prácticas G)
9. Aplicaciones (Prácticas H).
10. Guía de Programas Demo.
11. Guía de Actividades.





## GUÍA DE PROGRAMAS DEMO

1. [Uso de #define.](#)
2. [Generación de Métodos Básicos.](#)
4. [Entrada y Salida de un solo Bit.](#)
5. [Espera a Pulsar haciendo una Tarea.](#)
6. [Espera a Pulsar sin realizar ninguna Acción.](#)
7. [Esperar a Pulsar, Esperar a Soltar, Realizar Acción.](#)
8. [Salida de Datos – Asignación Directa.](#)
9. [Salida de Datos – Asignación Indirecta.](#)
10. [Convertidor de Decimal a Binario.](#)
11. [Generando un Corrimiento de Datos.](#)
12. [Generando Corrimientos con Bit de Inicio por Pulsador.](#)
13. [Agregando Bits a un Corrimiento.](#)
14. [Eliminando Bits a un Corrimiento.](#)
15. [Generando un Corrimiento de Datos por medio de Arreglos.](#)
16. [Manejo Básico de Display.](#)
17. [Manejo de Display por medio de Arreglos.](#)
18. [Contador Continuo de 0 a 20.](#)
19. [Intercambiando un Display de Ánodo Común por un Cátodo Común.](#)





## **GUÍA DE ACTIVIDADES**

1. [Actividad 01 –Realiza una Función Nula.](#)
2. [Actividad 02 –Realiza una Función con Respuesta.](#)
3. [Actividad 03 – Puerto Reflejado.](#)
4. [Actividad 04 – Esperando Pulsar para Detener un Corrimiento.](#)
5. [Actividad 05 – Esperando Pulsar para Generar Intermitente.](#)
6. [Actividad 06 – Espera a Pulsar, Espera a Soltar, Encendido Gradual y Apagado Gradual.](#)
7. [Actividad 07 – Puerto Espejo.](#)
8. [Actividad 08 - Convertidor de Binario a Decimal.](#)
9. [Actividad 09 – Va y viene.](#)
10. [Actividad 10 – Aplausos!!!](#)
11. [Actividad 11 – Brincando la cerca.](#)
12. [Actividad 12 – Rebotando la pelota.](#)
13. [Actividad 13 – Indicando el camino.](#)
14. [Actividad 14 – Contando Pasajeros.](#)
15. [Actividad 15 – Contador continuo de 0 a 99.](#)
16. [Actividad 16 – Convertidor de Binario a Decimal Extendido.](#)
17. [Actividad 17 – Eliminando el Efecto Rebote.](#)
18. [Actividad 18 – Contador continuo de 0 a 999 up/down.](#)





## INTRODUCCIÓN

Existen diferentes gamas de Microcontroladores MICROCHIP, cada familia tiene características similares como memoria RAM, EEPROM, ROM, además de la distribución de sus pines y las funciones de cada uno de ellos. Un microcontrolador puede tener funciones adicionales como módulos de comunicación Serial, SPI, I2C, además de pines para adquisición de señales analógicas. PWM, etc. Independientemente de estas características, la parte vital para usarlas es el dominar estructuras de programación para mandarlas llamar.

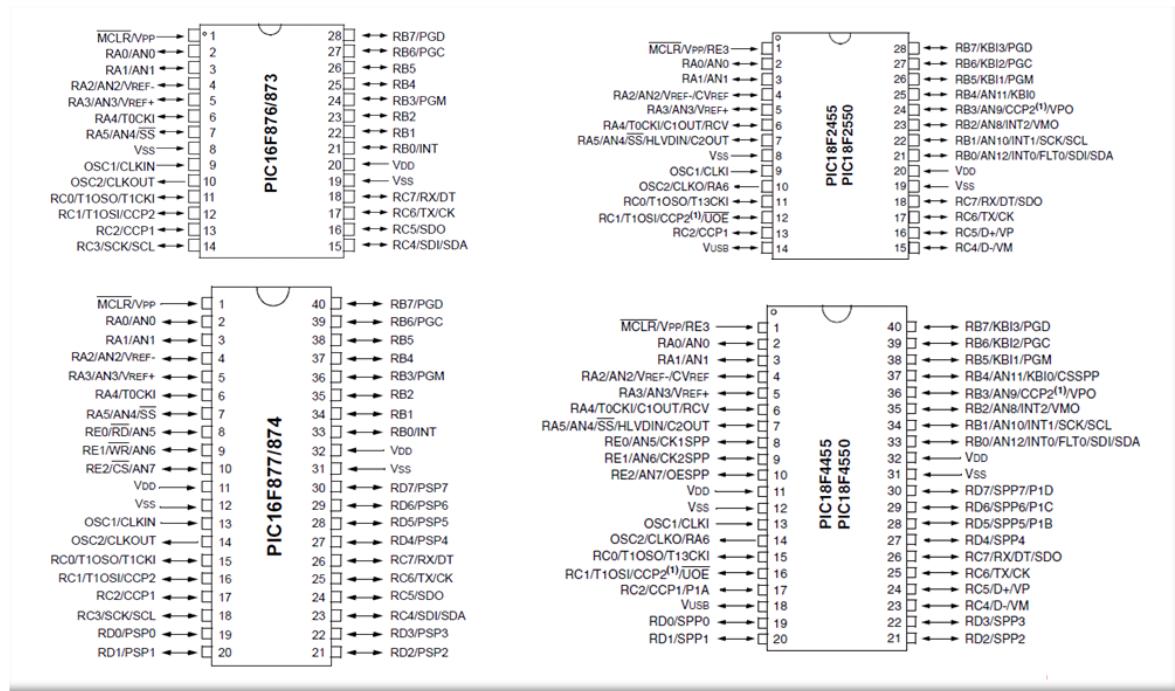


Figura 1. Gamas de Microcontroladores PIC.

Cuando uno se enfrenta a la generación de tecnología por medio de sistemas embebidos, el conocimiento de la estructura interna de un microcontrolador o microprocesador(Freescale, Philips, Cypress, Texas Instruments, etc.) es vital para lograr la configuración adecuada de cada bloque de elementos, en el caso de los Microcontroladores PIC y Arduino podemos saltarnos este conocimiento si es que conocemos estructuras de programación, ya que algunas de las instrucciones manejadas para estas tecnologías solo hay que conocer las diferentes posibilidades de los parámetros en una función en C.



## Manual – Aplicaciones Prácticas con Microcontroladores PIC

Elaborado por: M.I.E. Hugo Antonio Méndez Guzmán



Por este motivo este curso está orientado a la generación de aplicaciones prácticas en ambas Tecnologías basándose en el conocimiento de estructuras de programación en C.

De lo anterior un esqueleto de programación en C para Microcontroladores PIC puede ser la siguiente:

```
#include <16f877.h>
#FUSES XT,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )

// Definiciones Globales

// Definición de Variables Globales

// Declaración de Subrutinas o Métodos

// Programa Principal
void main ()
{
    // Definiciones de Variables Locales

    // Configuración de Puertos

    // Bucle Principal
    while (1){
        // Instrucciones del programa

        } // endwhile
} //endmain
```

Esqueleto que para fines prácticos dependerás de él para iniciar cualquier programa en un microcontrolador, sin depender de herramientas como MPLAB para el debugueo, tus herramientas serán la programación en C y Proteus solamente, inclusive para la detección de errores o partes del programa donde quieras verificar el funcionamiento de tu código.

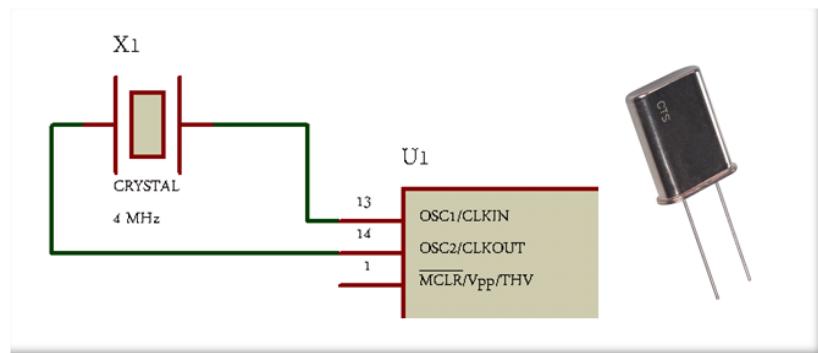
Como es sabido, las primera tres líneas del esqueleto nos servirán para configurar las características especiales del microcontrolador como oscilador, WatchdogTimer, Power-up Protect, Brown-out Reset, Protección de Código, etc (en el desarrollo del curso irán aumentando las directivas para habilitar funciones y las iremos modificando según se vaya ocupando).





Las partes de las que nos preocuparemos de modificar en este esqueleto frecuentemente serán las partes de “Definición de variables”, “Configuración de Puertos” y el “Bucle Principal”. Cabe mencionar que un microcontrolador actúa de manera secuencial por lo que normalmente realizaría una tarea solo una vez. Es por esto que aquello que nosotros deseemos que realice una sola vez lo pondremos en código antes del “Bucle principal” y lo que deseemos que constantemente este realizando el microcontrolador lo ubicaremos en seguida de “Instrucciones del programa”.

Por otra parte, un microcontrolador para realizar cada paso de su programación requiere de una señal pulsante, una señal de reloj que le este indicando cada cuando debe ejecutar cada instrucción que uno le programe, dependiendo de la velocidad a la que deseemos que trabaje el microcontrolador será la velocidad del oscilador que pondremos físicamente en sus pines OSC1 y OSC2,



**Figura 2. Señal de reloj.**

sin embargo el simple hecho de colocarlo físicamente no implica que sea suficiente para que el microcontrolador tome la señal de reloj de este, es necesario el configurarlo también en el programa en C.

#### Configuraciones Típicas del Oscilador:

XT – Oscilador menor a 4MHz.

HS – Oscilador mayor a 4MHz.





### **Power-up Timer**

**PUT** – Espera a que el voltaje suministrado al microcontrolador se estabilice, evitando errores en el funcionamiento de algunas instrucciones (por ejemplo cuando se adquiere una señal analógica donde su referencia depende del voltaje de alimentación).

**NOPUT** – Deshabilita el efecto anterior.

### **Delay**

Permite establecer la frecuencia de oscilación del cristal que coloquemos al microcontrolador. Nota: El valor colocado en esta directiva no tiene efecto si la configuración del oscilador no es la correcta.

## **DEFINICIONES GLOBALES**

Nos permite establecer etiquetas a valores numéricos de forma de tener una representación más práctica para el desarrollador. De tal manera que si queremos usar el 255, en vez de colocarlo como número lo podemos llamar por el nombre de su etiqueta. Para poder realizar esto se tendrá que programar:

Ejemplo:

```
#define maximo 255  
#define minimo 0
```

En programación embebida estas definiciones nos pueden servir para cambiar también el nombre de instrucciones para facilitar su uso a los practicantes

Ejemplo:

```
#define prende_pin(x) put_high(x)  
#define apaga_pin(x) put_low(x)
```

Donde por medio de estas definiciones le estamos dando a entender que la instrucción `put_high` puede ser utilizada o referenciada por medio de `prende_pin`. De esta





manera el programador puede tener una idea clara del código que está generando sin depender tanto de la forma establecida por el fabricante para la programación de un microcontrolador.

## Demo 01

```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )
// Definiciones Globales
#define prende_pin(x) output_high(x)
#define apaga_pin(x) output_low(x)

// Definiciones de Variables Globales

// Declaración de Subrutinas o Métodos

// Programa Principal
void main ()
{
// Definiciones de Variables Locales

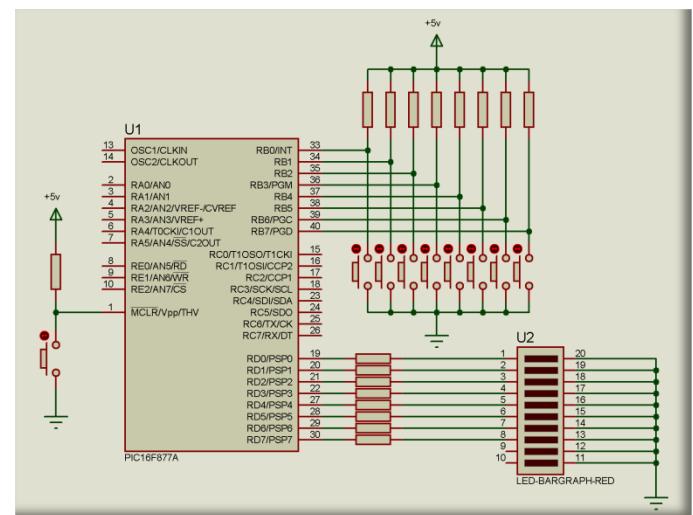
// Configuración de Puertos

// Prueba de Definiciones globales
prende_pin(PIN_D0);

// Bucle Principal
while (1){
// Instrucciones del programa

} // end while

} //end main
```



Este primer programa hace uso de las definiciones globales para modificar el nombre de una instrucción, `output_high` permite colocar un “1” en el pin que deseemos de nuestro microcontrolador, pero en este caso podemos llamar la función por medio de `prende_pin`, al igual `output_low` permite colocar un “0” en un pin, pero gracias a las definiciones globales que hemos hecho podemos llamar esta función por medio de `apaga_pin`. Como se puede observar en el programa, la instrucción `prende_pin` se encuentra antes del “Bucle principal” por lo que esta instrucción solo se realiza una vez y en la parte “Instrucciones del programa” no hemos colocado nada, por lo que enseguida de colocar un “1” en el `PIN_D0` el programa no realizará ninguna acción más.





## DEFINICIÓN DE VARIABLES

Los tipos de variables admitidos en los sistemas embebidos varían dependiendo del fabricante del microcontrolador, algunos tipos de variables admitidos por ejemplo para el caso de los PIC son los siguientes:

Type-Specifier	Size	Range		
		Unsigned	Signed	Digits
int1	1 bit number	0 to 1	N/A	1/2
int8	8 bit number	0 to 255	-128 to 127	2-3
int16	16 bit number	0 to 65535	-32768 to 32767	4-5
int32	32 bit number	0 to 4294967295	-2147483648 to 2147483647	9-10
float32	32 bit float	$-1.5 \times 10^{45}$ to $3.4 \times 10^{38}$		7-8

donde cada uno tiene su capacidad y rango que puede valer un numero y tiene su correspondencia al standard de C.

C Standard Type	Default Type
short	int1
char	unsigned int8
int	int8
long	int16
long long	int32
float	float32

La definición de qué tipo de variable vamos a utilizar, sea variable local o global, tiene que ver con el alcance o cantidad de eventos en los cuales puede influir. Una **Variable Global** tiene efecto en todas y cada parte del programa en C que se desarrolle, mientras una **Variable Local** solo tiene efecto en una sección del programa.

En los siguientes casos se indica la localización donde se declaran para cada caso, mientras en el primer caso la variable tendrá efecto sobre todo el código incluyendo la rutina principal (main), el segundo caso solo tendrá efecto en los procesos involucrados en la rutina principal.





Ejemplos:

VARIABLE GLOBAL

```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )

// Definiciones de Variables Globales
Int contador;

// Declaración de Subrutinas o Métodos

// Programa Principal
void main ()
{
// Definiciones de Variables Locales

// Bucle Principal
while (1){
    // Instrucciones del programa

} // end while

} //end main
```

VARIABLE LOCAL

```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )

// Definiciones de Variables Globales

// Declaración de Subrutinas o Métodos

// Programa Principal
void main ()
{
// Definiciones de Variables Locales
Int contador;

// Bucle Principal
while (1){
    // Instrucciones del programa

} // end while

} //end main
```

## DECLARACIÓN DE SUBRUTINAS O METODOS

Las subrutinas o métodos son segmentos de programa que nos sirve para destinar tareas que pueden o no repetirse constantemente. En otros casos es usado para separar una tarea compleja en pequeñas tareas realizable, de tal forma que el código sea legible y también nos permita detectar errores de forma más sencilla.

Dependiendo del caso, la declaración de un método tiene la siguiente sintaxis

### CASO 1:

**El método solo realiza una función sin esperar una respuesta.**

```
void prende_led()
{
    output_high(PIN_D0)
}
```





**CASO 2:**

**El método realiza una función y devuelve una respuesta.**

```
int suma(int valor1, int valor2)
{
    return(valor1+valor2);
}
```

En los siguientes ejemplos se deben usar las funciones en la parte principal del programa.

**Demo 02**

```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )

// Definiciones de Variables Globales

// Declaración de Subrutinas o Métodos
void prende_led()
{
    output_high(PIN_D0);
}

// Programa Principal
void main ()
{
// Definiciones de Variables Locales

// Bucle Principal
while (1){
    // Instrucciones del programa
    prende_led();
} // end while

} //end main
```

**Demo 03**

```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )

// Definiciones de Variables Globales
Intoperacion;

// Declaración de Subrutinas o Métodos
int suma(int valor1, int valor2)
{
    return(valor1+valor2);
}

// Programa Principal
void main ()
{
// Definiciones de Variables Locales

// Bucle Principal
while (1){
    // Instrucciones del programa
    operacion=suma(2,8);
    output_d(operacion);
} // end while

} //end main
```



# Manual – Aplicaciones Prácticas con Microcontroladores PIC

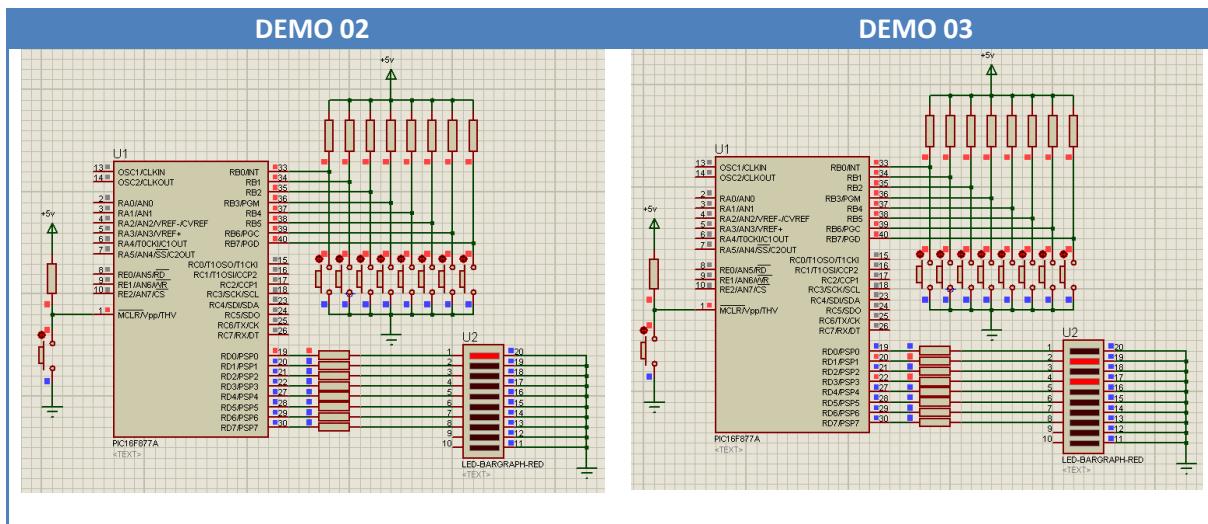
Elaborado por: M.I.E. Hugo Antonio Méndez Guzmán



En Demo 02, la función solo realiza una operación por lo que puede ser llamada solo por su nombre, mientras en Demo 03, la función requiere de dos valores para hacer la operación “2” y “8”, y en este caso de un espacio de memoria donde se arrojará el resultado “operación” para posteriormente ser utilizado.

Al igual que en la declaración de variables el tipo de resultado que arroja un método tendrá que ser declarado, en el caso 1 el método no arroja ningún resultado por lo que el tipo de variable será “void” o nulo, mientras en el caso 2 el resultado que se espera es un numero entero por lo que lleva “int” y de la misma forma los valores con los que hace la operación son valores enteros (int valor1, int valor2).

## SIMULACIONES





## FUNCIONES DEFINIDAS PARA EL MICROCONTROLADOR

Las principales funciones de entrada y salida de datos de un microcontrolador se especifican a continuación:

Instrucción	Función
<code>input_x()</code>	Lee la información de un Puerto ("x" puede ser A, B, C, etc.)
<code>output_x(y)</code>	Manda información a un Puerto ("x" puede ser A, B, C, etc., "y" es el dato)
<code>input_state(x)</code>	Lee la información de un Pin ("x" puede ser PIN_A0, PIN_B1, etc.)
<code>output_low(x)</code>	Manda un "0" a un Pin ("x" puede ser PIN_A0, PIN_B1, etc.)
<code>output_high(x)</code>	Manda un "1" a un Pin ("x" puede ser PIN_A0, PIN_B1, etc.)
<code>output_toggle()</code>	Commuta el valor de un PIN de "0" a "1" y viceversa ("x" puede ser PIN_A0, PIN_B1, etc.)
<code>output_float(x)</code>	Configura un pin como entrada. ("x" puede ser PIN_A0, PIN_B1, etc.)
<code>output_drive(x)</code>	Configura un pin como salida. ("x" puede ser PIN_A0, PIN_B1, etc.)

## FUNCIONES PARA RETARDOS

Dado que muchas de las aplicaciones para un microcontrolador son aplicaciones de baja velocidad y dado que un microcontrolador puede ejecutar sus instrucciones en unos cuantos microsegundos, es necesario el uso de retrasos en la ejecución de un programa en C. Estos retrasos son retardos prediseñados que podemos usar para disminuir la velocidad de operación de un programa. Entre los más comunes se encuentran los siguientes:

Instrucción	Función
<code>delay_ms(x)</code>	Genera un retardo de "x" milisegundos.
<code>delay_us(x)</code>	Genera un retardo de "x" microsegundos.
<code>delay_cycles(x)</code>	Genera un retardo de "x" instrucciones realizadas por el microcontrolador.

Por otro lado, si se requiere manejar retardos muy pequeños para la ejecución de un programa, un factor importante en la velocidad de ejecución es la cantidad de tiempo que el microcontrolador requiere para la ejecución de una instrucción, este tiempo depende de la velocidad a la que está operando su oscilador, donde para el caso de los Microcontroladores PIC está definido como sigue:

$$tiempo = \frac{4}{Fosc}$$





De este modo, por medio de la instrucción “delay\_cycles” podemos generar retardos muy pequeños, menores a 1 microsegundos. Por ejemplo, si usamos la instrucción “delay\_cycles(1)” este generará un retardo tal como si el microcontrolador estuviera ejecutando una instrucción, donde para una oscilador de 20MHz, el retardo que generará es de 0.2 microsegundos.

## OPERADORES

Por medio de los operadores un microcontrolador es capaz de hacer operaciones con valores numéricos o datos contenidos en algún registro de memoria de este. Algunos de estos operadores son los siguientes:

Operador	Función
+	Suma.
+=	Suma acumulativa, adhiere un valor numérico a un registro. Ejemplo: $x+=5$ , adhiere 5 al valor que ya contiene x.
-	Resta
-=	Resta acumulativa, resta un valor numérico a un registro. Ejemplo: $x-=5$ , resta 5 al valor que ya contiene x.
*	Multiplicación
*=	Multiplicación acumulativa, multiplica por un valor numérico a un registro. Ejemplo: $x*=5$ , multiplica por 5 al valor que ya contiene x.
/	División
/=	División acumulativa, divide por un valor numérico a un registro. Ejemplo: $x/=5$ , divide por 5 al valor que ya contiene x.
++	Incrementa el valor de un registro por 1.
--	Decrementa el valor de un registro por 1.
%	Obtiene el residuo de una operación.
&	Operador AND bit a bit.
^	Operador XOR bit a bit.
	Operador OR bit a bit.
~	Obtiene el complemento de un número.





## OPERADORES DE COMPARACIÓN

Los operadores de comparación nos permiten hacer comparaciones valores y registros de un microcontrolador, permitiéndonos tener control de procesos por medio de la información obtenida de estos. En la siguiente tabla se muestran los más usados en un programa en C.

Operador	Función
==	Devuelve verdadero si dos valores son iguales.
>	Devuelve verdadero si un número es mayor a otro.
>=	Devuelve verdadero si un número es mayor o igual a otro.
!=	Devuelve verdadero si un número es diferente a otro.
<	Devuelve verdadero si un número es menor a otro.
<=	Devuelve verdadero si un número es menor o igual a otro.
&&	Devuelve verdadero si dos condiciones son verdaderas.
	Devuelve verdadero si al menos una de dos condiciones es verdadera.

## SENTENCIAS Y CICLOS

Las sentencias de programación en C nos permiten llevar un control del flujo del programa, realizando comparaciones y asignando procedimientos si una condición se cumple o no.

### Condicional IF

Nos permite la ejecución de una o varias tareas, siempre y cuando se cumpla una condición. Estas tareas solo se realizaran una sola vez, ya ejecutadas la condicional IF termina su función.

#### Sintaxis:

```
If (condición1)
{
    Tarea1;
    Tarea2;
}
```





## Condicional IF-ELSE

Permite la ejecución de una o varias tareas, siempre y cuando se cumpla una condición. Si la condición no se cumple se le pueden asignar también tareas a realizar.

### **Sintaxis:**

```
If (condición1)
{
    Tarea1;
    Tarea2;
}
else
{
    Tarea3;
}
```

Como se puede observar en su sintaxis, si la condición 1 se cumple, la condicional IF ejecutara la Tarea 1 y Tarea 2, cuando esta no se cumple ejecuta la Tarea 3. Estas tareas solo se realizaran una sola vez, ya ejecutadas la condicional IF-ELSE termina su función.

## Condicional SWITCH

En casos cuando se ocupan realizar varias comparaciones para un mismo registro, podemos usar la condicional switch, la cual nos permite asignar una o varias funciones para cada posible valor que pueda tener un registro, y de manera similar a la sentencia IF-ELSE si el registro no corresponde a ninguna de las opciones que establezcamos, podemos asignar una función que realice por defecto.

### **Sintaxis:**

```
switch (registro)
{
    case “valor1”:
        función1;
    break;
    case “valor2”:
        función2;
    break;
    default:
        función3;
    break;
}
```





En la sintaxis podemos observar que para el caso donde registro valga “valor1” se realizará la función1, si el registro vale “valor2” hará la función2 y en el caso que no corresponda a “valor1” o “valor2” la condicional SWITCH realizará la función3.

## **Ciclo FOR**

Los ciclos por otra parte nos permiten la ejecución de una tarea repetitiva, siempre y cuando se cumplan las condiciones para que este se ejecute.

El ciclo FOR nos permite realizar una tarea repetitiva y a la vez llevar un control preciso de la cantidad de veces que se está ejecutando esta. En este ciclo establecemos un valor inicial con el cual el ciclo comenzará a ejecutarse, el segundo parámetro de esta función es una condición para seguir repitiendo la ejecución de las tareas y por último se establece de cuanto en cuanto se estará contabilizando cada vez que se ejecute una tarea. De tal manera que cuando el conteo llegue o supere la condición que se establezca, el ciclo finalizará.

**Sintaxis:**

```
for ("valor_inicial_del_conteo" ; "condición" ; "conteo")
{
    tarea1;
    tarea2;
}
```

El ciclo WHILE a diferencia del FOR no lleva una cuenta de las veces que se están ejecutando una tarea repetitiva, sino que este ejecuta la tarea o tareas siempre y cuando la condición que se le establezca se cumpla.

**Sintaxis:**

```
while ("condición")
{
    tarea1;
    tarea2;
}
```

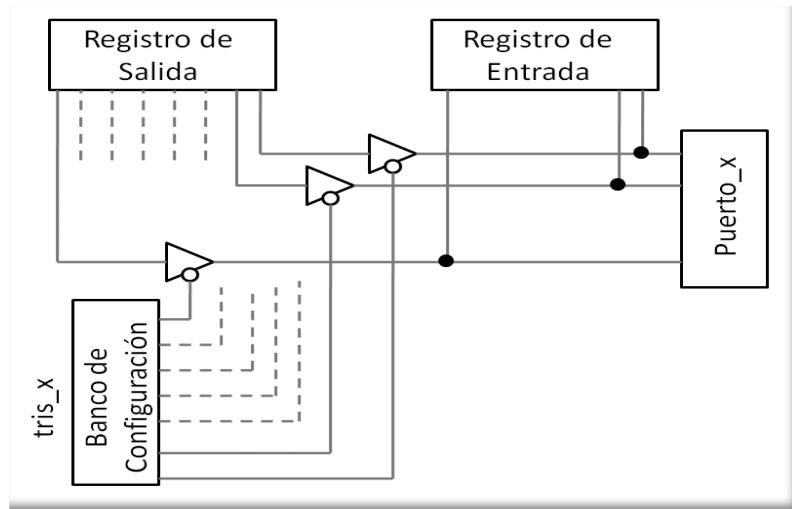




## MANEJO DE PUERTOS

La estructura general de un puerto en un microcontrolador está conformada principalmente por registros, registros de entrada y salida, y un registro de configuración. El registro de salida permite almacenar la información antes de que esta proceda al puerto físico. El registro de entrada, toma los datos del puerto físico y los almacena temporalmente mientras son utilizados. El registro de configuración nos permite elegir el comportamiento del puerto, ya sea entrada o salida.

Como se puede apreciar en la [figura 3](#), el registro de configuración actúa de forma similar a buffers, donde si estos son activados en “0” la información fluirá del Registro de Salida hacia el Puerto físico y al mismo tiempo una copia de esta información se almacenará en el Registro de Entrada. De forma contraria si el registro de configuración es puesto en “1” desactivará los buffers, impidiendo el flujo de información del registro de Salida al Puerto físico, por lo que el Registro de Entrada almacenará una copia de la información contenida en el Puerto físico.



**Figura 3. Estructura Genérica de un Puerto en un Microcontrolador.**

De lo anterior podemos asumir que si no configuramos adecuadamente el registro tris, la información no saldrá del microcontrolador a su puerto físico.

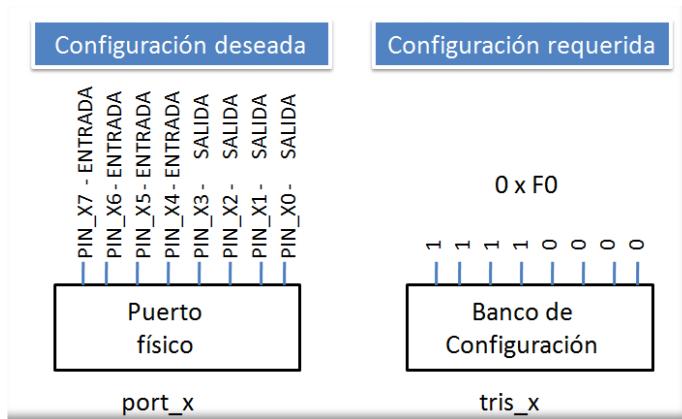
De igual manera si queremos que el puerto funcione como entrada, tendremos que deshabilitar la salida de información a través de la configuración del registro tris\_x.





## CONFIGURACIÓN DE PUERTOS

La configuración de un puerto como entrada o salida depende de la configuración que realicemos en el registro Tris, donde aquello que configuremos como “1” será entrada y lo configurado en “0” será salida, tal como se muestra en la [figura 4](#).



**Figura 4. Configuración de un Puerto.**

En programación en C, la configuración se realiza por medio de la instrucción

**Sintaxis:**  
set\_tris\_x (valor);

donde x es el puerto que deseamos configurar y el valor que pondremos será dependiendo de lo que deseamos configurar, tal como se muestra en la [figura 4](#).

## MODO BIT, ENTRADA Y SALIDA

La entrada y salida de datos en un pin del microcontrolador, se realiza por las instrucciones

```
input_state(PIN_x);
output_high(PIN_x);
output_low(PIN_x);
output_toggle(PIN_x);
```

que combinados con algunas sentencias de programación podemos condicionar el accionamiento de alguna de ellas.





El siguiente programa de ejemplo, realiza en conjunto con la condicional IF-ELSE, el chequeo de un pin del microcontrolador PIN\_B0, de tal manera que si nosotros provocamos en él un “1” este será enviado al PIN\_D0, y de forma contraria, si en PIN\_B0 hay un “0”, este será enviado al PIN\_D0.

## Demo 04

```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )
// Definiciones Globales

// Definiciones de Variables Globales

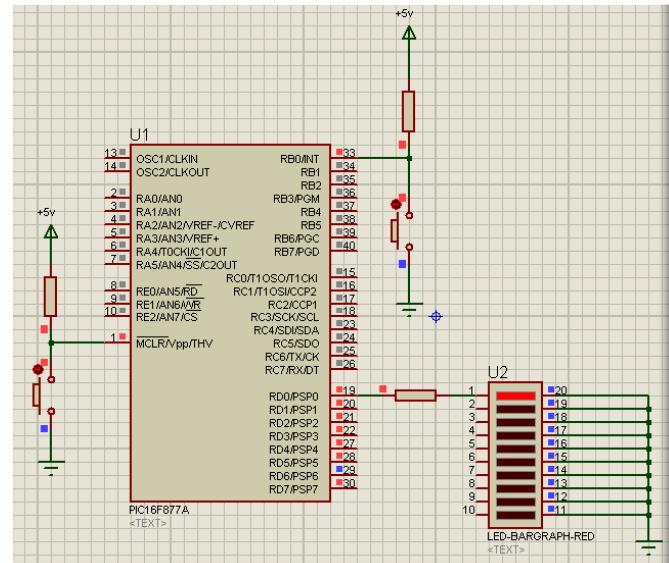
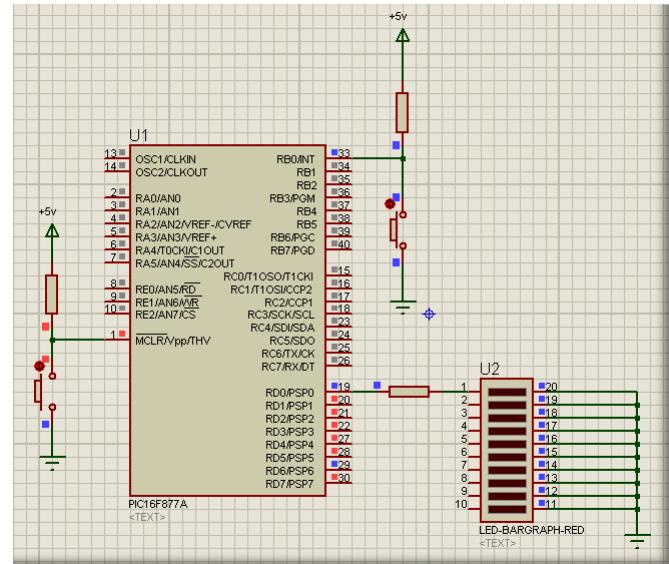
// Declaración de Subrutinas o Métodos

// Programa Principal
void main ()
{
// Definiciones de Variables Locales

// Configuración de Puertos
set_tris_b(0xFF);
set_tris_d(0x00);

// Bucle Principal
while (1){
    // Instrucciones del programa
    If(input_state(PIN_B0)==1)
    {
        output_high(PIN_D0);
    }
    else
    {
        output_low(PIN_D0);
    }

} // end while
} //end main
```



Como se puede apreciar, antes de ejecutar la rutina principal hemos configurado todo el puerto B como entrada (pines en gris), mientras todo el puerto D fue configurado como salida.





## Consideraciones en Práctica

Entre las consideraciones que tenemos que tomar en cuenta para usar pulsadores al generar acciones diferentes se encuentran **Espera a Pulsar**, **Espera a Soltar** y el **Efecto Rebote**.

En ocasiones tendremos aplicaciones que necesiten la presión de un botón para que se ejecute una tarea, una forma de solucionar el caso **Espera a Pulsar** es el uso del ciclo WHILE, donde podemos configurar el ciclo para que siempre este realizando una tarea o bien que no realice ninguna acción y solo este esperando a la presión del pulsador.

Un ejemplo de estos casos es el siguiente programa, este se encontrará encendiendo y apagando un led constantemente y colocado en el PIN\_D0 hasta que sea presionado un pulsador colocado en PIN\_B0.

### Demo 05

```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )
// Definiciones Globales

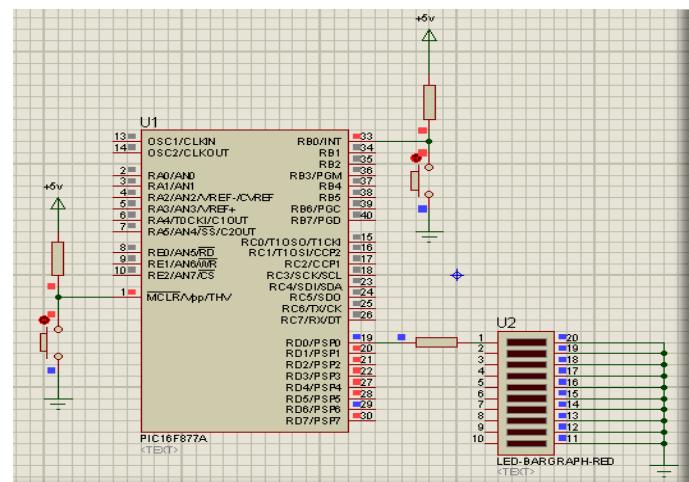
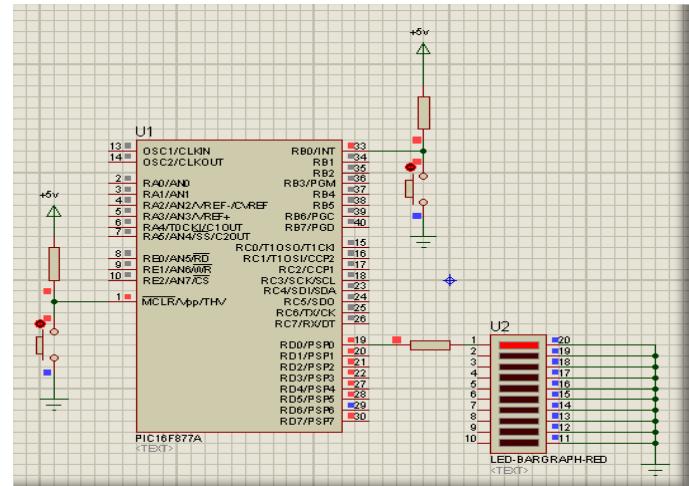
// Definiciones de Variables Globales

// Declaración de Subrutinas o Métodos

// Programa Principal
void main ()
{
// Definiciones de Variables Locales

// Configuración de Puertos
set_tris_b(0xFF);
set_tris_d(0x00);

// Bucle Principal
while (1){
    // Instrucciones del programa
    while(input_state(PIN_B0)==1)
    {
        output_high(PIN_D0);
        delay_ms(200);
        output_low(PIN_D0);
        delay_ms(200);
    }
} // end while
} //end main
```





dentro de la condición del WHILE está definida el leer constantemente el PIN\_B0 y compararlo con un “1”, si el estado de PIN\_B0 es igual a “1” entonces el ciclo estará ejecutando constantemente el encender el led, crear un retardo de 200 milisegundos, apagar el led y crear un retardo de 200 milisegundos de nuevo. De tal manera que al presionar el pulsador no se cumple la condición y deja de hacerlo.

**El siguiente ejemplo** también usa un ciclo WHILE, pero en este caso **no se encontrará realizando ninguna acción, solo se encontrará esperando la presión de un pulsador** en PIN\_B0, al presionarlo permitirá la continuación del programa poniendo en alto el PIN\_D0, este permanecerá así por 1 segundo y posteriormente se pondrá en bajo.

## Demo 06

```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )
// Definiciones Globales

// Definiciones de Variables Globales

// Declaración de Subrutinas o Métodos

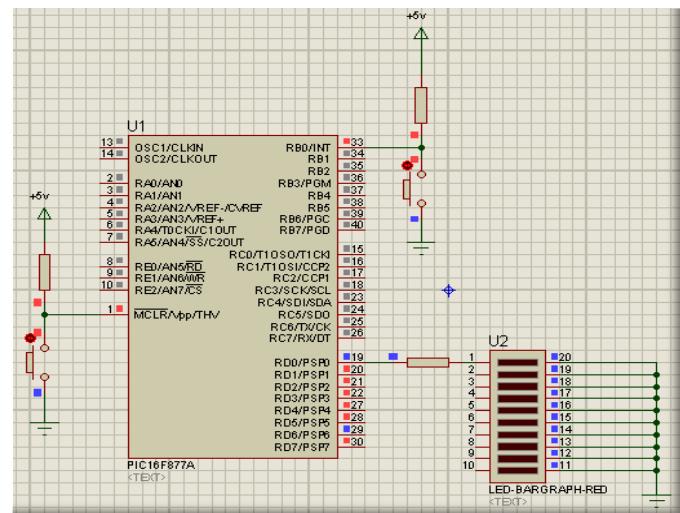
// Programa Principal
void main ()
{
// Definiciones de Variables Locales

// Configuración de Puertos
set_tris_b(0xFF);
set_tris_d(0x00);

// Bucle Principal
while (1){
    // Instrucciones del programa
    while(input_state(PIN_B0)==1);

        output_high(PIN_D0);
        delay_ms(1000);
        output_low(PIN_D0);
        delay_ms(1000);

    } // end while
} //end main
```





Una diferencia a notar en el ciclo WHILE esta vez, es que si al terminar la condición del ciclo le ponemos “;”, le estaremos indicando no realices nada o en otras palabras, “**mientras esperas a que la condición se cumpla no realices nada**”. Al momento de presionar el pulsador producimos un “0” por lo que el ciclo WHILE ya no cumple su condición y termina de ejecutarse permitiendo la acción de las demás instrucciones. En este caso al programa le estamos indicando “**espera sin hacer nada a que se presione el pulsador**”.

Por otro lado si dejamos presionado el pulsador, siempre estaremos produciendo un “0” por lo que el ciclo WHILE ya no cumple su condición y mientras continúe presionado este no tendrá efecto.

Partiendo de los programas anteriores, podemos generar una aplicación que primero espere a que presionemos el pulsador, luego esperemos a que se suelte y una vez libre realice una acción.

## Demo 07

```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )
// Definiciones Globales

// Definiciones de Variables Globales

// Declaración de Subrutinas o Métodos

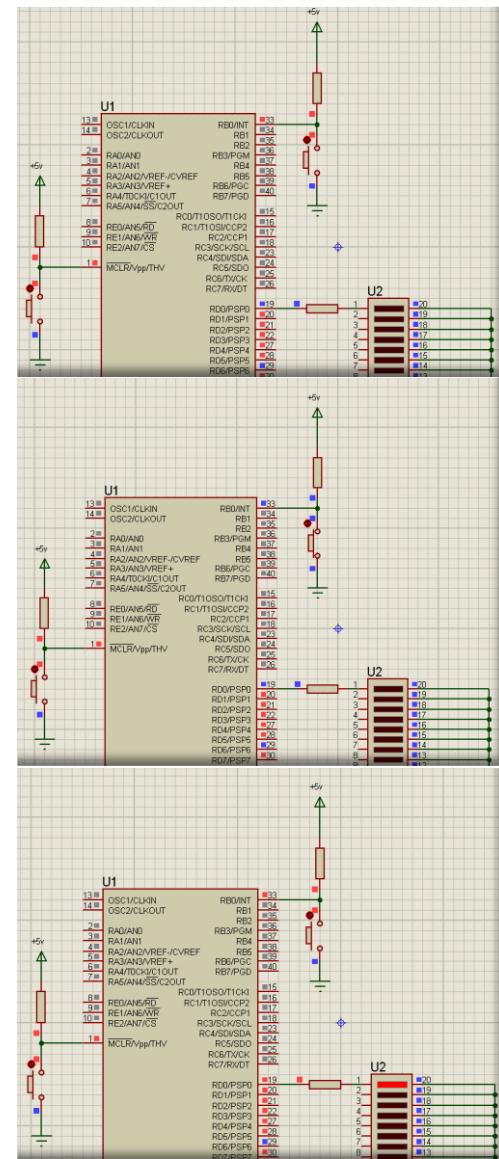
// Programa Principal
void main ()
{
// Definiciones de Variables Locales

// Configuración de Puertos
set_tris_b(0xFF);
set_tris_d(0x00);

// Bucle Principal
while (1){
// Instrucciones del programa
    while(input_state(PIN_B0)==1);
    while(input_state(PIN_B0)==0);

    output_high(PIN_D0);
    delay_ms(1000);
    output_low(PIN_D0);
    delay_ms(1000);

} // end while
} //end main
```





En este caso el primer WHILE espera sin realizar nada mientras tengamos un “1” en el PIN\_B0, cuando presionamos el pulsador sale de este ciclo, el segundo WHILE espera sin realizar nada mientras tengamos un “0” en el PIN\_B0, al soltar el pulsador termina este ciclo y por consiguiente el programa encenderá el led en el PIN\_D0, lo mantendrá 1 segundo, lo apagará y finalmente lo mantendrá 1 segundo también.

Al terminar estas condiciones vuelve a repetirse la secuencia, de modo que para encender y apagar el led, el microcontrolador espera a que se presione el botón y luego espera a que se suelte.

## Efecto Rebote

La consideración más importante cuando se manejan pulsadores para aplicaciones de un microcontrolador es el **Efecto Rebote**. Este efecto ocurre cuando soltamos un pulsador, idealmente cuando hacemos la presión de un pulsador el voltaje que entra al microcontrolador se haría 0 y de igual manera cuando lo soltamos debería retornar a 5v, tal como se muestra en la figura 5. Sin embargo, en la práctica cuando soltamos un pulsador, en este se crea un pequeño arco de voltaje que nosotros no podemos percibir, este arco puede crear falsas interpretaciones en la lectura de un pin del microcontrolador, ya que genera oscilaciones repentinas. En circuitos combinacionales y secuenciales es frecuente ver el uso de Flip-Flops para la eliminación de este efecto, en el caso de los microcontroladores bastará con generar un pequeño retardo, después de la rutina de espera para soltar botón, de forma que el microcontrolador no tome la lectura cuando se están generando estos arcos y pase por desapercibido este efecto.

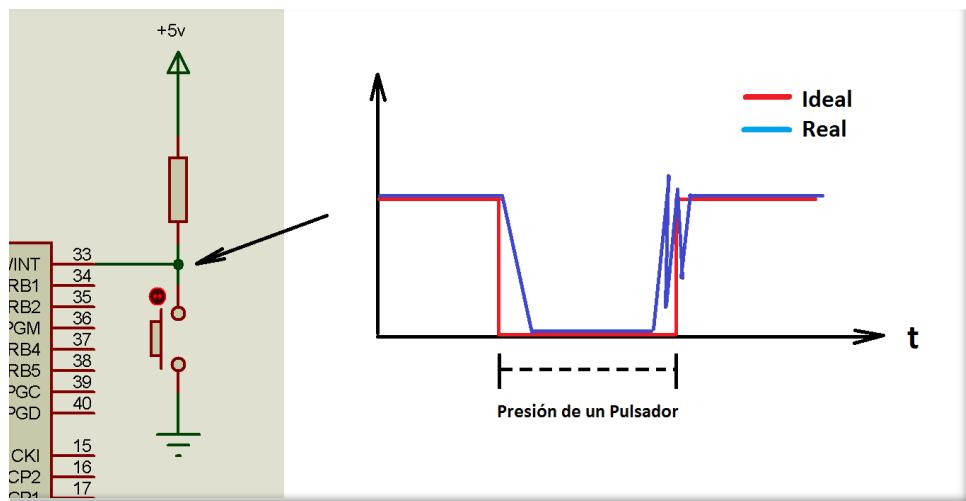


Figura 5. Efecto Rebote.





Como se puede apreciar en el siguiente código, solo basta el agregar un retardo de unos cuantos milisegundos, antes de que el microcontrolador siga con la secuencia de operaciones.

```
// Instrucciones del programa
while(input_state(PIN_B0)==1);
while(input_state(PIN_B0)==0);
// Retardo anti-rebote
delay_ms(100);

demás_funciones();
```

## **MODO BYTE, ENTRADA Y SALIDA**

La asignación y lectura de datos de un puerto en un microcontrolador se puede realizar de dos formas, una directa y otra indirecta.

La [asignación directa](#) consiste en enviar un dato o leer un dato de un puerto sin tomar en cuenta la configuración que hicimos en el registro tris, en otras palabras al usar este modo de operación el microcontrolador **forzará el puerto a que sea entrada o salida** una vez enviada o recibida dicha información.

La [asignación indirecta](#) por otra parte toma en cuenta la configuración hecha en el registro tris, de tal manera que **sí** nosotros **configuramos un puerto** o una sección del puerto **como entrada, aunque enviamos información, esta nunca saldrá ya que** el modo de operación indirecto **respeta la configuración de entrada y salida realizada en el registro tris**. Se puede decir que este modo de operación respeta la asignación dada por los buffers de configuración (modo de selección de entrada y salida de cada microcontrolador), mientras el modo directo obliga a los buffers a sacar la información por el puerto ([figura 3](#)).

Estos modos de operación son importantes, ya que en ocasiones tendremos aplicaciones donde no habrá pines suficientes en un microcontrolador para realizarla y nos veremos forzados a compartir un puerto como entrada y salida de datos al mismo tiempo.

### **Asignación Directa**

En la asignación directa, el registro tris no tiene importancia por lo cual podemos no escribirlo en el programa y esto no afectará a la ejecución de tal. Entre las principales instrucciones que trabajan en este modo de operación se encuentran las siguientes:





```
output_x(datos);
entrada=input_x();
output_low(PIN_x);
output_high(PIN_x);
```

Las primeras dos nos permiten enviar y recibir datos de capacidad de 8 bits a través de un puerto, mientras las siguientes solo mandaran 1 bit de información a un pin que deseemos.

En el siguiente programa muestra un ejemplo de la asignación directa, primeramente configuramos el puerto b como entrada, posteriormente hacemos uso de un while, el cual no hará ninguna acción, solo estará esperando la presión de un pulsador colocado en el PIN\_B0, al presionarlo producirá un 0, por lo que el primer ciclo while finalizará. Posteriormente, el ciclo while de enseguida, permanecerá sin realizar ninguna acción mientras el PIN\_B0 este en cero (el pulsador sigue presionado). Una vez al soltar el pulsador, mandaremos a poner en 0 todo el puerto B.

## Demo 08

```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )
// Definiciones Globales

// Definiciones de Variables Globales

// Declaración de Subrutinas o Métodos

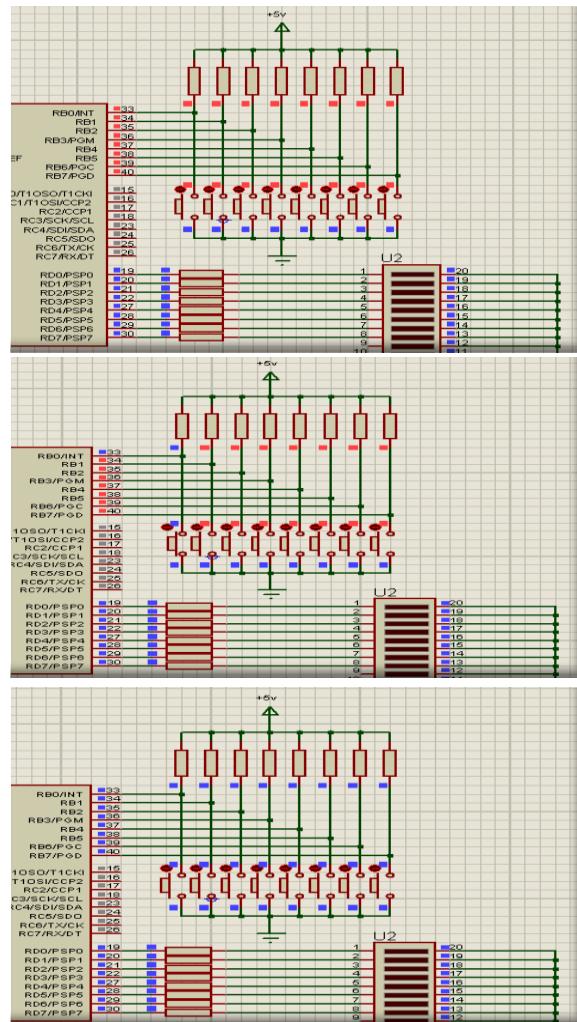
// Programa Principal
void main ()
{
// Definiciones de Variables Locales

// Configuración de Puertos
set_tris_b(0xFF);

// Bucle Principal
while (1){
// Instrucciones del programa
    while(input_state(PIN_B0)==1);
    while(input_state(PIN_B0)==0);

    output_b(0x00);

} // end while
} //end main
```





Un error típico al programar en C, es no considerar el efecto que se muestra en las imágenes anteriores, nosotros le programamos la lectura de un pin para asignarle una función y posteriormente usamos alguna instrucción de asignación directa sobre el mismo puerto, sin considerar que estamos forzando al puerto a ser salida, por lo que en siguientes instrucciones a pesar de que le damos nuevamente la instrucción de leer un pin, este no vuelve a su estado inicial de ser entrada, causándonos conflictos en las aplicaciones.

## Asignación Indirecta

En la asignación indirecta no hacemos uso de las instrucciones antes mencionadas, sino que usamos la jerarquía de la [figura 3](#), donde los datos serán mandados al registro de salida y de esta manera, los datos solo saldrán al puerto físico, sí el registro tris se lo permite, respetando así la configuración que nosotros le demos a cada puerto.

Para hacer uso de los registros de salida de cada puerto, al principio del programa tendremos que especificar la dirección en memoria de dichos registros, esta declaración se muestra a continuación.

```
#byte portA=0x05  
#byte portB=0x06  
#byte portC=0x07  
#byte portD=0x08
```

Y en lugar de usar las instrucciones de asignación directa, usaremos los nombres asignados a este puerto, de tal manera que si deseamos mandar un dato al puerto b por ejemplo, usaremos:

```
portB=dato;
```

o para leer el contenido del puerto en el registro de salida:

```
dato=portB;
```

Para diferenciar el efecto que tiene este procedimiento, en el siguiente ejemplo se muestra el uso de estas instrucciones, donde en un comienzo asignaremos la mitad del puerto b como entrada desde RB0 hasta RB3 y la otra mitad como salida, para posteriormente usar un pulsador en RB0 que mande ceros a todo el puerto B después de presionarlo y soltarlo.





## Demo 09

```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )
#byte portA=0x05
#byte portB=0x06
#byte portC=0x07
#byte portD=0x08

// Definiciones Globales

// Definiciones de Variables Globales

// Declaración de Subrutinas o Métodos

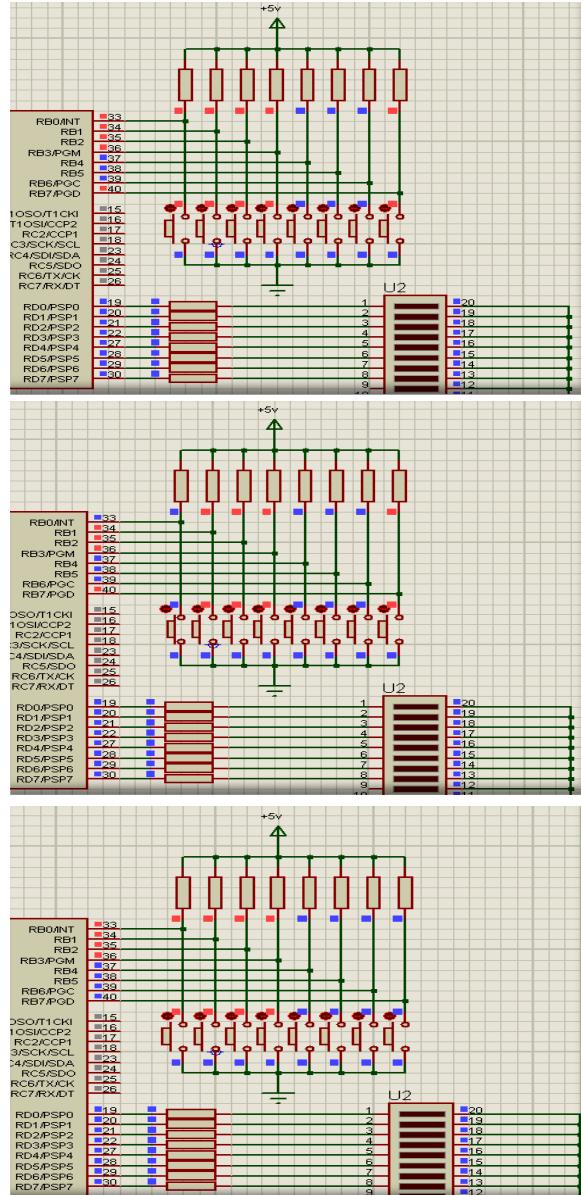
// Programa Principal
void main ()
{
// Definiciones de Variables Locales

// Configuración de Puertos
set_tris_b(0x0F);

// Bucle Principal
while (1){
    // Instrucciones del programa
    while(input_state(PIN_B0)==1);
    while(input_state(PIN_B0)==0);

    portB=0x00;

} // end while
} //end main
```



En las imágenes anteriores podemos observar que a pesar de haber enviado un 0 a todos los pines del puerto B, en el puerto físico solo se muestran la mitad debido a la configuración de entrada y salida que asignamos en el registro tris.

Como se puede apreciar es de suma importancia considerar estos modos de asignación, ya que se tendrán aplicaciones donde se requiera pines de entrada y salida en un solo puerto, y si usamos la asignación directa estaremos perdiendo la configuración que realicemos en el registro tris.

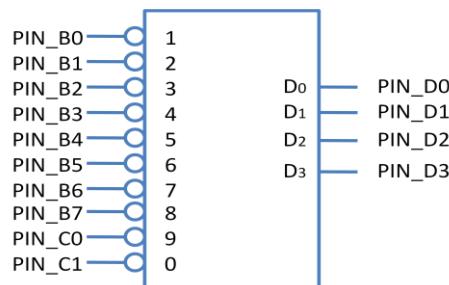




## Ejercicios prácticos

### Demo 10 - Convertidor de Decimal a Binario.

Se desea realizar un convertidor de decimal a binario por medio de un microcontrolador, donde las entradas del 0 al 9 se activan por medio de un “0” y las salidas se activan en alto. No considerar prioridad.



Para la programación de esta aplicación no es necesario el configurar el registro tris, ya que las entradas y salidas pueden ser importadas sin problemas por medio de asignación directa. Por lo que solo será necesario hacer uso de unos cuantos IF para detectar cuando se presione un pulsador y produzca el “0” que requerimos. Dado que un microcontrolador de forma automática tiene que representar un dato en forma binaria a un puerto, solo requeriremos la instrucción output\_x para enviar el valor correspondiente a la salida, tal que sí presionamos por ejemplo el PIN\_C0 mandaremos al puerto d un 9 que automáticamente será representado como “1001”.

```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )

// Definiciones Globales

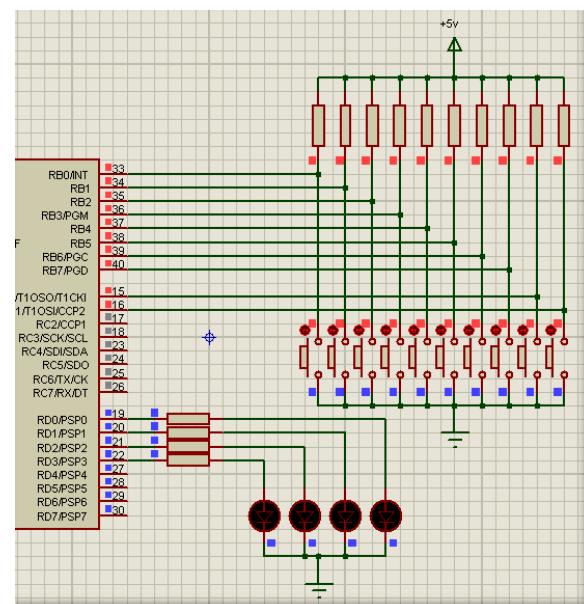
// Definiciones de Variables Globales

// Declaración de Subrutinas o Métodos

// Programa Principal
void main ()
{
// Definiciones de Variables Locales

// Configuración de Puertos

// Inicialización de Puertos
output_d(0);
}
```

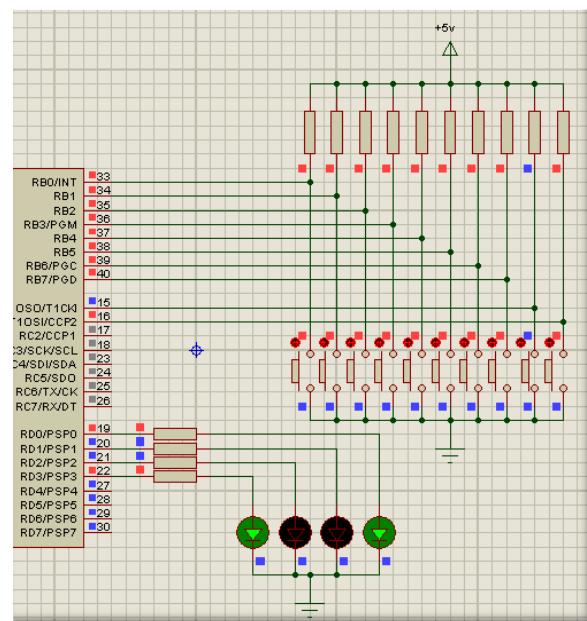
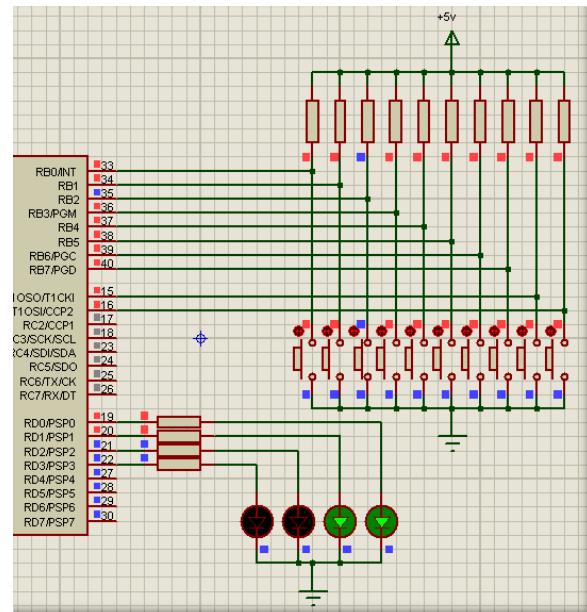


# Manual – Aplicaciones Prácticas con Microcontroladores PIC

Elaborado por: M.I.E. Hugo Antonio Méndez Guzmán



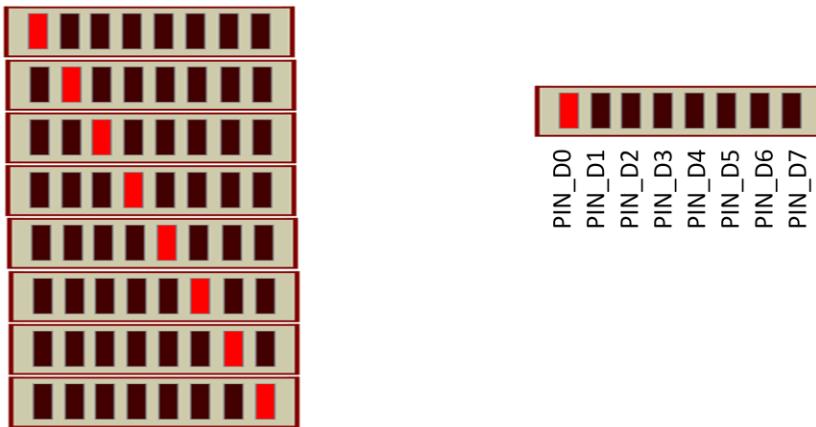
```
// Bucle Principal
while (1){
    // Instrucciones del programa
    if(input_state(PIN_B0)==0)
    {
        output_d(1);
    }
    if(input_state(PIN_B1)==0)
    {
        output_d(2);
    }
    if(input_state(PIN_B2)==0)
    {
        output_d(3);
    }
    if(input_state(PIN_B3)==0)
    {
        output_d(4);
    }
    if(input_state(PIN_B4)==0)
    {
        output_d(5);
    }
    if(input_state(PIN_B5)==0)
    {
        output_d(6);
    }
    if(input_state(PIN_B6)==0)
    {
        output_d(7);
    }
    if(input_state(PIN_B7)==0)
    {
        output_d(8);
    }
    if(input_state(PIN_C0)==0)
    {
        output_d(9);
    }
    if(input_state(PIN_C1)==0)
    {
        output_d(0);
    }
} // end while
}//end main
```





## Demo 11 - Generando un Corrimiento de Datos

Se desea generar un circuito que realice una secuencia de datos, donde solo un bit se estará desplazando de izquierda a derecha a través de un puerto tal como se muestra en la figura. Una vez concluido el recorrido tendrá que repetirse indefinidamente.



Dado que el desplazamiento de datos se tiene que repetir indefinidamente, solo tendremos que hacer una rutina que realice 8 movimientos partiendo de que el primer dato ya se encuentra activo. Por medio de un ciclo FOR realizaremos este desplazamiento y auxiliándonos del operador de rotación “registro<<=n;” donde este desplazará la información contenida en un registro n veces, que para este caso ocuparemos un desplazamiento de 1 bit a la izquierda (la información de un registro se almacena de derecha a izquierda donde el bit más a la derecha es el bit menos significativo) que se repetirá 8 veces por medio del ciclo FOR.

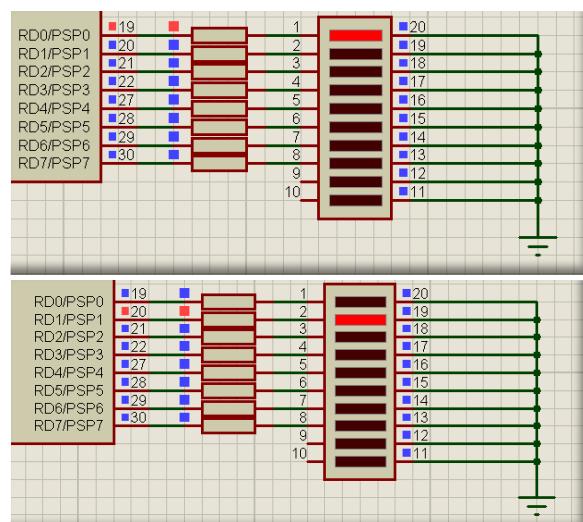
```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )

// Definiciones Globales

// Definiciones de Variables Globales

// Declaración de Subrutinas o Métodos

// Programa Principal
void main ()
{
    // Definiciones de Variables Locales
    int contador;
    int dato;
```

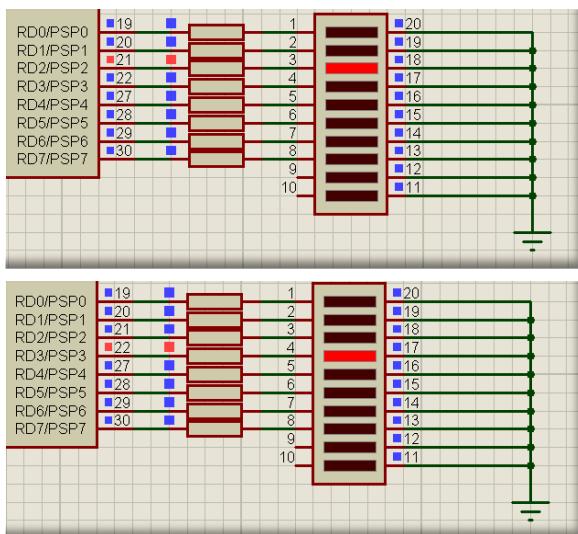




```
// Configuración de Puertos

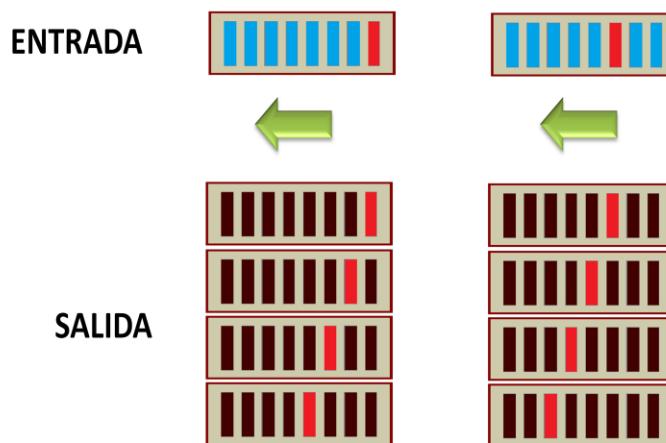
// Bucle Principal
while (1){
    // Instrucciones del programa
    dato=1;
    output_d(dato);
    for(contador=0;contador<8;contador++)
    {
        delay_ms(1000);
        dato<<=1;
        output_d(dato);
    }

} // end while
} //end main
```



## Demo 12 - Generando Corrimientos con Bit de Inicio por Pulsador

Se desea generar un circuito que realice una secuencia de datos, donde solo un bit se estará desplazando de derecha a izquierda través de un puerto, donde el inicio estará marcado por un pulsador colocado en el puerto B, tal que si presionamos el pulsador en el PIN\_B0 el corrimiento empezará en el PIN\_D0 hacia la izquierda, si presionamos el PIN\_B1 su inicio será en el PIN\_D1 y así sucesivamente (vease la figura).



# Manual – Aplicaciones Prácticas con Microcontroladores PIC

Elaborado por: M.I.E. Hugo Antonio Méndez Guzmán



Dado que el inicio está marcado por el BIT que pongamos en “1” del puerto B, y este dato será transferido al puerto D para crear el corrimiento de datos, declararemos una variable entera donde se almacenará la información del puerto B (inicio del corrimiento) para posteriormente usar un ciclo FOR que realizará el corrimiento 8 veces a partir del inicio, por lo que el código y el esquemático de esta aplicación queda como sigue.

```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )

// Definiciones Globales

// Definiciones de Variables Globales

// Declaración de Subrutinas o Métodos

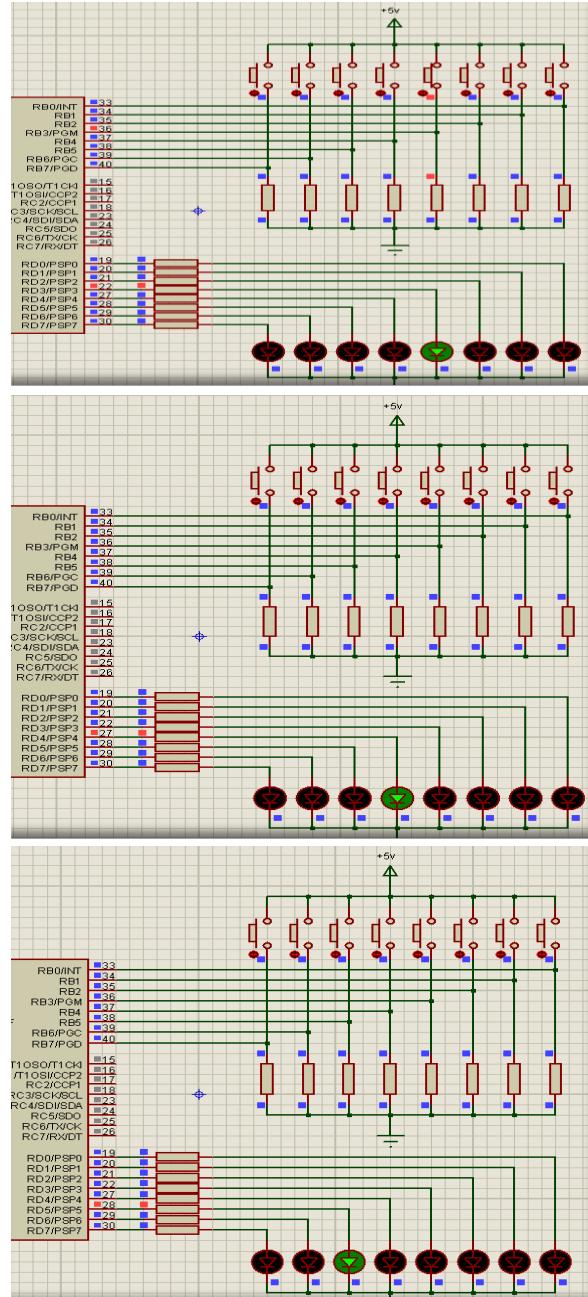
// Programa Principal
void main ()
{
// Definiciones de Variables Locales
int dato;
int contador;

// Configuración de Puertos

// Inicialización de Puertos
output_d(0);

// Bucle Principal
while (1){
    // Instrucciones del programa
    dato=input_b();
    for(contador=0;contador<8;contador++)
    {
        output_d(dato);
        delay_ms(250);
        dato<<=1;
    }

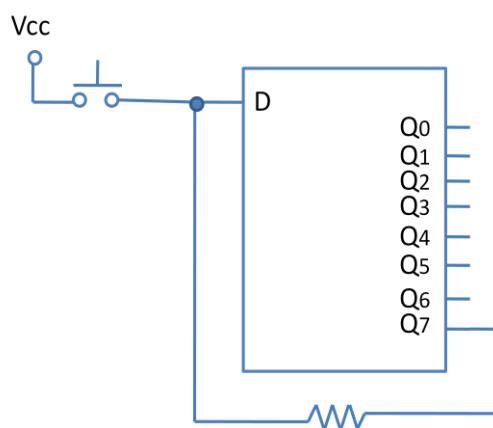
    } // end while
} //end main
```



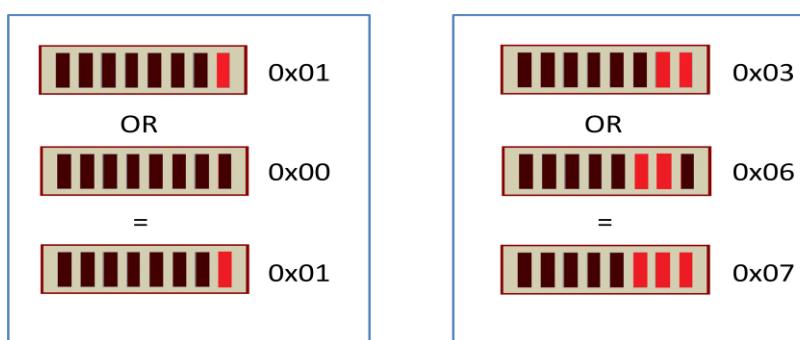


## Demo 13 - Agregando Bits a un Corrimiento (Sobre-escritura de un Registro)

Se tiene un circuito que genera corrimientos en un puerto de un microcontrolador, el circuito siempre está generando corrimientos hacia la izquierda, a pesar de que al comienzo solo tiene ceros. Los datos son introducidos por medio de un pin en un segundo puerto, cuando un dato es introducido por este, el dato es montado sobre al inicio del corrimiento. Genere un programa que realice este efecto.



Este es un problema típico de circuitos secuenciales, donde los datos siempre se están ciclando en un registro de corrimiento, donde el último dato almacenado en Q7 pasará en el siguiente paso de nuevo a la entrada. En programación la solución se torna un poco diferente, ya que en este caso el bit que está siendo montado o sobre-escrito sobre un registro, esta operación es conocida como la OR, de tal manera que si anteriormente ya existía un 1 y se le monta otro 1, este permanece, si había un 0 y se le monta un 1 el resultado será un 1.

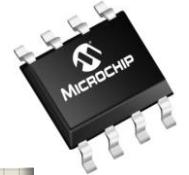


Dado esto, en el programa usaremos la operación OR para montar bits adicionales, realizando el mismo efecto y con la diferencia de usar ciclos para que este efecto se ejecute permanentemente al igual que en un circuito secuencial. Esto se muestra a continuación.



# Manual – Aplicaciones Prácticas con Microcontroladores PIC

Elaborado por: M.I.E. Hugo Antonio Méndez Guzmán



```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )

// Definiciones Globales

// Definiciones de Variables Globales

// Declaración de Subrutinas o Métodos

// Programa Principal
void main ()
{
    // Definiciones de Variables Locales
    int dato=0;
    int1 ultimo_bit;

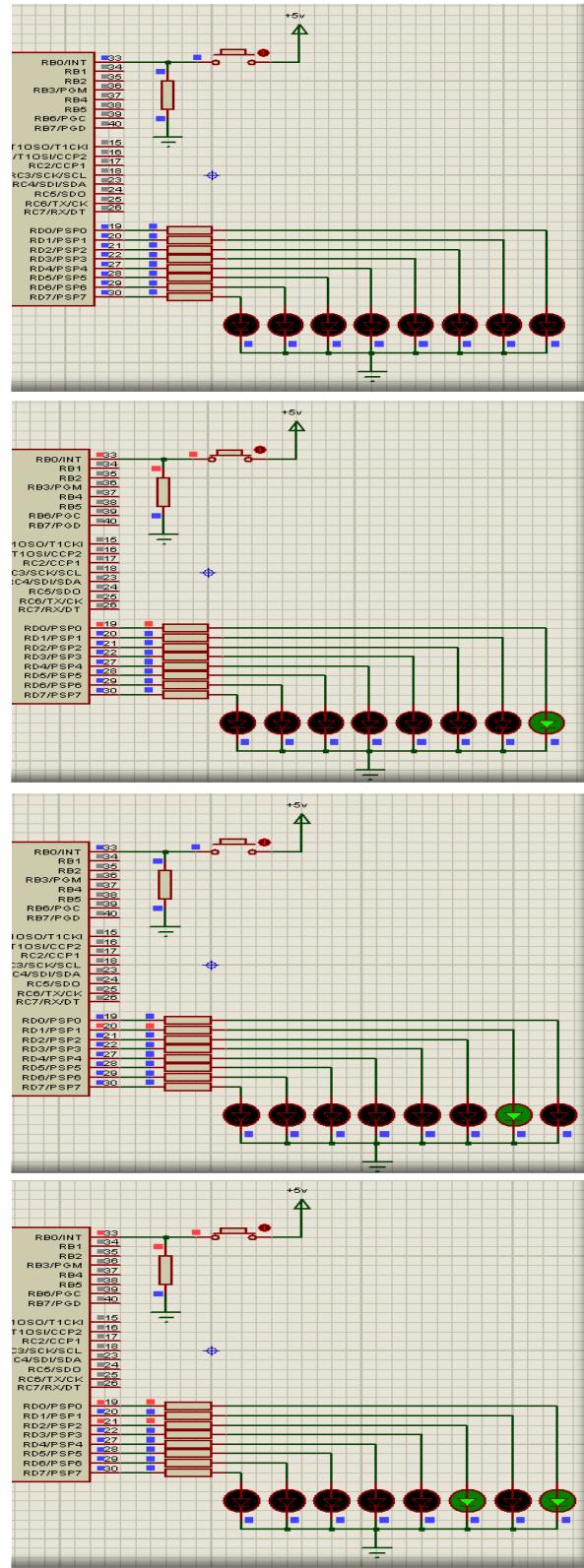
    // Configuración de Puertos

    // Inicialización de Puertos
    output_d(0);

    // Bucle Principal
    while (1){
        // Instrucciones del programa

        if(ultimo_bit==1)
        {
            dato<<=1;
            dato|=ultimo_bit;
        }
        else
        {
            dato<<=1;
        }
        if(input(PIN_B0)==1)
        {
            dato|=0x01;
        }
        output_d(dato);
        delay_ms(500);
        ultimo_bit=bit_test(dato,7);

    } // end while
} //end main
}
```



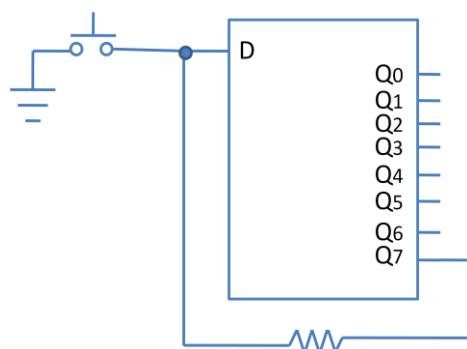


En este caso usamos la instrucción “**dato|=0x01;**” para hacer la operación OR entre el registro dato y un 0x01 cada vez que presionemos el pulsador, tal que un “1” se estará montando en cada desplazamiento. Por otro lado para detectar si hay un “1” en el ultimo bit usamos un IF junto con la instrucción “**ultimo\_bit=bit\_test(dato,7);**” que guardara en el registro ultimo\_bit solo el bit 7 del registro dato para posteriormente usar nuevamente la operación OR y montar un “1” si lo hay, si no lo hay simplemente desplazara la información normalmente.

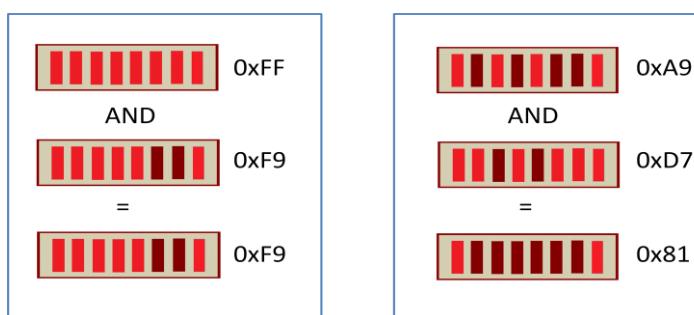
Este efecto no es muy usual en una aplicación, pero su importancia radica en la utilización de la **operación OR para combinar el contenido de dos registros**, que si es aplicable en codificación y enmascaramiento de datos.

### Demo 14 - Eliminando Bits a un Corrimiento

Se tiene un circuito que genera corrimientos en un puerto de un microcontrolador, el circuito siempre está generando corrimientos hacia la izquierda, al comienzo solo tiene unos. Los datos son introducidos por medio de un pin en un segundo puerto, cuando un dato es introducido por este, el dato es montado sobre al inicio del corrimiento. Genere un programa que realice este efecto.



Este caso es similar al anterior con la diferencia que ahora introduciremos ceros, sin embargo la operación OR ya no nos es funcional ya que no tendría efecto sobre el registro, en este caso tendremos que usar la operación AND, auxiliándonos de ella para eliminar bits.



# Manual – Aplicaciones Prácticas con Microcontroladores PIC

Elaborado por: M.I.E. Hugo Antonio Méndez Guzmán



```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )

// Definiciones Globales

// Definiciones de Variables Globales

// Declaración de Subrutinas o Métodos

// Programa Principal
void main ()
{
    // Definiciones de Variables Locales
    int dato=0xFF;
    int1 ultimo_bit=1;

    // Configuración de Puertos

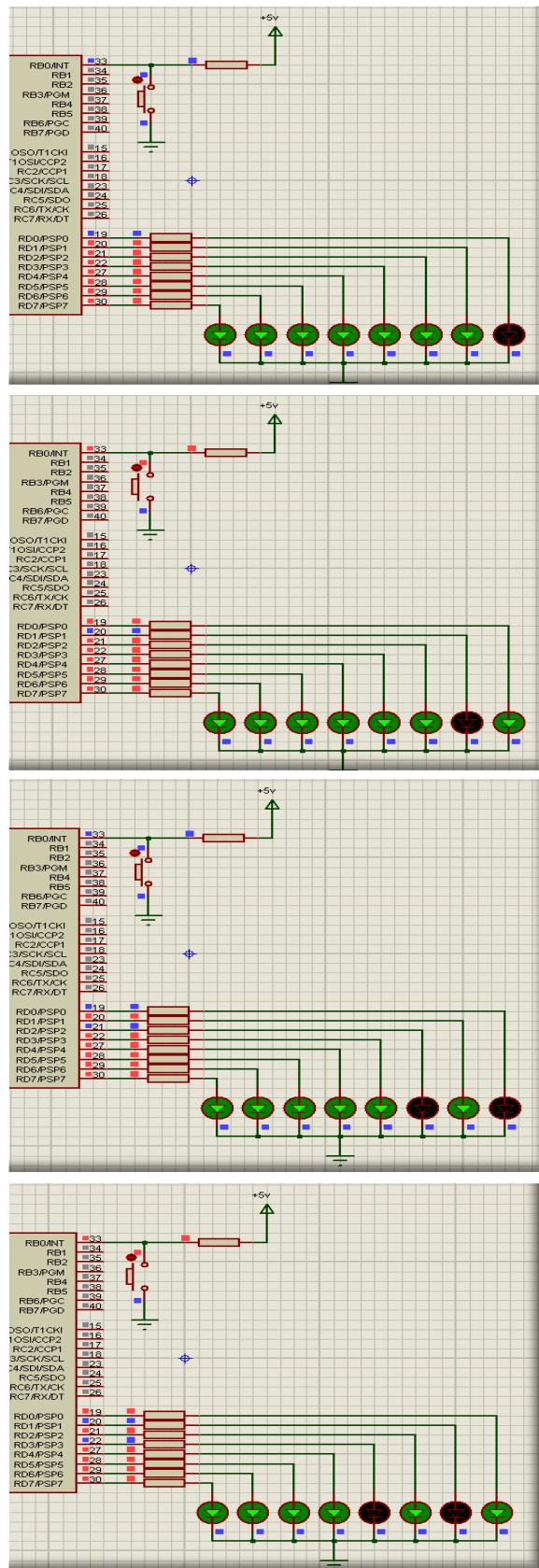
    // Inicialización de Puertos
    output_d(0xFF);
    delay_ms(1000);

    // Bucle Principal
    while (1){
        // Instrucciones del programa

        if(ultimo_bit==1)
        {
            dato<<=1;
            dato|=ultimo_bit;
        }
        else
        {
            dato<<=1;
        }
        if(input(PIN_B0)==0)
        {
            dato&=0xFE;
        }
        output_d(dato);
        delay_ms(500);
        ultimo_bit=bit_test(dato,7);

    } // end while
} //end main
}

```



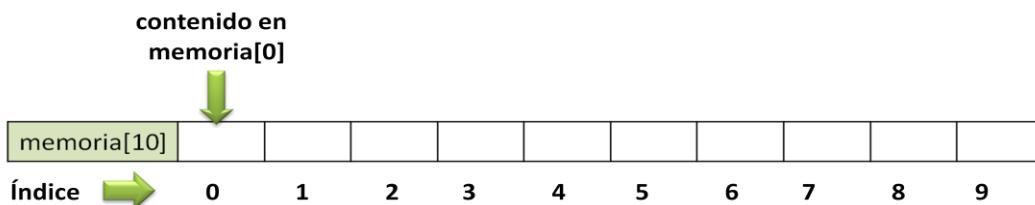


Al igual que el anterior el efecto no es usual en una aplicación, pero su importancia radica en la utilización de la **operación AND para la eliminación de contenido en un registro**, también aplicable en codificación y enmascaramiento de datos.

## USO DE ARREGLOS

Un arreglo es un espacio de memoria que tiene varios registros para almacenar información, de tal forma que para leer o escribir un dato en ellos serán referenciados por un índice o posición en memoria.

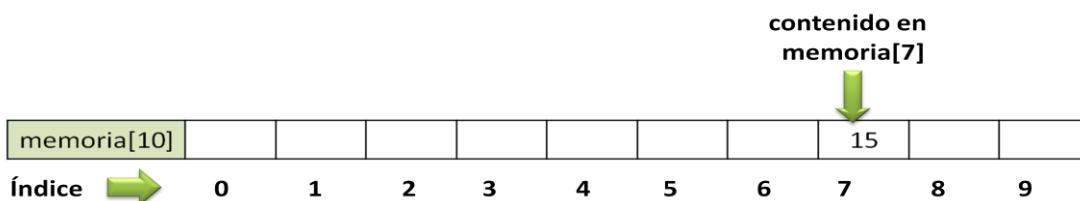
Un arreglo se declara al igual que una variable, tal que podemos hacer un arreglo de números enteros, flotantes, etc.



Suponiendo que deseamos escribir el valor “15” en la posición “7” del arreglo tendríamos que programar:

```
memoria[7]=15;
```

teniendo como resultado:



de igual manera para leer el contenido de un arreglo y transferirlo a otra variable sería

```
dato=memoria[7];
```





La declaración de un arreglo se puede realizar de dos formas, una en la cual solo reservamos la cantidad de datos que queremos almacenar.

```
int arreglo[10];
```

Y otra en la cual le asignamos valores iniciales a cada posición del arreglo,

```
int arreglo[]={ 10 , 15 , 7 , 6 , 5 , -1 , 8 , 20 , 15 , 10 }
```

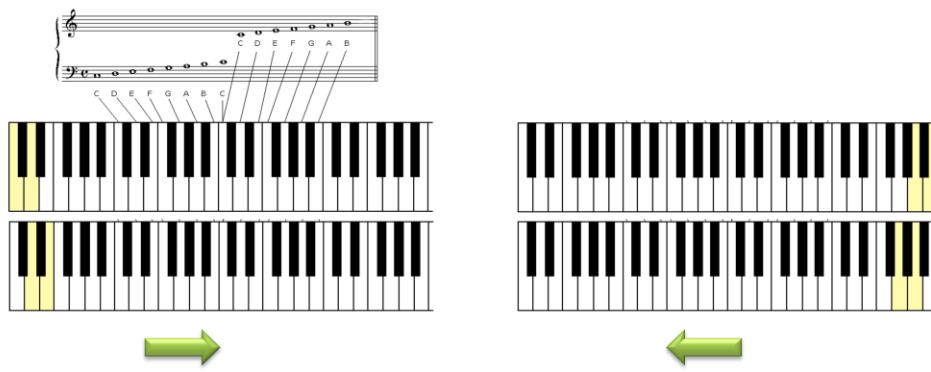
resultando

índice	0	1	2	3	4	5	6	7	8	9
arreglo[10]	10	15	7	6	5	-1	8	20	15	10

Estas dos formas de declarar un arreglo tienen una diferencia importante, mientras en la segunda asignamos valores iniciales, en la segunda no, por lo que en la primera forma podríamos tener inicialmente cualquier valor en cada uno de sus registros, por lo que en ocasiones será necesario el hacer una rutina que ponga en "0" cada uno de esos registros.

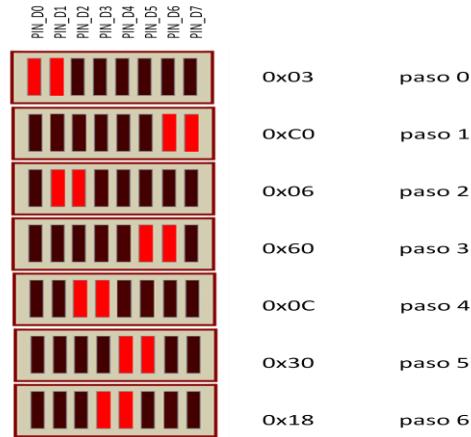
## Demo 15 - Generando un Corrimiento de Datos por medio de Arreglos

Se requiere generar un circuito de corrimiento para hacer la prueba de un piano digital, el corrimiento tendrá que hacer un recorrido de dos teclas en dos teclas, partiendo de un extremo a otro, acortando su distancia a cada paso hasta llegar al centro. Genere un programa que genere un corrimiento como el que se solicita.



# Manual – Aplicaciones Prácticas con Microcontroladores PIC

Elaborado por: M.I.E. Hugo Antonio Méndez Guzmán



El efecto que se pide en este caso requiere una secuencia, donde los datos varían y no hay constancia, por lo que se puede realizar un arreglo para almacenar los datos que estaremos utilizando y luego llamarlos en cada paso.

```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )

// Definiciones Globales

// Definiciones de Variables Globales

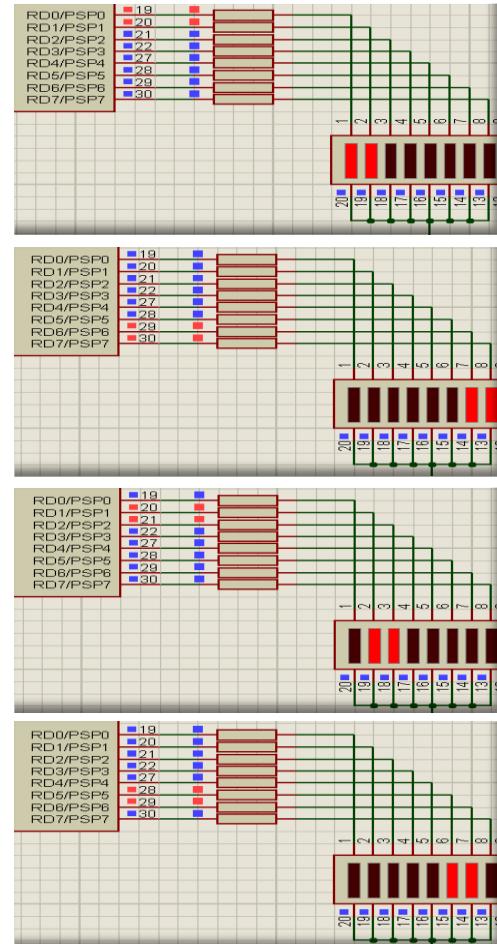
// Declaración de Subrutinas o Métodos

// Programa Principal
void main ()
{
    // Definiciones de Variables Locales
    int contador;
    int datos[]={0x03,0xC0,0x06,0x60,0x0C,0x30,0x18};

    // Configuración de Puertos

    // Inicialización de Puertos

    // Bucle Principal
    while (1){
        // Instrucciones del programa
        for(contador=0;contador<7;contador++)
        {
            output_d(datos[contador]);
            delay_ms(500);
        }
        } // end while
} //end main
```

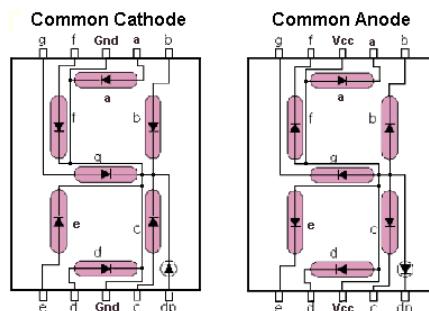




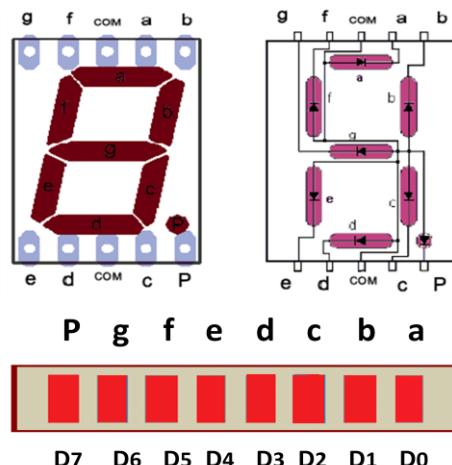
Como se puede apreciar en el programa en un arreglo llamado “datos” almacenamos cada uno de los códigos que requerimos para posteriormente ser llamados por medio de un contador, el cual empezara desde 0 hasta el 6 y por medio de “datos[contador]” hacemos el llamado a cada uno de esos datos y que saldrán al puerto por medio de la instrucción “output\_d”.

## USO DE DISPLAY 7 SEGMENTOS

Un display de 7 segmentos es un conjunto de leds colocados de tal manera que al iluminarse pueden formar dígitos. Dependiendo del tipo, sea ánodo o cátodo común son los niveles lógicos que debemos enviar a él para iluminar cada uno de sus segmentos, ya que en el caso del ánodo común, todos sus ánodos se encuentran interconectados, tal que para polarizar cada uno de los leds tendríamos que enviar un “0” a cada uno de sus segmentos y en el caso de cátodo común, enviar “1” a cada uno de ellos.

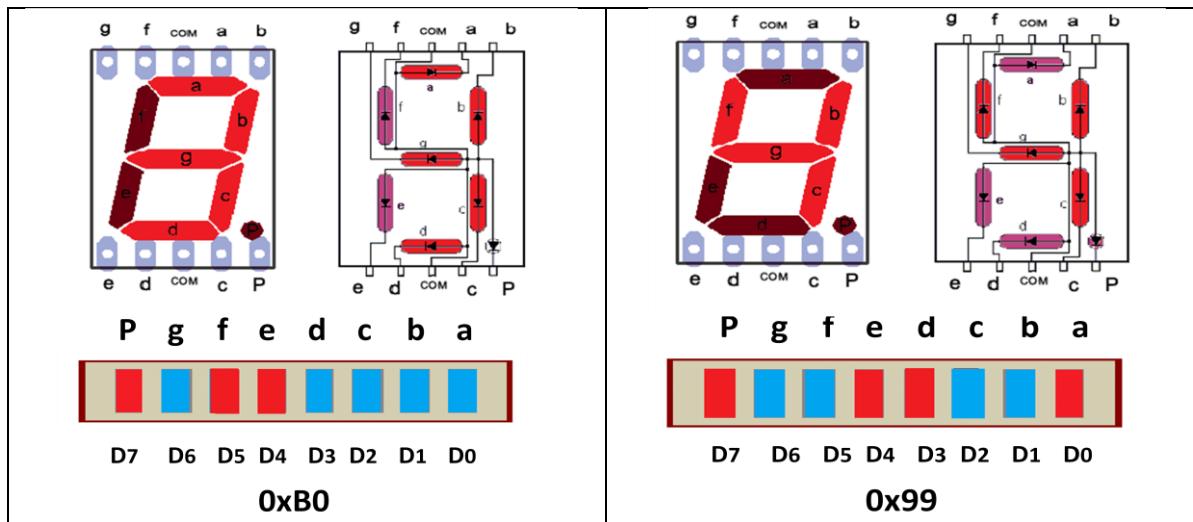


Si queremos realizar una conexión tal que el microcontrolador produzca cada uno de los dígitos, será necesario el verificar la cantidad de “0”s y “1”s a enviar para cada uno de sus segmentos, tal que si la conexión se realiza tal como sigue:





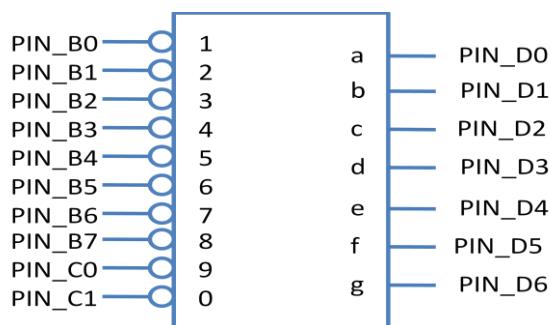
tendremos que analizar que códigos enviar para un Display ánodo común. Entonces para iluminar sus segmentos tendremos que enviar “0” y para apagarlos “1” formándose códigos como los de la figura.



Siendo de esta manera solo tendremos que preocuparnos en un programa en C, de enviar el código correcto cuando queramos iluminar un digito.

### Demo 16 - Manejo Básico de Display

El siguiente programa realiza la función de un codificador de decimal a 7 segmentos, tal que un puerto de nuestro microcontrolador será usado como las entradas y otro puerto como la salida hacia un display de 7 segmentos ánodo común.



# Manual – Aplicaciones Prácticas con Microcontroladores PIC

Elaborado por: M.I.E. Hugo Antonio Méndez Guzmán



```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )

// Programa Principal
void main ()
{
    // Definiciones de Variables Locales

    // Configuración de Puertos

    // Inicialización de Puertos
    output_d(0xC0);
    // Bucle Principal
    while (1){
        // Instrucciones del programa
        if(input_state(PIN_B0)==0)
            output_d(0xF9); // UNO

        if(input_state(PIN_B1)==0)
            output_d(0xA4); // DOS

        if(input_state(PIN_B2)==0)
            output_d(0xB0); // TRES

        if(input_state(PIN_B3)==0)
            output_d(0x99); // CUATRO

        if(input_state(PIN_B4)==0)
            output_d(0x92); // CINCO

        if(input_state(PIN_B5)==0)
            output_d(0x82); // SEIS

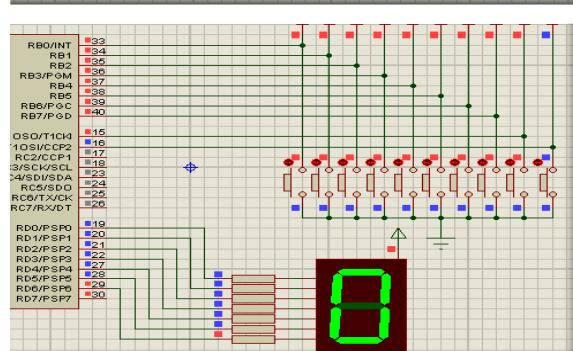
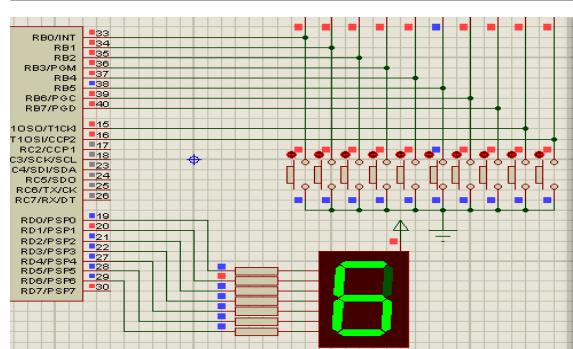
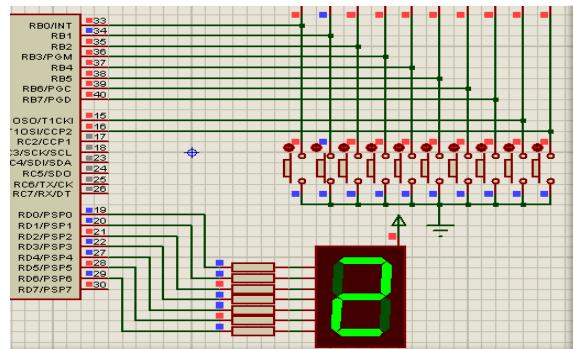
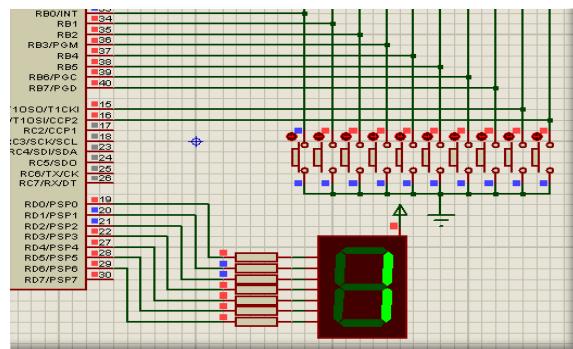
        if(input_state(PIN_B6)==0)
            output_d(0xF8); // SIETE

        if(input_state(PIN_B7)==0)
            output_d(0x80); // OCHO

        if(input_state(PIN_C0)==0)
            output_d(0x90); // NUEVE

        if(input_state(PIN_C1)==0)
            output_d(0xC0); //CERO

    } // end while
} //end main
```



⬅



## Demo 17 - Manejo de Display por medio de Arreglos

Al igual que en programas anteriores podemos asignar a un arreglo los códigos que queramos usar, y tomando en cuenta que el primer índice del arreglo es 0, por lo que el código del Display 0 será el primer elemento del arreglo. El efecto sería el mismo pero de forma más clara podemos ver que dato es el que se está mandando al puerto.

```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )

// Programa Principal
void main ()
{
// Definiciones de Variables Locales
int digito[]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};

// Inicialización de Puertos
output_d(digito[0]);

// Bucle Principal
while (1){
    // Instrucciones del programa
    if(input_state(PIN_B0)==0)      output_d(digito[1]);
    if(input_state(PIN_B1)==0)      output_d(digito[2]);
    if(input_state(PIN_B2)==0)      output_d(digito[3]);
    if(input_state(PIN_B3)==0)      output_d(digito[4]);
    if(input_state(PIN_B4)==0)      output_d(digito[5]);
    if(input_state(PIN_B5)==0)      output_d(digito[6]);
    if(input_state(PIN_B6)==0)      output_d(digito[7]);
    if(input_state(PIN_B7)==0)      output_d(digito[8]);
    if(input_state(PIN_C0)==0)      output_d(digito[9]);
    if(input_state(PIN_C1)==0)      output_d(digito[0]);

} // end while
} //end main
```





## Demo 18 - Contador Continuo de 0 a 20

Para generar un contador y separar los dígitos por valor posicional, uno de los métodos más sencillos es usar un contador que haga la cuenta y separar los dígitos por medio de dos operaciones, la división y el modulo. Donde la división nos dará el valor posicional más significativo y el modulo el menos significativo. Por ejemplo, si se quisiera separar el número 19 en decenas y unidades, tendríamos que realizar:

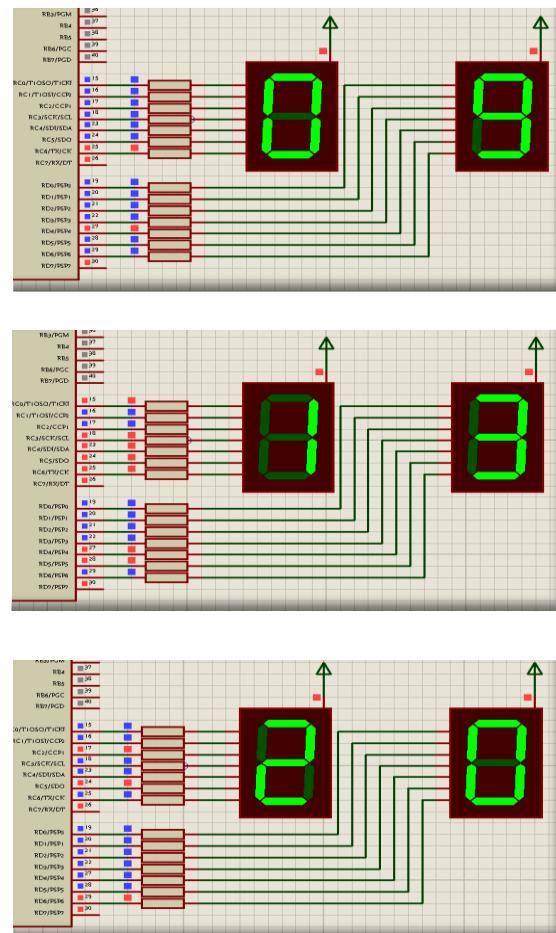
$$\begin{array}{r} 1 \\ 10 \sqrt{19} \\ 9 \end{array} \quad \begin{array}{l} \text{decena}=19/10 \\ \text{unidad}=19\%10 \end{array}$$

Si aprovechamos la propiedad de las variables int de dar solo valores enteros, la decena es igual a un número entre 10, y por otra parte usando la operación del modulo, estaríamos obteniendo la unidad por medio del residuo de un número entre 10. Siendo de esta manera solo tendríamos que mandar el código del Display de las decenas a un puerto y el código del Display para las unidades a otro.

```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )

// Programa Principal
void main ()
{
// Definiciones de Variables Locales
int digito[]={0xC0,0xF9,0xA4,0xB0,0x99,
              0x92,0x82,0xF8,0x80,0x90};
int cont, decena, unidad;

// Bucle Principal
while (1){
    // Instrucciones del programa
    for(cont=0;cont<21;cont++)
    {
        unidad=cont%10;
        decena=cont/10;
        output_c(digito[decena]);
        output_d(digito[unidad]);
        delay_ms(250);
    }
} // end while
} //end main
```





## Demo 19 - Intercambiando un Display de Ánodo Común por un Cátodo Común

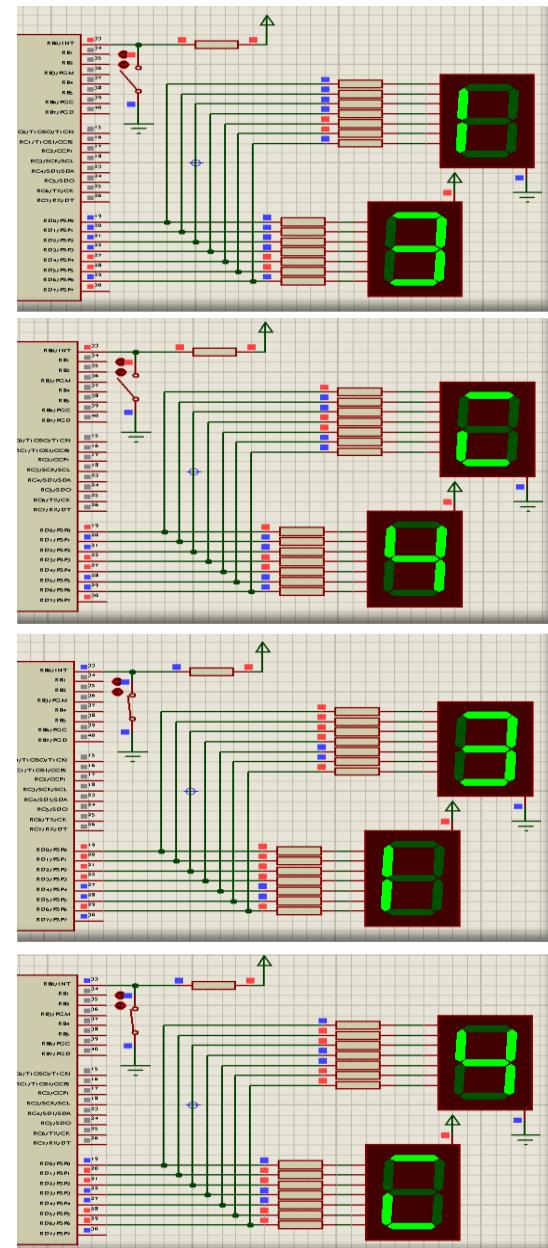
Un caso típico cuando se trabaja con códigos que pueden encender y apagar leds, es que en ocasiones por algún motivo las terminales quedan en una configuración contraria a la que deseamos, o imaginemos que por algún motivo ocupamos una aplicación rápida con dos displays y tenemos solo uno de cátodo y uno de ánodo. Una forma de solucionar este problema es generar los códigos para cada uno de ellos, siendo impráctico. Cuando podemos aprovechar que uno de ellos ocupa el código inverso al otro, siendo suficiente una modificación al código de nuestro programa sin generar códigos nuevos.

```
#include <16f877.h>
#FUSES HS,NOWDT,NOPROTECT,PUT
#USE DELAY( CLOCK=20000000 )

// Programa Principal
void main ()
{
// Definiciones de Variables Locales
int
digito[]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82
,0xF8,0x80,0x90};
int cont;

// Configuración de Puertos

// Inicialización de Puertos
output_d(digito[0]);
// Bucle Principal
while (1){
    // Instrucciones del programa
    for(cont=0;cont<10;cont++)
    {
        if(input(PIN_B0)==1)
            output_d(digito[cont]);
        else
            output_d(~digito[cont]);
        delay_ms(500);
    }
} // end while
}//end main
```





Bajo el diseño que se observa en las imágenes se tienen dos displays, uno de cátodo y otro de ánodo, conectados al mismo puerto pretendiéndose controlar a cual se le envía la información correctamente por medio de un switch al PIN\_B0. Se puede observar que cuando el PIN\_B0 está en “1”, la información se le está enviando correctamente al Display de ánodo común, mientras que cuando el PIN\_B0 está en “0” la información es enviada correctamente al Display de cátodo. Este efecto se puede lograr simplemente complementando los códigos, nosotros ya teníamos formado el arreglo con los códigos del Display de ánodo común y sabemos que uno es el contrario al otro, por lo que solo es necesario invertir los códigos antes de que salgan al puerto, esta operación se realiza por medio del símbolo “~” que es el operador complemento a 1. De tal manera que cuando el PIN\_B0 está en “1” saldrán los códigos de forma normal y cuando este en “0” los invertiremos.

Este operador tan simple es de gran utilidad cuando trabajamos con matrices de leds, ya que si por algún motivo no existe en el mercado la matriz que requerimos, solo tenemos que generar el programa para un tipo de matriz y si tenemos la contraria solo tendremos que usar el operador complemento a 1 para llevarlo a cabo.





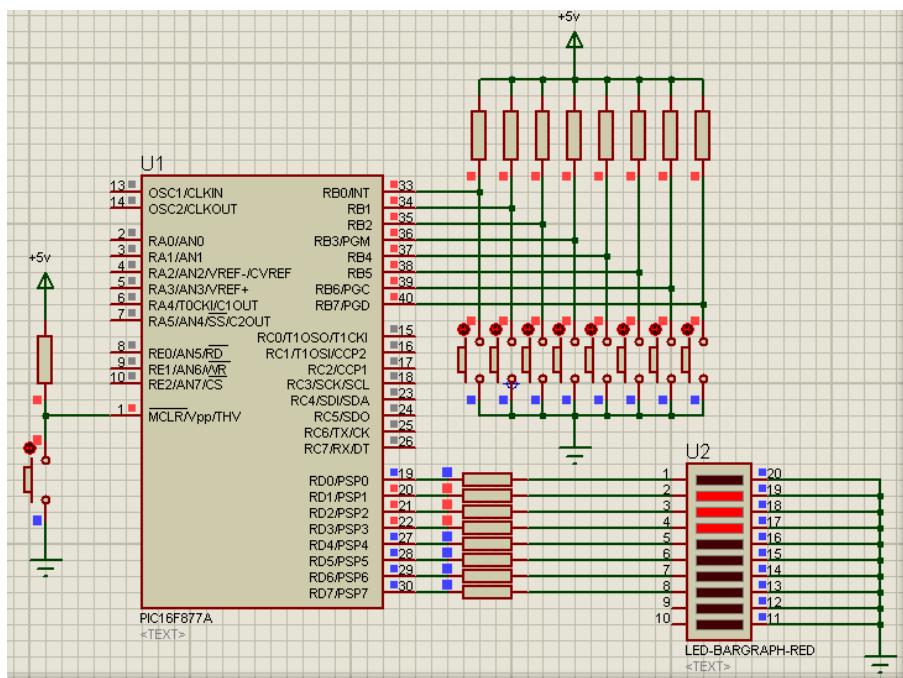
## RELACIÓN DE ACTIVIDADES

### ACTIVIDAD 01 – Realiza una función Nula.

#### **Descripción:**

Realice un programa donde se llame a una función, cuyo objetivo es mandar un “1” lógico a tres pines del puerto D, y manda un “0” a los demás pines del puerto.

#### **Resultado Esperado:**



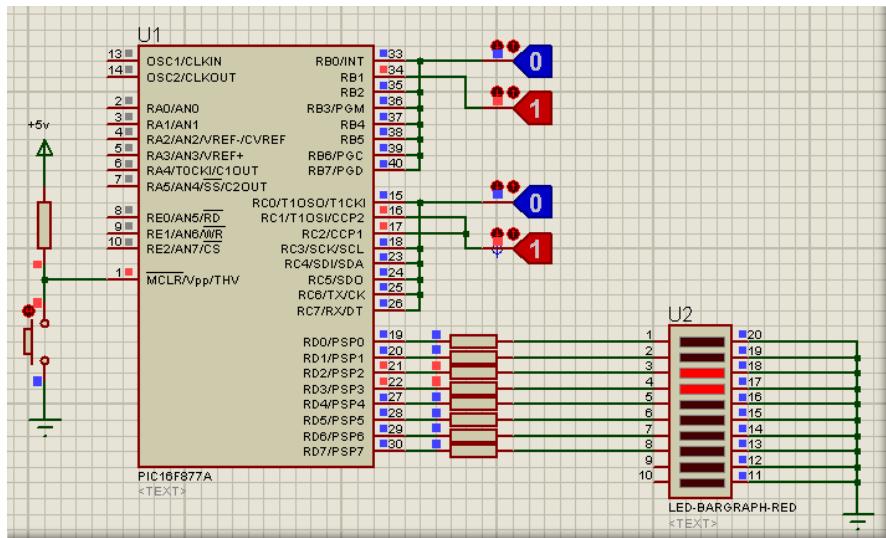


## ACTIVIDAD 02 – Realiza una función con Respuesta.

### Descripción:

Realice un programa que lea la información del puerto B y del puerto C, almacene estos valores en dos variables y por medio de una función con Respuesta devuelva el producto de estos dos valores. Nota: El resultado de la función será mostrado en el puerto D.

### Resultado Esperado:





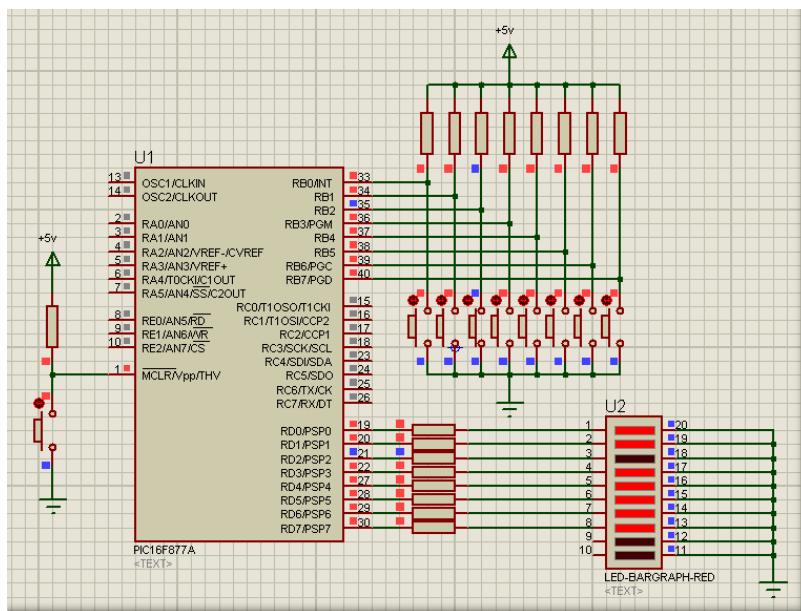
### ACTIVIDAD 03 – Puerto Reflejado.

#### Descripción:

Realice un programa que lea cada uno de los pines del puerto B y muestre su reflejo en cada uno de los pines del puerto D. Si existe un “1” o un “0” en el PIN\_B0, este deberá ser mostrado en el PIN\_D0. Realizar este funcionamiento para cada pin del puerto.



#### Resultado Esperado:



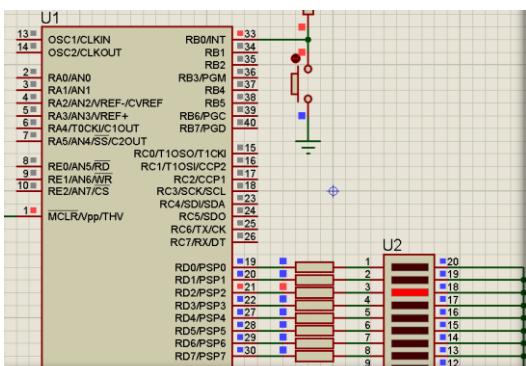
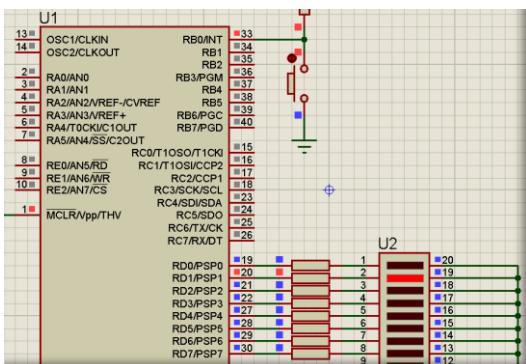
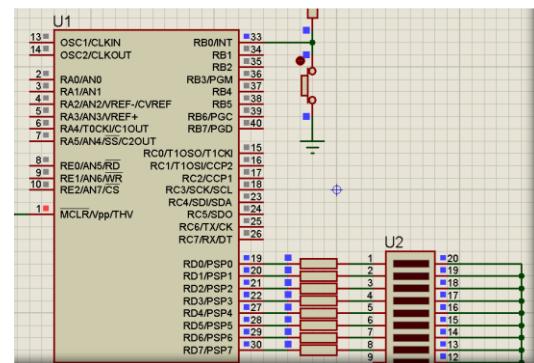
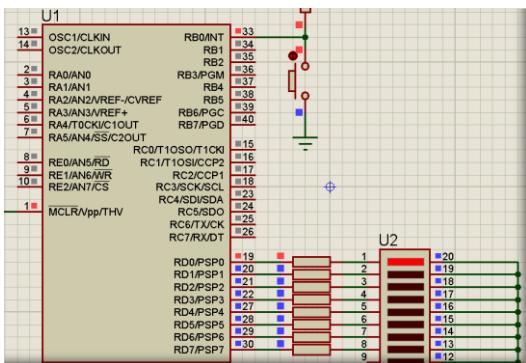


## ACTIVIDAD 04 – Esperando Pulsar para Detener un Corrimiento.

### Descripción:

Realice un programa por medio de un ciclo WHILE, donde el ciclo este generando un corrimiento a través de los pines del puerto D con un retardo de 100 milisegundos entre cada paso. Este corrimiento estará generándose continuamente mientras este sin presionar un pulsador colocado en el PIN\_B0.

### Resultado Esperado:



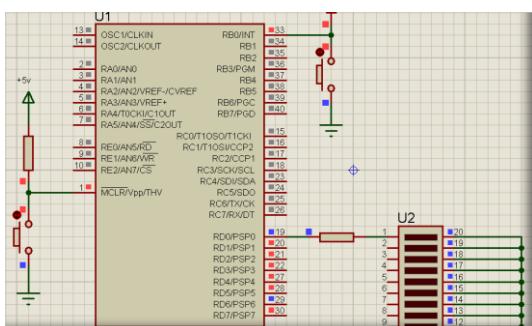
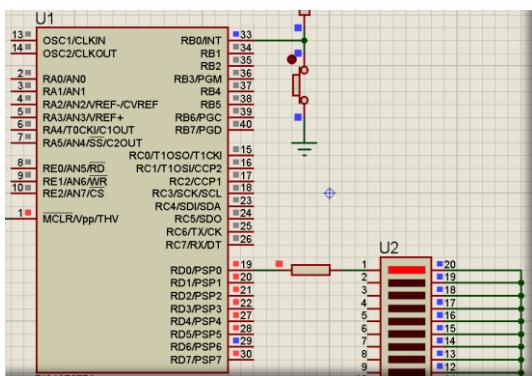
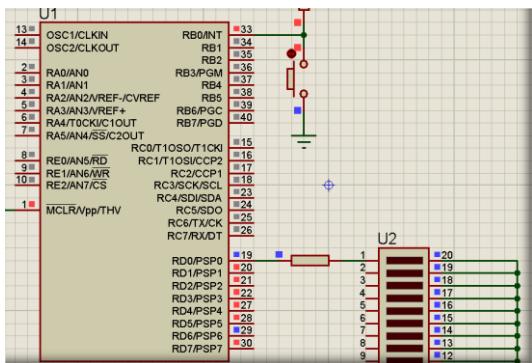


## ACTIVIDAD 05 – Esperando Pulsar para Generar Intermitente.

### Descripción:

Realice un programa que por medio de un ciclo WHILE espere sin realizar ninguna acción la presión de un pulsador. Al continuar con la ejecución del código, el programa deberá realizar el encendido intermitente de un led, por lo menos de 3 veces con un retardo por paso de 500 milisegundos.

### Resultado Esperado:



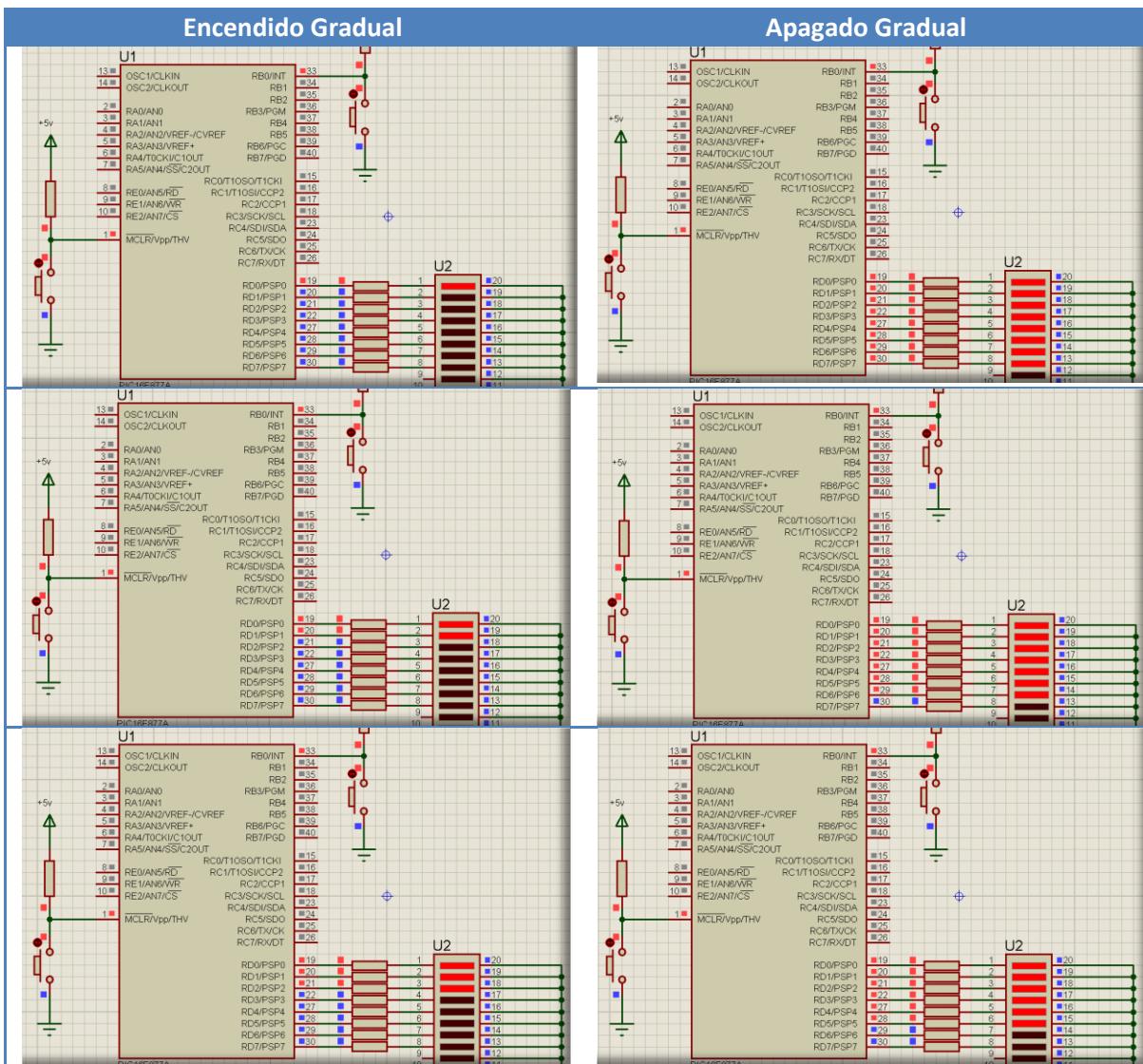


## ACTIVIDAD 06 – Espera Pulsar, Espera Soltar, Encendido Gradual y Apagado Gradual.

### Descripción:

Realice un programa que por medio de ciclos WHILE, espere a que se presione un pulsador, espere a que luego se suelte y finalmente al soltarlo realice el prendido y apagado gradual de leds colocados en el puerto D con un retardo por paso de 200 milisegundos.

### Resultado Esperado:

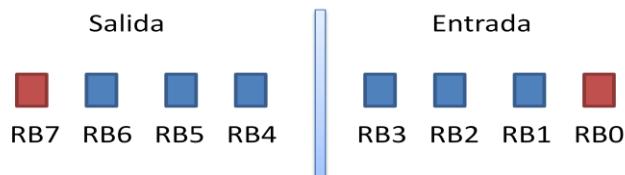




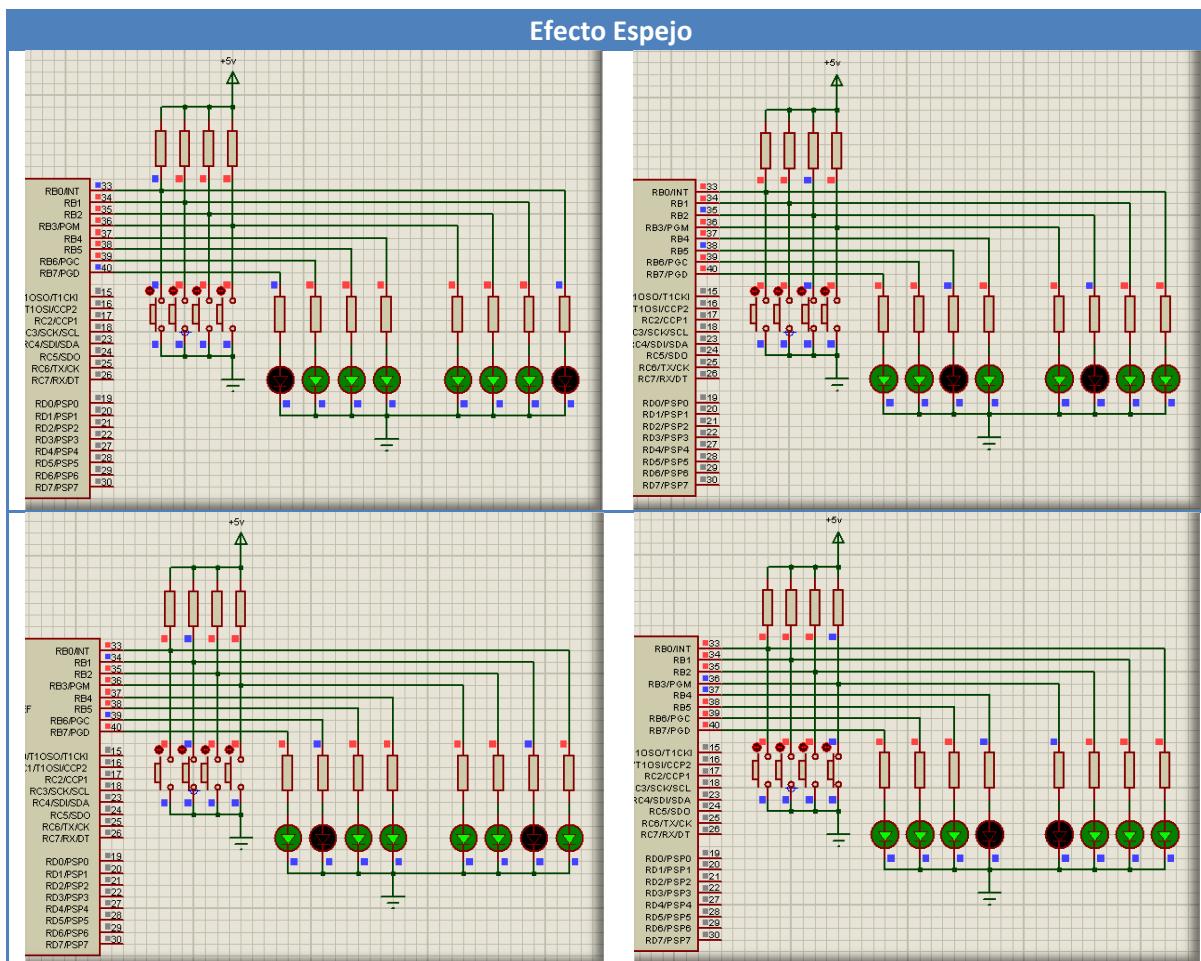
## ACTIVIDAD 07 – Puerto Espejo.

### Descripción:

Genere un programa que lea el contenido de los bits RB0, RB1, RB2 y RB3 y muestre el valor cada uno de ellos en los bits RB7, RB6, RB5 y RB4 respectivamente, de tal manera que simule la presencia de un espejo entre los dos grupos.



### Resultado Esperado:

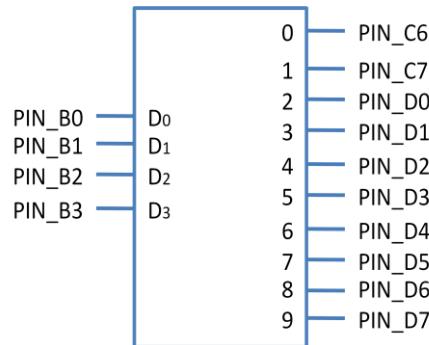




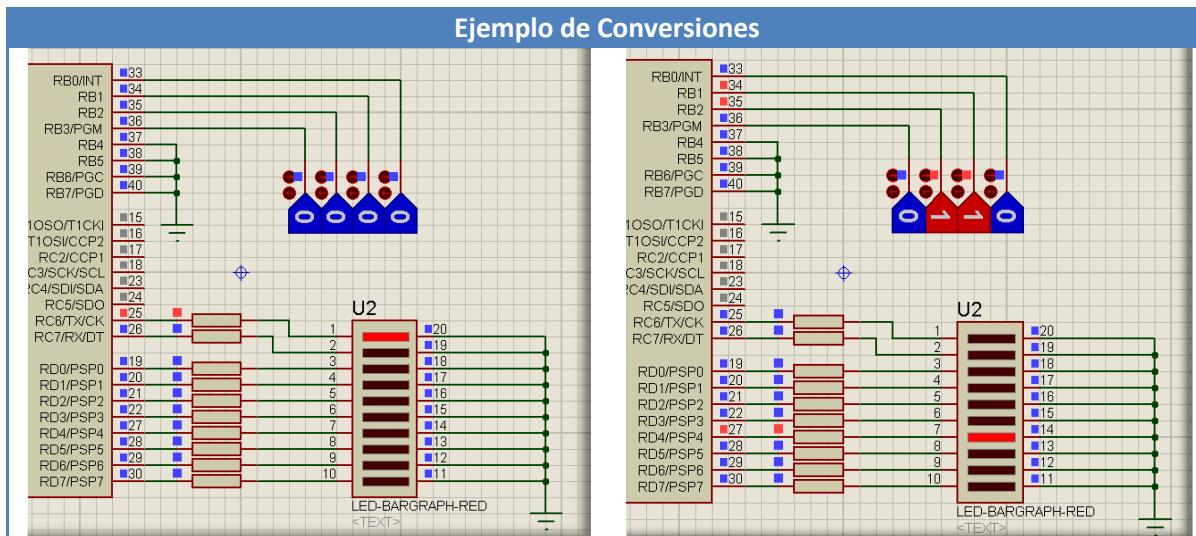
## ACTIVIDAD 08 – Convertidor de Binario a Decimal.

### Descripción:

Genere un programa que realice la función de un convertidor de binario a decimal, tal que para cada combinación en la entrada, desde el “0000” hasta “1001” mande activar una sola salida por cada una de ellas.



### Resultado Esperado:

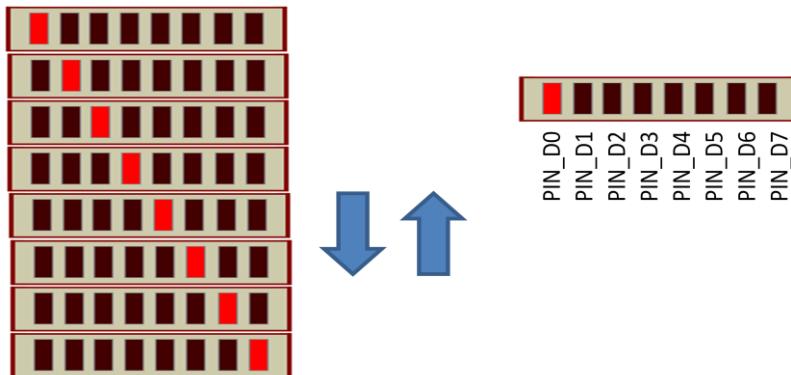




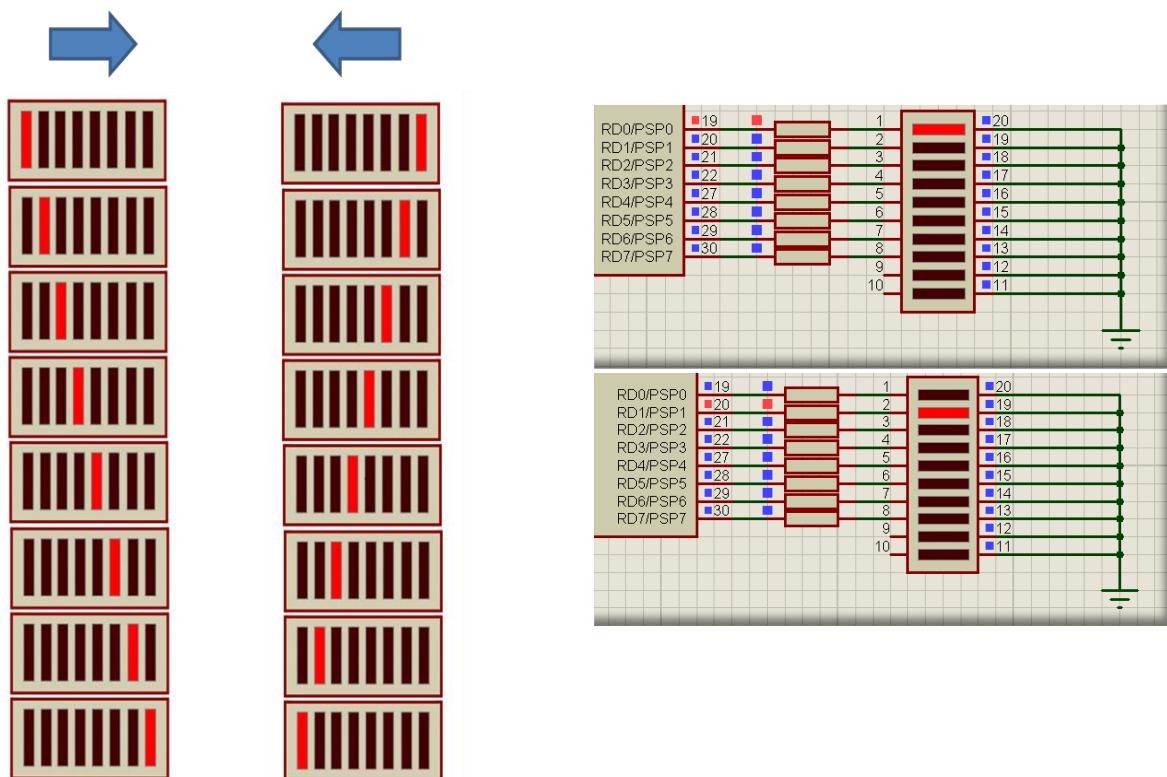
### ACTIVIDAD 09 – Va y Viene.

#### Descripción:

Genere un programa que realice una secuencia de datos, donde solo un bit se estará desplazando de izquierda a derecha a través de un puerto tal como se muestra en la figura. Una vez concluido el recorrido tendrá que regresarse, efectuando un corrimiento de derecha a izquierda. Este proceso deberá repetirse indefinidamente



#### Resultado Esperado:

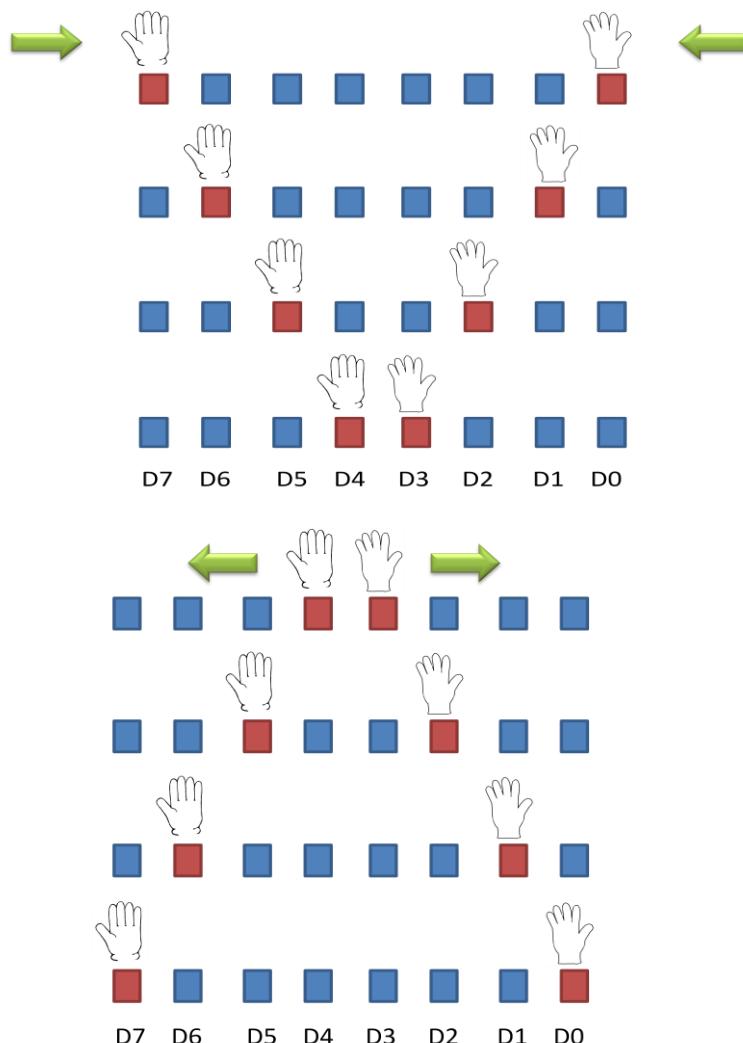




## ACTIVIDAD 10 – Aplausos!!!

### Descripción:

Genere un programa que a través de un arreglo realice un corrimiento en los bits del puerto D, de tal manera que genere el efecto de la figura. Este efecto deberá repetirse indefinidamente con un retardo por paso de 200 milisegundos.

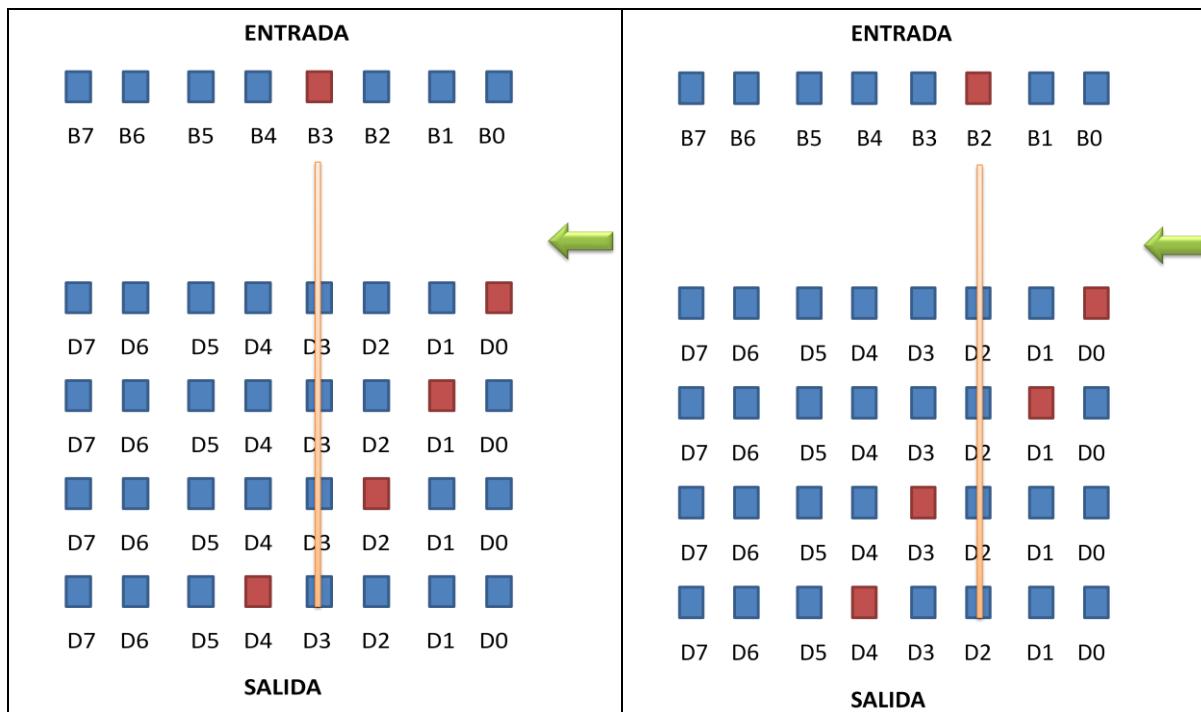




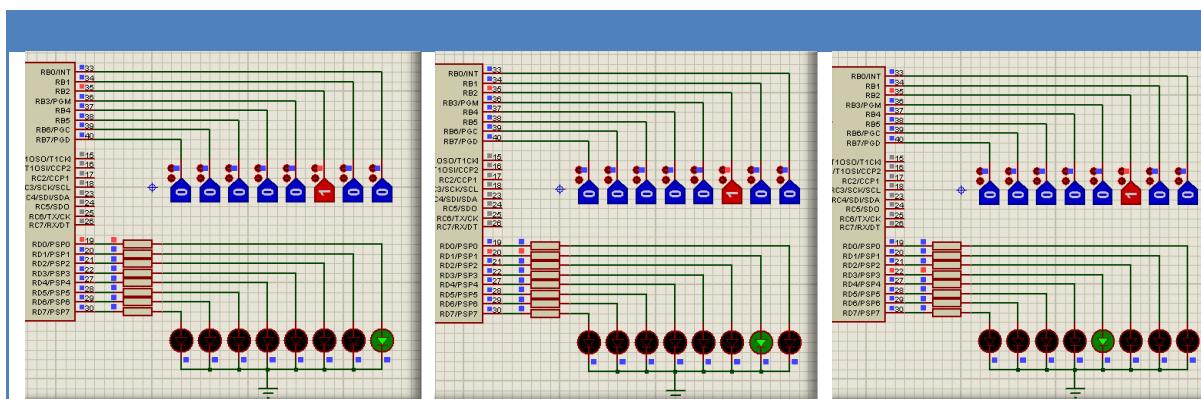
## ACTIVIDAD 11 – Brincando la Cerca

### Descripción:

Genere un programa que realice un corrimiento de datos de un bit en el puerto D. Este corrimiento se efectuará normalmente excepto en el bit que se marque con un “1” en el puerto B, tal que si en el PIN\_B3 hay un “1”, el corrimiento del puerto D deberá pasar por todo el puerto excepto en el PIN\_D3. Se sugiere el uso de operaciones lógicas.



### Resultado Esperado:

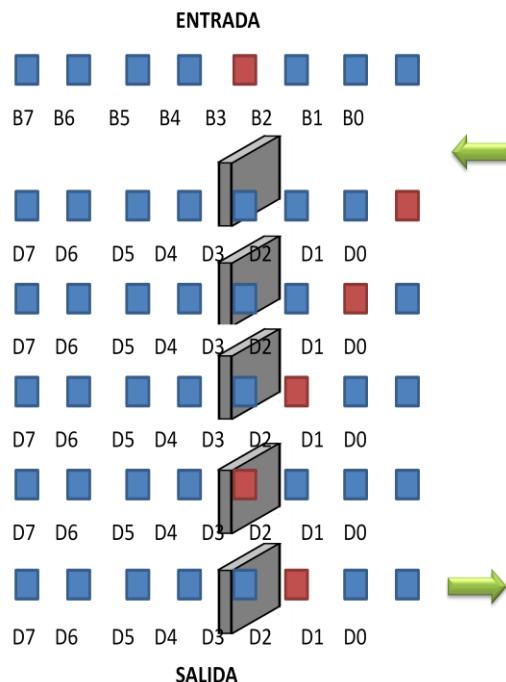




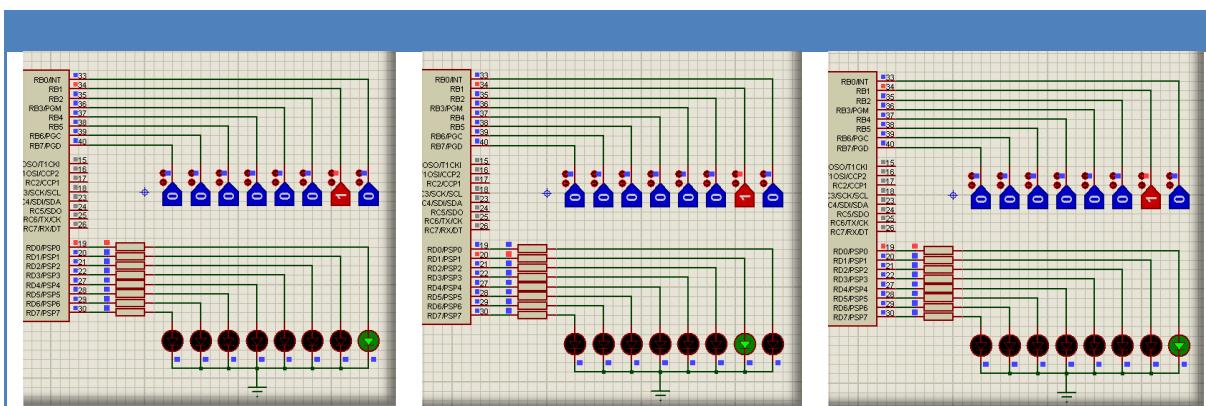
## ACTIVIDAD 12 – Rebotando la Pelota

### Descripción:

Genere un programa que realice un corrimiento de datos de un bit en el puerto D. Este corrimiento se efectuará normalmente hasta el bit que marque el puerto B, tal como si existiese una pared que hace rebotar el bit. Este corrimiento continuará indefinidamente de derecha a izquierda y al rebotar de izquierda a derecha.



### Resultado Esperado:

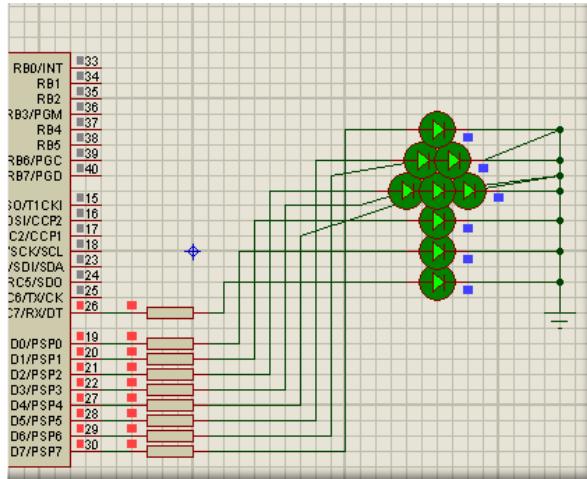
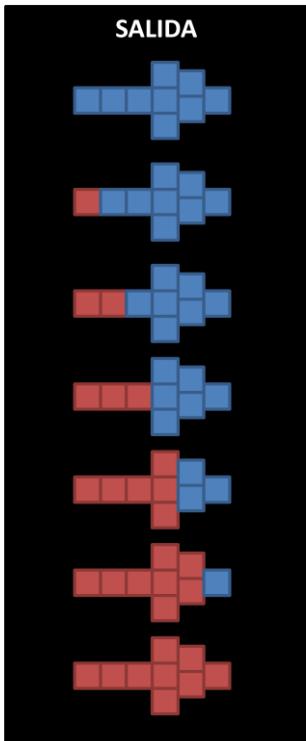




### ACTIVIDAD 13 – Indicando el camino

#### Descripción:

Genere un programa que realice un corrimiento de datos, tal que el efecto visual que se muestre sea como el de la figura. Este corrimiento continuará indefinidamente.





## ACTIVIDAD 14 – Contando pasajeros

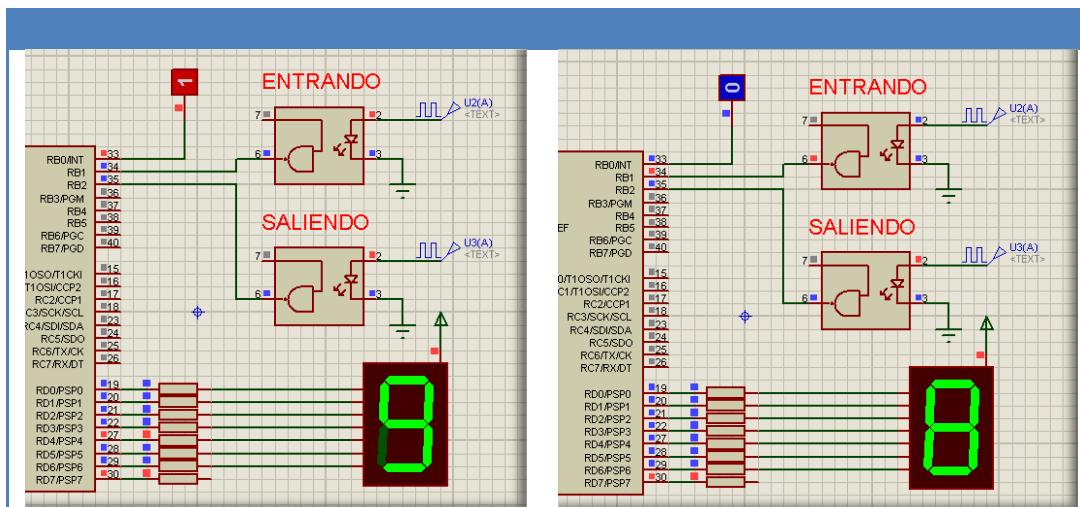
### Descripción:

Se tiene un sistema de sensores en un camión, uno de ellos contabiliza la cantidad de personas que entra al camión, mientras el segundo contabiliza las personas que salen. Al conductor le han dado la orden de no admitir a más de 9 personas en el camión, por lo que tiene que estar atento a la cantidad de personas dentro del camión. Sin embargo el chofer se le complica esta tarea ya que por minuto en promedio suben 3 mientras baja 1 y no puede estar atento mientras conduce. El chofer decide comprar un sistema que cuenta la cantidad de personas dentro del camión mientras suben y bajan y también le da una señal de alerta visual si ya hay nueve personas en el camión.

Realice un programa que realice esta cuenta y de la señal de alarma. Asigne generadores de pulsos para la simulación del programa, donde el sensor que cuenta las personas que entran y el que cuenta las que salen, están generando pulsos a 3Hz y 1Hz respectivamente.



### Resultado Esperado:



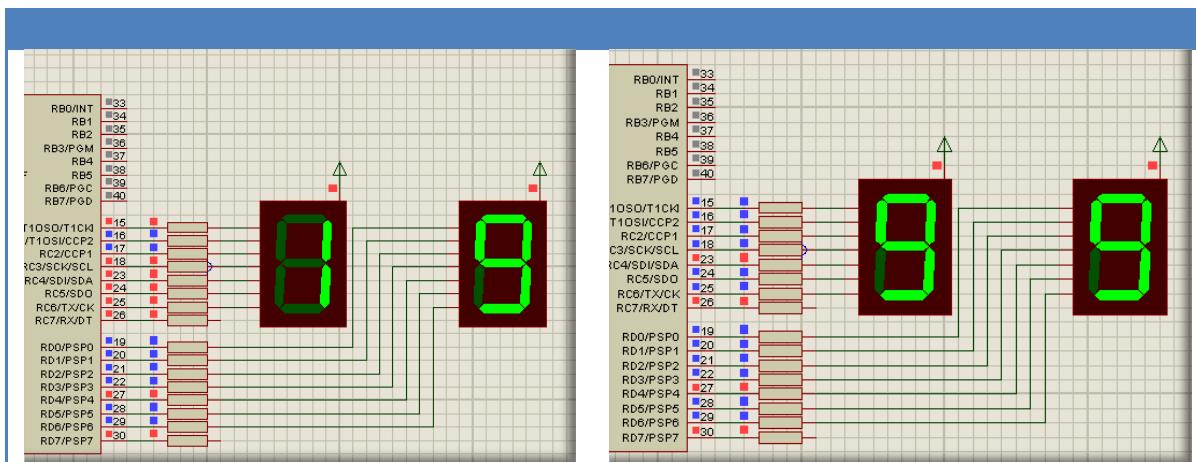


### ACTIVIDAD 15 – Contador continuo de 0 a 99

#### Descripción:

Genere un contador de 0 a 99, donde la cuenta se muestre en dos displays asignados a dos puertos del microcontrolador, uno para las decenas y otro para las unidades.

#### Resultado Esperado:





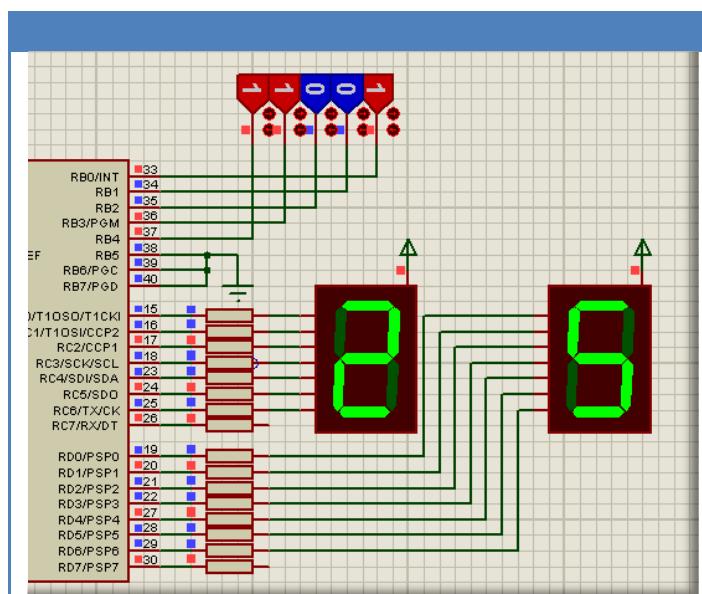
## ACTIVIDAD 16 – Convertidor de binario a decimal extendido

### Descripción:

Genere un programa que lea el contenido de un puerto y la información en el sea convertida a una representación decimal de dos dígitos. Tal que la información se muestre como sigue.



### Resultado Esperado:





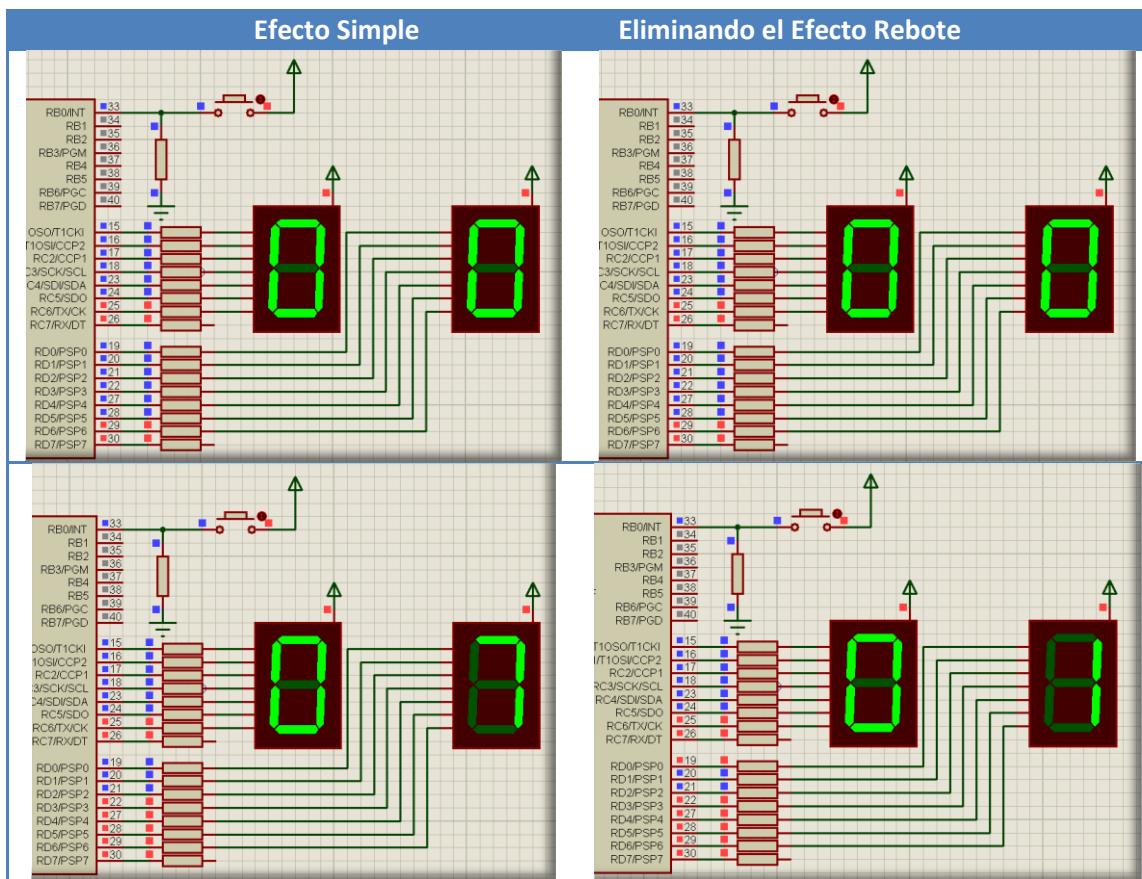
## ACTIVIDAD 17 – Eliminando el Efecto Rebote

### Descripción:

Genere un programa que realice una cuenta sencilla desde 0 a 99 de uno en uno por medio de la activación de un pulsador. Una vez realizado, verifique que es lo que sucede si se presiona el pulsador más de 3 veces en un segundo y anote el valor inicial con el cual comenzó y el valor final.

Posteriormente agregue al programa una rutina que elimine el efecto rebote, de tal manera que si se presiona el pulsador más de tres veces por segundo, solo se vea incrementada la cuenta en 1.

### Resultado Esperado:



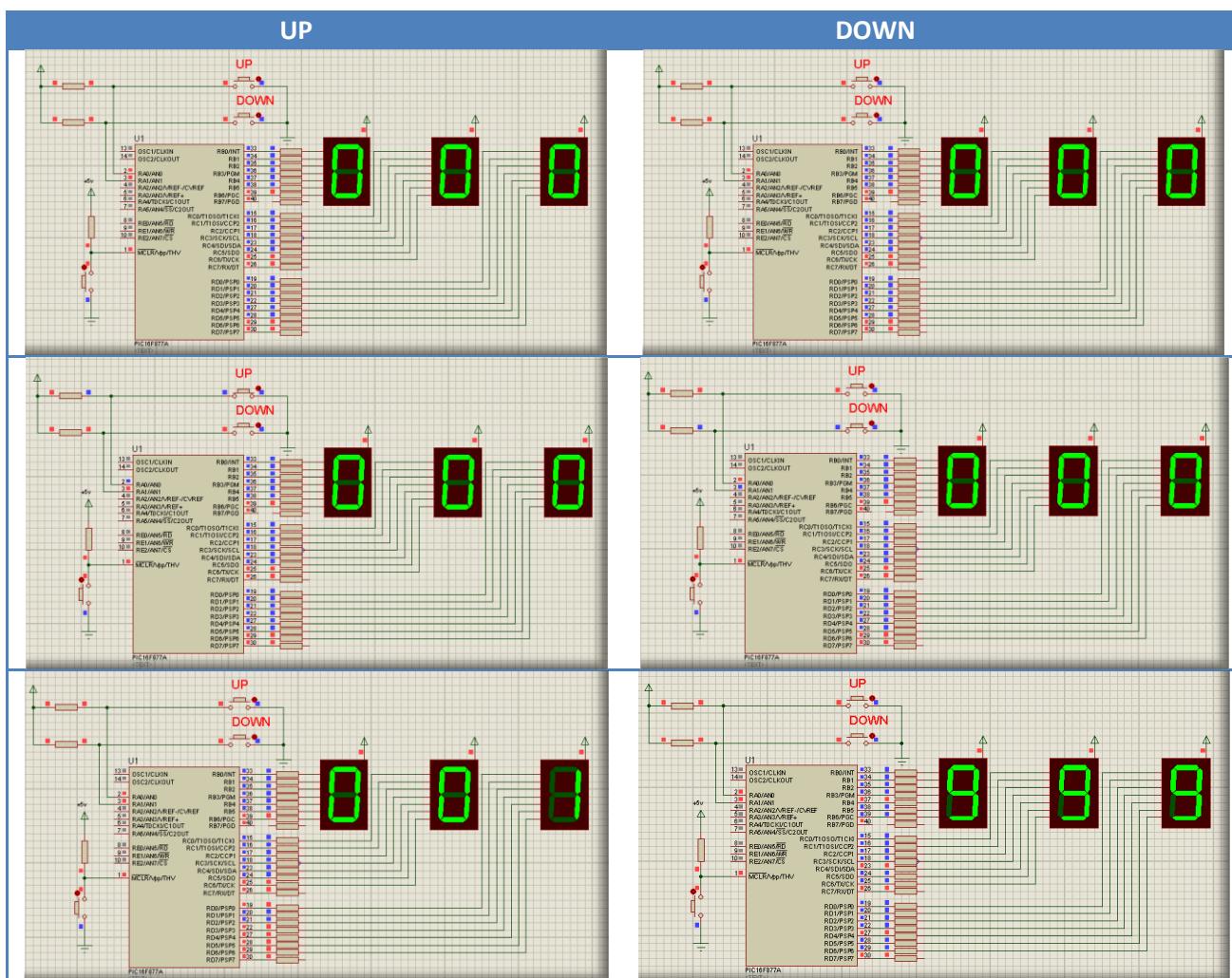


## ACTIVIDAD 18 – Contador continuo de 0 a 999 UP/DOWN.

### Descripción:

Genere un contador de 0 a 999, donde la cuenta se muestre en tres displays asignados a tres puertos del microcontrolador. En un principio el contador no debe realizar ninguna cuenta, si no hasta la presión de un pulsador, cuya función será indicarle al programa que la cuenta es ascendente. Para comenzar la cuenta, este pulsador deberá ser presionado y luego soltado. También existirá un segundo pulsador, el cual al presionarlo y luego soltarlo dará la indicación de que la cuenta ahora es descendente.

### Resultado Esperado:

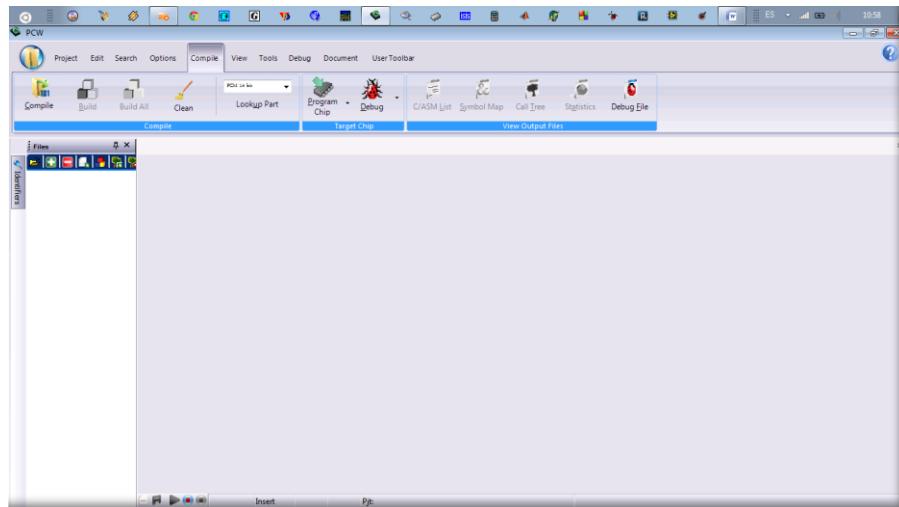




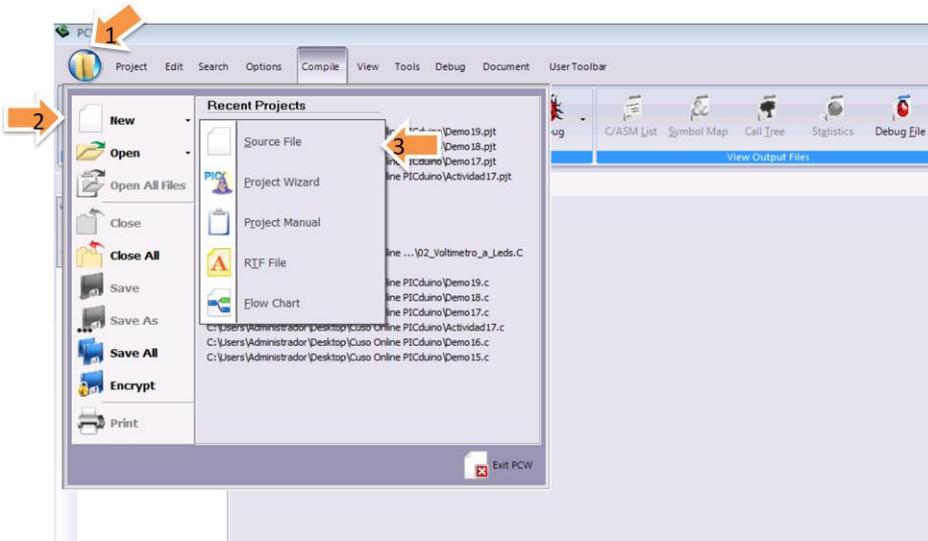
## Apéndice – A

### Compilando un primer programa en Pic C Compiler.

El entorno de desarrollo de Pic C Compiler posee varias herramientas como edición, búsqueda y remplazo de texto, generación de diagramas de flujo, compilación inclusive de programación para nuestro microcontrolador. En nuestro caso, la principal herramienta que usaremos será la pestaña de Compilación.



Para generar un nuevo programa iremos al menú File (1), luego New (2) y finalmente Source File (3) con el fin de generar un archivo de texto donde almacenaremos nuestro código del programa.

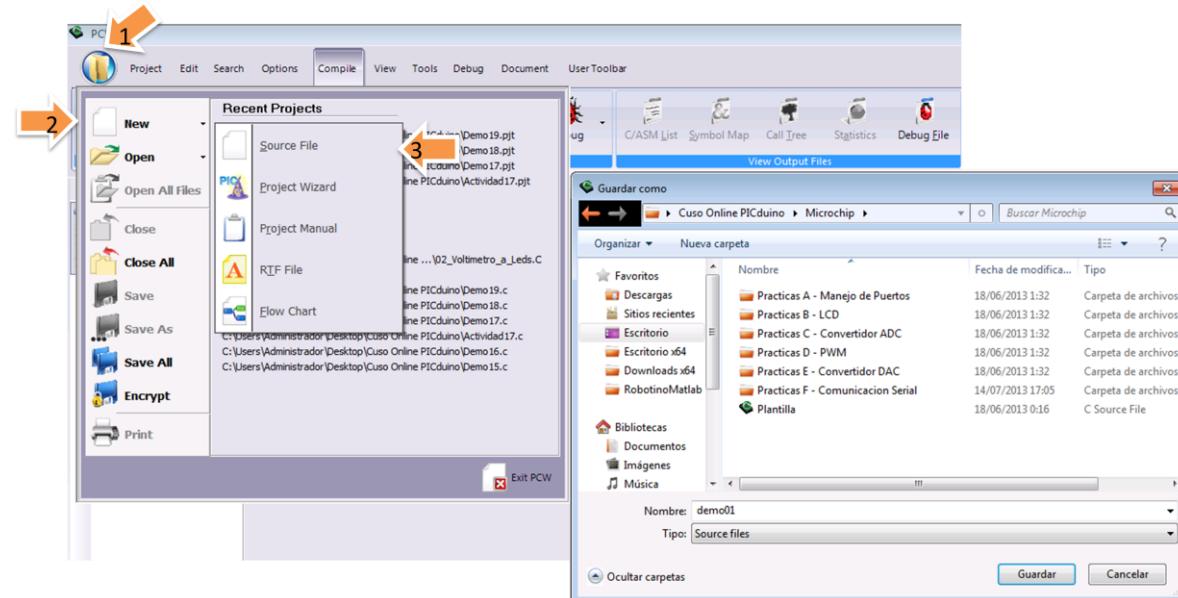


# Manual – Aplicaciones Prácticas con Microcontroladores PIC

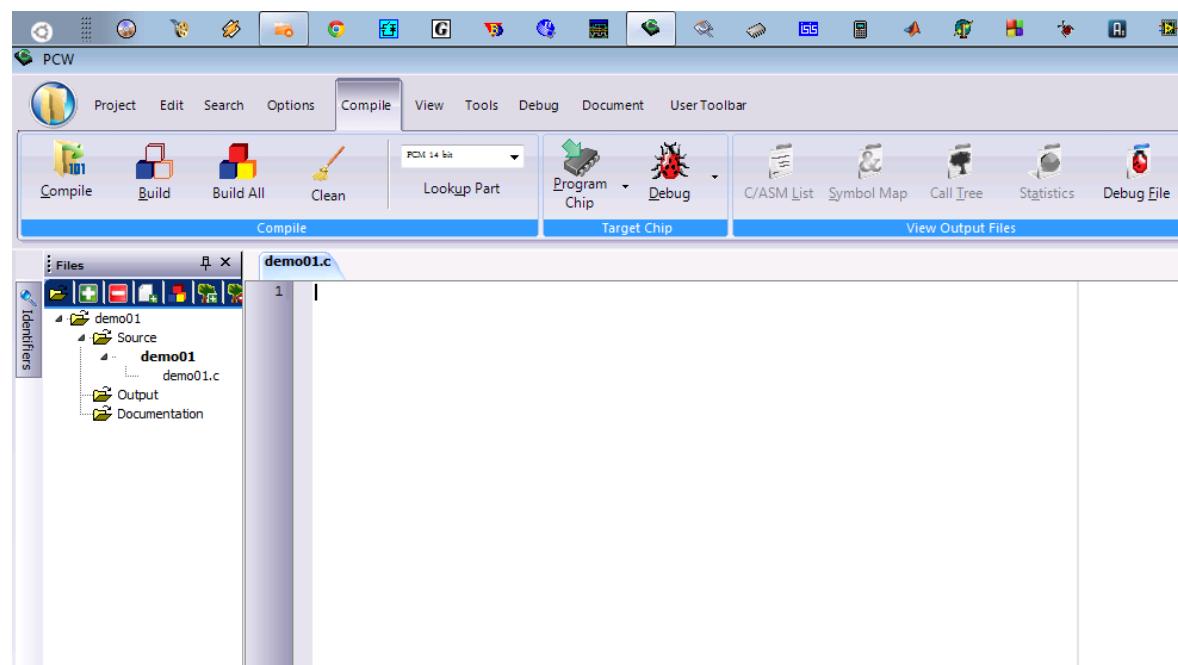
Elaborado por: M.I.E. Hugo Antonio Méndez Guzmán



Una vez hecho esto nos aparecerá una ventana de dialogo donde nos pedirá una ubicación y un nombre para guardar nuestro proyecto.



Una vez guardado nuestro archivo nos aparecerá un archivo de texto en blanco, donde escribiremos nuestros programas.



# Manual – Aplicaciones Prácticas con Microcontroladores PIC

Elaborado por: M.I.E. Hugo Antonio Méndez Guzmán



Para la compilación de nuestro primer programa transcribiremos el código del Demo 01, y le daremos al botón “Build All” para compilar nuestro código.

The screenshot shows the PCW (PIC C Compiler) interface. The toolbar at the top has buttons for Project, Edit, Search, Compile, View, Tools, Debug, Document, and User Toolbar. The 'Compile' button is highlighted with an orange arrow. Below the toolbar is a menu bar with 'PCM 14 kb', 'Lookup Part', 'Program Chip', 'Debug', 'C/ASM List', 'Symbol Map', 'Call Tree', 'Statistics', and 'Debug File'. The main window displays a code editor with the following C code:

```
877.h>
NDT,NOPROTECT,PUT
CLOCK=20000000

// Definiciones Globales
#define prende_pin(x) output_high(x)
#define apaga_pin(x) output_low(x)

// Definiciones de Variables Globales
// Declaración de Subrutinas o Métodos
// Programa Principal
void main ()
{
    // Definiciones de Variables Locales
    // Configuración de Puertos
    // Prueba de Definiciones globales
    prende_pin(PIN_D0);

    // Bucle Principal
    while (1){
        // Instrucciones del programa
        .
        } // end while
    } //end main
}
```

The screenshot shows the PCW interface with the 'Build All' button highlighted. A CCS C Compiler dialog box is overlaid on the main window. The dialog box title is 'CCS C Compiler' and 'PCM Compiler V4.104'. It displays the following information:

Project: C:\...ktop\Curso Online PICduino\Microchip\demo01.c

Complete  
No errors  
Files: 2, Statements: 3, Time: 1 Sec, Lines: 330  
Output files: ERR HEX SYM LST COF PJT TRE STA

RAM: 1%  
ROM: 0%

[www.ccslinfo.com](http://www.ccslinfo.com)



# Manual – Aplicaciones Prácticas con Microcontroladores PIC

Elaborado por: M.I.E. Hugo Antonio Méndez Guzmán



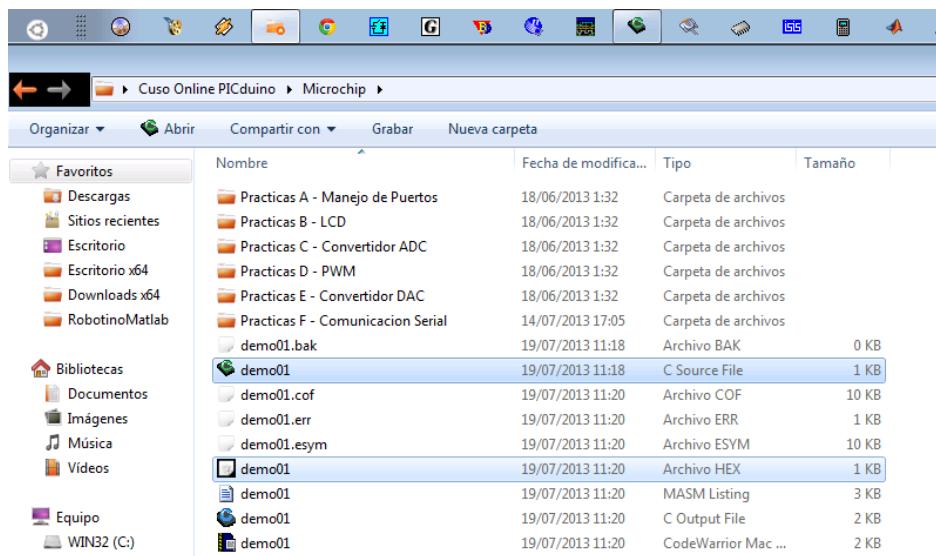
Si no hay errores en nuestro programa, nos aparecerá un mensaje similar al siguiente, donde marca cero errores.

The screenshot shows a software interface with a tab labeled "Output". Inside the window, the following text is displayed:

```
>>> Warning 203 "demo01.c" Line 23(1,1): Condition always TRUE
Memory usage: ROM=0% RAM=1% - 1%
0 Errors, 1 Warnings.
```

Below the window are two buttons: "Compiler" and "Find".

Al compilar nuestro programa se generan varios archivos en el proceso en el lugar donde guardamos nuestro código fuente. De estos, solo serán necesarios dos un archivo .C que es nuestro código fuente y un .hex que son los códigos que serán almacenados en nuestro microcontrolador para realizar nuestra aplicación.



Una vez hecho esto, estamos listos para simular nuestro microcontrolador en Proteus ISIS.



# Manual – Aplicaciones Prácticas con Microcontroladores PIC

Elaborado por: M.I.E. Hugo Antonio Méndez Guzmán



Ahora supongamos que por algún motivo nos equivocamos al escribir nuestro programa, por ejemplo el olvido de un punto y coma.

The screenshot shows the PCW (PIC C Workstation) software interface. The main window displays a C program named 'demo01.c'. The code includes comments and a main function. On line 20, there is a syntax error: 'prende\_pin[PIN\_D0];' is missing a closing semicolon at the end of the line. The line is highlighted in yellow, and the cursor is positioned at the end of the line where the semicolon should be.

```
9 // Declaración de Subrutinas o Métodos
10 // Programa Principal
11 void main ()
12 {
13     // Definiciones de Variables Locales
14     // Configuración de Puertos
15     // Prueba de Definiciones globales
16     prende_pin[PIN_D0];
17
18     // Bucle Principal
19     while (1){
20         // Instrucciones del programa
21
22     } // end while
23 }
```

En el programa eliminaremos un punto y coma a propósito generando errores para ver qué resultados da la compilación.

This screenshot shows the same PCW software interface after the error was corrected. The line 'prende\_pin[PIN\_D0];' now ends with a closing semicolon at the end of the line. The line is no longer highlighted in yellow, indicating that the syntax error has been resolved.

```
9 // Declaración de Subrutinas o Métodos
10 // Programa Principal
11 void main ()
12 {
13     // Definiciones de Variables Locales
14     // Configuración de Puertos
15     // Prueba de Definiciones globales
16     prende_pin[PIN_D0];
17
18     // Bucle Principal
19     while (1){
20         // Instrucciones del programa
21
22     } // end while
23 }
```





Al compilar el programa con este error el software nos genera un mensaje “Error 76 “demo01.c” Line 23(1, 6): Expect ;” el cual nos indica un posible error, en este caso nos dice que esperaba un punto y coma (Expect ;), sin embargo el software no nos envía directamente al lugar donde está este error, sino que generalmente nos ubica una instrucción antes o una instrucción después, por lo que tendremos que verificar en ocasiones antes y después de donde nos señale.

The screenshot shows a software interface for PIC microcontroller development. The main window is a code editor displaying C code for a project named 'demo01'. The code includes comments and a while loop. The 'Output' window at the bottom shows a single error message:

```
*** Error 76 "demo01.c" Line 23(1,6): Expect ;  
1 Errors, 0 Warnings.
```

Muchos de los posibles errores se generan por no escribir correctamente las instrucciones, otras muchas también por punto y comas omitidos, por lo que tendremos que cuidar estos detalles para la correcta compilación de nuestro programa.

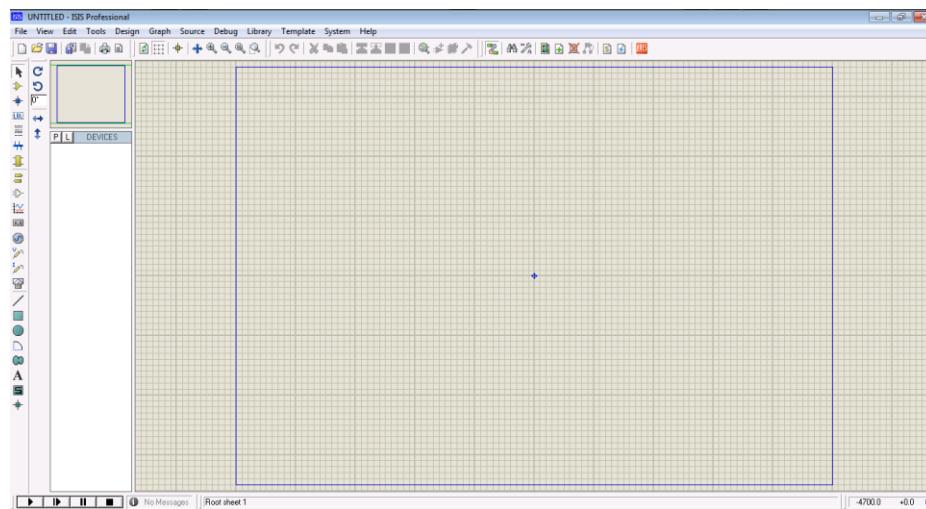




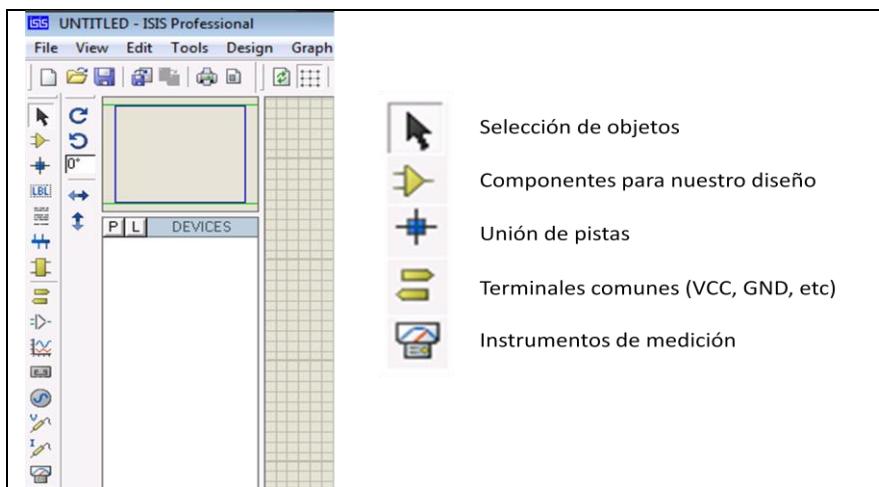
## Apéndice – B

### Armando un primer circuito en Proteus ISIS.

Proteus ISIS es una herramienta muy completa en la simulación de Microcontroladores, inclusive se pueden hacer prácticas donde en este se muestren valores reales que están siendo introducidos a la computadora.



Las principales herramientas que estaremos usando para crear nuestros circuitos son las siguientes:

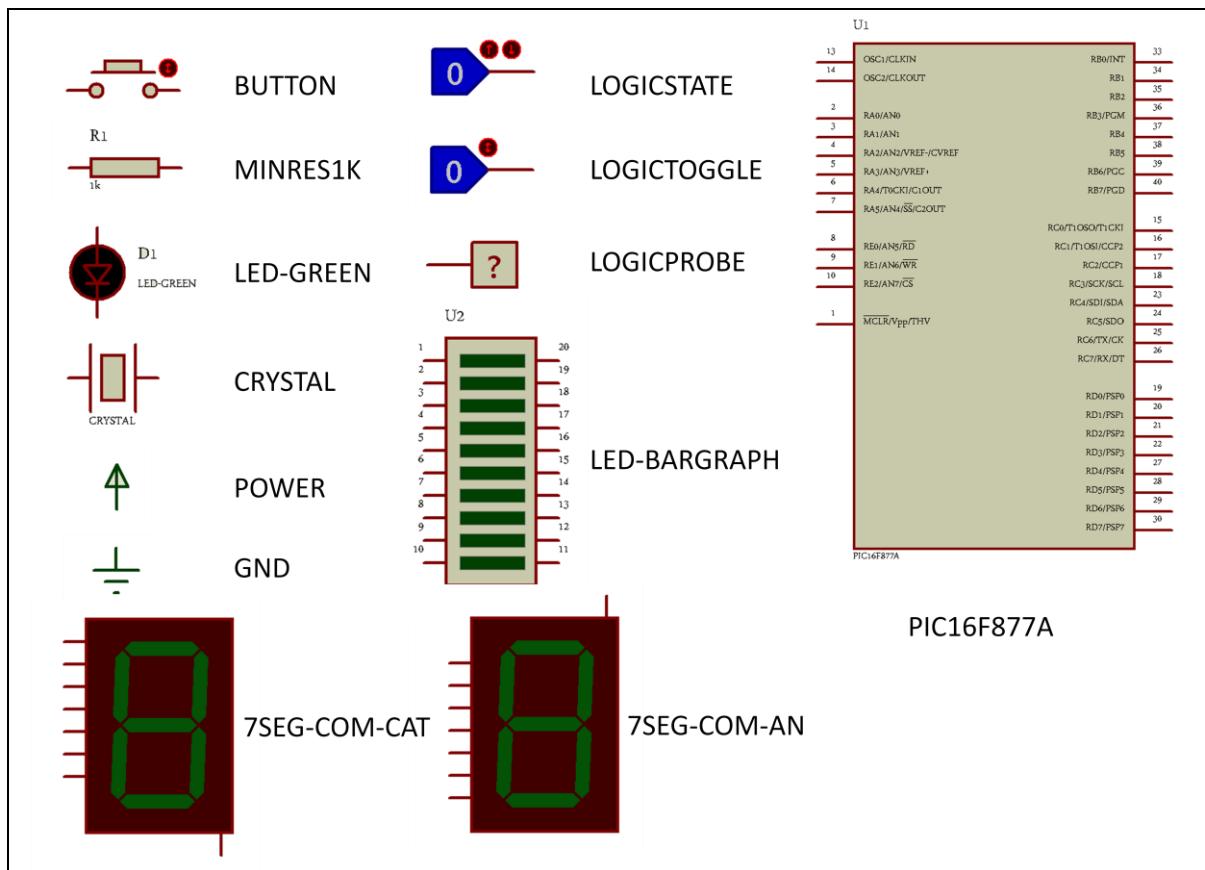


# Manual – Aplicaciones Prácticas con Microcontroladores PIC

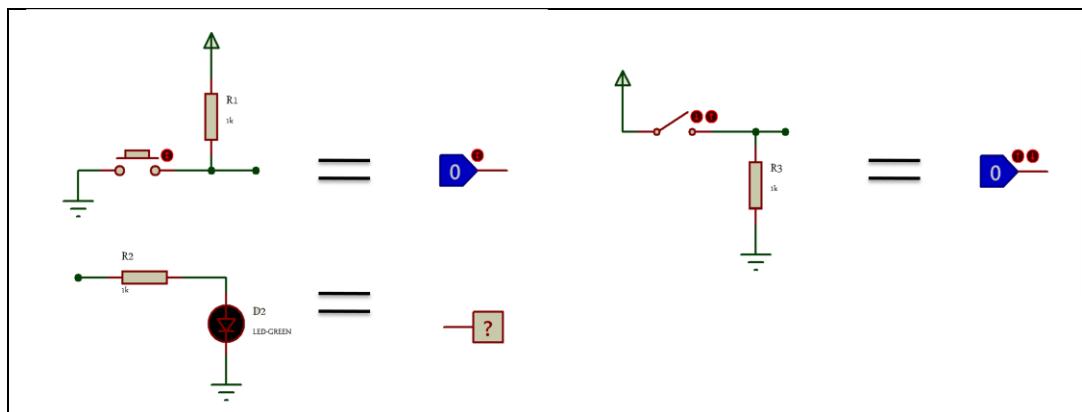
Elaborado por: M.I.E. Hugo Antonio Méndez Guzmán



Entre los principales componentes que usaremos comúnmente se enlistan los siguientes:



A excepción de POWER y GND, todos ellos se encuentran en la Herramienta de componentes, los otros se encuentran en Terminales comunes. Algunos de estos componentes son equivalentes entre ellos, por ejemplo cuando deseamos hacer pulsadores con los BUTTON o cuando armamos un visualizador por medio de un led. Estas equivalencias se muestran a continuación.

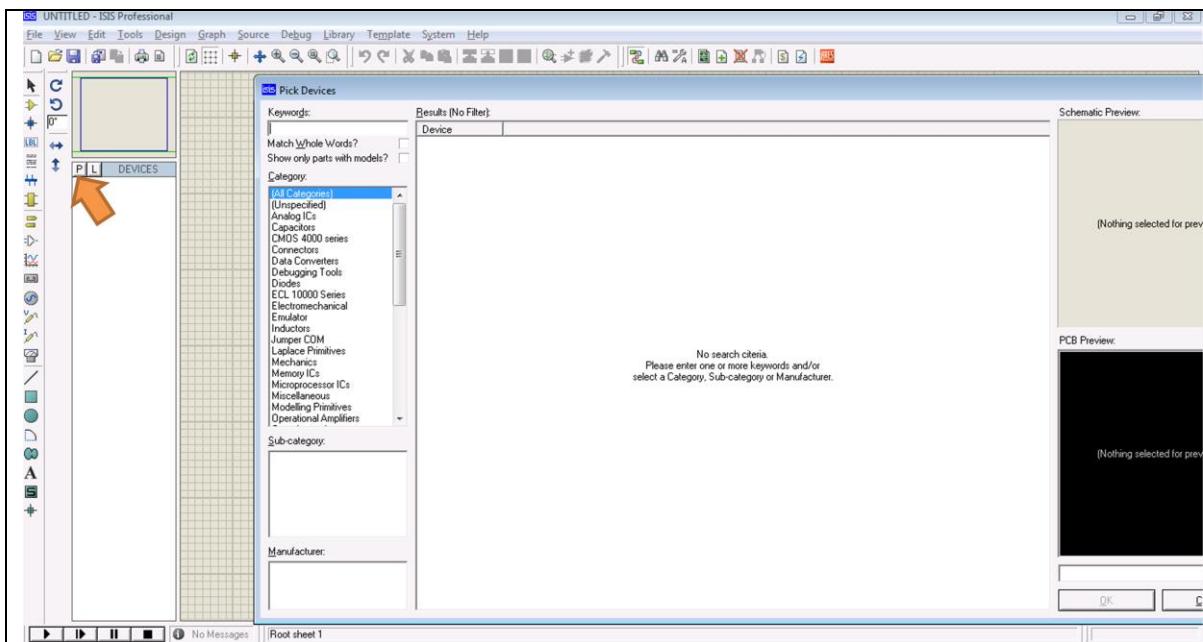


# Manual – Aplicaciones Prácticas con Microcontroladores PIC

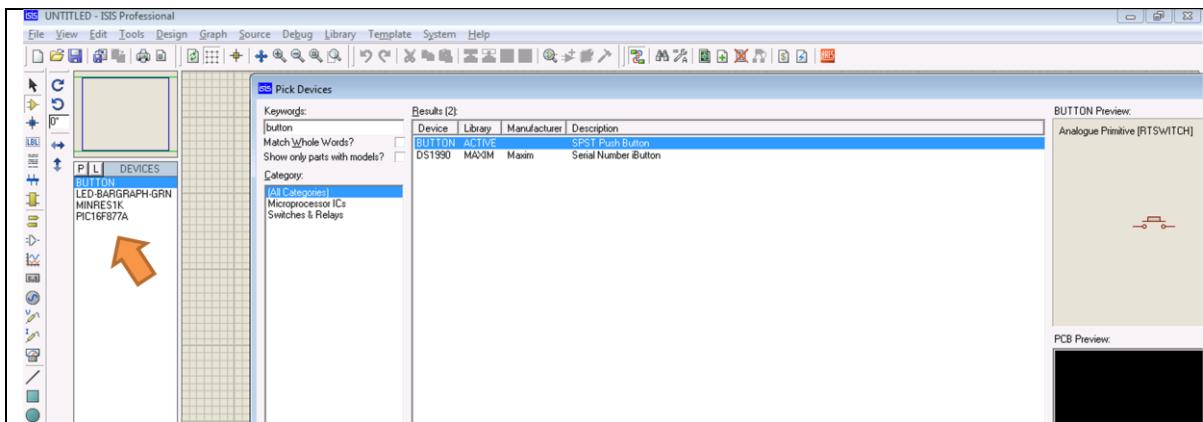
Elaborado por: M.I.E. Hugo Antonio Méndez Guzmán



Para poder usar un componente en PROTEUS ISIS es necesario primero generar nuestra lista de componentes, esto se hace mediante el botón Pick en la Herramienta de componentes



donde iremos buscando por nombre o por categoría los componentes que deseamos, una vez ubicados, le daremos dos click a cada uno de ellos para importarlos a nuestra lista de componentes.



Una vez colocados en nuestra lista daremos click al botón OK y ya podremos usarlos para generar nuestro diagrama esquemático. Y simplemente tendremos que dar click sobre ellos y luego ubicarnos en alguna parte de nuestra zona de diseño y dar nuevamente click para colocarlos.

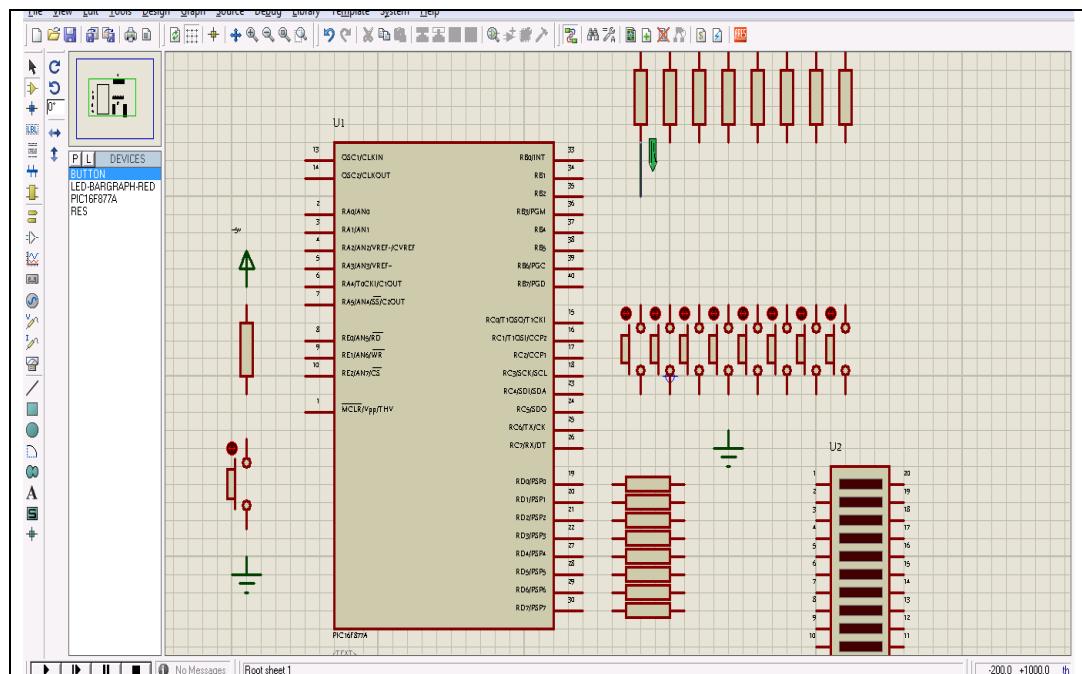
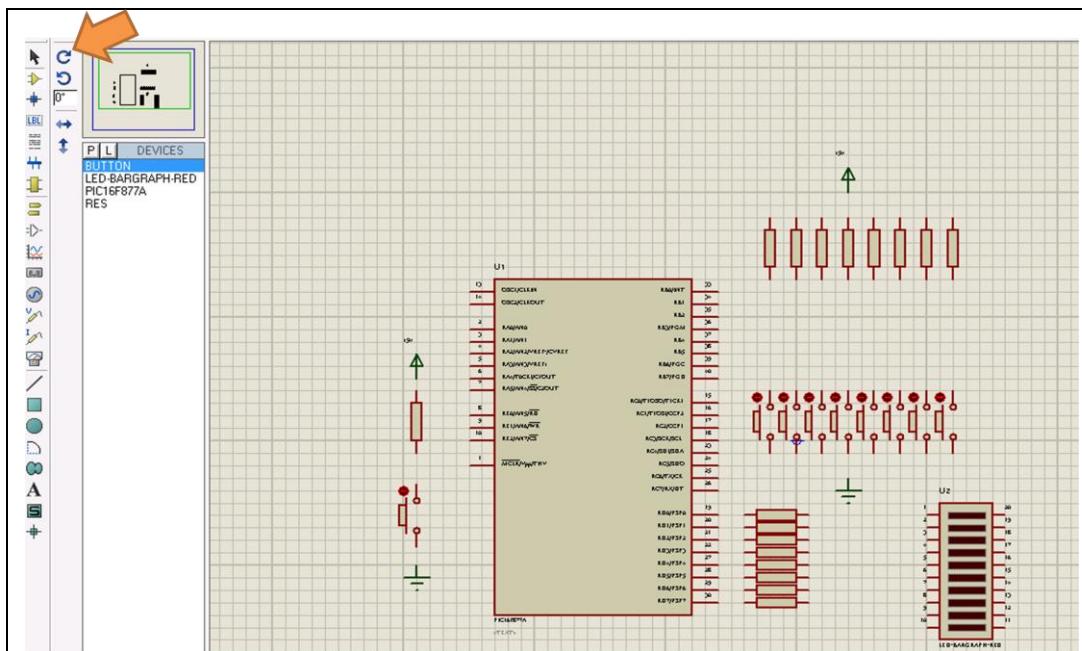


# Manual – Aplicaciones Prácticas con Microcontroladores PIC

Elaborado por: M.I.E. Hugo Antonio Méndez Guzmán

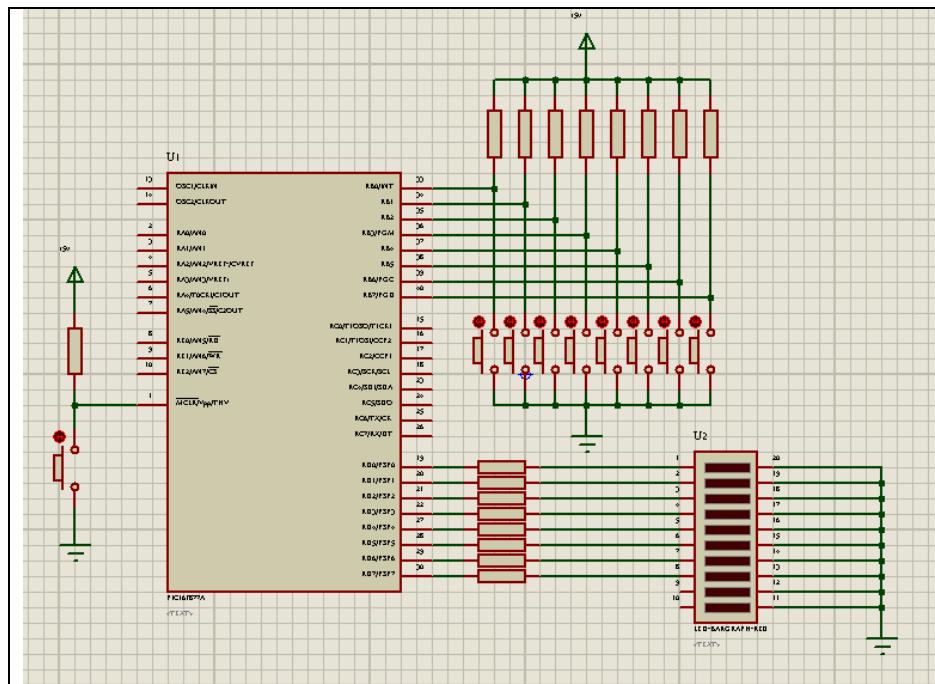


Apoyados en la herramienta de rotación y una vez colocados nuestros componentes, podemos realizar su interconexión, tan solo con poner el cursor sobre un pin, darle un click y luego poner el cursor en algún otro pin y darle click.

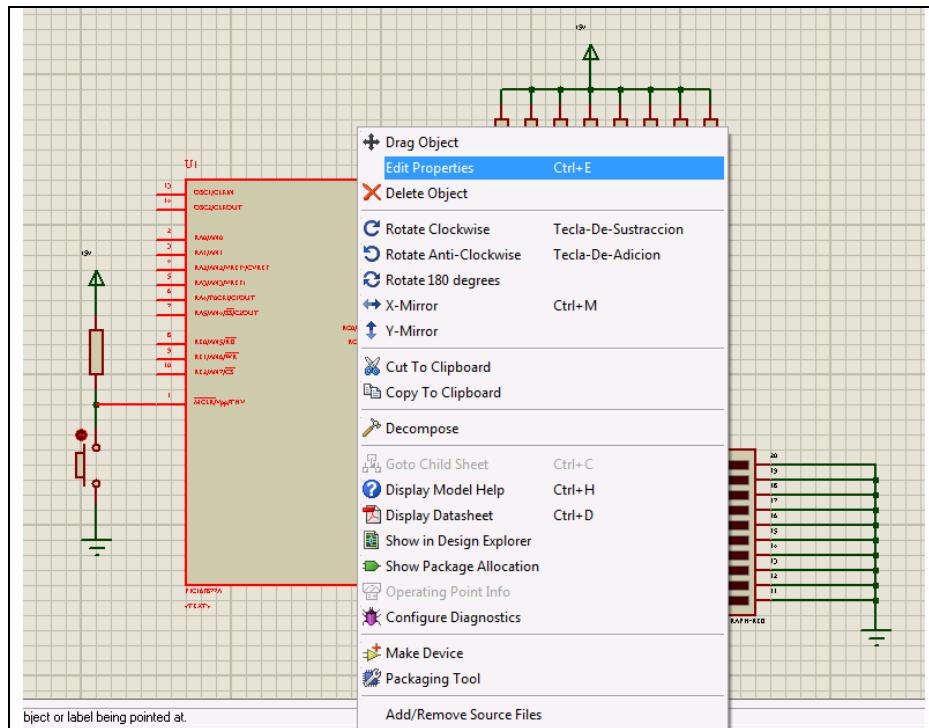


# Manual – Aplicaciones Prácticas con Microcontroladores PIC

Elaborado por: M.I.E. Hugo Antonio Méndez Guzmán



Una vez interconectados todos los elementos, tendremos que configurar nuestro microcontrolador para que reciba el programa que hemos compilado en Pic C Compiler, esto se logra dando click derecho en el PIC y luego en Edit Properties.

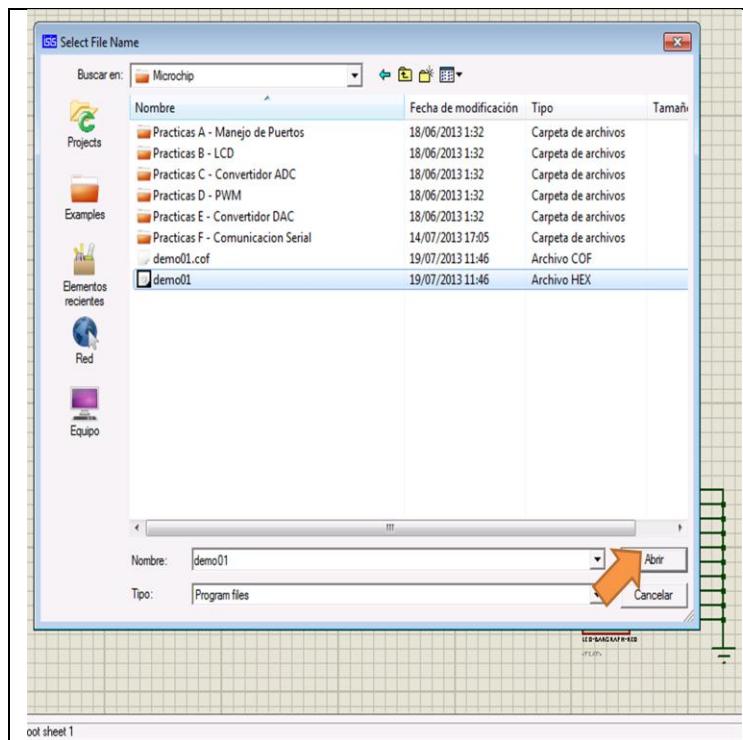
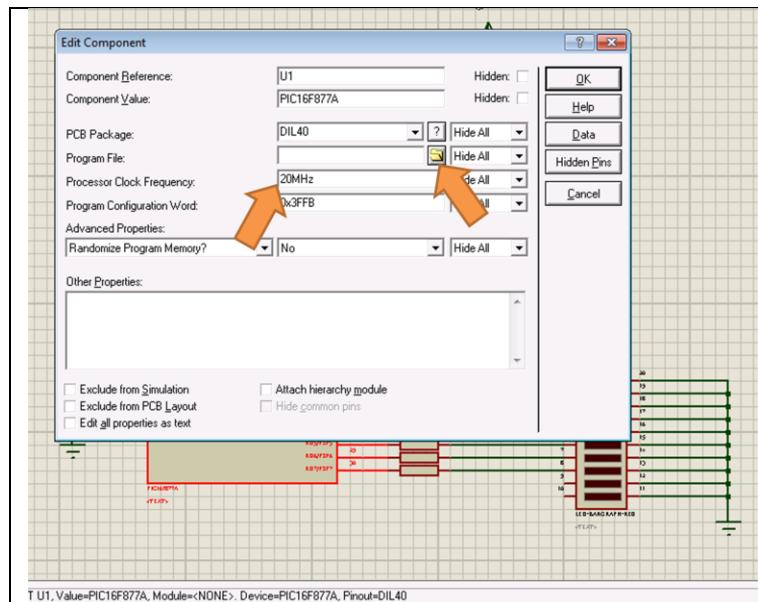


# Manual – Aplicaciones Prácticas con Microcontroladores PIC

Elaborado por: M.I.E. Hugo Antonio Méndez Guzmán



Enseguida nos aparecerá una ventana donde las opciones que tenemos que configurar son **Program File**, en la cual le asignaremos el archivo .hex que generamos en la compilación y **Processor Clock Frequency** donde tendremos que especificar al igual que en el programa en C la velocidad del oscilador con la cual estamos trabajando.

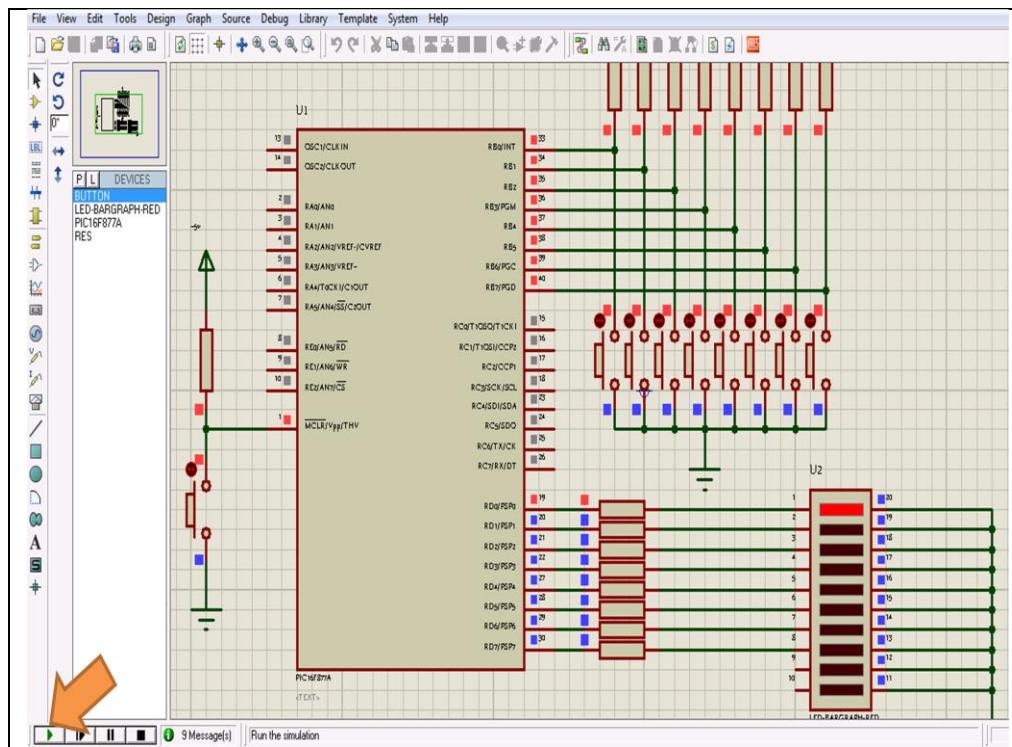
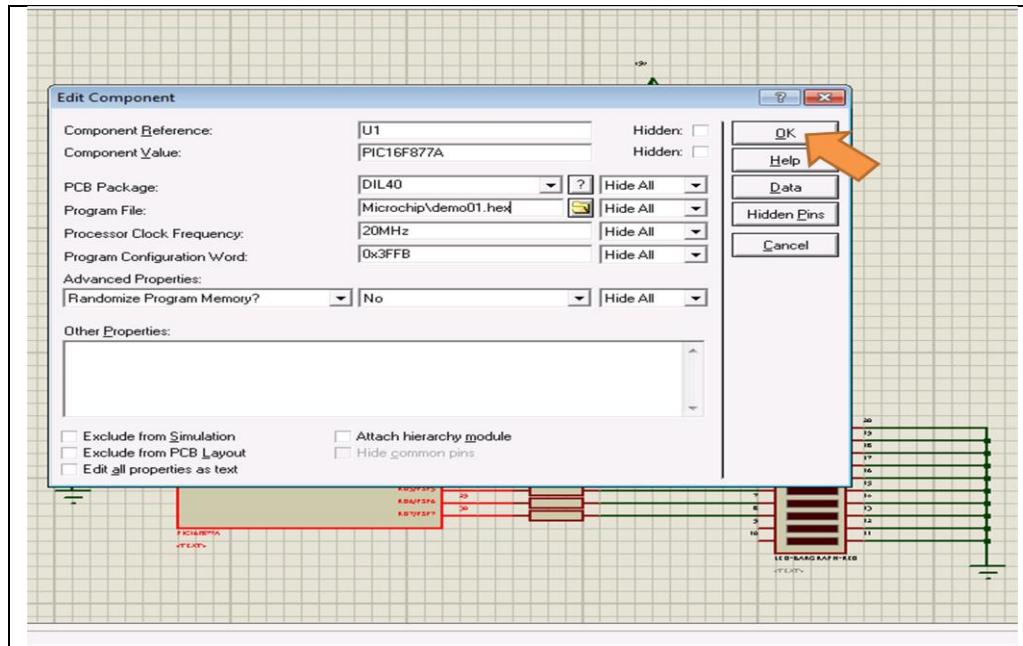


# Manual – Aplicaciones Prácticas con Microcontroladores PIC

Elaborado por: M.I.E. Hugo Antonio Méndez Guzmán



Una vez configuradas estas opciones daremos click en OK y nuestro circuito estará listo para simularse con el programa en C que hayamos generado.



Nota: El cristal que se requiere para la señal de reloj no es necesario incluirlo en el esquemático, bastará con las configuraciones anteriores (físicamente sí es necesario).

