

# Microcontroladores

## Guia PicC

HUGO ALBERTO MURILLO

# INTRODUCCIÓN

La guía de microcontroladores fue hecha pensando básicamente en un material escrito para los estudiantes, el cual pueda servirles como medio de consulta en las diferentes prácticas y proyectos que enfrentaran durante el curso de microcontroladores.

Inicialmente explica los conceptos básicos, como sistemas de numeración, continúa con una breve explicación sobre el lenguaje C, más adelante explica las características básicas del microcontrolador recomendado en el curso (PIC16F883 o PIC16F886) y por último plantea diferentes ejemplos con el microcontrolador, cada uno refiriéndose a un tema específico.

Los ejemplos que se desarrollan en esta guía son relativamente sencillos, y con seguridad no siempre será la mejor manera de desarrollarlos, solo es una de muchas formas de hacerlo.

# 1

## CONCEPTOS BÁSICOS

Antes de comenzar el estudio de los Microcontroladores se estudiarán algunos conceptos importantes para comprender bien el funcionamiento de los mismos.

### Sistemas de numeración

BINARIO	$2^7 \ 2^6 \ 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0$
DECIMAL	$\dots 10^3 \ 10^2 \ 10^1 \ 10^0$
HEXADECIMAL	$\dots 16^3 \ 16^2 \ 16^1 \ 16^0$

#### 1.1 SISTEMAS DE NUMERACIÓN DECIMAL (BASE 10)

El sistema decimal es un sistema de numeración en base 10 porque los símbolos que existen para representar cualquier número son 10, de 0 a 9. Pero más allá de representar cualquier número es importante conocer el peso de cada dígito.

Cuando se escribe un número decimal, cada dígito tiene un peso, por ejemplo:

1 4 1 2

miles  
decenas  
centenas  
unidades

Se puede decir que el número es igual a:

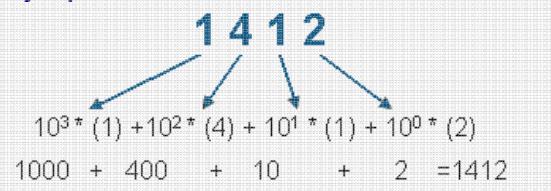
$$1000 * (1) + 100 * (4) + 10 * (1) + 1 * (2)$$

$$1000 + 400 + 10 + 2 = 1412$$

Como se había dicho antes cada bit tiene un peso y el sistema decimal, se puede representar:

$$\dots 10^4 \ 10^3 \ 10^2 \ 10^1 \ 10^0$$

##### Ejemplo 1.1.1 Numeración decimal



El valor de un número decimal es la suma de los dígitos después de haber multiplicado cada dígito por su peso.

#### 1.2 SISTEMA DE NUMERACIÓN BINARIO (BASE 2)

En electrónica digital es uno de los sistemas de numeración más utilizados. Es útil porque solo utiliza dos dígitos, 1 y 0. Los dígitos binarios se utilizan para representar dos niveles de voltaje ALTO O BAJO. En la mayoría de los sistemas digitales el nivel de voltaje alto se simboliza con el 1, mientras que el nivel de voltaje bajo o cero voltios lo simboliza el 0. El 1 representa el estado de encendido de un interruptor, de una luz o de un transistor, mientras el estado apagado está representado por un 0.

Solo se tienen dos dígitos para representar cualquier número en binario, todos los números binarios solo tienen unos y ceros y su base es dos y al igual que en el sistema decimal cada dígito tiene un peso.

$$\dots 2^3 \ 2^2 \ 2^1 \ 2^0$$

La palabra bit es una contracción de las palabras en Inglés binary digit (Dígito Binario). Cada posición de un número binario se conoce como bit. El número 10110 es un número de cinco bits. El primer lugar del extremo derecho recibe el nombre de bit menos significativo (o LSB por sus siglas en inglés), mientras que el lugar que está en el extremo izquierdo se conoce como

bit más significativo (MSB por sus siglas en inglés).

**1 0 1 1 0**  
MSB            LSB

1 palabra = 16 bits  
1 byte = 8 bits  
1 nible = 4 bits

1 byte = **1 0 1 0**    **1 1 0 0**  
                { Nibble Superior    { Nibble inferior

### 1.3 SISTEMA HEXADECIMAL (BASE 16)

Este sistema es en base 16, lo que significa que para cada columna es posible escoger uno entre 16 dígitos.

Estos son: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E y F.

Donde            A = 10            D = 13  
                  B = 11            E = 14  
                  C = 12            F = 15

Para contar en el sistema hexadecimal se inicia en la primera columna a la izquierda y se cuenta de 0 hasta F, una vez que se llena la primera columna, se pone un 0 en ella y se suma a la segunda columna.

0F
10
.
.
.
1F
20
21
.
.
.....

$$16^3 \ 16^2 \ 16^1 \ 16^0$$

Al igual que el sistema decimal y binario cada dígito tiene un peso.

Se suele poner una H al final del número para indicar que está representado en el sistema hexadecimal 17H

### 1.4 CONVERSIÓN DE BINARIO A DECIMAL

Para convertir un número binario en uno decimal, se hace la lista con los valores de cada posición y luego se suman los que corresponden a las posiciones donde hay un

#### Ejemplo 1.4.1 Conversión binario a decimal

$$\begin{array}{ccccccc} 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & & \\ 16 & + & 0 & + & 4 & + & 0 & + & 1 = 21 \end{array}$$

### 1.5 CONVERSIÓN DE DECIMAL A BINARIO

El número decimal se divide repetidamente entre 2, ignorando los residuos, hasta que se tiene un cociente igual a cero. Despues se emplean éstas para obtener la respuesta, por

#### Ejemplo 1.5.1 Conversión decimal a binario

Convertir  $101_{10}$  en su equivalente a binario.

$101 / 2 = 50$	Residuo	1	LSB
$50 / 2 = 25$	Residuo	0	
$25 / 2 = 12$	Residuo	1	
$12 / 2 = 6$	Residuo	0	
$6 / 2 = 3$	Residuo	0	
$3 / 2 = 1$	Residuo	1	
$1 / 2 = 0$	Residuo	1	MSB

RESPUESTA  $1\ 1\ 0\ 0\ 1\ 0\ 1_2$

### 1.6 CONVERSIÓN DE HEXADECIMAL A DECIMAL

Para convertir un número hexadecimal a decimal, se multiplica el valor de cada dígito por su correspondiente peso y luego se suman todos los valores.

#### Ejemplo 1.6.1 Conversión hexadecimal a decimal

2B616 → decimal?

$$\begin{array}{ccc} 16^2 & 16^1 & 16^0 \\ 2 & B & 6 \end{array}$$

$$2 * 256 + 11 * 16 + 6 * 1 = 694$$

## 1.7 CONVERSIÓN DE BINARIO A HEXADECIMAL

Cuatro bits binarios corresponden a un dígito hexadecimal, significa que se requieren cuatro bits para contar de 0 hasta F. Para representar números binarios como números hexadecimales, se forman grupos de cuatro bits, de izquierda a derecha. A continuación se convierte cada grupo en el correspondiente dígito hexadecimal.

### Ejemplo 1.7.1 Conversión binario a hexadecimal

Convertir 10111100 en un número hexadecimal

1011 1100  
B C

$$10111100 = BC_{16}$$

## 1.8 CONVERSIÓN DE HEXADECIMAL A BINARIO

La conversión de hexadecimal a Binario es igual de sencilla, por cada dígito hexadecimal se escriben los dígitos binarios correspondientes.

FF<sub>16</sub> Binario

F F  
1111 1111

$$FF_{16} = 11111111$$

## 1.9 DECIMAL CODIFICADO EN BINARIO: (BCD)

En BCD cada dígito decimal esta representado por cuatro bits.

0.....0000  
1.....0001  
.  
. .  
9.....1001

Para representar el 25 decimal en BCD

0010 1010  
2 5

## 1.10 REPRESENTACIÓN DE LOS NÚMEROS DE 0 A 15

Tabla 1.10.1 Representación de los números de 0 a 15.

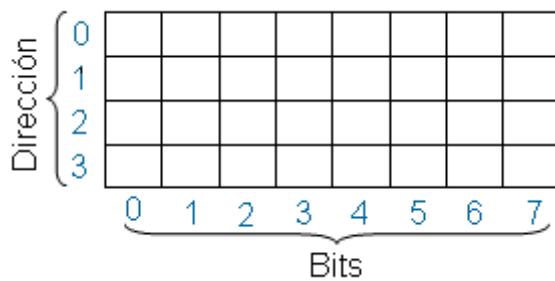
DECIMAL	BINARIO	HEXADECIMAL	BCD
0	0000	0	0000
1	0001	1	0001
2	0010	2	0010
3	0011	3	0011
4	0100	4	0100
5	0101	5	0101
6	0110	6	0110
7	0111	7	0111
8	1000	8	1000
9	1001	9	1001
10	1010	A	0001 0000
11	1011	B	0001 0001
12	1100	C	0001 0010
13	1101	D	0001 0011
14	1110	E	0001 0100
15	1111	F	0001 0101

# 2

## MEMORIAS

- Cada elemento de almacenamiento en una memoria puede almacenar un “1” ó un “0” y se le denomina celda. Las memorias están formadas por matrices de celdas. Cada bloque de la matriz de memoria representa una celda de almacenamiento y su situación se puede especificar mediante una fila y una columna.

La matriz de  $4 \times 8$ , se puede entender como una memoria de 32 bits o como una memoria de 4 bytes.



capacidad total es de 32 bits o 4 bytes.

Puesto que una memoria almacena datos binarios, los datos pueden introducirse en la memoria y deben poderse recuperar cuando se necesiten. La operación de escritura coloca los datos en una posición específica de la memoria y la operación de lectura extrae los datos de una dirección específica de memoria. La operación de direccionamiento, que forma parte tanto de la operación de lectura como de escritura, selecciona la dirección de manera específica.

La posición de una unidad de datos en una matriz de memoria se denomina dirección. La dirección de un bit en la matriz se especifica mediante la fila y la columna en la cual se encuentra. La dirección de un byte se especifica únicamente mediante la fila.

La capacidad de una memoria es el número total de unidades de datos que puede almacenar. En el ejemplo tratado la

### 2.1 MEMORIA RAM

La memoria RAM (Random-Access Memory; Memoria de Acceso aleatorio), es un tipo de memoria en la que se tarda lo mismo en acceder a cualquier dirección de memoria y éstas se pueden seleccionar en cualquier orden, tanto en una operación de lectura como de escritura. Todas las RAMs poseen la capacidad de lectura y escritura.

Las memorias RAM reciben el nombre de memoria volátil, ya que pierden los datos almacenados cuando se desconecta la alimentación.

### 2.2 MEMORIA ROM

La memoria ROM (Read\_Only Memory; Memoria de solo lectura), es un tipo de memoria en la cual los datos se almacenan en forma permanente o semipermanente. Los datos se pueden leer de una ROM, pero no existe la operación de escritura como en las RAM.

### 2.3 MEMORIA EPROM

Son memorias que se programan eléctricamente y que se borran con la exposición de la memoria a una luz ultravioleta que pasa a través de la ventana de cuarzo que tiene en la parte superior del encapsulado.

### 2.4 MEMORIA EEPROM

Son memorias que se pueden borrar y programar mediante impulsos eléctricos. Se pueden grabar y borrar eléctricamente, las EEPROM se pueden reprogramar dentro del propio circuito final.

### 2.5 MEMORIA FLASH

Las memorias flash son memorias de lectura / escritura de alta densidad (alta densidad se refiere a gran cantidad de almacenamiento de bits) no volátiles, esto significa que los datos se pueden almacenar indefinidamente sin necesidad de alimentación.

# 3

## INTRODUCCIÓN AL MICROCONTROLADOR

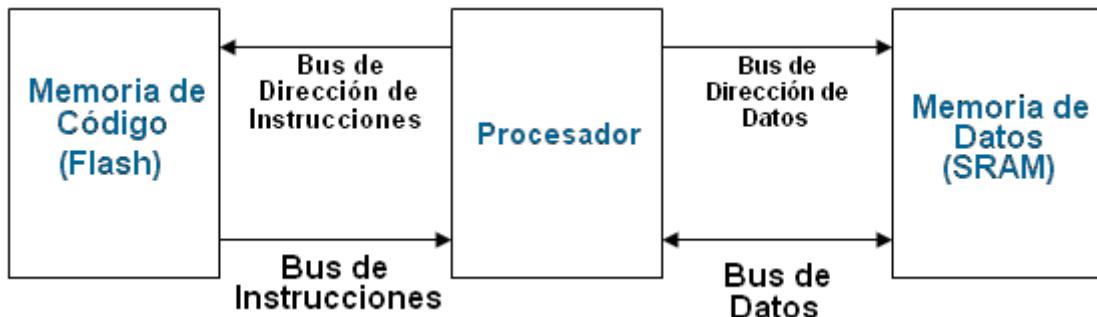
Un microcontrolador es un circuito integrado que contiene toda la estructura de un Microcomputador, es decir, unidad de proceso (CPU), memoria RAM, memoria ROM y circuitos de entrada/salida.

Es un dispositivo programable que puede ejecutar un sinnúmero de tareas y procesos. Un Microcontrolador está compuesto básicamente por cuatro componentes principales:

Memoria ROM, EEPROM, EPROM o FLASH: es la memoria donde se almacena el programa. Memoria RAM o SRAM: es la memoria donde se almacenan los datos. Líneas de Entrada / Salida (I / O): también llamada puertos, se utilizan para conectar elementos externos al microcontrolador. CPU: controla y ejecuta todas las instrucciones que conforman el programa.

Existen diferentes familias de microcontroladores: Intel, Motorola, Microchip, entre otras.

En este curso solo se estudiará el microcontrolador PIC16F873, que pertenece a la familia de Microchip; esta familia se caracteriza por tener procesador RISC y arquitectura Harvard caracterizada por la independencia entre la memoria de código (programa) y la de memoria de datos.



El conjunto de instrucciones es de solo 35, por esto se dice es un microcontrolador de tipo RISC (computador con set de instrucciones reducido). Aunque para el curso esta característica será transparente al programar el microcontrolador en C.

### 3.1 ORGANIZACIÓN DE LA MEMORIA DE DATOS RAM

Mapa de Memoria del Plc16F886 (Tabla 3.1.1 Mapa de memoria del Plc16F886)

### 3.2 DESCRIPCION DE LOS PINES

OSC1 / CLK IN (9): Entrada del cristal de cuarzo o del oscilador externo.

OSC2 / CLK OUT (10): Salida del cristal de cuarzo.

VSS (8 - 19): Conexión a tierra (GND)

VDD (20): Conexión a positivo (+5V)

MCLR# / VPP (1): Entrada de Reset o entrada del voltaje de programación.

Si no se va a utilizar se debe poner a +5V.

**Puerto A:** El puerto A del microcontrolador está compuesto por 6 líneas de entrada / salida que además nos permiten trabajar con señales Análogas.

RA0 / AN0 (2): puede funcionar como línea digital o analoga.

RA1 / AN1 (3): igual que la RA0 / AN0.

RA2 / AN2 (4): línea de entrada / salida digital.

RA3 / AN3 (5): Línea de entrada / salida digital.

RA4 (6): Línea de entrada / salida digital.

Nota: Cuando este pin se configura como salida funciona como salida de colector abierto, es decir, se debe conectar una resistencia a +V.

RA5 / AN4 (7): Línea de entrada / salida digital.

**Puerto B:** Este puerto esta compuesto por 8 líneas que se pueden configurar como entrada / salida digital y para interrupciones externas.

RB0 (21): Línea de entrada / salida digital.

RB1 (22): Línea de entrada / salida digital.

RB2 (23): Línea de entrada / salida digital.

RB3 (24): Línea de entrada / salida digital.

RB4 (25): Línea de entrada / salida digital.

RB5 (26): Línea de entrada / salida digital.

RB6 (27): Línea de entrada / salida digital.

RB7 (28): Línea de entrada / salida digital.

**Puerto C:** Este puerto esta compuesto por 8 líneas que se pueden configurar como entrada / salida digitales, además sirve para trabajar con los temporizadores del microcontrolador y la comunicación serial.

RC0 (11): Línea de entrada / salida digital.

RC0 (12): Línea de entrada / salida digital.

RC0 (13): Línea de entrada / salida digital.

RC0 (14): Línea de entrada / salida digital.

RC0 (15): Línea de entrada / salida digital.

RC0 (16): Línea de entrada / salida digital.

RC0 (17): Línea de entrada / salida digital.

RC0 (18): Línea de entrada / salida digital.

Cada patica de los diferentes puertos, se identifica según el puerto y el bit, como ejemplo si nos referimos a RB0, este corresponde al bit 0 del puerto B y se identifica como PORTB,0. De igual forma lo hacemos con los diferentes bits de cada puerto.

NOTA: El cristal determina la velocidad de ejecución del programa. Para ejecutar un programa se necesita garantizar las siguientes conexiones.

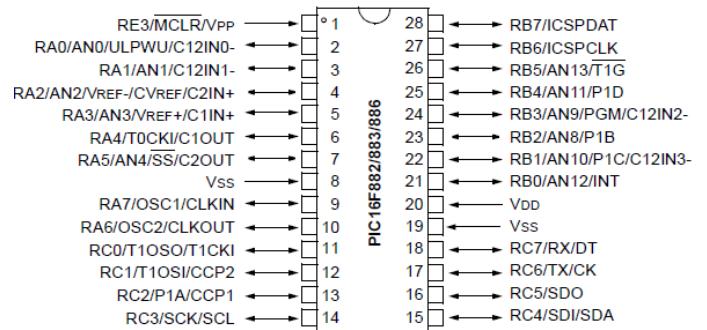


Figura 3.2.1 Descripción de los pines

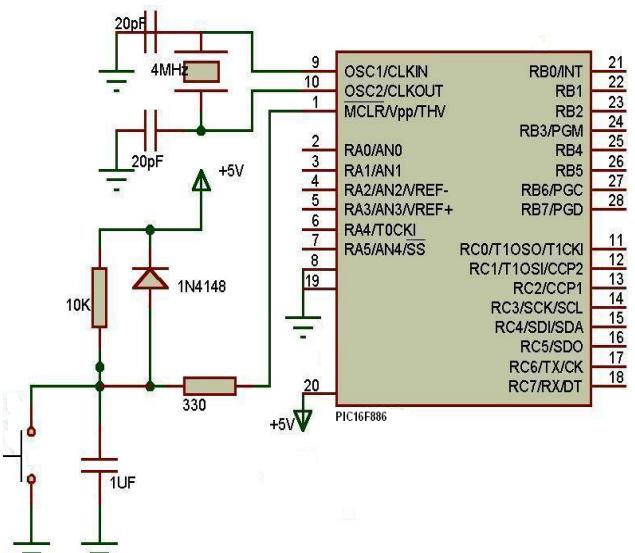


Figura 3.2.2 Conexiones

File Address	File Address	File Address	File Address	File Address
Indirect addr. (1) 00h	Indirect addr. (1) 00h	Indirect addr. (1) 80h	Indirect addr. (1) 100h	Indirect addr. (1) 180h
TMR0 01h	OPTION_REG 01h	TMR0 81h	OPTION_REG 101h	OPTION_REG 181h
PCL 02h	PCL 02h	PCL 82h	PCL 102h	PCL 182h
STATUS 03h	STATUS 03h	STATUS 83h	STATUS 103h	STATUS 183h
FSR 04h	FSR 04h	FSR 84h	FSR 104h	FSR 184h
PORTA 05h	TRISA 05h	WDTCON 85h	SRCON 105h	SRCON 185h
PORTB 06h	TRISB 06h	PORTB 86h	TRISB 106h	TRISB 186h
PORTC 07h	TRISC 07h	CM1CON0 87h	BAUDCTL 107h	BAUDCTL 187h
		CM2CON0 88h	ANSEL 108h	ANSEL 188h
PORTE 08h	TRISE 09h	CM2CON1 89h	ANSELH 109h	ANSELH 189h
PCLATH 0Ah	PCLATH 0Ah	PCLATH 8Ah	PCLATH 10Ah	PCLATH 18Ah
INTCON 0Bh	INTCON 0Bh	INTCON 8Bh	INTCON 10Bh	INTCON 18Bh
PIR1 0Ch	PIE1 0Ch	EEDAT 8Ch	EECON1 10Ch	EECON1 18Ch
PIR2 0Dh	PIE2 0Dh	EEADR 8Dh	EECON2 (1) 10Dh	EECON2 (1) 18Dh
TMR1L 0Eh	PCON 0Fh	EEDATH 8Eh	Reserved 10Eh	Reserved 18Eh
TMR1H 0Fh	OSCCON 10h	EEADDRH 8Fh	Reserved 10Fh	Reserved 18Fh
T1CON 10h	OSCTUNE 10h			190h
TMR2 11h	SSPCON2 11h			191h
T2CON 12h	PR2 12h			192h
SSPBUF 13h	SSPADD 13h			193h
SSPCON 14h	SSPSTAT 14h			194h
CCP1RL 15h	WPUB 15h			195h
CCP1RH 16h	IOCB 16h			196h
CCP1ICON 17h	VRCON 17h			197h
RCSTA 18h	TXSTA 18h			198h
TXREG 19h	SPBRG 19h			199h
RCREG 1Ah	SPBRGH 1Ah			19Ah
CCP2RL 1Bh	PWMICON 1Bh			19Bh
CCP2RH 1Ch	ECCPAS 1Ch			19Ch
CCP2CON 1Dh	PSTRCON 1Dh			19Dh
ADRESH 1Eh	ADRESL 1Eh			19Eh
ADCON0 1Fh	ADCON1 1Fh			19Fh
	General Purpose Registers 32 Bytes			1A0h
General Purpose Registers 96 Bytes	32 Bytes			
		EFh	16Fh	1EFh
		C0h	170h	1F0h
		FFh	17Fh	1FFh
Bank 0	Bank 1	Bank 2	Bank 3	

**Tabla 3.1.1 Mapa de memoria del PIC16F886**

### 3.3 CONFIGURACIÓN DE LOS PUERTOS

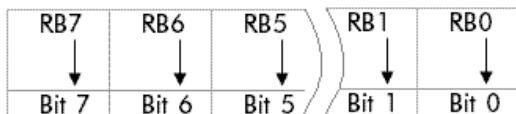
Cada pin de los puertos del microcontrolador se pueden configurar como entrada o salida digital.

Las entradas corresponden a sensores, suiches o pulsadores, es decir son los ojos del microcontrolador, el microcontrolador se da cuenta de lo que ocurre a través de las entradas.

Las salidas corresponden a los elementos que el microcontrolador va a controlar, un

bombillo, un led, un motor, una electro válvula, entre otros, es decir las salidas corresponden al elemento final de control.

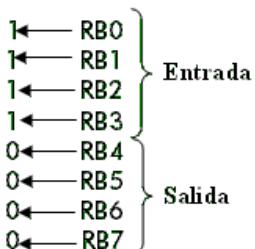
Cada línea de cada puerto representa un bit, por ejemplo el puerto B:



Para denotar un bit en particular, se puede decir:

`PORTB, 6` → el bit 6 del puerto B

Si una patica va a funcionar como entrada se coloca en "1" y si va a funcionar como salida se coloca en "0".



### Ejemplo 3.3.1 Configuración puerto B

Si se necesita configurar el puerto B de la siguiente manera:

En el encabezado del programa debe escribirse la linea:

`# BYTE PORTB=6.`

El número 6 corresponde a la dirección de memoria Ram en la que esta ubicado el puerto B. (ver tabla de memoria en pagina 16)

En el programa principal se digita la linea:

`SET_TRIS_B(0B00001111);`

Los últimos cuatro bits corresponden a salidas, por lo tanto se ponen en cero y los otros cuatro corresponden a entradas por consiguiente se ponen en uno.

La instrucción SET\_TRIS permite configurar el puerto y se añade la última letra de acuerdo al puerto que sé este configurando.

Si se quiere configurar el puerto C como salida se deben seguir los dos pasos anteriores, de la siguiente forma:

`# BYTE PORTC=7.` //Dirección de memoria Ram del puerto C

En el programa principal se digita la linea:

`SET_TRIS_C(0B00000000);`

Para el puerto B es SET\_TRIS\_B, para el puerto C SET\_TRIS\_C y para el puerto A SET\_TRIS\_A.

# 4

## LENGUAJE DE PROGRAMACIÓN

C es un lenguaje de alto nivel, aunque permite trabajar en bajo nivel, es decir manipular bits, es quizás uno de los lenguajes más utilizados y existen diferentes versiones del lenguaje C. En el curso se concentra en C básico, se analiza los diferentes tipos de datos, las estructuras de control, los tipos de variables y las funciones. Todo programa en C está compuesto por un encabezado, unas funciones si se necesitan y el programa principal, llamado main. Los diferentes tipos de datos que maneja el compilador PICC, son los siguientes:

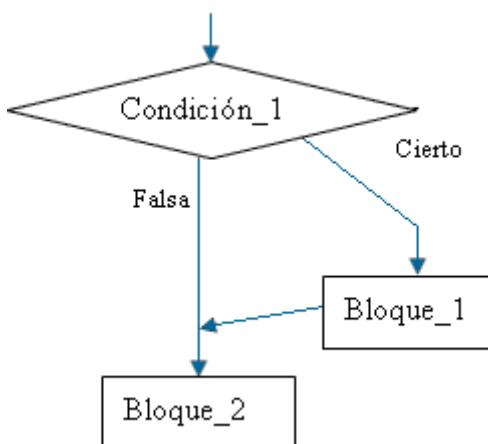
**unsigned define** un número de 8 bits sin signo  
**unsigned int define** un número de 8 bits sin signo  
**int define** un número de 8 bits sin signo  
**int16 define** un número de 16 bits sin signo  
**char define** un número de 8 bits sin signo  
**long define** un número de 16 bits sin signo  
**long int define** un número de 16 bits sin signo

**signed define** un número de 8 bits con signo  
**signed int define** un número de 8 bits con signo  
**signed long define** un número de 16 bits con signo  
**float define** un número de 32 bits en punto flotante  
**short define** un bit  
**short int define** un bit

### 4.1 ESTRUCTURAS DE CONTROL EN C

#### LA ESTRUCTURA DE CONTROL CONDICIONAL IF

La sentencia if nos permite elegir si se ejecuta o no un bloque de instrucciones



#### Sintaxis

if (condición)  
sentencia;

o un bloque de instrucciones:

```
if (condición)  
{  
bloque  
}
```

Donde bloque representa un bloque de instrucciones.

Consideraciones acerca del uso de la sentencia if

- Olvidar los paréntesis al poner la condición del if es un error sintáctico (los paréntesis son necesarios)

- Confundir el operador de comparación == con el operador de asignación = puede producir errores inesperados.

- Los operadores de comparación ==, !=, <= y >= deben escribirse sin espacios.

- => y =< no son operadores válidos en C.

- El fragmento de código afectado por la condición del if debe sangrarse para que visualmente se interprete correctamente el ámbito de la sentencia if:

```
if (condición)  
{  
// Aquí se incluye el código  
// que ha de ejecutarse sólo  
// si se cumple la condición del if  
// (sangrado para que se vea dónde  
// empieza y dónde acaba el if)  
}
```

Error común:

```
if (condición);
```

sentencia;

Lo anterior es interpretado como

```
if (condición)
; // Sentencia vacía
sentencia;
```

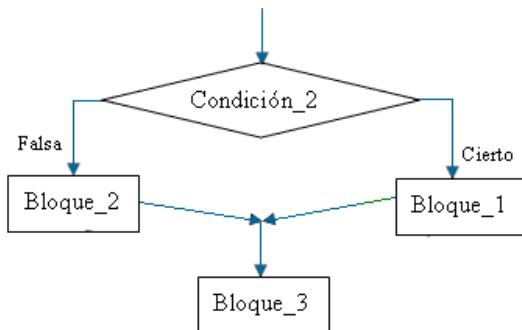
¡¡¡La sentencia siempre se ejecutaría!!!

Nota: Una costumbre buena es poner en el programa las llaves desde el momento en que se crea la estructura if.

```
if(Condición)
{
}
```

### CLÁUSULA ELSE

Una sentencia if, cuando incluye la cláusula else, permite ejecutar un bloque de código si se cumple la condición y otro bloque de código diferente si la condición no se cumple.



### Sintaxis

```
if (condición)
sentencia1;
else
sentencia2;
```

o un bloque de instrucciones:

```
if (condición)
{
bloque1
}
else
{
bloque2
}
```

Los bloques de código especificados representan dos alternativas complementarias y excluyentes.

### SELECCIÓN MÚLTIPLE CON LA SENTENCIA SWITCH

Permite seleccionar entre varias alternativas posibles

#### Sintaxis

```
switch (expresión) {
case expr_cte1:
sentencia1;
case expr_cte2:
sentencia2;
...
case expr_cteN:
sentenciaN;
default:
sentencia;
}
```

- Se selecciona a partir de la evaluación de una única expresión.
- La expresión del switch ha de ser de tipo entero.
- Los valores de cada caso del switch han de ser constantes.
- La etiqueta default marca el bloque de código que se ejecuta por defecto (cuando al evaluar la expresión se obtiene un valor no especificado por los casos del switch).
- En C, se ejecutan todas las sentencias incluidas a partir del caso correspondiente, salvo que explícitamente se use break:

```
switch (expresión)
```

```
{  
case 0:  
    sentencia1;  
    break ;  
  
case 1:  
    sentencia 2;  
    break ;  
  
case n:  
    sentencia n;  
    break ;  
  
default:  
sentencia;  
}
```

Si se trabaja con datos de tipo real, se tendrá que usar sentencias if encadenadas.  
Estructuras de control repetitivas/iterativas

## 4.2 ESTRUCTURAS DE CONTROL REPETITIVAS

Las estructuras de control repetitivas o iterativas, también conocidas como “bucles” se puede usar cuando se conoce el número de veces que deben repetirse las operaciones. Otras permiten repetir un conjunto de operaciones mientras se cumpla una condición.

**Iteración** Cada repetición de las instrucciones de un bucle

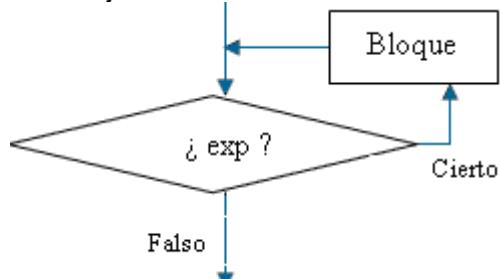
### BUCLE WHILE

Los ejecuta una instrucción o un bloque de instrucciones mientras la condición sea verdadera

Sintaxis:  
While (Condicion)  
Sentencia;

O un bloque de instrucciones:

```
While (Condición)  
{  
    Sentencia 1;  
    Sentencia 2;  
    .....  
    Sentencia n;  
}
```



Por lo general, dentro de la proposición ó del bloque de ellas, se modifican términos de la expresión condicional, para controlar la duración de la iteración.

### BUCLE FOR

Permite ejecutar una instrucción o un bloque de instrucciones una cantidad determinada de veces. Se suele emplear en sustitución del bucle while cuando se conoce el número de iteraciones que hay que realizar.

```
for (expr1; expr2; expr3) {  
    bloque;  
}
```

En un bucle for

La primera expresión, expr1, suele contener inicializaciones de variables separadas por comas. En especial, siempre aparecerá la inicialización de la variable que hace de contador.

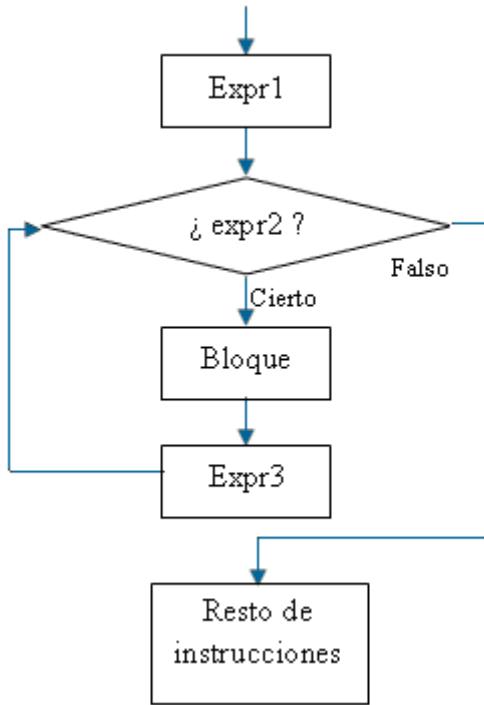
Las instrucciones que se encuentran en esta parte del for sólo se ejecutarán una vez antes de la primera ejecución del cuerpo del bucle (bloque).

La segunda expresión, expr2, contiene una expresión booleana (la cual aparecería en la condición del bucle while equivalente para controlar la ejecución del cuerpo del bucle).

La tercera expresión, expr3, contiene las instrucciones, separadas por comas, que se deben ejecutar al finalizar cada iteración del

bucle (p.ej. el incremento/decremento de la variable contador).

El bloque de instrucciones es el ámbito del bucle (el bloque de instrucciones que se ejecuta en cada iteración).



## EQUIVALENCIA ENTRE FOR Y WHILE

```
for(expr1; expr2; expr3)
{
    bloque;
}
```

Equivale a:

```
expr1;
while(expr2)
{
    bloque;
    expr3;
}
```

## BUCLLES INFINITOS

AUn bucle infinito es un bucle que se repite “infinitas” veces:

```
for (;;) /*bucle infinito*/
```

```
while (1) ó while (true) /*bucle infinito*/
```

Si nunca deja de cumplirse la condición del bucle, nuestro programa se quedará indefinidamente ejecutando el cuerpo del bucle, sin llegar a salir de él.

En las estructuras de control If, Else, while, la condición puede estar compuesta por una o más condiciones, finalmente el programa evalúa si la condición es cierta o falsa.

Para lograr este objetivo existen operadores de asignación y operadores de comparación, que se presentan en las siguientes del siguiente capítulo.

# 5

## OPERADORES

Las variables, como base de información de un lenguaje, pueden ser creadas, modificadas y comparadas con otras por medio de los llamados operadores. En el presente capítulo se dará constancia de ellos.

### 5.1 OPERADORES ARITMÉTICOS

Tal como era de esperarse los operadores aritméticos, mostrados en la TABLA 5.1.1, comprenden las cuatro operaciones básicas, suma, resta, multiplicación y división, con un agregado, el operador módulo.

Tabla 5.1.1 Operadores Aritméticos

SÍMBOLO	DESCRIPCIÓN	EJEMPLO
+	SUMA	$a + b$
-	RESTA	$a - b$
*	MULTIPLICACIÓN	$a * b$
/	DIVISION	$a / b$
%	MODULO	$a \% b$
	SIGNO	$-a$

El operador módulo ( %) se utiliza para calcular el resto del cociente entre dos ENTEROS.

### 5.2 OPERADORES RELACIONES

Todas las operaciones relacionales dan sólo dos posibles resultados: VERDADERO ó FALSO. En el lenguaje C, Falso queda representado por un valor entero nulo (cero) y Verdadero por cualquier número distinto de cero. En la TABLA 5.2.1 se encuentra la descripción de los mismos.

Tabla 5.2.1 Operadores Relacionales

SÍMBOLO	DESCRIPCIÓN	EJEMPLO
<	menor que	$(a < b)$
>	mayor que	$(a > b)$
<=	menor o igual que	$(a <= b)$
>=	mayor o igual que	$(a >= b)$
==	igual que	$(a == b)$
!=	distinto que	$(a != b)$

### 5.3 OPERADORES LÓGICOS

Hay tres operadores que realizan las conectividades lógicas Y (AND), O (OR) y NEGACIÓN (NOT) y están descriptos en la TABLA 5.3.1.

Tabla 5.3.1 Operadores Logicos

SÍMBOLO	DESCRIPCIÓN	EJEMPLO
&&	Y (AND)	$(a > b) \&\& (c < d)$
	O (OR)	$(a > b)    (c < d)$
!	NEGACIÓN (NOT)	$!(a > b)$

Los resultados de las operaciones lógicas siempre adoptan los valores CIERTO ó FALSO. La evaluación de las operaciones lógicas se realiza de izquierda a derecha y se interrumpe cuando se ha asegurado el resultado.

El operador NEGACIÓN invierte el sentido lógico de las operaciones, así será  
 $!(a >> b)$  equivale a  $(a < b)$   
 $!(a == b)$  " "  $(a != b)$   
etc.

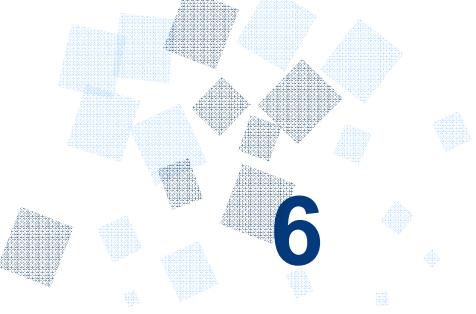
### 5.4 OPERADORES DE INCREMENTO Y DECREMENTO

Los operadores de incremento y decremento son sólo dos y están descriptos en la TABLA 5.4.1

Tabla 5.2.1 Operadores Relacionales

SÍMBOLO	DESCRIPCIÓN	EJEMPLO	ORDEN DE EVALUACIÓN
++	incremento	$+i$ ó $i++$	1
--	decremento	$-i$ ó $i-$	1

Hasta el momento se ha dicho que todo programa en el compilador PICC debe tener un encabezado, funciones si son necesarias y el programa principal.



# 6

## ENCABEZADO DE UN PROGRAMA

```
#INCLUDE <16f886.h>
#USE DELAY(CLOCK=4000000)
#FUSES XT,NOPROTECT,NOWDT,NOBROWNOUT,PUT,NOLVP
#define SW1 PORTB,2
#define SW2 PORTB,1
#define LED PORTB,0
#define BYTE PORTB= 6
INT CONT;
```

- Con la primera línea se le indica al compilador con que tipo de microcontrolador se va a trabajar.
- La segunda línea indica que se esta trabajando con un cristal de 4Mhz.

- La tercera línea consiste en la configuración de los fusibles:

XT	Tipo de oscilador cristal
NOPROTECT	Código no protegido para lectura
NOWDT	No activa el perro guardián
NOBROWNOUT	No resetea por bajo voltaje
NOPUT	No active el temporizador que retarda el funcionamiento ante la presencia de tensión de alimentación
NOLVP	No bajo voltaje de programación

- La cuarta, quinta y sexta línea consiste en definir un nombre a los diferentes bits que se van a utilizar en el programa.
- La séptima línea indica la dirección de memoria RAM del puerto B.
- La octava línea indica que se declara la variable CONT tipo entero, esta variable es global, ya que fue declarada en el encabezado del programa y se podrá utilizar tanto en el programa principal como en las diferentes funciones.

Antes de hacer un programa en C completo, es importante saber la forma de preguntar si alguna entrada esta activada o desactivada y la forma de activar o desactivar una salida.

# 7

## INSTRUCCIONES BÁSICAS

¿Cómo preguntar si una entrada esta activada?

```
IF(BIT_TEST(SW1))
{
    Sentencia;
}
```

La sentencia corresponde a la decisión que se va a tomar en caso de que la entrada este activada.

¿Cómo preguntar si una entrada esta desactivada?

```
IF(!BIT_TEST(SW1))
{
    Sentencia;
}
```

La sentencia corresponde a la decisión que se va a tomar en caso de que la entrada este desactivada.

¿Cómo activar una salida?

```
BIT_SET(LED);
```

¿Cómo desactivar una salida?

```
BIT_CLEAR(LED);
```

¿Cómo llevar un valor a un puerto?

(Tener en cuenta que cada puerto tiene máximo 8 bits)

PORTE = 15; o (Decimal)	PORTE = 0X0F; o (Hexadecimal)	PORTE = 0B00001111: (Binario)
----------------------------	----------------------------------	----------------------------------

Las tres instrucciones equivalen exactamente a lo mismo, llevar el valor 15 al puerto E, solo que en diferente formato.

### Ejemplo 7.1 Instrucciones básicas

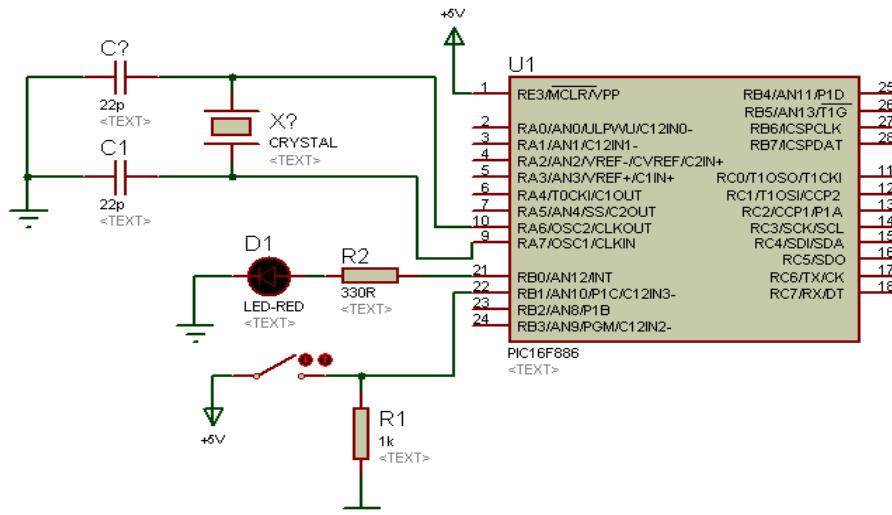


Figura 7.1 Conexión ejemplo

Encender un led conectado a RB0 si el SW conectado a RB1 esta activado

```
#INCLUDE <16f886.h>
#USE  DELAY(CLOCK=4000000)
#FUSES XT,NOPROTECT,NOWDT,NOBROWNOUT,PUT,NOLVP
#define SW PORTB,1
#define LED PORTB,0
#BYTE PORTB= 6

VOID MAIN()
{
    SET_TRIS_B(0B11111110);                                // Configura el puerto B
    WHILE(TRUE)                                            // Haga por siempre
    {
        IF(BIT_TEST(SW))                                    // Si SW esta activado
        {
            BIT_SET(LED);                                 // Active el led
        }
        ELSE
        {
            BIT_CLEAR(LED);                             // Sino, es decir si SW esta desactivado
        }
    }
}                                                       // Apagar led
}                                                       // Todo lo que se escriba después de
                                                       // estas barras se considera comentario
                                                       // y no altera para nada la ejecución del
                                                       // programa.
```

### Ejemplo 7.2 Instrucciones básicas.

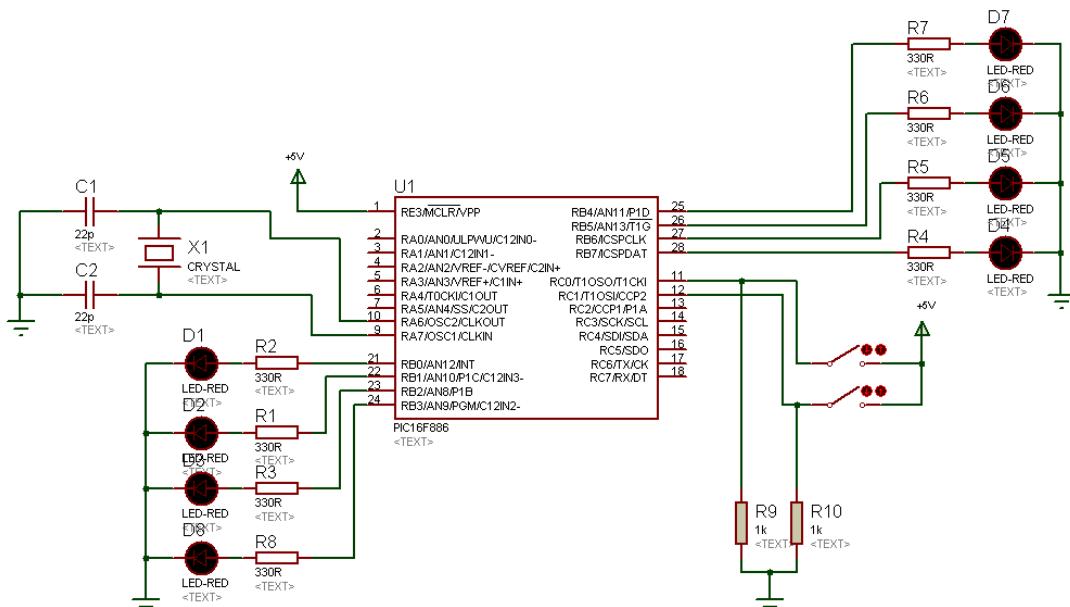


Figura 7.2 Conexión ejemplo

Encender y apagar un led conectado a RB0 cada segundo.

```
#INCLUDE <16f886.h>
#USE  DELAY(CLOCK=4000000)
#FUSES XT,NOPROTECT,NOWDT,NOBROWNOUT,PUT,NOLVP
#define LED PORTB,0
#BYTE  PORTB= 6

VOID MAIN()
{
    SET_TRIS_B(OB11111110); // Configura el puerto B
    WHILE(TRUE) // Haga por siempre
    {
        BIT_SET(LED); // Active el led
        DELAY_MS(1000) // Retardo de 1 segundo
        BIT_CLEAR(LED); // Apagar el led
        DELAY_MS(1000) // Retardo de 1 segundo
    }
}
```

### Ejemplo 7.3 Instrucciones básicas.

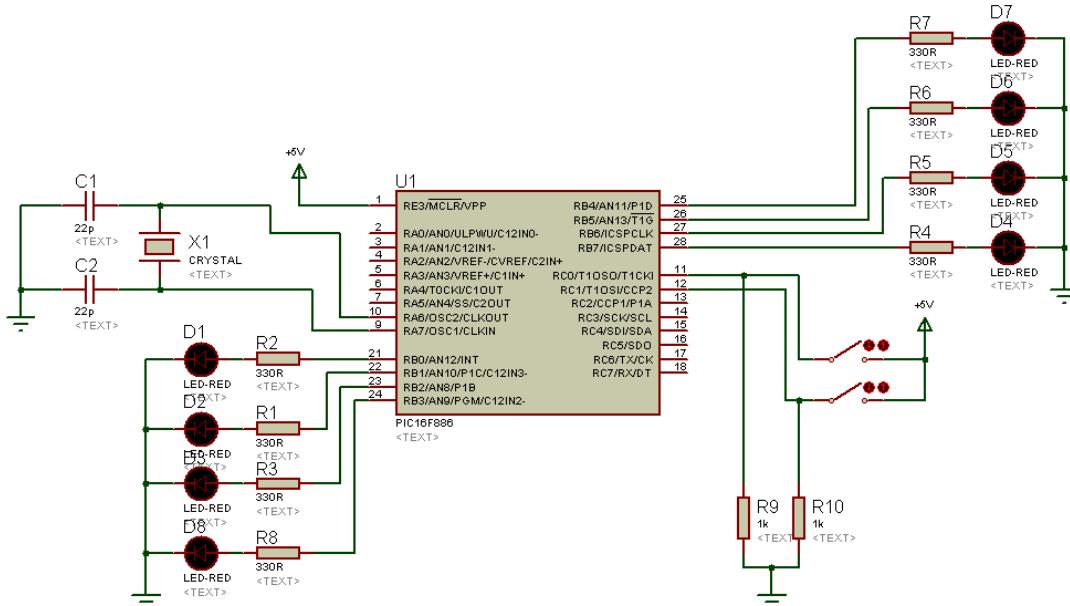


Figura 7.3 Conexión ejemplo

Encender los 8 leds conectados al puerto B si los suiches conectados a RC0 y RC1 están activados y apagarlos si están desactivados.

```
#INCLUDE <16F886.h>
#USE  DELAY(CLOCK=4000000)
#FUSES XT,NOPROTECT,NOWDT,NOBROWNOUT,PUT,NOLVP
#define SW1 PORTC,0
#define SW2 PORTC,1
#BYTE  PORTB= 6
#BYTE  PORTC= 7

VOID MAIN()
{
    SET_TRIS_C(OB11111111); // Configura el puerto C
```

```

SET_TRIS_B(0B00000000); // Configura el puerto B

WHILE(TRUE) // Haga por siempre
{
    IF(BIT_TEST(SW1)&&(BIT_TEST(SW2))) // Si SW1 Y SW2 están en 1
    {
        PORTB=OB1111111; // Active los 8 leds
    }
    IF(!BIT_TEST(SW1)&&!BIT_TEST(SW2)) // Si SW1 en 1 Y SW2 en 0
    {
        PORTB = 0; // Apague los 8 leds
    }
}
} //Cierra while
} //Cierra Main

```

#### Ejemplo 7.4 Instrucciones básicas.

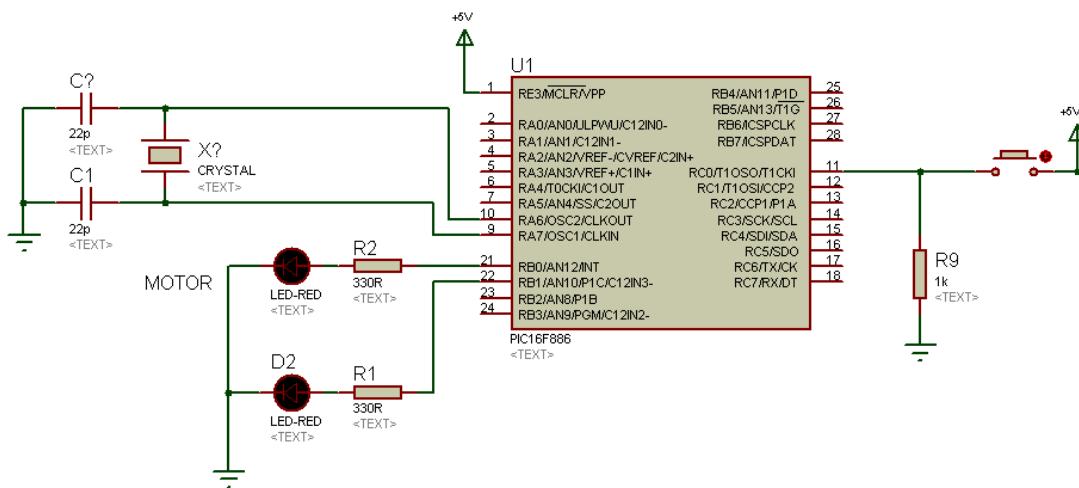


Figura 7.4 Conexión ejemplo

Hacer un programa en microcontrolador que controle un secador de manos de tal manera que cuando se oprima el pulsador encienda el motor y la resistencia durante 10 segundos.

```

#define _C16F886_
#include <16f886.h>
#use delay(clock=4000000)
#fuses XT,NOPROTECT,NOWDT,NOBROWNOUT,PUT,NOLVP
#byte PORTB=6
#byte PORTC=7
#define MOTOR PORTB,0
#define RESISTENCIA PORTB,1
#define PULSADOR PORTC,0

void main()
{
    set_tris_b(0B00000000);
    set_tris_c(0B11111111); // Configura el puerto B
    portb=0;
}

```

```

WHILE(TRUE)                                // Haga por siempre
{
    IF(BIT_TEST(PULSADOR))
    {
        BIT_SET(MOTOR);                  // Active el Motor
        BIT_SET(RESISTENCIA);            // Active la Resistencia
        DELAY_MS(10000);                // Retardo de 10 segundos
        BIT_CLEAR(MOTOR);               // Apagar el Motor
        BIT_CLEAR(RESISTENCIA);          // Apague la Resistencia
    }
}
}
}

```

---

#### Ejemplo 7.5 Instrucciones básicas.

Hacer un programa en microcontrolador que me indique si se quedaron encendidas las luces del carro después de apagarlo, mediante un sonido que prende y apaga cada 0,5 segundos y solo es posible desactivarlo si se apagan las luces.

```

#include <16F886.H>
#use   delay(clock=4000000)
#fuses xt,noprotect,nowdt,nobrownout,put,nolvp
#byte  portb=6
#byte  portc=7
#define SONIDO      PORTB,0
#define ENCENDIDO   PORTC,0
#define LUCES       PORTC,1

void main()
{
    set_tris_b(0b0000000);
    set_tris_c(0b1111111);           // Configura el puerto B
                                    // Configura el puerto C

    WHILE(TRUE)                    // Haga por siempre
    {
        if(!bit_test(ENCENDIDO)&&(bit_test(LUCES)))
        {
            bit_set(SONIDO);
            delay_ms(500);
            bit_clear(SONIDO);
            delay_ms(500);
        }
        if(!bit_test(ENCENDIDO)&&(!bit_test(LUCES)))
        {
            bit_clear(SONIDO);
        }
    }
}

```

# 8

## INSTRUCCIONES DE ROTACIÓN



En C existen dos instrucciones de Rotación, una a la izquierda y otra a la derecha:

>> 1: Rotación a la derecha, la cantidad de rotaciones es especificada por el número que tiene enseguida.

<< 1: Rotación a la izquierda, la cantidad de rotaciones es especificada por el número que tiene enseguida.

Al rotar el registro el bit es ocupado con un cero.

### Ejemplo 8.1 Instrucciones de rotación.

```
PORTB = PORTB<<1; //Rota el puerto B a la izquierda una vez
PORTB = PORTB>>1; //Rota el puerto B a la derecha una vez
```

### Ejemplo 8.2 Instrucciones de rotación.

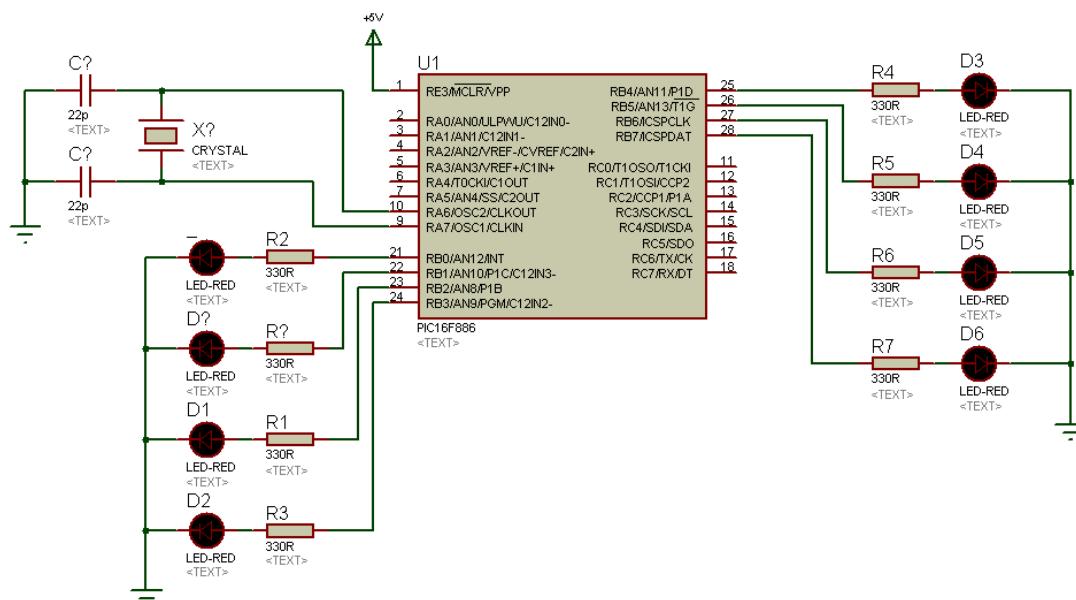


Figura 8.2 Conexión ejemplo

Encender uno a uno los bits del puerto B, cada medio segundo, comenzando por RB0. En ningún momento se pueden encender dos leds al mismo tiempo.

```
#INCLUDE <16f886.h>
#USE DELAY(CLOCK=4000000)
#FUSES XT,NOPROTECT,NOWDT,NOBROWNOUT,PUT,NOLVP
#BYTE PORTB= 6
VOID MAIN()
{
    SET_TRIS_B(OB00000000);                                // Configura el puerto B
    WHILE(TRUE)                                            // Haga por siempre
```

```

    {
        PORTB=OB00000001;                                // Active el led RB0
        DELAY_MS(500);                                  // Retardo de 500 mS
        WHILE(!BIT_TEST(PORTB,7))                        // Mientras RB7=0
        {
            PORTB= PORTB<<1;                            // Rote el PORTB una vez a la
            DELAY_MS(500);                                // izquierda
        }
    }
}

```

---

#### Ejemplo 8.3 Instrucciones de rotación.

Encender uno a uno los bits del puerto B (0.5seg) desde RB0 hasta RB7 y luego apagarlos en sentido contrario desde RB7 hasta RB0.

```

#include <16F886.h>
#use   delay(clock=4000000)
#fuses xt,noprotect,nowdt,nobrownout,put,nolvp
#define sw0 portc,0
#byte  portb=6

void main()
{
    set_tris_b(0b00000000);
    portb=0;
    while(true)
    {
        portb=0b00000001;
        delay_ms(500);

        while (!bit_test(portb,7))
        {
            portb=portb<<1;
            delay_ms(500);
        }
        delay_us(500);

        while (!bit_test(portb,0))
        {
            portb=portb>>1;
            delay_ms(500);
        }
    }
}

```

# 9

## MOTORES PASO A PASO



*Los motores paso a paso son un tipo especial de motores que permiten el movimiento de su eje en ángulos muy precisos y por pasos, tanto a la izquierda como a la derecha. Aplicando a ellos una secuencia de pulsos.*

*Cada paso tiene un ángulo muy preciso, determinado por la construcción del motor lo que permite realizar movimientos exactos.*

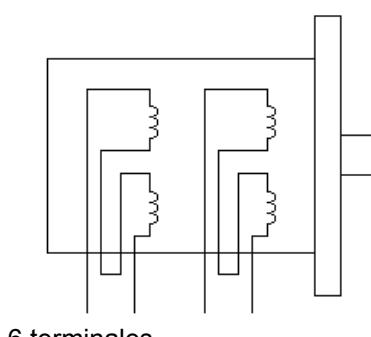
*Son utilizados para generar movimientos precisos, por ejemplo en robots, en equipos con movimientos X-Y, entre otros.*

Figura 8.2 Motores bipolares

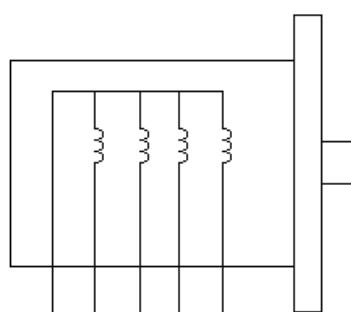
Existen dos tipos de motores paso a paso:

**Motores Unipolares:** este tipo de motor tiene dos bobinas en cada uno de los estatores y cada par de bobinas tienen un punto común, es decir, tiene 5 ó 6 terminales.

Figura 8.1 Motores unipolares

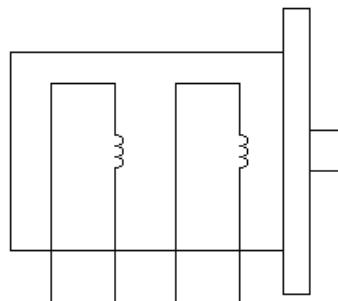


6 terminales



5 terminales

**Motores Bipolares:** este tipo de motor tiene dos bobinas y no poseen puntos comunes, es decir tiene cuatro terminales.

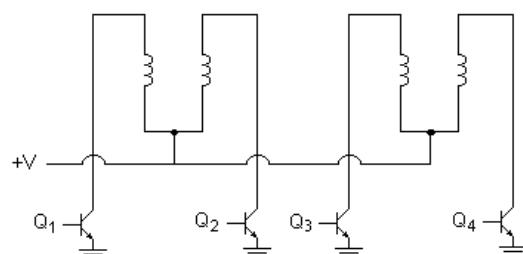


Para controlar este tipo de motor paso a paso bipolar es necesaria usar 8 transistores o circuitos integrados especiales.

### 9.1 CONTROL DEL MOTOR PASO A PASO UNIPOLAR

Para controlar el motor paso a paso se debe conocer su secuencia y sus terminales, de tal manera que el circuito o el programa en microcontrolador generen la secuencia lógica de cuatro bits que energiza una bobina del motor en el orden correcto.

Figura 9.1 Controlador motor paso a paso



Para que el motor gire a la derecha se aplica la secuencia de pulsos en un sentido y para girar a la izquierda simplemente se invierte la secuencia. Por ejemplo

**Tabla 9.1.1 Control del motor paso a paso**

### GIRO A LA DERECHA

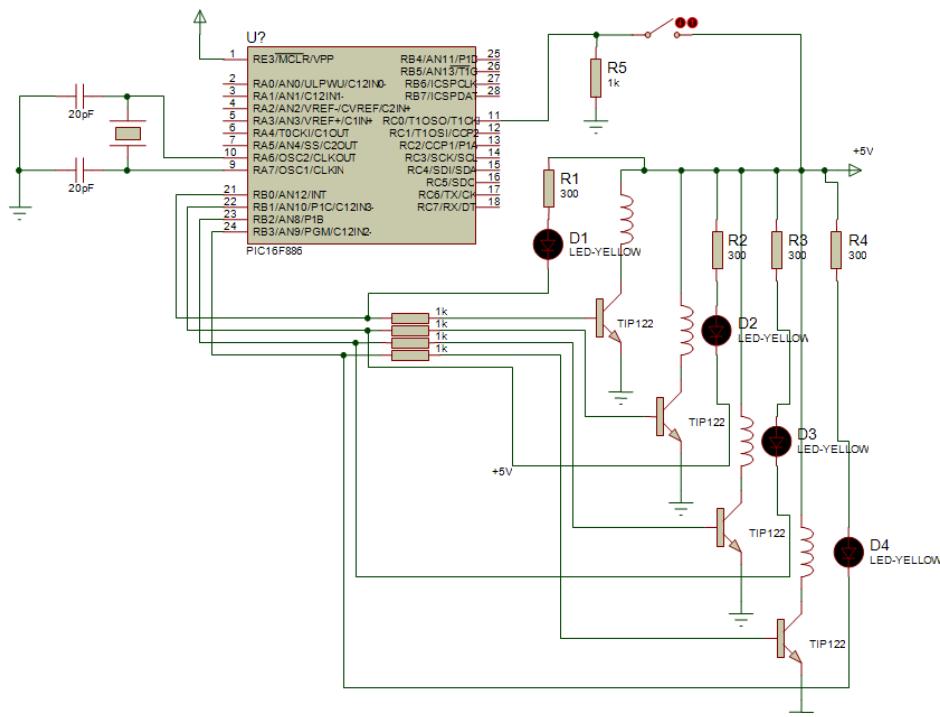
PASOS	Q <sub>1</sub>	Q <sub>2</sub>	Q <sub>3</sub>	Q <sub>4</sub>
1	ON	OFF	OFF	OFF
2	OFF	ON	OFF	OFF
3	OFF	OFF	ON	OFF
4	OFF	OFF	OFF	ON
1	ON	OFF	OFF	OFF

### GIRO A LA IZQUIERDA

PASOS	Q <sub>1</sub>	Q <sub>2</sub>	Q <sub>3</sub>	Q <sub>4</sub>
1	OFF	OFF	OFF	ON
2	OFF	OFF	ON	OFF
3	OFF	ON	OFF	OFF
4	ON	OFF	OFF	OFF
1	OFF	OFF	OFF	ON

### Ejemplo 9.1.1 Motor paso a paso.

Controlar un motor paso a paso conectado a los cuatro bits menos significativos del puerto B, de tal manera que si el suiche conectado a RC0 esta en uno gire a la derecha y si el suiche esta en cero, el motor gira a la izquierda.



**Figura 9.1.1 Conexión ejemplo**

Las señales que habilita cada transistor pueden venir desde un circuito integrado o desde el microcontrolador, el transistor lleva el negativo a la bobina del motor, en el momento en que es habilitado.

La velocidad del motor depende de la frecuencia con que se envíen los pulsos a cada transistor.

Controlar un motor paso a paso conectado a los cuatro bits menos significativos del puerto B, de tal manera que si el suiche conectado a RC0 esta en uno gire a la derecha y si el suiche esta en cero, el motor gira a la izquierda.

```
#INCLUDE <16f886.h>
#fuses XT,NOPROTECT,NOWDT,NOBROWNOUT,PUT,NOLVP,WRT
#use delay(clock=4000000)
#byte PORTB=6
#byte PORTC=7
#define SW PORTC,0

INT CONT;
byte const HORARIO[4] = {0b1100,
                        0b0110,
                        0b0011,
                        0b1001};

byte const ANTIH[4] = {0b1001,
                      0b0011,
                      0b0110,
                      0b1100};

VOID MAIN()
{
    SET_TRIS_C(OB11111111); // Configura el puerto C
    SET_TRIS_B(OB00000000); // Configura el puerto B

    WHILE(TRUE)
    {
        IF(BIT_TEST(SW)) // Pregunta si SW esta encendido
        {
            CONT=0; // Se pone Cont en cero
            WHILE((CONT<4)&&(BIT_TEST(SW))) // Mientras que cont sea menor a 4
            // y SW=1(encendido)

            {
                PORTB=(HORARIO[CONT]); // Envíe al puerto B la información
                DELAY_MS(100); // de la tabla de horario
                CONT++; // Retardo de 30 milisegundos
                // Incremente la variable cont
            }
        }
        ELSE // de lo contrario
        {
            CONT=0; // la variable cont =0
            WHILE((CONT<4)&&(!BIT_TEST(SW))) // Mientras que cont sea menor a 4
            // y SW=0(apagado)

            {
                PORTB=(ANTIH[CONT]); // Envíe al puerto B la información
                DELAY_MS(100); // de la tabla de horario
                CONT++; // Retardo de 100 milisegundos
                // Incremente la variable cont
            }
        }
    }
}
```

---

### Ejemplo 9.1.2 Motor paso a paso.

Controlar un motor paso a paso conectado a los cuatro bits menos significativos del puerto B, de tal manera que si el suiche conectado a RC0 esta en uno gire a la derecha y si el suiche conectado a RC1 esta en uno, el motor gire a la izquierda.

```
#INCLUDE <16f886.h>
#fuses    XT,NOPROTECT,NOWDT,NOBROWNOUT,PUT,NOLVP,WRT
#use      delay(clock=4000000)
#byte     PORTB=6
#byte     PORTC=7
#define   SW PORTC,0
#define   SW1 PORTC,1

INT CONT;
byte const HORARIO[4] = {0b1100,
                        0b0110,
                        0b0011,
                        0b1001};
byte const ANTIH[4]   = {0b1001,
                        0b0011,
                        0b0110,
                        0b1100};

VOID MAIN()
{
    SET_TRIS_C(0B11111111);                                // Configura el puerto C
    SET_TRIS_B(0B00000000);                                // Configura el puerto B

    WHILE(TRUE)
    {
        WHILE(BIT_TEST(SW))                                // Pregunta si SW esta encendido
        {
            CONT=0;

            WHILE((CONT<4))                                // Mientras que cont sea menor a 4
            {
                PORTB=(HORARIO[CONT]);                      // Envíe al puerto B la información de la
                DELAY_MS(500);                             // tabla de horario.
                CONT++;                                 // Retardo de 500 milisegundos
                                                // Incremente la variable cont.

            }

            PORTB=0;                                         // de lo contrario
            WHILE(BIT_TEST(SW1))                            // los bits del puerto b se apagan
            {
                CONT=0;                                     //la variable contador se carga con cero.
                WHILE((CONT<4))                            //Mientras que cont sea menor a 4
                {
                    PORTB=(ANTIH[CONT]);                  //Envíe al puerto B la información de la
                    DELAY_MS(500);                         //tabla de antihorario
                    CONT++;                               //Retardo de 500 milisegundos
                                                //Incremente la variable cont
                }

                PORTB=0;
            }
        }
    }
}
```

# 10

## MANEJO DE DISPLAY 7 SEGMENTOS Y ANTIREBOTE

### Conexión de Display 7 segmentos

Cuando se quiere mostrar datos en el display, existen dos opciones para hacerlo, una utilizar un decodificador BCD a 7 segmentos después del microcontrolador, y otra es generar con el mismo

microcontrolador el código 7 segmentos equivalente a cada número de 0 a 9. En este caso se hará el decodificador BCD a 7 segmentos con el mismo microcontrolador. Antes de explicar como mostrar datos en un display con el microcontrolador, se recuerda que hace un decodificador BCD a 7 segmentos

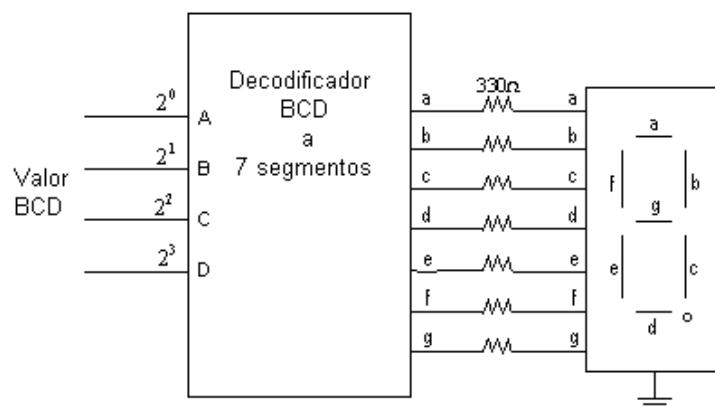


Figura 10.1 Conexión del decodificador con el display

El decodificador BCD a 7 segmentos me convierte el código BCD a código 7 segmentos, encendiendo los segmentos correspondientes al número, por ejemplo el número cero (0000 en BCD) debe encender los segmentos a, b, c, d, e y f, el número cuatro (0100 en BCD) debe encender los segmentos b, c, f y g.

Así se podría seguir indicando que segmentos se encienden con cada número, Sin embargo se explicará la forma como el microcontrolador lo hace.

Se pretende que el mismo microcontrolador maneje directamente el display sin utilizar decodificadores. La siguiente es la conexión del microcontrolador con el display de cátodo común

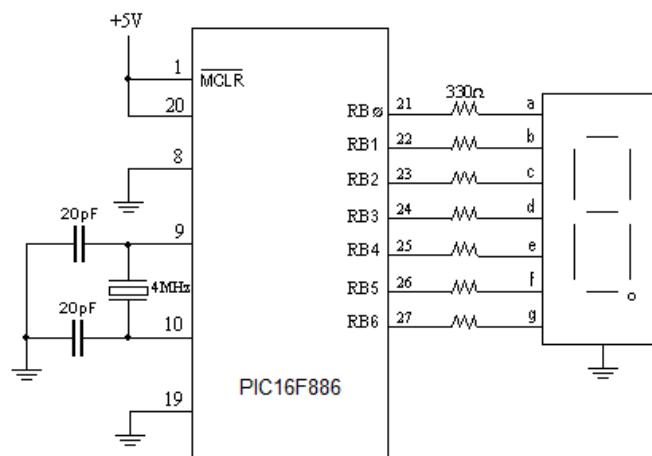


Figura 10.2 Conexión microcontrolador con el display

Para lograr este objetivo se necesita entender el manejo de estructuras en C, que funcionan como una tabla donde el programa puede consultar el contenido de cada

posición. Lógicamente en cada posición se encuentra el código 7 segmentos de cada número de 0 a 9.

**Tabla 10.1 Tabla para los códigos 7 segmentos.**

NUMERO	CODIGO 7 SEGMENTOS								Numero en hexadecimal
	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	
		g	f	e	d	c	b	a	
0	0	0	1	1	1	1	1	1	3F
1	0	0	0	0	0	1	1	0	06
2	0	1	0	1	1	0	1	1	5B
3	0	1	0	0	1	1	1	1	4F
4	0	1	1	0	0	1	1	0	66
5	0	1	1	0	1	1	0	1	6D
6	0	1	1	1	1	1	0	0	7D
7	0	0	0	0	0	1	1	1	07
8	0	1	1	1	1	1	1	1	7F
9	0	1	1	0	0	1	1	1	67

Para manejar una estructura en C, lo único que se debe hacer es añadir la estructura al encabezado.

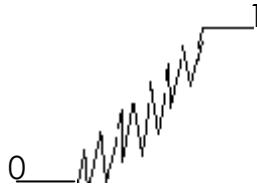
```
Byte CONST display[10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x67};
```

El número 10 indica la cantidad de posiciones de la estructura, los códigos 7 segmentos están en hexadecimal.

# 11

## TRABAJOS CON PULSADORES (ANTIRREBOTE)

- Cuando se trabaja con pulsadores o suiches, en el momento en que estos cambian de estado se genera una señal como la que se muestra en la figura:



Es decir, el suiche o pulsador genera unos y ceros hasta que finalmente se estabiliza debido a que es un dispositivo electromecánico, el microcontrolador con su velocidad de trabajo se da cuenta de esto.

Para evitar errores en la detección de pulsadores y suiches se utilizan retardos de 150 a 200ms aproximadamente, en el momento en que se detecta que un pulsador o suiche cambió de estado, este tiempo es suficiente mientras se estabiliza y luego, se pregunta nuevamente por el estado del suiche.

### Ejemplo 11.1 Display 7 segmentos.

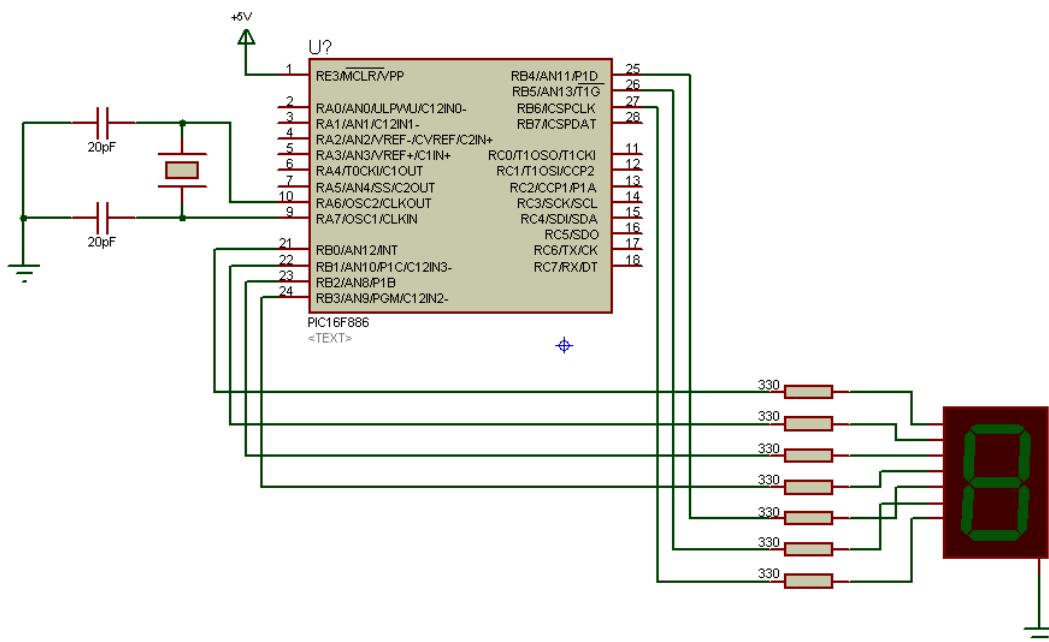


Figura 11.1 Conexión ejemplo

Hacer un contador de 0 a 9 cada segundo, mostrando el valor del contador en un display 7 segmentos. Al llegar a 9, el contador comienza de nuevo.

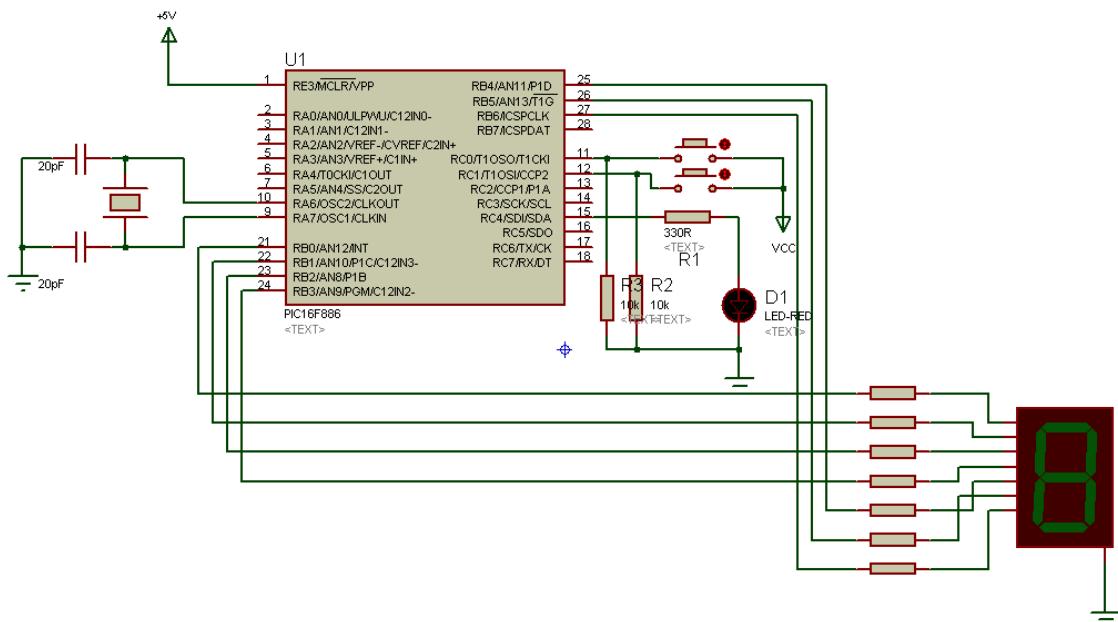
```
#INCLUDE <16F886.H>
#USE    DELAY(CLOCK=4000000)
#FUSES  XT,NOPROTECT,NOWDT,NOBROWNOUT,PUT,NOLVP
BYTE CONST DISPLAY[10]= {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x67};
#BYTE   PORTB=6

INT CONTADOR;

VOID MAIN()
{
    SET_TRIS_B(0B00000000);
    WHILE(TRUE)                                // Configura el puerto B
                                                // Haga por siempre
```

```
{  
    CONTADOR = 0;                                // Inicializa contador en cero  
  
    WHILE(CONTADOR<10)                          // Mientras contador < 10  
    {  
        PORTB= DISPLAY[CONTADOR];                // Muestre el valor en el display  
        CONTADOR++;                            // Incrementa contador  
        DELAY_MS(1000);                         // Retardo de 1 segundo  
    }  
}
```

## Ejemplo 11.2 Display 7 segmentos.



**Figura 11.2 Conexión ejemplo**

Realizar un temporizador programable con 2 pulsadores, con un suiche incrementa y con el otro da Start para empezar a decrementar.

```
#INCLUDE <16F886.H>
#USE    DELAY(CLOCK=4000000)
#FUSES  XT,NOPROTECT,NOWDT,NOBROWNOUT,PUT,NOLVP
BYTE CONST DISPLAY[10]= {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x67};
#BYTE   PORTB=6
#BYTE   PORTC=7
#define SW1 PORTC,0
#define SW2 PORTC,1
#define LED PORTC,4
INT CONTADOR;

VOID MAIN()
{
    SET_TRIS_B(0B00000000);          // Configura el puerto B
    SET_TRIS_C(0B00001111);          // Configura el puerto B
    CONTADOR = 0;                   // Inicializa contador en cero
```

```

WHILE(TRUE)                                // Haga por siempre
{
    PORTB= DISPLAY[CONTADOR];
    BIT_CLEAR(LED);
    IF (BIT_TEST(SW1))
    {
        DELAY_MS(200);                      // Antirrebote
        CONTADOR++;
        PORTB= DISPLAY[CONTADOR];
        IF(CONTADOR==10)
        {
            CONTADOR=0;
        }
    }
    IF (BIT_TEST(SW2))
    {
        DELAY_MS(200);
        BIT_SET(LED);
        WHILE (CONTADOR>0)
        {
            PORTB= DISPLAY[CONTADOR];
            DELAY_MS(1000);
            CONTADOR--;
        }
        BIT_CLEAR(LED);
    }
}

```

---

# 12

## MULTIPLEXAJE DE DISPLAY

En muchas ocasiones se requiere mostrar números en el display de más de un dígito, es decir, 2, 3, o 4 dígitos.

Si se pretende controlar cada display, se necesitan siete (7) líneas del microcontrolador por cada uno, esto ocuparía todas las líneas disponibles en cada puerto del microcontrolador, sin embargo existe una técnica llamada multiplexaje que consiste en conectar a las mismas 7 líneas los 2,3 o 4 display e ir encendiendo uno a uno los display, a través de un transistor, tan rápidamente que parece encenderse todos al mismo tiempo.

Cualquier elemento que se encienda y apague con una frecuencia mayor a 25Hz es imperceptible para el ojo humano, éste lo verá encendido en todo momento.

El circuito para manejar 2 display multiplexados puede ser el siguiente:

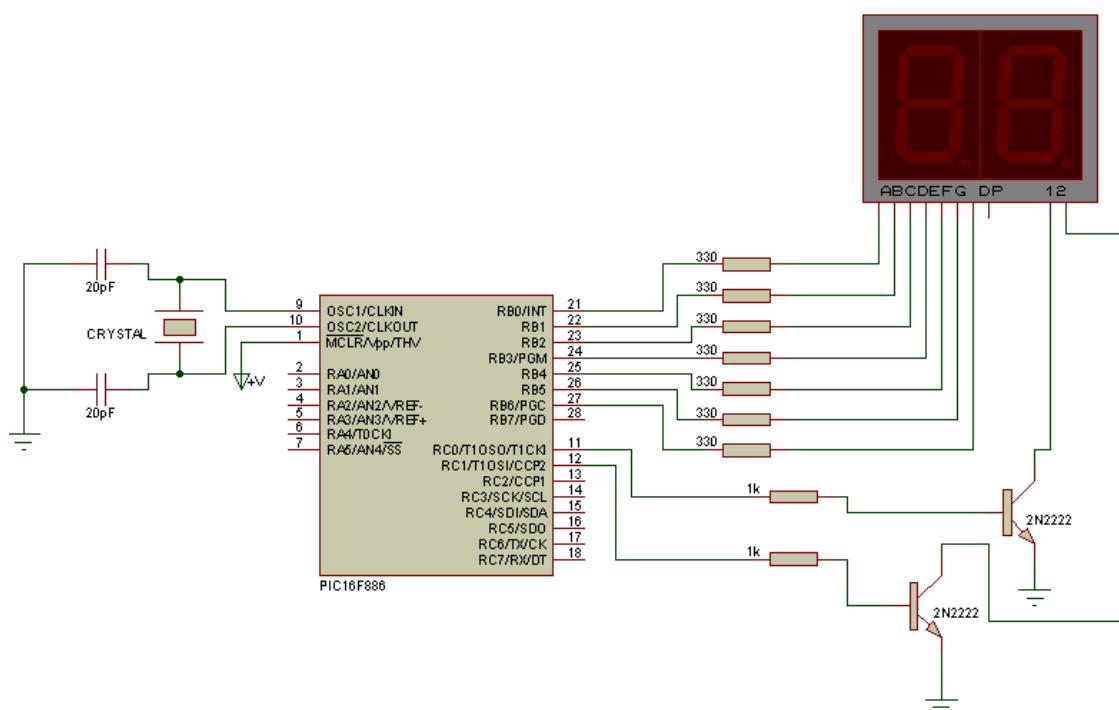


Figura 12.1 Circuito para manejar dos display multiplexados

Nota: los segmentos de cada display van unidos entre sí, es decir a con a, b con b, hasta el g con g, por cada display adicional se necesita un transistor y sólo una línea más del microcontrolador.

En este diagrama se asume que las unidades están en el display de la derecha y las decenas en el display de la izquierda.

### Ejemplo 12.1 Multiplexaje de display.

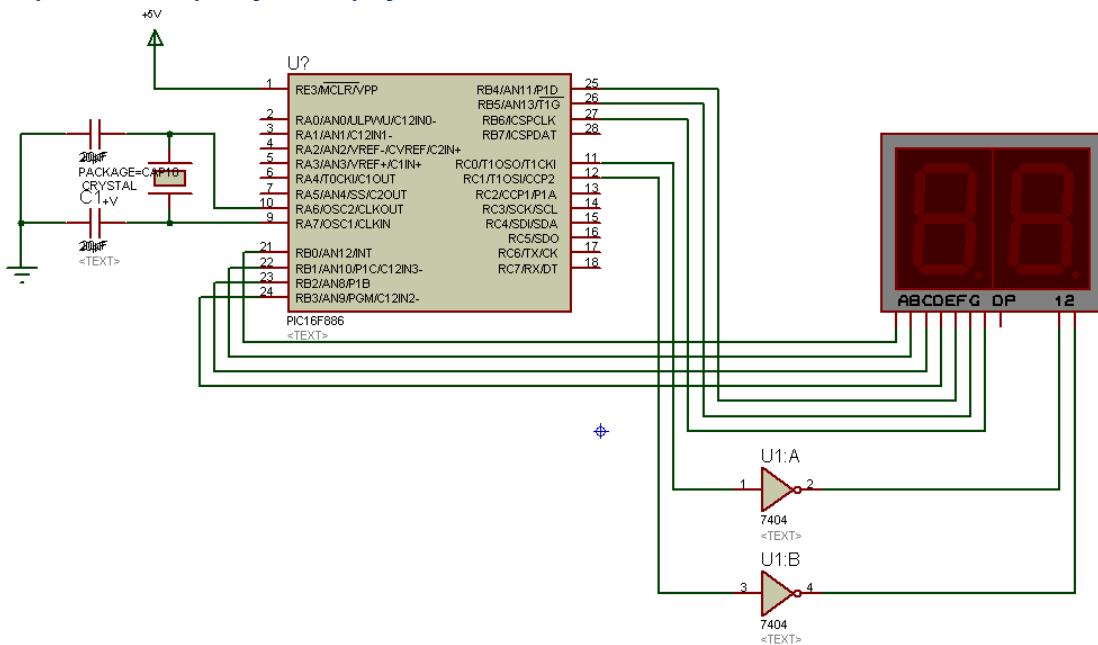


Figura 12.2 Conexión ejemplo

Hacer un contador de 0 a 99 que incremente cada segundo.

```
#INCLUDE <16F886.h>
#USE   DELAY(CLOCK=4000000)
#FUSES XT,NOPROTECT,NOWDT,NOBROWNOUT,PUT,NOLVP
Byte CONST display[10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x67};
#BYTE  PORTB= 6
#BYTE  PORTC= 7
#define TUNI PORTC,1           // Definición de variables
#define TDEC PORTC,0           // Definición de variables

INT CONT;                      // Declarar la variable CONT como
                                // un entero, es decir de 8 bits
LONG CONTRRET;                 // Declarar la variable CONTRRET
                                // como long, es decir de 16 bits

VOID MOSTRAR()
{
    INT UNI,DEC;              // Declarar las variables UNI, DEC
                                // como un entero, es decir de 8bits

    DEC=CONT/10;               // Apaga el display de decenas
    UNI=CONT%10;               // Enciende el display de unidades
                                // Muestra lo que hay en unidades
                                // en el display
                                // Retardo de 1 milisegundos

    BIT_CLEAR(TDEC);           // Apaga el display de unidades
    BIT_SET (TUNI);            // Enciende el display de decenas
    PORTB=(DISPLAY[DEC]);      // Muestra lo que hay en unidades
                                // en el display
                                // Retardo de 1 milisegundos

    DELAY_MS(1);               // Apaga el display de unidades
                                // Enciende el display de decenas
                                // Muestra lo que hay en unidades
                                // en el display
                                // Retardo de 1 milisegundos

    BIT_CLEAR(TUNI);
    BIT_SET (TDEC);
    PORTB=(DISPLAY[DEC]);
    DELAY_MS(1);
}
```

```

VOID RET1SEG()
{
    CONTRET=500;                                // Rutina RET1SEG
    WHILE (CONTRET>0)                            // Cargue con 500 la variable
                                                // CONTRET
    {
        MOSTRAR();                               // Mientras que la variable
        CONTRET--;                             // CONTRET sea mayor que cero
    }
}

VOID MAIN()
{
    SET_TRIS_B(0);                            // El puerto B esta configurado
                                                // como salida
    SET_TRIS_C(0B11001111);                  // El puerto C esta configurado
                                                // como entrada
    CONT=0;                                    // la variable CONT se inicializa con cero
    WHILE(TRUE)                                // Haga por siempre
    {
        CONT=0;                                // mientras la variable CONT es
        WHILE(CONT<100)                         // menor que 100
        {
            RET1SEG();                          // Llama la rutina RET1SEG
            CONT++;                            // Incrementa la variable CONT
        }
    }
}

```

En el ejemplo anterior se utilizan funciones, son pequeños subprogramas que se pueden llamar desde el programa principal. Se debe digitar antes del programa principal que es quien las llama.

Si la función no retorna ningún valor se pone la directiva VOID antes del nombre de la función indicando que no retorna ningún valor.

### Ejemplo 12.2 Multiplexaje de display.

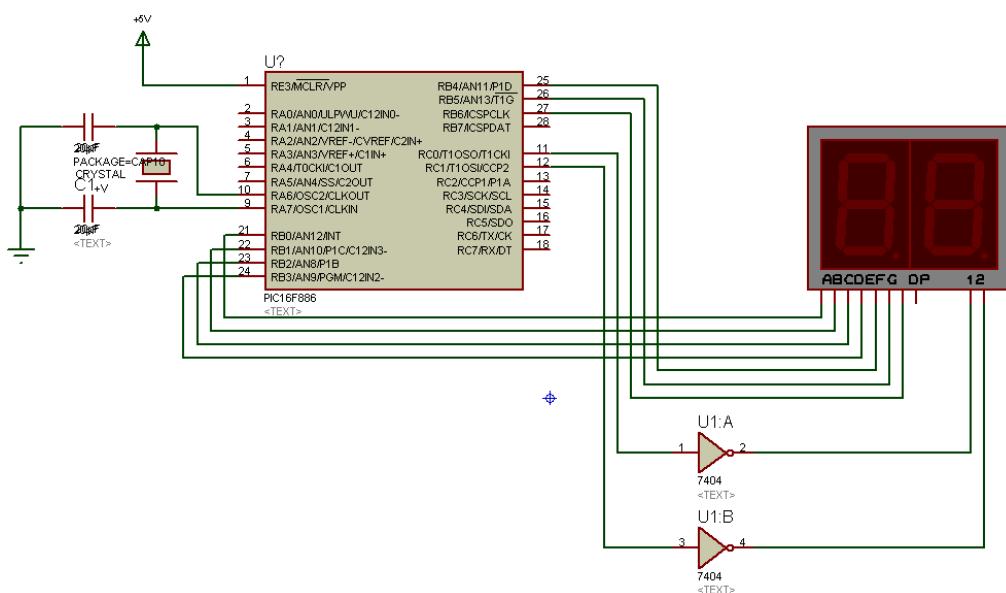


Figura 12.2 Conexión ejemplo

Mostrar un valor fijo en el display, partiendo de un número decimal.

```
#INCLUDE <16F886.h>
#USE    DELAY(CLOCK=4000000)
#FUSES  XT,NOPROTECT,NOWDT,NOBROWNOUT,PUT,NOLVP
Byte CONST display[10]= {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x67};
#BYTE   PORTB= 6
#BYTE   PORTC= 7
#define  TUNI PORTC,1           // Definición de variables
#define  TDEC PORTC,0           // Definición de variables
INT CONT;                      // Declarar la variable CONT
                                // como un entero.

VOID MOSTRAR( )                // Rutina mostrar
{
    INT UNI,DEC;               // Declarar las variables UNI
                                // DEC como un entero
    DEC=CONT/10;               // Lo que hay en la variable.
    UNI=CONT%10;               // CONT, llévelo a Y

    BIT_CLEAR(TDEC);           // Apaga el display de decena
    BIT_SET (TUNI);            // Prende el display de unidades
    PORTB=(DISPLAY[DEC]);      // Muestra lo que hay en unidades

    DELAY_MS(1)                // Retardo de 1 milisegundos
                                // en el display
    BIT_CLEAR(TUNI);           // Apaga el display de unidades
    BIT_SET (TDEC);            // Prende el display de decenas
    PORTB=(DISPLAY[DEC]);      // Muestra lo que hay en unidades
                                // en el display
    DELAY_MS(1);               // Retardo de 1 milisegundos
}

VOID MAIN()
{
    SET_TRIS_B(0);
    SET_TRIS_C(0B11001111);    // El puerto B esta configurado como salida
                                // El puerto C esta configurado como entrada,
                                // excepto los bits RC4 y RC5

    CONT=0;                    // la variable CONT se inicializa con cero
    WHILE(TRUE)                // Haga por siempre
    {
        CONT=76;
        WHILE(TRUE)              // Haga por siempre
        {
            MOSTRAR();           // Llamar la rutina Mostrar
        }
    }
}
```

# 13

## INTERRUPCIONES

- Si Las llamadas funciones desde el programa principal hacen que el programa ejecute un subprograma y luego regrese al programa principal, sin embargo el programador esta definiendo en que momento debe saltar a ejecutar la función mediante las instrucciones, por esta razón se considera sincronas.

Las interrupciones son desviaciones de flujo de control del programa originadas asincrónicamente por diversos sucesos que no dependen del programador, es decir, ocurren en cualquier momento.

Las interrupciones ocurren por sucesos externos como la generación de un flanco o nivel en una patica del microcontrolador o eventos internos tales como el desbordamiento de un contador, terminación del conversor análogo a digital, entre otras.

El comportamiento del microcontrolador ante la interrupción es similar al procedimiento que se sigue al llamar una función desde el programa principal. En ambos casos se detiene la ejecución del programa en curso, se guarda la dirección a donde debe retornar cuando termine de ejecutar la interrupción, atiende o ejecuta el programa correspondiente a la interrupción y luego continua ejecutando el programa principal, desde donde lo dejó cuando fue interrumpido. Existen 13 diferentes causas que producen una interrupción, por lo tanto el primer paso de la rutina de interrupción será identificar la causa de la interrupción.

Sólo se trata en esta guía la interrupción externa por cambio de estado en la patica RB0.

Los pasos que se deben seguir para atender una interrupción, son los siguientes:

- Digitar la función correspondiente a la interrupción. La función debe comenzar con # y la interrupción correspondiente, por ejemplo para la función de interrupción por RB0 se digita #int\_EXT
- En el programa principal, habilitar las interrupciones en forma global, con la instrucción:  
enable\_interrupts(GLOBAL);
- En el programa principal, habilitar la interrupción correspondiente, como ejemplo se muestra como habilitar la interrupción externa por RB0:  
enable\_interrupts(INT\_EXT);

### Ejemplo 13.1 Interrupciones.

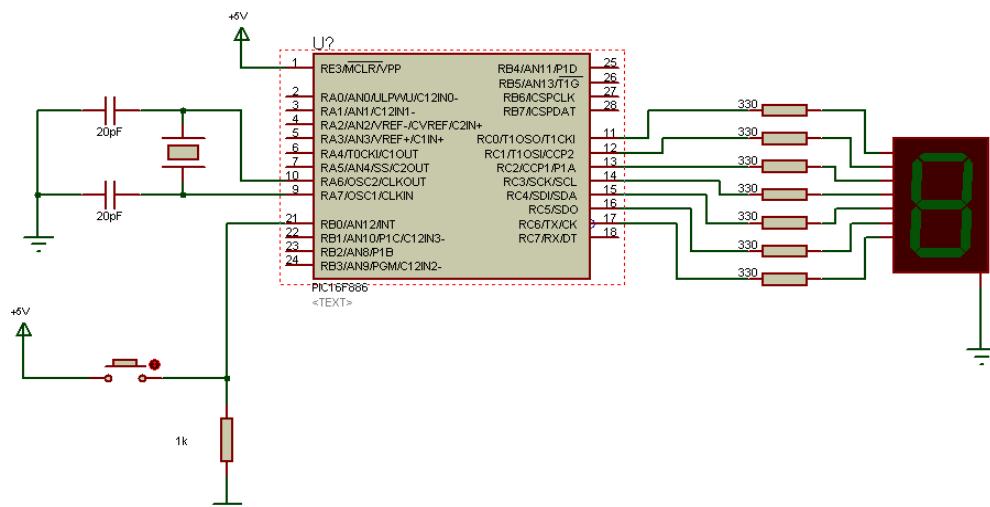


Figura 13.1 Conexión ejemplo

Hacer un programa que me incremente un contador de 0 a 9, cuando ocurra una interrupción externa por RB0. Cuando el contador llegue a 9, comienza de nuevo en cero.

```
#INCLUDE <16F886.H>
#FUSES XT,NOLVP,NOWDT,PUT
#USE DELAY(CLOCK=4000000)
Byte CONST display[10] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x67};
#BYTE PORTC = 7
#BYTE PORTB = 6
INT CONT=0;

#INT_EXT
VOID INTERRUPCIÓN()
{
    DELAY_MS(200);
    CONT++;
    // Incrementa la variable CONT
    IF(CONT==10)
        CONT=0;
    // Si la variable CONT es igual a 10
    // Ponga en cero la variable CONT
}

VOID MAIN()
{
    SET_TRIS_B(0B11111111);           // Configurar el puerto B
    SET_TRIS_C(0);                  // Configurar el puerto C
    ENABLE_INTERRUPTS(GLOBAL);      // Habilita todas las interrupciones
    ENABLE_INTERRUPTS(INT_EXT);    // Habilita la interrupción externa

    WHILE(TRUE)
    {
        PORTC= DISPLAY[CONT];       // Muestra lo que hay en la variable
        // CONT en el display
    }
}
```

Existen diferentes tipos de interrupción en el microcontrolador, algunas de ellas se mencionan a continuación:

```
#INT_EXT  INTERRUPCIÓN EXTERNA
#INT_RTCC  DESBORDAMIENTO DEL TIMER0(RTCC)
#INT_RB   CAMBIO EN UNO DE LOS PINES B4,B5,B6,B7
#INT_AD   CONVERSOR A/D
#INT_EEPROM  ESCRITURA EN LA EEPROM COMPLETADA
#INT_TIMER1  DESBORDAMIENTO DEL TIMER1
#INT_TIMER2  DESBORDAMIENTO DEL TIMER2
```

# 14

## TIMER

### Control del Timer con interrupciones

El microcontrolador PIC16F873 tiene 3 temporizadores el Timer 0 (8 bits), el Timer 1(16 bits) y el Timer 2(8 bits). A pesar del Timer 0 ser de 8 bits es el temporizador principal.

El Timer 0 también llamado RTCC se puede cargar con un valor cualquiera entre 0 y 255 y puede ser incrementado a través del Reloj interno y dividido por un valor que se puede escoger entre los que se indican a continuación.

RTCC_DIV_2,	RTCC_DIV_4,	RTCC_DIV_8,
RTCC_DIV_16,		RTCC_DIV_32,
RTCC_DIV_64,		RTCC_DIV_128,
RTCC_DIV_256.		

La interrupción RTCC se produce cada vez que el contador TIMER0 pasa de 255 a 0. Si se trabaja el Microcontrolador con un cristal de 4 Mhz, esta frecuencia se divide internamente por 4, es decir realmente trabaja a 1Mhz, o sea que cada ciclo de reloj dura aproximadamente 1 microsegundo.

Para entender el funcionamiento del Timer 0, como ejemplo se supone que se necesita generar una interrupción cada 20 ms. (20.000 microsegundos).

¿Qué fórmula usar para determinar con qué valor se debe cargar inicialmente el Timer 0 y qué valor de preescaler o división se debe utilizar?

La fórmula para aplicar después de pasar el tiempo de temporización a microsegundos es:

Tiempo en microsegundos/ Valor de división = valor del timer, sin embargo se debe tener en cuenta que la interrupción se genera cuando el timer pasa de 255 a 0, es decir:

Realmente el valor inicial del timer es: (256 - Valor inicial del Timer 0)

Al seleccionar el preescaler o división se debe tratar de obtener un valor entero al dividir el tiempo del retardo sobre el

preescaler. Este valor no puede ser mayor a 256. En caso de ser mayor, significa que antes de cumplir el retardo el Microcontrolador habrá generado más de una interrupción.

En este caso se combina entonces la programación del Timer 0 y cada vez que el Timer 0 genere una interrupción se decrementa un contador tantas veces como sea necesario hasta completar el tiempo de temporización.

Finalmente el procedimiento que se debe seguir para calcular el valor inicial del timer y el valor del contador a decrementar es el siguiente:

Tiempo en Microsegundos/Valor de división o preescaler

Me genera un valor mayor a 255, por lo tanto es necesario calcular cuantas veces va a generar una interrupción antes de completar el retardo de un segundo.

El resultado de la división anterior debe ser igual a:

Valor inicial del Timer 0 x Valor del contador a decrementar.

Posterior a esta multiplicación,

256 - Valor inicial del timer 0

### Ejemplo 14.1 Timer.

Para un retardo de un segundo, con preescaler de 256, se procede de la siguiente manera:  
1 segundo = 1000000 microsegundos

1000000/RTCC\_DIV\_"X"

X = Este valor puede ser cualquiera de los indicados al principio, el que se elija será con el que se seguirá trabajando en la programación. En este caso el RTCC escogido es 256.

$$1000000/256 = 3906.25 \text{ aproximadamente } 3906$$

El valor anterior es decir 3906 debe ser igual a la multiplicación entre el valor inicial del timer 0 y el valor del contador a decrementar.

3906 = valor inicial del timer \* valor del contador a decrementar

Observación: estos dos valores son aleatorios y deben ser menores a 256.

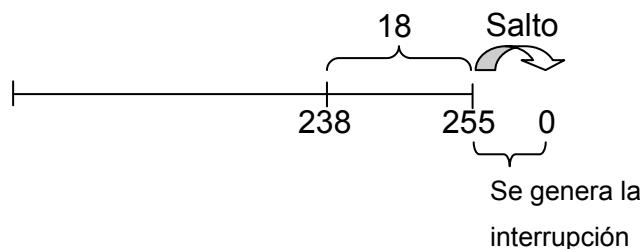
$$3906 = 18 * 217$$

Cualquiera de estos dos valores pueden ser el valor inicial del timer 0. En este caso, se elige 18 como valor inicial del timer 0. Al obtener el valor inicial del timer 0 se debe restar el RTCC utilizado, en este caso 256 para obtener el número donde el temporizador debe iniciar para que en el momento que incremente las 18 veces llegue al valor del RTCC utilizado (256) y produzca la interrupción ó salto. Es decir:

RTCC – Valor inicial del timer

$$256 - 18 = 238$$

xisten diferentes tipos de interrupción en el microcontrolador, algunas de ellas se mencionan a continuación:



#### Ejemplo 14.2 Timer.

Realizar un temporizador de 0 a 60 segundos.

```
#INCLUDE <16f886.h>
#USE   DELAY(CLOCK=4000000)
#fuses XT,NOPROTECT,NOWDT,NOBROWNOUT,PUT,NOLVP
Byte CONST display[10] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x67};
#define U PORTC,0
#define D PORTC,1
#define PORTB=6
#define PORTC=7
INT UNI,DEC,I,VECES,SEG;

VOID MOSTRAR()                                // Muestra en el display los segundos
{
    I=SEG;
    UNI=0;
    DEC=0;

    WHILE(I>=10)
    {
        I=I -10;
        DEC++;
    }
    UNI=I;

    BIT_SET(U);
    BIT_CLEAR(D);
```

```

PORTB=display[UNI];
DELAY_MS(1);

BIT_SET(D);
BIT_CLEAR(U);
PORTB=display[DEC];
DELAY_MS(1);
}

#INT_RTCC                                // Rutina de interrupción por RTCC
VOID RELOJ()
{
    VECES--;
    SET_RTCC(238);

    IF(VECES==0)
    {
        SEG++;
        VECES=217;
    }
}

VOID MAIN()
{
    SET_TRIS_B(0);
    SET_TRIS_C(0);
    UNI=0;
    DEC=0;
    VECES=217;
    SEG=0;

    SET_RTCC(238);
    SETUP_COUNTERS(RTCC_INTERNAL, RTCC_DIV_256);
    ENABLE_INTERRUPTS(INT_RTCC);
    ENABLE_INTERRUPTS(GLOBAL);

    WHILE(TRUE)
    {
        IF(SEG==60)
        SEG=0;
        ELSE
        MOSTRAR();
    }
}

```

---

#### Ejemplo 14.3 Timer.

Hacer un contador de 0 a 9, de tal manera que incremente cada segundo por interrupción a través del timer, al llegar a 9 comienza de nuevo.

```

#define _XTAL_FREQ 4000000
#include <16F886.H>
#use delay(clock=_XTAL_FREQ)
#fuses XT,NOLVP,NOWDT
byte const display[10]={0X3F,0X06,0X5B,0X4F,0X66,0X6D,0X7D,0X07,0X7F,0X67};
#byte portb=6
#define ints_per_second 15          // (4000000/(4*256*256)) 4=4mhz, 256=
                                         // División 256, 256=Timer 256 se carga

```

```

// con cero.

BYTE SEGUNDOS;
BYTE INT_CONTADOR;
#INT_RTCC
VOID CLOCK_ISR() {
    --INT_CONTADOR; // Esta función es llamada cada vez que el RTCC pasa de (255->0).
    // Para este programa es aproximadamente 76 veces por segundo
}

IF(INT_CONTADOR==0)
{
    ++SEGUNDOS;
    INT_CONTADOR=INTS_PER_SECOND;
}

VOID MAIN()
{
    SET_TRIS_B(0);
    INT_CONTADOR=INTS_PER_SECOND;
    SET_RTCC(0);
    SETUP_COUNTERS(RTCC_INTERNAL, RTCC_DIV_256);
    ENABLE_INTERRUPTS(INT_RTCC); //Habilita la interrupción por RTCC
    ENABLE_INTERRUPTS(GLOBAL); // Habilita todas las interrupciones

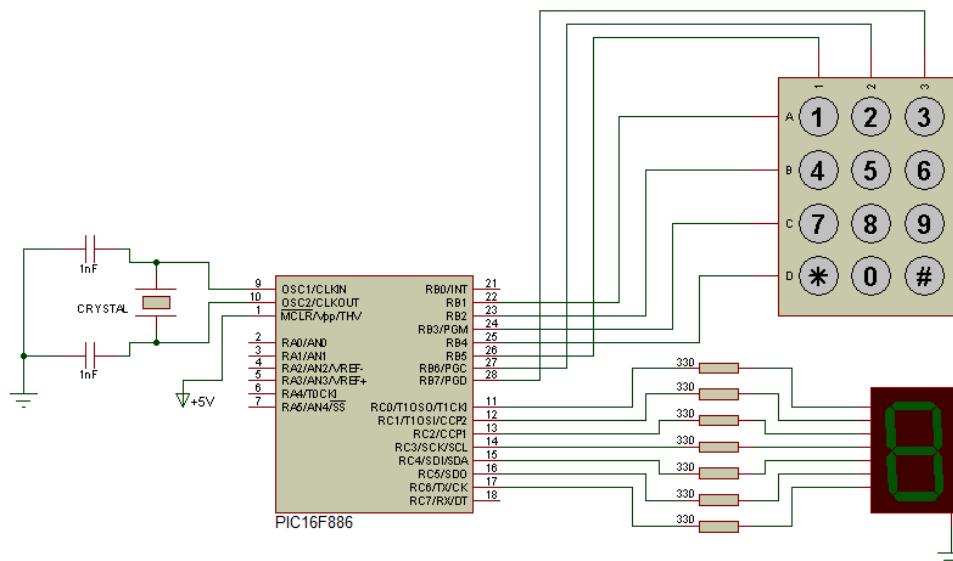
    DO
    {
        IF(SEGUNDOS==10)
        {
            SEGUNDOS=0;
        }
        PORTB=(DISPLAY[SEGUNDOS]); // Muestra lo que hay en la variable // SEGUNDOS en el display
    }
    WHILE (TRUE);
}

```

# 15

## MANEJO DEL TECLADO TELEFÓNICO

- En el compilador hay un driver para manejar un teclado telefónico, que es de gran utilidad en algunas aplicaciones donde el usuario necesita digitar un número.  
La conexión entre el teclado y el microcontrolador es la siguiente:



**Figura 15.1 Conexión del microcontrolador con el teclado y el display**

Los pasos que se deben seguir para manejar un teclado telefónico son:

- Incluir en el encabezado el driver para manejar el teclado telefónico:  
`#INCLUDE<KBD.C>`
- Por defecto el teclado se conecta al puerto D, como el microcontrolador que se usa no tiene puerto D se conecta al puerto B y se debe añadir al encabezado la siguiente línea:  
`#DEFINE USE_PORTB_KBD`
- En el programa principal habilitar las resistencias pullup del puerto B, con esto simplemente se habilitan internamente unas resistencias del puerto B a +V.  
`PORT_B_PULLUPS(TRUE);`

### Ejemplo 15.1 Teclado telefónico.

Mostrar en un display 7 segmentos el valor digitado en el teclado.

- Inicializar el driver del teclado en el programa principal.  
`KBD_INIT()`

- Llamar la función del teclado y almacenar el valor digitado en una variable tipo carácter. Si no se oprime ninguna tecla el teclado retorna el carácter nulo.  
`K=KBD_GETC(); // K debe ser una variable tipo caracter (char)`

NOTA: Si se utiliza otro microcontrolador y se conecta el teclado telefónico al puerto D se debe poner resistencias a +5V en RD1, RD2, RD3 y RD4.

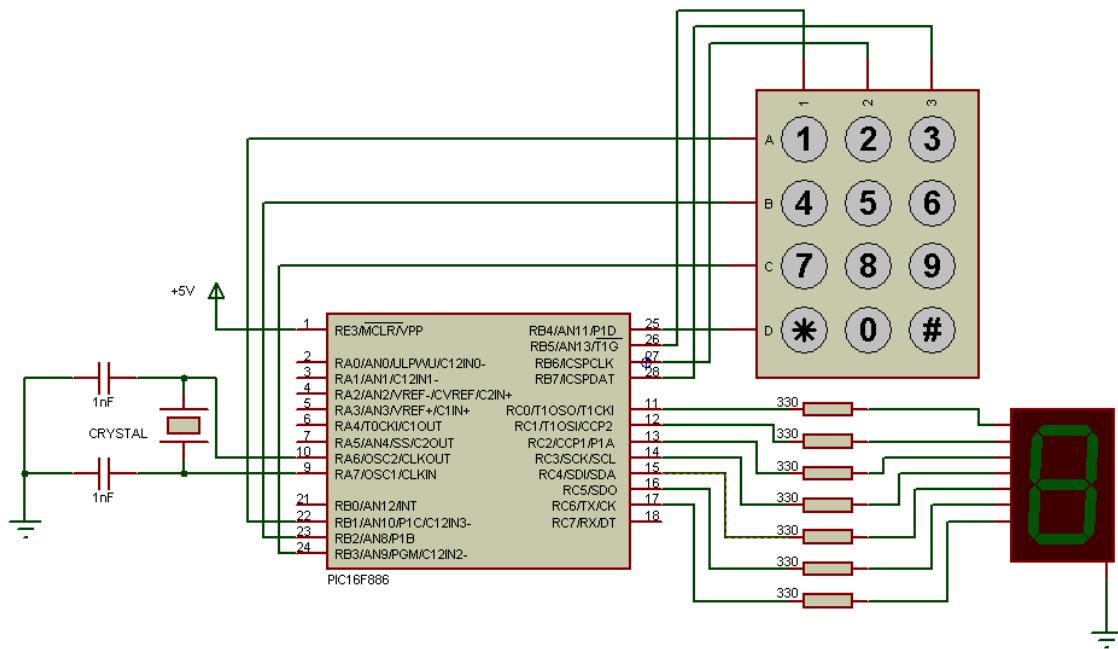


Figura 15.1.1 Conexión ejemplo

```

#include <16F886.H>
#FUSES XT,NOPROTECT,NOWDT,NOBROWNOUT,PUT,NOLVP
#USE DELAY(CLOCK=4000000)
Byte CONST display[10] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x67};
#define USE_PORTB_KBD
#include <KBD.C>
#BYTE PORTC= 7
#BYTE PORTB= 6
#BYTE WPUB=0x95
CHAR K;

VOID MAIN()
{
    PORT_B_PULLUPS(TRUE);
    WPUB=255; // Habilitar resistencias del puerto B a
    KBD_INIT(); // positivo // Inicializar la rutina del teclado
    SET_TRIS_C(0); // Configurar el puerto C como salida
    PORTC=(display[0]); // Muestra el cero en el display

    WHILE (TRUE)
    {
        K=KBD_GETC(); // Captura cualquier tecla oprimida
        IF(K=='0') // Si la tecla que se oprime es igual al
                    // caracter cero
                    // Muestre en el display el n mero cero
        IF(K=='1') // Si la tecla que se oprime es igual al
                    // caracter uno
                    // Muestre en el display el n mero uno
        IF(K=='2') // Tecla que se oprime igual al
                    // caracter dos
                    // Muestre en el display el n mero dos
        IF(K=='3') // Tecla que se oprime igual al caracter tres
                    // Muestre en el display el n mero tres
        IF(K=='4') // Tecla que se oprime igual caracter
                    // cuatro
    }
}

```

```

PORTC=( display [4]);
IF(K=='5')

PORTC=( display [5]);
IF(K=='6')

PORTC=( display [6]);
IF(K=='7')

PORTC=( display [7]);
IF(K=='8')

PORTC=( display [8]);
IF(K=='9')

PORTC=( display [9]);
}

}

```

// Muestre en el display el número cuatro  
// Tecla que se oprime igual caracter  
// cinco  
// Muestre en el display el número cinco  
// Tecla que se oprime igual caracter  
// seis  
// Muestre en el display el número seis  
// Tecla que se oprime igual caracter  
// siete  
// Muestre en el display el número siete  
// Tecla que se oprime igual caracter  
// ocho  
// Muestre en el display el número ocho  
// Tecla que se oprime igual caracter  
// nueve  
// Muestre en el display el número nueve

### Ejemplo15.2 Teclado telefónico.

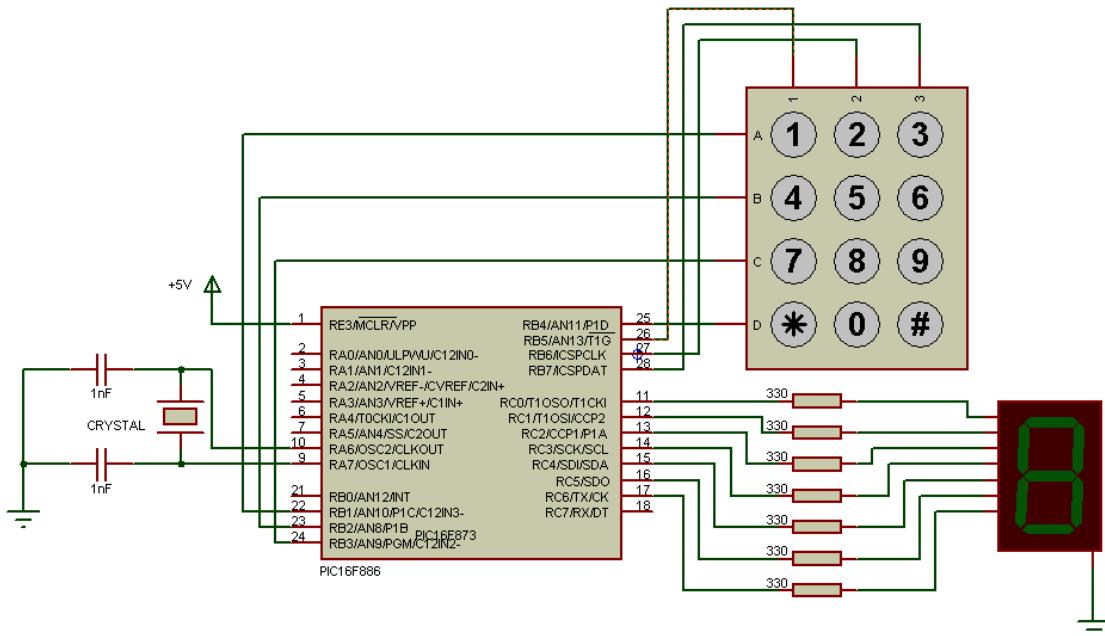


Figura 15.2 Conexión ejemplo

Mostrar el numero digitado por el teclado en el display y cuando se oprima la tecla #, el display comienza a decrementar cada segundo hasta llegar a cero.

```

#include <16F886.H>
#FUSES XT,NOPROTECT,NOWDT,NOBROWNOUT,PUT,NOLVP
#USE DELAY(CLOCK=4000000)
Byte CONST display[10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x67};
#define USE_PORTB_KBD
#include <KBD.C>
#define PORTC=7
#define PORTB=6
#define WPUB=0x95
CHAR K;

```

```

INT CONT;

VOID MAIN()
{
    PORT_B_PULLUPS(TRUE);
    WPUB=255;
    KBD_INIT();
    SET_TRIS_C(0B00000000);
    CONT=0;
    PORTC=( display [CONT]);
    WHILE (TRUE)
    {
        K=KBD_GETC();
        IF(K=='0')
        {
            CONT=0;
            PORTC=( display [CONT]);
        }
        IF(K=='1')
        {
            CONT=1;
            PORTC=( display [CONT]);
        }
        IF(K=='2')
        {
            CONT=2;
            PORTC=( display [CONT]);
        }
        IF(K=='3')
        {
            CONT=3;
            PORTC=( display [CONT]);
        }
        IF(K=='4')
        {
            CONT=4;
            PORTC=( display [CONT]);
        }
        IF(K=='5')
        {
            CONT=5;
            PORTC=( display [CONT]);
        }
        IF(K=='6')
        {
            CONT=6;
            PORTC=( display [CONT]);
        }
        IF(K=='7')
        {
    }
    // Habilitar resistencias del puerto B a
    // positivo
    // Inicializar la rutina del teclado
    // Configurar el puerto C como salida
    // Carga el contador con cero
    // Muestra el cero en el display
    // Si la tecla que se oprime es
    // igual al caracter uno
    // Muestre en el display el uno
    // Si la tecla que se oprime es igual al
    // caracter uno
    // Muestre en el display el número uno
    // Si la tecla que se oprime es igual
    // al caracter dos
    // Muestre en el display el número dos
    // Si la tecla que se oprime es igual a
    // al caracter tres
    // Muestre en el display el número tres
    // Si la tecla que se oprime es igual
    // al caracter cuatro
    // Muestre en el display el número cuatro
    // Si la tecla que se oprime es igual al
    // caracter cinco
    // Muestre en el display el número cinco
    // Si la tecla que se oprime es igual
    // al caracter seis
    // Muestre en el display el seis
    // Si la tecla que se oprime es igual
    // al caracter siete
}

```

```

CONT=7;
PORTC=( display [CONT]);
// Muestre en el display el número
// siete

}
IF(K=='8')
{
CONT=8;
PORTC=( display [CONT]);
// Muestre en el display el número ocho

}
IF(K=='9')
{
CONT=9;
PORTC=( display [CONT]);
// Muestre en el display el número nueve

}
IF(K=='#')
{
WHILE(CONT>0)
{
DELAY_MS(1000);
CONT--;
PORTC=( display [CONT]);
// Mientras el contador sea mayor a cero
// Llama retardo de un segundo
// Decrementa el contador
// Muestra el contador en el display.

}
}
}

```

---

# 16

## MANEJO DEL LCD (DISPLAY DE CRISTAL LÍQUIDO)

Existe en el compilador un driver para manejar un display de cristal líquido de 2 líneas por 16 caracteres cada una.

El procedimiento para trabajar con el LCD es parecido al procedimiento del teclado telefónico.  
La conexión entre el LCD y el microcontrolador es la siguiente:

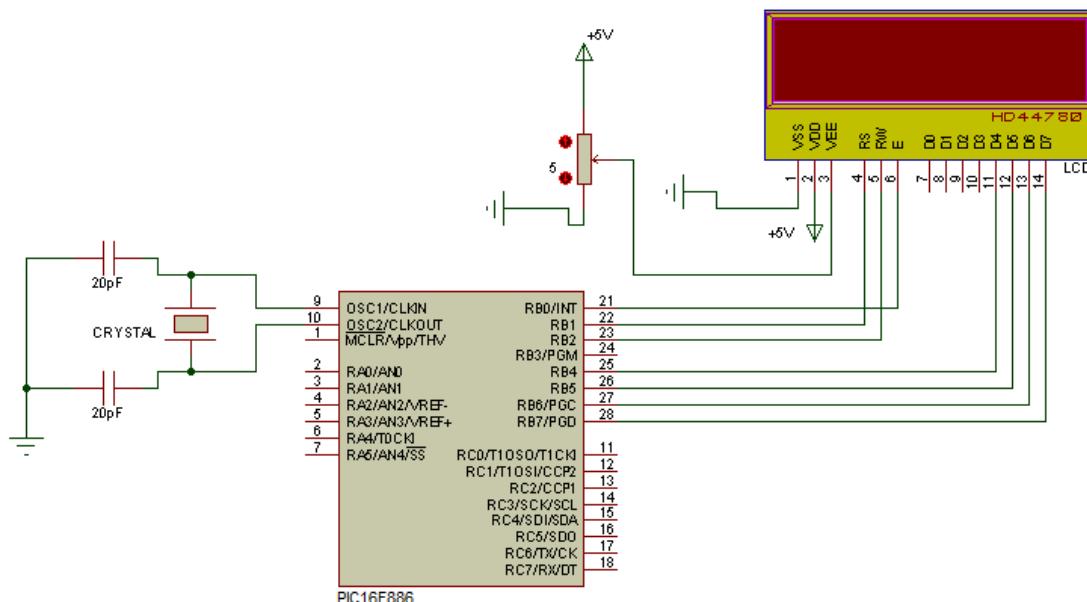


Figura 16.1 Conexión entre el LCD y el microcontrolador

Los pasos que se deben seguir para manejar el LCD son:

1. Incluir en el encabezado del programa el driver para manejar el teclado. #INCLUDE<LCD.C>}
2. Por defecto el LCD se conecta al puerto D, pero como no se tiene puerto D, se conecta al puerto B y se incluye esta línea en el encabezado: #DEFINE USE\_PORTB\_LCD TRUE
3. En el programa principal se inicializa el driver LCD\_INIT();
4. Se usan las funciones del LCD que tiene implementadas el driver:

**LCD\_PUTC(c):** Muestra el carácter “C” en la próxima posición del LCD.

**LCD\_PUTC(“/f”):** Borra todo lo que haya en el display.

**LCD\_PUTC(“/n”):** Va al inicio de la segunda línea.

**LCD\_PUTC(“/b”):** Se mueve una posición hacia atrás.

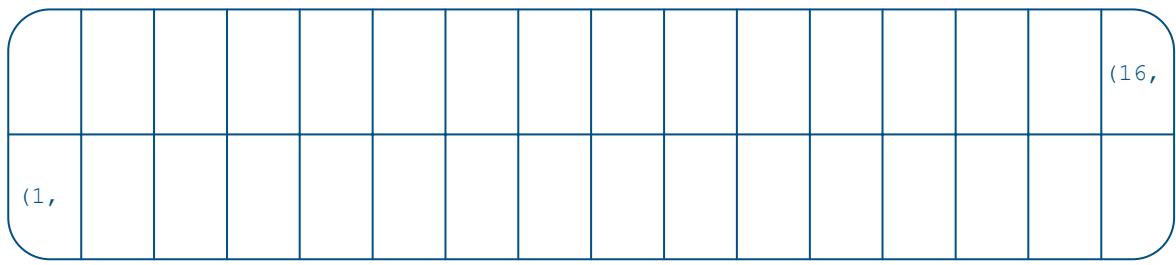
**LCD\_GOTOXY(x,y):** Ubica el cursor en la posición indicada por “X” y “Y”. La posición de la esquina superior izquierda es (1,1).

**LCD\_GETC(x,y):** Retorna el carácter ubicado en la posición X,Y del LCD.

### 16.1 EL LCD Y LA CONFIGURACIÓN DE LA PANTALLA

Las filas y las columnas en el LCD están distribuidas como se muestra en la figura:

El primer número equivale a la fila y el segundo numero a la columna.



### Ejemplo 16.1 LCD.

Mostrar un mensaje fijo en el LCD que en la primera fila “GUIA EN PIC C” y en la segunda “EAFIT”.

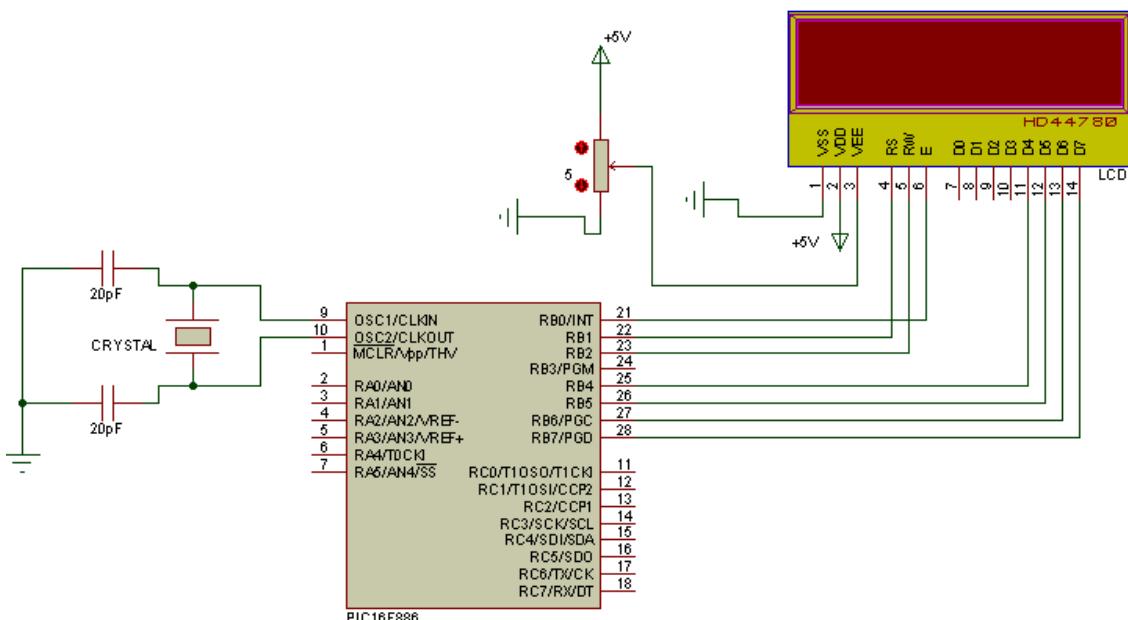


Figura 16.2 Conexión ejemplo

```
#INCLUDE <16F886.H>
#USE DELAY(CLOCK=4000000)
#define USE_PORTB_LCD TRUE
#include <LCD.C>
#FUSES XT,NOPROTECT,NOWDT,NOBROWNOUT,PUT,NOLVP,WRT
#BYTE PORTB= 6

VOID MAIN()
{
    SET_TRIS_B(0); // Define el puerto B como salida
    PORTB=0; // Inicializa el puerto B en cero
    LCD_INIT(); // Inicializa el LCD
    LCD_PUTC("f"); // Borrar el contenido del LCD

    WHILE(TRUE)
    {
        LCD_GOTOXY(1,1); // En la fila 1
        LCD_PUTC(" GUIA EN PIC C "); // Muestre el mensaje "GUIA EN
        // PIC C
        LCD_GOTOXY(1,2); // Ubicarse en la columna 1 fila 2
    }
}
```

```

        LCD_PUTC(" EAFIT   ");
    }
}

```

---

### Ejemplo 16.2 LCD.

Mostrar un mensaje diferente de acuerdo al pulsador que se oprima.

```

#include <16F886.h>
#use delay(clock=4000000)
#fuses xt,noprotect,nowdt,nobrownout,noinput,nolvp
#define use_portb_lcd true
#include <lcd.c>
#byte portc=7
#define pulsador1 portc,0
#define pulsador2 portc,1
#define pulsador3 portc,2

void main()
{
    set_tris_c(0b11111111);
    lcd_init();
    while(true)
    {
        if(bit_test(pulsador1))
        {
            delay_ms(200);
            lcd_putc("\f");
            lcd_gotoxy(1,1);
            lcd_putc("SELECCIONE MENU:");//Mostrar el "SELECCIONE MENU"
        }

        if(bit_test(pulsador2))
        {
            delay_ms(200);
            lcd_putc("\f");
            lcd_gotoxy(1,2);
            lcd_putc("MENU 1:");//Mostrar el mensaje "MENU 1"
        }

        if(bit_test(pulsador3))
        {
            delay_ms(200);
            lcd_putc("\f");
            lcd_gotoxy(1,1);
            lcd_putc("MENU 2:");//Mostrar el mensaje "MENU 2"
        }
    }
}

```

---

### Ejemplo 16.3 LCD.

Mostrar el mensaje “BIENVENIDOS” en la primera línea del LCD, que se vaya desplazando y luego el mensaje fijo “SELECCIONE MENU” en la segunda línea.

```

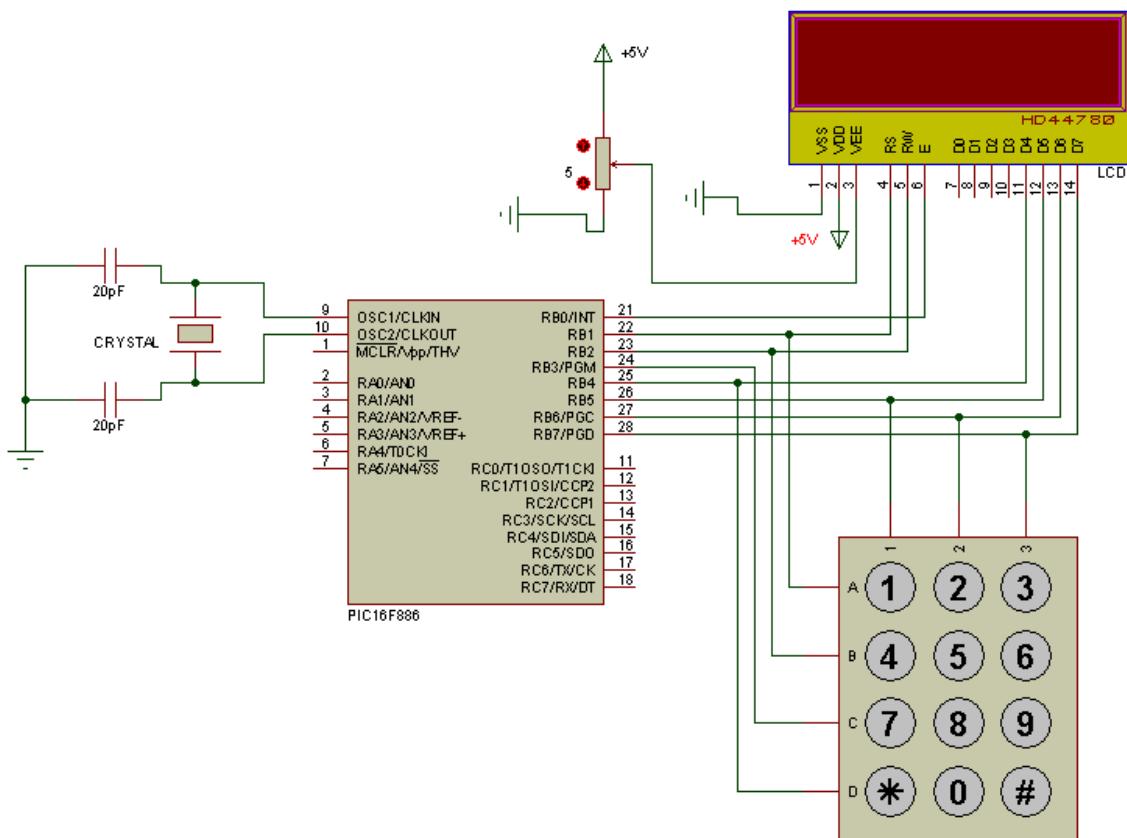
#define <16F886.h>
#use   delay(clock=4000000)
#fuses xt,noprotect,nowdt,nobrownout,noinput,nolvp
#define use_portb_lcd true
#include <lcd.c>
#byte  PORTC= 7

byte j;
void main()
{
    set_tris_c(0b11111111);
    lcd_init(); // Configurar el puerto C como entrada
                // Inicializar el LCD

    j=16;
    while (j>0) // Mientras J sea mayor que 0
    {
        lcd_putc("f");
        lcd_gotoxy(j,1); // Borrar el contenido del LCD
        lcd_putc(" BIENVENIDOS "); // En la fila 1
        delay_ms(150); // Mostrar el mensaje "BIENVENIDOS"
                        // Esperar 150 milisegundos
        j--;
    }
    while(1) // Haga por siempre
    {
        lcd_gotoxy(j,2); // En la fila 2
        lcd_putc("SELECCIONE MENU:"); // Mostrar "SELECCIONE MENU"
    }
}

```

NOTA: Se puede conectar el LCD y el teclado al mismo puerto, de la siguiente manera:



**Figura 16.3 Conexión del teclado y el LCD al mismo tiempo**

**Ejemplo 16.4 LCD y teclado.**

Mostrar en el LCD los números digitados por el teclado telefónico, hasta completar 16 números.

```
#INCLUDE <16F886.h>
#USE DELAY(CLOCK=4000000)
#FUSES XT,NOPROTECT,NOWDT,NOBROWNOUT,NOPUT,NOLVP
#define USE_PORTB_LCD TRUE
#define USE_PORTB_KBD           // Por defecto el teclado se conecta al puerto D,
                                // como el microcontrolador que se esta usando
                                // no tiene puerto D se conecta al puerto B.*/

#include <LCD.C>
#include <KBD.C>                // Incluir en el encabezado el driver para
                                // manejar el teclado telefónico:*/
#define PORTC= 7
#define INC PORTC,0
#define DEC PORTC,1
#define START PORTC,2
CHAR K;
INT DIR;

VOID MAIN()
{
    port_b_pullups(TRUE);        // En el programa principal habilitar las
                                // resistencias pullup del puerto B, con esto
                                // simplemente se habilitan internamente unas
                                // resistencias del puerto B a +V.*/
    SET_TRIS_C(255);
    LCD_INIT();                  // Inicializar el driver del teclado en el
    KBD_INIT();                  // programa principal
    LCD_PUTC("\f");

    WHILE(TRUE)
    {
        DIR=0;
        LCD_PUTC("\f");
        while (DIR<17)
        {
            k=kbd_getc();          // Llamar la función del teclado y almacenar el
                                // valor digitado en una variable tipo carácter. Si
                                // no se oprime ninguna tecla el teclado
                                // retornara el carácter nulo.*/
            WHILE( (k=='\0'))      // si no se oprime ninguna tecla sigue
                                // llamando al teclado.
            {
                k=kbd_getc();
            }
            if( (k!=='*')&&(k!='#'))
            {
                lcd_putc(k);
                DIR++;
            }
        }
    }
}
```

### Ejemplo 16.5 LCD y teclado.

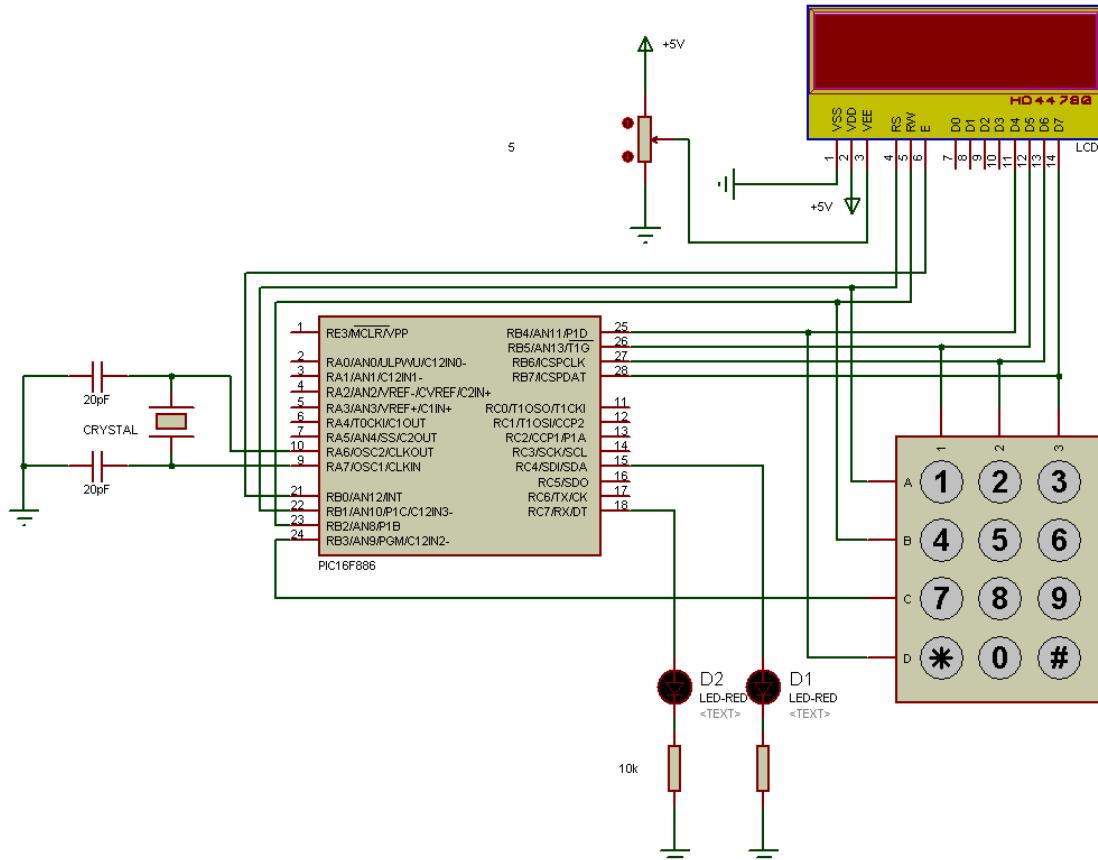


Figura 16.5 Conexión ejemplo

Digitar una clave en el teclado telefónico, en el LCD mostrar si es correcta o incorrecta. En el momento de estar digitando la clave en el LCD se muestran asteriscos en vez de los números.

```

#include <16f886.h>
#use     delay(clock=4000000)
#fuses   xt,noprotect,nowdt,nobrownout,noinput,nolvp
#define   use_portb_lcd true
#define   use_portb_kbd // Se añade esta instrucción porque por defecto
                     // el teclado se conecta al puerto D en este
                     // caso se está usando el puerto B.*/
                     // Incluir en el encabezado el driver para
                     // manejar el teclado telefónico:*/
#include<kbd.c>
#include<lcd.c>
#byte   portc=7
#byte   portb=6
#byte   wpub=0x95
#define   led1 portc,7
#define   led2 portc,4

int cont;
byte j=16;
char k;
int mil,cen,dec,uni,val;

void teclado()
{

```

```

k=kbd_getc();                                // Llamar la función del teclado y almacenar
                                              // el valor digitado en una variable tipo
                                              // carácter. Si no se oprime ninguna tecla el
                                              // teclado retornara el carácter nulo.*/
                                              // si no se oprime ninguna tecla sigue
                                              // llamando al teclado.

    {
        k=kbd_getc();
    }
IF( (k!='\0'))
{
    IF(K=='0')
        VAL=0;
    IF(K=='1')
        VAL=1;
    IF(K=='2')
        VAL=2;
    IF(K=='3')
        VAL=3;
    IF(K=='4')
        VAL=4;
    IF(K=='5')
        VAL=5;
    IF(K=='6')
        VAL=6;
    IF(K=='7')
        VAL=7;
    IF(K=='8')
        VAL=8;
    IF(K=='9')
        VAL=9;
}
VOID MAIN()
{
    PORT_B_PULLUPS(TRUE);                      // En el programa principal habilitar las
                                              // resistencias pullup del puerto B, con esto
                                              // simplemente se habilitan internamente unas
                                              // resistencias del puerto B a +V.*/

    WPUB=255;
    SET_TRIS_C(0B00000000);
    KBD_INIT();                                 // Inicializar el driver del teclado en el
                                              // programa principal

    LCD_INIT();
    PORTC=0;
    WHILE(TRUE)
    {
        LCD_GOTOXY(1,1);
        LCD_PUTC(" BIENVENIDOS ");
        LCD_GOTOXY(1,2);
        LCD_PUTC(" DIGITE CLAVE ");
        DELAY_MS(1000);

        TECLADO();
        LCD_PUTC("\f");
        LCD_GOTOXY(1,1);

        IF((k]!='#')&&(k]!='*')
        {
            lcd_putc('*');
        }
    }
}

```

```

UNI=VAL;
}
TECLADO();

IF((k]!='#')&&(k!='*'))
{
lcd_putc('*');
DEC=VAL;
}
TECLADO();
IF((k]!='#')&&(k!='*'))
{
lcd_putc('*');
CEN=VAL;
}
TECLADO();
IF((k]!='#')&&(k!='*'))
{
lcd_putc('*');
MIL=VAL;
}
TECLADO();
WHILE((k]!='#'))
{
TECLADO();
}

IF((UNI==1)&&(DEC==2)&&(CEN==3)&&(MIL==4)) // Aquí se compara si
// los números digitados
// están correctos.*/

{
LCD_PUTC("\f");
LCD_GOTOXY(1,1); // Se borra LCD
// Se ubica en la posición 1,1
LCD_PUTC(" CLAVE CORRECTA ");
BIT_SET(LED1);
DELAY_MS(2000);
BIT_CLEAR(LED1);
}

ELSE

{
LCD_PUTC("\f");
LCD_GOTOXY(1,2);
LCD_PUTC(" CLAVE INVALIDA ");
BIT_SET(LED2);
DELAY_MS(4000);
BIT_CLEAR(LED2);
}
}
}

```

# 17

## ALMACENAMIENTO EN MEMORIA EEPROM INTERNA

El microcontrolador tiene memoria eeprom interna, almacenar información en esta memoria tiene la gran ventaja de que los datos almacenados en ésta no se borren a menos que se sobrescriba sobre ellos, es decir la información almacenada allí, no se borrara así se desenergice el microcontrolador. Aunque esta memoria es limitada es de gran utilidad en algunos controles.

La instrucción para almacenar información en la memoria eeprom interna es la siguiente:  
WRITE\_EEPROM(dirección, valor)

La **dirección** puede ser de 0 a 63, **valor** es un byte.

La instrucción para leer información en la memoria eeprom interna es la siguiente:

READ\_EEPROM(dirección)

Dirección puede ser de 0 a 63.

### Ejemplo 17.1 Memoria eeprom interna

Hacer un programa que:

Si RC3=1, inicializa el contador y la dirección de memoria.

Si RC0=1, almacena un contador de 0 a 14.

Si RC2=1, almacena números pares de 0 a 14.

Si RC1=1, muestra el valor almacenado

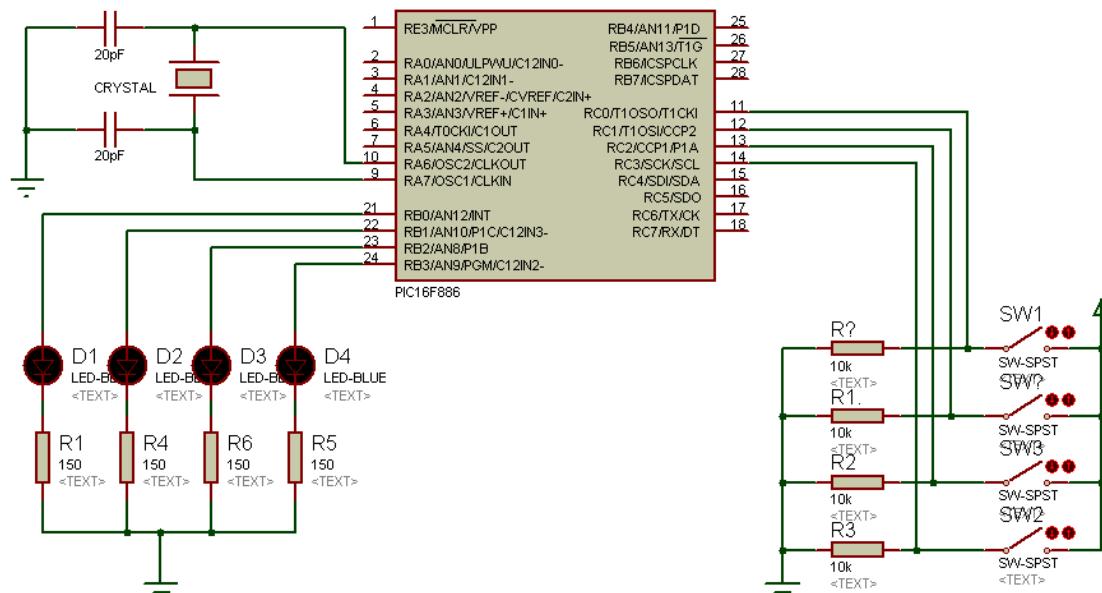


Figura 17.1 Conexión ejemplo

```
#INCLUDE <16F886.H>
```

```
#FUSES XT,NOPROTECT,NOWDT,NOBROWNOUT,PUT,NOLVP,WRT
```

```
#USE DELAY(CLOCK=4000000)
```

```
#BYTE PORTC= 7
```

```

#BYTE PORTB= 6

INT CONT,DIRECCION,CONT2;           // Declarar las variable CONT,
                                     // DIRECCION, CONT2 como un entero

VOID MAIN()
{
SET_TRIS_C(0B11111111);           // Configura el puerto C como entrada
SET_TRIS_B(0);                    // Configura el puerto B como salida
PORTB=0;                          // Poner el puerto B en 0
CONT=0;                           // Poner la variable CONT en cero
DIRECCION=0;                      // Poner la variable DIRECCION en cero
CONT2=2;                          // Poner la variable CONT2 en dos
WHILE (TRUE)                      // LOOP INFINITO
{
  IF(BIT_TEST(PORTC,3))           // Si el bit 3 del puerto C esta en 1
  {
    CONT=0;                       // Poner la variable CONT en cero
    DIRECCION=0;                  // Poner la variable DIRECCION en cero
  }
  IF(BIT_TEST(PORTC,0)&&(CONT<15)) // Si RC0=1 y la variable CONT es
//menor que 15
  {
    WRITE_EEPROM(DIRECCION,CONT); // Escribir en eeprom el valor de CONT
    DELAY_MS(500);               // Retardo de 500 milisegundos
    CONT++;                      // Incrementa la variable CONT
    DIRECCION++;                // Incrementa la variable DIRECCION
    PORTB=CONT;                 // Lo que hay en CONT llévelo al puerto B
  }
  IF(BIT_TEST(PORTC,2)&&(CONT<15)) // Si RC2=1 y la variable CONT es menor que
15
  {
    WRITE_EEPROM(DIRECCION,CONT2); // Escribir en eeprom el valor de CONT2
    DELAY_MS(500);               // Retardo de 500 milisegundos
    CONT2=CONT2+2;               // Incrementa en dos la variable CONT2
    CONT++;                      // Incrementa la variable CONT
    DIRECCION++;                // Incrementa la variable DIRECCION
    PORTB=CONT2;                // Lo que hay en CONT2 llévelo al puerto B
  }
  IF(BIT_TEST(PORTC,1)&&(CONT<15)) // Si RC1=1 y la variable CONT es menor que
15
  {
    CONT=0;                      // Poner la variable CONT en cero
    DIRECCION=0;                  // Poner la variable DIRECCION en cero
    WHILE(CONT<15)               // Mientras que CONT sea menor a 15
    {
      PORTB=READ_EEPROM(DIRECCION); // Leer el valor de la EEPROM
      DELAY_MS(500);              // Retardo de 500 milisegundos
      CONT++;                     // Incrementa la variable CONT
      DIRECCION++;                // Incrementa la variable DIRECCION
    }
  }
}
}
}

```

# 18

## ALMACENAMIENTO EN MEMORIA EEPROM EXTERNA

- Como se tiene limitaciones para almacenar información en la memoria eeprom interna, hay memorias eeprom seriales externas con diferente capacidad que permiten almacenar mayor información. En el compilador hay drivers que permiten manejar diferentes memorias eeprom externas seriales entre ellos la 2402.

La conexión de la memoria eeprom externa es la siguiente:

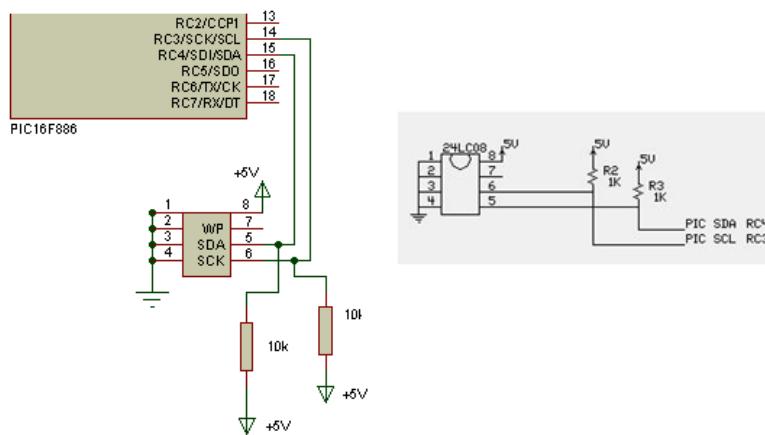


Figura 18.1 Conexión memoria eeprom externa

Los pasos que se deben seguir para almacenar datos en memoria eeprom externa son:

1. En el encabezado del programa incluir el driver para manejar la memoria eeprom externa (en el ejemplo se trabaja con la 24LC02)  
#INCLUDE<2402.C>
2. En el programa principal inicializar la memoria  
INIT\_EXT\_EEPROM();
3. Para escribir en la memoria se utiliza la instrucción  
WRITE\_EXT\_EEPROM(Dirección, Valor)  
**Dirección:** esta limitada por la capacidad de la memoria eeprom externa.  
**Valor:** es un byte.  
Esta función puede durar varios milisegundos.  
Para leer el contenido de la memoria eeprom externa se utiliza la siguiente instrucción:  
K=READ\_EXT\_EEPROM(Dirección)

---

### Ejemplo 18.1 Memoria eeprom externa

Con un teclado telefónico, un LCD y una memoria eeprom externa digitar 4 números y luego recuperarlos, en el LCD mostrar el valor digitado y el valor recuperado.

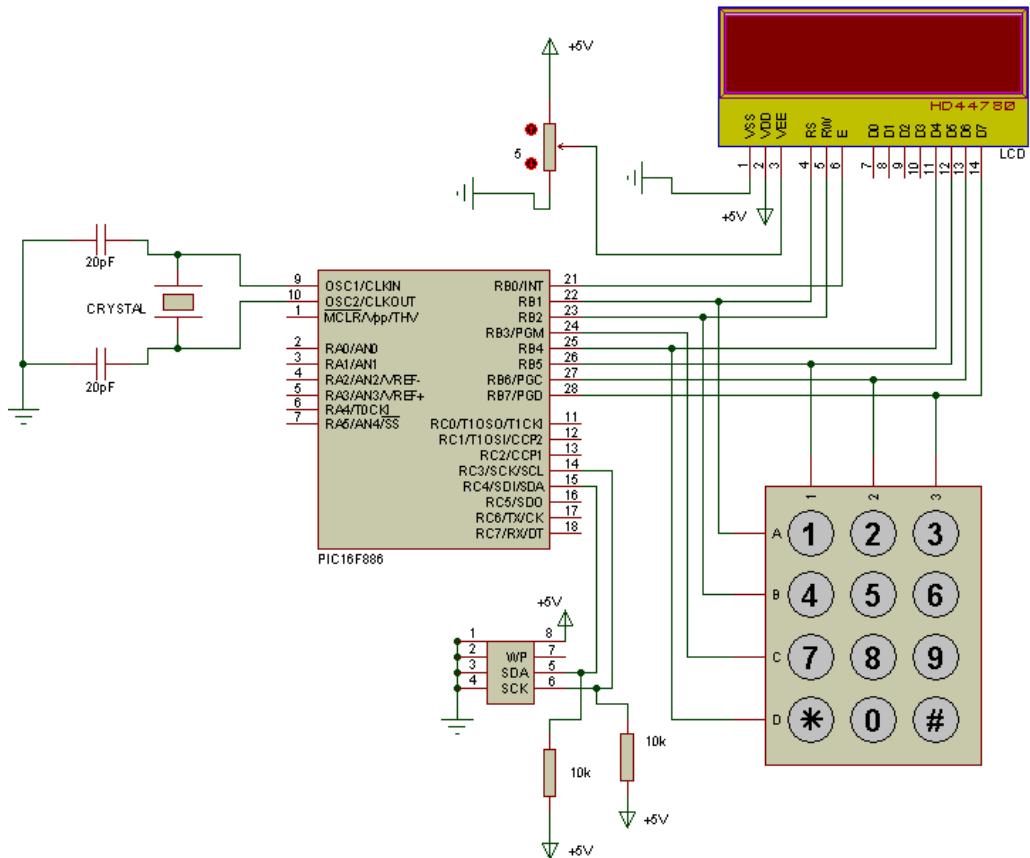


Figura 18.2 Conexión ejemplo 14

```

#include<16F886.H>
#USE DELAY(CLOCK=4000000)
#define USE_PORTB_KBD
#FUSES XT,NOPROTECT,NOWDT,NOBROWNOUT,PUT,NOLVP,WRT
#define USE_PORTB_LCD TRUE
#include <LCD.C>
#include <KBD.C>
#include <2402.C>
INT K; // Defino la variable K como carácter
BYTE J=16,DIR;

VOID MAIN()
{
    PORT_B_PULLUPS(TRUE); // Activar las resistencias a positivo de
                           // Puerto B
    LCD_INIT();
    KBD_INIT();
    INIT_EXT_EEPROM();

    FOR (J=12; J>=2; --J)
    {
        LCD_GOTOXY(J,1);
        (LCD_PUTC("DIGITADO:    "));
        DELAY_MS(100); // Retardo de 100 milisegundos
                        // LCD_PUTC("\F");

    }

    LCD_GOTOXY(12,1); // Ubicarse en la columna 12 fila 1
    DIR=0; // Poner la variable DIR en cero
    WHILE (DIR<4) // Mientras que DIR sea menor que 4

```

```

{
    K=0;                                // Poner la variable K en cero
    K=KBD_GETC();
    IF( (K!="0"))
    {
        LCD_PUTC(K);
        WRITE_EXT_EEPROM(DIR,K);
        DIR++;
    }
}
DELAY_MS(500);                         // Retardo de 500 milisegundos
FOR (J=12; J>=2; --J)
{
    LCD_GOTOXY(J,2);                  // Ubicarse en la fila 2 y la columna varía
    (LCD_PUTC("RECUPERADO:   "));
    DELAY_MS(100);                   // Retardo de 100 milisegundos
}
LCD_GOTOXY(13,2);                     // Ubicarse en la columna 13 fila 2
DIR=0;                                 // Poner la variable DIR en cero
WHILE (DIR<4)                         // Mientras que DIR sea menor a 4
{
    K=READ_EXT_EEPROM(DIR);
    LCD_PUTC(K);
    DIR++;                            // Incrementa la variable DIR
}
}

```

# 19

## CONVERSOR ANÁLOGO/DIGITAL (A/D)

- Los microcontroladores PIC16F886 poseen un conversor análogo/digital de 10 bits y 13 canales de entrada.

Una señal análoga es una señal continua, por ejemplo una señal de 0 a 5V es una señal análoga y puede tomar valores (1V, 2V, 2.5 V, etc.).

Una señal digital solo puede tomar dos valores 0V ó 5V, abierto o cerrado, activado o desactivado.

Un sensor de nivel que me genere una señal análoga no solo me indica si el tanque está lleno o vacío, sino que además me indica que nivel tiene el tanque en todo momento y la señal será proporcional al nivel del tanque.

Un conversor A/D me convierte la señal análoga en un número digital (binario), éste número es proporcional a la señal análoga.

En el caso del microcontrolador PIC16F873 el conversor A/D tiene 10 bits y la señal análoga de entrada puede estar entre 0V y 5V, sin embargo el conversor A/D tiene dos niveles de referencia  $V_{REF+}$  y  $V_{REF-}$  que me indican entre qué valores será la señal análoga de entrada. El voltaje mínimo diferencial es de 2V, es decir la diferencia entre  $V_{REF+}$  y  $V_{REF-}$  no puede ser mayor a 2V.

- Con 10 bits el mayor número binario que se puede tener es 1024, por lo tanto la resolución del conversor A/D está dada por la fórmula:

$$resolución = \frac{(V_{REF} - GND)}{1024}$$

Así, por ejemplo, si  $V_{REF} = +5V$  y  $V_{REF-}$  es 0V, la resolución es  $5V/1024 = 4.8mV$ , cuando la señal análoga sea 0V le corresponde un número digital = 0000000000 y para 5V será 1111111111.

Si  $V_{REF+} = +5V$  y  $V_{REF-}$  se puede determinar en todo momento a qué número digital aproximado corresponde cualquier señal análoga de entrada, con la fórmula:

$$\frac{V_{Entrada}}{4.8mV} = \frac{V_{Entrada}}{0.0048V}$$

Por ejemplo si la señal análoga es 2V, el número digital aproximado, es:

$$2V = \frac{416}{0.0048}$$

La tensión de referencia  $V_{REF+}$  puede implementarse con la tensión interna de alimentación  $V_{DD}$ , o bien, con una externa que se pone en la pata RA2/AN2/  $V_{REF-}$ .

### PASOS PARA TRABAJAR CON EL CONVERSOR A/D

- En el encabezado del programa incluir la siguiente línea, si se va a trabajar el conversor A/D a 10 bits ya que por defecto funciona a 8 bits.

#DEVICE ADC=10

- En el programa principal

- Configurar las entradas análogas.
- Seleccionar el tipo de reloj del conversor A/D.
- Especificar el canal a utilizar para la conversión.

- SETUP\_ADC\_PORTS(Valor);

Esta función configura los pines del ADC para que sean entradas análogas, digitales o alguna combinación de ambos. Las combinaciones permitidas varían de acuerdo al microcontrolador.

Las constantes (ALL\_ANALOG) todas las entradas como análogas y (NO\_ANALOG) ninguna entrada como análoga son válidas para todos los microcontroladores.

b. SETUP\_ADC (Modo)

Selecciona el tipo de reloj del conversor A/D

Modo puede ser:

ADC\_CLOCK\_DIV\_2

ADC\_CLOCK\_DIV\_8

ADC\_CLOCK\_DIV\_32

ADC\_CLOCK\_INTERNAL

c. SET\_ADC\_CHANNEL (Canal)

Especifica el canal a utilizar por la función

READ\_ADC()

3. Leer el valor de la conversión

I=READ\_ADC ()

Esta función lee el valor digital del conversor análogo a digital.

**Ejemplo 19.1 Conversor análogo – digital.**

Mostrar en el LCD el valor numérico correspondiente a la señal análoga que entra por RA0.

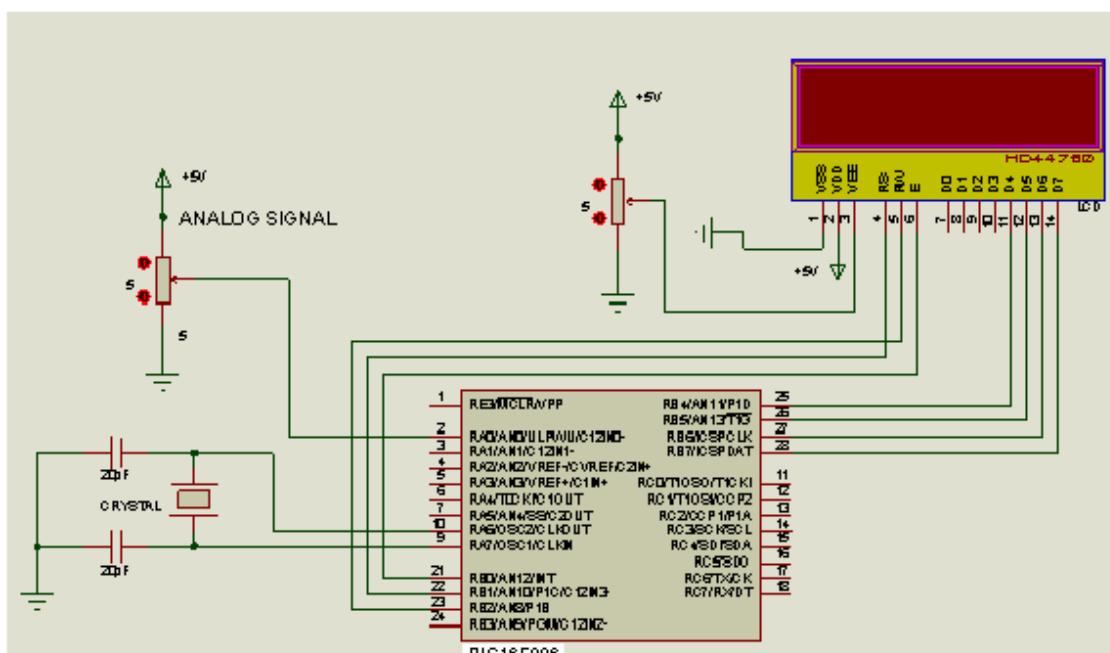


Figura 19.1 Conexión ejemplo

```
#INCLUDE <16F886.H>
#DEVICE ADC=10 // Define que el conversor trabaja a 10 bits
#USE DELAY(CLOCK=4000000)
#FUSES XT,NOPROTECT,NOWDT,NOBROWNOUT,PUT,NOLVP
#BYTE PORTA=5
#define USE_PORTB_LCD TRUE
#include <LCD.C>
LONG VOLTAJE; // Definir la variable voltaje como una
// Variable tipo long
```

```

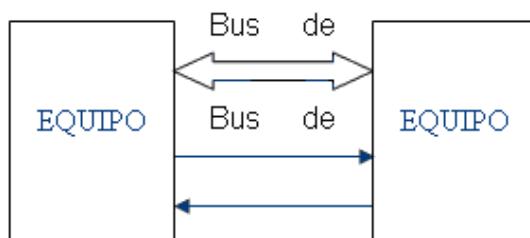
VOID MAIN()
{
    SET_TRIS_A(0B11111111); // Configura todo el puerto a como ENTRADA
    SETUP_ADC_PORTS(SAN0); // Define todo el puerto a como
                           // Entradas análogas
    SETUP_ADC(ADC_CLOCK_INTERNAL); // Define que el conversor trabaje
                                   // Con el reloj interno
    LCD_INIT(); // Inicializa Idc

    WHILE(TRUE)
    {
        SET_ADC_CHANNEL(0); // Selecciona el canal 0 (ra0)
        DELAY_MS(1); // Llama un retardo de 1 milisegundo
        VOLTAJE=READ_ADC(); // Almacena en voltaje el valor
                           // De la conversión
        LCD_PUTC("\F");
        LCD_GOTOXY(1,1);
        LCD_PUTC("CONVERSOR A/D");
        LCD_GOTOXY(1,2);
        PRINTF(LCD_PUTC,"VALOR %LU",VOLTAJE); // Muestra el valor numérico de la
                                              // Conversión.
        DELAY_MS(200);
    }
}

```

- Existen dos formas comunes de transferir información binaria entre dos equipos, la comunicación paralela y la comunicación serial.

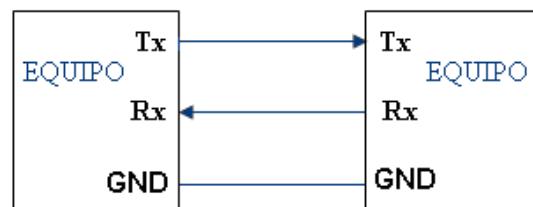
En la comunicación en forma paralela los datos se transfieren en forma simultánea y existen algunas líneas adicionales de control que permite la comunicación entre los dos equipos.



Una de las desventajas de la comunicación paralela es la cantidad de líneas que necesita, esto aumenta los costos y más cuando la distancia entre los equipos es grande.

La comunicación serial sólo utiliza tres líneas, una para recibir los datos Rx, otra para trasmisir los datos Tx y la línea común GND.

Cada dato se transmite bit a bit, un bit a la vez, por lo tanto se hace mucho más lenta, pero tiene la ventaja de necesitar menos líneas y las distancias a las cuales se puede transferir la información son mayores, además con el uso de los módem se puede trasmisir a cualquier parte del mundo.



Existen dos formas de comunicación serial:

- Sincrónica
- Asincrónica

## 20.1 COMUNICACIÓN SINCRÓNICA:

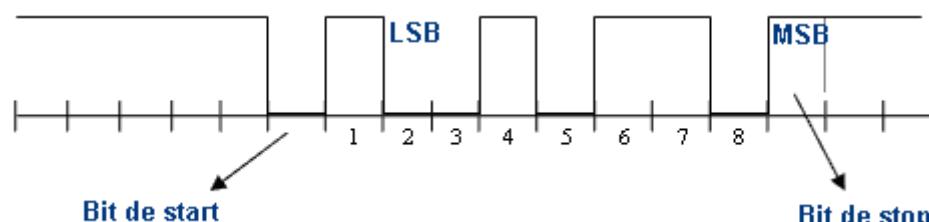
En esta comunicación además de una línea sobre la que se transfieren los datos, se necesita otra que contenga pulsos de reloj que indiquen que el dato es válido; la duración del bit está determinada por la duración del pulso de sincronismo.

## 20.2 COMUNICACIÓN ASINCRÓNICA

En esta comunicación los pulsos de reloj no son necesarios y se utilizan otros mecanismos para realizar la transferencia de datos. La duración de cada bit está determinada por la velocidad con la cual se realiza la trasferencia de datos, por ejemplo si se transmite a 1200 bits por segundo (baudios), la duración de cada bit es de 833 microsegundos.

Las velocidades de transmisión más comunes son 300, 600, 1200, 2400, 9600, 14400 y 28800 baudios.

En este curso solo se estudia la comunicación asincrónica.

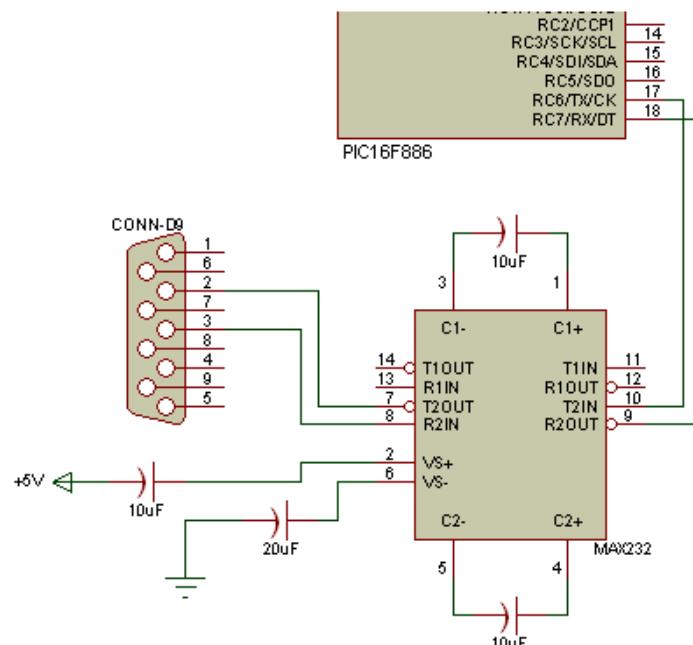


En la figura se muestra la forma como se transmite un dato cuando se realiza alguna transferencia, la línea del transistor permanece en alto. Para empezar a transmitir datos esta línea se pone en bajo durante un bit, lo cual se conoce como bit de Start, y luego comienza a transmitir los bits correspondientes al dato, empezando por el bit menos significativo (LSB) y terminando con el más significativo (MSB). Al finalizar se agrega el bit de paridad, si está activada esta opción, y por último los bits de stop, que pueden ser 1 o 2, en los cuales la línea regresa a nivel alto.

En el ejemplo de la figura, después del bit de start se transmite el dato 01101001 y al final hay un bit de stop.

Esto significa que la configuración de la transmisión serial es: 1 bit start, 8 bits dato, no paridad y 1 bit de stop.

El receptor no está sincronizado con el transistor y no sabe cuando va a recibir datos. La transición de alto a bajo de la línea del transmisor, activa el receptor y este genera un conteo de tiempo de tal manera que realiza una lectura de la línea medio bit después del evento; si la lectura realizada es un estado alto, asume que la transición ocurrida fue ocasionada por ruido en la línea; si por el contrario la lectura es un estado bajo, considera como válida la transición y empieza a realizar lecturas secuenciales a intervalos de un bit hasta conformar el dato transmitido. Lógicamente tanto el transmisor como el receptor deberán tener los mismos parámetros de configuración (velocidad, número bits del dato, paridad y bits de parada)



**Figura 20.1 Conexión con el puerto serial**

Pasos para trabajar con comunicación serial con el microcontrolador:

1. En el encabezado del programa incluir la directiva:  
#USE RS232(BAUD=BAUDIOS, XMIT=PIN, RCV=PIN)  
Baud: Velocidad en baudios (bits por segundo).  
Xmit: Bit que transmite (Tx)  
RCV: Bit que recibe (Rx)
2. En el programa principal enviar o recibir un carácter.

Para recibir un carácter se usa la instrucción.

```
C=getc(); // Esta instrucción espera un carácter por el pin RCV del
//puerto RS232 y retorna el carácter recibido.
```

Hacen lo mismo las instrucciones GETCH() y GETCHAR()

Para enviar un carácter se usa la instrucción

```
PUTC() //Esta instrucción envía un carácter a la patica XMIT del //dispositivo
RS232.
```

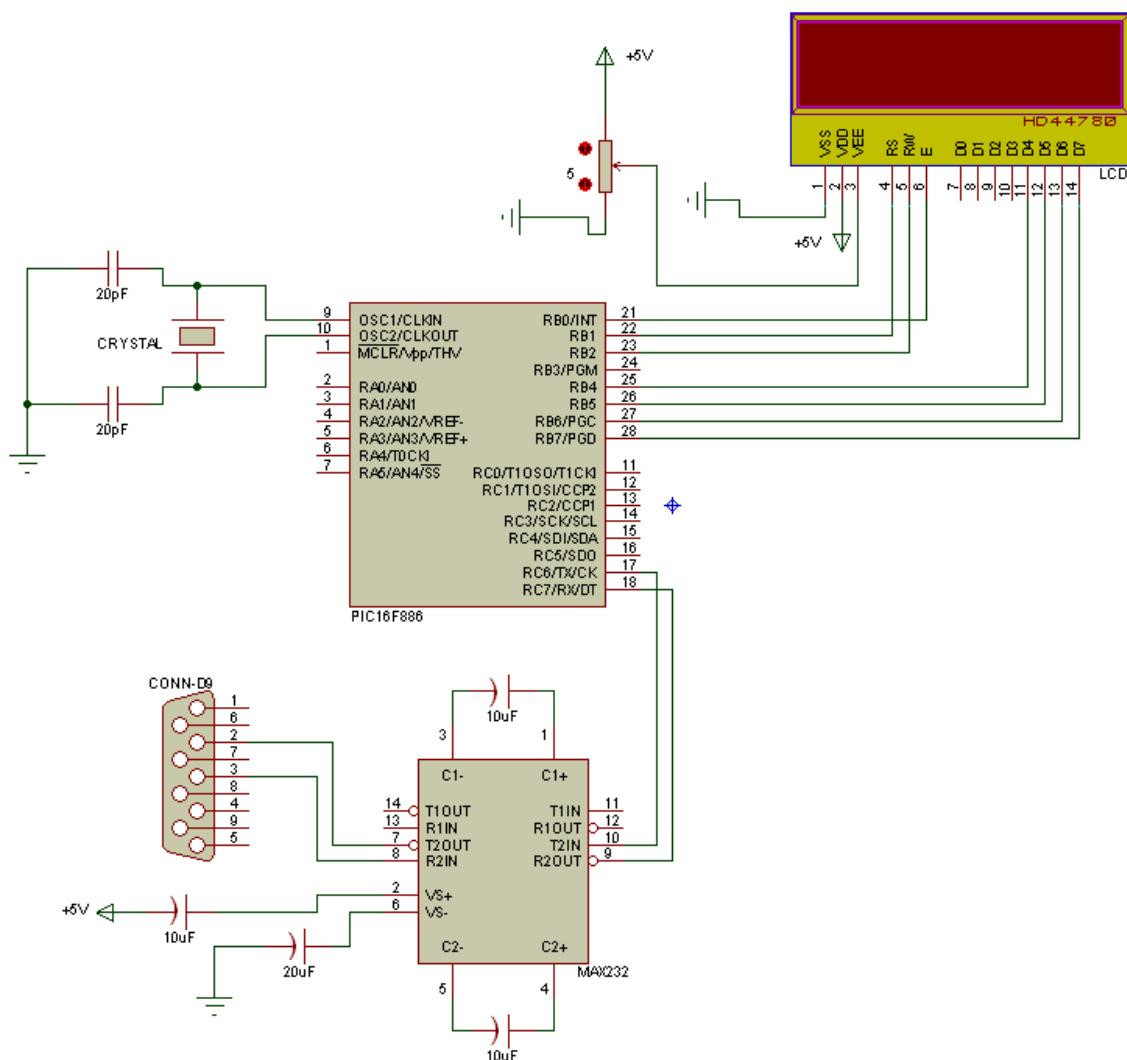
Hace lo mismo la instrucción PUTCHAR()

**NOTA 1:** Entre los paréntesis va el carácter a enviar.

**NOTA 2:** Si se quiere enviar el carácter a "R" se hace de la siguiente manera **PUTC('R');**

### Ejemplo 20.1 Comunicación serial.

Mostrar en el LCD el carácter digitado en el computador.



**Figura 20.2 Conexión ejemplo**

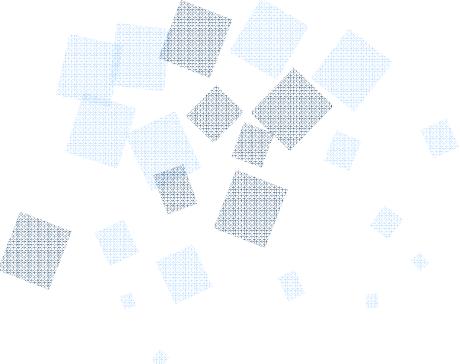
```

#define _XTAL_FREQ 4000000
#include <16F886.H>
#use delay(clock=4000000)
#use rs232(baud=9600, xmit=pin_c6, rcv=pin_c7)
#define use_portb_lcd true
#include <lcd.c>
#fuses xt,noprotect,nowdt,nobrownout,put,nolvp,wrt
#byte portb=6
#byte portc=7

void main()
{
    char k;                                // Definir la variable K como una variable
                                            // Tipo caracter

    lcd_init();
    lcd_putc("\digite el texto?\n");
    while (true)
    {
        k=getc();                           // Guardar en K el valor capturado del
                                            // Puerto serial
        lcd_putc(k);                      // Mostrar K en el LCD
    }
}

```



## BIBLIOGRAFÍA

- ANGULO USATEQUI, José María. Microcontroladores PIC: Diseño Práctico de aplicaciones. Segunda Edición. Editorial Mc Graw-Hill. 1999.
- ANGULO USATEQUI, José María. Microcontroladores PIC: Diseño Práctico de aplicaciones. Segunda Parte. Editorial Mc Graw-Hill. 2000.
- FLOYD, T.L. Fundamentos de Sistemas Digitales. Sexta Edición. Editorial Prince Hall. 1997
- PIC C [en línea]. Reno, NV (USA): SCM International, Inc [citado en 2008-10-01]. Disponible en Internet: <<http://scmstore.com/micros/PICC/>>

# ANEXOS

A

## PIC C COMPILER

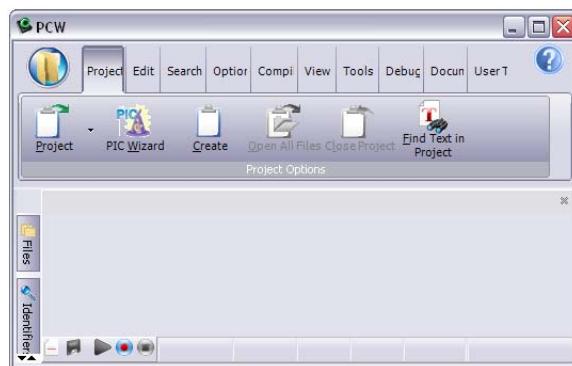
El **PicC C Compiler** fue desarrollado para cumplir con las especificaciones del lenguaje ANSI C. El compilador produce tres tipos de archivos. Archivos con extensión **.hex** que le permitirá grabar el programa ejecutable en el PIC por medio del uso de un programador. El archivo con extensión **.asm** contendrá un listado en assembler del programa compilado con la información del mapeo de memoria. Estos archivos son muy útiles para el debugging de los programas y para determinar la cantidad de pasos de programas (ciclos de ejecución) tiene la aplicación. Los archivos con extensión **.pre** contienen la información preprocesada del programa, #defines, #includes, etc. La cual es expandida y guardada en el archivo.

Es el producto ideal para aquellas personas que le gusta desarrollar en bajo nivel con los recursos de un lenguaje de alto nivel como el C. Se recomienda ser utilizado por personas vinculadas con la programación y sintaxis de C.

### Beneficios

- Esta basado en el ANSI C.
- Soporte completo de la familia de microcontroladores PIC de 14 bit.
- Salida Assembly.
- Industry standard Intel Hex 8 bit Merged format (INHX8M).
- Soporta interrupciones.
- Tipos de datos 8 y 16 bit - int, char, long, pointers, unsigned, etc.
- Inserción de código asamblea - asm( );
- Todos los operadores aritméticos - incluyendo multiplicación, división, modulo y otros.
- Las variables y funciones no utilizadas son borradas automáticamente.
- Reutilización de ram.
- Windows 95 y Windows 3.1 compatible.
- Instrucciones simples en castellano.
- Dispositivos soportados: 16F84, 16C83, 16C554, 16C556, 16C558, 16C61, 16C62, 16C620, 16C621, 16C622, 16C63, 16C64, 16C641, 16C642, 16C65, 16C66, 16C661, 16C662, 16C67, 16C71, 16C710, 16C711, 16C715, 16C72, 16C73, 16C74, 16C76, 16C77, 16C9xx, 14C000, 16CE623, 16CE624, 16CE625, 12C671, 12C672, 12C673, 12C674, 16F873, 16F874, 16F876, 16F877
- Notas de aplicaciones

El primer pantallazo que muestra el programa Pic C Compiler, al abrirse es el siguiente:



**Figura A.1 Primer pantallazo de Pic C Compiler.**



Los iconos de guardar, abrir, nuevo, funcionan igual que en archivo de Microsoft, se encuentran en el menú del PIC C

Los iconos nuevos y más utilizados son:



Save all files: Guarda todos los archivos.



Close file: Cierra solo un archivo.



Close all file: Cierra todos los archivos.

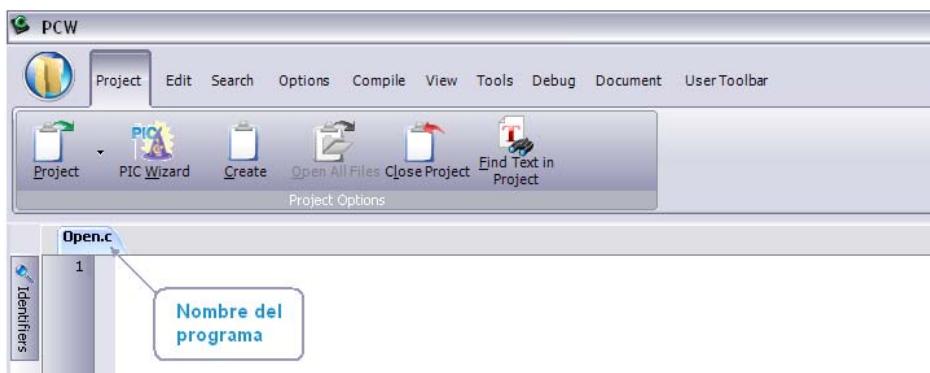


Compile: Compila el programa.

---

#### Ejemplo A.1 Pic C compiler

1. File – New.
2. Se guarda el archivo
3. Sale una hoja con el nombre que se le puso



**Figura A.2 Hoja en blanco Pic C Compiler.**

4. Se escribe el programa

5. Se compila: Para esto se presiona el ícono “Compile” en el menú “Complile”



Cuando termina la compilación este genera un cuadro donde muestra: el nombre del archivo, los errores, la memoria que se utilizó, etc.

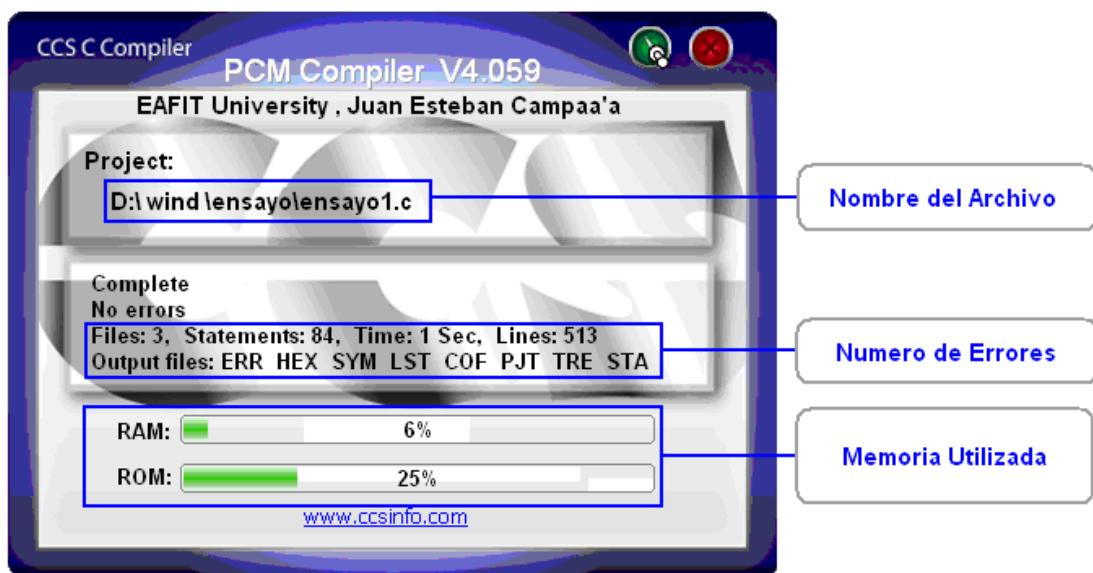


Figura A.3 Compilar Pic C Compiler.

6. Si sale un error en la programación, el programa no muestra el cuadro anterior sino que saca un aviso en rojo que dice el tipo de problema y en donde se encuentra el error (este lo muestra subrayando la palabra posterior).  
En este ejemplo lo que faltó fue el corchete que cierra la instrucción anterior.

```

18 port_b_pullups(TRUE);
19 LCD_INIT();
20 set_tris_a(0B00000001);
21 set_tris_b(1);
22 set_tris_c(0);
23 setup_adc_ports(RAO_ANALOG);

```

Figura A.4 Errores en Pic C Compiler.

Nota: Los errores más comunes son: corchetes, punto y coma, palabras mal escritas, etc.

7. Al compilar se debe tener mucho cuidado con el archivo que se esta compilando, porque cuando hay varios abiertos el programa no sabe cual compilar.

---

#### Ejemplo A.2 Pic C Compiler.

Se realizó otro ejercicio, llamado “ejercicio1” y se compila. Sin embargo, si no se tiene en cuenta lo anterior, el programa esta compilando otro ejercicio.

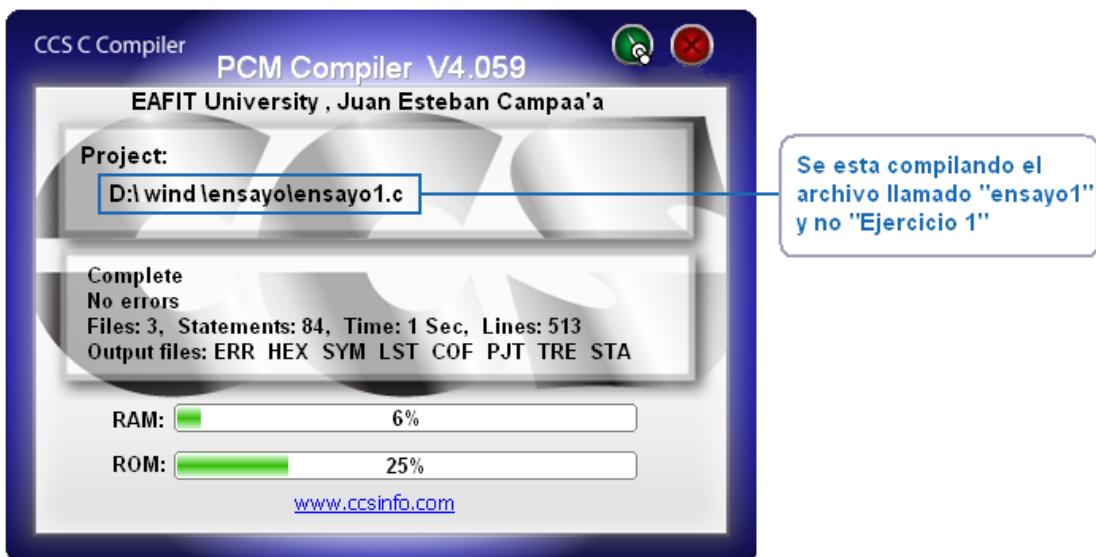


Figura A.5 Compilar en Pic C Compiler.

Nota: Para que se compile el que debe ser, se debe cerrar los otros archivos.

- Para saber que archivo es el que se quiere cerrar o el que se esta utilizando, sólo se debe observar que pestaña es la que sobresale.

Nota: La que sobresale es el archivo que esta activo.

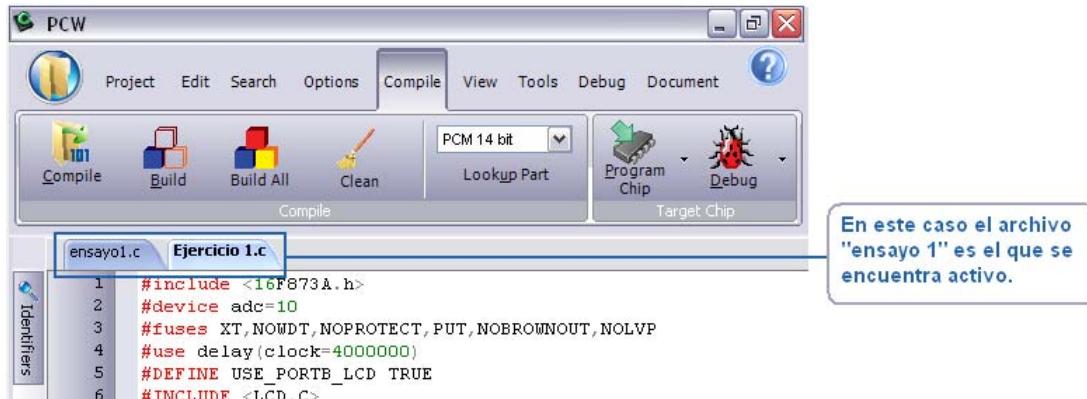
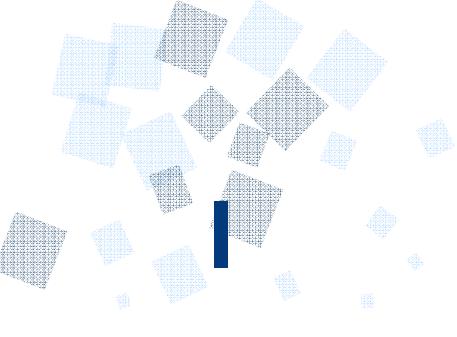


Figura A.6 Archivo activo en Pic C Compiler.

- 9. Si hay varios archivos abiertos y se quiere cerrar todos, se utiliza el icono Close all file.
- 10. Si se quiere cerrar solo un archivo, se usa el icono Close file.
- 11. Si se quiere guardar todos los archivos abiertos se utiliza el icono Save all files



# CONTENIDO

INTRODUCCIÓN	1
1 CONCEPTOS BÁSICOS	2
1.1 SISTEMAS DE NUMERACIÓN DECIMAL (BASE 10)	2
1.2 SISTEMA DE NUMERACION BINARIO (BASE 2)	2
1.3 SISTEMA HEXADECIMAL (BASE 16)	3
1.4 CONVERSIÓN DE BINARIO A DECIMAL	3
1.5 CONVERSIÓN DE DECIMAL A BINARIO	3
1.6 CONVERSIÓN DE HEXADECIMAL A DECIMAL	3
1.7 CONVERSIÓN DE BINARIO A HEXADECIMAL	4
1.8 CONVERSIÓN DE HEXADECIMAL A BINARIO	4
1.9 DECIMAL CODIFICADO EN BINARIO: (BCD)	4
1.10 REPRESENTACIÓN DE LOS NÚMEROS DE 0 A 15	4
2 MEMORIAS	5
2.1 MEMORIA RAM	5
2.2 MEMORIA ROM	5
2.3 MEMORIA EPROM	5
2.4 MEMORIA EEPROM	5
2.5 MEMORIA FLASH	5
3 INTRODUCCIÓN AL MICROCONTROLADOR	6
3.1 ORGANIZACIÓN DE LA MEMORIA DE DATOS RAM	6
3.2 DESCRIPCION DE LOS PINES	6
3.3 CONFIGURACION DE LOS PUERTOS	8
4 LENGUAJE DE PROGRAMACION EN C	10
4.1 ESTRUCTURAS DE CONTROL EN C	10
La estructura de control condicional if	10
Cláusula else	11
Selección múltiple con la sentencia switch	11
4.2 ESTRUCTURAS DE CONTROL REPETITIVAS	12
Bucle while	12
Bucle for	12
Equivalencia entre For y While	13
Bucles infinitos	13
5 OPERADORES	14
5.1 OPERADORES ARITMETICOS	14
5.2 OPERADORES RELACIONES	14
5.3 OPERADORES LÓGICOS	14
5.4 OPERADORES DE INCREMENTO Y DECREMENTO	14
6 ENCABEZADO DE UN PROGRAMA	15
7 INSTRUCCIONES BASICAS	16
8 INSTRUCCIONES DE ROTACION	21
9 MOTORES PASO A PASO	23
9.1 CONTROL DE MOTOR PASO A PASO UNIPOLAR	23
10 MANEJO DE DISPLAY 7 SEGMENTOS Y ANTIREBOTE	27
11 TRABAJOS CON PULSADORES (ANTIRREBOTE)	29
12 MULTIPLEXAJE DE DISPLAY	32
13 INTERRUPCIONES	36
14 TIMER	38
15 MANEJO DEL TECLADO TELEFONICO	42

16	MANEJO DEL LCD (DISPLAY DE CRISTAL LIQUIDO)	47
16.1	EL LCD Y LA CONFIGURACIÓN DE LA PANTALLA	47
17	ALMACENAMIENTO EN MEMORIA EEPROM INTERNA	55
18	ALMACENAMIENTO EN MEMORIA EEPROM EXTERNA	57
19	CONVERSOR ANALOGO/DIGITAL (A/D).	60
20	COMUNICACIÓN SERIAL	63
20.1	COMUNICACIÓN SINCRÓNICA:	63
20.2	COMUNICACIÓN ASINCRÓNICA	63
	BIBLIOGRAFÍA	67
	ANEXOS	68

# Microcontroladores

## Guia PicC

Para establecer comunicación con nosotros puede hacerlo al correo electrónico [hmurillo@eafit.edu.co](mailto:hmurillo@eafit.edu.co)

Elaborado por:

Hugo Alberto Murillo Hoyos

Ana Cristina Tamayo

Diseño de portada e interior:

Santiago García Ochoa