Capítulo VI

Rutinas de programación

- ***La suma***
- ***La resta***
- ***La rotación***
- ***La multiplicación***
- ***La división***
- ***Conversión binario a BCD***
- ***Suma BCD***
- ***Conexión de displays al PIC***

Las rutinas son uno de los recursos más valiosos cuando se trabaja en programación; ellas permiten que los programas sean más simples, debido a que el programa principal se disminuye cuando algunas tareas se escriben en forma de programas pequeños e independientes.

En programación, una rutina se define como un procedimiento independiente (un bloque), que realiza una labor específica y a la cual se puede llamar desde cualquier parte del programa principal. Dado que cada rutina realiza una labor en particular, como encender un LED o establecer una base de tiempo de un segundo, por ejemplo, el programador, cuando está seguro de su funcionamiento, puede mantenerla almacenada y disponible en un «banco» o «librería», para utilizarla en cualquier programa, sin volver a escribir las líneas que realicen lo deseado: sólo necesitará copiar el bloque de la rutina o tener el archivo disponible para cuando se realice la compilación del programa. Posteriormente, deberá hacer el llamado al procedimiento adecuado, en el instante preciso.

Para la elaboración de cualquier rutina, es necesario conocer más a fondo las instrucciones que la forman y cuales son las condiciones o estados que se modifican ante cada una de ellas. Los estados, banderas o *flags* son *bits* que se afectan por algunas operaciones lógicas o aritméticas y la consulta de estos se utiliza para tomar decisiones dentro del programa. Debemos tener presente que en el microcontrolador PIC 16C84, los estados se encuentran localizados en el registro 03, llamado *STATUS*. Los primeros tres *bits* de este registro (0 al 2) son los estados sobre los cuales debemos ejercer una vigilancia, para determinar así los siguientes pasos a ejecutar. El bit menos significativo (bit 0) es el correspondiente al estado del *Carry* o acarreo, el siguiente (*bit* 1) corresponde al DC o acarreo de dígito y el tercero de ellos (*bit* 2) es el correspondiente al estado Z o cero.

Cuando escribimos el programa en un editor de textos, para posteriormente utilizar el ensamblador , debemos recordar que, en el formato de las instrucciones primero se escribe el registro fuente y a continuación el registro destino de la operación que se está realizando. Se llama *registro fuente* aquel registro que se utiliza como origen de la información y *registro destino* aquel en el cual se almacena el resultado de la operación. Si deseamos que el destino sea el registro W, podemos optar por escribir en la parte correspondiente, W ó 0; si deseamos que se haga en el mismo registro fuente, podemos escribir 1 u omitirlo. Por ejemplo, las siguientes instrucciones son equivalentes:

comf	10,w	equivale a	comf	10,0
xorwf	15	equivale a	xorwf	15,1

La suma

Esta operación se implementa en los microcontroladores PIC por una sola instrucción: ADDWF; en la figura 1 se observan sus detalles. Lo más importante para considerar en todos los casos son los estados afectados ya que, como lo habíamos

mencionado, ellos son los indicadores que se deben utilizar para tomar decisiones en el programa.

Instrucción:	ADICION	
Sintaxis:	ADDWF	f,d
	Cod. Op.	Dest. Fuente
Codificación:	000111 d	fffff
Operación:	(f) + (W) - d	
Estados afectados:	C, DC, Z	
Descripción:	El contenido del registro W se agrega al contenido del registro f. Si 'd' es 0 el resultado se almacena en el registro W; si 'd' es 1, se hará en el registro f.	

Figura 1. Detalles de la suma en los microcontroladores PIC

Uno de los aspectos que conviene recalcar en esta instrucción, es que el estado del *carry* no es considerado para los resultados de la operación; se afecta, pero no interviene. Consideremos los ejemplos que se muestran en la figura 2, y observemos la forma en la cual se afectan los estados y el registro destino en cada una de las instrucciones.

	Antes de la instrucción		Instrucción	Después de la instrucción				
	Fuente	W		Fuente	W	Z	DC	C
1	1001 0011	0101 1000	ADDWF Fuente,**W**	1001 0011	**1110 1011**0	0	0	0
2	1010 1011	0010 0101	ADDWF Fuente	**1101 0000**	0010 0101	0	1	0
3	1010 1011	0110 1100	ADDWF Fuente,**0**	1010 1011	**0001 0111**0	1	1	1
4	1100 1101	0011 0011	ADDWF Fuente,**1**	**0000 0000**	0011 00111	1	1	1

Figura 2. Ejemplos de la suma

En estos ejemplos, las cantidades se expresan en binario, para visualizar más fácilmente las operaciones bit a bit. Los resultados, ubicados en el destino de la operación, se han resaltado para comprender mejor la forma en que las instrucciones manejan este parámetro.

En las operaciones el *carry* (C) se coloca a 1 cuando la suma supera el valor más alto que puede contener un byte (0FF hex o 225 decimal); en caso contrario permanece en 0, indicando que no se presentó sobreflujo o acarreo. El acarreo de dígito (DC) se activa cuando la suma de los *nibbles* (partes inferiores o 4 bits menos significativos de los *bytes*) supera el valor que este puede contener (0F hex o 15 decimal). El estado Z (*Zero* o cero) se activa cuando la operación da como resultado 0 en el registro destino, como se muestra en el ejemplo 4.

La resta

De manera similar a la suma, esta operación se implementa con una sola instrucción: SUBWF; en la Figura 3 se observan sus detalles más importantes.

Instrucción:	RESTA		
Sintaxis:	SUBWF	f,d	
	Cód. Op.	Dest.	Fuente
Codificación:	000010	d	fffff
Operación:	(f) - (W) - d		
Estados afectados:	C, DC, Z		
Descripción:	El contenido del registro W se resta del contenido del registro f. Si 'd' es 0 el resultado se almacena en el registro W; si 'd' es 1, se hará en el registro f.		

Figura 3. Detalles de la resta en los microcontroladores PIC

Consideremos los ejemplos de la figura 4 y observemos la forma en la cual se afectan los estados y el registro destino ante cada una de las instrucciones. De nuevo, las cantidades están expresadas en binario y los resultados, ubicados en el destino de la operación, se han resaltado para visualizar cómo las instrucciones manejan este parámetro. Como sucedía con la suma, el *carry* se afecta por la operación, pero no interviene en ella.

	Antes de la instrucción		Instrucción	Después de la instrucción				
	Fuente	W		Fuente	W	Z	DC	C
1	1011 1001	0100 0010	SUBWF Fuente,**W**	1011 1001	**0111 0111**	0	1	1
2	1011 1001	0100 1100	SUBWF Fuente	**0110 1101**	0100 1100	0	0	1
3	1011 1001	1110 0011	SUBWF Fuente,**0**	1011 1001	**1101 0110**	0	1	0
4	1011 1001	1011 1001	SUBWF Fuente,**1**	**0000 0000**	1011 1001	1	1	1

Figura 4. Ejemplos de la resta

En las sustracciones, el *carry* o acarreo (C) se coloca en 1 cuando el valor absoluto del registro de trabajo W (sustraendo) es menor que el valor absoluto del registro fuente (minuendo); por lo tanto, es un indicador que el resultado es un número positivo. Si por el contrario, el *carry* resultante es 0, indica que el resultado es un número negativo, ya que el minuendo era menor que el sustraendo.

El acarreo de dígito (DC), mientras tanto, se coloca en 1 cuando el valor absoluto del *nibble* menos significativo del registro de trabajo W es menor que el *nibble* correspondiente del registro fuente y se pondrá en 0 en caso contrario. Este estado es muy útil en rutinas de conversión de números binarios a números BCD, como se verá más adelante.

Por último, el estado Z sólo se activará cuando ambas cantidades sean iguales y el resultado de la operación, por lo tanto, sea cero, siendo éste el valor que se almacena en el registro destino.

La rotación

En los microcontroladores PIC se encuentran dos instrucciones de rotación: una hacia la izquierda y otra hacia la derecha; ambas incluyen el *carry* dentro de la operación. Ya que la diferencia entre las dos rotaciones está sólo en el sentido en que se realiza, sólo mostraremos los detalles de una de ellas, figura 5.

Instrucción:	ROTACION A LA IZQUIERDA		
Sintaxis:	RLF	f,d	
	Cód. Op.	Dest.	Fuente
Codificación:	001101	d	fffff
Operación:	f<n> - d<n+1>; f<7>-C, C - d<0>;		
Estados afectados:	C		
Descripción:	El contenido del registro ‘f’ se rota un bit hacia la izquierda a través del carry. Si ‘d’ es 0, el resultado se almacena en el registro W; si ‘d’ es 1, se hará en el registro f.		

Figura 5. Detalles del desplazamiento a la izquierda

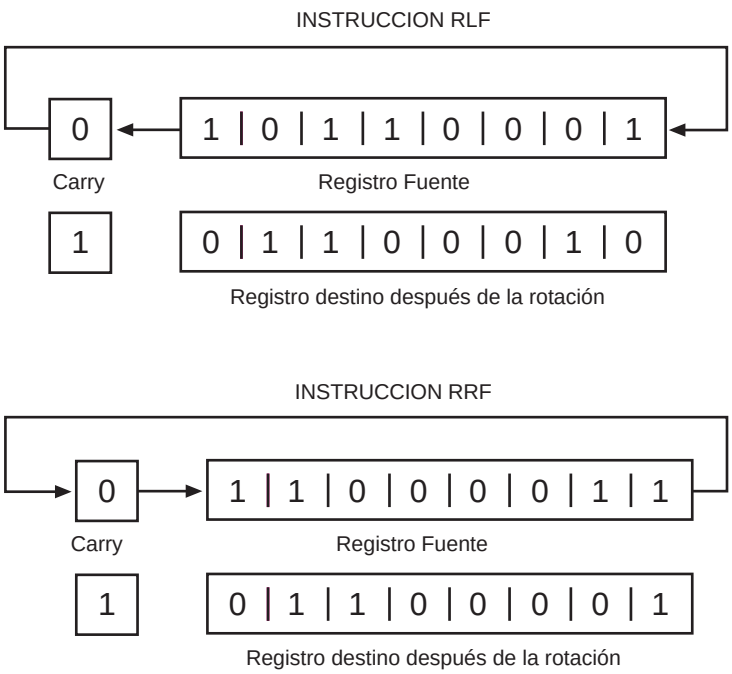

Figura 6. Rotaciones

En la figura 6, se observa como operan estas instrucciones, en donde el *carry* participa activamente en cada una de ellas. Un dato interesante, es que una rotación a la derecha puede considerarse como una división entre dos, mientras que una hacia la izquierda es una multiplicación por dos. Cuando es necesario realizar operaciones de este tipo, con potencias de dos, se pueden implementar fácil y rápidamente con rotaciones hacia uno u otro lado.

Las cuatro instrucciones anteriores son la base para una gran cantidad de rutinas, entre las que se incluyen multiplicaciones, divisiones, conversiones de diverso tipo, etc, de las cuales vamos a desarrollar algunas. Al final de cada uno de las rutinas se incorpora un pequeño programa de prueba, el cual llama la rutina respectiva, para que ésta sea ejecutada.

La multiplicación

Esta rutina se desarrolla con base en las instrucciones anteriormente descritas; aquí se implementa la multiplicación de dos cantidades, de un *byte* cada una. El diagrama de flujo correspondiente se muestra en la figura 7 y el programa respectivo en la figura 8. Los comentarios hechos en el programa informan sobre cada uno de los pasos desarrollados.

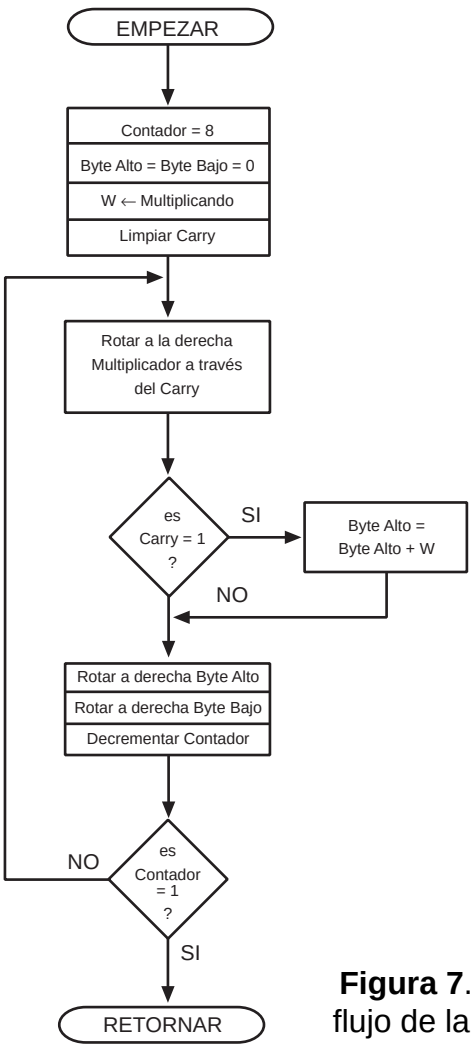

Figura 7. Diagrama de flujo de la multiplicación

```
;=============================================================
;                    Multiplicación 8 x 8
;     El resultado (de 16 bits) es guardado en 2 bytes
;_____________________________________________________________

STATUS    equ       03              ;Registro de estados
CARRY     equ       00              ;Bandera de acarreo
mulcnd    equ       19              ;Contiene Multiplicando
mulplr    equ       10              ;Contiene Multiplicador
H_byte    equ       12              ;Byte alto del resultado
L_byte    equ       13              ;Byte bajo del resultado
count     equ       14              ;Contador del loop

          org       00
          goto      main
Multip    clrf      H_byte          ;Limpiar byte alto
          clrf      L_byte          ;Limpiar byte bajo
          movlw     8               ;Cargar el
          movwf     count           ;contador
          movf      mulcnd,w        ;Multiplicando va a W
          bcf       STATUS,CARRY    ;limpiar carry
loop      rrf       mulplr          ;Rotar a la derecha
          btfsc     STATUS,CARRY    ;Probar estado del carry
          addwf     H_byte          ;Si uno, agregar multi-
                                    ;plicador al byte alto
          rrf       H_byte          ;Rotar derecha byte alto
          rrf       L_byte          ;Rotar derecha byte bajo
          decfsz    count           ;Decrementa contador
                                    ;y verifica si es cero
          goto      loop            ;Repite si aún no cero
          retlw     0               ;Retorna de la rutina

;_____________________________________________________________
;             Programa de Prueba de la rutina
;_____________________________________________________________
main      movlw     0BA             ;
          movwf     mulplr          ;
          movlw     078             ;
          movwf     mulcnd

          call      Multip          ;

queda     goto      queda           ;En byte alto queda 57H
                                    ;y en byte bajo 30H

          END
```

Figura 8. Rutina de multiplicación

La división

Esta es una operación ampliamente utilizada en los microcontroladores y los microprocesadores; permite la conversión de bases numéricas de diverso tipo. El diagrama de flujo se muestra en la figura 9 y el programa respectivo en la figura 10.

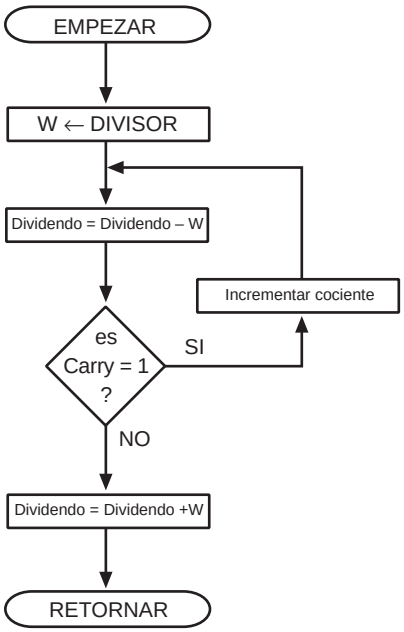

Figura 9. Diagrama de flujo de la división

Para lograr la división, esta rutina utiliza una técnica de restas sucesivas, donde el sustraendo se coloca en el registro de trabajo W. Después de cada resta, la rutina debe consultar el estado del *carry*. Como se indicó anteriormente, si el acarreo es 1, indica que el resultado es un número positivo, (el minuendo era mayor que el sustraendo), la resta era válida y, por lo tanto, el cociente puede ser incrementado; si por el contrario, el *carry* es cero, indica que el resultado es un número negativo (el minuendo no contenía el sustraendo). En el segundo caso se debe realizar una suma adicional, con el fin de restaurar el valor que dio origen al numero negativo en el residuo.

Conversión de un número binario en su equivalente BCD

Son numerosas las oportunidades en las que es necesario convertir un valor binario, proveniente de algún proceso, en un valor tal que, al sacarse por uno de los puertos, pueda ser fácilmente visualizado e interpretado por el usuario del sistema. En este caso, la rutina que se presenta realiza la conversión de un número binario de 8 bits, en su equivalente BCD (Decimal Codificado en Binario). En la figura 11 se muestra el diagrama de flujo correspondiente y en la figura 12 el programa.

Esta rutina emplea las restas sucesivas descritas en la rutina anterior. Como el valor más alto contenido en un *byte* es el decimal 255, se realizan inicialmente restas de 100 unidades y, posteriormente, restas de 10. En este proceso se consigue el número de veces que fue posible restar 100, cuyo resultado son las centenas y el número de veces que fue posible restar 10, obteniéndose las decenas. El residuo de este proceso son las unidades.

```
;=============================================================
;***       DIVIDE UN NUMERO DE UN BYTE  ENTRE OTRO    ***
; Divide un número binario existente en la posición
; Divndo, entre otro localizado en posición Divsor.
; El cociente se almacena en la misma posición de memoria
; RAM del divisor y el residuo se guarda en
; la del dividendo.
;-------------------------------------------------------------


STATUS   equ     3              ;Registro de estados
CARRY    equ     0              ;Bit de acarreo
Divndo   equ     18             ;Contiene número a dividir
Divsor   equ     19             ;Contiene divisor
Cocinte  equ     19             ;Guarda el resultado
Resduo   equ     18             ;Se almacena el residuo
         org     00
         goto    main
Div      movf    Divsor,w
         clrf    Cocinte
Repite   subwf   Resduo         ;Restarle el divisor
         btfss   STATUS,CARRY   ;Verificar el carry
         goto    SUM1           ;Si cero, dejar de restar
         incf    Cocinte        ;Si uno, incrementa cociente
         goto    Repite         ;y repite resta divisor
SUM1     addwf   Resduo         ;Sumarle divisor al residuo
         RETLW   0              ; Regresa con 0`


;-------------------------------------------------------------
;                PROGRAMA DE PRUEBA
;-------------------------------------------------------------


main     movlw   .254           ;Cargar
         movwf   Divndo         ;el dividendo
         movlw   .10            ;Cargar
         movwf   Divsor         ;el divisor
         call    Div            ;Llamar la subrutina
                                ;El cociente es 19hex
                                ;y el residuo 4hex
queda    goto    queda          ;Quedar en un loop infinito

         END
;=============================================================
```

Figura 10. Rutina de división

A la rutina que efectúa la conversión es posible introducirle cambios, logrando un código diferente. La modificación que se presenta en la figura 13, aparentemente más desventajosa por ser de mayor longitud y utilizar un registro adicional, tiene la rutina de división por separado, la cual se llama desde la rutina de conversión y puede ser invocada por otras.

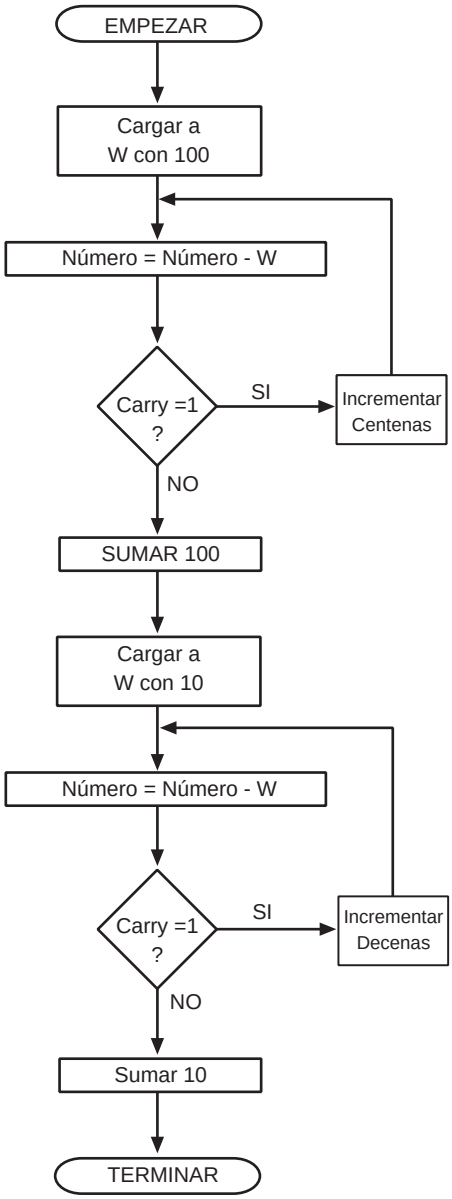

Figura 11. Diagrama de flujo de la conversión binario a BCD

Debemos recordar en este punto que el microcontrolador PIC16C84 permite llamar cuantas rutinas se posean, pero únicamente pueden estar anidadas ocho de ellas, ya que la pila sólo puede sostener ocho direcciones de retorno. El anidamiento de una novena rutina ocasionará la pérdida sobre el control del programa y los resultados serán impredecibles.

Suma de dos números BCD

Para realizar la suma de dos números BCD se muestra el diagrama de flujo en la figura 14 y el programa respectivo en la figura 15. La rutina suma las cantidades contenidas en dos posiciones de la memoria RAM del microcontrolador y coloca el resultado en una de ellas; como el resultado de la operación puede tener tres dígitos BCD, el tercero de ellos se almacena en la otra posición, aprovechándose el valor de retorno de la rutina.

```
;=============== CONVIERTE UN NUMERO BINARIO A BCD ===========
; La rutina sólo requiere que se coloque el valor
; a convertir en un registro, denominado aquí Unidad.
; Ella retorna las decenas y las centenas (dígitos
; decimales) en un par de registros adicionales.
;_____________________________________________________________

STATUS   equ      3              ;Registro de estados
CARRY    equ      0              ;Bit de acarreo
DC       equ      1              ;Bit de acarreo de dígito
Unidad   equ      18             ;Para almacenar unidades
Decena   equ      19             ;Para almacenar decenas
Centena  equ      1A             ;Registro para centenas
AUX      equ      1B             ;Registro auxiliar

         org      00
         goto     main
Bin_BCD  clrf     Decena         ;Limpiar decenas
         clrf     Centena        ;Limpiar centenas
         movlw    .100           ;Cargar a W con decimal 100

Otra     subwf    Unidad         ;Restarle 100 al valor
         btfss    STATUS,CARRY   ;verificar estado del carry
         goto     SUM            ;Si 0, dejar de restar 100
         incf     Centena        ;Si 1, incrementar centenas
         goto     Otra           ;Repetir resta

SUM      addwf    Unidad         ;Sumarle 100
         movlw    .10            ;cargar a W con 10
Repite   subwf    Unidad         ;Restarle al valor digital 10
         btfss    STATUS,CARRY   ;Verificar estado del carry
         goto     SUM1           ;Si cero, dejar de restar 10
         incf     Decena         ;Si uno, incrementa decenas
         goto     Repite         ;y repite la resta de 10

SUM1     addwf    Unidad         ;Sumarle 10 al valor digital
         RETLW    0              ;Regresa con 0

;_____________________________________________________________
;               Programa de Prueba
;_____________________________________________________________
main     movlw    .254           ;Cargar valor a convertir
         movwf    Unidad         ;el registro Unidad

         call     Bin_BCD        ;Llamar la subrutina
                                 ;Centenas = 2
                                 ;Decenas = 5
                                 ;Unidades = 4

queda    goto     queda          ;Quedar en un loop infinito

         END
```

Figura 12. Rutina de conversión binario a BCD

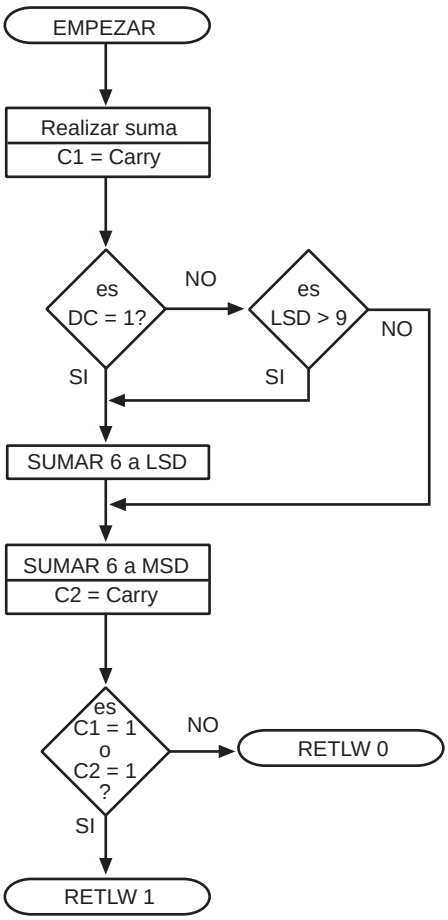

Figura 13. Diagrama de flujo de la suma BCD

Conexión de displays al PIC

En este ejercicio observaremos algunas de las opciones que se presentan al momento de decidir que configuración utilizar para conectar algunos elementos indispensables, como son los visualizadores de información o *displays*. También veremos con detalle algunas instrucciones y uno de los aspectos más simples e interesantes de los PIC, como es el manejo de las tablas, las cuales nos pueden ayudar a agilizar numerosos procesos, como la conversión de códigos y lecturas de teclado, por ejemplo.

Para cumplir nuestros objetivos, haremos un pequeño ejercicio, que consiste en leer un teclado de cuatro elementos y enviar el dato correspondiente a un *display* de siete segmentos; el diagrama de flujo respectivo se muestra en la figura 15.

Aunque en nuestro caso hemos optado por obtener el valor de la lectura de un teclado de cuatro elementos, el valor binario que se pretende mostrar puede provenir de fuentes diferentes, tal como un conteo de tiempo o de eventos externos, un proceso de conversión análogo a digital, etc.

```
;====================== Suma BCD sin signo ==================
; Se asume que los números a sumar están en las
; posiciones Sum1 y Sum2. El resultado de la
; suma se almacena en la posición de Sum2 y el carry,
; si existe, regresa en el registro de trabajo W,
; aprovechándose el valor de retorno de la rutina
;------------------------------------------------------------
STATUS     equ      3
CARRY      equ      0
DC         equ      1
Sum1       equ      18             ;Sumando 1 y bit alto
Acarreo    equ      18
Sum2       equ      19             ;Sumando 2 y resultado
           org      00
           goto     main
SumaBCD    movf     Sum1,w         ;Colocar valor de Sum1 en W
           addwf    Sum2           ;realizar suma binaria
           clrf     Sum1           ;limpiar contenido de Sum1
           rlf      Sum1           ;rotar por si hubo carry
           btfsc    STATUS,DC      ;Es DC = 0 ?
           goto     AdLSD          ;si 1, ajustar LSD
           movlw    6              ;Ajuste binario-BCD para LSD
           addwf    Sum2           ;Probar para LSD > 9
           btfss    STATUS,DC      ;verificar Carry medio
           subwf    Sum2           ;LSD < 9, retornar valor
           bcf      STATUS,CARRY   ;Limpiar carry
           goto     AdMSD          ;Ir a probar MSD

AdLSD      movlw    6              ;Ajuste de binario a BCD
           addwf    Sum2           ;para dígito (LSD)

AdMSD      movlw    60             ;Ajuste Binario-BCD para MSD
           addwf    Sum2           ;Sumar al resultado
           btfsc    STATUS,CARRY   ;Consultar carry
           retlw    1              ;si 1, retornar con 1
           btfsc    Sum1,0         ;si 0, probar carry inicio
           retlw    1              ;si hubo, retornar con 1
           subwf    Sum2           ;MSD < 9, retorna valor
           RETLW    0              ;Regresa con 0
;------------------------------------------------------------
;               Programa de Prueba
;------------------------------------------------------------
main       movlw    99
           movwf    Sum1           ;Sum1 = 99 (máximo número BCD)
           movlw    99
           movwf    Sum2           ;Hacer Sum2 = 99
           call     SumaBCDA       ;Después de suma, Sum2 = 98
           movwf    Sum1           ;y Sum1 = 01 (99+99 = 198)

queda      goto     queda          ;Se queda en un loop infinito

           END
```

Figura 14. Rutina de suma BCD

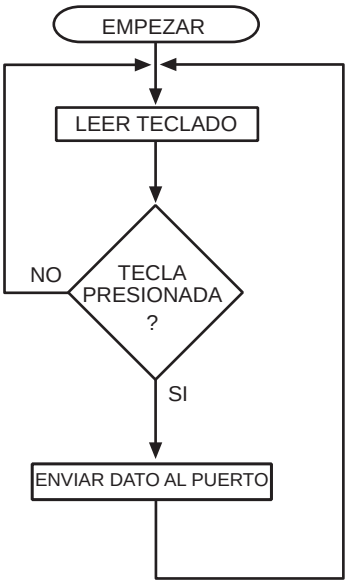

Figura 15. Diagrama de flujo del ejercicio

Nuestro teclado se encuentra conectado a las cuatro líneas del puerto A del microcontrolador, como se puede observar en las figuras 19 y 20. Para obtener el estado de las teclas, bastará con leer el puerto correspondiente y realizar un pequeño tratamiento al dato leído. En esta configuración, para leer el teclado se realiza la siguiente instrucción:

```
movf        Puerto_A,w
```

Por lo regular, a la lectura se le realiza algún tipo de tratamiento, para determinar si se encuentra pulsada o no una tecla. Para nuestro caso, utilizamos una de las instrucciones más útiles e interesantes: la operación XOR.

La operación OR exclusiva

Recordemos que la función XOR es una operación lógica que toma uno a uno los *bits* de dos cantidades y genera los siguientes resultados: si los dos *bits* son diferentes (uno y cero o viceversa) el *bit* resultante será uno; si los dos *bits* son iguales (ambos cero o ambos uno) el *bit* resultante será cero, como puede observarse en la figura 16.

B	A	A .XOR. B
0	0	0
0	1	1
1	0	1
1	1	0

Figura 16. Tabla de la verdad de la operación lógica XOR

Un elemento curioso e importante con la operación XOR es el siguiente: si a una valor se le realiza esta operación dos veces consecutivas con un número cualquiera, vuelve a presentarse la misma cantidad, como puede observarse en los siguientes ejemplos :

Ej. 1	1010 0111 (0A7h)	**Ej. 2**	1101 0001 (0D1h)
	0111 0110 (76h)		0111 0110 (76h)
XOR	1101 0001 (0D1h)		1010 0111 (0A7h)

A la cantidad original (el hexadecimal 0A7h) se le realiza inicialmente la operación XOR con el código deseado (en este caso 76h) y al resultado (0D1h) se le realiza de nuevo la operación XOR con el mismo código, dando como resultado el valor original (el hexadecimal 0A7h); el lector puede intentar realizar esta operación con cualquier número y observará que esto siempre se cumple.

Esta característica se utiliza en sistemas en los cuales se necesita codificar o encriptar una información, de tal manera que solamente el usuario original tenga acceso a ella, ocultándola a los intrusos. La condición para encriptar la información es realizarle la operación XOR (o la XNOR) con el código de nuestro interés; la decodificación posterior exige realizarle de nuevo la operación en cuestión con el código correspondiente. Este código puede tener la longitud que desee el usuario.

En esta oportunidad, la operación XOR la utilizaremos con el propósito de comparar dos cantidades, ya que cuando en los PICs resultan ser iguales, el operando destino será cero; por lo tanto, esta condición activa (coloca en uno) la bandera de Cero (*Zero Flag*), que corresponde al *bit* 2 del registro STATUS o de estados.

Existen dos versiones para esta instrucción en la familia de los microcontroladores PIC: XORWF y XORLW. En la Figura 17 se observan los detalles de las dos instrucciones; ambas utilizan el registro de trabajo W y son muy convenientes para consultar el estado de un registro, averiguando el momento en el cual se obtiene un determinado valor. Consideremos los ejemplos para la instrucción XORWF que se muestran en la figura 2: cuando los dos valores resultan ser iguales, el registro destino efectivamente se hace ‘cero’ y la bandera Z se coloca en ‘uno’.

La instrucción XORLW actúa de manera similar, con la salvedad que no actúa sobre dos registros, sino entre el registro de trabajo W y una constante de 8 *bits* que reside en la memoria de programa; el destino siempre será el registro de trabajo W.

	Antes de la instrucción		**Instrucción**	**Después de la instrucción**		
	Fuente	W		Fuente	W	Z
1	1001 0011	1001 0011	XORWF Fuente,**W**	1001 0011	**0000 0000**	1
2	1010 1011	1010 1011	XORWF **Fuente**	**0000 0000**	1010 1011	1
3	1010 1001	0110 1100	XORWF Fuente,**0**	1010 1001	**1100 0101**	0
4	1101 1101	0001 0011	XORWF Fuente,**1**	**1100 1110**	0001 0011	0

Figura 17. Ejemplo de la instrucción XOR

En nuestro caso, la instrucción XOR nos servirá para leer el teclado, ya que haremos la operación OR Exclusiva con el número 0FF. Como se puede observar en las figuras 19 y 20, al no presionar ninguna tecla la lectura que se obtiene del puerto es el número binario '1111'b, ya que las teclas normalmente están colocadas a un nivel alto a través de resistencias de 10KΩ. Cuando una tecla se presiona, se conectará a masa y la lectura que se realice en ese momento será un 0 lógico por el *bit* respectivo. Por lo tanto, realizada la lectura del teclado y después de guardar el valor en un registro o posición de la memoria RAM del microcontrolador, podemos realizar una operación XOR entre el valor leído y 0FF. Si el resultado de la operación es cero, significa que ambas cantidades eran iguales (ninguna de las teclas se presionó), la bandera de cero se activará y este es el indicador que verificamos para determinar el siguiente paso a seguir, como se puede observar en el diagrama de flujo del programa, figura 15.

Instrucción:	OR Exclusiva
Sintaxis:	XORWF f,d
	Cod. Op. dest Fuente
Codificación:	000110 d fffff
Operación:	(W .XOR. f) Æ d
Estados afectados:	Z
Descripción:	Se realiza la operación OR Exclusiva entre el registro de trabajo W y el registro fuente. Si «d» es cero, el resultado se almacena en el registro de trabajo W; si «d» es uno, el resultado se almacena en el registro fuente.
Instrucción:	OR Exclusiva
Sintaxis:	XORLW k
	Cod. Op. Fuente
Codificación:	1111 kkkkkkkk
Operación:	(W .XOR. k) Æ W
Estados afectados:	Z
Descripción:	Al contenido del registro W se le reali za la operación OR Exclusiva con el literal k de 8 bits. El resultado se almacena en el registro de trabajo W.

Figura 18. Descripción de las instrucciones OR exclusiva

A cada tecla se le ha asignado un peso o valor relativo, de tal manera que la primera de ellas tiene un valor de 1 y cada una de las otras el doble del valor de la anterior; por lo tanto, los valores de las teclas son 1, 2, 4 y 8. El programa se realizará de tal forma que ante la presión simultánea de varias teclas, la suma de ellas se muestre en el *display*; por tal motivo, éste último podrá mostrar valores comprendidos entre 0 y F hexadecimal.

Una vez que se detecta la pulsación de una tecla, y que el valor correspondiente se encuentre en uno de los registros del microcontrolador, debemos buscar la forma de visualizar este valor a través de uno de los elementos más populares y útiles: el *display* de siete segmentos, el cual nos permitirá generar todos los números y algunos caracteres especiales y alfabéticos.

Aquí necesitamos detenernos y pensar en cual es la configuración electrónica (el diagrama esquemático) que necesitamos. En este caso, podemos tener dos alternativas para mostrar los dígitos en el *display*: o lo hacemos directamente, manejando cada uno de los siete segmentos con las líneas del microcontrolador, como se muestra en la figura 19, o bien, lo hacemos a través de un decodificador, como el 7447 o el 7448, figura 20.

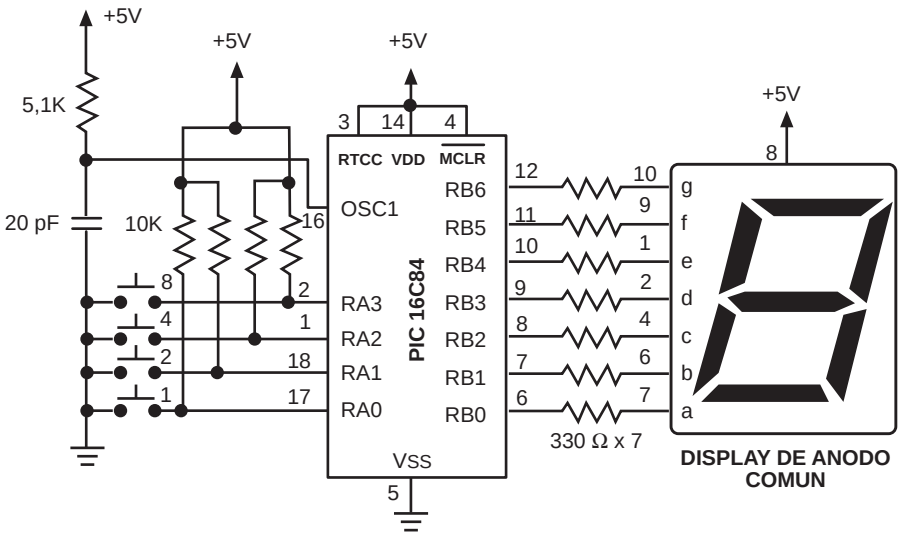

Figura 19. Control de display directo desde el microcontrolador

Cada una de estas opciones tiene sus pros y sus contras. La primera opción tiene la ventaja de que se puede manejar directamente el *display*, sin necesidad de elementos adicionales, ahorrando espacio y materiales que necesariamente significan dinero. Si pensamos en la segunda opción, la del decodificador, presenta la ventaja de que solamente se están dedicando cuatro de las líneas para mostrar los dígitos; representando un ahorro de tres líneas sobre la primera opción, las cuales pueden resultar valiosas en un diseño en el cual precisemos de muchos dispositivos de Entrada/Salida y las líneas se podrán dedicar a otras funciones dentro del circuito.

Al momento de sacar la información por el puerto, los dos tratamientos han de ser diferentes. Para el control por medio del decodificador, bastará con tomar el valor binario que se encuentra en el registro de interés y enviarlo a través del puerto respectivo, sin necesidad de realizar ningún tratamiento adicional, siempre y cuando el control se esté realizando con la parte baja del puerto; el decodificador externo se encargará de realizar la conversión. Las instrucciones necesarias serían algo como lo siguiente:

```
movf    Valor,w      ;cargar valor a sacar por el puerto
movwf   Puerto_B     ;enviarlo al puerto
```

Para el control directo de los *displays*, se precisa un tratamiento diferente y debemos establecer nuestro propio decodificador por programa. En este caso, debemos tomar el valor binario del registro de interés y acudir a una tabla que contenga los códigos correspondientes para manejar cada uno de los LED's del *display* de siete segmentos. Las instrucciones respectivas serían algo como lo siguiente:

```
movf    Valor,w     ;cargar valor a sacar por el puerto
call    tabla       ;llamar tabla de datos
movwf   Puerto_B    ;enviarlo al puerto visualizarlo
```

Las tablas se manejan como si se trataran de rutinas. La clave para el tratamiento de tablas está en colocar un valor índice en el registro de trabajo W y hacer el llamado a la dirección base de la tabla. Dentro de la tabla debe existir una primera instrucción que realiza una suma sobre el contador del programa, lográndose un salto relativo a éste. La característica de la suma ya la habíamos descrito en el número anterior. Debemos recordar que el contador de programa se trata como un registro cualquiera, sobre el cual es posible realizar operaciones lógicas y aritméticas.

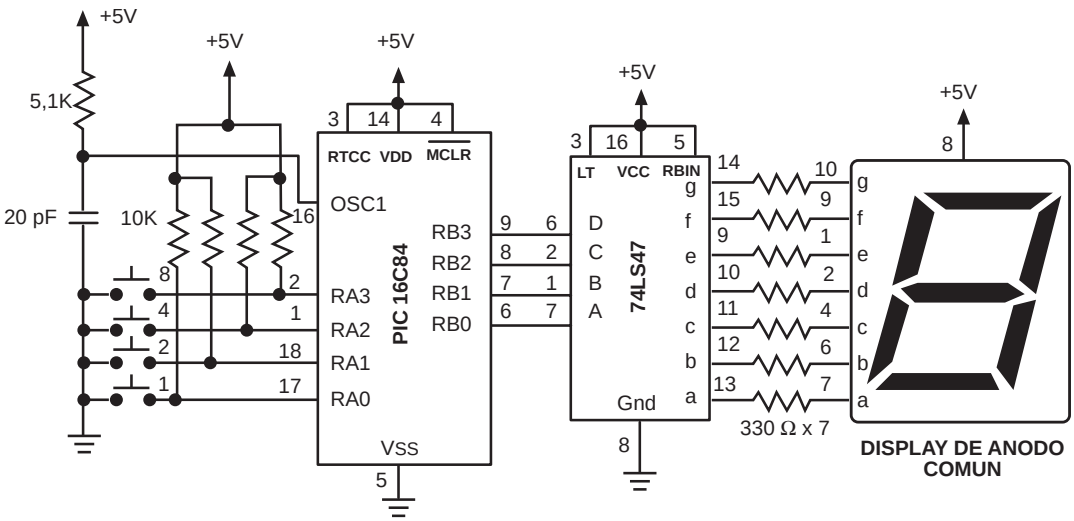

Figura 20. Control del display usando un decodificador

En el lugar en el cual el programa salte deberá incluir alguna instrucción que le diga al microcontrolador cual es el siguiente paso a realizar; ya se trate de un salto hacia un sitio en el cual se hace el tratamiento al valor índice, o bien una instrucción de retorno que obliga al registro de trabajo a regresar con el código correspondiente. En nuestro caso, hemos empleado la segunda opción, como lo podemos observar en las siguientes líneas:

```
tabla   addwf pc            ;PC = PC + W
        retlw b'11000000'   ;código para el 0
        retlw b'11111001'   ;código para el 1
        retlw b'10100100'   ;código para el 2
        retlw b'10110000'   ;código para el 3
        retlw b'10011001'   ;código para el 4
        retlw b'10010010'   ;código para el 5
        retlw b'10000010'   ;código para el 6
        retlw b'11111000'   ;código para el 7
        retlw b'10000000'   ;código para el 8
        retlw b'10011000'   ;código para el 9
```

La única precaución, la cual es realmente importante y debe tenerse siempre presente, es que cuando se hace el llamado a la tabla, el registro W no debe contener un valor mayor o igual al número de elementos que contiene la tabla. Esta tabla, por ejemplo posee 10 elementos y si el registro W llega con el valor 11, se presentará una pérdida sobre el control del programa. Como se puede deducir, una tabla con estas características puede llegar a contener hasta 256 elementos, por ser W un registro de 8 *bits*.

En la figura 21 se muestra el programa correspondiente al caso en el cual se está manejando el *display* directamente. Para el caso en el cual se posee el decodificador, se deben eliminar las líneas que se encuentran dentro de los bloques grises. El nuevo programa así obtenido omite los llamados a la tabla de datos y procede a tomar directamente el dato contenido en el registro de interés y sacarlo al puerto correspondiente. Debe tener presente que los decodificadores 7447 y 7448 han sido diseñados para mostrar cantidades decimales y por lo tanto números superiores a 9 se visualizarán de manera diferente. Ambos programas aprovechan el cerrojo de las salidas (*latch*) para dejar un dato sobre el puerto, cambiándose únicamente cuando el dato leído es diferente al último registrado.

Las rutinas descritas en este capítulo son de mucha utilidad, se pueden emplear en cualquier programa sin mayores cambios. Por esto, es recomendable que se guarden en formato texto en el disco duro del computador, para ir formando un banco o librería de rutinas, con aquellas que sean de mayor uso, y tenerlas disponibles para cualquier proyecto que se realice.

```
;=============================================================
; La tabla establece en términos de UNOS Y CEROS*
; el dato que se va a sacar por el display    *
; La tabla se considera para lógica negativa   *
; (Display de ánodo común)            *
;_____________________________________________________________
cero       equ 2
PC         equ 2
Status     equ 3
Puerto_A   equ 5
Puerto_B   equ 6
dato       equ 18

           org 00                ;
           goto inicio           ;

tabla      addwf pc              ;PC = PC + W
           ;datos: .gfedcba      ;orden de los segmentos
           retlw b'11000000'     ;código para el 0
           retlw b'11111001'     ;código para el 1
           retlw b'10100100'     ;código para el 2
           retlw b'10110000'     ;código para el 3
           retlw b'10011001'     ;código para el 4
           retlw b'10010010'     ;código para el 5
```

```
            retlw b'10000010'    ;código para el 6
            retlw b'11111000'    ;código para el 7
            retlw b'10000000'    ;código para el 8
            retlw b'10011000'    ;código para el 9
            retlw b'10001000'    ;código para la A
            retlw b'10000011'    ;código para la b
            retlw b'11000110'    ;código para la c
            retlw b'10100001'    ;código para la d
            retlw b'10000110'    ;código para la E
            retlw b'10001110'    ;código para la F

;_____________________________________________________________
;*          ESTE ES EL PROGRAMA EN SI              *
;_____________________________________________________________
inicio      movlw 00             ;configurar puerto B como
            tris puerto_B        ;salidas
            call tabla           ;llama rutina
            movwf puerto_B       ;para mostrar '0' en display
            movlw 0ff            ;configura
            tris puerto_A        ;puerto A como entradas
leer        movf puerto_A,w      ;lee estado de las teclas
            movwf dato           ;guardar dato
            xorlw 0              ;consultar si tecla
            btfsc Status,cero    ;presionada
            goto leer            ;si no tecla, vuelve y lee
            movf dato,w          ;si hubo, recupera valor
            call tabla           ;llama a rutina
            movwf puerto_B       ;Envía dato a LEDs
            goto leer            ;y vuelve a leer
            end
```

Figura 21. Programa de lectura de teclado y visualización