

Modificado por Luis Frino www.frino.com.ar

Fuente www.micro1.com.ar

directamente (por su número) o a través de su nombre (mnemotécnico definido en el ensamblador, como lo es en el ejemplo OPERANDO1). Es muy útil, combinada con las dos anteriores, para probar fácilmente todas las variantes de una rutina o zona de código determinada sin tener que ejecutar para cada una de nuevo todo el código.



Con este comando se pueden definir puntos de parada (**Breaks**) en la ejecución para, mediante **Run**, no necesitar recorrer línea a línea todo el programa si deseamos ejecutar todo un proceso de golpe hasta esa línea.



Define condiciones de parada (**Conditional Breaks**), es decir, valores de variables o pines (E/S) ante las que parar si se producen.

Por favor, experimente con todas estas opciones. Piense que la excesiva sencillez de **suma.asm** hace que **Run** sea poco útil, pero sí lo es **Step**. Observe como la directiva del ensamblador **END** no tiene equivalente en las instrucciones del procesador, con lo que el micro no se detendrá en ese punto. Téngalo en cuenta en el futuro y solucione el problema, por ejemplo, con un **GOTO**.

Si no le funcionan estos comandos, lea el siguiente punto.

5.2.5 Otras opciones del MPLAB

Es interesante también comprobar qué opciones contiene el menú **Options**. Por ejemplo, con su comando **Development mode...**, en el que es posible elegir el tipo de microprocesador sobre el que simular y **activar el modo de simulación (MPLAB-SIM simulator)**, no siempre activo por defecto. Si ha tenido problemas con la simulación en el apartado anterior, seguramente no tendrá activa aún esta opción.

Con **Default Editor Modes** y sus **Current Editor Modes** podrá cambiar las condiciones generales del editor (por ejemplo, cambiar de ensamblador a C, si tiene el C), incluso diferenciando cuál se debe emplear en cada nodo del proyecto.

Con el submenú **Processor Setup** podremos cambiar cosas como la velocidad del reloj (para controlar el tiempo de ejecución) o la activación del **WatchDog**.

A través del menú **Window**, podrá ver la memoria de programa (submenú **Program Memory**) y la de la EEPROM (si lo desea y la va a utilizar; submenú **EEPROM Memory**).

5.3 Ejemplos básicos de programación

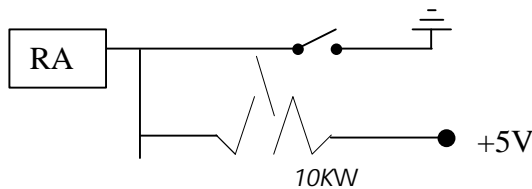
5.3.1 El sistema de E/S. interrupciones y LED's

Debiendo escoger un modelo de PIC para utilizar durante estas prácticas hemos creído conveniente recurrir al más utilizado en la bibliografía como ejemplo, el PIC16C84. Esta elección tampoco es aleatoria y está bien motivada por las siguientes razones: Pertenecer a la gama media, con lo que, al sólo contar con 2 instrucciones más que la gama baja, no necesitaremos grandes explicaciones sobre la gama baja. No teniendo tantas funcionalidades como la gama alta, éstas no siempre son necesarias, y sólo suponen un paso más si la gama media es bien comprendida. Su memoria E²PROM lo hace indispensable en muchas aplicaciones (aunque no las más sencillas) y muy recomendable durante la fase de desarrollo, puesto que ésta es de fácil regrabación, y permite no perder el micro en caso de errores en el código. El modelo PIC16F84 tiene memoria FLASH.

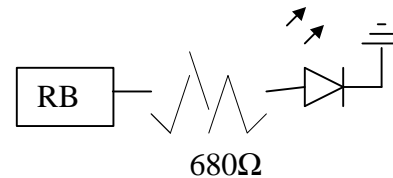
El siguiente paso en esa línea para la comprensión del micro será el estudio de su sistema de E/S. Para ello planteamos el siguiente problema:

Se colocan tres interruptores en las líneas RA0, RA1 y RA2 (puerto A) de un PIC16C84 y cuatro leds en las líneas RB0, RB1, RB2 y RB3 (puerto B), tal y como se refleja en las figuras:

Conexión de interruptores en RA0, RA1 y RA2



Conexión de leds en RB0, RB1, RB2 y RB3



Tomando los interruptores (RA2, RA1, RA0) como un número binario, le sumaremos 2 y sacaremos el resultado por los leds (led apagado=0, led encendido = 1, RB3 bit de mayor peso).

Usaremos el listado del programa **sbinaria.asm**, si bien será desarrollado paso a paso.

Lo primero será definir las posiciones de las variables que queremos utilizar. Esto se puede hacer a través de la directiva **include <p16c84.inc>**, que incluye ya la definición (en base a la literatura inglesa) de todos los registros y bits. Busque, como curiosidad, el fichero "16C84.inc" y lea su contenido. Encontrará un archivo ***.inc** por cada micro de los soportados por el MPLAB.

Nosotros respetaremos la nomenclatura inglesa, pero escribiremos a mano las variables que nos interesan. El motivo es muy simple: en ese p16c84.inc no se ha considerado el problema de los bancos de memoria, que se deben intercambiar a través del registro de estado, de modo que TRISA y TRISB no son funcionales.

De este modo para nosotros PORTA representará al puerto A, PORTB representará al puerto B, TRISA y TRISB serán ellos mismos y W será el registro de estado. Hemos castellanizado el STATUS como ESTADO por comodidad, y hemos llamado BANCO al bit dentro de ESTADO que determina el banco de datos con el que se va a trabajar (recordad que el PIC16C84 tiene dos bancos de datos).

El programa, por lo demás, es bastante evidente. Necesitamos sólo introducir el sentido de los TRIS. Este registro está asociado a los distintos puertos y en él cada bit representa un pin del puerto al que se refiera. Un 0 en uno de sus bits representará que el pin es de salida, y un 1 que es de entrada. De este modo cada pata será independiente, dándonos mayor flexibilidad para la implementación física de los diseños. Es posible incluso, aunque no habitual, cambiar la dirección del pin durante la ejecución del programa. Esta opción puede llegar a darnos la posibilidad de manejar varios dispositivos con un solo puerto.

Ejecute el programa en modo simulación y observe detalladamente como funciona. Recuerde que con la misma dirección de registro referenciamos al puerto y a su tris; todo depende del valor del 5º bit del

LIST P=16C84

```
porta EQU 0x05 ;La dirección 0x05 corresponde al registro porta (puerto A) en el banco0
TRISA EQU 0x05 ;
portb EQU 0x06 ;La dirección 0x06 corresponde al registro portb (puerto B) en el banco1
TRISB EQU 0x06 ;
estado EQU 0x03 ; La dirección del registro de estado es la 0x03
banco EQU 5 ; Bit del registro de estado correspondiente al banco de datos
```

ORG 0

```
BSF estado,banco; Activamos banco 1, que contiene los TRISA y TRISB
```

```
MOULW 0xFF ; Ponemos a 1's el registro W
MOUWF TRISA ; El registro TRIS del puerto A se pone entero a 1's
; cada bit de TRIS representa una patilla idéntica a su
; número de orden. Si se pone a 1 representa que es una
; entrada.
```

```
MOULW 0x00 ; Ponemos a 0's el registro W
MOUWF TRISB ; El registro TRIS del puerto B se pone entero a 0's,
; cada 0 representa que su pin correspondiente es de salida.
```

```
BCF estado,banco; Activamos banco 0, que contiene los puertos A y B
```

```
inicio MOVF porta,W ; Llevamos a W la entrada de los pulsadores por el puerto A
ADDLW 0x02 ; Sumamos 2 al W
MOUWF portb ; Y sacamos el resultado por el puerto B
```

```
GOTO inicio ; Volvemos a empezar
```

END

registro STATUS, que cambia el banco de datos.

Con el **goto** final nos aseguramos que la rutina se ejecute constantemente, atendiendo a las posibles nuevas entradas.

Observe mediante el simulador la imposibilidad de cambiar el valor del **PORTA**, intentando alterar el valor de su dirección de memoria, puesto que es una entrada y depende sólo de valores eléctricos. En cambio sobre **PORTB** (salida) sí podremos. Observe también como **trisa** no queda con el valor 0xff, tal y como sería

aparentemente lógico, sino con el 0x1f, ya que los pins que quedan a 0 no están implementados (el puerto A sólo tiene 5 entradas).

Por último observe la diferencia entre el valor del inspector (el **watch**) en mayúsculas (**TRISA**) y el minúsculas (**trisa**). En mayúsculas **TRISA** referencia siempre a la dirección de registro 0x05, y variará según el banco al que apunte **ESTADO**, puesto que ha sido definida como variable. Se la puede modificar por programa. En minúsculas **trisa** es una variable interna que referencia al banco 1 (dirección 0x85) y no referenciable por programa. De hecho es la única manera de visualizarla y cambiarla en tiempo de simulación, ya que, bajo Windows, no estamos sometidos a la limitación de los bancos de registros. Puede llegar a ser muy engañoso; no se despiste o variará accidentalmente la salida de un puerto si no cambia convenientemente de banco.

Esta es otra de las incomodidades aún existentes en el manejo del **MPLAB**, todavía en desarrollo, junto a la de que las herramientas de cortar y pegar también sean internas, es decir, ni dejen texto en el portapapeles ni sean capaces de tomarlo¹.

Pruebe a variar los valores de **trisa** y **PORTA**. Comprobará como cuando un puerto es de entrada se toma su valor directamente del patillaje, independientemente del valor del registro del puerto, pero conserva el original (su valor de memoria) si se lo vuelve a considerar de salida. Recuerde, ante posibles respuestas curiosas, que sólo se utilizan en el puerto A los bits implementados (RA4-RA0), y el resto se consideran 0, aunque intente modificarlos

¿ Cómo, pues, cambiar los estímulos de una entrada de un puerto? Emplearemos el menú **DEBUG** con el submenú **SIMULATOR STIMULUS** y la opción **ASINCRONOUS STIMULUS...**, lo cuál dará una ventana como la que sigue:



Cada uno de estos botones simula un estímulo sobre una patilla. La forma de editarlos es pulsar el botón de propiedades del ratón (el derecho), sobre uno de ellos, seleccionar la patilla a la que queremos vincularlo y el tipo de cambio que deseamos realizar con él cada pulsación (poner la entrada a uno o High, a cero o Low, que cambie de valor cada vez que se pulse o Toggle). Tras pulsar el botón habrá de ejecutarse la siguiente instrucción antes de ver los cambios a través del inspector.

¹ Este problema parece estar solucionado en la versión de desarrollo del MPLAB 4.99.07, de reciente aparición.

En el ejemplo RA1(L) pondrá a cero el bit 1 del puerto A, RA2(T) cambiará el valor del bit 2 del puerto a y el resto de botones están sin definir.

También es posible crear secuencias sincronas por programa, en función del tiempo.

Plantéese como ejercicio modificar el programa para sumar las entradas RA3-RA0 y RB3-RB0 (todas pulsadores) en el registro 0x0c y poner las patas RA2, RA1 y RB1 con valor 1.

5.3.2 Contar y visualizar

Además de las tan habituales instrucciones de salto (GOTO) y de rutina (GOSUB) contamos con un tercer tipo de salto condicional muy especial, ya que sólo saltará una línea. Las llamaremos de brinco, como traducción de *skip*. Estas instrucciones son BTFSS registro,bit (que brincaré en caso de que el bit valga 1) y BTFSC registro,bit (que lo hará si es 0).

Otro problema al que nos enfrentaremos por primera vez es el del vector de reset y el vector de interrupción. En la familia de los PIC16CXX² el **vector de reset** (situación del contador de programa en caso de reset) está situado en la dirección de programa 0x00 y el de interrupción en la 0x04. De este modo convendrá comenzar nuestros programas en la dirección 0x05, y rellenar la dirección 0x00 con un simple salto a la misma (GOTO 0x05).

Nos planteamos un nuevo problema: *Crearemos un programa para un PIC16C84 funcionando a 4MHZ encargado de contar hasta 0x5f. Cuando lo alcance se detendrá en un bucle no operativo. El valor del contador se visualizará en 8 díodos LED conectados al puerto B.*

La solución es el programa **cuenta.asm**, cuyo listado hallaremos en la página siguiente.

```
LIST P=16C84           ; Seleccionamos el micro concreto que deseamos emplear
; Asignación de etiquetas a registros.
f                       EQU  0x01           ; Indica que el valor de una operación
                               ; se guardará en el registro, no en el W
portb                   EQU  0x06           ; Dirección del registro del puerto B
estado                 EQU  0x03           ; Dirección del registro de estado
conta                  EQU  0x0C           ; Registro sin asignar
                               ; Lo usamos como variable contadora

ORG 0                   ; El programa comienza en la dirección 0 y
GOTO inicio            ; salta a la dirección 5 para sobrepasar
                               ; el vector de interrupción.
ORG 5

inicio BSF  estado,5    ; Selecciona el banco 1 para poder acceder al TRISB
        MOVLW 0x00
```

² Los PIC16C5X tienen situado su vector de reset en la última dirección física de memoria

```

MOVWF portb      ; Y se especifica que es de salida
BCF estado,5    ; Selección del banco 0 para trabajar directamente
                  ; con el puerto

CLRF  conta      ; Ponemos nuestro contador a 0

bucle1 INCF  conta,f      ; conta + 1 --> conta (incrementa el contador)
        MOVF  conta,W      ; conta se carga en W
        MOVWF portb      ; W se carga en el registro de datos del puerto B
        MOVLW 0x5f      ; W <-- 0x5f (Final de cuenta deseado)
        SUBWF conta,W    ; conta - W --> W. Si es cero, la cuenta está acabada
        BTFSS estado,2  ; Explora Z y si vale 1 es que W vale 0
                  ; se produce "brinco" en ese caso por fin de cuenta
        GOTO  bucle1      ; Si Z = 0 se vuelve a bucle1

bucle2 GOTO  bucle2      ; Si Z = 1 se produce un bucle infinito

END

```

Si ejecuta este programa tal cuál se llevará una gran decepción: el estado de la cuenta pasará directamente a 0x5F. ¿A qué se debe esto? Pensemos que el reloj se mueve a una velocidad de 4 Mhz, lo cuál significa 0'25 μ segundos. Como cada instrucción necesita cuatro ciclos de reloj esto implica un ciclo de instrucción de 1 μ segundo, y un total de 7 instrucciones (con una de salto, que ocupa 2 ciclos) de ejecutará en 8 μ segundos. La cuenta es demasiado rápida. Puede cambiar la frecuencia de reloj a través del menú **OPTIONS**, submenú **clock frequency**³.

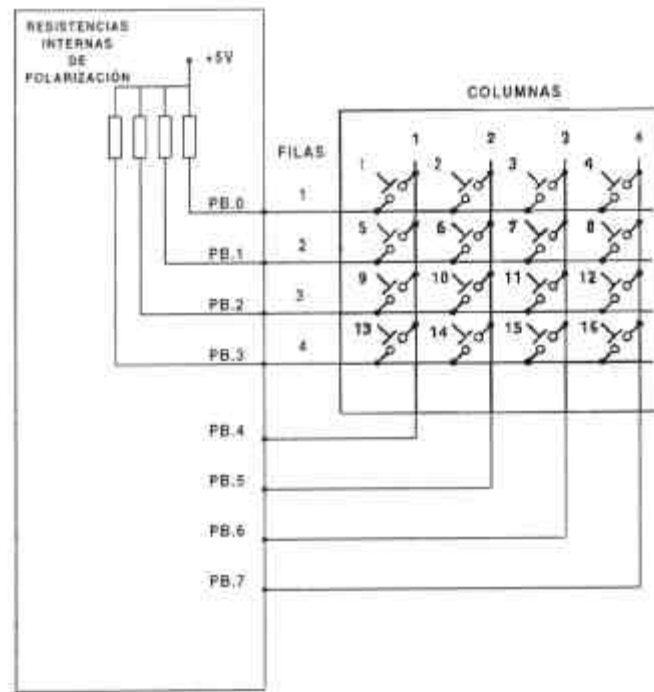
Proponemos al lector como ejercicio extra que incorpore un retardo al programa que consiga que sea visualizable. Nosotros planteamos una solución en **cuenta2.asm**. ¿ Podría conseguir contar segundos con un reloj de 1 Mhz? Lamentamos comunicarle que no hemos encontrado ningún método, aparte del manual, para calcular el tiempo de ejecución a través del MPLAB.

Nos enfrentamos, además, por primera vez al problema del WatchDog. Este es un mecanismo que asegura el reset automático del micro si, pasado un tiempo programable, no se ha ejecutado el comando CLRWDT (CLear WatchDog Timer). Por este motivo no se quedará el micro en modo de bucle infinito salvo que desactive la opción WDT (menú OPTIONS, submenú PROCESSOR SETUP...)

5.3.3 Teclado matricial

³ En la versión 4.99.07 la activación del WDT se realiza a través del submenú **Development Mode**.

Una buena manera de ahorrar líneas de entrada al micro es, en lugar de dedicar una línea exclusiva por cada tecla utilizada, emplear un teclado matricial. El teclado matricial se caracteriza por estar cada una de las teclas conectada a dos líneas (una columna y una fila) que la identifican. De este modo el número de teclas que pueden conectarse es el producto de filas por el de columnas. En el siguiente esquema se muestra la manera correcta de conectar un teclado matricial a un PIC.



La técnica de programación requiere tanto de entradas como de salidas. Las filas están conectadas a las patillas de salida y las columnas a las de entrada.

Se comienza el testeo colocando a '0' la primera fila, y a '1' las restantes. Si la tecla pulsada estuviese en la columna '0', ésta colocaría en su línea un '0' lógico. Bastará con hacer un muestreo de las columnas, buscando el 0, para saber la tecla exacta que fue pulsada en la matriz.

Si no es pulsada ninguna tecla en una fila, las entradas se encuentran en estado flotante, razón por la que son necesarias las resistencias de polarización internas, que mantienen las entradas a nivel alto. Dichas resistencias permiten conectar el teclado matricial sin añadir componentes externos. Su activación se realiza a través del bit **RBPU del registro OPTION**.

El programa ejemplo, llamado **Teclado.asm**, gestiona un teclado hexagesimal (4 filas y 4 columnas) situado en la puerta B. Con el LED situado en RA4 marcaremos si hay o no una tecla pulsada (se encenderá si alguna lo está), mientras que con el resto de LED's conectados al puerto A indicaremos el número binario equivalente a la tecla pulsada.

El WatchDog no está siendo controlado, con lo que le sugerimos que lo desactive para la simulación.

```

LIST P = 16F84          ; Indicamos el modelo de PIC a utilizar

; Definición de registros

opcion EQU 0X01        ; Dirección del registro OPTION
leds   EQU 0X05        ; Hemos conectado los LED's al puerto A
                        ; La dirección 0x05 corresponde al registro porta
                        ;                               (puerto A) en el banco0
TRISAEQU EQU 0X05      ;                               y TRISA en banco 1
teclado EQU 0x06       ; Hemos conectado el teclado al puerto B
                        ; La dirección 0x06 corresponde al registro
                        ;                               portb (puerto B) en el banco1
TRISBEQU EQU 0X06      ;                               y TRISB en banco 1
estado  EQU 0X03       ; La dirección del registro de estado es la 0x03

; Definición de bits

bancoEQU 5             ; Bit del registro de estado correspondiente
                        ;                               al banco de datos
pulsada  EQU 4         ; Bit del LED que indica si una hay teclas pulsadas
rbpu     EQU 7         ; Bit que activa las resistencias internas del puerto B
Z        EQU 2         ; Bit de registro W con valor cero
C        EQU 0         ; Bit del acarreo. Se pone en el bit 0 al rotar un registro.

; Definición de variables

cont1 EQU 0X0C        ; Contador1 de la pausa
cont2 EQU 0X0D        ; Contador2 de la pausa
cont3 EQU 0X0E        ; Contador3 de la pausa
tecla  EQU 0X0F       ; Número de Tecla

ORG 0X00              ; Colocamos el vector de reset

                GOTO inicio          ; Y saltamos a la dirección 0x05, para evitar
                ; el vector de interrupción, en 0x04

```

ORG 0X05

```

Inicio      BSF estado,banco      ; Cambio de página para cambiar
           ;                   ; los TRIS de cada puerto
           MOVLW 0x00          ; El puerto A es todo de salida
           MOVWF TRISA
           MOVLW 0xF0          ; Las patillas RB0-RB3 representan las filas del
           ;                   ; teclado, y son salidas
           ;                   ; Las patillas RB4-RB7 representan las columnas,
           ;                   ; y son entradas
           MOVWF TRISB
           BSF opcion,rbpu    ; Conectamos las resistencias de polarización
           ;                   ; interna del puerto B
           BCF estado,banco   ; Cambio de página de nuevo al banco 0

borrarCLRF cont1      ; Borra el contador
           CLRF cont2         ; Borra el contador
           MOVLW 0x08
           MOVWF cont3       ; Carga el retardo de 0.5 segundos

           BCF leds,pulsada ; Borra que la tecla siga pulsada

cuerpo      CLRF tecla        ; Tecla actual=0
           MOVLW 0x0E
           MOVWF teclado    ; Saca 0 a la fila 1, para testearla

chkcol     BTFSZ teclado,4    ; Chequea la columna 0 en busca de un '0'
           GOTO numtecla     ; si encuentra un 0 muestra el número
           ;                   ; de tecla pulsada
           INCF tecla        ; si no encuentra el 0, incrementa el número de tecla

           BTFSZ teclado,5    ; Chequea la columna 1 en busca de un '0'
           GOTO numtecla     ; si encuentra un 0 muestra el número
           ;                   ; de tecla pulsada
           INCF tecla        ; si no encuentra el 0, incrementa el número de tecla

           BTFSZ teclado,6    ; Chequea la columna 2 en busca de un '0'
           GOTO numtecla     ; si encuentra un 0 muestra el número de tecla pulsada
           INCF tecla        ; si no encuentra el 0, incrementa el número de tecla

           BTFSZ teclado,7    ; Chequea la columna 3 en busca de un '0'
           GOTO numtecla     ; si encuentra un 0 muestra el número
           ;                   ; de tecla pulsada
           INCF tecla        ; si no encuentra el 0, incrementa el número de tecla

UltTec?    MOVLW 0x10        ; Carga W con el número de teclas +1
           SUBWF tecla,w     ; y lo compara con el valor actual de tecla
           BTFSZ estado,Z    ; Si hemos llegado a Tecla+1 acabamos
           ;                   ; el ciclo de filas
           GOTO cuerpo       ; y no se habrá pulsado ninguna tecla
           BSF estado,C      ; Y ponemos a 1 el bit C para...
           RLF teclado       ; que la fila 1 pase a ser un 1 en la rotación de pilas,
           ;                   ; y el 0 se desplace
           GOTO chkcol

numtecla   MOVF tecla
           MOVWF leds        ; Pasa el valor de la tecla a los LED's
           BSF leds,pulsada ; y activa el indicador de tecla pulsada

pausa DECFSZ cont1      ; Bucle anidado de 3 niveles
           GOTO pausa       ; que retarda aprox. 0.5 segundos
           DECFSZ cont2
           GOTO pausa
           DECFSZ cont3

```

GOTO pausa
GOTO borrar
END

5.3.4 Tablas y subrutinas

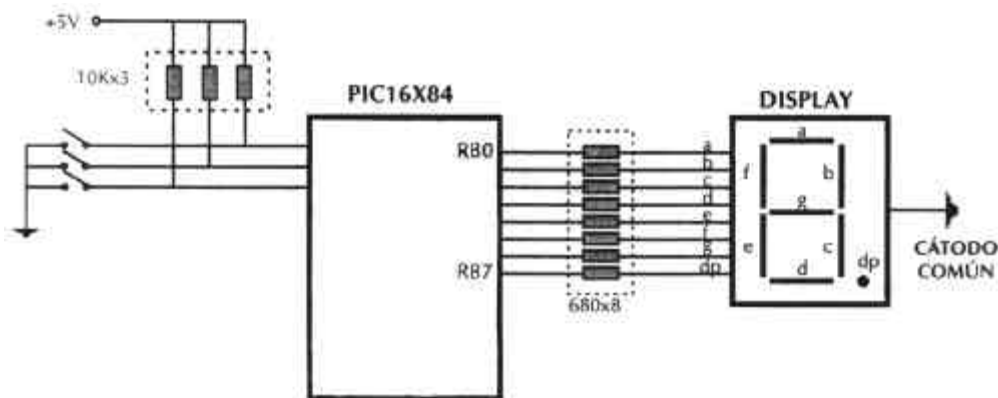
El uso de rutinas, siendo sencillo y no requiriendo un apartado específico, es esencial, puesto que simplifica los programas, haciéndolos, además, más modulares. Llamaremos a una subrutina con la orden **CALL**, seguida de la etiqueta que la encabeza o su dirección en memoria. Para regresar a la siguiente instrucción tras el CALL bastará con situar, en la última línea de la subrutina, el comando **RETURN**; también será posible emplear **RETLW k**, el cuál se diferencia del anterior por situar en el registro W el valor k.

En los procesadores de gama media y alta existe una manera específica de regresar en caso de interrupción: **RETFIE**. Las interrupciones generan un salto a la dirección 0x04 que es tratado como una subrutina para permitir la continuación normal del programa a partir del punto en que se llevó a cabo la interrupción. Este regreso se lleva a cabo mediante RETFIE, que, además de restaurar el contador de programa (PC) habilita de nuevo las interrupciones (ya que estas son deshabilitadas mientras se está atendiendo una).

La pila de la gama media permite guardar hasta 8 saltos, es decir, admite un máximo de 8 saltos a subrutina anidados. Recuerde este dato para no excederse en las llamadas a las mismas ni en los niveles de un método tipo "Divide y Vencerás". Recuerde también siempre un posible salto por interrupción, para no pasar de 7 si ésta está habilitada. En la gama baja sólo contará con dos niveles en la pila.

Una tabla, en cambio, es una lista ROM de constantes en la memoria de programa, y que pueden ser desde notas musicales a caracteres ASCII. Son especialmente útiles para la conversión de unos códigos a otros, como, por ejemplo, para pasar un hexagesimal a un display. Para generarlas se aprovecha la cualidad de RETLW para situar un dato en el registro W.

Supongamos que deseamos sacar, de forma secuencial, un número del 0 al 9 en un display de 7 segmentos conectado a la puerta B, tal y como figura en el esquema siguiente:



En el siguiente programa, llamado **Cuenta.asm**, se necesitan ambas cosas, tanto las rutinas como las tablas. El display elegido es de cátodo común, es decir, los segmentos se iluminan cuando las salidas de la puerta B están a nivel alto.

En la siguiente tabla podrás comparar el código hexagesimal y su equivalente en 7 segmentos:

HEXAGESIMAL	7-
\$00	\$3F
\$01	\$06
\$02	\$5B
\$03	\$4F
\$04	\$66
\$05	\$6D
\$06	\$7D
\$07	\$07
\$08	\$7F
\$09	\$6F

```

LIST P = 16F84          ; Indicamos el modelo de PIC a utilizar

; Definición de registros

portb      EQU    0x06    ; Hemos conectado el display al puerto B
              ; La dirección 0x06 corresponde al registro PORTB
              ; (puerto B) en el banco1

TRISBEQU   EQU    0x06    ; y TRISB en banco 1
Estado     EQU    0x03    ; La dirección del registro de estado es la 0x03
pc         EQU    0x02    ; Contador de Programa, es decir,
              ; dirección de memoria actual de programa

; Definición de bits

bancoEQU   EQU    0x05    ; Bit del registro de estado correspondiente
              ; al banco de datos
Z          EQU    0x02    ; Bit indicador de que el registro W está a cero

; Definición de constantes

w         EQU    0        ; Destino de operación = w
f         EQU    1        ; Destino de operación = registro

; Definición de variables

contador  EQU    0x0C    ; Contador
digito    EQU    0x0D    ; Para almacenar el dígito

; Comienzo del programa.

ORG 0x00          ; Cubrimos el vector de reset

          GOTO inicio    ; Saltamos a la primera dirección tras el vector de interrupción

; ***** Inicialización de variables *****

ORG 0x05

inicioBSF estado,banco; Cambiamos a la segunda página de memoria
          CLRF TRISB    ; Programa la puerta B como de todo salidas
          BCF estado,banco ; Volvemos a la página 0.
          CLRF portb    ; Apaga el display, por si había residuos
          CLRF contador ; Borra el contador (dirección 0x0C)

```

```

CLRW                ; Borramos el registro W
; ***** Cuerpo Principal *****
Reset              CLRf digito          ; Comienza a contar por el 0

Siguien           MOVf digito,w         ; Coloca el siguiente dígito a evaluar en W
                  CALL Tabla           ; Llama a la subrutina para coger el dato
                  MOVWF portb          ; y hacer la conversión decimal-7 segmentos

Pausa DECFSZ contador; Decrementa contador y repite
GOTO Pausa        ; hasta que valga 0
INCF digito,f     ; Incrementa el valor del dígito al siguiente
MOVf digito,w    ; Pone el valor del dígito en W
XORLW 0x0A       ; Chekea si el dígito sobrepasa el valor 9
BTFSC estado,Z  ; Comprobando con un xor si W vale 0 (Z=1)
GOTO Reset       ; Si Z=1 resetea el dígito y comienza de nuevo la cuenta
GOTO Siguien     ; En caso contrario, continua la cuenta
                  ; con el siguiente dígito

Tabla ADDWF pc,f  ; Suma al contador de programa el valor de offset, es decir,
                  ; el valor del dígito. Así se genera un PC distinto según su valor,
                  ; asegurando que vaya a la línea correcta de la tabla
RETLW 0x3F       ; 0 en código 7 segmentos
RETLW 0x06       ; 1 en código 7 segmentos
RETLW 0x5B       ; 2 en código 7 segmentos
RETLW 0x4F       ; 3 en código 7 segmentos
RETLW 0x66       ; 4 en código 7 segmentos
RETLW 0x6D       ; 5 en código 7 segmentos
RETLW 0x7D       ; 6 en código 7 segmentos
RETLW 0x07       ; 7 en código 7 segmentos
RETLW 0x7F       ; 8 en código 7 segmentos
RETLW 0x6F       ; 9 en código 7 segmentos

END

```

5.3.5 Manejo de interrupciones

GENERALIDADES SOBRE LAS INTERRUPCIONES

La **interrupción** es una técnica que coloca al programa temporalmente en suspenso mientras el microcontrolador ejecuta otro conjunto de instrucciones en respuesta a un suceso. Las causas de una interrupción pueden ser externas, como la activación de una patilla con el nivel lógico apropiado, e internas, como las que pueden producirse al desbordarse un temporizador como el TMR0.

Los **sucesos internos capaces de producir una interrupción** más destacables son el *desbordamiento del temporizador TMR0, fin de la escritura de la EEPROM, finalización de la conversión A/D*. Los **sucesos externos** principales son *la activación del pin 0 de la puerta B (PBO/INT), el cambio de estado en las patitas 4-7 de la puerta B, y el desbordamiento del temporizador TMR0*.

Cuando se produce una interrupción el procesador ejecuta una **Rutina de Servicio de Interrupción (RSI)**, y, al terminar, el programa principal continúa donde fue interrumpido. La dirección en la que se debe situar la rutina de interrupción es la 0x04, y es recomendable, para terminarla, usar la instrucción RETFIE, en

lugar de RETURN, puesto que, al activarse una interrupción, el mecanismo de las mismas se deshabilita como medida de seguridad. RETFIE sirve para rehabilitarlas.

Como las rutinas pueden modificar el contenido de los registros del procesador, al iniciarlas conviene guardar en la pila el valor de los mismos y restaurarlos antes del RETURN. Antes de regresar la RSI debe determinar la causa de la interrupción, borrar la bandera apropiada antes de salir y, por supuesto, dar servicio a la interrupción.

A primera vista, **salvar y restaurar los registros sin modificar sus contenidos** no es una tarea fácil. El contenido del registro W debe guardarse primero, junto con todos los registros que han pasado por W para el almacenamiento temporal de sus posiciones. El hecho de mover W a otro registro corrompe la bandera Z, modificando el registro de Estado. Microchip recomienda una secuencia de código que permite salvar y restaurar los registros sin modificarlos. La mostramos en la siguiente secuencia de código:

```
, ***** SALVAR *****  
MOVWF Almacen1_W          ; Guardamos contenido de W en su sitio  
SWAPF estado,w           ; Swap del contenido de estado en W  
MOVWF Almacen2_S         ; Guarda el contenido de ESTADO  
...  
...  
...  
  
, ***** FIN RUTINA RSI *****  
SWAPF Almacen2_S,w       ; Deja estado como estaba  
MOVWF estado             ; Y lo restaura  
SWAPF Almacen1_W,f  
SWAPF Almacen1_W,w  
RETFIE
```

La instrucción SWAPF mueve los datos sin afectar a la bandera Z del registro de ESTADO. Aunque los conjuntos de 4 bits se invierten en el proceso, posteriormente son restaurados en su situación inicial. Si se empleara la instrucción MOVF se corrompería el bit Z.