2.- Debug:

Antes de continuar debemos decir que todos los comentarios que se hacen sobre la visualización de pantallas, están referidos a la configuración de la Figura 41, a no ser que se especifique lo contrario.

Cuando activamos esta opción de la barra de menú, aparece el menú desplegable de la Figura 42.

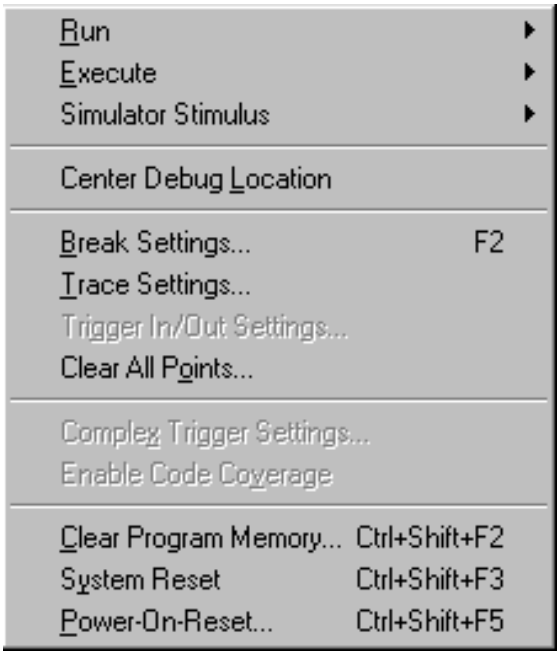


Figura 42.- Menú desplegado de la opción Debug de la barra de herramientas

2.1.- Run: Cuando se selecciona *Run* aparece el menú desplegable de la Figura 43 con las opciones:

- Run
- Reset
- Halt
- Halt Trace
- Animate
- Step
- Step Over
- Update All Registers
- Change Program Counter

Que seguidamente analizaremos

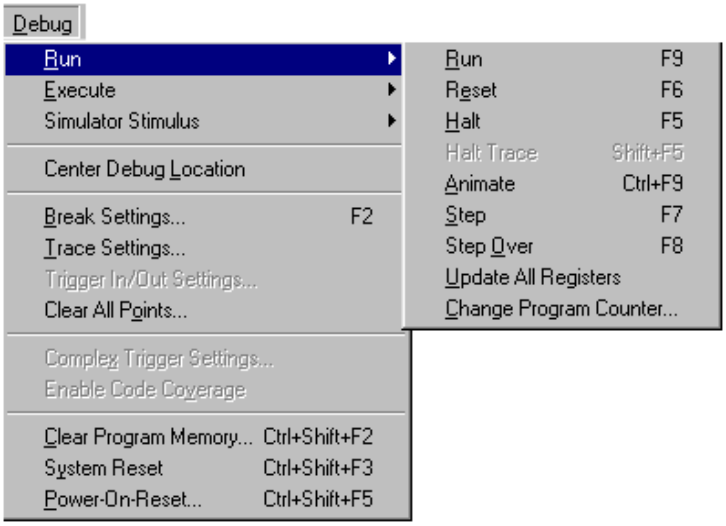


Figura 43.- Menú desplegable de la opción Debug>Run de la barra de menú

2.1.1.-Debug>Run>Reset: Esta opción inicializa el sistema. El Contador de Programa (PC), que es la dirección de memoria donde se encuentra la primera instrucción que ejecuta el microcontrolador cuando se realiza un *Reset* en el sistema; se pone a cero, a esta dirección de memoria se le denomina *vector de Reset*. La línea del código fuente de esta dirección queda resaltada con una barra en vídeo inverso, además se puede comprobar que el PC se pone a 0x00

Para activar esta acción también se puede activar la tecla [F6] o el siguiente icono de la barra de herramientas 

2.1.2.-. Debug>Run>Step: Esta acción ejecuta la instrucción cuya dirección de memoria coincida con el valor al que apunta el PC antes de activarla. Para comprobar su funcionamiento en nuestro programa ejemplo si, después de haber realizado un *Reset,* se ejecuta la opción *Step* podemos ver como el PC pasa a valer "0x05", que es lo que esperamos después de realizar la instrucción *goto INICIO*, donde INICIO = 05h, utilizamos esta posición para salvar el *vector de Interrupción* que está en la posición de memoria 04h, además podemos ver como se resalta en vídeo inverso esta dirección de memoria. Si algún registro más se hubiera modificado este quedaría resaltado en color rojo como lo han hecho los registros ***pcl*** y ***t0pre*** del *SFR*.

Si se pulsa la tecla [F7] o se activa el siguiente icono de la barra de herramientas  el efecto es exactamente el mismo. Si volvemos a ejecutar la opción ***Step*** (paso) veremos que al ejecutar la instrucción *movlw 0x07* se alteran los siguientes registros ***W***=07, ***pcl***= 06 y

t0pre=03, que es justamente lo que esperaríamos de la ejecución de esta instrucción “cargar el literal 07 en el registro W”, el PC se incrementa en uno apuntando a la siguiente instrucción a realizar y el *timer0* también se incrementa en uno.

Como puede apreciarse esta opción nos permite comprobar paso a paso como se ejecuta nuestro programa, y de esta forma, ver si hace lo que nosotros queremos o por el contrario si nos hemos equivocado al escribir el programa y donde no está haciendo lo que nosotros esperábamos que hiciera. Esta opción es especialmente interesante cuando se comienza a estudiar el ensamblador del microcontrolador y queremos comprobar el *set* de instrucciones del mismo o bien para depurar los programas.

2.1.3.-. Debug>Run>Animate: Esta opción es similar a la anterior solo que en este caso en lugar de ejecutar de forma controlada el programa, este se ejecuta de forma automática desde la dirección que indique en el instante de activarla el PC hasta que finaliza, en nuestro caso con la instrucción ***Sleep***, aunque podemos comprobar como, estando en el modo “dormido” el contador ***t0pre*** se sigue incrementando por si se quiere “despertar” por desbordamiento del “perro guardián”, en el caso de estar habilitado, que no es el nuestro.

Se puede notar que cuando se está trabajando en el modo ***Animate*** la línea de estados, que está en la parte mas baja de la pantalla, tiene el fondo de color amarillo. También se puede activar esta opción pulsando simultáneamente las teclas [Control]+[F9].

Para parar la ejecución se activa la opción ***Halt***.

Cuando se quiere ejecutar el programa y que cuando se ejecute una instrucción determinada este se detenga, para poder de esta forma analizar que es lo que ha pasado hasta este momento en el programa, debemos utilizar un ***Break Point*** *o Punto de Ruptura*. La forma más sencilla de introducir un *Break Point*, es ponerse con el ratón en la línea de programa donde queremos colocar le y pulsar el botón de la derecha de ratón, esta acción hace que se despliegue una cuadro de acciones como el de la Figura 44, seguidamente se selecciona la opción *Break Point* y en la pantalla queda resaltada en rojo línea de programa donde queremos que el programa se detenga cuando se esté ejecutado en modo *Animate* o *Run*.

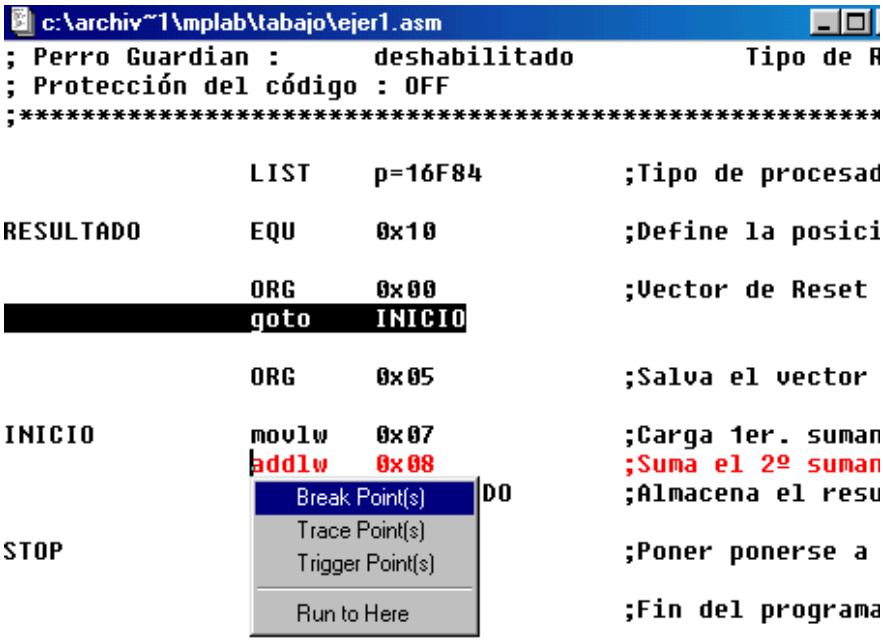


Figura 44.- Activación de un Break Point durante la simulación.

Para comprobar esto vamos a seguir los pasos siguientes:

- Hacemos un Reset en el programa de simulación.
- Activamos la ejecución del programa en cualquiera de los dos modos continuos, es decir, en modo *Animate* o *Run*
- Verificamos que al llegar a la línea donde hemos puesto el *Break Point*, el programa se para.

Si lo que nos interesa es ver como evoluciona el microcontrolador al ejecutar una determinada instrucción, lo que tenemos que hacer es una *Traza*, es decir, anotar el valor que van tomando los registros que modifica la instrucción cada vez que se pasa por ella en el programa. Para ello antes de empezar la simulación de la ejecución del programa, activaremos la opción ***Trace Point*** en el menú que aparece en la pantalla de la Figura 44. Esta acción resalta en color verde la línea de programa seleccionada, En este caso, para ver la evolución del contenido de los registros a los que afecta esta instrucción, debemos abrir la ventana ***Windows>Memory Trace***. Para comprobar el funcionamiento de este nuevo comando:

- Ejecutar el programa en el modo *Animate* o *Run*.
- Verificar que el programa sigue ejecutándose pero que en la ventana de *Memoria de Traza*, quedan registrados los cambios que se producen en el microcontrolador cuando se ejecuta la instrucción y el instante de tiempo de su ejecución.

2.1.4.-. Debug>Run>Halt: Esta acción detiene el programa que se está ejecutando en este momento, tal y como se ha indicado en el apartado anterior. También se puede activar pulsando la tecla [F5] o el icono del semáforo rojo  . Al hacerlo la línea de estado se vuelve a poner de color blanco y se actualizan los registros con los valores correspondientes a la última instrucción ejecutada. Esta opción hace lo mismo cuando se está ejecutando el programa en el modo ***Debug>Run>Run***.

2.1.5.-. Debug>Run>Run: Al activarse esta opción el programa se ejecuta a toda velocidad "modo tiempo real", no obstante, no hay que olvidar que se trata de una simulación por ordenador de la ejecución del programa y que, por lo tanto, es mucho más lenta. Otra forma de activar esta acción es pulsar la tecla [F9] o el icono del semáforo verde  Cuando esto ocurre, el fondo de la barra de estado se pone de color amarillo y no se actualizan los registros. Para parar la ejecución se pulsa la opción ***Debug>Run>Halt***.

2.1.6.-. Debug>Run>Step Over: Esta opción ejecuta paso a paso las instrucciones de igual forma que lo hace la opción ***Debug>Run>Step***, pero cuando llega a la ejecución de una subrutina (instrucción *call*) ésta se ejecuta de igual forma que si fuera una sola instrucción, si es una subrutina larga como puede ser por ejemplo una temporización, lógicamente esta tardará más tiempo en ejecutarse que una sola instrucción. Para activar esta opción, también se puede pulsar la tecla [F8] o bien activando con el ratón el siguiente icono  .

2.1.7.-. Debug>Run>Chagr Program Counter..: Cuando se activa esta opción aparece un cuadro de diálogo como el de la Figura 45

Figura 45.- Cuadro de diálogo para modificar el contador de programa.

En el campo de PC, podemos poner cualquier posición de memoria o bien, si se activa la flecha, se despliega un menú con todas las etiquetas que tenga nuestro programa. Seguidamente se activa el botón de ***Change***. Esta opción, que nos permite realizar saltos, puede ser utilizada en la depuración de programas cuantas veces se desee. Con ello se evitará entrar en bucles del programa de los cuales es muy difícil salir si no se cumplen determinadas condiciones o tardan mucho tiempo en ejecutarse como pueden ser las temporizaciones.

Otra forma de activar esta acción es pulsar el icono de la barra de herramientas  .

2.1.8.-. Debug>Run>Hall Trace: Permite al emulador cargar el buffer de memoria con los datos de los registros en los puntos marcados con un ***Trace Poins(s)***. Para más información ver la documentación del emulador que se esté utilizando.

2.1.9.-. Debug>Run>Update All Registers: Actualiza el contenido de los registros después de haber ejecutado una instrucción. Como podemos apreciar, los registros se actualizan cuando se está ejecutando el programa en modo paso a paso, pero si queremos dejar registrada una traza del programa y tenemos abierta la ventana de memoria de traza ***Windows>Trce Memoy*** cuando se ejecuta la instrucción en la que se quiere registrar la traza, esta no se actualiza al no ser que se active la opción ***Debug>Run>Update All Registers***.

2.2.- Execute

Esta opción del menú permite controlar el estado de los registros durante la ejecución y el *firmware* del procesador designado. Al activarla se despliega el menú de la Figura 46.

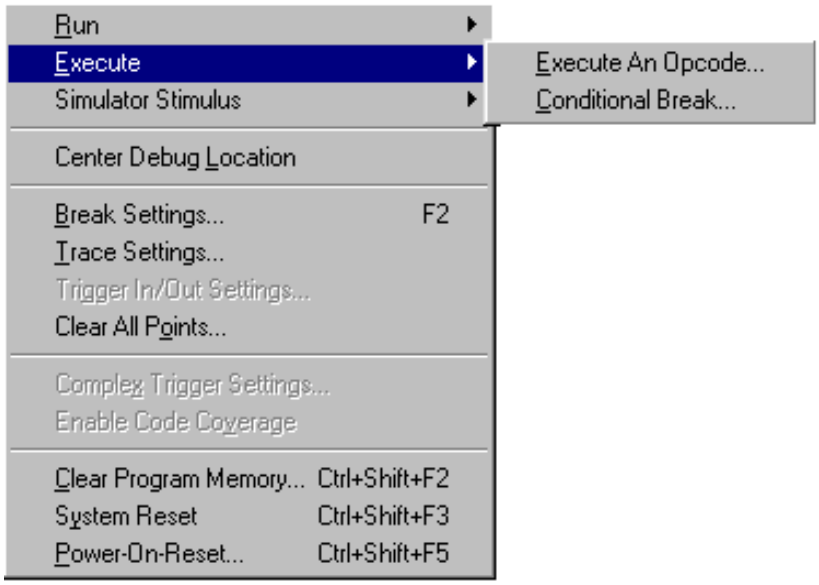


Figura 46.- Menú de la opción Execute de la barra de menú.

2.2.1.- Execute an Opcode: activando ***Debug>Execute>Execute an Opcode*** se ejecuta una sola instrucción o una serie de instrucciones sin modificar el programa código objeto de la memoria. Después de ejecutar la instrucción, se puede seguir ejecutando el programa desde el estado de actual de la memoria de programa.

Al activar esta opción aparece el cuadro de diálogo de la Figura 47.

Figura 47.- Cuadro de diálogo de la opción Execute an Opcode

- ***Opcode***: Introducir la instrucción en hexadecimal o más cómodamente en ensamblador (*movlw 0x025*). Haciendo clic en la flecha que hay junto al campo de *Opcode*, se despliega una ventana que contiene las ocho últimas ordenes que se ha introducido del modo anteriormente expuesto (si se han ejecutado dos instrucciones iguales solo guarda una de ellas).
- ***Execute***, Al activar este botón se ejecuta la instrucción pero no se modifica el PC ni los temporizadores.

2.2.2.- Conditional Break:

Fija un punto de ruptura condicional, el MPLAB se detiene cuando el valor de un registro específico alcanza un valor o una condición prefijada de antemano.

Si se activa la opción ***Debug>Execute>Condicional Break***, se despliega un cuadro de diálogo como el que se muestra en la Figura 48.

Se puede ejecutar parte del programa de forma automatizada. La ejecución comienza al pulsar el botón de ***Start*** y se ejecuta hasta que se de la condición de parada o se active el botón de ***Halt*** del cuadro de diálogo.

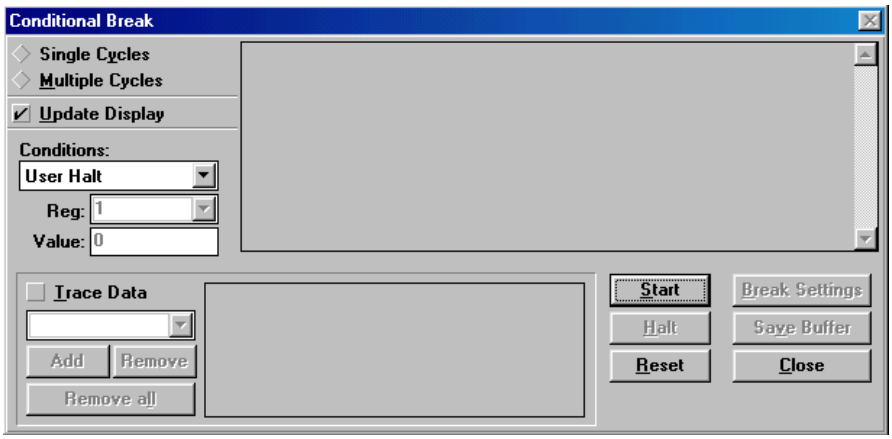


Figura 48.- Cuadro de dialogo de la opción Conditional Break

Veamos el significado de cada uno de los campos de este cuadro de diálogo:

Conditions: (Condiciones): el MPLAB se detendrá en un *Break Point* si en el cuadro de diálogo se activaron alguna de las siguientes condiciones:

- ***Halt***: Se ejecuta el programa hasta que se active el botón *Halt* en el cuadro de diálogo de *Conditional Break* o se dé la condición de Parada.
- ***Multiple Cycles***: Al activar esta opción, se activa el botón *Break Setting* que hasta este instante estaba deshabilitado. Cuando se pulsa este botón aparece el cuadro de diálogo de la Figura 49, en el que se puede indicar el numero de veces que queremos que se repita una instrucción o la condición que queremos que cumpla el *Break Point*.

  Se detiene la ejecución después de que el microcontrolador ejecute el número especificado de ciclos.

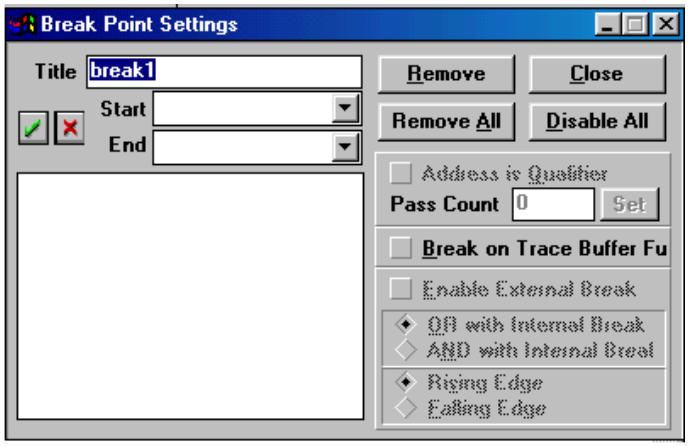


Figura 49.- Cuadro de dialogo de Break Point Settings

- ***Logic Condition satisfied***: Se detiene la ejecución cuando se cumple una Condición Lógica

Trace Data:

La Traza de Datos permite rastrear el valor de los registros en el cuadro de diálogo de los puntos de ruptura condicionales.

Single Cycle:

El modo de Ciclo único del MPLAB ejecuta las instrucciones del microcontrolador hasta que se cumple la condición de parada.

Multiple Cycles

En el modo de Ciclos Múltiples se cumple que:

- ***Conditional Break*** ejecuta las instrucciones en tiempo real (en el emulador), el usuario selecciona los puntos de parada, se verifica la condición especificada y el microcontrolador continúa ejecutando las instrucciones hasta que encuentre la condición especificada.

- Los valores que deben tomar los registros para que se produzca el *Break point* se especifican en el cuadro de diálogo de *Conditional Break*.

Para comprobar todo lo dicho hasta el momento vamos a simular el funcionamiento del programa Pendulo.asm, el programa consiste en la realización de una secuencia de datos en el PortB que simula el movimiento de un péndulo de diodos LED's, además se puede comprobar el funcionamiento de programa utilizando el módulo 01 y 02 que han sido publicados en las Revistas números 162 y 163 o bien se puede montar sobre una placa de inserción rápida el circuito de la Figura 51.

```
;*****************************************************************************
; Programa Pendulo.asm                          Fecha : 7 - Enero - 98
; Este programa genera una secuencia de Led que simula el movimiento de
; un péndulo de los diodos LED conectados al PORTB
; Revisión : 0.0                                Programa para PIC16F84
; Velocidad del Reloj: 4 MHz                    Reloj Instrucción: 1 MHz = 1 uS
; Perro Guardián: deshabilitado                 Tipo de Reloj: XT
; Protección del código: OFF
;*****************************************************************************
;*****************************IGUALDADES ***************************************
;
; ********** Igualdades que designa los destinos ***********************************
w               EQU     0               ;El resultado se guarda en w
f               EQU     1               ;El resultado se guarda en el registro

;******************Igualdades que designan los registros SFR**************************
                LIST    p=16F84         ;Tipo de procesador

PORTA           EQU     05h     ;PortA
PORTB           EQU     06h     ;PortB
TRISA           EQU     05h     ;Registro Direccionamiento PortA
TRISB           EQU     06h     ;Registro Direccionamiento PortB
STATUS          EQU     03h     ;Registro Status
RP0             EQU     05h     ;Bit RP0 del registro de STATUS
C               EQU     00h     ;Bit Flag C del registro de STATUS
;************************ Igualdades de registros auxiliares************************
CONTA1          EQU     0Ch     ;Registro utilizado en la temporización
CONTA2          EQU     0Dh     ;Registro utilizado en el temporización
;************************Código en Reset ***************************************
                ORG     00h             ;Dirección del vector de Reset
                goto    INICIO          ;Comienza el programa después
                                        ;del vector de interrupción
                ORG     05h             ;Una posición detrás vector de Interrupción
;************************** Sección Inicializa **********************************

INICIO      bsf     STATUS,RP0      ;Selecciona la página 1 de la memoria poniendo
                                    ;a 1 el bit RPO por que el registro TRISB esta
                                    ;en la página 1
            clrf    TRISB           ;Coloca la Puerto B como salida
                                    ;borrando TRIS B
            bcf     STATUS,RP0      ;Vuelve a la página 0
            clrf    PORTB           ;Apaga los LEDs borrando la PortB
            clrf    CONTA1          ;Inicializa CONTA1
            clrf    CONTA2          ;Inicializa CONTA2
```

```
;************************ Principal ***************************************
ROTAR       bsf     PORTB,0         ;Bit 0 PortB =1 enciende el led del bit 0
            bcf     STATUS,C        ;Bit de acarreo C=0
ROTA_IZQ    call    TEMPO           ;Llama a la subrutina TEMPO
            rlf     PORTB,f         ;Rota un bit a la izquierda el PORTB con C
            btfss   PORTB,7         ;Si el bit 7 de PORTB =1 sale
            goto    ROTA_IZQ        ;Salta a ROTA_IZQ
            bcf     STATUS,C        ;Bit de acarreo C=0
ROTA_DER    call    TEMPO           ;Llama a la subrutina TEMPO
            rrf     PORTB,f         ;Rota un bit a la derecha el PORTB con C
            btfss   PORTB,0         ;Si el bit 0 del PORTB=1 sale del bucle
            goto    ROTA_DER
            goto    ROTAR

;************************ Subrutina TEMPO *********************************
TEMPO       clrf    CONTA1          ;Borra el contenido de CONTA1
            clrf    CONTA2          ;Borra el contenido de CONTA2
BUCLE       decfsz  CONTA1,f        ;resta 1 al contenido de CONTA1
                                    ;Si CONTA1 llega a cero: salta
                                    ;la instrucción GOTO BUCLE
                                    ;Si CONTA1no llega a cero: ejecuta
                                    ;la instrucción goto BUCLE
            goto    BUCLE           ;Cierra el primer bucle de retardo
            decfsz  CONTA2,f        ;Lo mismo que el caso anterior, pero
                                    ;esta vez aplicado a CONTA2
            goto    BUCLE           ;Cierra el segundo bucle de retardo
            return                  ;Retorno de subrutina
            END
```

El programa Pendulo.asm sigue el organigrama de la Figura 50.

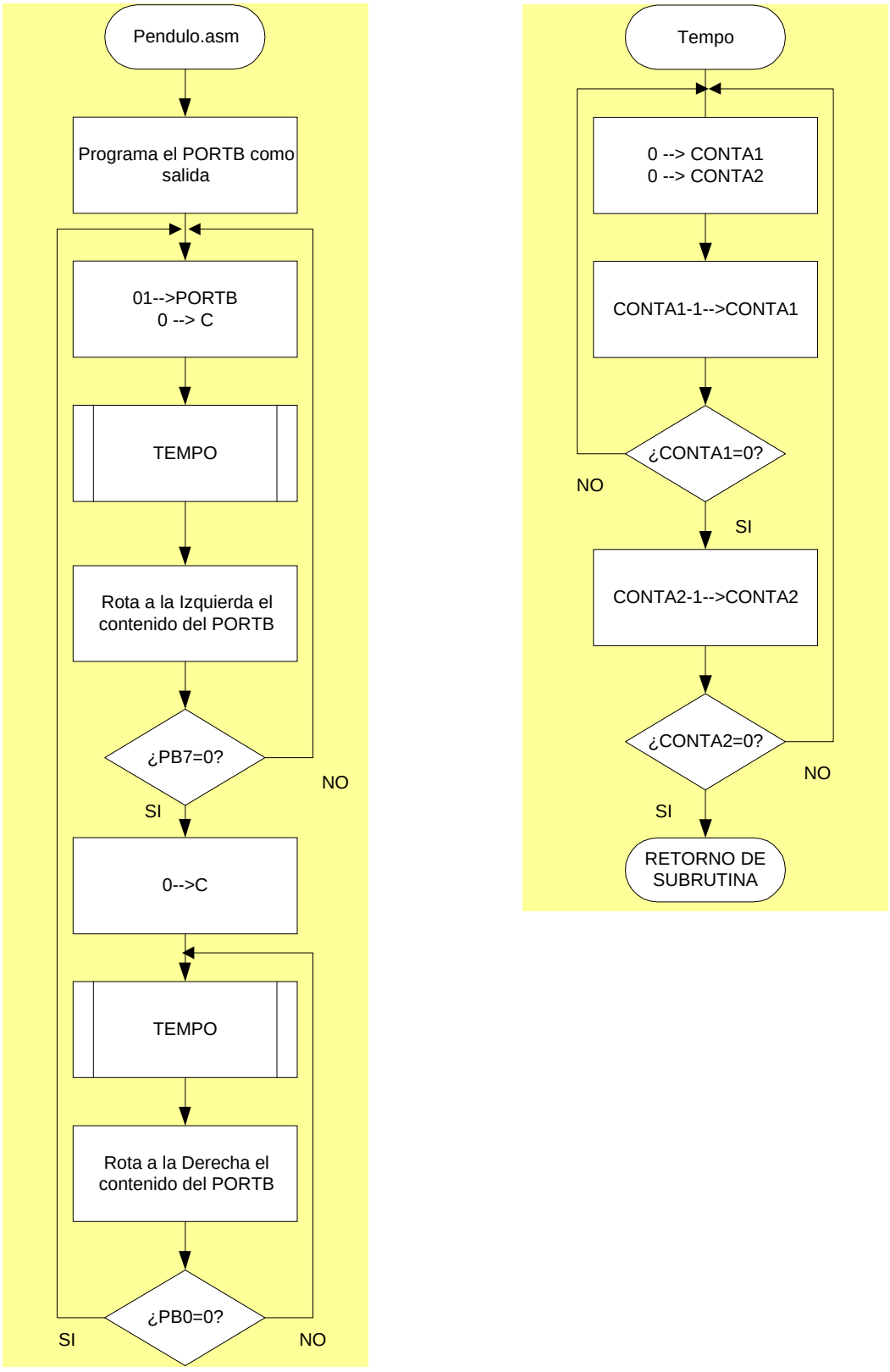


Figura 50.- Organigrama del programa Pendulo.asm y la subrutina TEMPO

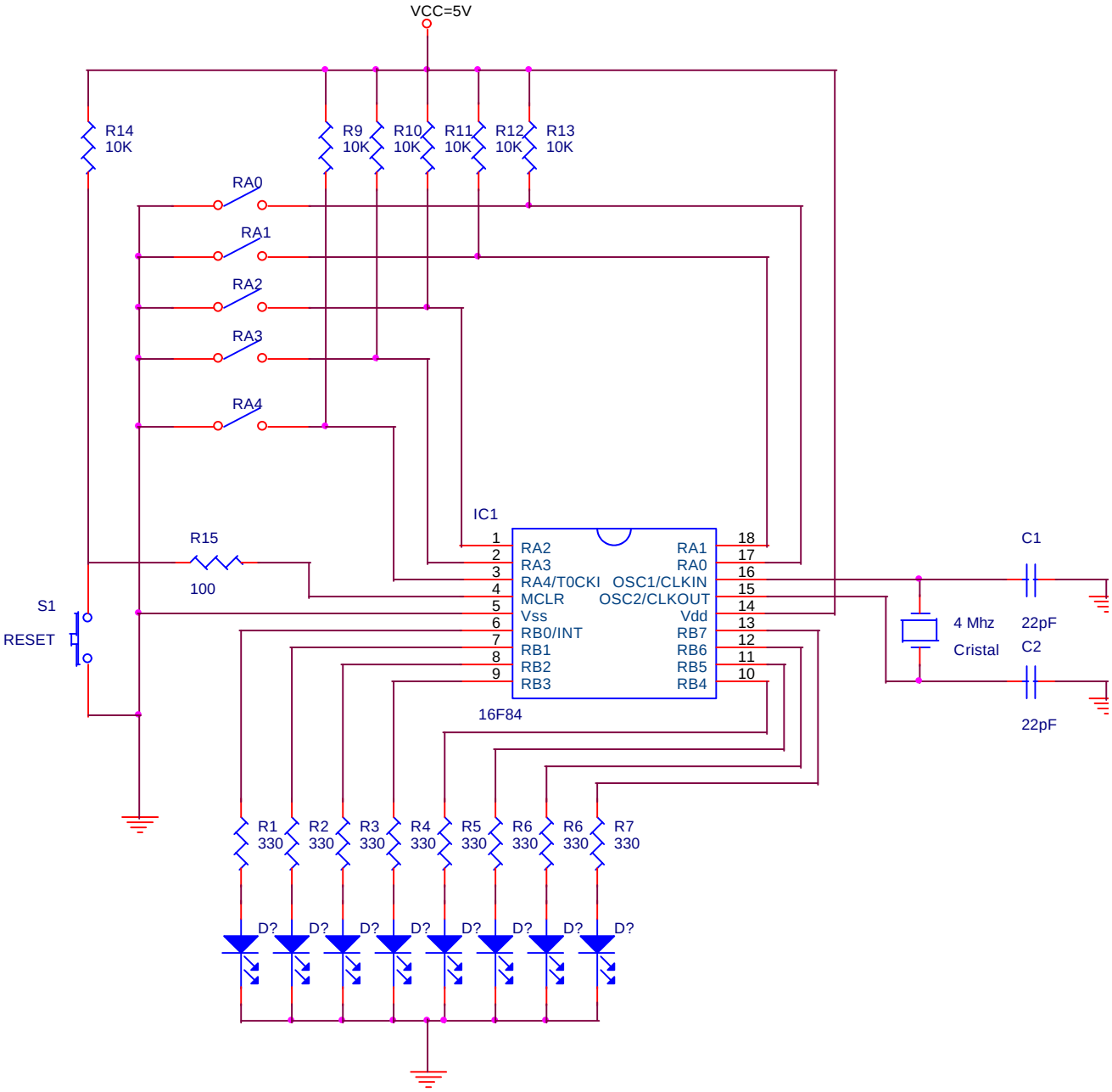


Figura 51.- Circuito para la ejecución de programas con los microcontroladores PIC16F84 y 16C84

En primer lugar comentaremos que en el programa *Pendulo.asm* además de las instrucciones propias del lenguaje ensamblador de los microcontroladores PIC presenta una serie de **Directivas** propias del MPLAB y que permiten mejorar la escritura de los programas. A lo largo de los comentarios que haremos sobre los programas que se presentan en esta serie de artículos iremos describiendo lo que hace cada una de estas Directivas.

Directiva ORG :

Sintaxis:

[<etiqueta> ORG <expr>

Descripción:

Esta directiva especificada en <expr> la dirección de memoria a partir de la que se colocará el código generado. Si se especifica una <etiqueta> se le da el valor de <exp>. Si no hay ningún ORG especificado, la generación del código comienza en la dirección cero.

Directiva EQU – Define una constante para el ensamblador

Sintaxis:

<etiqueta> EQU <expr>

Descripción:

Esta directiva permite asignar el valor de <expr> a un identificador <etiqueta>. El resultado puede ser el resultado de una expresión compuesta por otros identificadores y tan compleja como se desee.

Generalmente, el identificador es un nombre que describe el valor de manera más significativa para el programador. Suele utilizarse para definir constantes y direcciones de memoria. Así, es más fácil recordar SEG_POR_HORA que recordar el valor 3600 o en el caso de una dirección de memoria TRISA que 0x05.

Directiva END

Sintaxis*:*

END

Descripción*:*

Esta directiva indica el final del programa y es obligatoria. Si se detecta el fin de fichero y no se ha encontrado la directiva END se produce error. Todas las líneas posteriores a la línea en la que se encuentra esta directiva, se ignoran y no se ensamblan.

Directiva LIST

Sintaxis:

LIST [<lis_option>,……,<lis_option>

Descripción*:*

Esta directiva es única, es decir, solo puede utilizarse una vez en cada programa y tiene efecto solo sobre el archivo de extensión **.lst* , que es un fichero listable. Además puede ir

acompañada de las siguientes opciones, que controlan la estructura del proceso de ensamblado o del archivo *.lst:

Opción	Por defecto	Descripción
b=nnn	8	Espacios de tabulación
c=nnn	132	Fija la anchura de las columnas
f=<format>	INHX8M	Fija el fichero hexadecimal de salida . <format> puede ser INHX32, INHX8M, o INHX8S.
Free	FIXED	Usa el analizador de formato libre. Suministra la compatibilidad hacia atrás
Fixed	FIXED	Usa el analizador de formato fijo
Mm=on/off	On	Imprime el mapa de memoria en un fichero tipo listado.
n=nnn	60	Fija las líneas por página.
P=<tipe>	Ningún tipo	Fija el tipo de procesador; por ejemplo , PIC16F84
R=<radix>	hex	Pone por defecto el RADIX: hex, dec, oct
St=ON/OFF	On	Imprime la tabla de símbolos en un fichero tipo listado
t=ON/OFF	Off	Corta las líneas de listado (oculta)
w=0|1|2	0	Fija el nivel de mensaje. Ver ERRORLEVEL (nivel de error)
x=ON/OFF	On	Activa o desactiva la expansión de macro

Directiva #INCLUDE

Sintaxis:

INCLUDE <<include_file>>

INCLUDE “<include_file>”

Descripción:

El archivo especificado se lee en como código fuente. El efecto es igual que si el texto entero del archivo INCLUDE se pusiera aquí. Al final del archivo, el código fuente al ensamblar asumirá el archivo fuente original. Se permiten seis niveles de anidamiento. El < include_file> puede escribirse entre comillas o entre los símbolos de mayor que y menor que (< >). Si se especifica totalmente el camino del fichero include, sólo se buscará en dicho. Si no se indica camino, el orden de la búsqueda es: el directorio activo actual, el directorio de archivo fuente, el directorio ejecutable de MPASM.

Ejemplo

```
INCLUDE <p16f84.inc>          ;define el archivo donde están definidos todos los
                              ; registros del PIC16F84
INCLUDE “p16f84.inc”          ; también se puede definir de esta forma
INCLUDE “c:\sys\system.inc”   ; define system con su camino
INCLUDE <regs.h>              ;define regs.h
```

La directiva INCLUDE no se utiliza en este programa, pero si en los siguiente, para comprobar como actúa, se recomienda sustituir las líneas comprendidas entre los asteriscos que indican las *Igualdades que designa los destinos* y las *Igualdades que designan los registros SFR* por la línea INCLUDE "P16F84.INC" y volver a ensamblar el programa, comprobando que no se produce ningún error, para ver este fichero podemos buscarlo en el MPLAB activado *File>Open* , lo que hará que aparezca el cuadro de diálogo de la Figura 52, sobre el seleccionamos la carpeta C:\Archiv~1\MPLAB y en Mostrar archivos del tipo: **.h* y **.inc*. Por último, seleccionamos el archivo *p16f84.inc*. Estas acciones abrirán el archivo que hemos insertado mediante la directiva INCLUDE, y como puede verse presenta las definiciones de todos los registros y bit que componen estos.

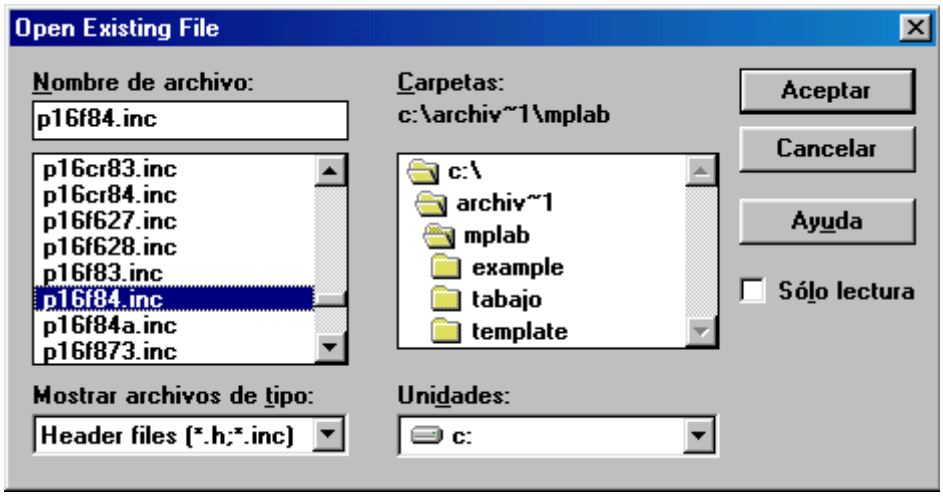


Figura 52.- Selección del archivo p16f84.inc para su visualización.

En primer lugar creamos un nuevo proyecto que llamaremos "Pendulo.pjt". Una vez que hayamos escrito el código del programa será necesario ensamblarlo y compilarlo. Si se ha cometido algún error en la escritura en MPLAB lo indicará. Si así fuera, bastará con hacer doble clip sobre la línea que muestra el error para que nos lleve directamente a la línea de código donde lo hemos cometido.

Seguidamente abrimos en el escritorio las siguientes ventanas:

- La del programa ensamblado ***Fille>Open>pendulo.asm***
- La de los registros especiales ***Windows>Special Function Registers***.o activando el icono 
- La del reloj del procesador ***Windows>Stop Watch***

Ahora utilizando los *Break Points* podemos medir por ejemplo el tiempo que tarda en ejecutarse la subrutina TEMPO, para ello ponemos un *Break Point* en la primera línea de la subrutina “TEMPO clrf CONTA1”, posicionando el ratón sobre ella, pulsando el botón de la derecha y seleccionando la opción *Break Point*, lo mismo hacemos sobre la instrucción “Return”, unas líneas mas abajo. Seguidamente pasamos a simular el funcionamiento del programa, para ello hacemos primero un *Reset* y después seleccionamos *Debug>Rum>Rum*, y cuando llegue al programa encuentre el primer *Breack Point* se parará, ahora ponemos a *Zero* el reloj del sistema *Stop Watch* y volvemos a ejecutar el programa con *Debug>Rum>Rum*. Esta opción hará que el programa corra sin refrescar el display, pero veremos como el contador de ciclos del sistema se va incrementando, al llegar al siguiente Break Point, la simulación se detendrá y el reloj indicará el tiempo que ha tardado en ejecutarse el programa, que en este caso es de 197,12 ms o 197.121 ciclos de reloj. Ver la Figura 53.

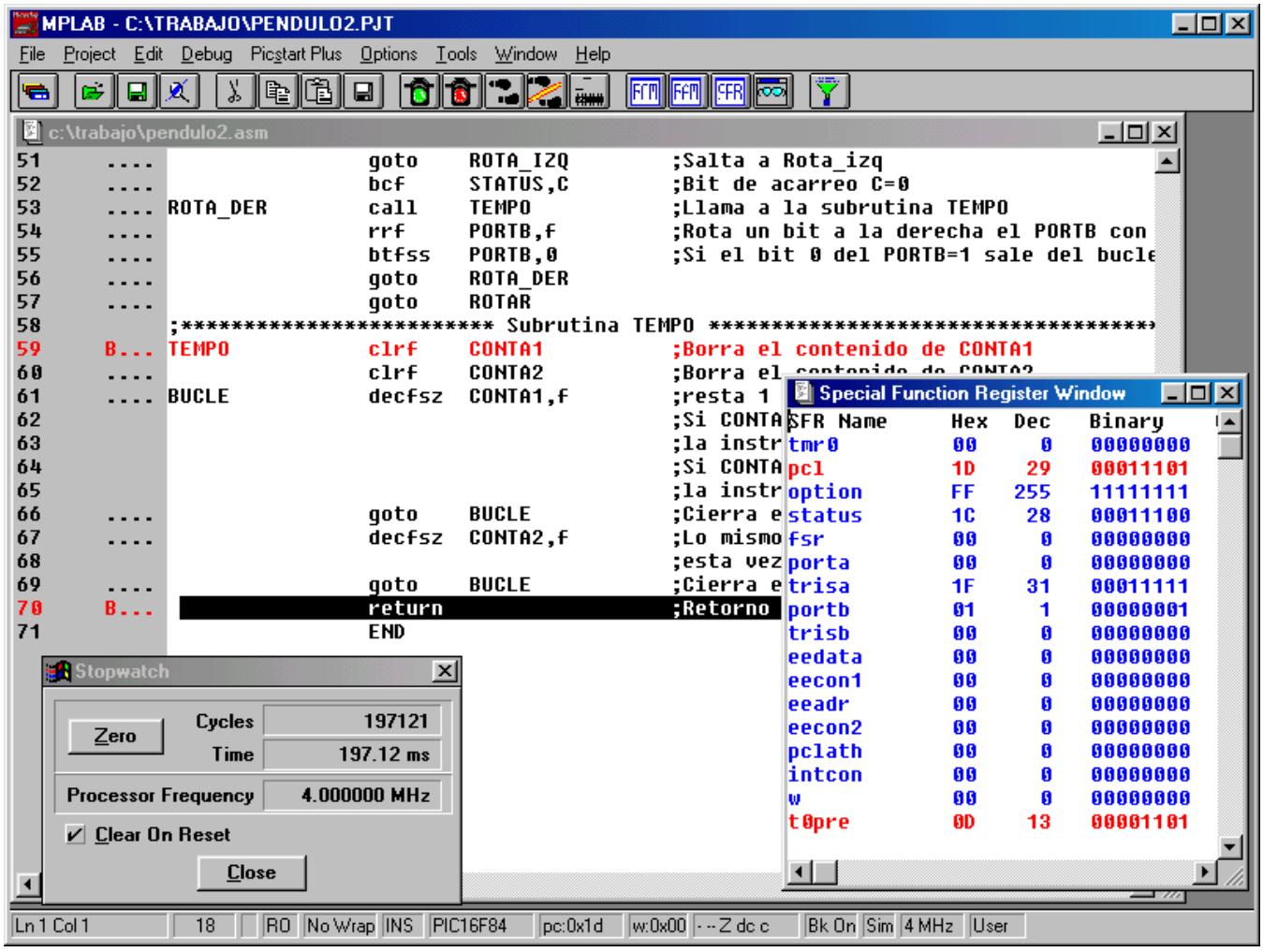


Figura 53.- Pantalla de simulación de la subrutina TEMPO en el programa pendulo.asm

Como hemos podido comprobar el tiempo que el MPLAB ha tardado en simula la temporización es muy largo, una vez que hemos comprobado esto, podemos simular el funcionamiento del programa pero disminuyendo el tiempo de la subrutina TEMPO. Para ello, después de la etiqueta TEMPO ponemos directamente la instrucción “return” y volvemos a ensamblar el programa, esto provocará que cada vez que saltamos a TEMPO en el siguiente ciclo máquina se ejecute la instrucción de retorno de subrutina y siga ejecutando el programa principal.

Manejaremos ahora los *Break condicionales*, se trata de hacer que la simulación se detenga cuando el PortB tenga el valor binario 0001000 = 10 h y llegue a la etiqueta TEMPO, para ello seleccionamos *Debug>Execute>Conditional Break* y se abre un cuadro de diálogo como el de la Figura 48, en primer lugar seleccionamos las opciones *Multiple Cycles y Update Display* de la parte superior y en la ventana de condiciones buscaremos la de igualdad. Después buscaremos en la de registros, el registro PORTB y ponemos el valor deseamos para él, en este caso 0x10. Seguidamente activamos el switch de *Trace Data*, buscamos el registro PORTB y pulsamos sobre *Add*. De esta forma hemos elegido el registro que deseamos ir viendo en la ventana según vaya evolucionando la ejecución del programa. Nos falta fijar el punto donde queremos que el programa se pare cuando el registro PORTB llegue al valor 0x10, para lo cual pulsamos el botón de ***Break Settings***. La ventana que aparece es la de la Figura 49 que es la correspondiente a ***Break Point Settings,*** en ella buscamos la etiqueta TEMPO en ***Start*** y hacemos lo mismo en el campo ***End***. Seguidamente pulsamos sobre ☑ para habilitar los break points y cerraremos esta ventana pulsado el botón de ***Close***.

Para iniciar la ejecución activamos el botón de ***Start***, y podemos ver como aparecen los distintos valores que toma el registro PORTB hasta que llega al valor 0x10 que se detiene. En la Figura 54, se presenta la secuencia de pantallas que se acaban de describir.

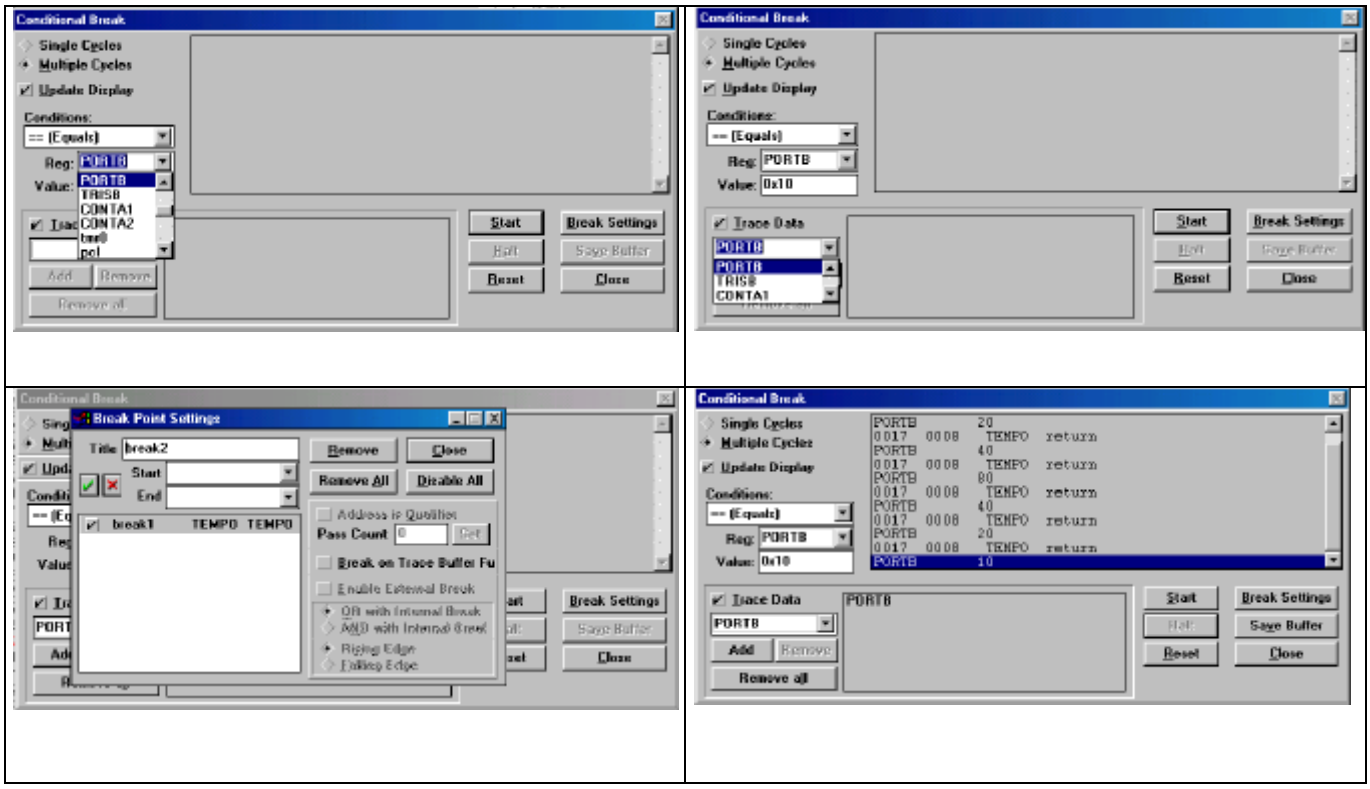


Figura 54.- Secuencia de pantallas para fijar y simular la Condición de Break

2.3.- Simulator Stimulus

La Función de simulación de estímulos simula la generación de señales de entrada en las patillas del microcontrolador. Los cuales se pueden poner a nivel alto o bajo, además de inyectar los valores directamente en los registros. Hay cuatro modos de generar los estímulos tal y como se aprecia en la Figura 55 y cuyas acciones de forma abreviada son las siguientes:

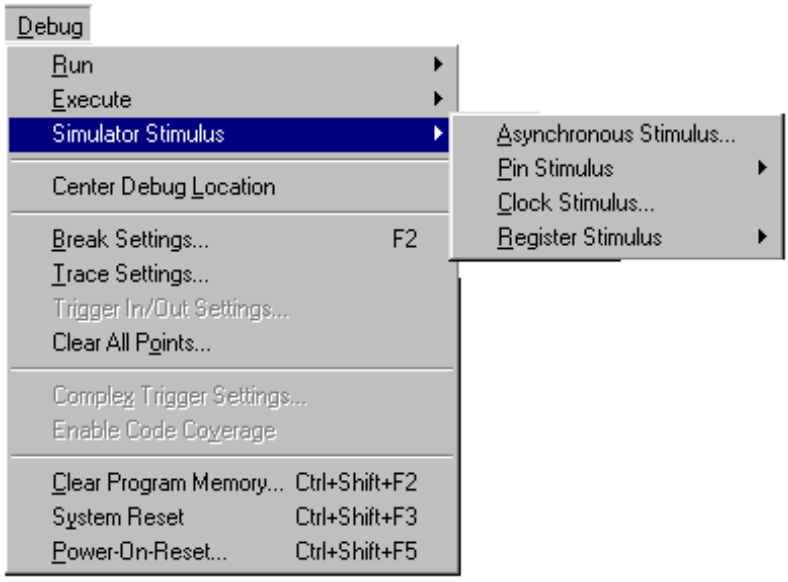


Figura 55.- Cuadro de diálogo con las opciones de simulación de señales de entrada.

- *Asynchronous stimulus* (Estímulo Asíncrono): Abre un cuadro de diálogo interactivo para controlar las señales de entrada en los pines de entrada del microcontrolador.
- *Stimulus Pin File* (Archivo de estímulos en los pines): Mediante un archivo del texto se describen las señales de entrada en los pines.
- *Stimulus Register File* (Archivo de Registro de Estímulos): Se usa el contenido de un archivo de texto para poner directamente a uno o cero cada uno de los 8 bits de un registro.
- *Clock Stimulus*: (Señales de reloj). Genera una señal cuadrada programable.

Veamos las opciones que presentan cada una de los modos de generar estímulos.

2.3.1.-Debug>Simulator Estimulus>Asynchronous Stimulus: Al activar esta opción emerge un cuadro como el de la Figura 56 en el que aparecen doce botones que son el número de líneas que tienen los dos puertos del 16F84, es decir, las 8 líneas del *PORTB* y las 5 líneas del *PORTA*.

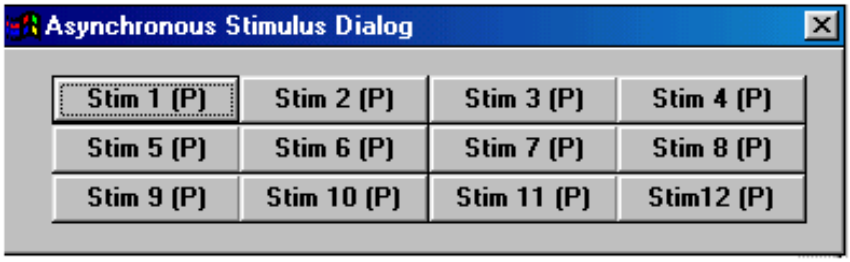


Figura 56.- Cuadro de asignación de pines y estímulos asíncronos.

Si activamos, con el botón de la derecha del ratón, cualquiera de los botones de estímulos, por ejemplo *Stim 1(P)*, aparece una ventana como la de la Figura 57.

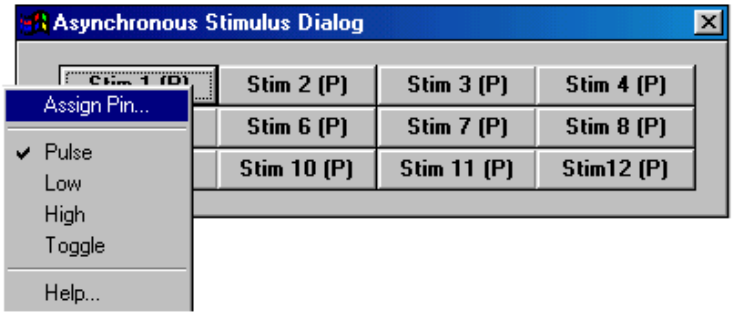


Figura 57.- Ventana de asignación de Pines y tipo de impulsos.

Seguidamente si seleccionamos ***Assign Pin..*** aparece un menú como el de la Figura 58 en el que hay un listado con de todos los pines de entrada del microcontrolador.

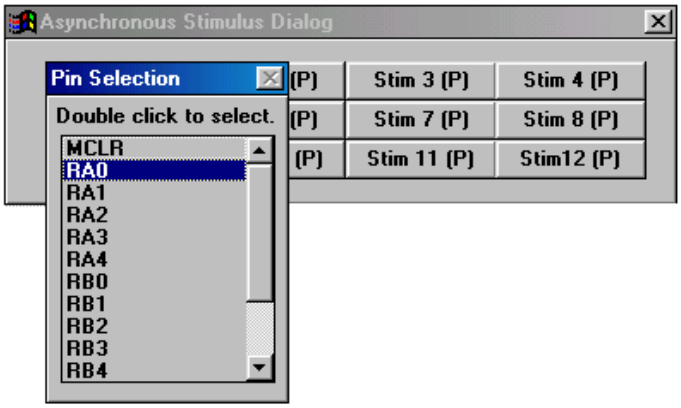


Figura 58.- Cuadro de diálogo de asignación de pines

Realizando los pasos anteriores se pueden asignar el resto de los pines. La siguiente acción será indicar al simulador el tipo de señal de entrada que vamos a producir cada vez que se pulse en el botón correspondiente. Para ello en el menú de la Figura 57 se selecciona cualquiera de las siguientes posibilidades:

- ***Pulse*** : un solo impulso
- ***Low*** : un nivel bajo
- ***High*** :un nivel alto
- ***Toggle*** :cada vez que se pulsa cambia de estado y permanece en ese valor hasta que se vuelve a pulsar, entonces genera una señal contraria, es decir si tenia +5V pasa a 0V y viceversa.

Para comprobar lo dicho hasta este momento, proponemos un nuevo programa que consiste en leer el valor de las entradas del *PORTA* y mostrar dicho valor en el *PORTB*. El programa en lenguaje ensamblador es el siguiente. Como en el caso anterior puede ejecutarse utilizando los módulo 01 y 02 o bien utilizando el circuito de la Figura 51.

```
;*******************************************************************************
; Programa PortA_B.ASM          Fecha : 16 - Noviembre  - 99
; Este programa lee el estado de los 5 conmutadores de RA4 - RA0  y refleja
; el nivel lógico de los mismos sobre los leds RB4-RB0 conectados a la Port B
; Revisión : 0.0                Programa para PIC16C84 y PIC16F84
; Velocidad del Reloj: 4 MHz    Reloj Instrucción: 1 MHz = 1 µS
; Perro Guardián:  deshabilitado  Tipo de Reloj : XT
; Protección del código: OFF
;*******************************************************************************
            List    p=16F84         ;Tipo de procesador
            include "P16F84.INC"    ;Definiciones de registros internos
            ORG     0x00            ;Vector de Reset
            goto    Inicio
            ORG     0x05            ;Salva el vector de interrupción
Inicio      clrf    PORTB           ;Borra los latch de salida
            bsf     STATUS,RP0      ;Selecciona banco 1
            clrf    TRISB           ;El PortB se configura como salida
            movlw   b'00011111'
            movwf   TRISA           ;El PortA se configura como entrada
            bcf     STATUS,RP0      ;Selecciona el banco 0
Bucle       movf    PORTA,W         ;Leer las entradas RA0-RA4
            movwf   PORTB           ;Reflejar en las salidas
            goto    Bucle           ;Bucle sin fin
            END                     ;Fin del programa fuente
```

En primer lugar crearemos un nuevo proyecto como ya se ha descrito en esta publicación, en este caso lo llamaremos “PortA_B.pjt”. Una vez que hayamos escrito el código del programa será necesario ensamblarlo y compilarlo.

Seguidamente seleccionamos la opción ***Debug>Simulator Estimulus>Asynchronous Stimulus*** donde asignaremos los pines RA0 (T), RA1(T),RA2(T), RA3(T) y RA4(T) a los cinco primeros *Stim* y el tipo de señal *Toggle* y habilitamos la ventana de los registros SFR, tal y como se muestra en la Figura 59.

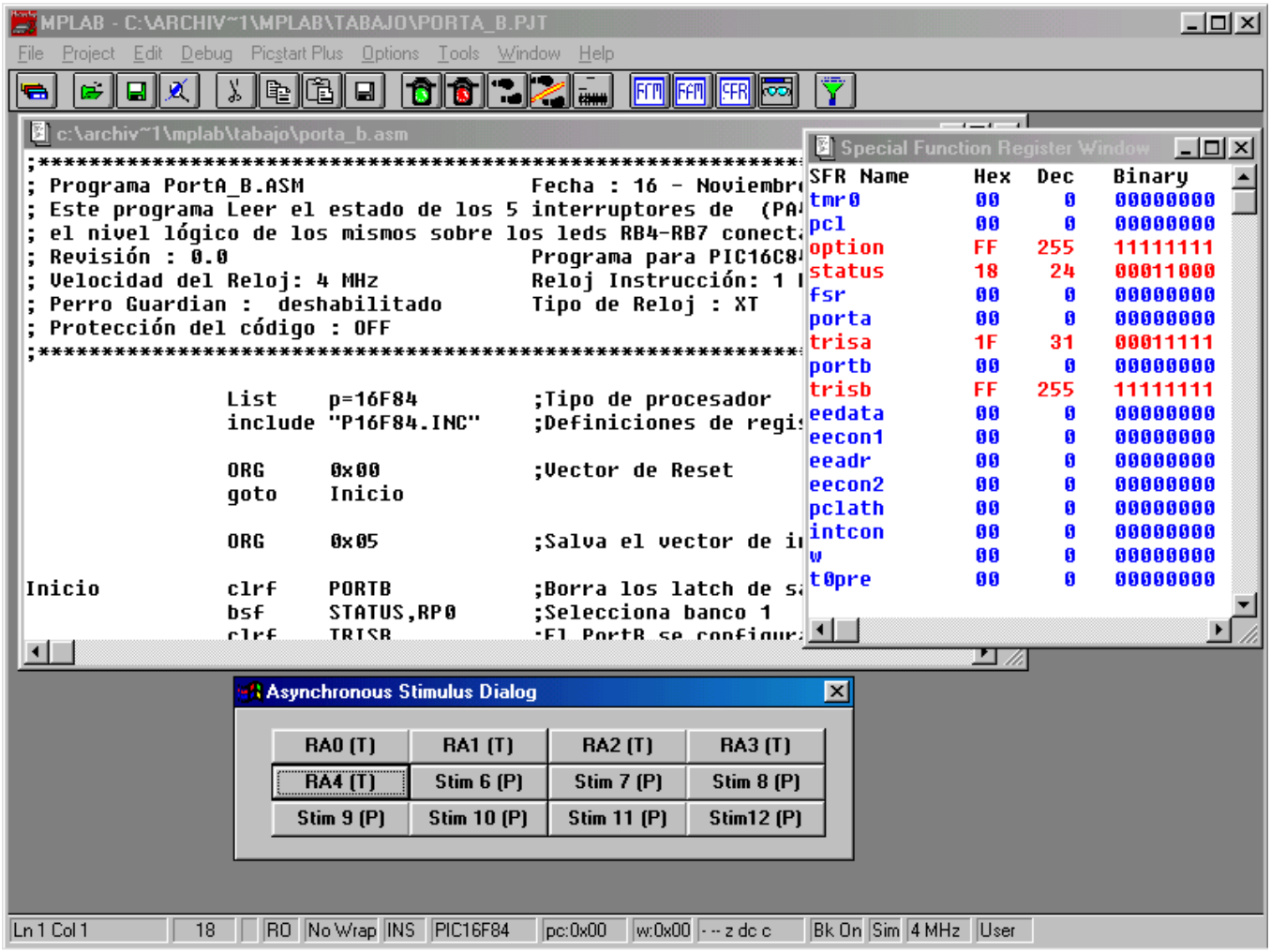


Figura 59.- Pantalla dispuesta para simular el programa "PortA-B.asm"

Si ahora seleccionamos la opción ***Debug>Run>Animate*** o la combinación de teclas [Control]+[F9], el programa se ejecutará y veremos como se van actualizando los registros del microcontrolador. Si se pulsa el botón que hemos asignado como *RA0* se verá como primero cambia el contenido del *PortA*, independientemente de la instrucción que se esté ejecutando (si no se puede apreciar en el modo *Animate*, hacerlo en el modo *Step*). Cuando se ejecuta la instrucción *movf PORTA,W* el registro *W* toma el valor 00000001 y al ejecutarse la instrucción *movwf PORTB* se puede comprobar que el PortB toma el valor 00000001 que coincide con el valor que tiene en ese momento el *PortA*. Si se activa cualquiera de las otras líneas de entrada se aprecia que las salidas del *PortB* van cambiando también de acuerdo con el valor de las entradas del *PortA*.

2.3.2.- Debug>Simulator Stimulus>Pin Stimulus: Un archivo de estímulo de pines consiste en columnas de unos y ceros de entrada que serán aplicados a los pines correspondientes cuando la columna del "Ciclo" coincida con el valor en el cronómetro de CICLO de la simulación del programa.

Para comprobar como funciona de esta opción del simulador activar *File>New File* y teclear el siguiente texto, no hace falta escribir el texto que hay después del ";" y "!" ya que estos signos delimitan los comentarios.

```
CYCLE       RA1         RA0
20          0           0
41          1           0       ; aplica 1 en el bit1 del PortA
52          0           1       ; aplica 1 en el bit 0 del Port A y un cero en el bit 0 del PortA
55          1           1
60          0           0
65          1           0       ; toggle bit 1, entonces...
76          0           1       ! ...toggle bit 0.
```

Utilizar la opción ***File>Save As...*** para guardar el archivo con el nombre ***PortA_B.sti***.

Después de la palabra **CICLO** en la primera línea del archivo se escriben los nombres con que se han designado los pines del microcontrolador que recibirán los estímulos de entrada, niveles altos y bajos. En este el ejemplo hemos fijado RA1 y RA0 como dos entradas del *PortA*.

En este archivo, la segunda columna contiene valores que se aplicarán a RA1 (bit 1 del *PORTA*) y en la tercera columna los valores para RB0 (bit 0 del *PORTB*). Éstos nombres deben coincidir con los que se han fijado para el microcontrolador que se va a simular. Se

puede ver un listado de todos los pines en el menú desplegable que aparece en la Figura 58 para los estímulos asíncronos.

Para realizar la simulación es aconsejable visualizar también el reloj que se obtiene activando ***Windows>Stopwatch***, de manera que, además del reloj, podamos visualizar en la misma pantalla el archivo de estímulos, el programa escrito en ensamblador y los FSR tal y como se puede ver en la Figura 61.

Antes de realizar la simulación deberemos de activar la opción ***Debug>Simulator Stimulus>Pin Stimulus>Enable***, el programa presenta entonces un cuadro de diálogo como el de la Figura 60 en el que debemos de seleccionar el archivo de estímulos ****.sti***, en nuestro caso *PortA_B.sti*

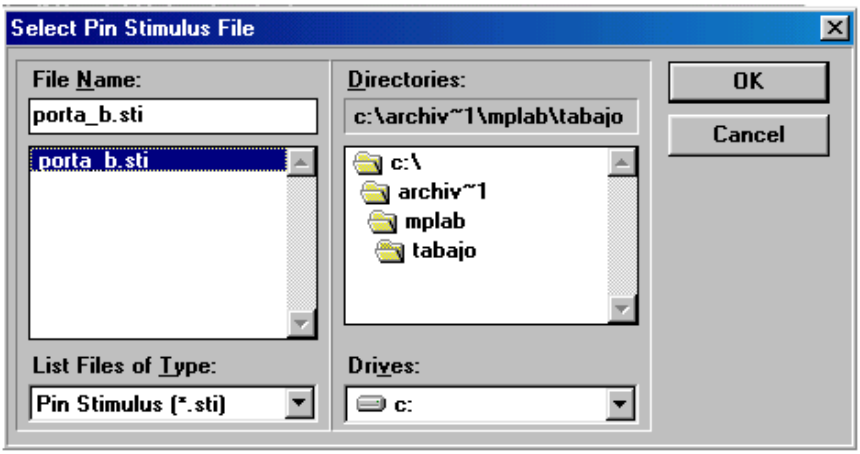


Figura 60.- Cuadro de diálogo para seleccionar el archivo de simulación de estímulos.

Durante la simulación, cada vez que se pone a cero el reloj el archivo de estímulos también se reinicializa comenzando desde el primer ciclo de reloj.

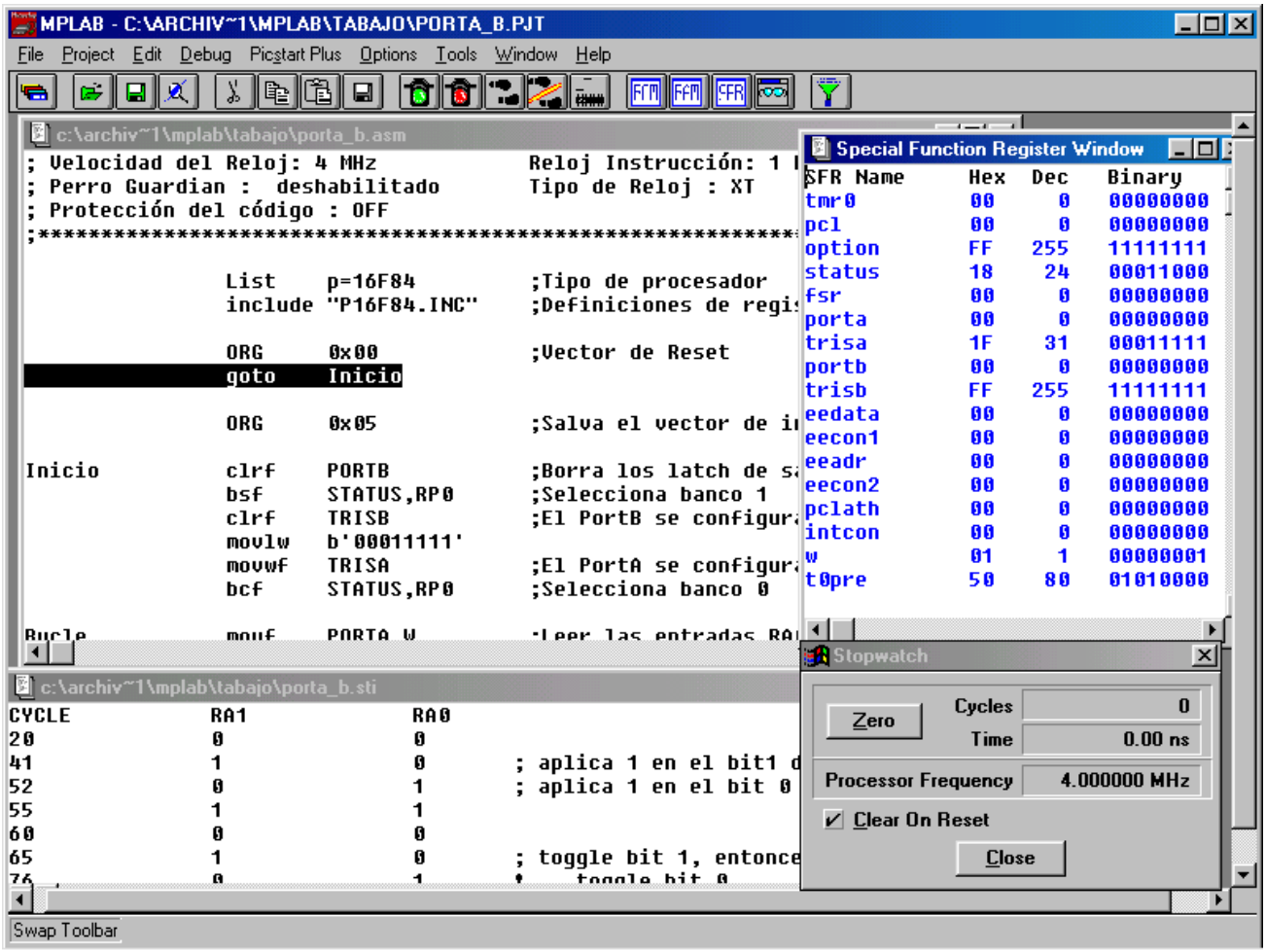


Figura 61.-Pantalla de simulación con las ventanas del programa en ensamblador, el archivo de impulsos, SFR y reloj del sistema

2.3.3.- Debug>Simulator Stimulus>Register Stimulus:

Un archivo de "estímulo de registros" consiste en una sola columna de valores que se enviarán a un registro cuando la dirección de memoria de programa alcance la situación del cuadro de diálogo del Registro de Estímulos. Esto es útil para simular el funcionamiento por ejemplo de un convertidor A/D.

Para crear este archivo abrir un nuevo archivo con ***File>New File*** y teclear el siguiente listado de valores:

```
01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
00
```

Salvar el archivo con ***File>Save as...***. y nombre de *PortA_B.reg*. Este archivo se usará para inyectar estos valores secuencialmente en un registro.

Seguidamente seleccionar la opción ***Simulator Stimulus> Register Stimulus>Enable...***, aparece un cuadro de diálogo como el de la Figura 62, en el cual se define el registro en el que queremos que vayan cargándose los datos del fichero *PortA_B.reg*, por ejemplo la posición de memoria RAM 0x0d, y cuando queremos que esto ocurra, por ejemplo cada vez que se pase por la posición de memoria que corresponde con la etiqueta Bucle.

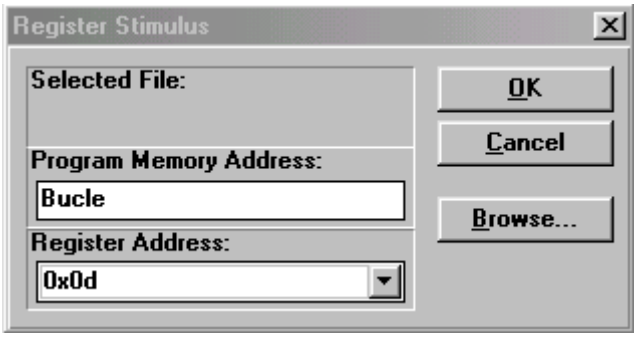


Figura 62.- Cuadro de diálogo del Registro de Estímulos.

Seguidamente se carga el archivo para lo cual se selecciona el botón de ***Browse..*** , aparezca un cuadro de diálogo como el de la Figura 63, donde seleccionamos el archivo *PortA_B.reg*.

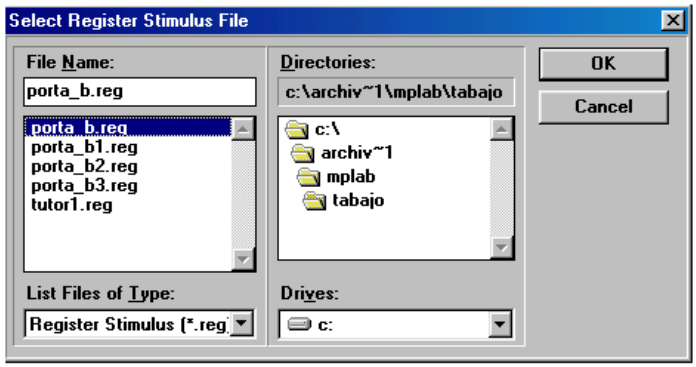


Figura 63- Selección del archivo de estímulos de registro

Para comprobar el funcionamiento de lo dicho hasta el momento

1. Presentar una pantalla en el escritorio del MPLAB en el que se visualicen como se muestra en la Figura 64 las siguientes ventanas:

- El programa en ensamblador
- El archivo ***.reg**
- Los registros SFR
- La memoria RAM
- El reloj

2. Seguidamente activar ***Debug>Run>Step*** y ejecutar paso a paso el programa comprobando como el registro 0x0d se va cargando con los valores 0x01, 0x02, 0x03, 0x04,…, cada vez que el programa ejecuta la instrucción que se encuentra en la dirección de memoria *Bucle*.

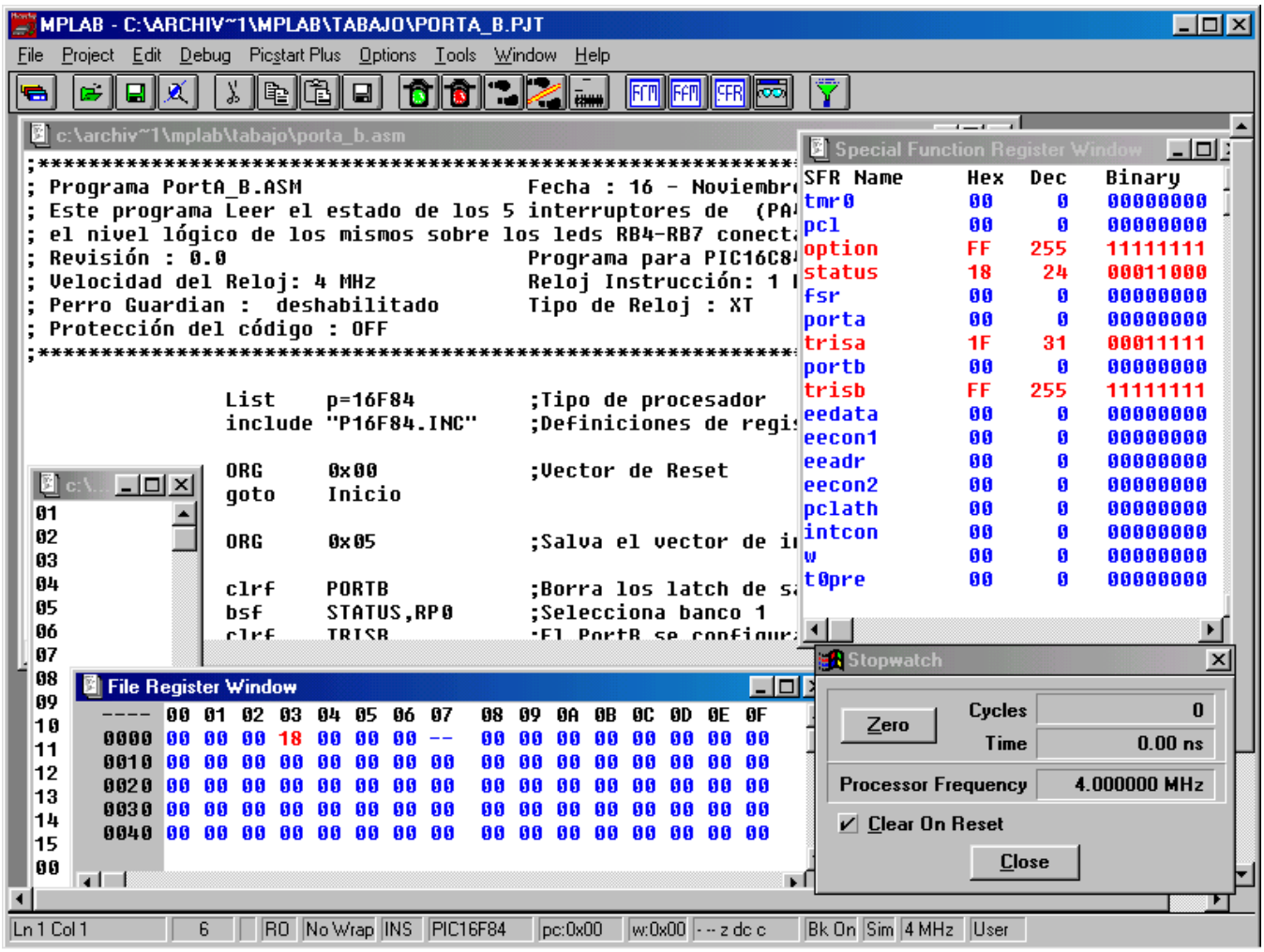


Figura 64.- Pantalla de simulación en la que se pueden ver las ventanas del programa en ensamblador, la memoria RAM, los SFR, el reloj y el fichero de estímulos de registros portA_B

2.3.4.- Debug>Simulator Stimulus>Clock Stimulus

Clock stimulus genera un tren de impulsos periódico en un determinado pin con un ciclo de trabajo referido a los ciclos de trabajo del microcontrolador.

Si se selecciona la opción ***Debug>Simulator Stimulus>Clock Stimulus...*** aparece un cuadro diálogo como el de la Figura 65 donde se pueden introducir varias entradas de estímulos. Si se hace la simulación en modo *Step* o en modo *Animate* usando las de la Figura 65, RA3 estará a nivel alto durante 4 ciclos del reloj, posteriormente estará a nivel bajo durante 6 ciclos del reloj. *RA4* estará a nivel alto durante 8 ciclos del reloj y a nivel bajo durante 8 ciclos del reloj.

Las dos secuencias se repiten hasta que se termine la ejecución en el *MPLAB* o se anule este modo de generar los estímulos de entrada.

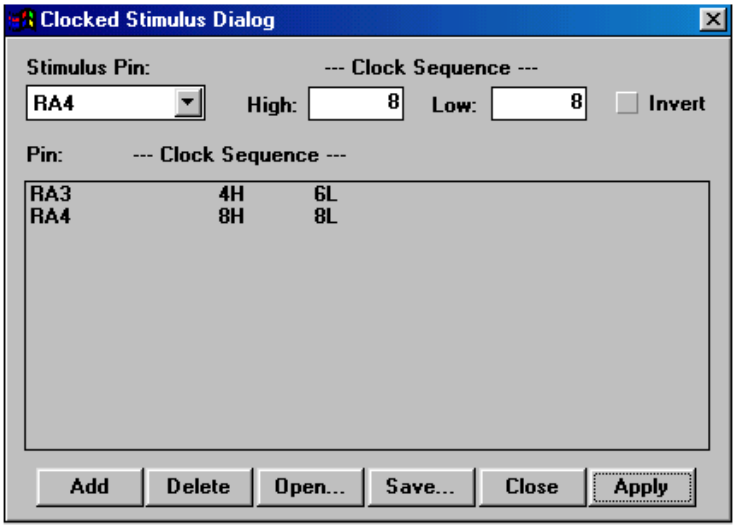


Figura 65.- Cuadro de diálogo para introducir trenes de impulsos por las líneas del microcontrolador.

Seguidamente activar el botón de *Apply* , si ahora se selecciona la ejecución del programa en modo paso a paso (*Step*) o en modo continuo (*Animate*), podemos apreciar como se van introduciendo los impulsos que se han determinado en el cuadro de diálogo de la Figura 65.

2.4.- Center Debug Location

La opción ***Seleccione Debug > Center Debug Location*** pone en el centro de la pantalla de depuración del programa la posición de memoria que indica el PC .

Esta función trabaja solo con las Ventanas de:

- Código de Fuente del programa
- La de Programa de Memoria
- La de *Absolute Listing*.

2.5.- Break Settings

Seleccione ***Select Debug > Break Settings*** para desplegar un cuadro de diálogo de *Breack Point* como el de la Figura 66 en el que se pueden definir hasta 16 rangos de nombre para los *Breack Point*

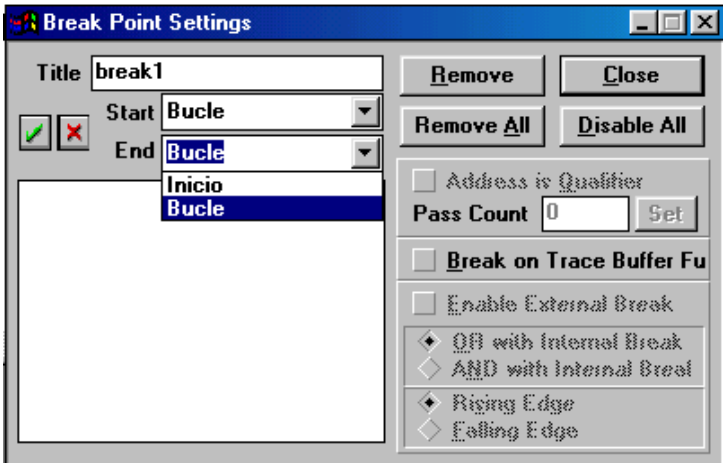


Figura 66.- Cuadro de diálogo de selección de los *Breack Point*

Después de introducir en un nombre de *Breack Point*, la dirección de inicio y la de fin, hacer clic en o pulsar botón de ***Enter*** para aceptar la definición del mismo. Esto hará resaltar las líneas comprendida entre las dos posiciones de memoria (la de inicio y la de fin) en rojo, de manera que cuando se esté simulando el programa este se pare en dichas líneas.
Esta opción también activarse pulsando la tecla [F2].

2.6.- Trace Settings

Esta opción es similar a la anterior pero en este caso, en lugar de *Breack Point* se activan las direcciones de memoria de *Traza* que se quiere rastrear posteriormente.

2.7.- Clear All Points

Al selecciona ***Debug > Clear All Points*** se eliminan todos los *Breack Point*, trazas y puntos de disparo seleccionados cuando se activa el botón de *Yes* del cuadro de diálogo que aparece como el de la Figura 67.

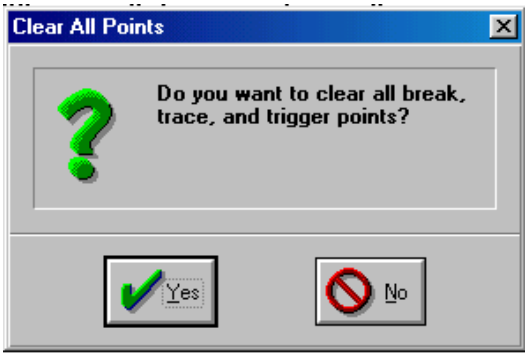


Figura 67.- Cuadro de selección de limpiar todos los *Breack Point, Traza y puntos de disparo*.

2.8.- Clear Program Memory (Ctrl+Shift+F2)

Si seleccionamos ***Debug > Clear Program Memory*** se limpia la memoria de programa poniéndola a 0x3FF. Esta función pone todos los bits de memoria de programa a unos, cuando se activa el botón de *Yes* del cuadro de diálogo que aparece como el de la Figura 68

Figura 68.- Cuadro de selección de limpiar todas las posiciones de memoria de programa poniéndola toda a unos.

2.9.- System Reset (Ctrl+Shift+F3)

Seleccionando ***Debug > System Reset*** se inicializa el emulador sistema incluido el hardware del emulador de MPLAB-ICE (si está conectado), el software y el microcontrolador. *System Reset* realiza la misma operación que cuando se inicializa el MPLAB. Para realizar un *Reset* (*MCLR*), seleccionar ***Debug>Run>Reset***.

2.10.- Power-On Reset (Ctrl+Shift+F5)

Seleccionando ***Select Debug > Power-On-Reset*** se despliegar el cuadro de diálogo *Power-On-Reset* de la Figura 69 si se activa el botón *Power On Reset* aparece una nueva configuración. Como puede verse la memoria RAM y los registros SFR se cargan con valores

aleatorios, tal y como haría el microcontrolador al realizar un Reset por aplicarle alimentación. La ventana de la memoria RAM puede presentar por ejemplo el aspecto de la Figura 69.

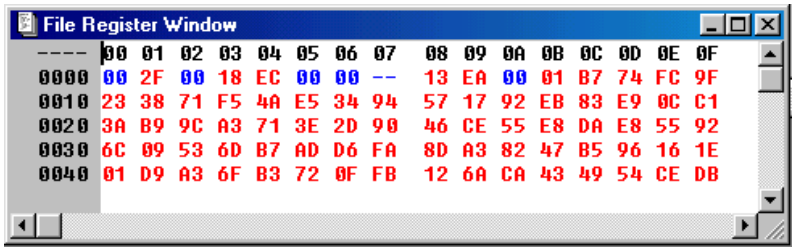


Figura 69.- Aspecto de la memoria RAM después de un Power On Reset.

Además, si se visualizan los registros especiales de la misma manera que en la Figura 70, se puede comprobar que quedan de acuerdo con la siguiente tabla

Registro	Posición de memoria	Valor del registro
PCL	02h	0000h
STATUS	03	0001 1XXX
PCLATH	0Ah	---0 0000
INTCON	0Bh	0000 000X
OPTION	81h	1111 1111
TRISA	85h	---1 1111
TRISB	86h	1111 1111
EECON1	88h	---0 X000

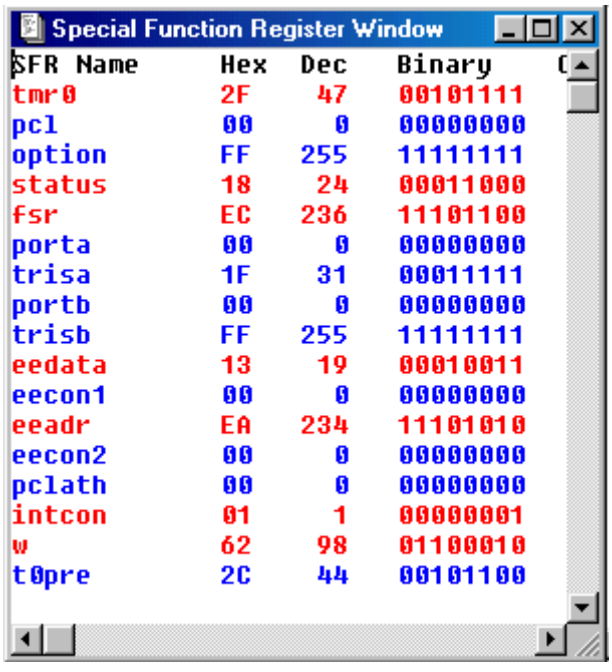


Figura 70. Los registros FSR después de realizar un Power On Reset.

Ejercicios Propuestos:

Llegado a este punto vamos a proponer una serie de ejercicios para que nos permitirán fijar los conocimientos sobre el funcionamiento del programa MPLAB y la programación del microcontrolador PIC16F84.

Ejercicio 1 (Para simular en el MPLAB).- El siguiente programa realiza la suma de dos números de 16 bits que previamente cargaremos en cuatro registros (posiciones de memoria), NUMEROA_L , NUMEROA_H para el primer sumando y NUMEROB_L , NUMEROB_H para el segundo sumando, dejando el resultado en REAUL_L y REUL_H.

En este programa es interesante como se resuelve el problema de no existir en el set de instrucciones la operación de suma con acarreo. En primer lugar se comprueba si se produce acarreo en la suma de las partes bajas de los números, consultando el *flag C* del registro de **STATUS** y si es “1”, se incrementa en una unidad el registro W antes sumar la parte alta de los números a sumar.

El organigrama del programa es el que se muestra en la Figura 71.

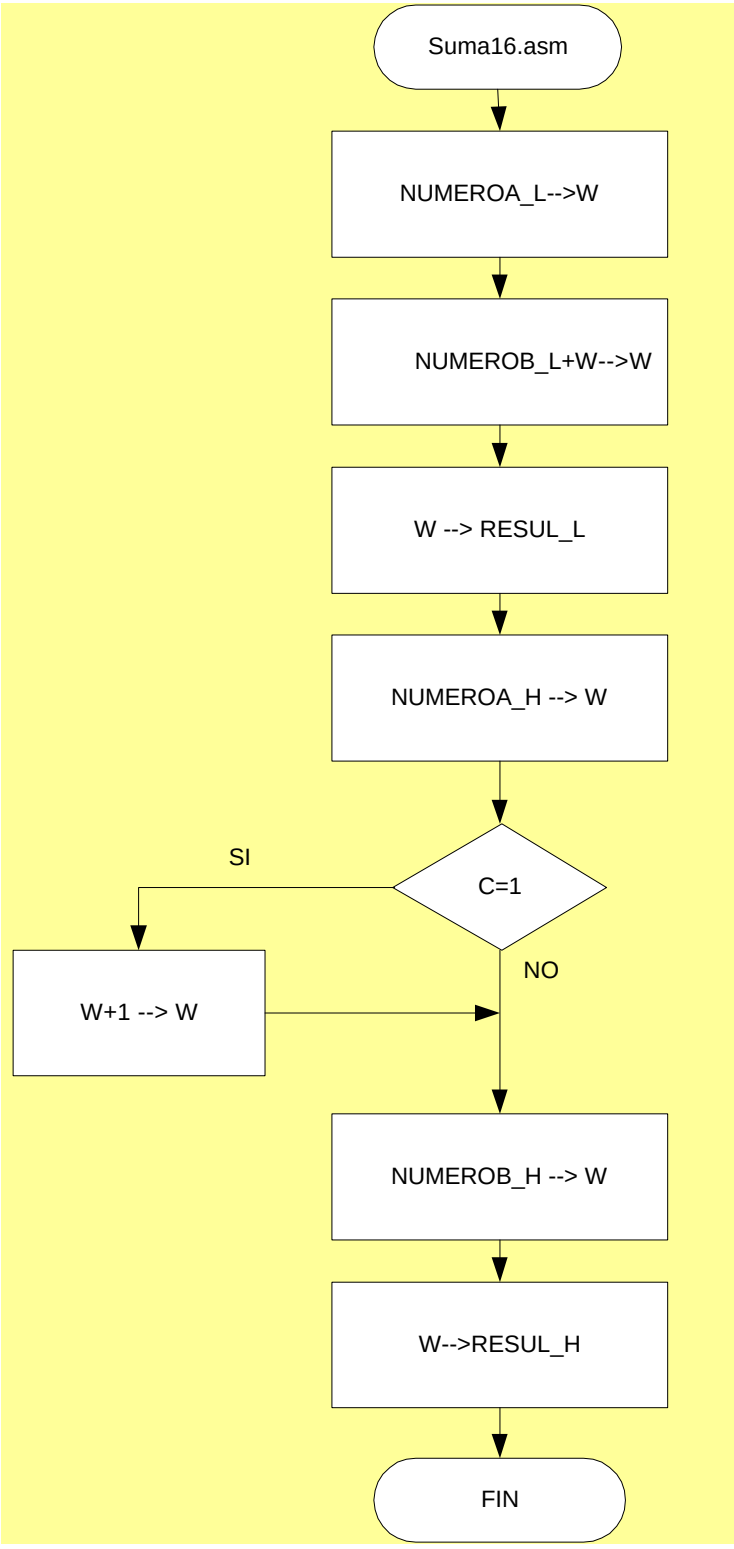


Figura 71.- Organigrama del ejercicio de simulación Suma16

El programa en ensamblador será por lo tanto será el siguiente:

```
;*******************************************************************************
; Programa: Suma16.ASM                  Fecha: 24 - Abril  - 2000
; Este programa realiza la suma de doble precisión, es decir de operandos de operandos de 16 bits
; cada uno, NUMEROA_L,NUMEROA_H con NUMEROB_L,NUMEROB_H que se
; encuentran en las posiciones de memoria 0x10, 0x11 y 0x12, 0x13 respectivamente y
; el resultado se guarda en la posición de memoria 0x14, 0x15 que denominamos RESUL_L
; y RESUL_H respectivamente.
; Los datos en estas posiciones se incluirán durante la simulación del programa
; Revisión : 0.0                        Programa para PIC16C84 y PIC16F84
; Velocidad del Reloj: 4 MHz            Reloj Instrucción: 1 MHz = 1 uS
; Perro Guardián: deshabilitado         Tipo de Reloj: XT
; Protección del código: OFF
;*******************************************************************************
                List    p=16F84             ;Tipo de procesador

STATUS          equ     0x03                ; Registro de STATUS
C               equ     0x0                 ; bit de Carry del registro de STATUS
W               equ     0
F               equ     1
NUMEROA_L       equ     0x10                ;Define la posición del numero A (bajo)
NUMEROA_H       equ     0x11                ;Define la posición del numero A (alto)
NUMEROB_L       equ     0x12                ;Define la posición del numero B (bajo)
NUMEROB_H       equ     0x13                ;Define la posición del numero B (alto)
RESUL_L         equ     0x14                ;Define la posición del resultado (bajo)
RESUL_H         equ     0x15                ;Define la posición del resultado (alto)

                ORG     0x00                ;Vector de Reset
                goto    INICIO

                ORG     0x05                ;Salva el vector de interrupción

INICIO          movf    NUMEROA_L,W         ;Carga menos peso del numero A
                addwf   NUMEROB_L,W         ;Suma menos peso del numero B
                movwf   RESUL_L             ;Almacena el resultado
                movf    NUMEROA_H,W         ;Carga más peso del numero A
                btfsc   STATUS,C            ;¿Hubo acarreo en la suma anterior ?
                addlw   1                   ;Si, suma 1 al acumulador
                addwf   NUMEROB_H,W         ;Suma más peso del numero B
                movwf   RESUL_H             ;Almacena el resultado
                sleep                       ;dormir
                END                         ;Fin del programa fuente
```

En este ejercicio si por ejemplo los datos que se introducen el los registros RAM son los siguiente:

0x10 = 1010 1010 = AA h

0X11 = 1010 1010 = AA h

0x12 = 1111 1111 = FF h

0x13 = 0000 0000 = 00 h

al simular la suma en el MPLAB, el resultado que se debe obtener es:

0x14 = 1010 1001 = A9h

0x15= 1010 1011 = AB h

Ejercicio 2.- Con este ejercicio se pretende comprobar como el microcontrolador hace comparaciones de datos, utilizando la operación de resta y el valor que en cada caso toman los *flags* del registro de STATUS. El programa compara el valor del NUMERO_A con el del NUMERO_B que se introducirán por el usuario, realizando la siguiente operación dependiendo del resultado de la comparación:

- Si NUMERO_A>NUMERO_B entonces REUL = NUMERO_A + NUMERO_B.
- Si NUMERO_A=NUMERO_B entonces REUL = 00h
- Si NUMERO_A<NUMERO_B entonces REUL = NUMERO_A - NUMERO_B.

El organigrama del programa es el que se muestra en la Figura 72.

Recordemos que la instrucción SUBWF f,d realiza la resta en complemento a 2 del contenido del registro f menos el contenido del registro W y deja el resultado en W si d=0 y en el registro f su d=1. Si el resultado es negativo el *flag C* del registro de **STATUS** se pone a "0".

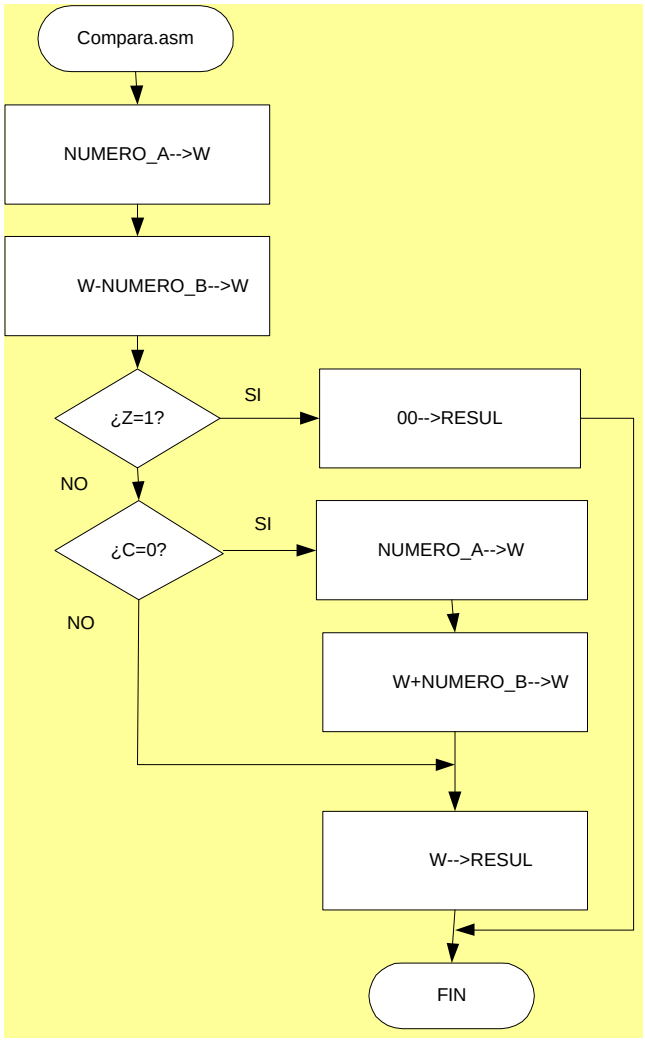


Figura 72.- Organigrama del programa Compara.asm

El programa en ensamblador será por lo tanto será el siguiente:

```
;*******************************************************************************
; Programa Compara.ASM                          Fecha: 24-Abril-2000
; Este programa compara el contenido de los registros NUMERO_A y NUMERO_B que se
; encuentran en las posiciones de memoria 0x10 y 0x11 respectivamente
; y el resultado de se almacena en la posición de memoria 0x12, siendo este
; 0 si NUMERO_A=NUMERO_B. Si NUMERO_A>NUMERO_B, el resultado deberá de ser: A-B.
; Si NUMERO_A<NUMERO_B el resultado es NUMERO_A+NUMERO_B.
; Hay que destacar que, al no haber instrucciones de comparación, esta se realiza
; mediante restas.
; Revisión : 0.0                        Programa para PIC16C84 y PIC16F84
; Velocidad del Reloj: 4 MHz            Reloj Instrucción: 1 MHz = 1 uS
; Perro Guardián :deshabilitado         Tipo de Reloj: XT
; Protección del código: OFF
;*******************************************************************************


                List    p=16F84             ;Tipo de procesador
                include "P16F84.INC"        ;Definiciones de registros internos

NUMERO_A        equ     0x10                ;Variable del NUMERO_A
NUMERO_B        equ     0x11                ;Variable del NUMERO_B
RESUL           equ     0x12                ;Variable para el resultado

                org     0x00                ;Vector de Reset
                goto    INICIO

                org     0x05                ;Salva el vector de interrupción

INICIO          movf    NUMERO_B,W  ;Carga el NUMERO_B
                subwf   NUMERO_A,W  ;Resta/compara con NUMERO_A
                btfsc   STATUS,Z    ;Son iguales (Z=1)??
                goto    A_IGUAL_B   ;Si
                btfsc   STATUS,C    ;No. A MAYOR que B (C=0)??
                goto    A_MAYOR_B   ;Si

A_MENOR_B       movf    NUMERO_A,W  ;No, A es MENOR que B
                addwf   NUMERO_B,W  ;Suma A más B
                movwf   RESUL       ;Guarda W en RESUL
                goto    STOP

A_MAYOR_B       movwf   RESUL       ;Guarda W en RESUL
                goto    STOP

A_IGUAL_B       clrf    RESUL       ;Pone a 0 el resultado

STOP            sleep               ;Poner break point de parada

                END                 ;Fin del programa fuente
```

En este ejercicio si por ejemplo los datos que se introducen el los registros RAM son los siguiente:

0x10 = 1010 1010 = AA h

0X11 = 1010 1010 = AA h

0x12 = 1111 1111 = FF h

0x13 = 0000 0000 = 00 h

al simular la suma en el MPLAB, el resultado que se debe obtener es:

0x14 = 1010 1001 = A9h

0x15= 1010 1011 = AB h

Ejercicio 3.- Vamos a comprobar ahora como se pueden convertir ahora números binarios a un display de 7 segmentos, para ello usaremos el circuito de la Figura 51, en el que habremos sustituido los diodos LED's por un display de 7 segmentos del tipo cátodo común, y que conectaremos de la tal y como se muestra en la Figura 73.

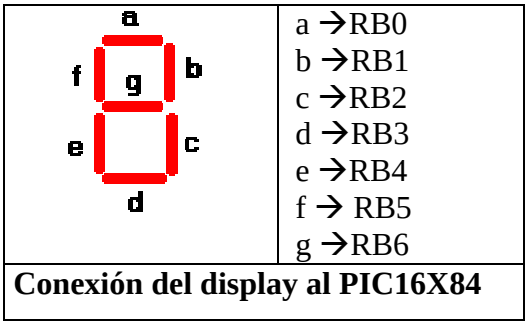
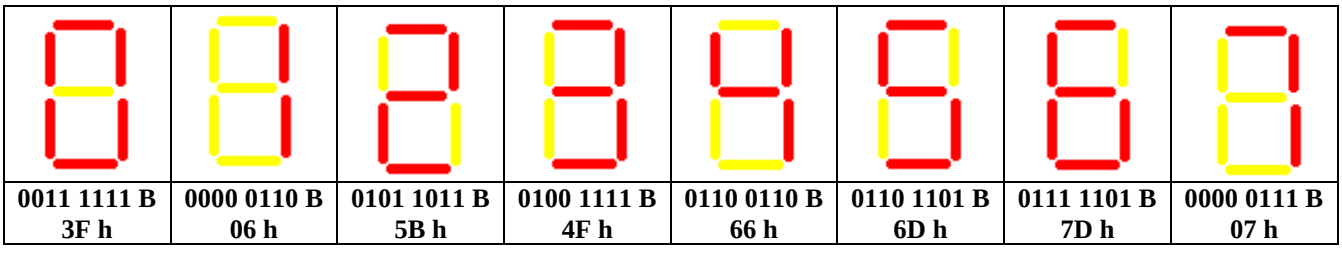


Figura 73.- Conexión de los segmentos del display de 7-segmentos cátodo común al PIC16F84

Por lo tanto, los códigos que habrá que escribir para representar los datos en el display serán los que se muestran en la Figura 74



0011 1111 B 3F h	**0000 0110 B 06 h**	**0101 1011 B 5B h**	**0100 1111 B 4F h**	**0110 0110 B 66 h**	**0110 1101 B 6D h**	**0111 1101 B 7D h**	**0000 0111 B 07 h**

							
0111 1111 B 7F h	**0110 0111 B 67 h**	**0111 0111 B 77 h**	**0111 1100 B 7C h**	**0011 1001 B 39 h**	**0101 1110 B 5E h**	**0111 1001 B 79 h**	**0111 0001 B 71 h**

Figura 74.- Datos que hay que escribir en el PORTB para representar los distintos caracteres en el display de 7 segmentos

El programa digitos.asm representará en el display el guarismo hexadecimal correspondiente al número hexadecimal que tengan las entradas RA0 a RA3.
En este programa hemos utilizado para leer los elementos de una tabla, la instrucción **retlw**, que carga el registro **W** con un literal ***k*** , y después carga el **PC** con el valor que se encuentra en la parte alta de la **PILA**, efectuándose así un retorno de subrutina. Además de una instrucción que modifica el contenido del contador de programa **addwf PLC,f** de tal manera que valor de leído en el **PORTA** sirve como índice de desplazamiento sobre el contador de programa que al llegar a esta instrucción siempre vale 06h.

El organigrama de este programa es el que se muestra en la Figura 75.

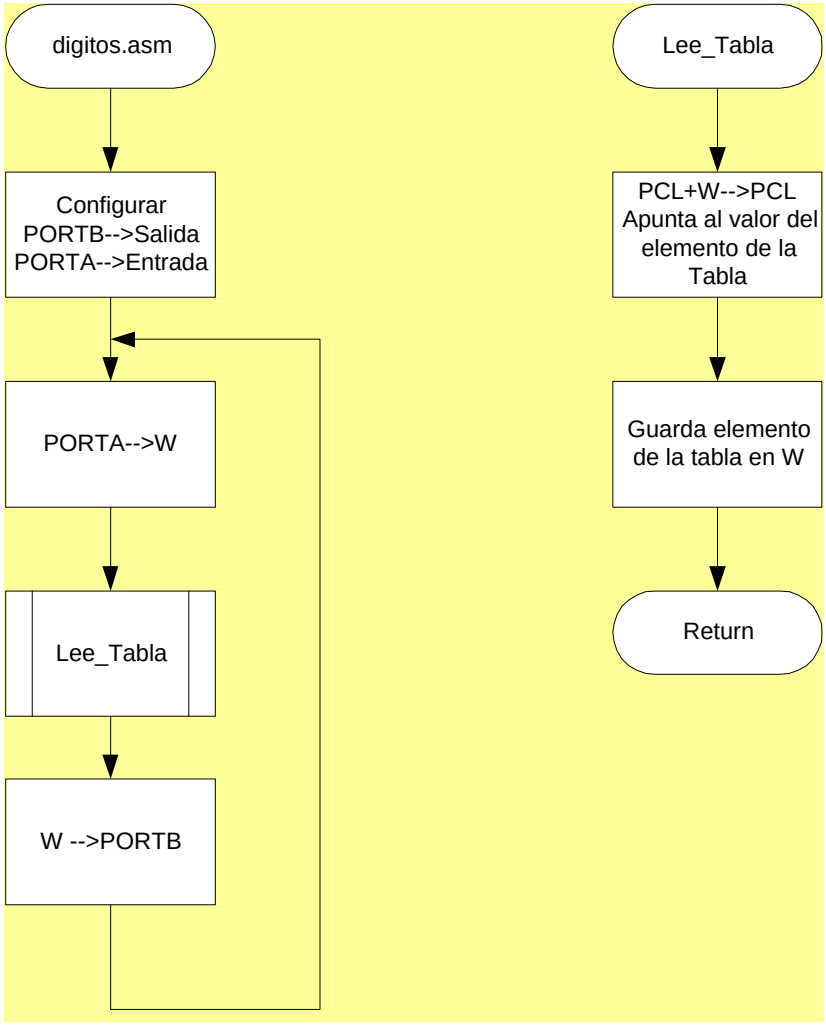


Figura 75 .- Organigrama del programa digitos.asm

El programa por lo tanto será el siguiente:

```
;*******************************************************************************
; Programa 7segmen.ASM          Fecha : 22 de Mayo de 2000
; Este programa  presenta en un display conectado al PORTB el número hexadecimal
; correspondiente al valor binario que tengan los conmutadores conectados a RA3,
; RA2;RA1,RA0, los segmentos del display cátodo común a, b, c, d, e, f, g están conectados
; a las líneas PB0, PB1, PB2, PB3, PB4, PB5, PB6
; Revisión : 1.0                        Programa para PIC16C84 y PIC16F84
; Velocidad del Reloj: 4 MHz            Reloj Instrucción: 1 MHz = 1 uS
; Perro Guardián:  deshabilitado        Tipo de Reloj : XT
; Protección del código: OFF
;*******************************************************************************

                LIST    p=16F84                 ;Tipo de procesador
                INCLUDE         "P16F84.INC"    ;Definiciones de registros internos
#DEFINE         BANCO0          bcf     STATUS,5    ;Siempre que escriba BANCO0 se ejecuta
                                                    ;bcf STATUS,0
#DEFINE         BANCO1          bsf     STATUS,5    ;Siempre que escriba BANCO1 se ejecuta
                                                    ;bsf STATUS,0
                ORG     0x00                    ;Vector de Reset
                goto    INICIO

                org     0x05                    ;Salva el vector de interrupción
LEE_TABLA       addwf   PCL,f
                retlw   b'00111111'     ;retorna 0
                retlw   b'00000110'     ;retorna 1
                retlw   b'01011011'     ;retorna 2
                retlw   b'01001111'     ;retorna 3
                retlw   b'01100110'     ;retorna 4
                retlw   b'01101101'     ;retorna 5
                retlw   b'01111101'     ;retorna 6
                retlw   b'00000111'     ;retorna 7
                retlw   b'01111111'     ;retorna 8
                retlw   b'01100111'     ;retorna 9
                retlw   b'01110111'     ;retorna A
                retlw   b'01111100'     ;retorna B
                retlw   b'00111001'     ;retorna C
                retlw   b'01011110'     ;retorna D
                retlw   b'01111001'     ;retorna E
                retlw   b'01110001'     ;retorna F


INICIO          BANCO1                  ;Seleciona el banco 1 de registros
                clrf    TRISB           ;Coloca el PORTB como salida
                movlw   b'11111111'     ;Coloca el PORTA como entrada
                movwf   TRISA
                BANCO0                  ;Selecciona el banco 0 de registros
                movf    PORTA,w         ;Carga el valor de los conmutadores en W
                call    LEE_TABLA
                movwf   PORTB           ;Sacar dato al diaplay
                goto    INICIO
                END
```

Directiva #DEFINE –Define una Etiqueta de Substitución de Texto

Sintaxis

#define <name> [<string>]

Descripción

Esta directiva define una cadena de substitución de texto. Dondequiera que <string> se encuentre en el ensamblador, se sustituirá por < string >.
Usando la directiva sin < string > genera una definición de tipo <name> para ser utilizada internamente y puede ser utilizada por la directiva IFDEF.
Esta directiva emula el ANSI 'C ' standard como #define. Define símbolos con este método no está disponible para ser usado por el MPLAB.

Ejemplo 1

```
#define     BANCO0          bcf     status,5
#define     BANCO1          bsf     status,5
```

Siempre que a lo largo del programa se escriba BANCO0, se selecciona el banco 0 de registros. De igual modo, siempre que se escriba BANCO1, se selecciona el banco 1 de registos.

Ejemplo 2

```
#define     longitud    20
#define     control     0x19,7
#define     posicion    (X,Y,Z)     (Y-(2 * Z +X))
            .
            .
            .

test_label  dw      posición (1, longitud, 512)
            bsf     control                     ; set bit 7 de la posición 19 RAM
```