SISTEMAS ELECTRÓNICOS Y AUTOMÁTICOS
PRACTICAS DE MICROCONTROLADORES PIC

PRÁCTICA 1:

Entorno de desarrollo MPLAB-IDE. Primeros ejemplos de programación.

- ***Introducción a la programación en ensamblador***
- ***Entorno de desarrollo MPLAB-IDE***
- ***Ejemplos y ejercicios***

---

1. Objetivos

- Introducir al alumno en la programación en ensamblador
- Conocer el entorno de desarrollo MPLAB
- Simular el funcionamiento de programas-ejemplo.

2. Introducción a la programación en ensamblador

Desde que se concibe un programa hasta que se graba en la memoria del microcontrolador, suceden una serie de operaciones que se muestran en la Figura 1.

El primer paso de un diseño basado en microcontrolador consiste en escribir el código fuente del programa en el lenguaje seleccionado. Posteriormente, si el lenguaje usado ha sido el ensamblador, se convierte a código ejecutable mediante el uso de un programa de ensamblador como el de la empresa Microchip, que se llama MPASM.

La depuración y simulación se realizarán mediante el simulador software MPSIM (entorno DOS) o el MPLAB (entorno Windows).

3. Escritura del código fuente

El código fuente del lenguaje ensamblador está estructurado en columnas. Cualquier texto que comience en la primera columna se considerará una ***etiqueta*** y será parte del ***campo de etiquetas***.

Las siguientes tres columnas contienen el ***campo de instrucciones***, el ***campo de datos*** y el ***campo de comentarios***. Los comentarios deben empezar con punto y coma (;) y pueden ir también en la primera columna.

Campo de etiquetas

Las etiquetas son nombres de subrutinas o secciones de código fuente. Dando nombres a partes del programa, se posibilita que las instrucciones puedan saltar o hacer referencia a esas partes sin necesidad de recordar las direcciones físicas donde están ubicadas.

El ensamblador MPASM permite etiquetas de hasta 32 caracteres. Una etiqueta puede ir seguida de dos puntos (:), espacios, tabuladores o RETURN. Deben empezar por un carácter alfanumérico o de subrayado (_) y pueden contener cualquier combinación de caracteres alfanuméricos.

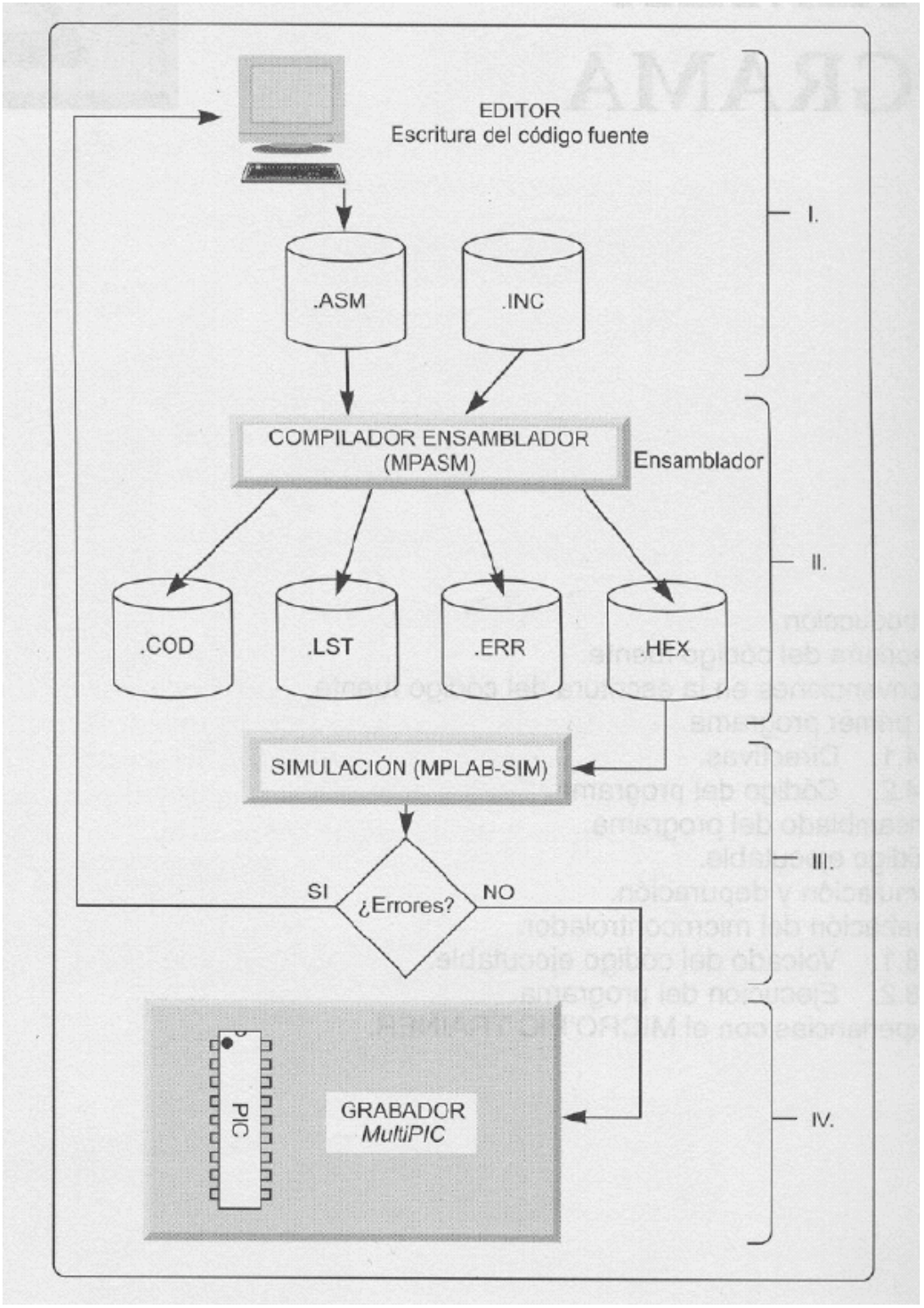

Figura 1. Organigrama de la secuencia a seguir para el grabado del microcontrolador.

Campo de instrucciones

Las segunda columna corresponde al ***campo de instrucciones*.** Puede ser una instrucción del microcontrolador o una instrucción para el ensamblador, llamada ***directiva***.

Campo de datos

Contiene datos u operandos para las intrucciones. En los PICs, los datos pueden ser un registro, un bit de un registro, una etiqueta o un número constante (llamado ***literal***). Algunas instrucciones no llevan datos. Si una instrucción necesita múltiples datos, deben separarse por comas (,).

La especificación de la base en la que se expresan los datos u operandos es opcional: d'65', b'01000001',0x41, o'101' y 'A' son el mismo dato. El sufijo **d** o **D** se emplea para expresar un valor en decimal. El sufijo **b** o **B** expresa un valor en binario. Para expresar un valor en hexadecimal se emplea el sufijo **0x** ó **0X**. Un número en base octal se representa mediante el sufijo **o** ó **O**. Finalmente, un valor se puede expresar mediante un caracter ASCII si se encierra entre comillas simples como 'A'.

Campo de comentarios

El último campo es el del comentario, siempre que haya un punto y coma (;) como primer carácter. Puede colocarse en cualquier lugar del código fuente.

Los comentarios en ensamblador son muy importantes. Los comentarios deben describir las tareas que realizan las instrucciones y/o subrutinas. Sin comentarios, es muy difícil descifrar, algunos días después, el código fuente que uno mismo ha creado.

El listado que se presenta a continuación en la Figura 2 presenta el aspecto del código fuente del programa EJEMPLO1.ASM con los campos de etiquetas, instrucciones, datos y comentarios recuadrados para su clara distinción.

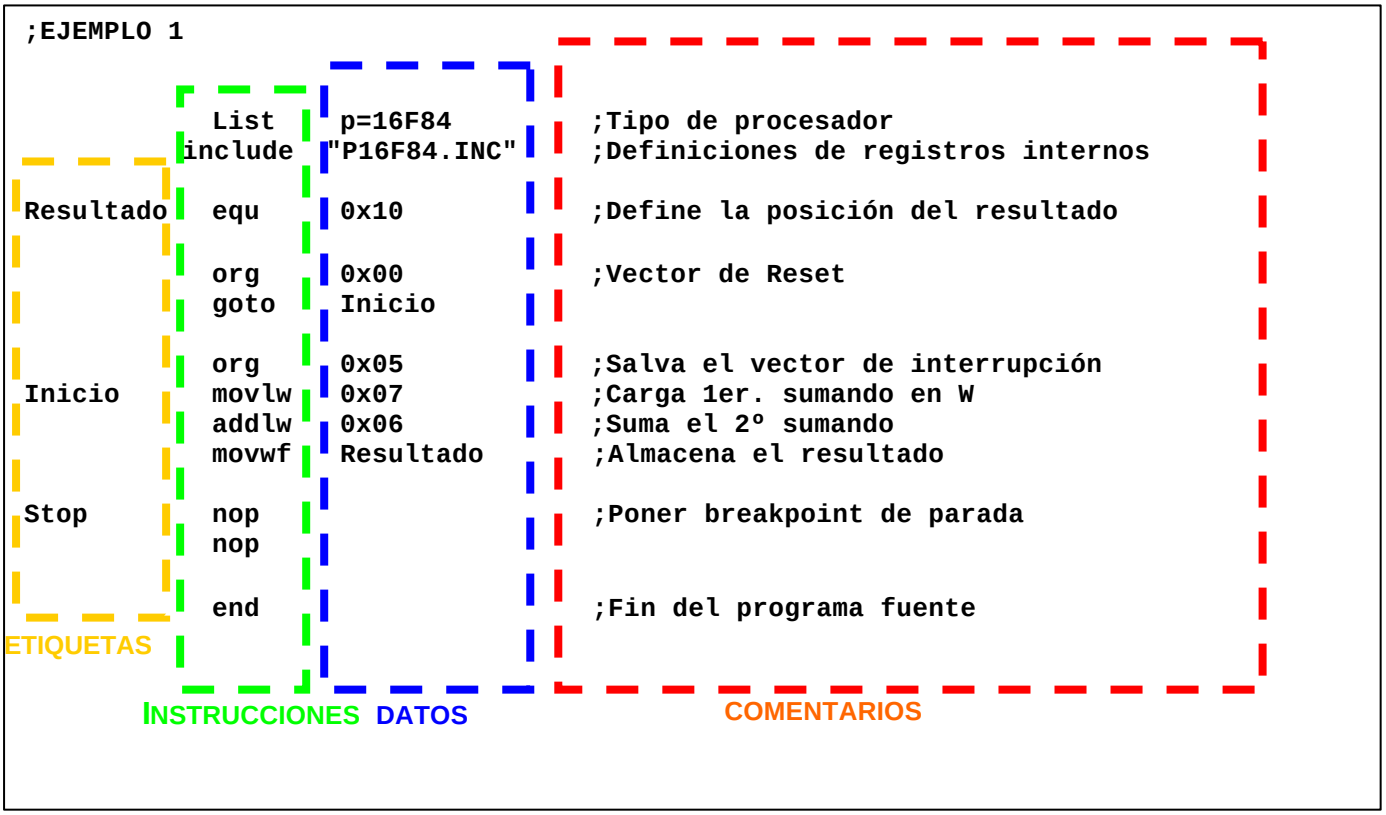

Figura 2. Listado del programa EJEMPLO1.ASM.

4. Convenciones en la escritura del código fuente

Para hacer la tarea del programador más grata, se usan algunas convenciones. Con ellas, se facilita la revisión y comprensión de un programa.
Algunas de las convenciones más utilizadas son:

- *Los ficheros de código fuente llevarán la extensión *.ASM o *.SRC*
- *Los ficheros de listado llevarán la extensión *.LST*
- *Los ficheros de código objeto llevarán la extensión *.OBJ*
- *Los ficheros ejecutables llevarán la extensión *.HEX*
- *Los ficheros de errores de ensamblado llevarán la extensión *.ERR*
- *Los nemónicos escritos en mayúsculas hacen que el código escrito sea más visible.*
- *Comentarios explicando cada línea de código.*
- *El espacio entre caracteres se escribe "_". RB0_ES_1 es más fácil de leer que RB0ES1.*

5. El primer programa: EJEMPLO1.ASM

Vamos a ir examinando las distintas partes del programa EJEMPLO1.ASM y explicando cada una de ellas.

Directivas

Una directiva es un comando escrito en el código fuente para realizar un control directo o ahorrar tiempo a la hora de ensamblar. El resultado de incorporar directivas se puede ver en el fichero *.LST después de ensamblar el programa.

En el programa EJEMPLO1.ASM aparecen las siguientes directivas:

- **LIST** → permite elegir, entre otras cosas, el tipo de microcontrolador a utilizar (P), número de carácteres por línea (C), tamaño de los tabuladores (B), base de numeración por defecto (R), niveles de mensajes de salida (W), etc.
- **EQU** → se utiliza para asignar valores a las etiquetas deseadas. Así, *Resultado*, tiene asignado el valor *0x10*, y puede referirse a un registro de la memoria de datos del PIC.
- **ORG** → indica al ensamblador dónde debe comenzar a colocar las instrucciones en la memoria de programa. Es decir, es el **ORiG**en para todo el código que sigue.
  La dirección de comienzo (origen) es en la posición **0**, debido a que la familia de microcontroladores PIC de gama media después del encendido o RESET siempre ejecutan la instrucción situada en la dirección0. Se denomina ***Vector de Reset.***
  La dirección **4** es el ***Vector de Interrupción***. Si se genera una interrupción el microcontrolador ejecuta la instrucción que se encuentre aqui. Es una buena práctica dejar libre la dirección 4 por si más adelante deseamos añadir capacidad de interrupción a nuestro programa. El programa salta por encima del Vector de interrupción y comienza en la dirección **5**.
- **INCLUDE** → permite añadir listados de programas al listado del programa actual.
- **END** → indica al ensamblador el final del código fuente, donde ha de finalizar el proceso de ensamblado

Código del programa

Las tres instrucciones que siguen a la etiqueta **Inicio** realizan la suma de dos numeros *literales*, 7 y 6, y guardan el resultado de la suma en la posición de memoria 0x10 etiquetada como **Resultado**.

```
movlw   0x07            ;Carga 1er. sumando en W
addlw   0x06            ;Suma el 2º sumando
movwf   Resultado       ;Almacena el resultado
```

Figura 3. Detalle del listado del programa EJEMPLO1.ASM.

Por último, **nop** es una instrucción que produce que el PIC no haga nada, significa ***no*** *o**p**eración.*

6. Entorno de desarrollo MPLAB-IDE

Para editar, compilar y depurar los programas fuente de los microcontroladores PIC utilizaremos el entorno de desarrollo MPLAB IDE 6.40, basado en ventanas y que se puede bajar gratis a través de Internet o pedirse, también gratis, a Sagitrón, su distribuidor en España. Además es gráfico, funcionando perfectamente bajo Windows.

Este entorno, que a continuación pasaremos a describir, funciona tipo *Container*, es decir, sus distintas opciones son asociadas a programas, que serán ejecutados cuando se las pulse. De este modo bastará con definirle un ensamblador, un emulador o un grabador distinto a los que lleva por defecto para incorporarlo inmediatamente al entorno.

El ensamblador que utiliza por defecto el MPLAB IDE es el MPASM, que conserva de sus tiempos bajo MS-DOS.

Desde MPLAB IDE es posible abrir un fichero en ensamblador (*.asm) (*fichero fuente*) y ensamblarlo para poder obtener el fichero de entrada de un grabador (*.hex) (*fichero binario*), pero también es posible el uso de proyectos que utilicen varios *.asm, permitiendo así reutilizar código con mayor facilidad, al ser este más modular. También es posible elegir el tipo de microcontrolador sobre el que simular y activar el modo de simulación o depuración (*debugger*) denominado MPLAB SIM.

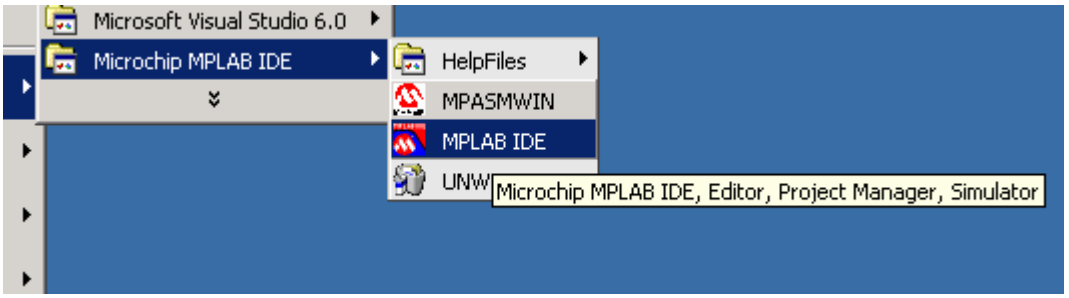

Figura 4. Entorno de desarrollo MPLAB-IDE

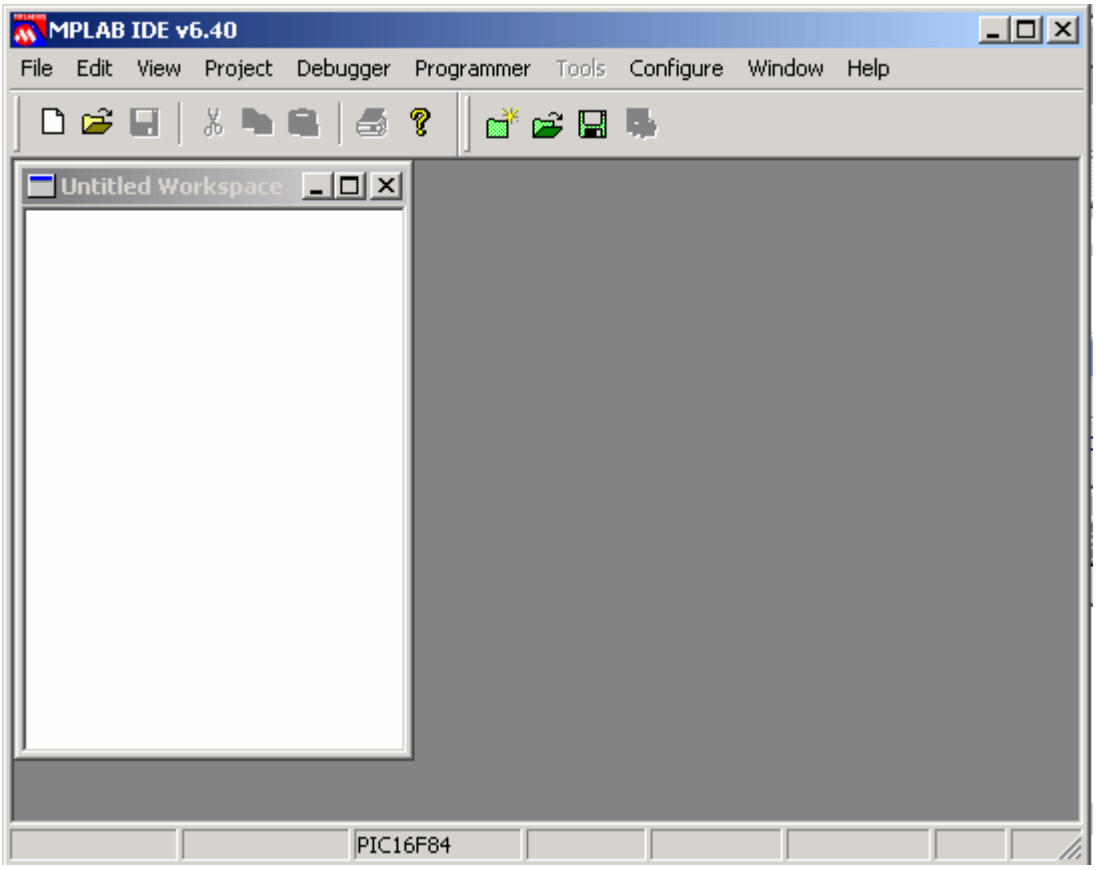

Figura 5. Entorno de desarrollo MPLAB-IDE

7. Creando un proyecto nuevo.

7.1. Introducción.

Para generar el código ejecutable de un microcontrolador, primero hay que generar el fichero fuente (*.asm) donde se escribe el programa en lenguaje ensamblador; y luego hay que ensamblarlo para obtener el fichero ejecutable o binario (*.hex) que será el utilizado por el grabador para volcarlo sobre la memoria de programa del dispositivo.

Desde MPLAB-IDE es posible escribir el programa fuente (*.asm), ensamblarlo para obtener el el fichero ejecutable o binario (*.hex), grabarlo a un dispositivo, simular su funcionamiento y depurar el código. A continuación se describen los pasos a seguir para realizar todas esas tareas:

7.2. Creando el fichero fuente.

® 1. Seleccionad *File>New* .

Se abre una ventana en blanco sobre la que se escribe el código de nuestro programa:

```
;EJEMPLO 1


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

Resultado   equ   0x10        ;Define la posición del resultado

            org   0x00        ;Vector de Reset
            goto  Inicio

            org   0x05        ;Salva el vector de interrupción

Inicio      movlw 0x07        ;Carga 1er. sumando en W
            addlw 0x06        ;Suma el 2º sumando
            movwf Resultado   ;Almacena el resultado

Stop        nop               ;Poner breakpoint de parada
            nop

            end               ;Fin del programa fuente
```

Figura 6. Listado del programa EJEMPLO1.ASM.

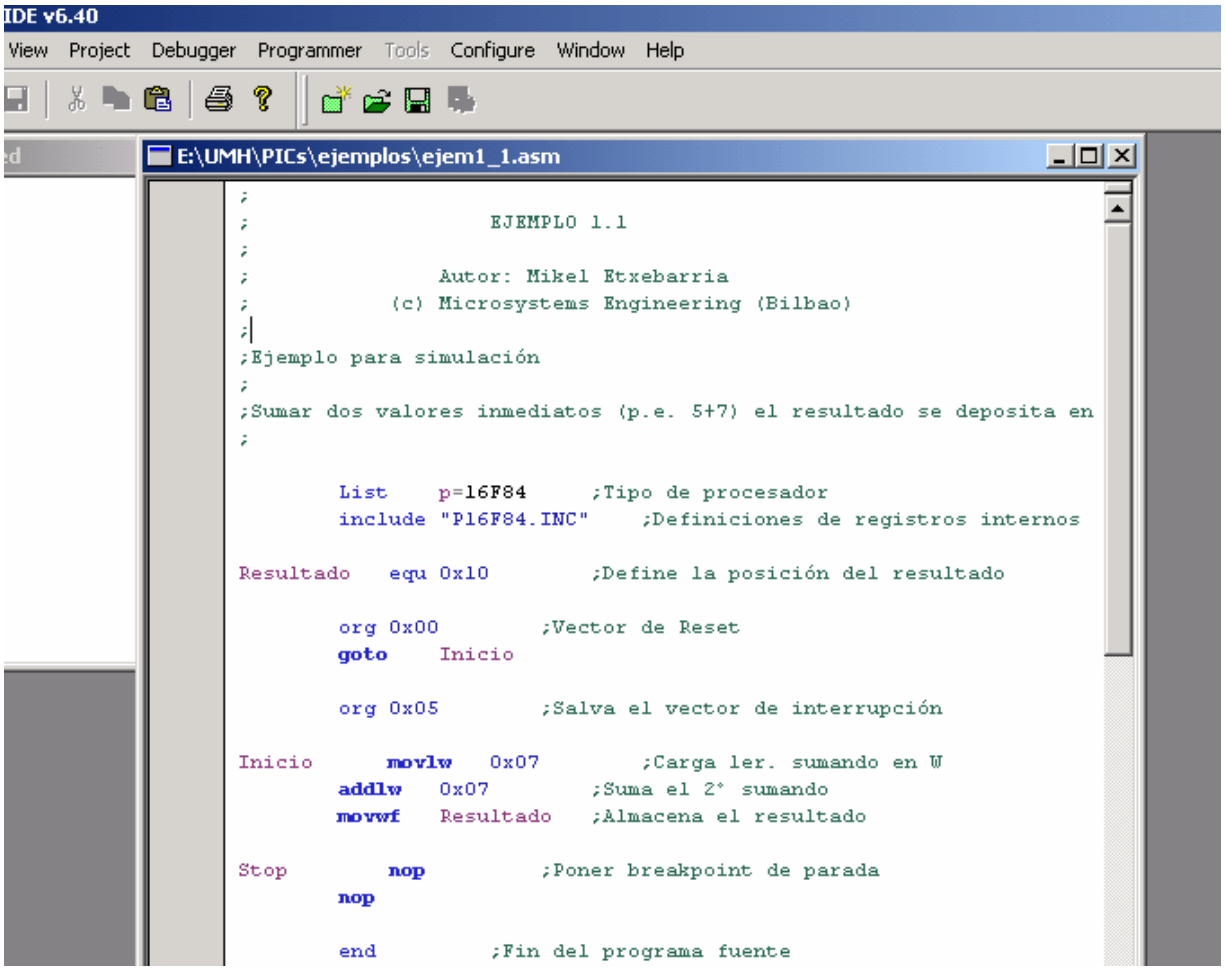

Figura 7. Creando el fichero fuente

→2. Después de copiar el código anterior, seleccionad *File>Save* y guardad el fichero en un directorio determinado con el nombre **ejemplo1.asm.**

7.3. Creando un proyecto.

El siguiente paso para desarrollar una aplicación desde MPLAB-IDE es creando un proyecto, la forma más sencilla es utilizar la herramienta MPLAB Project Wizard.

→1. Abrid la herramienta MPLAB Project Wizard seleccionando *Project>Project Wizard.*

→2. Elegid el modelo de microcontrolador **PIC16F84**.

→3. Confirmad la localización de **Microchip Toolsuite**, y seleccionad **MPASM Assembler.**

⟶4.Introducid un nombre para denominar el proyecto (**Proyecto1**) y seleccionad el directorio donde se desea guardar.

⟶5.Añadid el fichero fuente creado anteriormente (**ejemplo1.asm**) al proyecto.

⟶6.Pulsad el botón de *Finalizar* para salir del asistente.

7.4. Ensamblando el proyecto.

Una vez generado el proyecto, hay que ensamblarlo, para ello,

⟶Seleccionad *Project>Build all*.

Si el proyecto no se ensambla correctamente, comprobar los mensajes de error obtenidos y volver a ensamblar.

Al ensamblar con éxito se genera el fichero ejecutable **ejemplo1.hex**.

7.5. Grabación.

Si se desea grabar el fichero ejecutable **ejemplo1.hex** en la memoria del microcontrolador, hay que utilizar un dispositivo externo a nuestro PC denominado grabador o programador, por ejemplo el PICSTART Plus.

Si conectamos el programador a nuestro PC, y colocamos un microcontrolador en su zócalo, desde MPLAB IDE es posible leer y grabar programas mediante la selección de **Programmer>Select Programmer>PICSTART Plus**

Pero no es el caso que nos interesa, en esta práctica sólo utilizaremos el entorno MPLAB_IDE para editar y simular programas, no para grabarlos.

En prácticas posteriores sí utilizaremos dispositivos programadores, en concreto la tarjeta EasyPIC.

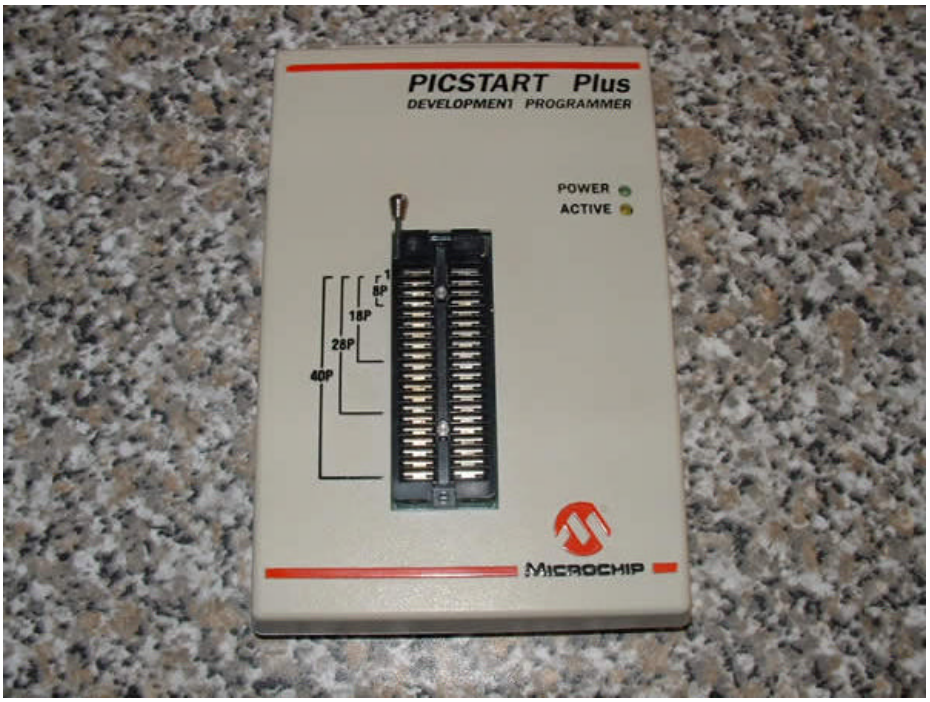

Figura 8. Programador PICSTART Plus

7.6. Simulación y depuración del programa.

Una vez que el proyecto está correctamente ensamblado, para comprobar su funcionamiento, se utiliza la herramienta de simulación MPLAB SIM.

El simulador es un software que se ejecuta sobre el PC y simula la ejecución de las instrucciones en el interior de la CPU del microcontrolador.

→ 1.Seleccionad MPLAB SIM como simulador seleccionando la opción *Debugger>Select>Tool>MPLAB SIM*

Al seleccionar MPLAB SIM aparecerán nuevos botones en la barra de herramientas que permiten seleccionar las opciones de simulación y depuración (ver Figura 9)

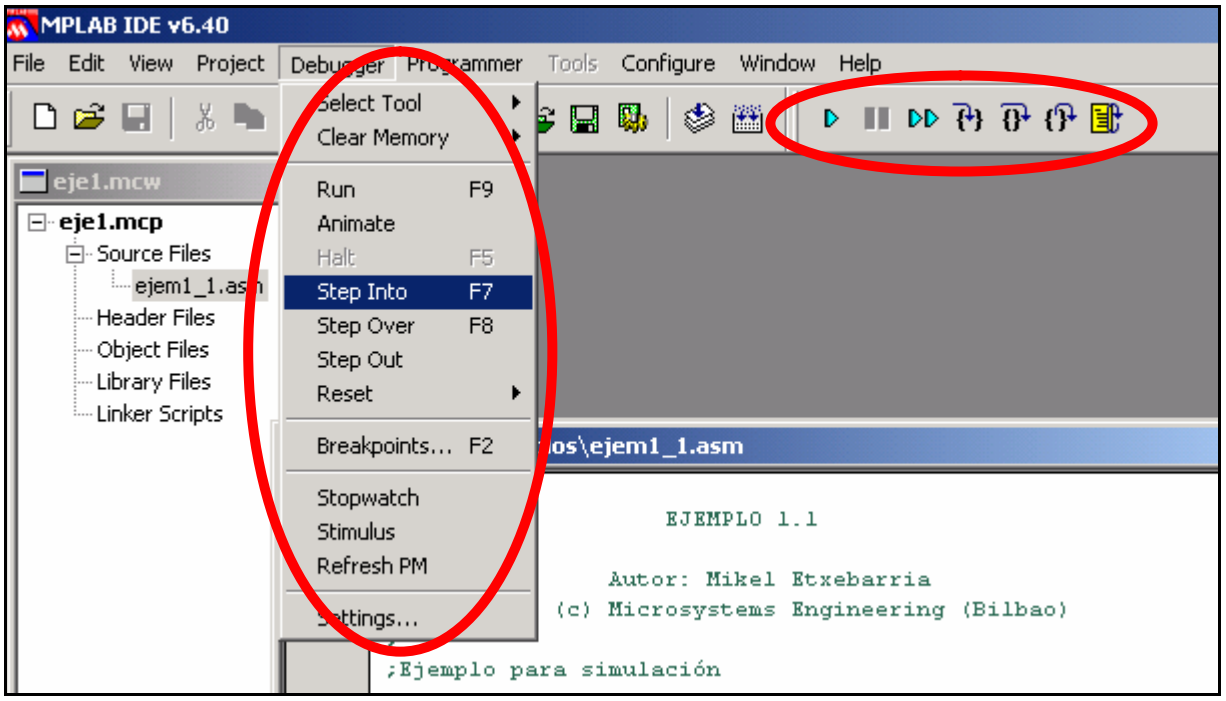

Figura 9. Depurador MPLAB SIM

→ 2. Seleccionad *Debugger>Reset* para inicializar la depuración.
Debe aparecer una flecha verde a la izquierda de la ventana de programa señalando a la primera línea del código que se ejecutará.

→ 3. Seleccionad *Debugger>Run* para ejecutar el programa. El mensaje *"Running..."* aparecerá en la barra de estado.

→ 4. Para detener la ejecución, seleccionad *Debugger> Halt.*

→ 5. Seleccionad *Debugger> Step Into* para ejecutar el programa paso a paso.

Existen teclas rápidas para realizar las funciones de simulación anteriores:

Debugger Menu	Toolbar Buttons	Hot Key
Run		F9
Halt		F5
Step Into		F7
Reset		F6

Mientras se realiza la ejecución del programa con las opciones de depuración anteriores es posible visualizar los valores de las variables superponiendo el cursor del ratón sobre los nombres de las variables en la ventana de programa.

También se puede utilizar la ventana Watch para visualizar siempre el valor de ciertas variables de interés (como el acumulador, otros registros especiales, u otras posiciones de memoria). Para ello,

→ Seleccionad View>Watch y marcad los registros especiales que se desean visualizar o escribid el nombre de las variables.

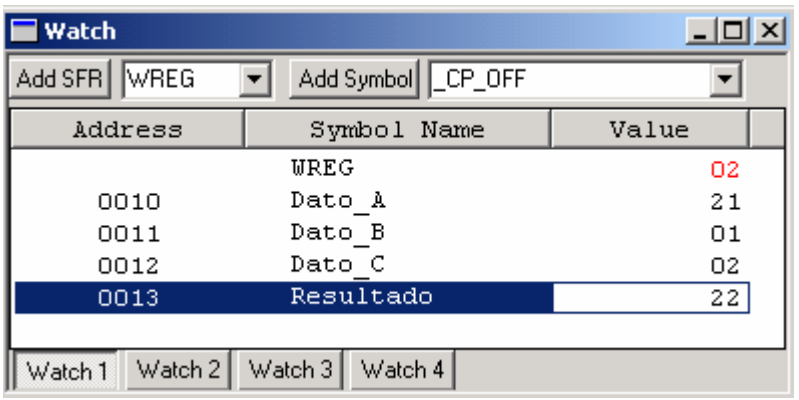

Figura 10. Ventana Watch.

También es posible visualizar de forma global toda la memoria del microcontrolador agrupada por tipos:

→ Seleccionad *View>Special Function Registers*

→ Seleccionad *View>Program Memory*

→ Seleccionad *View>File Registers*

→ Seleccionad *View>EEPROM*

7.7. Simulación mediante breakpoints y traza

Un ***punto de ruptura*** o *BreakPoint* es un punto o instrucción donde la ejecución del programa se detiene, por ello también se le suele denominar ***punto de paro***, permitiendo el análisis del estado del microcontrolador. Para continuar la ejecución del programa hay que volver a pulsar sobre *Run* o *Animate*.

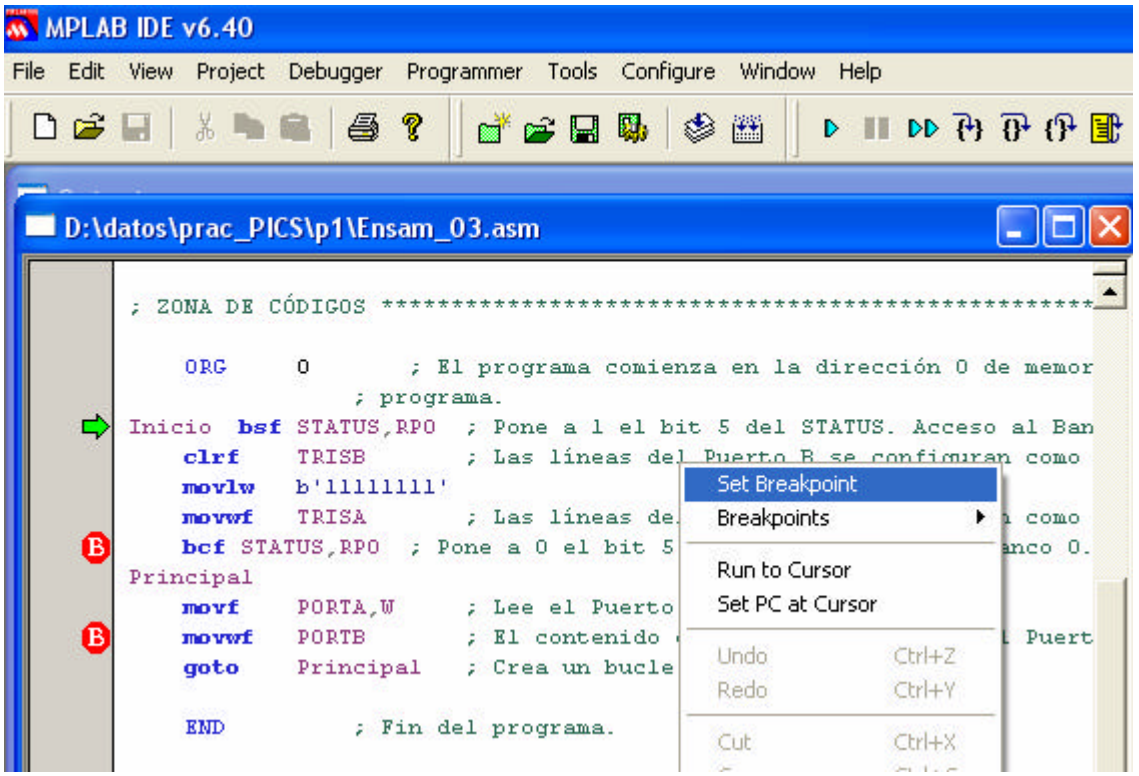

Figura 11. Situar un *Breakpoint*

Para situar un *Breakpoint* sobre una línea señalada por el cursor se pulsa el botón derecho del ratón, de manera que aparece el menú desplegable, como en la figura 11. Selecciona *Set Breakpoint* y aparecerá sobre el programa una "**B**" en rojo en la posición donde se ha situado el punto de paro. Otra forma de situar o eliminar un *Breakpoint* es realizando una doble pulsación con el ratón sobre el número de línea donde se quiere situar el punto de paro.

La ventana **memoria de traza** es una herramienta que ayuda a simular los programas (Figura 12). El *Simulator Trace* toma una instantánea de la ejecución del programa. En el simulador el buffer de traza o memoria de traza es útil para visualizar un registro a lo largo de la ejecución del programa, de manera que se puede registrar por dónde pasa el programa y después analizarlo. El simulador toma datos desde la última vez que se pulsó *Run* o

Animate hasta que se detiene la simulaciónde programa (normalmente por un *Breakpoint*).

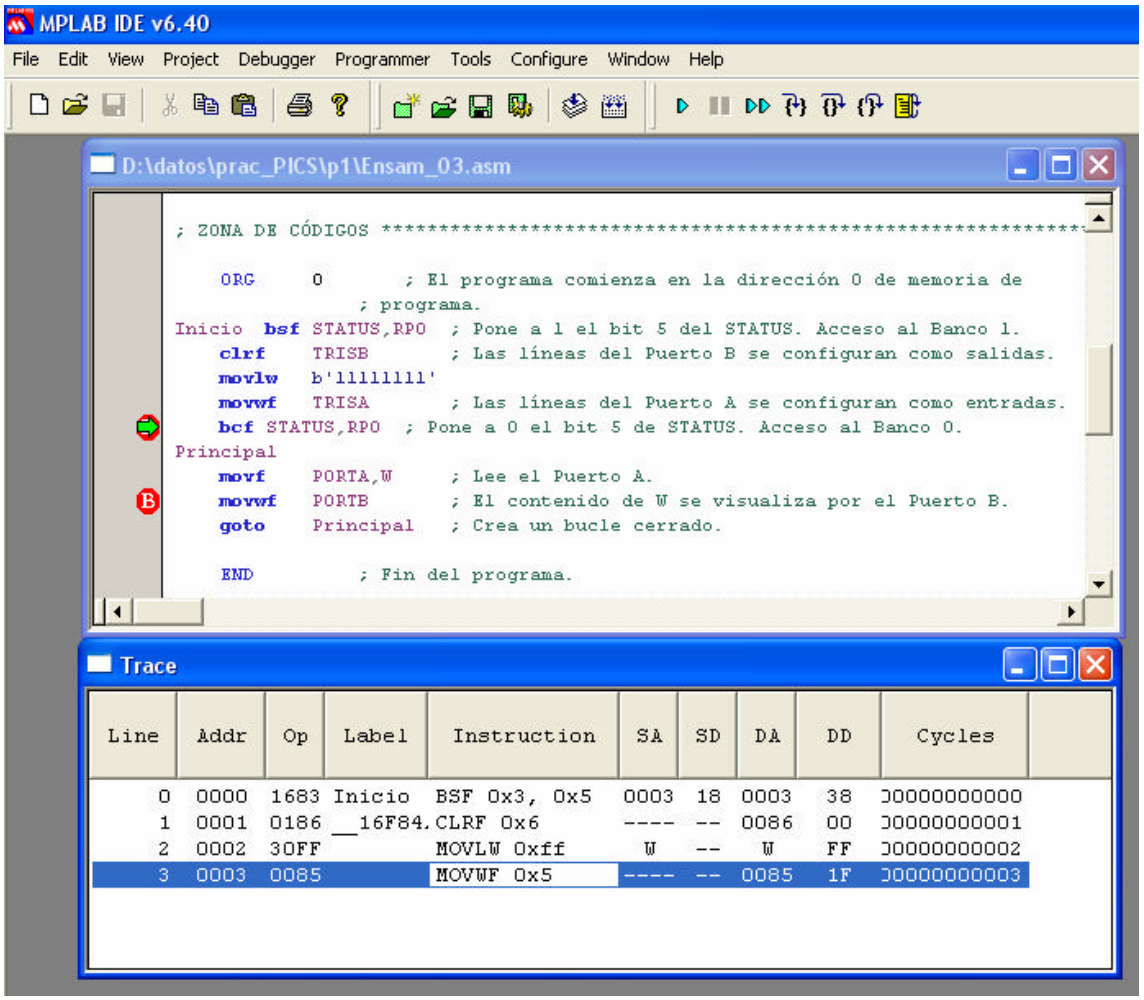

Figura 12. Simulador de traza

Para visualizar la ventana de memoria de traza hay que seleccionar el menú *View* > *Simulator Trace*. El simulador muestra en la ventana *Trace* cualquier variación sobre los registros al ejecutarse el código de instrucción. Esta ventana tiene las columnas cuyos significados se citan:

Line	Número de líneas ejecutadas desde que se pulsó *Run* por última vez.
Addr	Dirección de la memoria de programa donde se encuentra la instrucción.
Op	Código de operación numérico de la instrucción.
Label	Etiqueta de la instrucción si la tuviese.
Instruction	Instrucción ejecutada.
SA	Dirección numérica del registro fuente
SD	Dato del registro fuente
DA	Dirección numérica del registro destino
DD	Dato del registro destino
Cycles	Ciclos máquina transcurridos

El contenido de la memoria de traza se pues salvar a un fichero para un posterior análisis. Para ello, estando situado sobre esta ventana pulsar el botón derecho del ratón y seleccionar *Output to File* salvando por el procedimiento conocido en Windows.

7.8. Simulación de entradas

Una de las operaciones más habituales de cualquier simulación consiste en variar los valores de las líneas de entrada. A esto se denomina “estimular” la entrada. Para cambiar los estímulos de una entrada de un puerto hay que seleccionar el menú *Debugger* > *Stimulus*. En la ventana que aparece, selecciona la pestaña *Pin Stimulus* (Figura13).

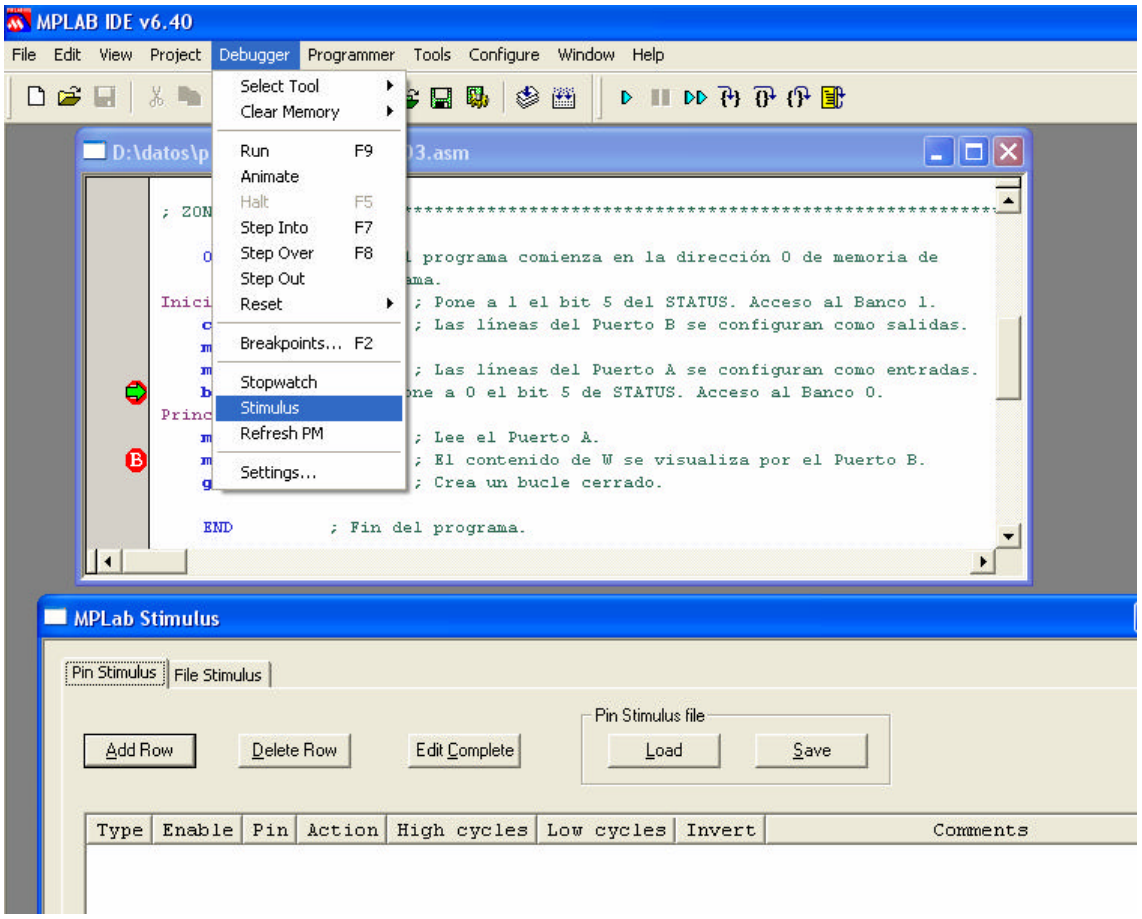

Figura 13. Menú para entrar en la ventana de estímulos

Si pulsa sobre *Add Row*, se irán añadiendo diferentes filas. Cada una de estas filas corresponde a un estímulo sobre una línea de entrada. La forma de editarlos es pulsar sobre la casilla correspondiente y seleccionar la patilla a la que se quiere vincular y el tipo de cambio que se desea realizar con ese pin para cada pulsación:

High	Poner la entrada a "1"
Low	Poner la entrada a "0"
Toggle	Cambiar de valor cada vez que se pulse. Ésta es la más habitual.
Pulse	Cambia el estado del pin y retorna de nuevo a su valor actual

La figura 14 muestra como se ha configurado para las cinco líneas del puerto A como entrada y en modo *Toggle*.

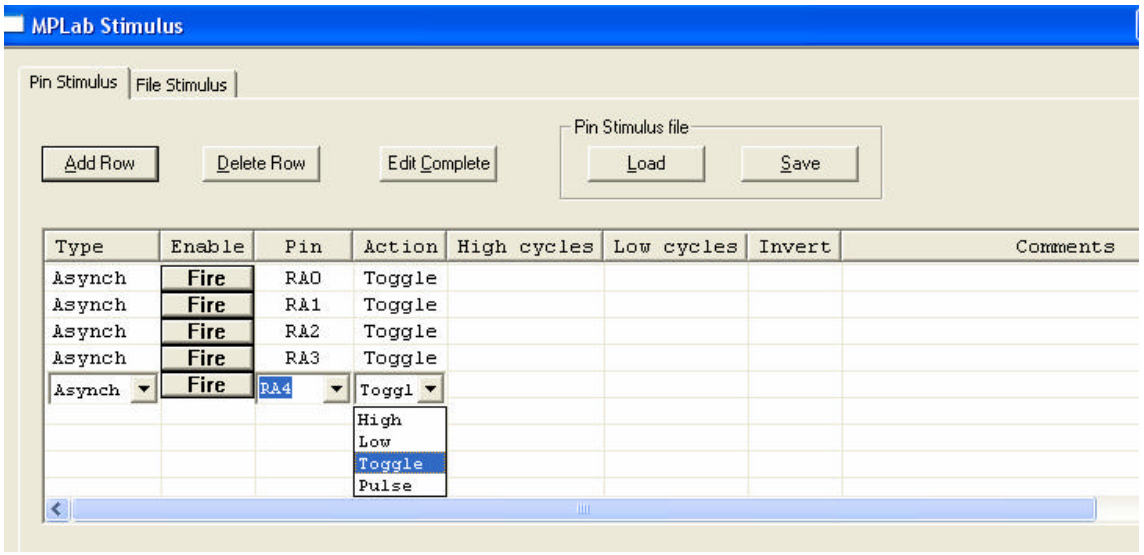

Figura 14. Configurar los estímulos para el Puerto A como entrada y modo *Toggle*.

Tras pulsar el botón *Fire* habrá de ejecutarse la siguiente instrucción antes de ver los cambios a través de las ventanas de visualización.

Es posible salvar la configuración realizada mediante el botón *Save* para recuperarla en posterior ocasión con el botón *Load*.

8. Ejercicios

8.1. Ejercicio:

Cread un proyecto y simulad el funcionamiento del programa EJEMPLO1.ASM.

8.2. Ejercicio:

Modificad el programa EJEMPLO1.ASM para que realice la suma de tres números situados en las posiciones de memoria 0x11, 0x12, 0x13 y el resultado se almacene en 0x15.

8.3. Ejercicio:

Cread un proyecto y simulad el funcionamiento del siguiente programa que realiza la suma de dos números de 16 bits cada uno.

Realizad las siguientes sumas hexadecimales: 00FF + 0001, 01E4 + 02C4

```
;EJEMPLO 2
;Suma de dos números, A y B, de 16 bits cada uno.


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

Dato_A_L    equ   0x10        ;Define la posición del dato A (bajo)
Dato_A_H    equ   0x11        ;Define la posición del dato A (alto)
Dato_B_L    equ   0x12        ;Define la posición del dato B (bajo)
Dato_B_H    equ   0x13        ;Define la posición del dato B (alto)
Resultado_L equ   0x14        ;Define la posición del resultado (bajo)
Resultado_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  Dato_A_L,W  ;Carga menos peso del dato A
            addwf Dato_B_L,W  ;Suma menos peso del dato B
            movwf Resultado_L ;Almacena el resultado
            movf  Dato_A_H,W  ;Carga más peso del dato A
            btfsc STATUS,C    ;Hubo acarreo anterior ??
            addlw 1           ;Si, suma 1 al acumulador
            addwf Dato_B_H,W  ;Suma más peso del dato B
            movwf Resultado_H ;Guarda el resultado

Stop        nop               ;Poner breakpoint de parada
            nop

            end               ;Fin del programa fuente
```

8.4. Ejercicio:

Cread un proyecto y simulad el funcionamiento del programa Ensam_01.ASM.

8.5. Ejercicio:

Cread un proyecto y simulad el funcionamiento del programa Ensam_02.ASM.

8.6. Ejercicio:

Cread un proyecto y simulad el funcionamiento del programa Ensam_03.ASM.