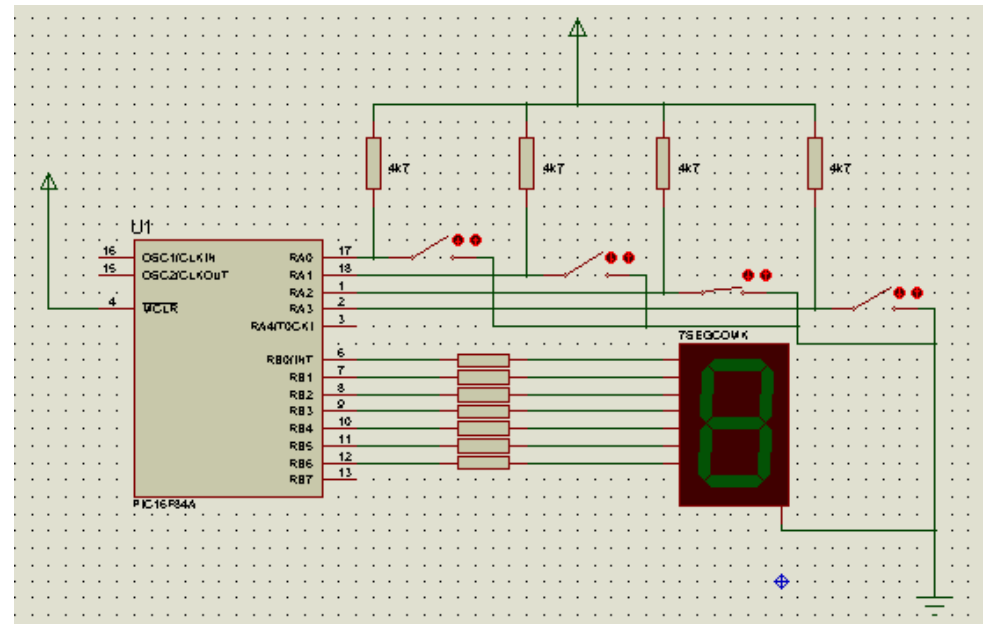


# PROTEUS

## Depuración de programas para microprocesadores



# Introducción

**Como ya se ha indicado en temas anteriores, la aplicación PROTEUS, tiene entre sus utilidades la simulación de los esquemas realizados mediante el módulo Proteus ISIS.**

**Una parte fundamental es la simulación de circuitos que incluyen microcontroladores, que es la que en principio se hace mas atractiva para nuestro curso.**

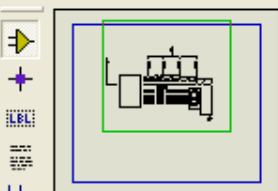
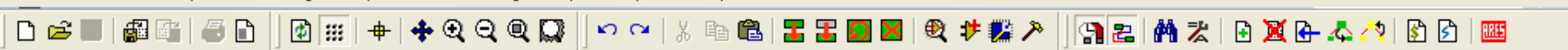
**Todos los puntos que se estudiarán a continuación harán referencia a un diseño y un entorno de trabajo para microcontroladores PIC.**

# Depuración de Programas para Microprocesadores

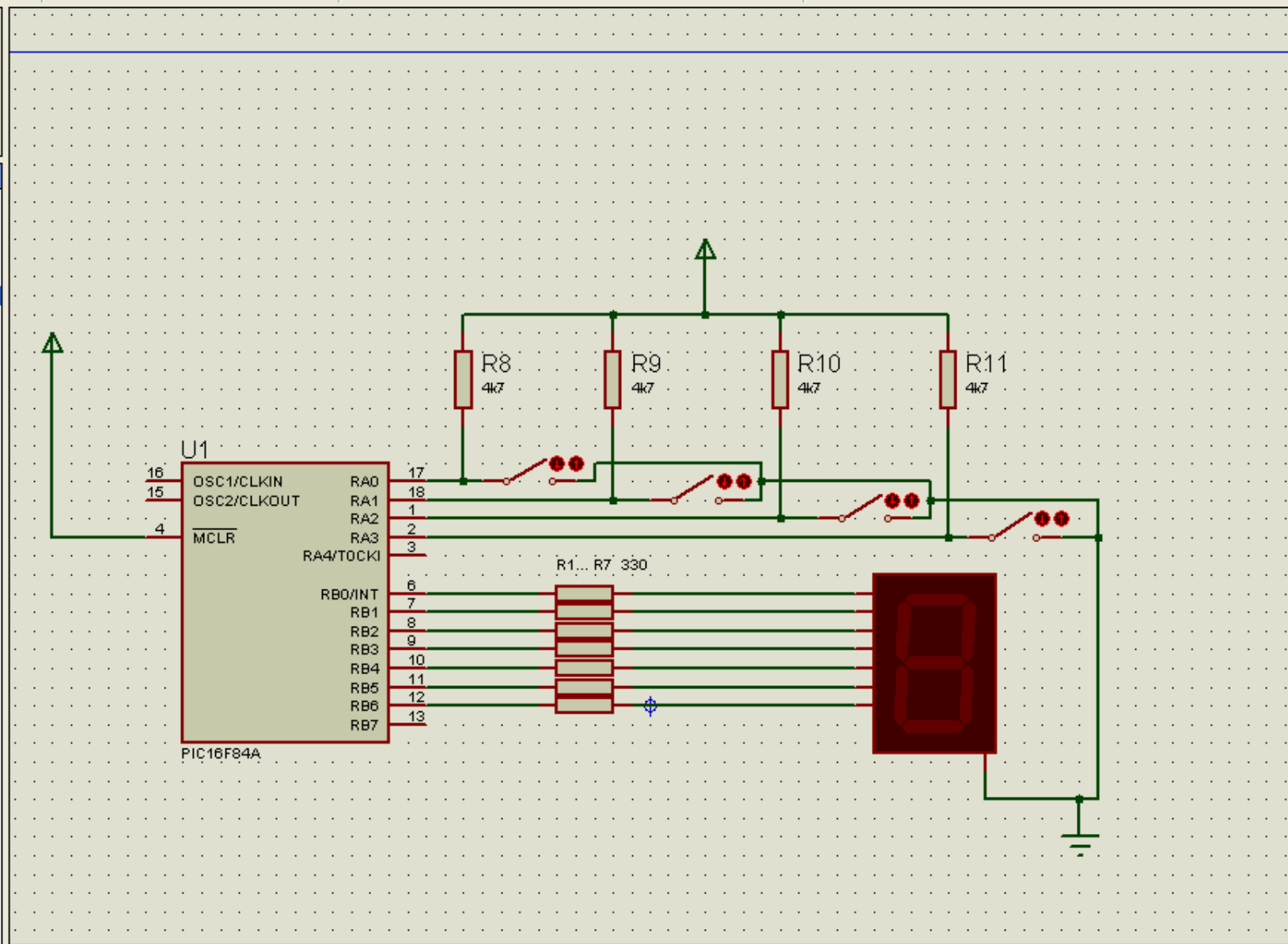
**Evidentemente el primer paso para simular cualquier circuito electrónico es diseñar el hardware sobre el que haremos “correr” el programa o software.**

**En nuestro caso se trata de un PIC 16F84, al que se introducen valores hexadecimales, a través de unos switch's, conectados desde RA0 hasta RA3 y se desea visualizar el valor introducido en un un display de 7 segmentos que se ha conectado al puerto B.**

**El circuito sería:**



- P L DEVICES**
- 7SEG-COM-CATHODE
  - BUTTON
  - CAP
  - CRYSTAL
  - PIC16F84A
  - PULLUP
  - SWITCH

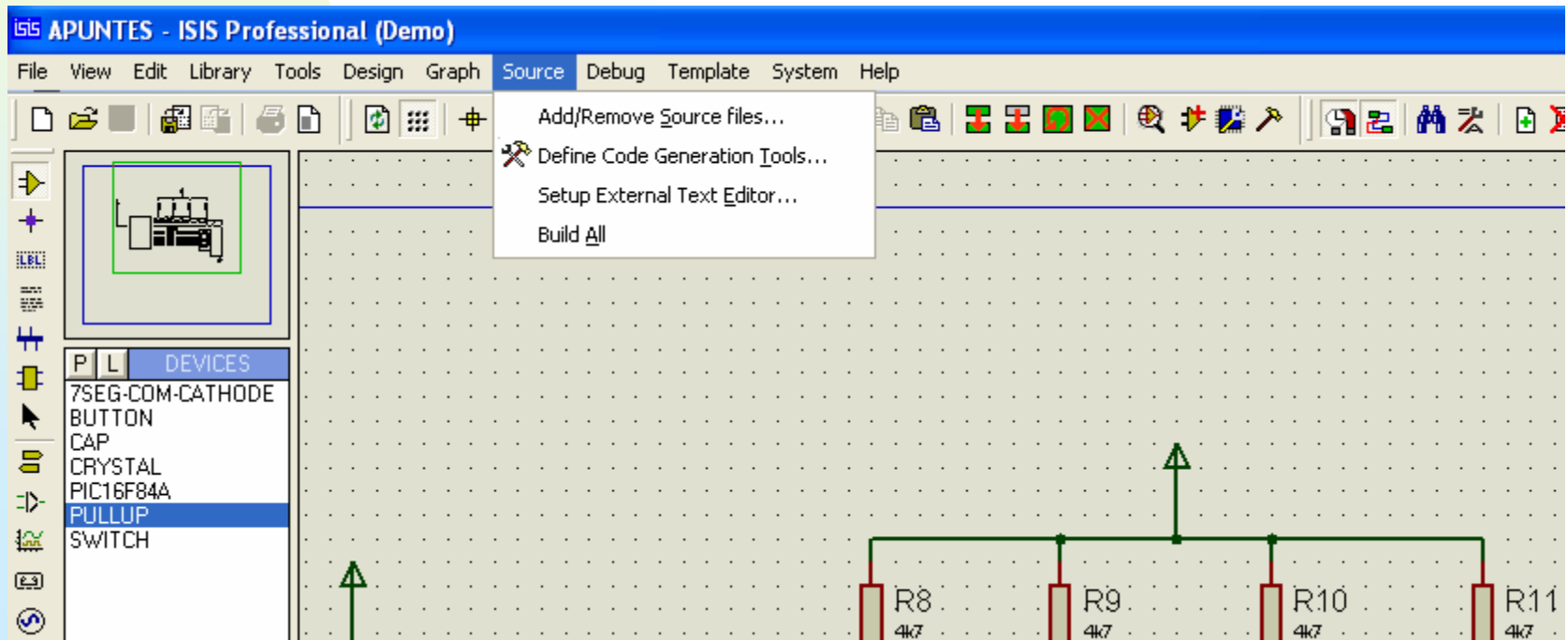


- **Una vez realizado el circuito, comenzaremos a definir el entorno en el que se encuentra el programa que escribiremos y que posteriormente “correrá” sobre el micro.**

# CONFIGURACIÓN BÁSICA DEL ENTORNO DE TRABAJO

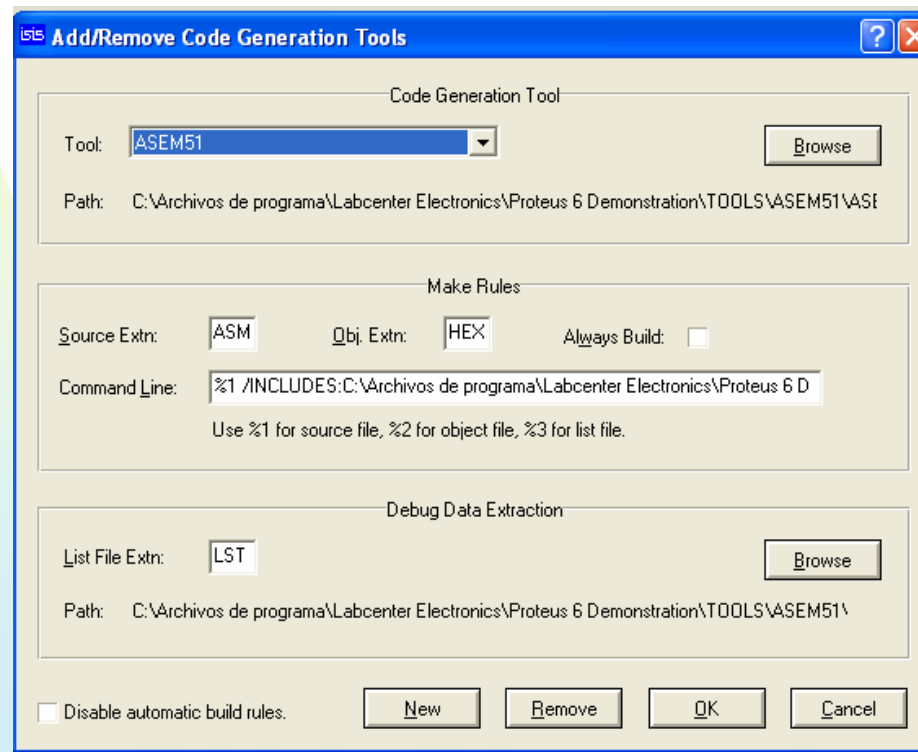
Esta configuración la tendremos que realizar para microcontrolador que utilicemos.

Para ello, activamos el menú desplegable **Source** (fuente), mostrándose las siguientes opciones:



# CONFIGURACIÓN DEL ENTORNO DE TRABAJO

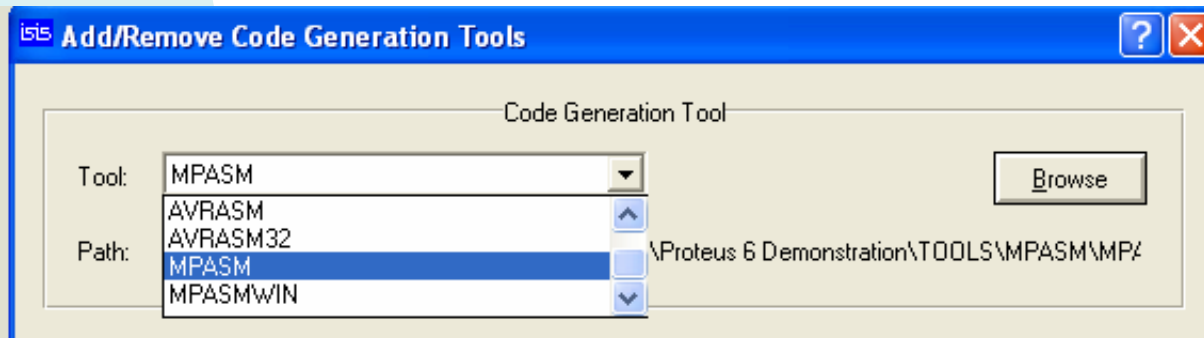
Escogemos opción *Define Code Generation Tools...*:  
con lo que se nos abrirá la siguiente ventana



# CONFIGURACIÓN DEL ENTORNO DE TRABAJO

En la ventana anterior podremos escoger el ensamblador o compilador que se utilizará para generar el fichero ejecutable a partir del fichero fuente, así como las opciones del mismo.

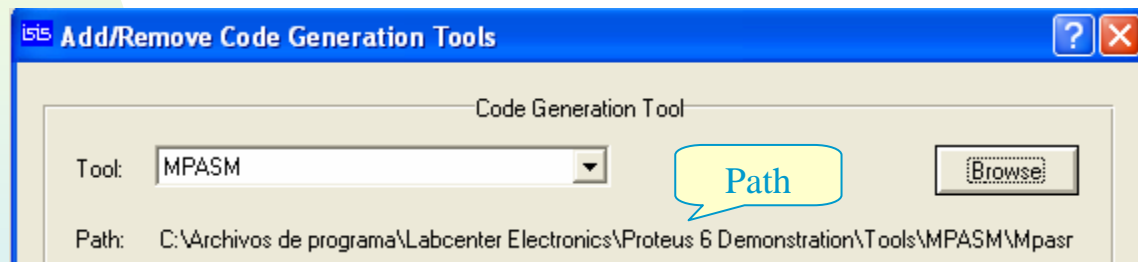
En nuestro caso escogemos el ensamblador *MPASM* de *Microchip*





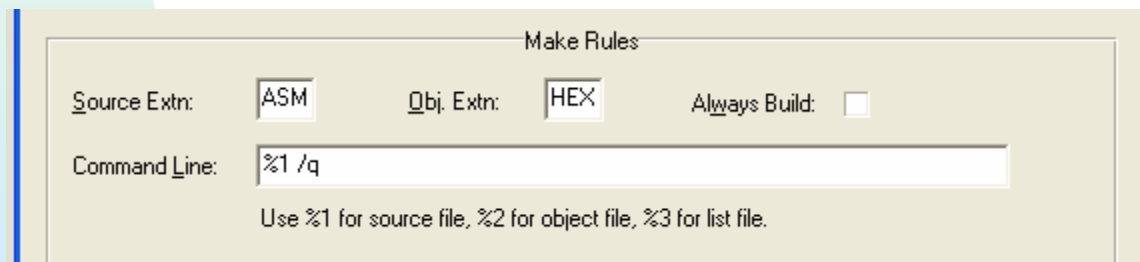
# CONFIGURACIÓN DEL ENTORNO DE TRABAJO

Seguidamente damos el camino para encontrar dicha herramienta, para ellos pinchamos en *Browse*



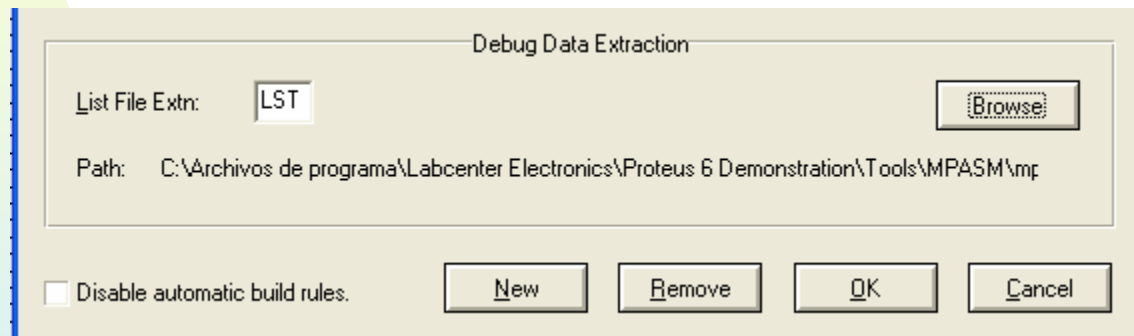
# CONFIGURACIÓN DEL ENTORNO DE TRABAJO

Las opciones contenidas en *Make Rules* nos muestran el tipo de fichero que deseamos ensamblar (**ASM**), y el código objeto que se producirá después de esta acción (**HEX**).



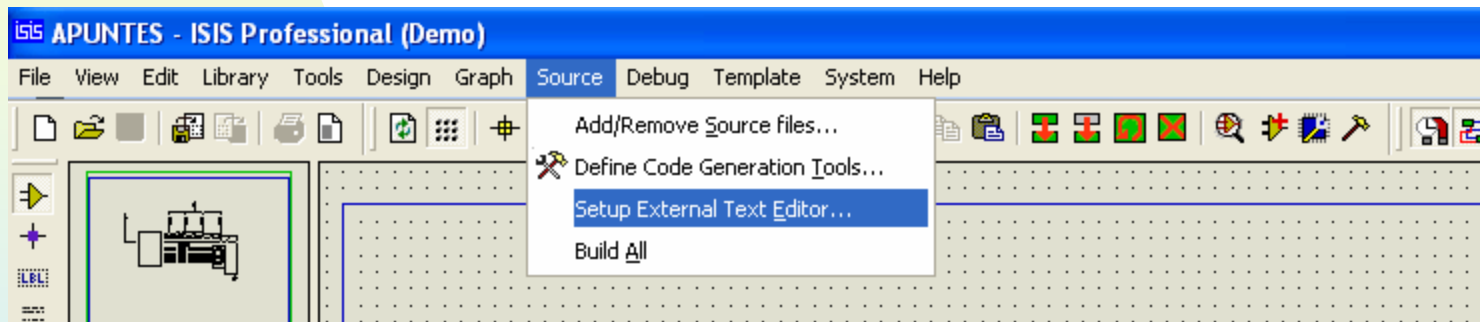
# CONFIGURACIÓN DEL ENTORNO DE TRABAJO

**La última opción consiste en indicar el Path de la herramienta de trabajo para ficheros listables, en nuestro caso MPASAMDDX.**



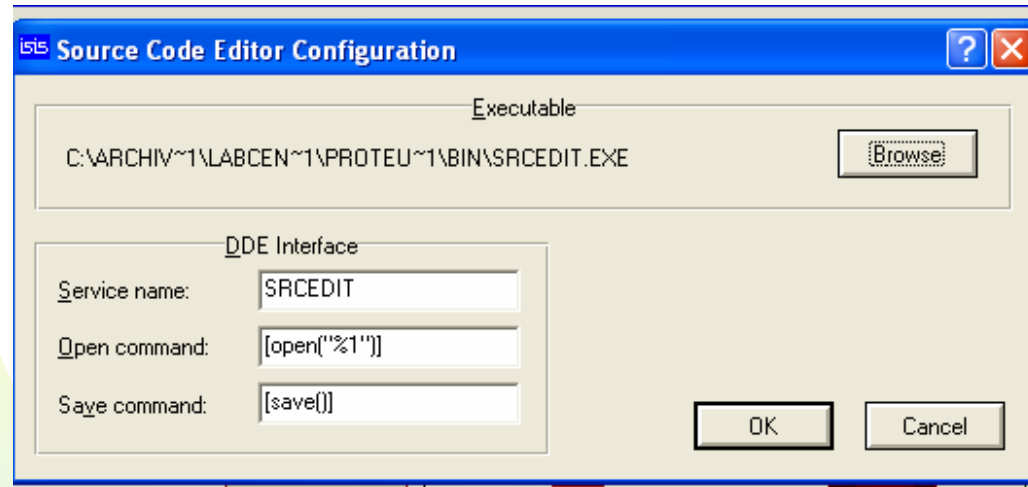
# CONFIGURACIÓN DEL ENTORNO DE TRABAJO

**Edición del programa fuente:** Nos resta editar el programa que queremos correr sobre el micro, pero previamente deberemos fijar el editor de texto que utilizaremos



Al seleccionar *Setup External Text Editor...* , aparece la siguiente pantalla:

# CONFIGURACIÓN DEL ENTORNO DE TRABAJO

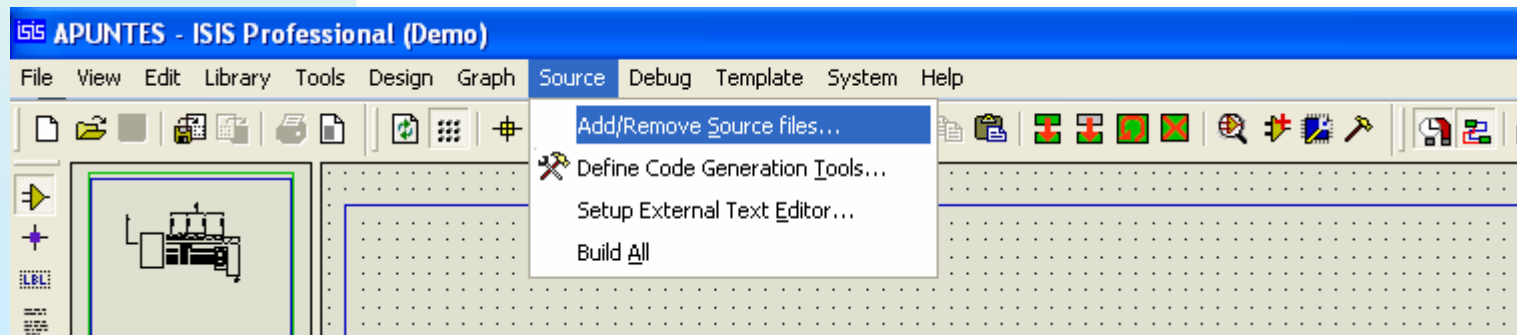


- Por defecto se nos da un editor (**SRDEDIT**), dicho editor puede ser cambiado por otro.
- Con el editor este podemos escribir el programa, o bien cambiar el **\*.ASM** y ensamblarlo sin necesidad de salir del entorno de PROTEUS.

# AÑADIENDO EL PROGRAMA FUENTE

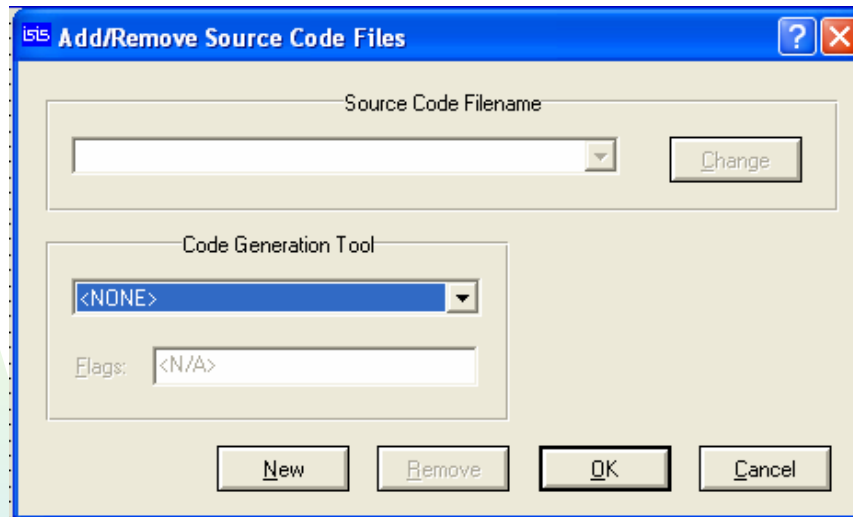
Téngase en cuenta que el programa ha podido ser escrito en otro entorno y con otro editor (por ejemplo en MPLAB).

Si el programa **\*.ASM** está ya escrito, lo deberemos asociar al diseño, para ello activamos la opción *Add/Remove Source files...*



# AÑADIENDO EL PROGRAMA FUENTE

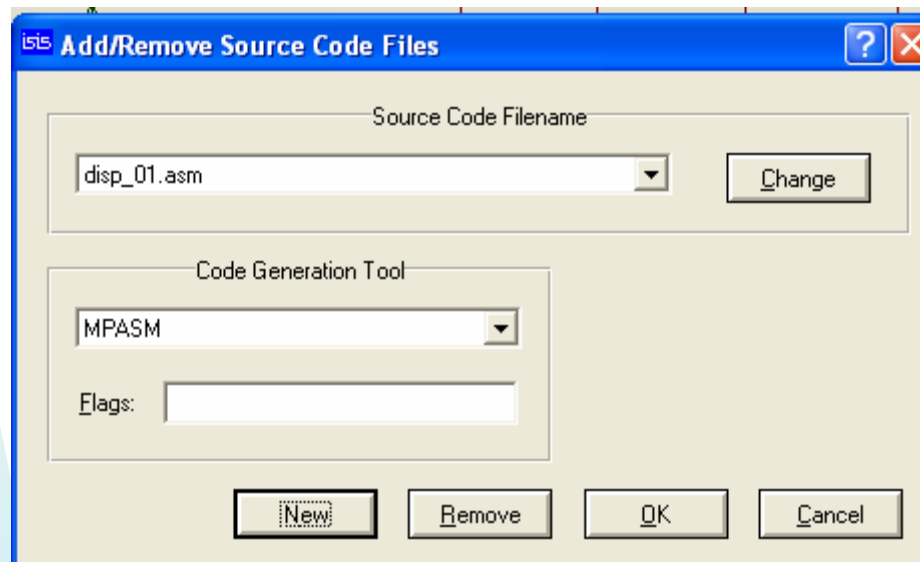
**El cuadro de diálogo que se nos presenta es:**



**En él seleccionamos la herramienta de ensamblado, e indicamos cual será el fichero a ensamblar**

# AÑADIENDO EL PROGRAMA FUENTE

**En nuestro caso damos el fichero que ha sido generado en el entorno MPLAB y en con el camino indicado:**

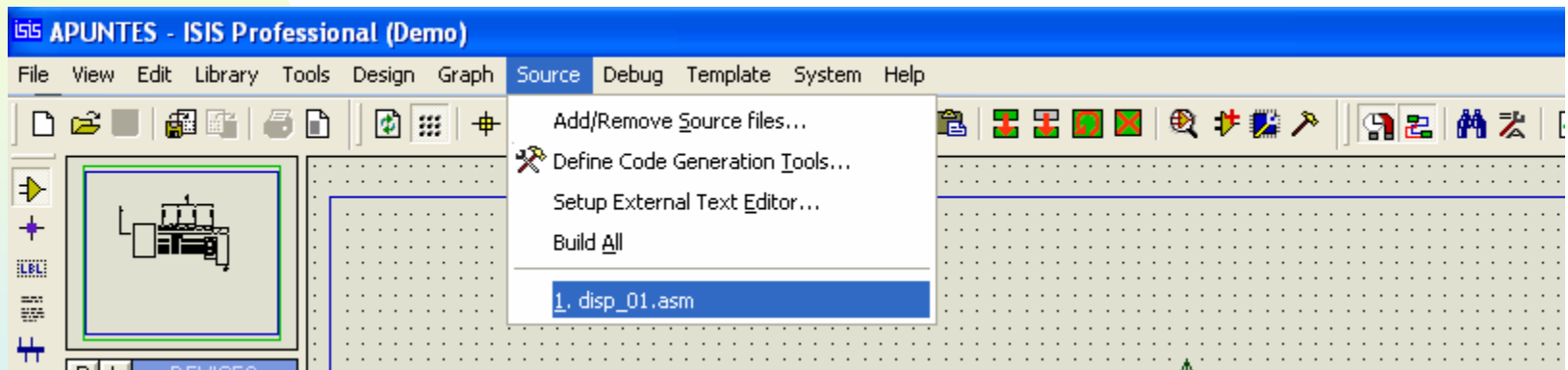


**Existe una opción de Flags, que sirve para ensamblados especiales**



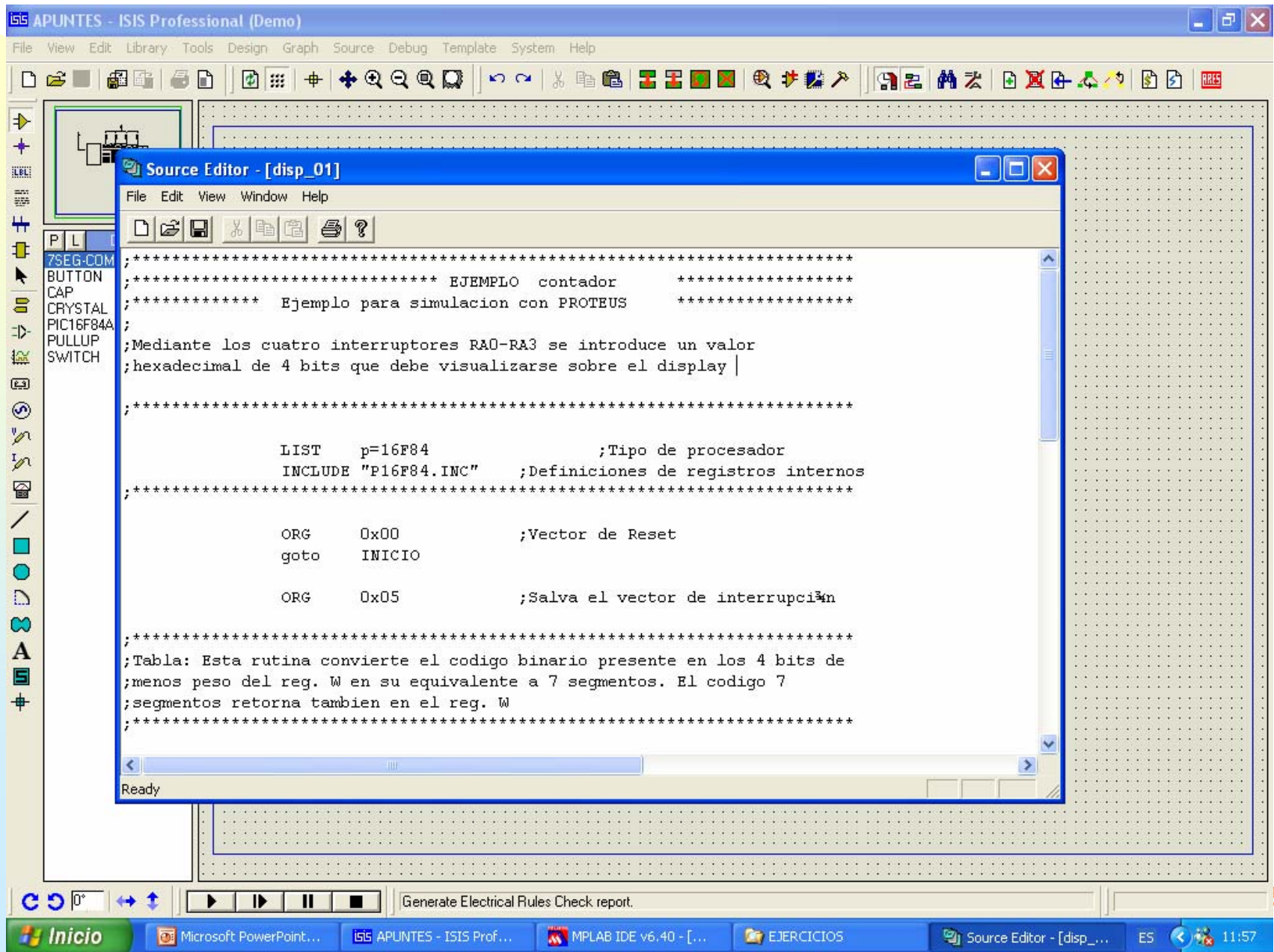
# AÑADIENDO EL PROGRAMA FUENTE

Una vez que hayamos pulsado **OK**, y si observamos de nuevo el menú **Source**, comprobaremos que el fichero ha sido añadido:

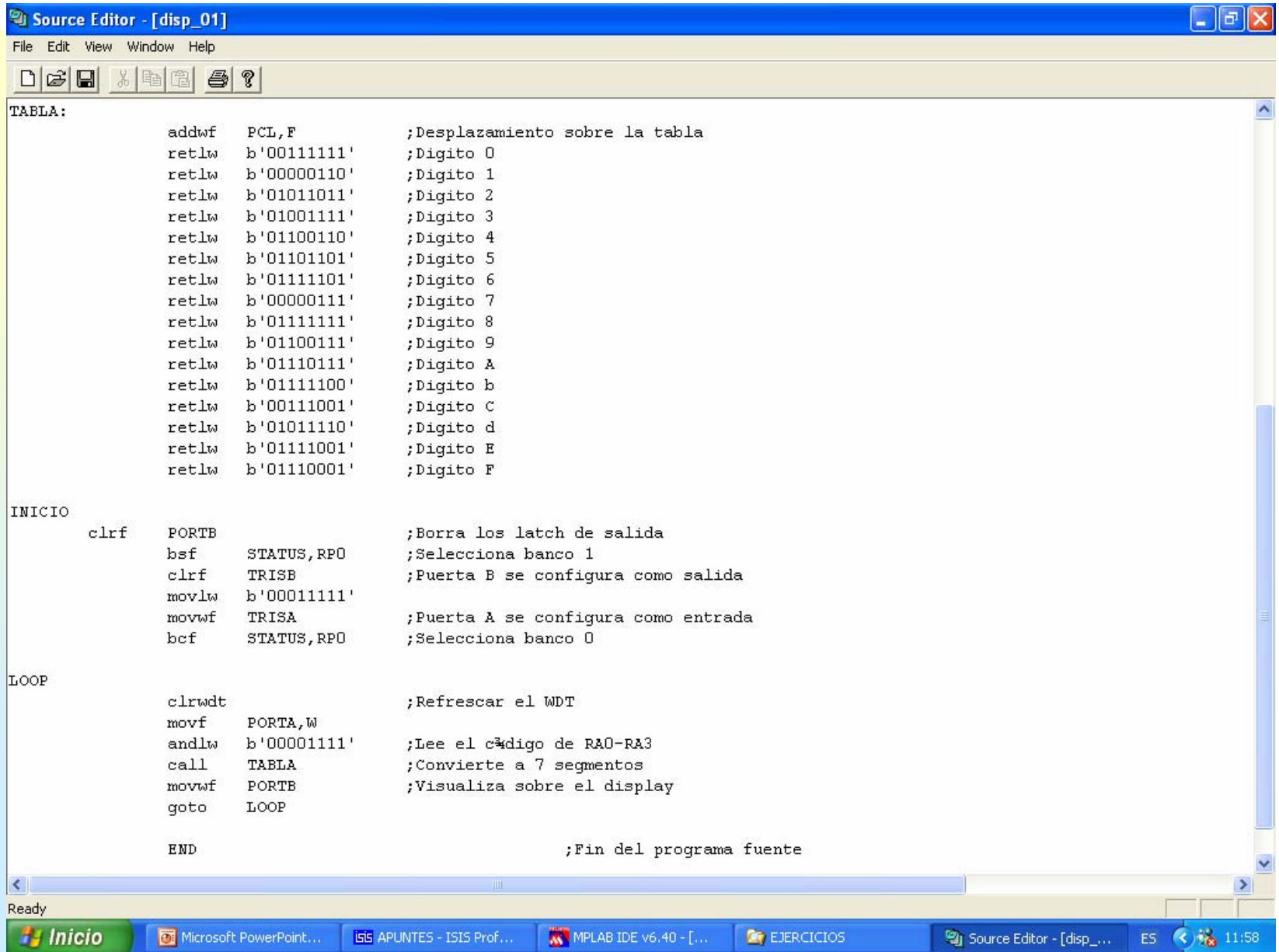


Si hacemos clic sobre él, se abrirá y podríamos hacer modificaciones

# AÑADIENDO EL PROGRAMA FUENTE



# AÑADIENDO EL PROGRAMA FUENTE



The screenshot shows a 'Source Editor' window with a menu bar (File, Edit, View, Window, Help) and a toolbar. The code is written in assembly language and is organized into three sections: TABLA, INICIO, and LOOP.

```
TABLA:
    addwf    PCL,F           ;Desplazamiento sobre la tabla
    retlw    b'00111111'     ;Digito 0
    retlw    b'00000110'     ;Digito 1
    retlw    b'01011011'     ;Digito 2
    retlw    b'01001111'     ;Digito 3
    retlw    b'01100110'     ;Digito 4
    retlw    b'01101101'     ;Digito 5
    retlw    b'01111101'     ;Digito 6
    retlw    b'00000111'     ;Digito 7
    retlw    b'01111111'     ;Digito 8
    retlw    b'01100111'     ;Digito 9
    retlw    b'01110111'     ;Digito A
    retlw    b'01111100'     ;Digito b
    retlw    b'00111001'     ;Digito C
    retlw    b'01011110'     ;Digito d
    retlw    b'01111001'     ;Digito E
    retlw    b'01110001'     ;Digito F

INICIO
    clrf     PORTB           ;Borra los latch de salida
    bsf      STATUS,RPO      ;Selecciona banco 1
    clrf     TRISB           ;Puerta B se configura como salida
    movlw    b'00011111'     ;
    movwf    TRISA           ;Puerta A se configura como entrada
    bcf      STATUS,RPO      ;Selecciona banco 0

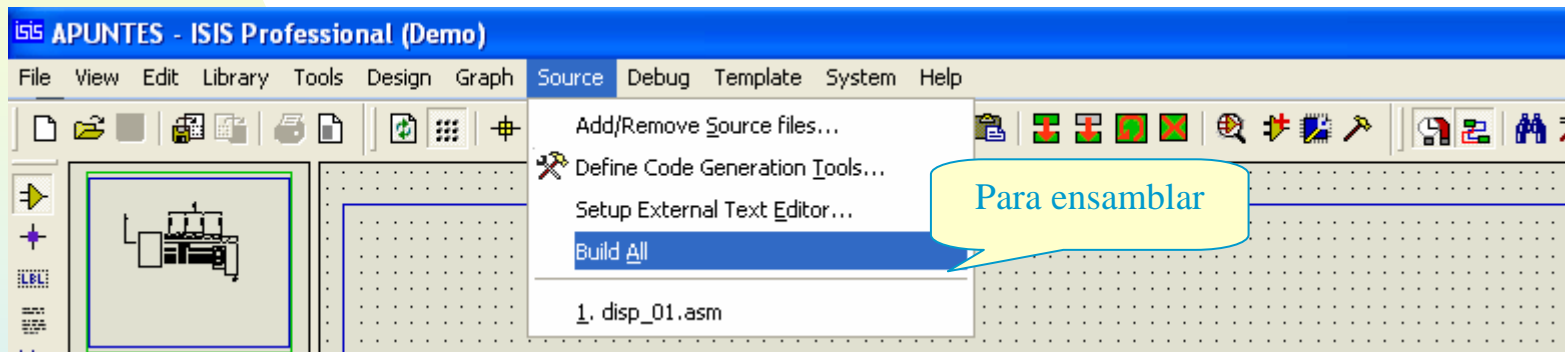
LOOP
    clrw     PORTA,W          ;Refrescar el WDT
    movf     PORTA,W          ;Lee el código de RA0-RA3
    andlw    b'00001111'     ;
    call     TABLA            ;Convierte a 7 segmentos
    movwf    PORTB           ;Visualiza sobre el display
    goto     LOOP

    END                      ;Fin del programa fuente
```

The status bar at the bottom shows 'Ready' and a taskbar with several open applications: Inicio, Microsoft PowerPoint..., ISIS APUNTES - ISIS Prof..., MPLAB IDE v6.40 - [...], EJERCICIOS, and Source Editor - [disp\_...]. The system clock shows 11:58.

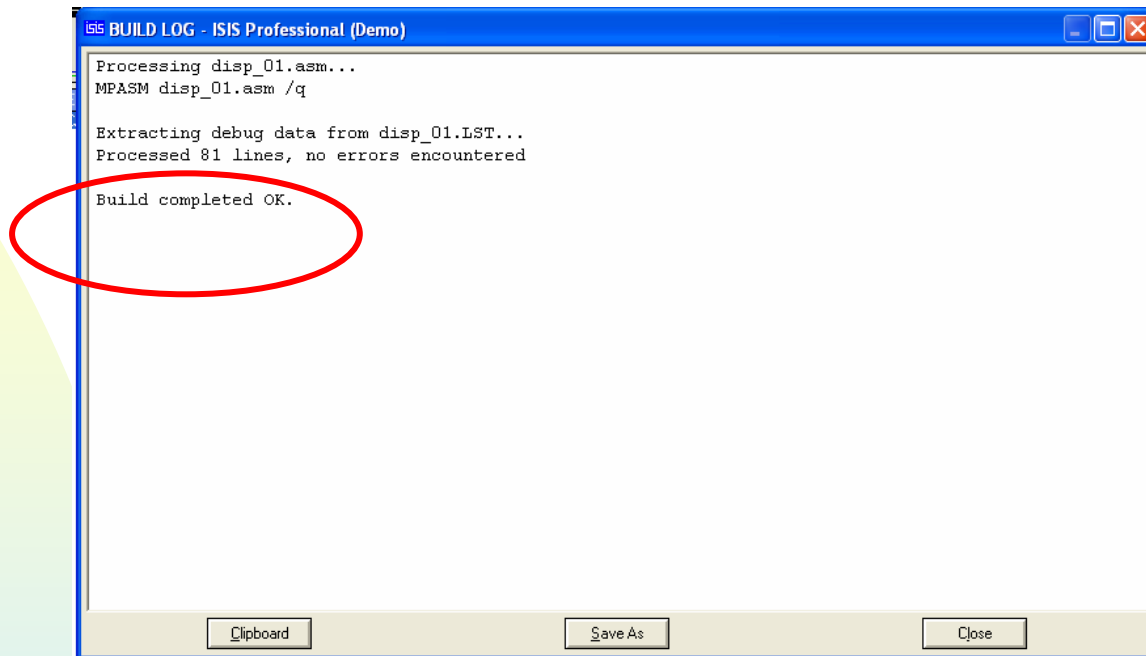
# ENSAMBLADO DEL PROGRAMA FUENTE

Una vez editado el programa, nos resta ensamblarlo, esto es posible sin salir del entorno. Para ello lanzamos la opción **Build All** (construir).



Esta acción provoca que se nos presente una ventana de mensajes en las que se nos indicará si todo ha ido bien y que se han generado los ficheros **HEX y LST**.

# ENSAMBLADO DEL PROGRAMA FUENTE



The screenshot shows a window titled "ISIS BUILD LOG - ISIS Professional (Demo)". The text inside the window is as follows:

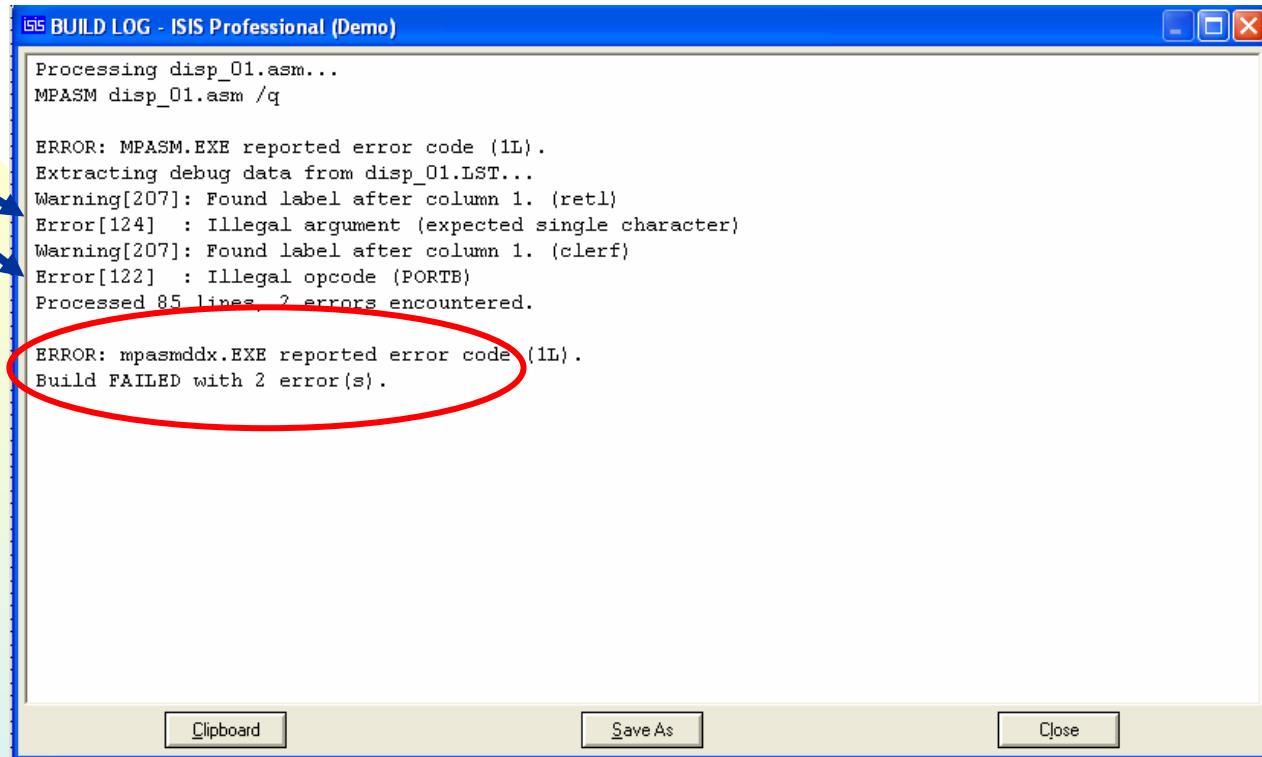
```
Processing disp_01.asm...  
MPASM disp_01.asm /q  
  
Extracting debug data from disp_01.LST...  
Processed 81 lines, no errors encountered  
  
Build completed OK.
```

The text "Build completed OK." is circled in red. At the bottom of the window, there are three buttons: "Clipboard", "Save As", and "Close".

**En el ejemplo anterior se ha producido el ensamblado del programa sin errores.**

**Si existieran errores se nos indicará que tipo de error existe en nuestro programa.**

# ENSAMBLADO DEL PROGRAMA FUENTE



```
Processing disp_01.asm...
MPASM disp_01.asm /q

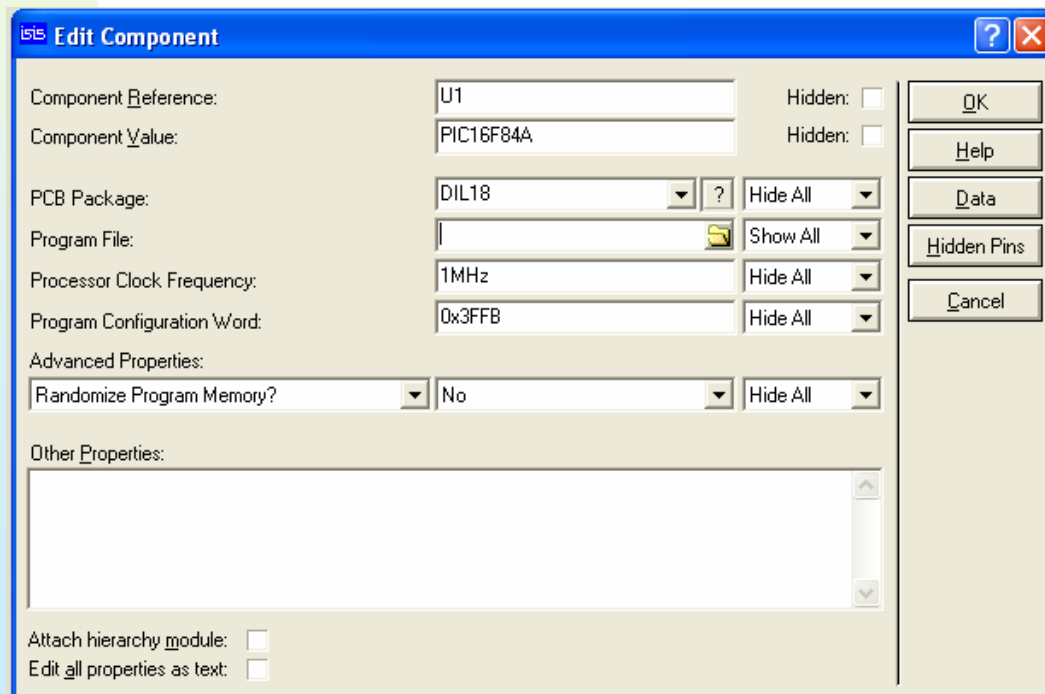
ERROR: MPASM.EXE reported error code (1L).
Extracting debug data from disp_01.LST...
Warning[207]: Found label after column 1. (retl)
Error[124] : Illegal argument (expected single character)
Warning[207]: Found label after column 1. (clerf)
Error[122] : Illegal opcode (PORTB)
Processed 85 lines, 2 errors encountered.

ERROR: mpasmdx.EXE reported error code (1L).
Build FAILED with 2 error(s).
```

En el caso del ejemplo, existen errores, por lo que se deberá editar de nuevo el programa \*.ASM y corregir los errores.

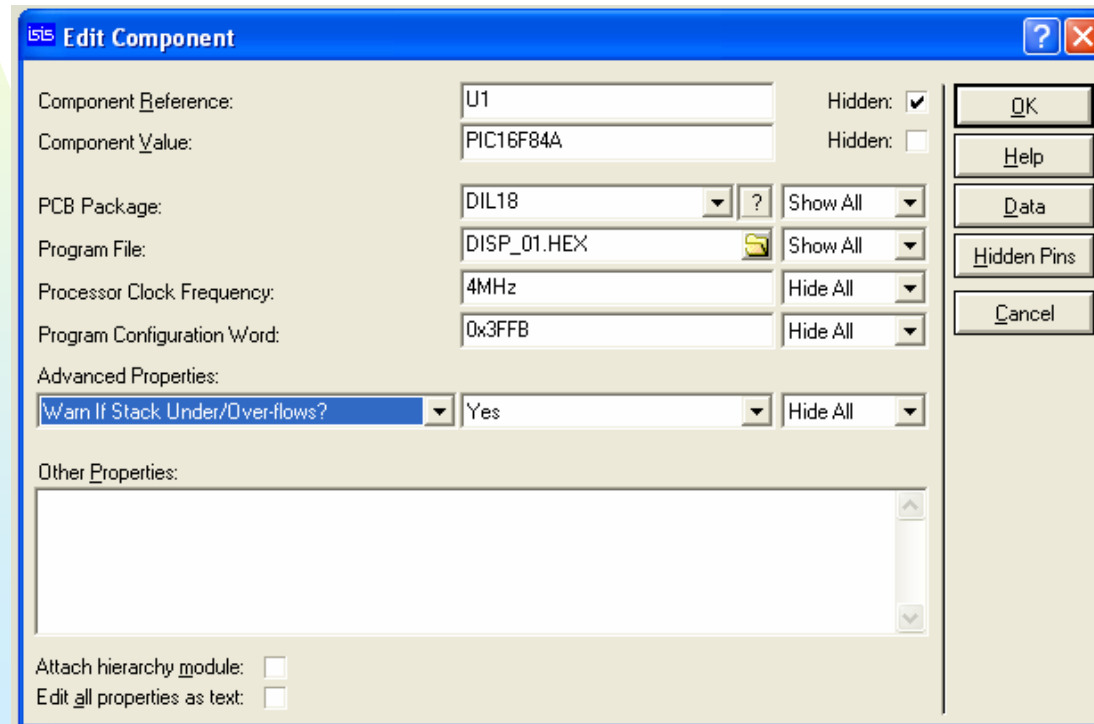
# “INSERCIÓN” DEL PROGRAMA EN EL $\mu$ CONTROLADOR

Una vez generados los ficheros HEX y LST, deberemos asociar al  $\mu$ controlador el programa **.HEX**, para ello y en el diseño, seleccionamos el micro y lo editamos, esto provocará que se nos presente el siguiente cuadro de diálogo:



# “Inserción” Del Programa En El $\mu$ Controlador

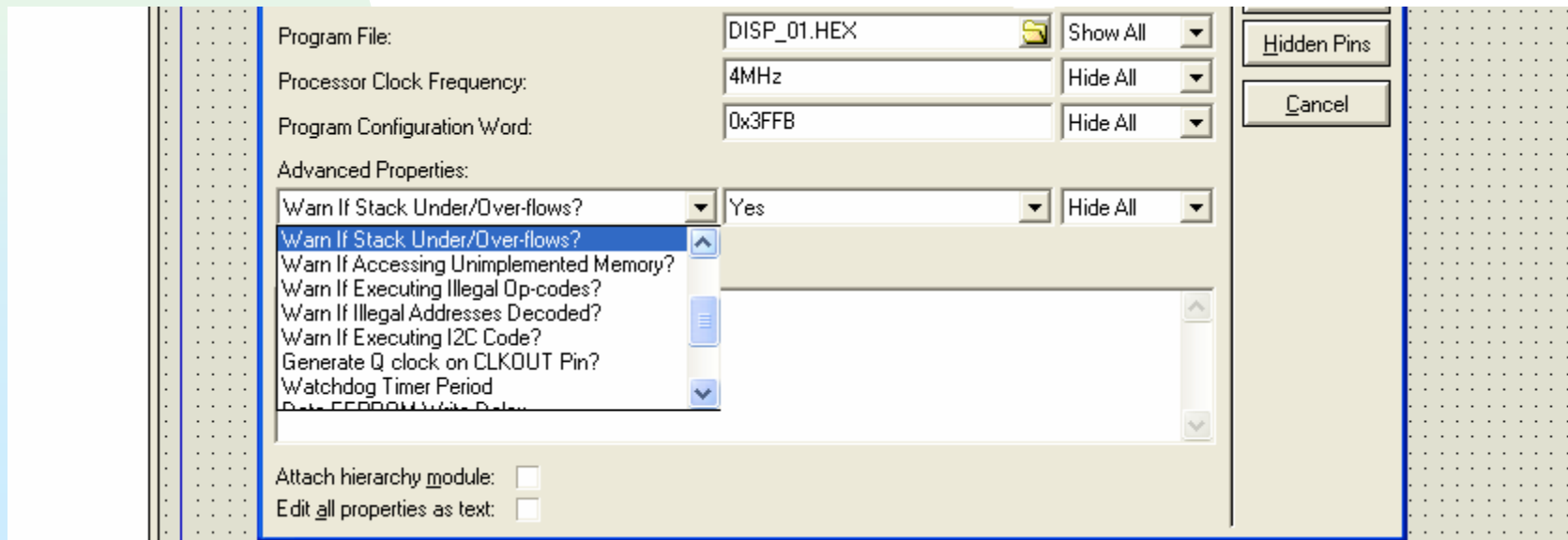
Sobre este cuadro de diálogo, indicamos el programa que se ejecutará en el microcontrolador, la frecuencia del reloj, palabra de configuración...





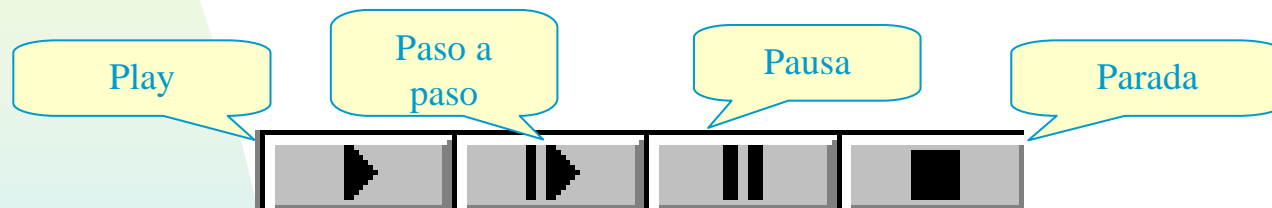
# “Inserción” Del Programa En El $\mu$ Controlador

**Existen otras opciones mas avanzadas y que pueden se seleccionadas, como periodo del perro guardián, avisos si se produce overflow o underflow de la pila, avisos si se intenta acceder a una posición de memoria no implementada etc.**




# SIMULACIÓN DE CIRCUITOS CON μCONTROLADORES EN PROTEUS

Llegados a este momento ya podemos pasar a la simulación y animación del circuito para ello disponemos de los botones de control de la barra de animación y que funcionan como en un cassette de audio.



**Play:** A su pulsación, el botón cambia a color verde, el PIC comienza a ejecutar el programa en modo continuo, se inicia la animación.

# SIMULACIÓN Y DEPURACIÓN μCONTROLADORES EN PROTEUS

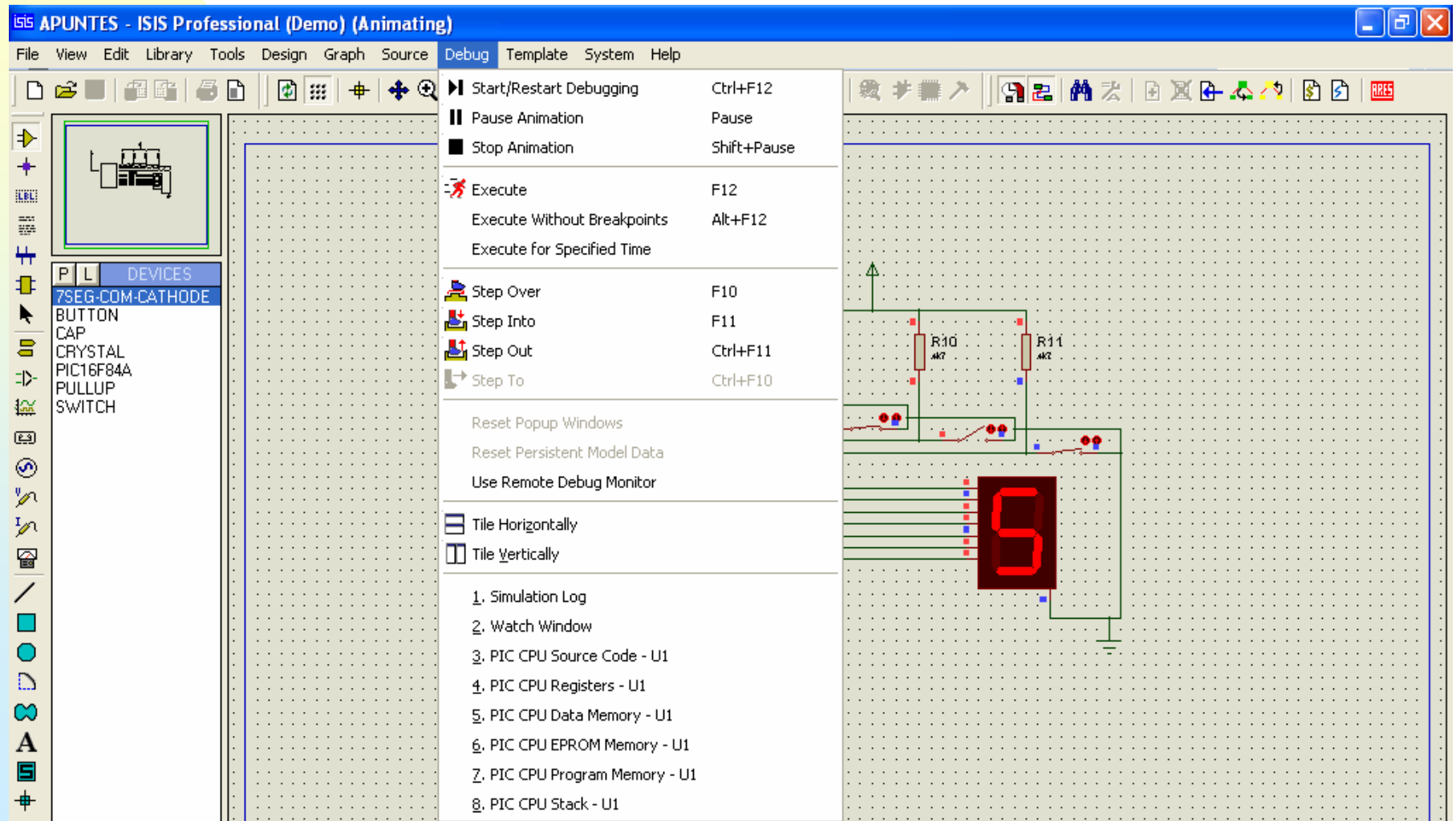
Si el programa no funciona como deseamos, deberemos depurarlo, para ello se hace preciso parar el programa, mediante la tecla de **STOP** .

Y seguidamente ejecutar el programa **PASO A PASO**, mediante la tecla .

En este modo de trabajo, podemos utilizar las opciones de depuración del menú *Debug*.

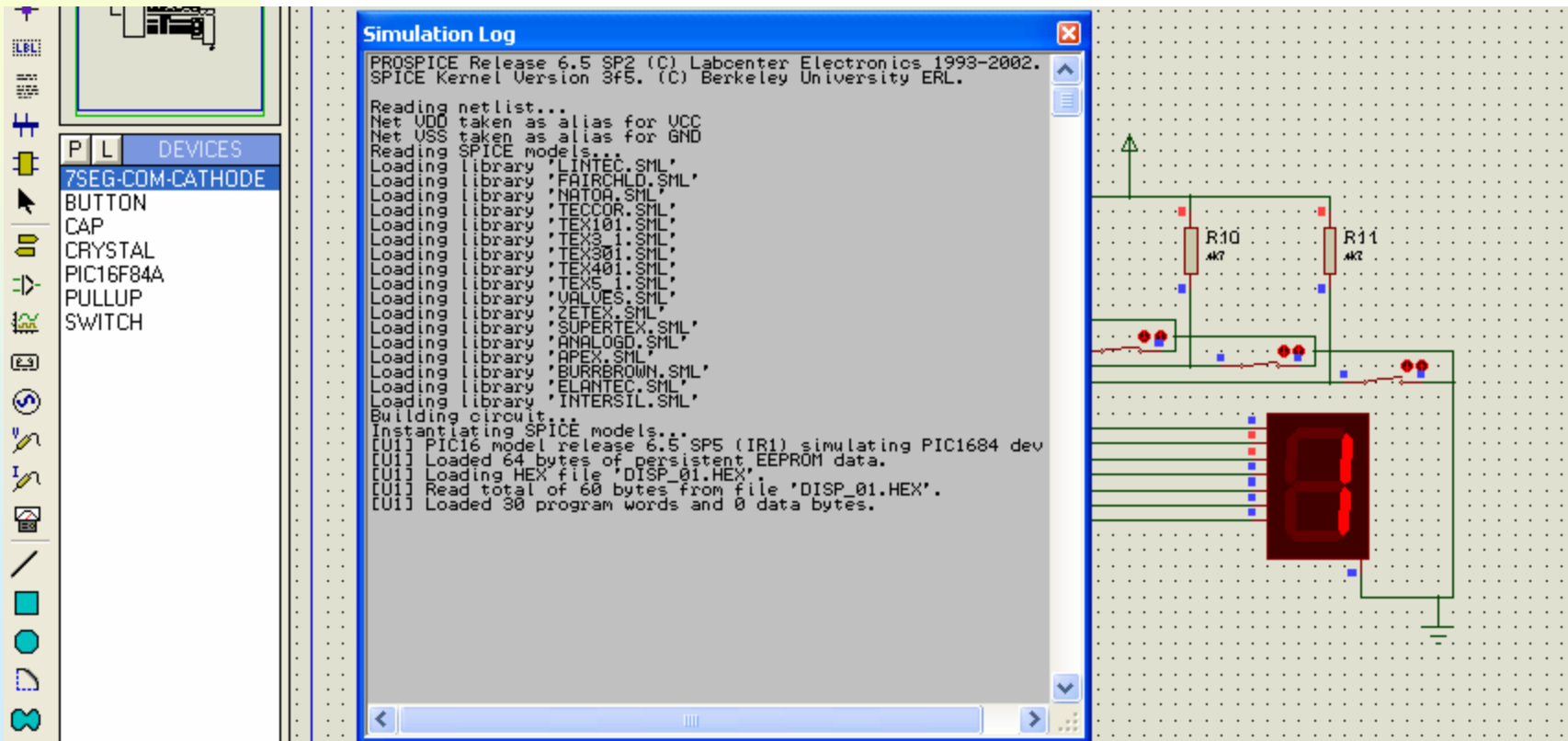
# LA DEPURACIÓN (Debug)

Las opciones de depuración que posee Proteus son:



# LAS OPCIONES DE DEBUG

## 1. *Simulation Log*: muestra los mensajes propios de la simulación

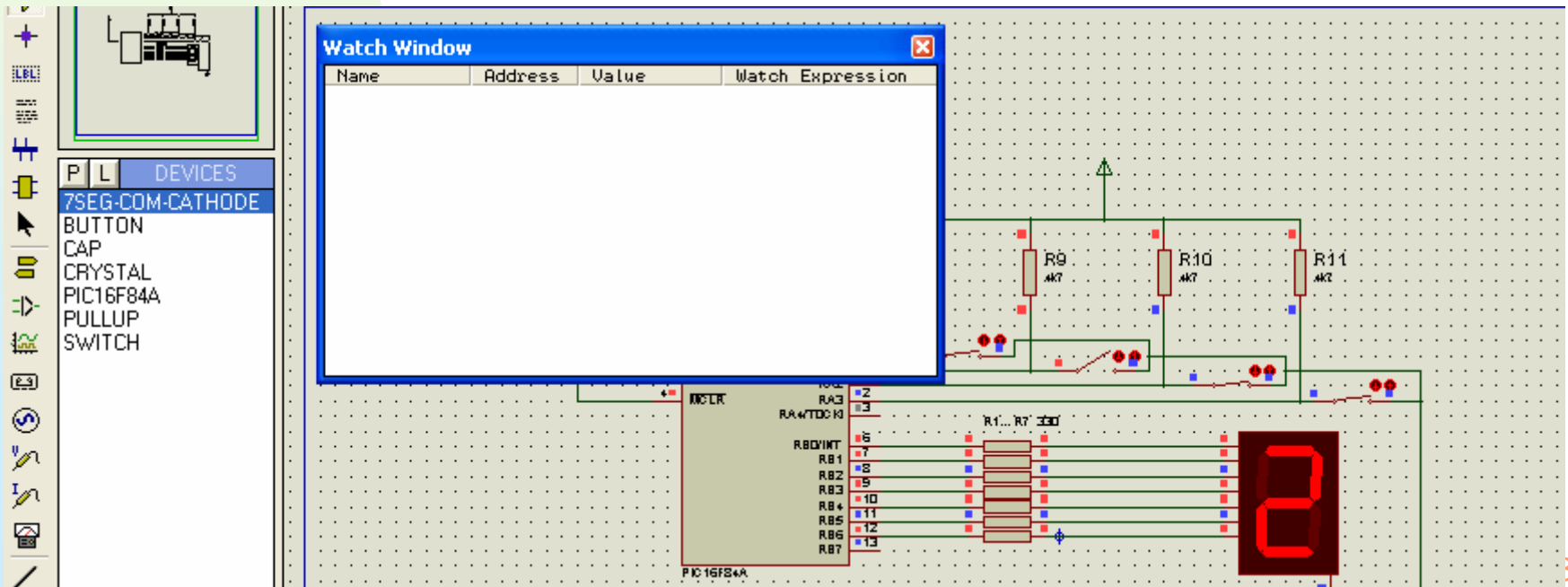


# LAS OPCIONES DE DEBUG

## Watch Window

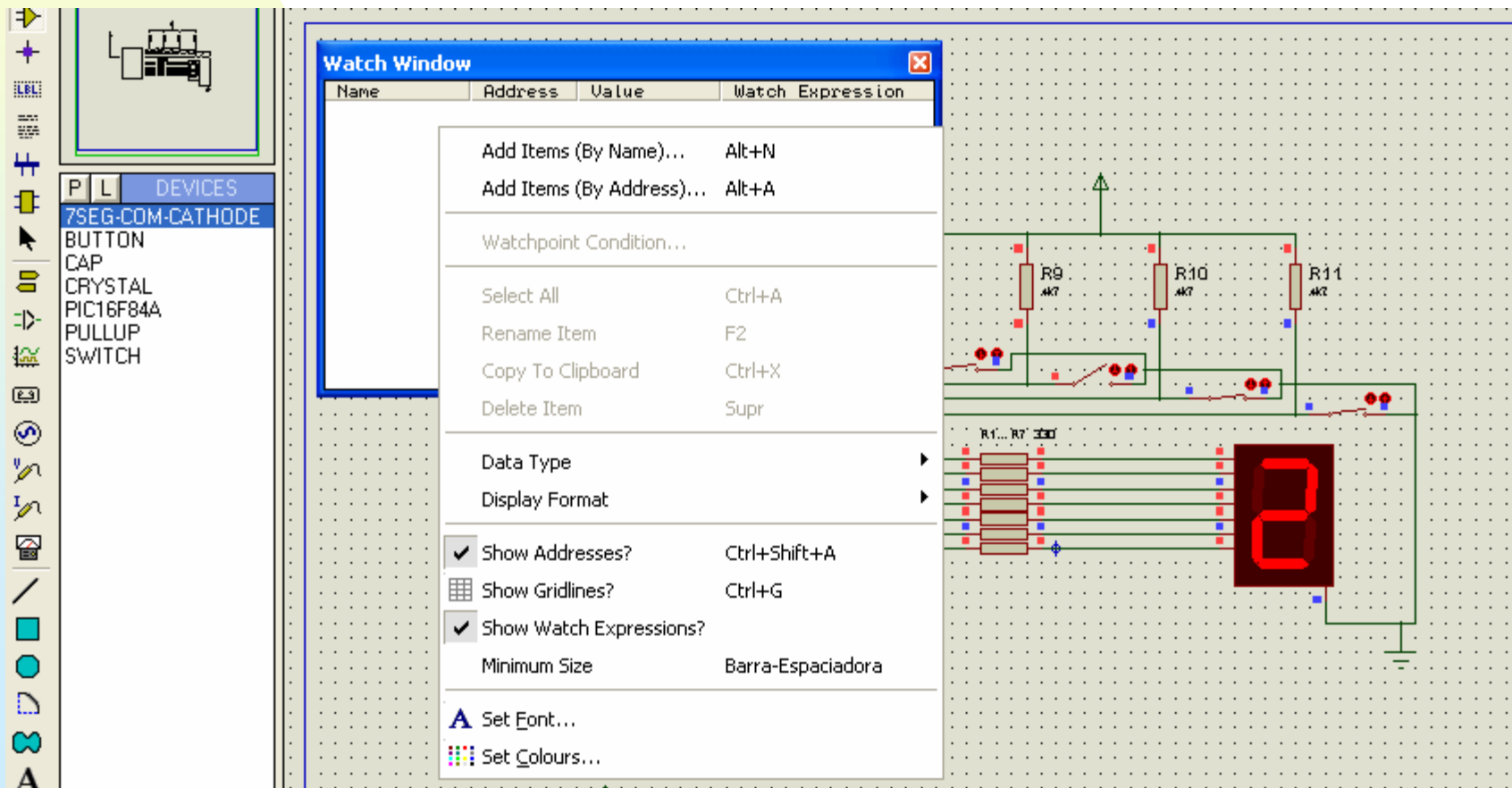
2. *Watch Window*: Permite observar el valor de variables , etiquetas y posiciones de memoria en un instante determinado de la ejecución del programa.

Su pulsación hace que se nos muestre la siguiente ventana:



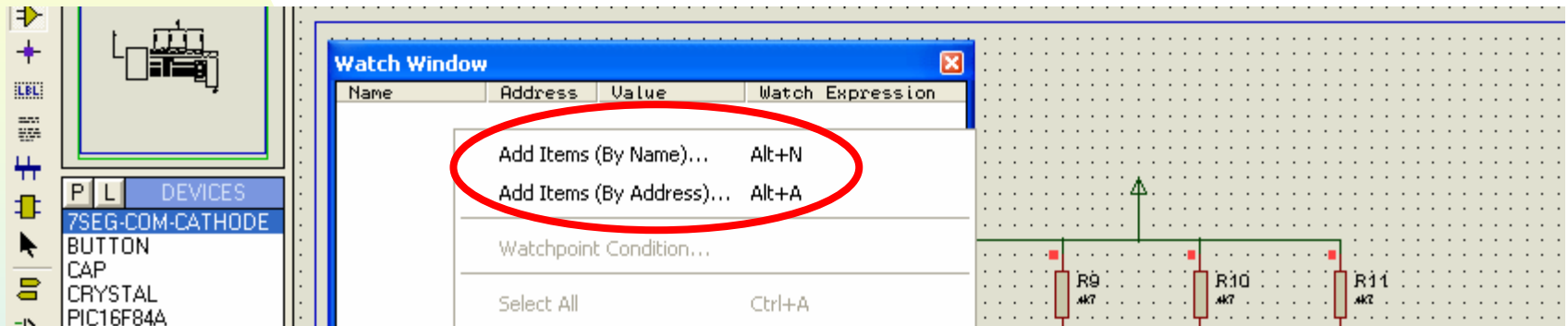
# LAS OPCIONES DE DEBUG

Para incluir variables, hacemos clic con el botón secundario el ratón dentro de dicha ventana.



# LAS OPCIONES DE DEBUG

Como se puede comprobar se pueden incluir variables indicando su nombre o bien su dirección:

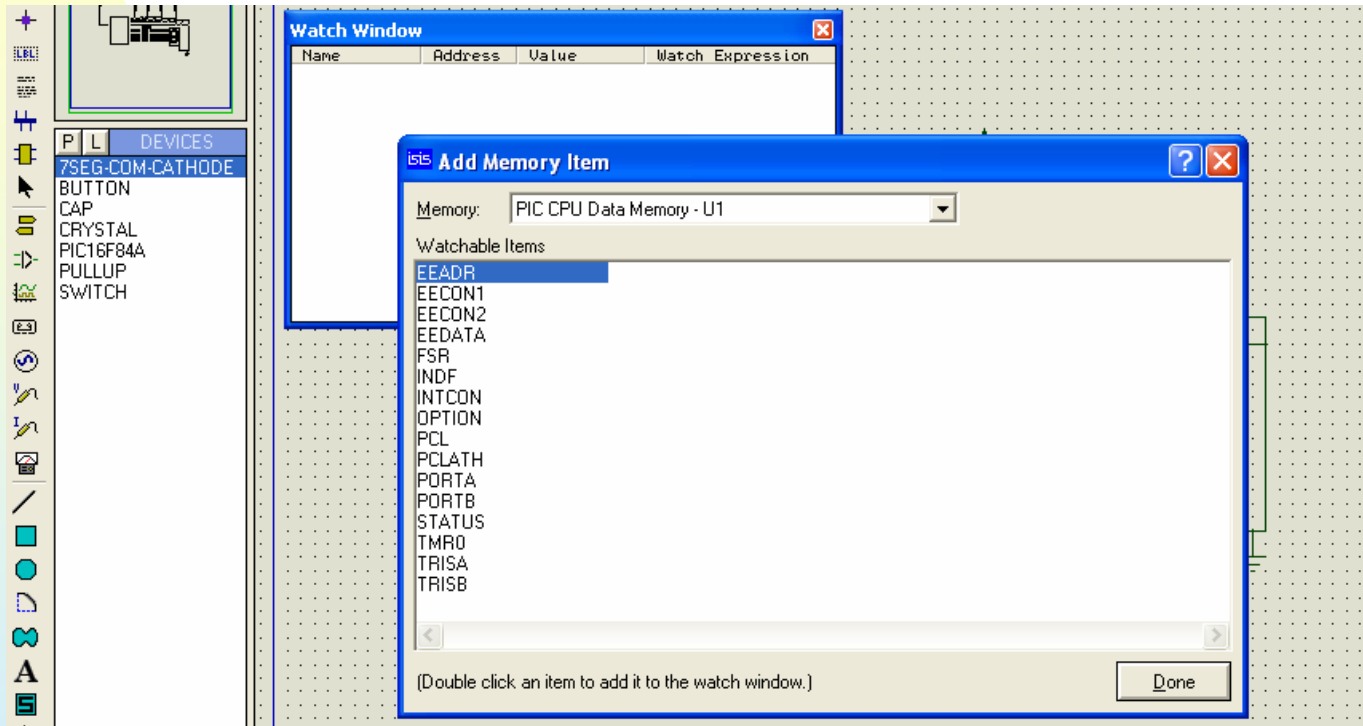


Pasemos a incluir alguna de ellas, mediante el nombre, para ello hacemos clic en *Add Items (By Name)*



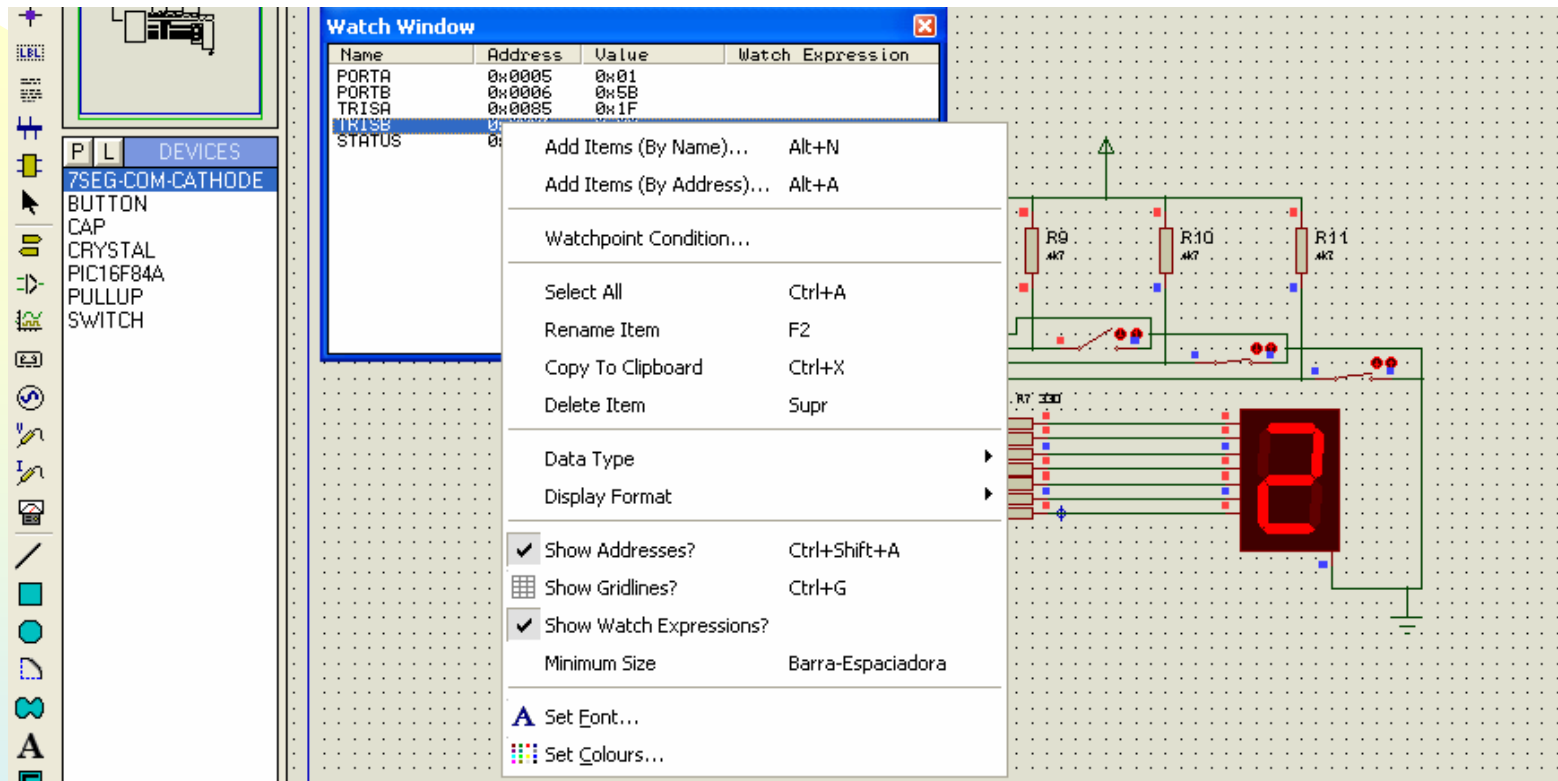
# LAS OPCIONES DE DEBUG

**La pantalla que se nos presenta es:**



**Haciendo doble clic sobre cualquier variable queda insertada en la ventana:**

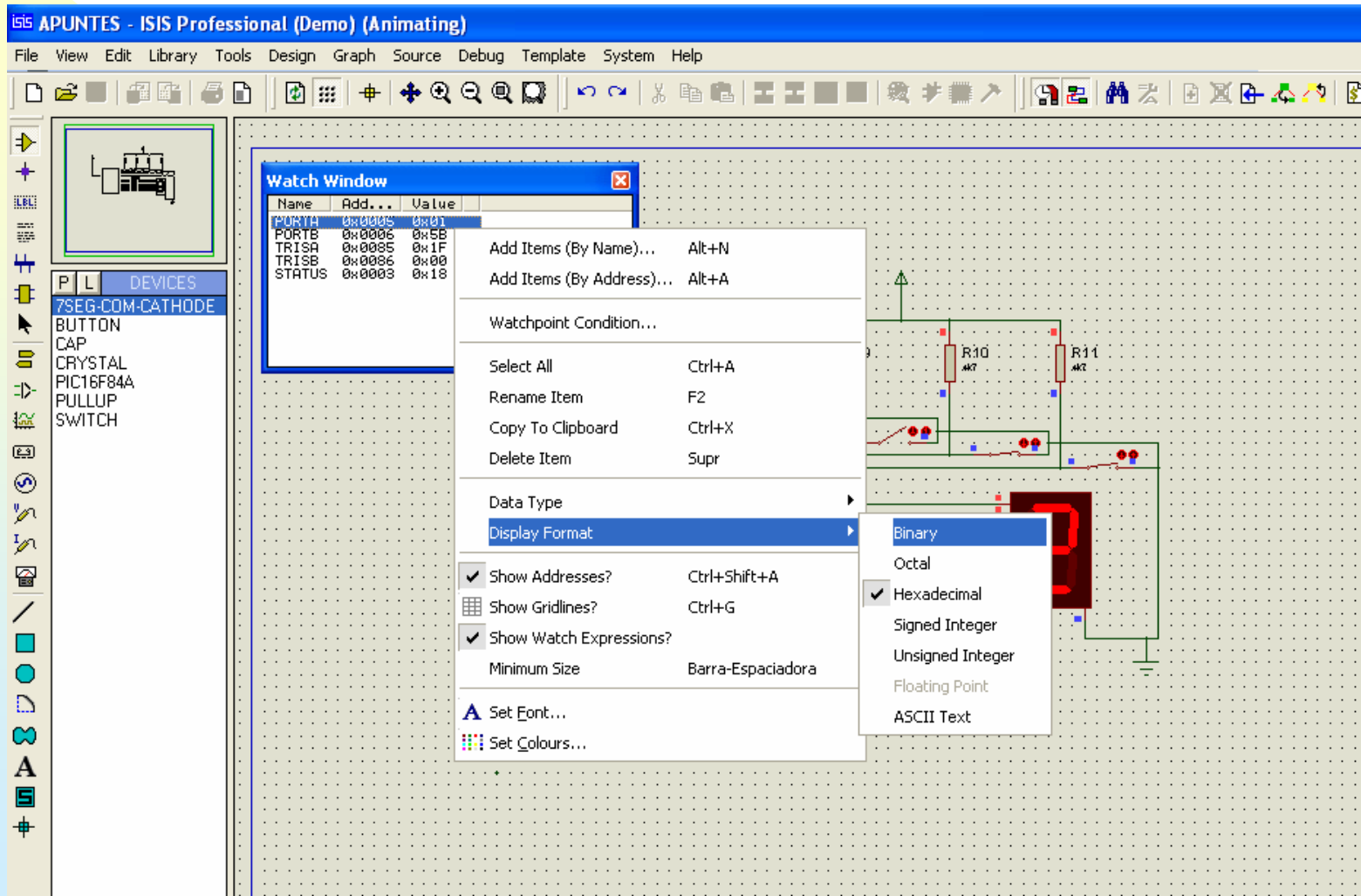
# LAS OPCIONES DE DEBUG



**Tenga presente que en cualquier momento se puede editar esta ventana para incluir, eliminar variables o modificar los formatos de visualización.**

# LAS OPCIONES DE DEBUG

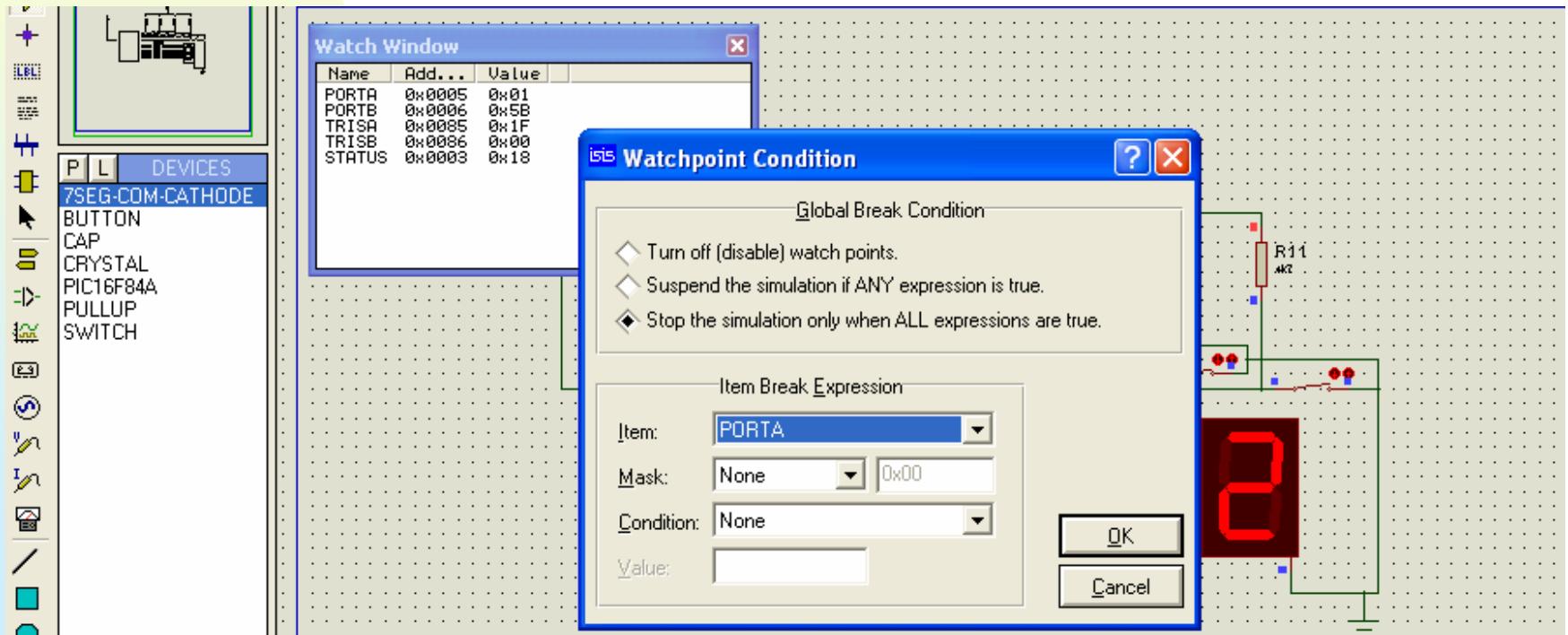
## WATCH WINDOW



# LAS OPCIONES DE DEBUG

## Watch point

**También es posible fijar una condición en las variables para detener la animación**



# LAS OPCIONES DE DEBUG

## PIC CPU Source Code –U1

Nos permite ver el código fuente del programa en una ventana (en este caso en ensamblador)

The screenshot displays the 'PIC CPU Source Code - U1' window in Proteus, showing the assembly code for a program named 'DISP\_01.SDI'. The code is written in assembly language and includes comments in Spanish. The program is a 4-bit counter that uses four pushbuttons (RA0-RA3) to input a hexadecimal value, which is then displayed on a 7-segment display. The code is organized into sections: initialization, a table of 7-segment patterns, and a main loop.

```
***** EJEMPLO contador *****
***** Ejemplo para simulacion con PROTEUS *****
;Mediante los cuatro interruptores RA0-RA3 se introduce un valor
;hexadecimal de 4 bits que debe visualizarse sobre el display
;*****
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 interrupcion
;*****
;Tabla: Esta rutina convierte el codigo binario presente en los 4 bits de
;menos peso del reg. W en su equivalente a 7 segmentos. El codigo 7
;segmentos retorna tambien en el reg. W
;*****
TABLA:
addwf PCL,F          ;Desplazamiento sobre la tabla
retlw b'00111111'      ;Digito 0
retlw b'00000110'      ;Digito 1
retlw b'01011011'      ;Digito 2
retlw b'01001111'      ;Digito 3
retlw b'01100110'      ;Digito 4
retlw b'01101101'      ;Digito 5
retlw b'01111101'      ;Digito 6
retlw b'00000111'      ;Digito 7
retlw b'01111111'      ;Digito 8
retlw b'01100111'      ;Digito 9
retlw b'01110111'      ;Digito A
retlw b'01111100'      ;Digito B
retlw b'00111001'      ;Digito C
retlw b'01011110'      ;Digito D
retlw b'01111001'      ;Digito E
retlw b'01110001'      ;Digito F
INICIO
clrf PORTB          ;Borra los latch de salida
bsf STATUS,RP0      ;Selecciona banco 1
clrf TRISB          ;Puerta B se configura como salida
movlw b'00011111'    ;Puerta A se configura como entrada
movwf TRISA
bcf STATUS,RP0      ;Selecciona banco 0
LOOP
clawdt          ;Refrescar el WDT
movf PORTA,W
andlw b'00011111'    ;Lee el código de RA0-RA3
call TABLA          ;Convierte a 7 segmentos
movwf PORTB          ;Visualiza sobre el display
goto LOOP
END
;Fin del programa fuente
```

On the right side of the screenshot, a circuit diagram is visible, showing a 7-segment display connected to a PIC microcontroller. The display is currently showing the number '3'. The circuit includes a resistor R11 and a 5V power supply.

# **LAS OPCIONES DE DEBUG**

## **PIC CPU Source Code –U1**

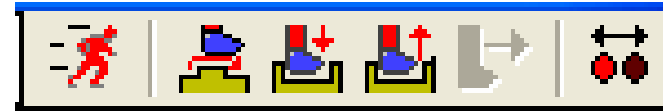
**Dentro de esta ventana podemos entre otras cosas:**

- **Ejecutar el programa en modo paso a paso**
- **Establecer puntos de ruptura**
- **Ir a una determinada línea de programa**
- **Buscar un texto en el código fuente**
- **Ir a una determinada dirección de memoria**
- **Activar/desactivar la visualización de los códigos de operación**
- **Cambiar la asignación de colores para direcciones, instrucciones, datos ...**

# LAS OPCIONES DE DEBUG


## PIC CPU Source Code –U1

Controles de la simulación:



 ***Run the simulation***: la simulación pasa a ejecutarse en modo continuo. No sirve para la depuración.

 ***Step Into*** : ejecuta una única instrucción, si se encuentra con una función/subrutina, entra en ella.

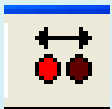
 ***Step Over*** : ejecuta una única instrucción, si se encuentra con una función/subrutina, la ejecuta como si se tratara una única instrucción (no entra en ella).

# LAS OPCIONES DE DEBUG

## PIC CPU Source Code –U1



***Step Out:*** Ejecuta en modo continuo todas las instrucciones hasta que encuentra un retorno de subrutina. Si la simulación se encuentra dentro de una subrutina (nos saca de la subrutina), si no encuentra un retorno de subrutina, pasa a ejecutar el programa en modo continuo.

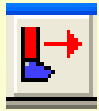


***Toggle (Set/Clear) Breakpoint:*** Habilita/deshabilita puntos de ruptura en la instrucción seleccionada mediante el ratón.



# LAS OPCIONES DE DEBUG

## PIC CPU Source Code –U1

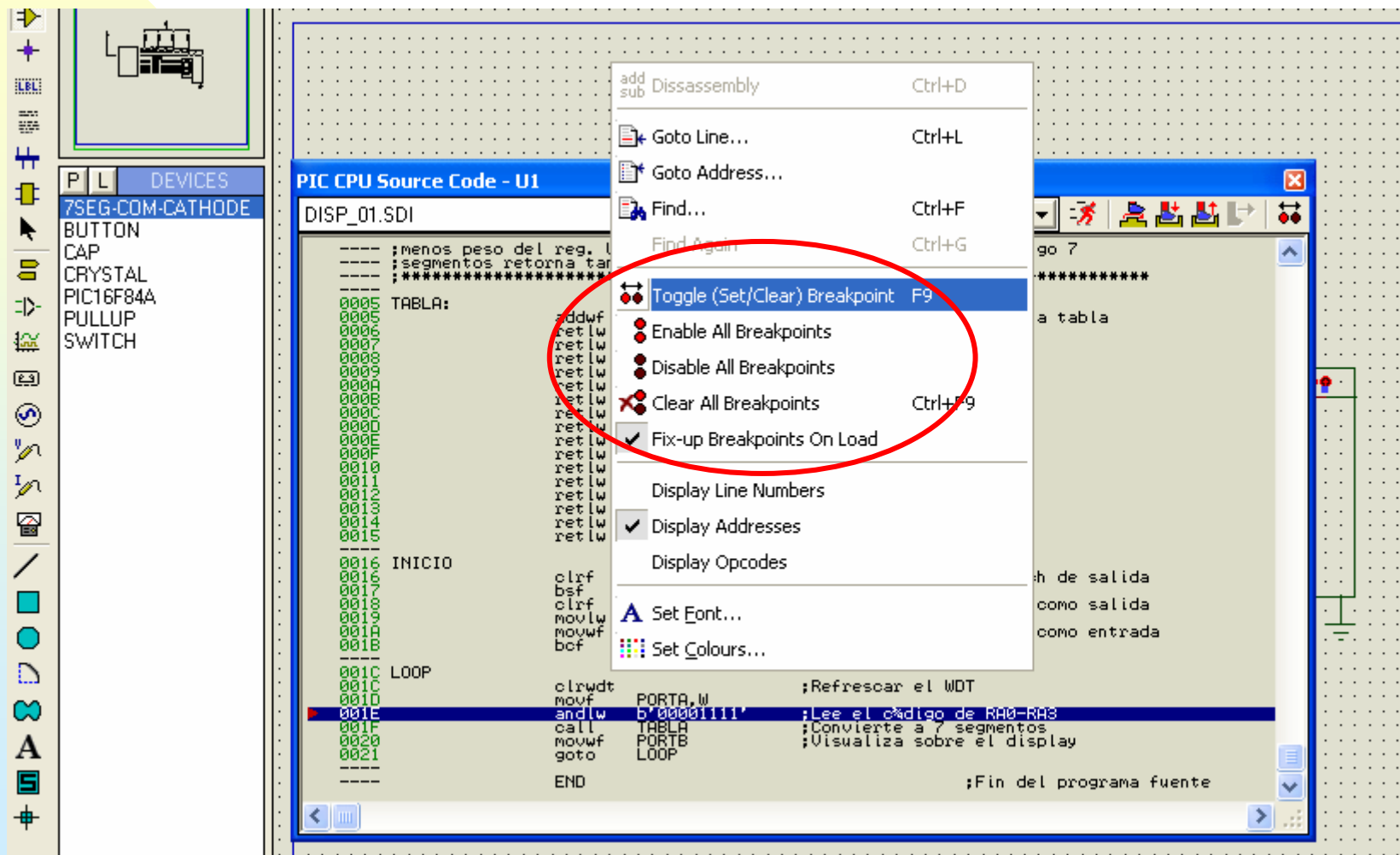


***Step To:*** Ejecuta en modo continuo todas las instrucciones hasta que encuentra un punto de ruptura, se activa si hay algún break point activado.

Todas las opciones anteriores así como las referentes a los puntos de ruptura (habilitación, eliminación se pueden seleccionar si hacemos clic con el botón secundario del ratón sobre la ventana *Source Code*.

# LAS OPCIONES DE DEBUG

## PIC CPU Source Code –U1



# LAS OPCIONES DE DEBUG

## PIC CPU Source Code –U1

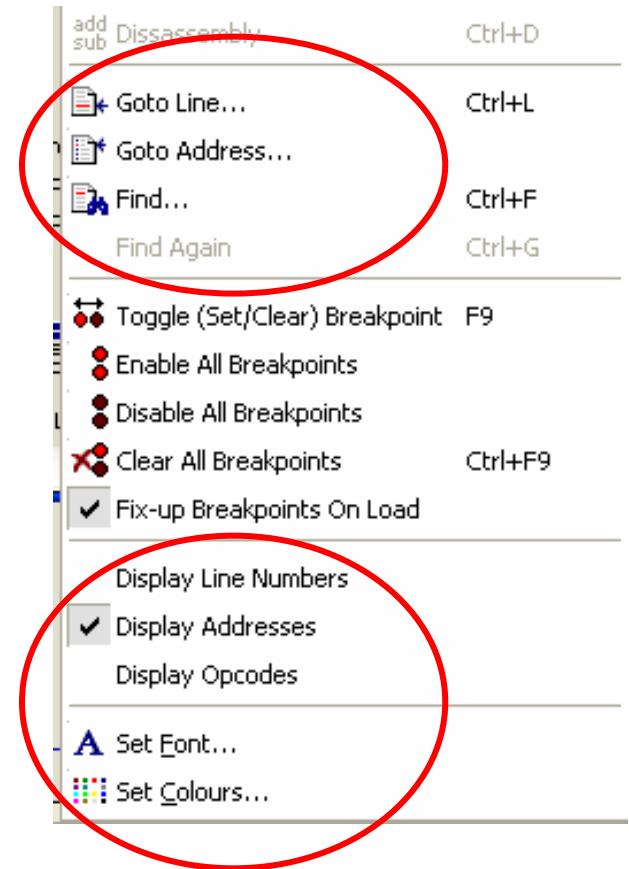
**En el cuadro de selección se pueden seleccionar otras opciones de edición y visualización como:**

**Ir a una línea determinada de programa**

**Ir a una dirección**

**Cambiar el tipo de letra**

**Cambiar la asignación de colores para direcciones instrucciones, direcciones etc.**



# LAS OPCIONES DE DEBUG

## PIC CPU Register –U1

*Pic CPU Register U1:* muestra el contenido de los registros del micro en estudio.

El contenido de los registros se va actualizando según se ejecutan las instrucciones

The image shows two windows from a PIC debugger. The left window, titled "PIC CPU Source Code - U1", displays the assembly code for a program named "DISP\_01.SDI". The code includes a table of 7-segment display patterns, an initialization section, and a loop that reads from a register and displays the value. The right window, titled "PIC CPU Registers - U1", shows the current state of the PIC's registers. The PC is \$001E, and the instruction is ANDLW 0x0F. The STATUS register is 24, FSR is 28, and OPTION is 255. The PORT A and B registers are 1 and 102, respectively. The TRIS A and B registers are 31 and 0. The W.MSGS register is 0. Below the registers, a circuit diagram shows a 7-segment display connected to PORT A, displaying the number 4.

**PIC CPU Source Code - U1**

```
DISP_01.SDI
-----;menos peso del reg. W en su equivalente a 7 segmentos. El cod
-----;segmentos retorna tambien en el reg. W
-----;*****
0005 TABLA:      addwf    PCL,F      ;Desplazamiento sobre
0006             retlw    b'00111111' ;Digito 0
0007             retlw    b'00000110' ;Digito 1
0008             retlw    b'01011011' ;Digito 2
0009             retlw    b'01001111' ;Digito 3
000A             retlw    b'01100110' ;Digito 4
000B             retlw    b'01101101' ;Digito 5
000C             retlw    b'01111101' ;Digito 6
000D             retlw    b'00000111' ;Digito 7
000E             retlw    b'01111111' ;Digito 8
000F             retlw    b'01101111' ;Digito 9
0010             retlw    b'01111100' ;Digito A
0011             retlw    b'00111100' ;Digito B
0012             retlw    b'00111101' ;Digito C
0013             retlw    b'01011110' ;Digito D
0014             retlw    b'01111001' ;Digito E
0015             retlw    b'01110001' ;Digito F
0016 INICIO      clrf     PORTB      ;Borra los lat
0017             bsf     STATUS,RP0    ;Selecciona banco 1
0018             clrf     TRISB        ;Puerta B se configura
0019             movlw    b'00011111'  ;Puerta A se configura
001A             movwf   TRISA         ;Selecciona banco 0
001B             bcf     STATUS,RP0
001C LOOP        clrwdt    ;Refrescar el WDT
001D             movf    PORTA,W       ;Lee el código de RAM
001E             andlw    b'00001111'  ;Convierte a 7 segmentos
001F             call    TABLA         ;Visualiza sobre el di
0020             movwf   PORTB
0021             goto    LOOP
-----
END                                     ;Fin d
```

**PIC CPU Registers - U1**

PC:	\$001E	W.MSGS:	0
INSTR.:	ANDLW 0x0F		
W:	4	\$04	%00000100
STATUS:	24	\$18	%00011000
FSR:	28	\$1C	%00011100
OPTION:	255	\$FF	%11111111
PORT A:	1	\$01	%00000001
PORT B:	102	\$66	%01100110
SP:	0		
RPX:	0		
Z:	0	DC:	0
C:	0		
PCLATH:	0	\$00	%00000000
INTCON:	0	\$00	%00000000
TRIS A:	31	\$1F	%00011111
TRIS B:	0	\$00	%00000000

# LAS OPCIONES DE DEBUG

## PIC CPU Data Memory –U1

***PIC CPU Data Memory –U1:*** muestra el contenido de la memoria de datos en estudio.

El contenido de las posiciones de memoria se va actualizando según se ejecutan las instrucciones

The screenshot displays two windows from a PIC development environment. The left window, titled "PIC CPU Source Code - U1", shows the assembly code for a program named "DISP\_01.SDI". The code includes a table of 7-segment patterns and a loop that updates the display. The right window, titled "PIC CPU Data Memory - U1", shows the memory contents, with the 7-segment display currently showing the digit '5'.

**PIC CPU Source Code - U1**

```
DISP_01.SDI
-----;menos peso del reg. W en su equivalente a 7 segmentos. El cod
-----;segmentos retorna tambien en el reg. W
-----;*****
0005 TABLA:      addwf   PCL,F           ;Desplazamiento sobre
0006              retlw   b'00111111'    ;Digito 0
0007              retlw   b'00000110'    ;Digito 1
0008              retlw   b'01011011'    ;Digito 2
0009              retlw   b'01001111'    ;Digito 3
000A              retlw   b'01100110'    ;Digito 4
000B              retlw   b'01101101'    ;Digito 5
000C              retlw   b'01111101'    ;Digito 6
000D              retlw   b'00000111'    ;Digito 7
000E              retlw   b'01111111'    ;Digito 8
000F              retlw   b'01100111'    ;Digito 9
0010              retlw   b'01111011'    ;Digito A
0011              retlw   b'01111100'    ;Digito B
0012              retlw   b'00111001'    ;Digito C
0013              retlw   b'01011110'    ;Digito D
0014              retlw   b'01111001'    ;Digito E
0015              retlw   b'01110001'    ;Digito F
0016 INICIO      clrf     PORTB          ;Borra los lat
0017              bsf     STATUS,RP0      ;Selecciona banco 1
0018              clrf   TRISE           ;Puerta B se configura
0019              movlw   b'00011111'
001A              movwf  TRISA           ;Puerta A se configura
001B              bcf     STATUS,RP0      ;Selecciona banco 0
001C LOOP        clrwdt  WDT            ;Refrescar el WDT
001D              movf   PORTA,W         ;Lee el c%digito de RA0-I
001E              andlw  b'00001111'    ;Convierte a 7 segmentos
001F              call   TABLA          ;Visualiza sobre el di
0020              movwf  PORTB
0021              goto   LOOP
-----
END                                     ;Fin d
```

**PIC CPU Data Memory - U1**

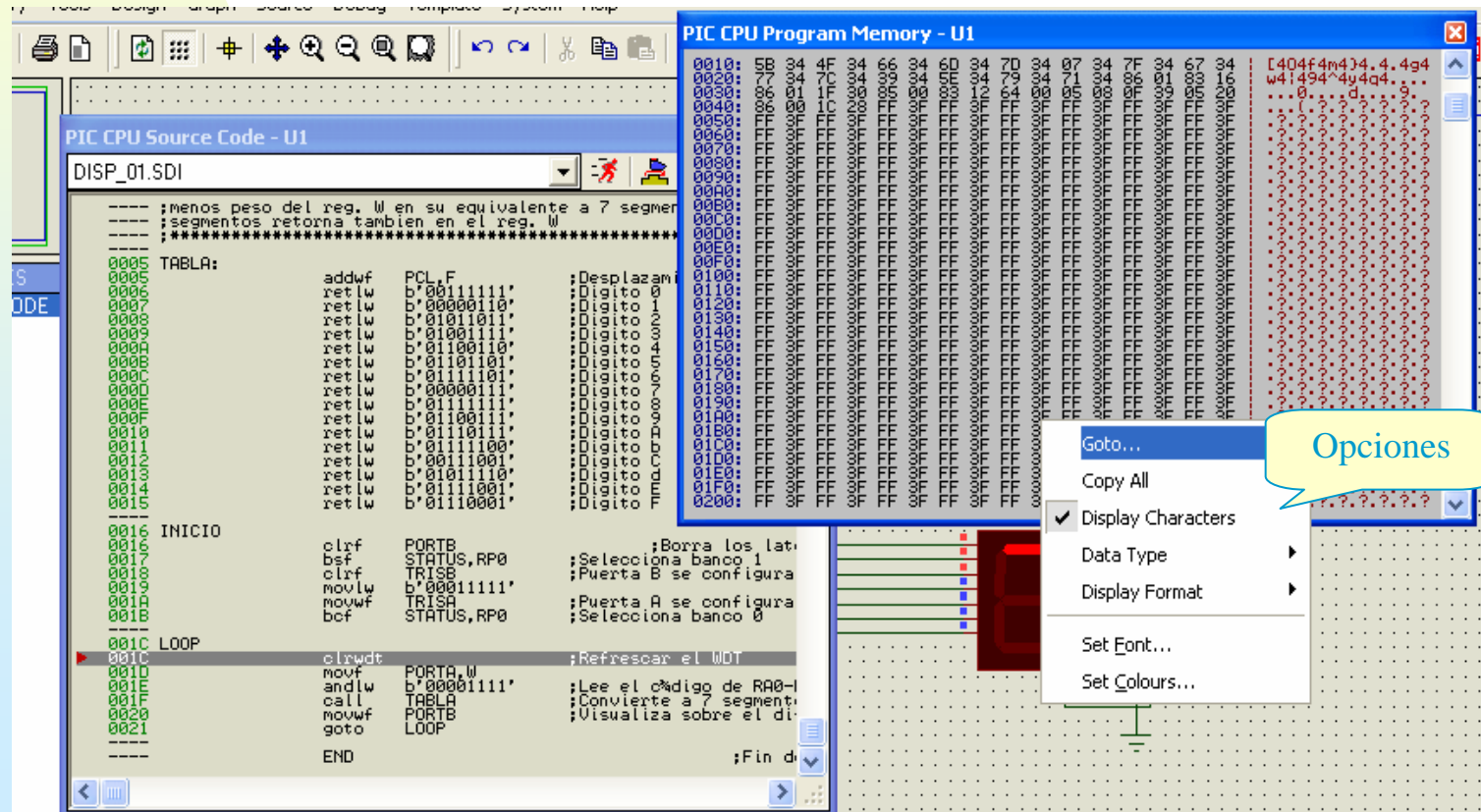
Address	Value
0000	00 BD 00 18 1C 01 6D 00
0008	BF 31 00 00 00 00 00 00
0010	00 00 00 00 00 00 00 00
0018	00 00 00 00 00 00 00 00
0020	00 00 00 00 00 00 00 00
0028	00 00 00 00 00 00 00 00
0030	00 00 00 00 00 00 00 00
0038	00 00 00 00 00 00 00 00
0040	00 00 00 00 00 00 00 00
0048	00 00 00 00 00 00 00 00
0050	00 00 00 00 00 00 00 00
0058	00 00 00 00 00 00 00 00
0060	00 00 00 00 00 00 00 00
0068	00 00 00 00 00 00 00 00
0070	00 00 00 00 00 00 00 00
0078	00 00 00 00 00 00 00 00
0080	FF 00 19 BB 1F 00 00
0088	00 00 01 00 00 00 00 00
0090	00 00 00 00 00 00 00 00
0098	00 00 00 00 00 00 00 00
00A0	00 00 00 00 00 00 00 00
00A8	00 00 00 00 00 00 00 00
00B0	00 00 00 00 00 00 00 00
00B8	00 00 00 00 00 00 00 00
00C0	00 00 00 00 00 00 00 00
00C8	00 00 00 00 00 00 00 00
00D0	00 00 00 00 00 00 00 00
00D8	00 00 00 00 00 00 00 00
00E0	00 00 00 00 00 00 00 00
00E8	00 00 00 00 00 00 00 00
00F0	00 00 00 00 00 00 00 00
00F8	00 00 00 00 00 00 00 00

# LAS OPCIONES DE DEBUG

## PIC CPU Program Memory –U1

*PIC CPU Program Memory U1:* muestra el contenido de la memoria de programa del  $\mu$  en estudio.

Existe diferentes opciones de visualización



# LAS OPCIONES DE DEBUG

## PIC CPU Stack –U1

**PIC CPU Stack:** nos indica en todo el valor de este registro

The image shows a screenshot of a PIC debugger interface. The main window, titled "PIC CPU Source Code - U1", displays the source code for a file named "DISP\_01.SDI". The code is written in assembly and includes comments in Spanish. A blue arrow points to the instruction at address 000E, which is "retlw b'01111111' ; Digito 8". A yellow callout bubble labeled "Dirección de retorno" points to this instruction. To the right, a smaller window titled "PIC CPU Sta..." shows the stack contents. A blue arrow points to the value 0x0020 at address 1, which is the return address. A yellow callout bubble labeled "Dirección de retorno" points to this value. Below the source code window, a circuit diagram is visible, showing a PIC microcontroller connected to a 7-segment display and two resistors (R10 and R11).

```
DISP_01.SDI
-----; menos peso del reg. W en su equivalente a 7 segmentos. El cod
-----; segmentos retorna tambien en el reg. W
-----; *****
0005 TABLA:
0005      addwf    PCL,F           ;Desplazamiento sobre
0006      retlw    b'00111111'    ;Digito 0
0007      retlw    b'00000110'    ;Digito 1
0008      retlw    b'01011011'    ;Digito 2
0009      retlw    b'01001111'    ;Digito 3
000A      retlw    b'01100110'    ;Digito 4
000B      retlw    b'01101101'    ;Digito 5
000C      retlw    b'01111101'    ;Digito 6
000D      retlw    b'00000111'    ;Digito 7
000E      retlw    b'01111111'    ;Digito 8
000F      retlw    b'01100111'    ;Digito 9
0010      retlw    b'01111011'    ;Digito 10
0011      retlw    b'01111100'    ;Digito 11
0012      retlw    b'00111001'    ;Digito 12
0013      retlw    b'01011110'    ;Digito 13
0014      retlw    b'01111100'    ;Digito 14
0015      retlw    b'01111001'    ;Digito 15
0016 INICIO
0016      clrf     PORTB           ;Borra los lat
0017      bsf      STATUS,RP0      ;Selecciona banco 1
0018      clrf     TRISB           ;Puerta B se configura
0019      movlw    b'00011111'
001A      movwf   TRISA           ;Puerta A se configura
001B      bcf      STATUS,RP0      ;Selecciona banco 0
001C LOOP
001C      clrw     PORTB           ;Refrescar el WDT
001D      movf    b'00001111'
001E      andlw   b'00001111'
001F      call    TABLA           ;Lee el c%digito de RA0-1
0020      movwf   PORTB           ;Convierte a 7 segmentos
0021      goto    LOOP            ;Visualiza sobre el di
-----
END                               ;Fin de
```

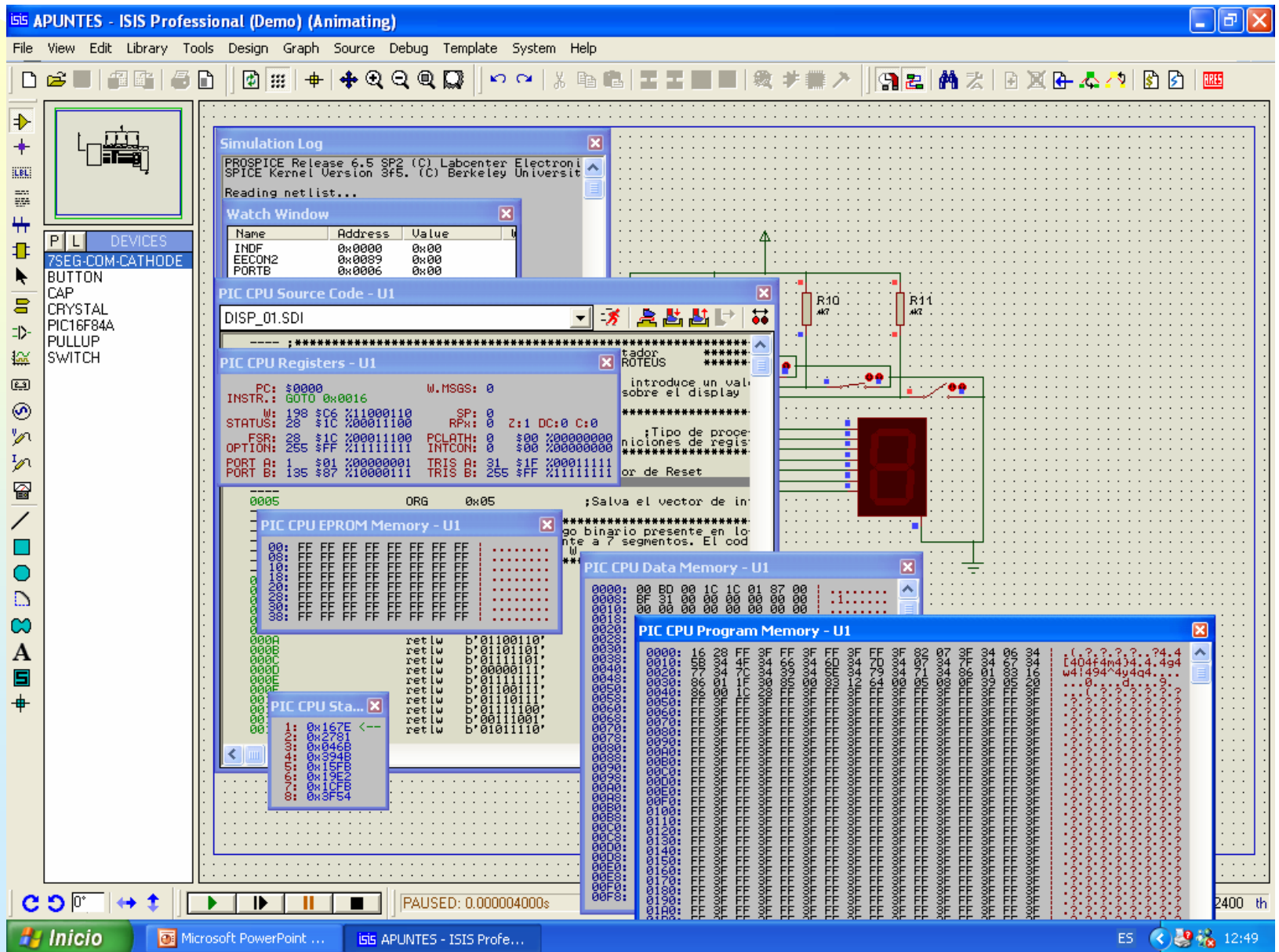
PIC CPU Sta...  
1: 0x0020  
2: 0x2781  
3: 0x046B  
4: 0x394B  
5: 0x15FB  
6: 0x19F3  
7: 0x1CF5  
8: 0x3F54

# LAS OPCIONES DE DEBUG

**Evidentemente en cada instante podemos visualizar las ventanas que se deseen de entre las estudiadas. Ahora bien es una buena costumbre no tener activadas aquellas ventanas que no sean necesarias.**

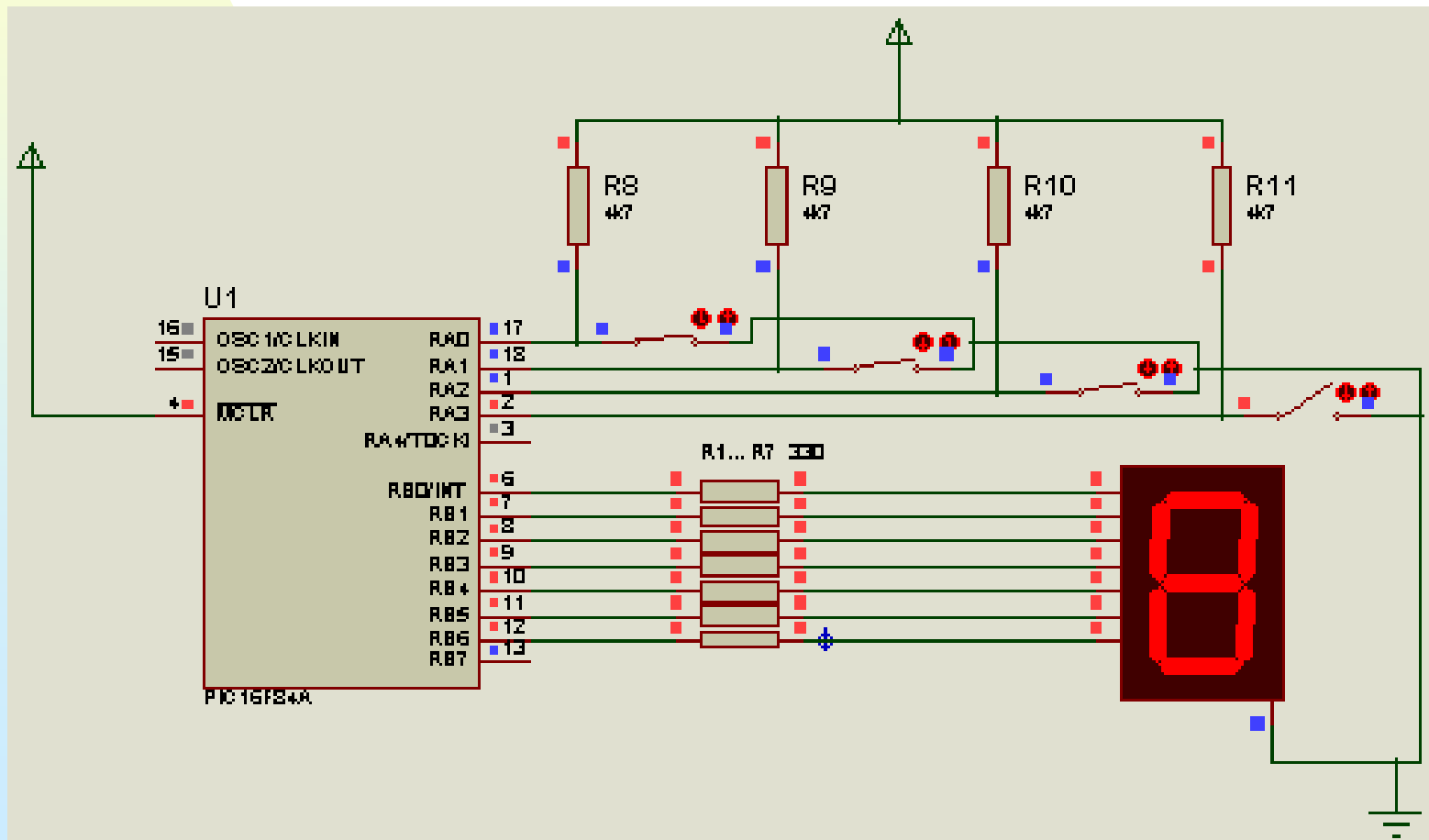


# LAS OPCIONES DE DEBUG



# LA SIMULACIÓN

Una vez que hemos depurado totalmente el programa, simulemos su funcionamiento, para ello pulsamos la tecla *PLAY*, el circuito que obtendríamos es:



# LA SIMULACIÓN

Sobre él explicamos diferentes símbolos que aparecen:

  : cuadrados que indican el rojo polaridad positiva (“1”) y azul polaridad negativa (“0”).



:switch con dos flechas, (una de apertura y otra de cierre), permiten que dicho elemento se abra y cierre, simulando un interruptor real.

Si nos encontramos en simulación continua y actuamos sobre los interruptores obtendremos:

