

# Práctica 1

## Programación de Microcontroladores PIC

- ❑ Entorno de trabajo
  - MPLAB
- ❑ Programación en ensamblador
  - Buenas prácticas de programación
  - Juego de instrucciones
  - Directivas al compilador
- ❑ Ejemplos
- ❑ Tareas a realizar

# Introducción

## ❑ Objetivos

- Saber manejar el entorno de desarrollo MPLAB.
- Ser capaz de desarrollar programas en ensamblador que se ejecuten en el microcontrolador.
- Conocer el juego de instrucciones de los microcontroladores PIC.
- Desarrollar buenas prácticas de programación en ensamblador:
  - Documentar adecuadamente los programas.
  - Realizar código sencillo, fácil de entender y modificar.



## Entorno de trabajo

- ❑ Se trata de desarrollar código ASM o C que podrá integrarse posteriormente en un microcontrolador de la familia PIC
  - Nosotros desarrollaremos código para el **PIC 16F877A**.
  - Usaremos para ello MPLAB v8.92 (2013)
- ❑ **IDE** (Integrated Development Environment)
  - Entorno software que nos permite ensamblar, compilar, *linkar*, exportar y depurar los programas.
- ❑ **ICE** (In Circuit Emulator)
  - Hardware que permite testear el desarrollo de programas en hardware no real (emulado).
  - Para testear ICEs necesitamos un IDE.
- ❑ **Programmer**
  - Es un sistema hardware (por ejemplo PICkit 2, PM3, **UNI-DS3**) que nos permite cargar los ficheros .hex a la memoria de programa de los procesadores de los PIC.
  - Está dirigido por el IDE.

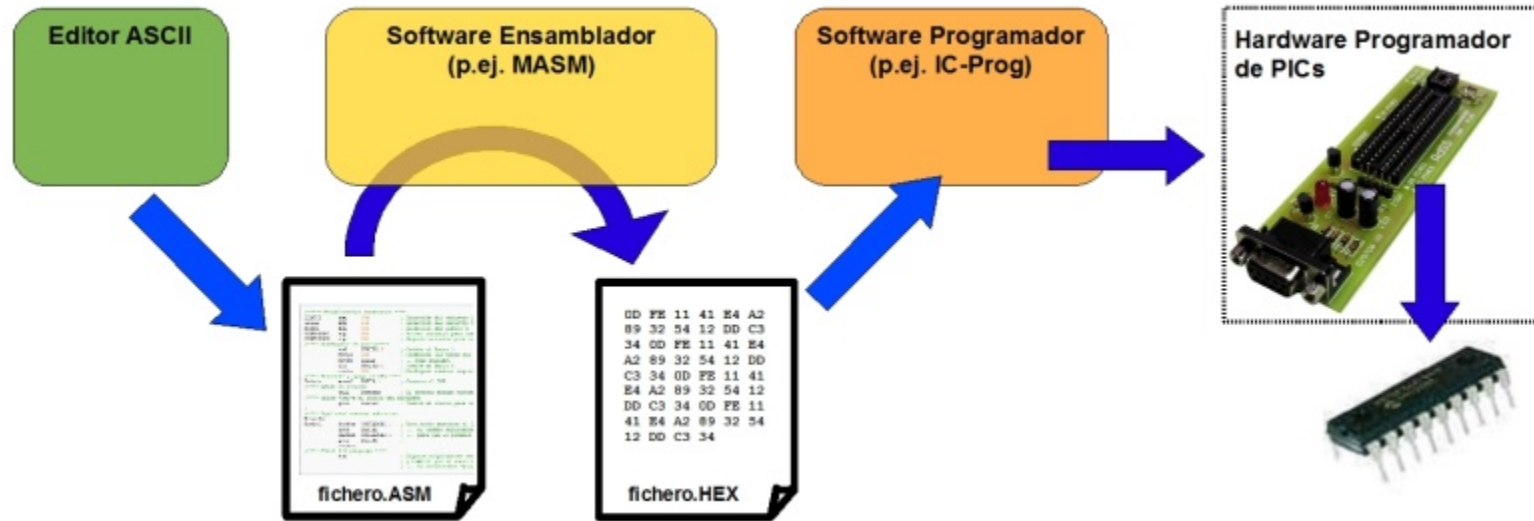


Descargar aquí:

[http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB\\_IDE\\_8\\_92.zip](http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB_IDE_8_92.zip)

## Entorno de trabajo

□ Así pues, la **metodología** que vamos a seguir es la siguiente:



□ Utilizaremos:

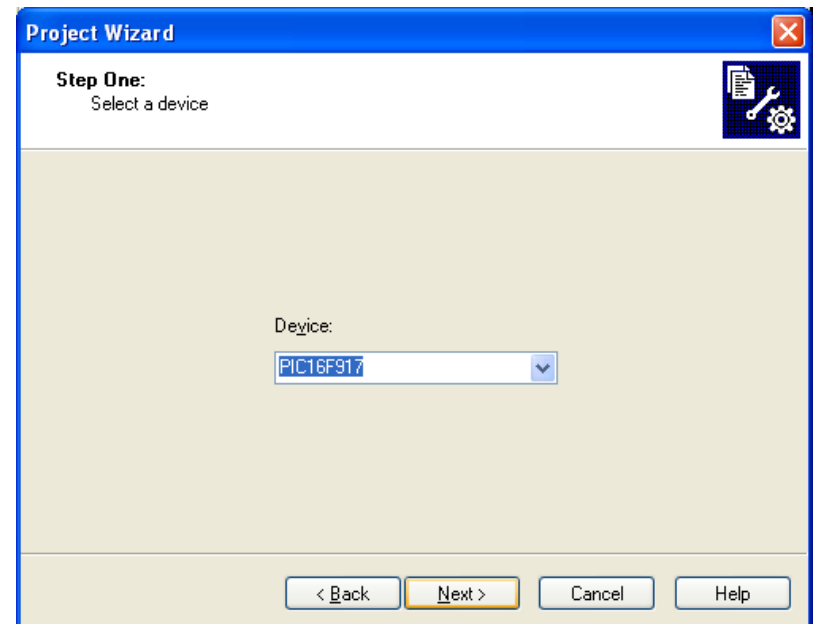
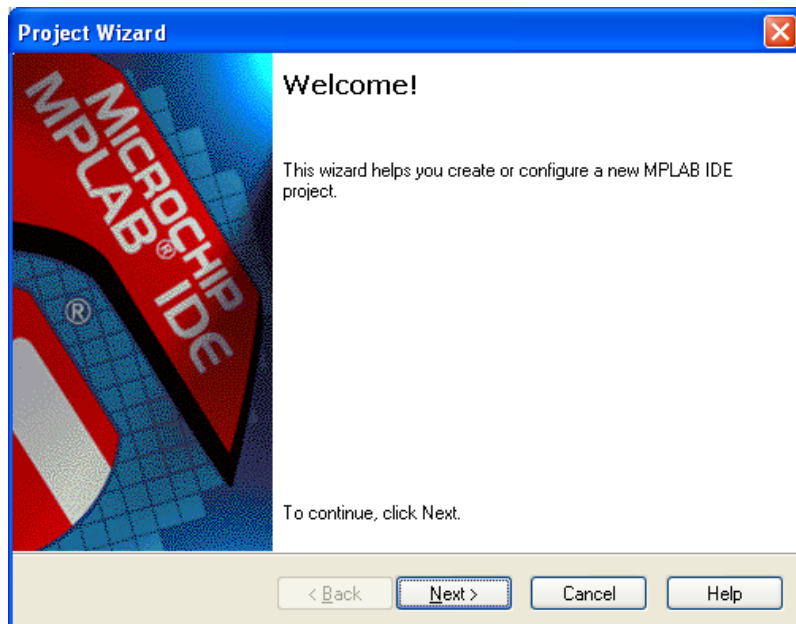
- **MPLAB**: para editar, compilar y ensamblar los programas en ensamblador.
- Las placas **UNI-DS3** que incorporan el PIC 16F877A y que conectamos por USB con nuestro PC.
- **PICFlash**: para transferir los programas ya testeados en el IDE a la memoria de programa del microcontrolador.

# MPLAB

## ❑ Debemos crear un **proyecto**:

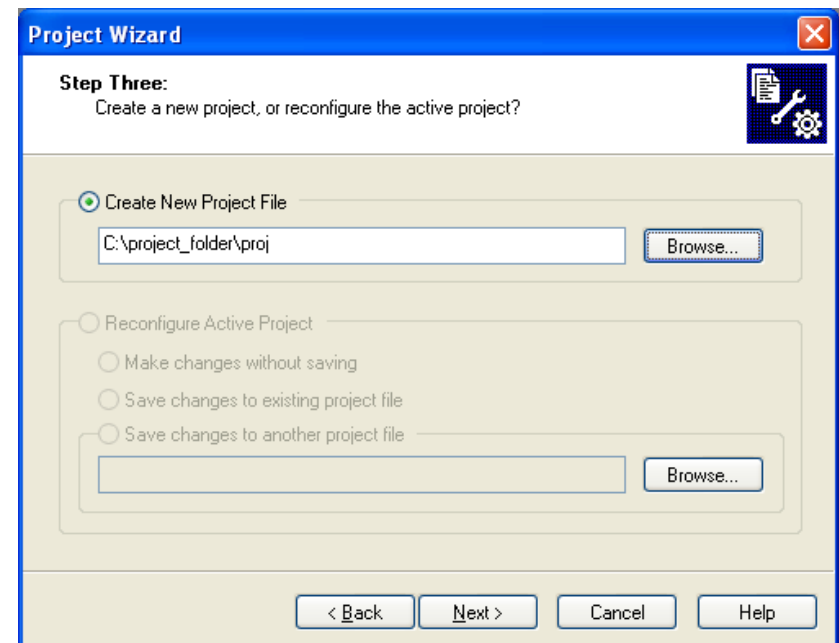
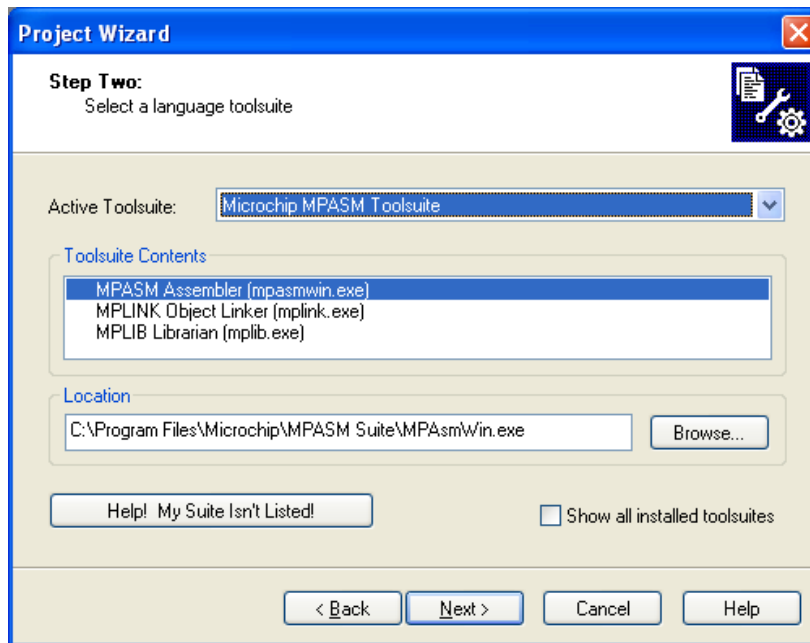
- Especificar el PIC para el que queremos programar
- Crear y editar los ficheros
- Compilar y linkar nuestro proyecto
- Programar el dispositivo

## ❑ Para crear el proyecto: Project→Project Wizard...



# MPLAB

- ☐ Paso 2: Seleccionar la herramienta Microchip MPASM
- ☐ Paso 3: Crear el nuevo fichero del proyecto



# MPLAB

## ❑ Paso 4: Añadir ficheros al proyecto

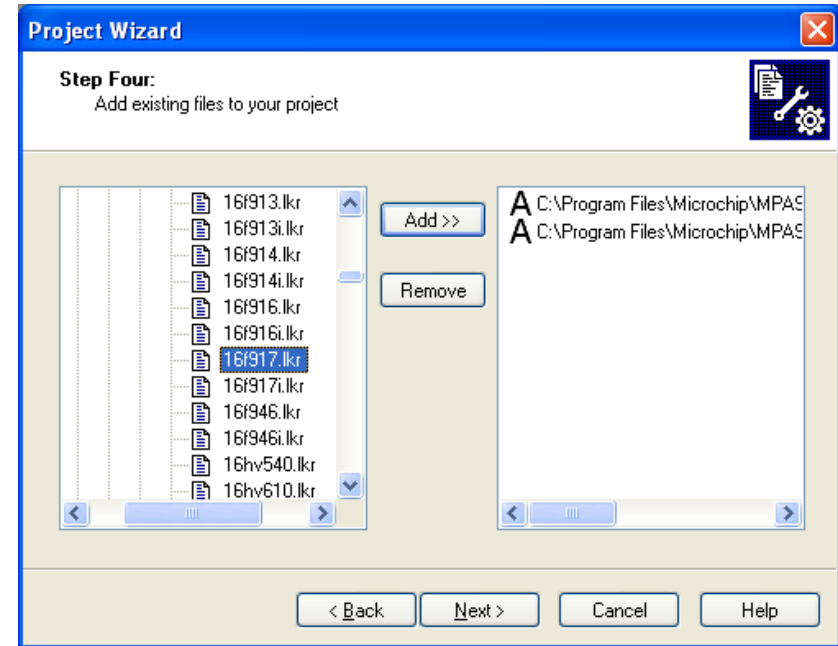
### ❑ Ficheros ASM

- Punto de inicio para el código ensamblador.

- <device name>tmpo.asm

### ❑ Linker script

- <device name>\_g.lkr



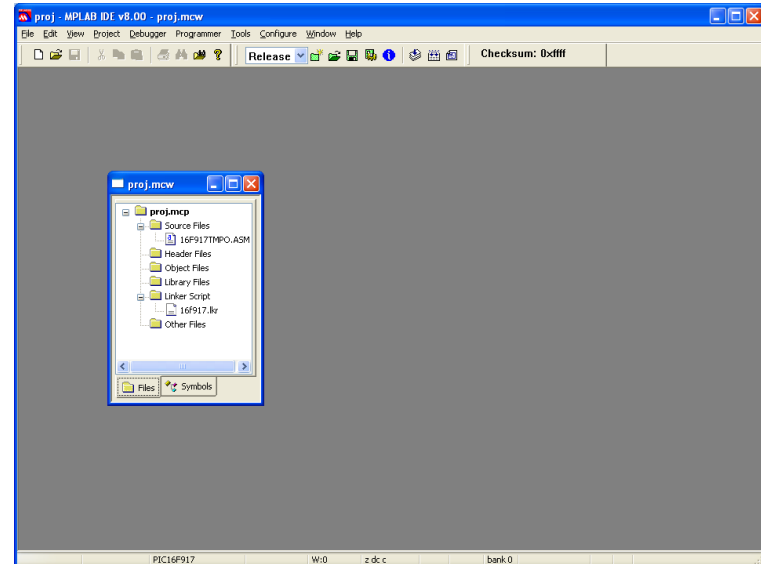
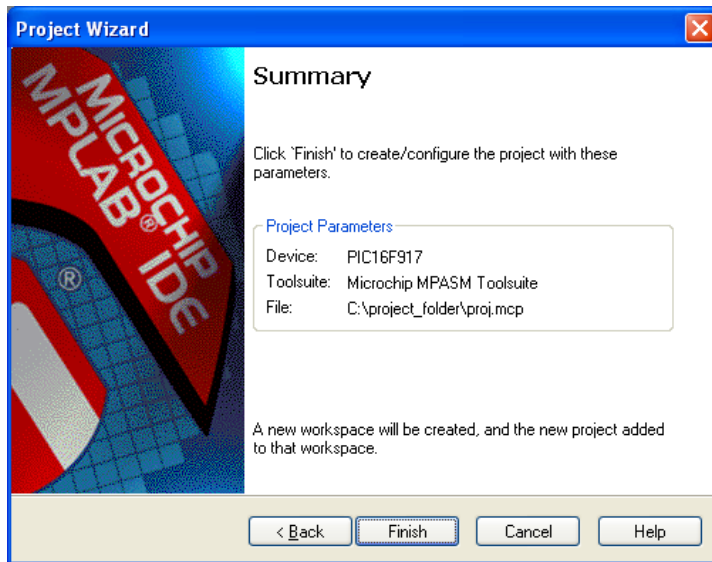
Ejemplo: Device PIC16F917

C:\Program Files\Microchip\MPASM Suite\Template\Object\16f917tmpo.asm

C:\Program Files\Microchip\MPASM Suite\16f917\_g.lkr

# MPLAB

- ❑ Terminar la creación del proyecto y regresar al entorno del IDE
- ❑ Ahora podemos ver los detalles del proyecto
  - View→Project

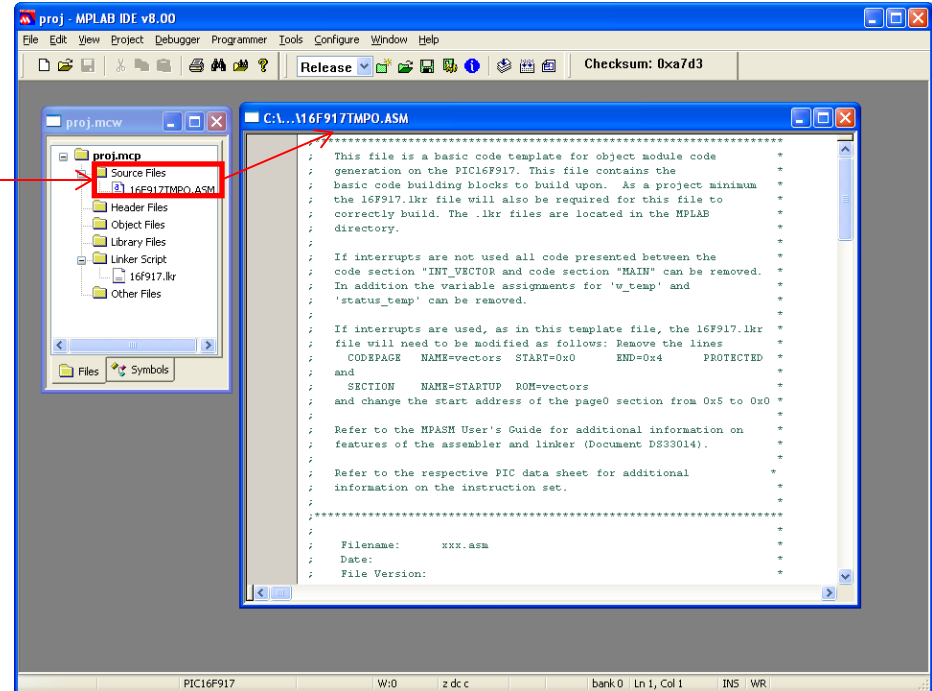




# MPLAB

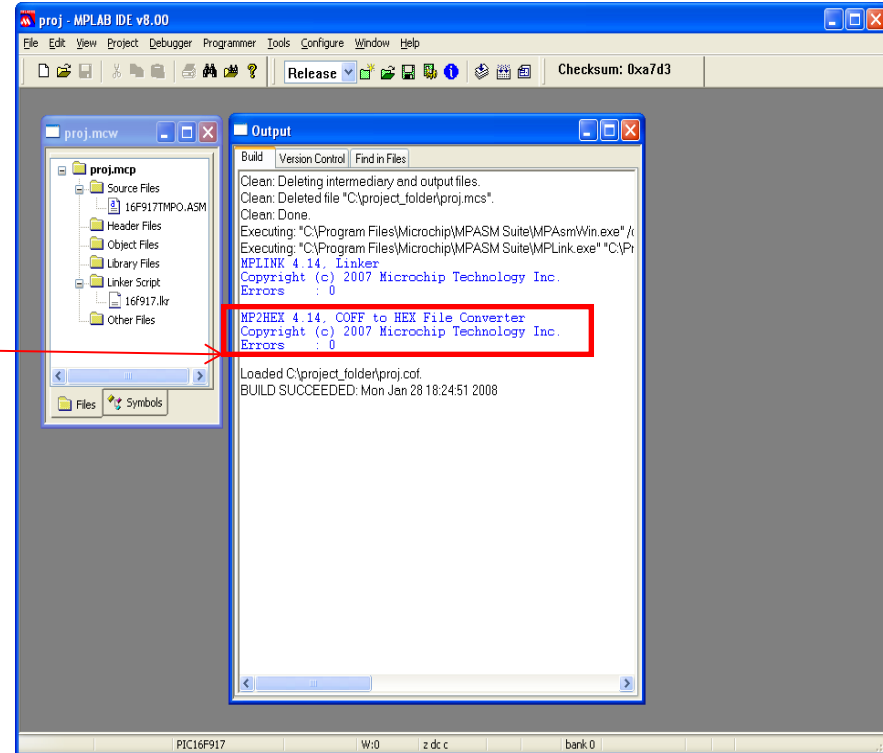
## ❑ Podemos modificar los ficheros .asm

- En la ventana del proyecto, en la entrada 'Source Files'
- Abrir y editar los ficheros para implementar nuevas funcionalidades
- Podemos crear y añadir nuevos ficheros



# MPLAB

- ❑ Compilamos el proyecto
  - Project→Build All (o Ctrl+F10)
- ❑ La ventana que se abre indica si ha habido algún error
- ❑ Cuando se compila con éxito el proyecto se genera un fichero HEX
  - Se usa para programar el microcontrolador



# MPLAB

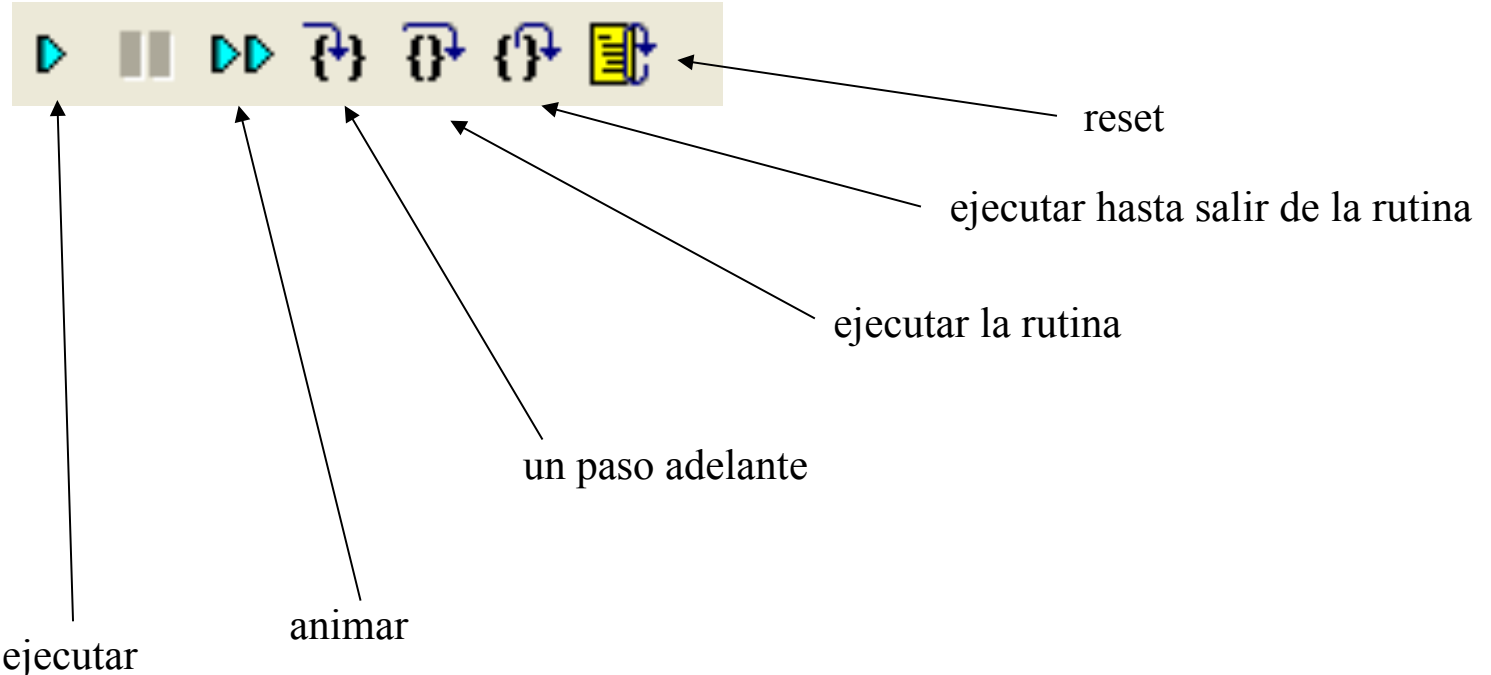
## ❑ Corregir errores

- Si sucede un error el compilador nos proporciona un mensaje indicándonos dónde se ha producido para que podamos corregirlo.
  - Si hacemos doble pulsación de ratón en el mensaje de error el sistema nos lleva a la línea del programa que ha causado el error para que podamos corregirlo.
- ❑ Error[113]  
Y:\TEACHING\ASSEMBLER\DB.ASM  
117 : Symbol not previously defined (TRISIO9)
  - ❑ Message[302]  
Y:\TEACHING\ASSEMBLER\DB.ASM  
119 : Register in operand not in bank 0.  
Ensure that bank bits are correct.
  - ❑ Warning[207]  
Y:\TEACHING\ASSEMBLER\DB.ASM  
225 : Found label after column 1.  
(inittim)
  - ❑ Halting build on first failure as requested.
  - ❑ BUILD FAILED: Tue Sep 21 14:40:30 2004

# MPLAB

❑ El entorno MPLAB nos permite también **simular** la ejecución del código que hemos programado:

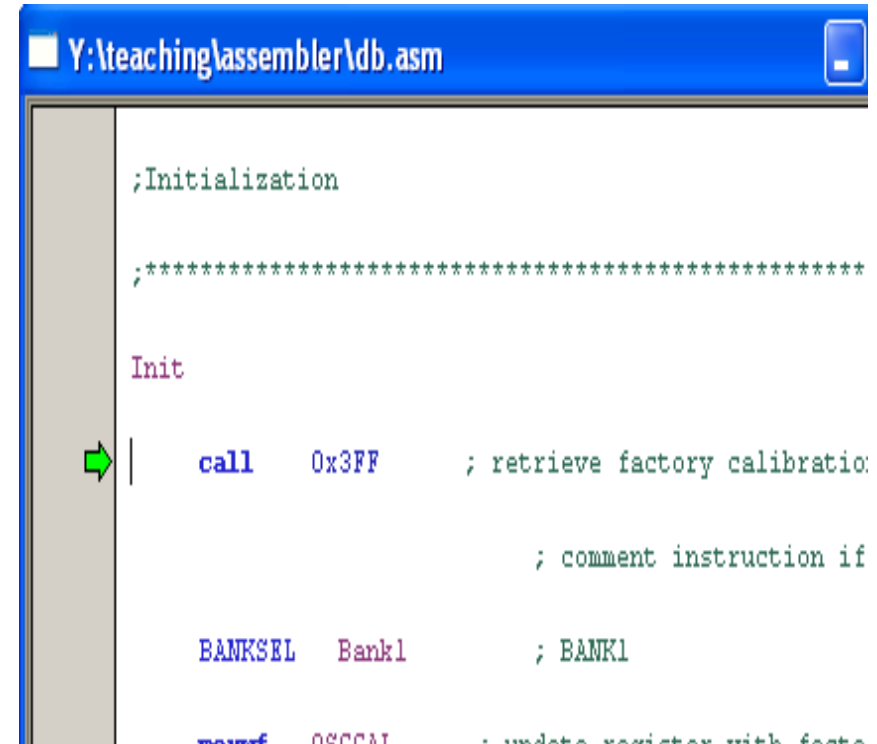
- Si vamos al menú de depuración y seleccionamos la herramienta de simulación podemos simular la ejecución del programa usando un emulador.
- Esto nos permite comprobar rápidamente que el programa es correcto
- La barra de simulación:



# MPLAB

## ❑ Ejecución en **modo emulación**:

- La línea que se está ejecutando en cada momento se muestra marcada por una flecha en la ventana de código.



```
Y:\teaching\assembler\ldb.asm

;Initialization

;*****

Init

    call    0x3FF      ; retrieve factory calibration

                        ; comment instruction if

    BANKSEL Bank1      ; BANK1

    movwf   0x0001      ; update register with facts
```

## Buenas prácticas para programar

- ❑ Para facilitar la **legibilidad** de los programas así como la **depuración** cuando surjan los errores, debemos comentar qué hacen las instrucciones de nuestros programas:

- Poniendo ; (punto y coma) de la instrucción, el compilador ignora el resto de la línea y eso nos permite escribir ahí la explicación de qué queremos hacer con esa instrucción.
- Poniendo una explicación delante de un conjunto de instrucciones (bucle, subrutina, etc) explicando qué se pretende hacer con ese código.

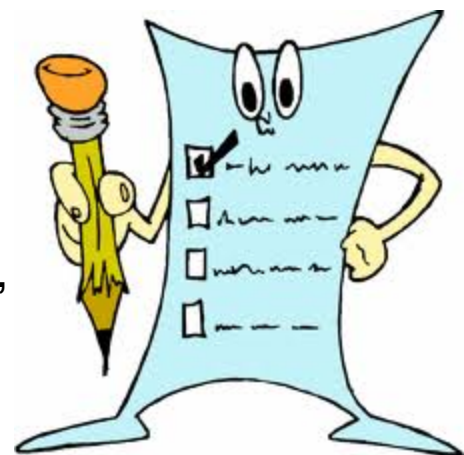


- ❑ Debemos asignar **nombres** a las constantes que nos permitan identificar qué queremos hacer con ellas (por ejemplo: CONTADOR, SUMA, ENCENDIDO, etc).
- ❑ **Diseña el programa** sobre papel antes de iniciar la programación:
  - Diagramas de flujo o algoritmos.

## Buenas prácticas para programar

- ❑ El ensamblador exige una cierta **tabulación** mínima de sus distintos elementos.

- De este modo la definición de variables podrá escribirse en la 1ª columna de cualquier línea.
- Las directivas e instrucciones deberán ir en la 2ª columna, como mínimo.

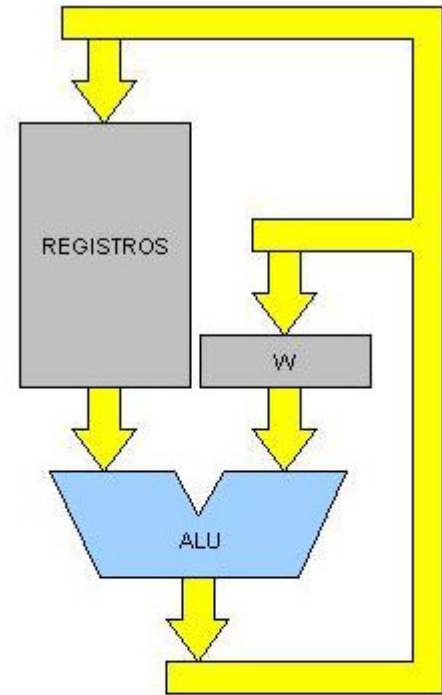


- ❑ Las tabulaciones características son las empleadas por nosotros, ya que, aunque no son imprescindibles, clarifican la lectura del programa.

Etiquetas	Operación	Operandos	Comentarios
INICIO			
	movlw	0x07	;Carga primer sumando en W
	addlw	0x08	;Suma W con segundo sumando
	movwf	RESULTADO	;Almacena el resultado
	END		;Fin del programa fuente

# Juego de instrucciones

- ❑ Los microcontroladores PIC tienen un **registro de trabajo** que se denomina W (Working Register).
- ❑ Todas las operaciones se realizan sobre el acumulador.
  - La salida del acumulador esta conectada a una de las entradas de la ALU, y por lo tanto éste es siempre uno de los dos operandos de cualquier instrucción.
  - Por convención, las instrucciones de simple operando (borrar, incrementar, decrementar, complementar), actúan sobre el acumulador.
  - La salida de la ALU va solamente a la entrada del acumulador, por lo tanto el resultado de cualquier operación siempre quedara en el acumulador.
  - Para operar sobre un dato de memoria, después de realizar la operación tendremos que mover siempre el acumulador a la memoria con una instrucción adicional.





# Arquitectura de los PIC de gama media

## ❑ Registro STATUS:

- Indica el estado de las operaciones aritméticas de la ALU
- Indica el estado el estado del RESET
- Controla la selección de bancos en la memoria de datos

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

**Bit7 IRP:** bit de selección de banco de registros usado en direccionamiento indirecto.

1 = Bank 2, 3 (100h - 1FFh)

0 = Bank 0, 1 (00h - FFh)

Para circuitos con Banco 0 y Banco 1 este bit esta reservado y siempre esta cero

**Bit 6:5 RP1:RP0:** bit de selección de banco de registros usado en direccionamiento directo.

11 = Banco 3 (180h - 1FFh)

10 = Banco 2 (100h - 17Fh)

01 = Banco 1 (80h - FFh)

00 = Banco 0 (00h - 7Fh)

Cada banco es de 128 bytes.

**Bit 4 TO':** bit Time-out.

1 = Después del encendido, de la instrucciones CLRWDT o SLEEP.

0 = Time-out del WDT.

# Arquitectura de los PIC de gama media

## ❑ Registro STATUS (continuación)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

**Bit 3 PD':** bit power-down

1 = Después del encendido o por la instrucción CLRWDT.

0 = Por la ejecución de la instrucción SLEEP.

**Bit 2 Z:** bit cero

1 = El resultado de una operación aritmética o lógica es cero.

0 = El resultado de una operación aritmética o lógica no es cero.

**Bit 1, DC:** bit Digit carry/borrow' (instrucciones ADDWF, ADDLW, SUBLW, SUBWF)

1 = El cuarto bit de menor peso del resultado produce acarreo.

0 = El cuarto bit de menor peso del resultado no produce acarreo.

**Bit 0, C:** bit Carry/borrow' (instrucciones ADDWF, ADDLW, SUBLW, SUBWF)

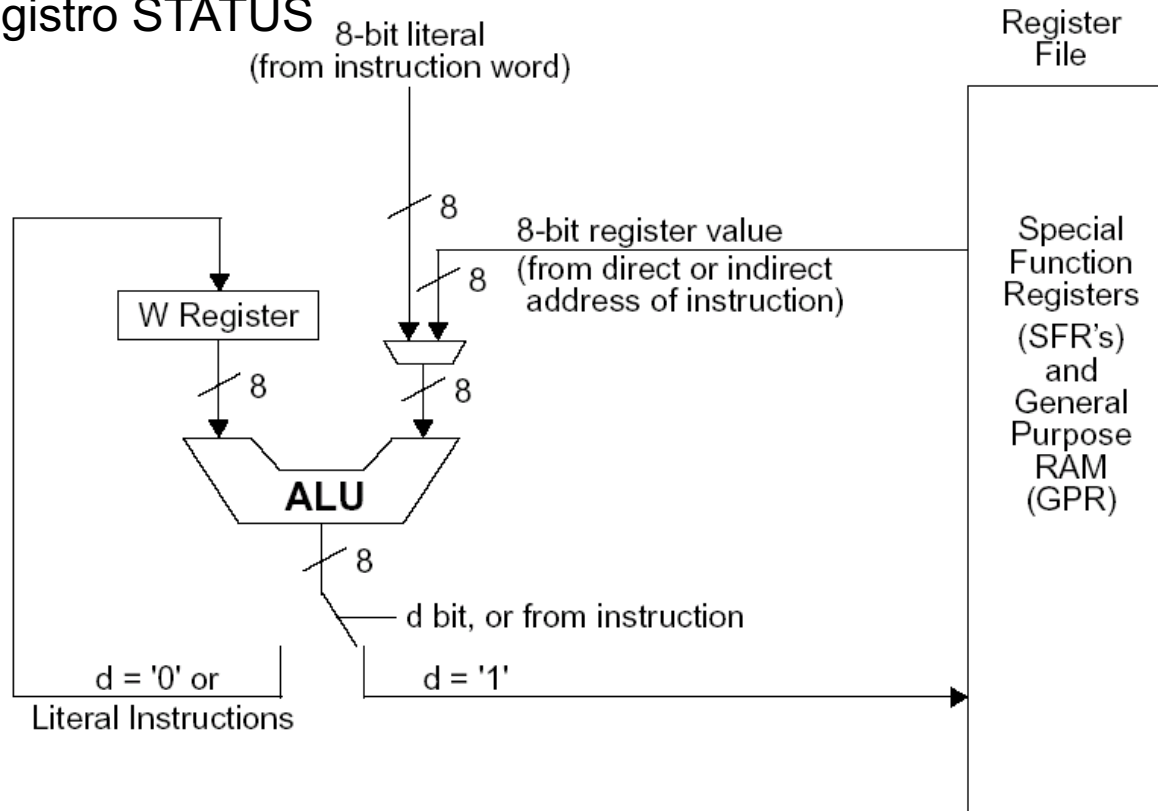
1 = El resultado produce acarreo.

0 = El resultado no produce acarreo.

# Arquitectura de los PIC de gama media

## Unidad Aritmético lógica:

- Longitud de palabra: 8 bits.
- Operaciones de suma, resta , desplazamiento y lógicas.
- Operaciones aritméticas en complemento a dos.
- Los acarrees (C), acarreo decimal (DC) y resultado cero (Z), se reflejan en el registro STATUS



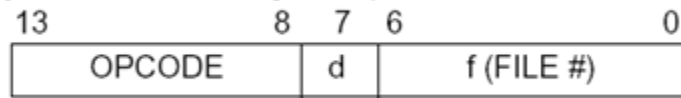
.....  
 BTFSC STATUS, Z  
 goto Destino  
 return

# Juego de instrucciones

## □ Existen tres tipos de instrucciones:

- Instrucciones de operación de bytes en registros

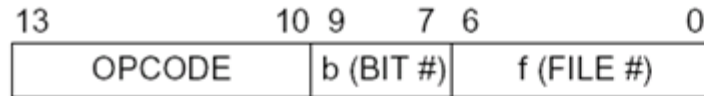
**Byte-oriented** file register operations



d = 0 for destination W  
d = 1 for destination f  
f = 7-bit file register address

- Instrucciones de manipulación de bits de registros

**Bit-oriented** file register operations

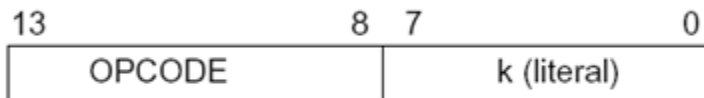


b = 3-bit bit address  
f = 7-bit file register address

- Instrucciones de control y operación con literales

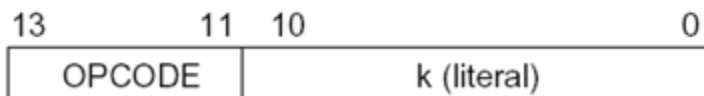
**Literal and control operations**

General



k = 8-bit immediate value

CALL and GOTO instructions only



k = 11-bit immediate value

# Juego de instrucciones

## ❑ El juego de instrucciones completo de los PIC 16F87XX

### ○ Instrucciones de operación de bytes en registros

Mnemonic, Operands	Description	Cycles	14-Bit Instruction Word				Status Bits Affected	
			MSb		LSb			
BYTE-ORIENTED FILE REGISTER OPERATIONS								
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z
CLRF	f	Clear f	1	00	0001	1fff	ffff	Z
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff	
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff	
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z
MOVWF	f	Move W to f	1	00	0000	1fff	ffff	
NOP	-	No Operation	1	00	0000	0xx0	0000	
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff	
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z

## Juego de instrucciones

- Instrucciones de manipulación de bits de registros

BIT-ORIENTED FILE REGISTER OPERATIONS								
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff	
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff	
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff-	ffff	
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff	

- Instrucciones de control y operación con literales

LITERAL AND CONTROL OPERATIONS								
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk	
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}, \overline{PD}$
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk	
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk	
RETFIE	-	Return from interrupt	2	00	0000	0000	1001	
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk	
RETURN	-	Return from Subroutine	2	00	0000	0000	1000	
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}, \overline{PD}$
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z

# Juego de instrucciones (detallado)

<p><b>ADDLW Suma un literal</b></p> <p>Sintaxis: [label] ADDLW k  <b>Operandos:</b> <math>0 \leq k \leq 255</math>  <b>Operación:</b> <math>(W) + (k) \Rightarrow (W)</math>  <b>Flags afectados:</b> C, DC, Z  <b>Código OP:</b> 11 111x kkkk kkkk</p> <p><b>Descripción:</b> Suma el contenido del registro W y k, guardando el resultado en W.</p> <p><b>Ejemplo:</b> ADDLW 0xC2</p> <p>Antes: W = 0x17  Después: W = 0xD9</p>	<p><b>ADDWF W + F</b></p> <p>Sintaxis: [label] ADDWF f,d  <b>Operandos:</b> <math>d \in [0,1], 0 \leq f \leq 127</math>  <b>Operación:</b> <math>(W) + (f) \Rightarrow (dest)</math>  <b>Flags afectados:</b> C, DC, Z  <b>Código OP:</b> 00 0111 dfff ffff</p> <p><b>Descripción:</b> Suma el contenido del registro W y el registro f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p> <p><b>Ejemplo:</b> ADDWF REG,0</p> <p>Antes: W = 0x17., REG = 0xC2  Después: W = 0xD9, REG = 0xC2</p>	<p><b>ANDLW W AND literal</b></p> <p>Sintaxis: [label] ANDLW k  <b>Operandos:</b> <math>0 \leq k \leq 255</math>  <b>Operación:</b> <math>(W) \text{ AND } (k) \Rightarrow (W)</math>  <b>Flags afectados:</b> Z  <b>Código OP:</b> 11 1001 kkkk kkkk</p> <p><b>Descripción:</b> Realiza la operación lógica AND entre el contenido del registro W y k, guardando el resultado en W.</p> <p><b>Ejemplo:</b> ADDLW 0xC2</p> <p>Antes: W = 0x17  Después: W = 0xD9</p>
<p><b>ANDWF W AND F</b></p> <p>Sintaxis: [label] ANDWF f,d  <b>Operandos:</b> <math>d \in [0,1], 0 \leq f \leq 127</math>  <b>Operación:</b> <math>(W) \text{ AND } (f) \Rightarrow (dest)</math>  <b>Flags afectados:</b> Z  <b>Código OP:</b> 00 0101 dfff ffff</p> <p><b>Descripción:</b> Realiza la operación lógica AND entre los registros W y f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p> <p><b>Ejemplo:</b> : ANDWF REG,0</p> <p>Antes: W = 0x17., REG = 0xC2  Después: W = 0x17, REG = 0x02</p>	<p><b>BCF Borra un bit</b></p> <p>Sintaxis: [label] BCF f,b  <b>Operandos:</b> <math>0 \leq f \leq 127, 0 \leq b \leq 7</math>  <b>Operación:</b> <math>0 \Rightarrow (f \leftarrow b)</math>  <b>Flags afectados:</b> Ninguno  <b>Código OP:</b> 01 00bb bfff ffff</p> <p><b>Descripción:</b> Borra el bit b del registro f</p> <p><b>Ejemplo:</b> : BCF REG,7</p> <p>Antes: REG = 0xC7  Después: REG = 0x47</p>	<p><b>BSF Activa un bit</b></p> <p>Sintaxis: [label] BSF f,b  <b>Operandos:</b> <math>0 \leq f \leq 127, 0 \leq b \leq 7</math>  <b>Operación:</b> <math>1 \Rightarrow (f \leftarrow b)</math>  <b>Flags afectados:</b> Ninguno  <b>Código OP:</b> 01 01bb bfff ffff</p> <p><b>Descripción:</b> Activa el bit b del registro f</p> <p><b>Ejemplo:</b> : BSF REG,7</p> <p>Antes: REG = 0x0A  Después: REG = 0x8A</p>



# Juego de instrucciones (detallado)

<p><b>BTFSC</b> Test de bit y salto</p> <p>Sintaxis: [label] BTFSC f,d  <b>Operandos:</b> <math>d \in [0,1]</math>, <math>0 \leq f \leq 127</math>  <b>Operación:</b> Salto si <math>(f &lt; b) = 0</math>  <b>Flags afectados:</b> Ninguno  <b>Código OP:</b> 01 10bb bfff ffff</p> <p><b>Descripción:</b> Si el bit b del registro f es 0, se salta una instrucción y se continúa con la ejecución. En caso de salto, ocupará dos ciclos de reloj.</p> <p><b>Ejemplo:</b> BTFSC REG,6  GOTO NO_ES_0  SI_ES_0 Instrucción  NO_ES_0 Instrucción</p>	<p><b>BTFSS</b> Test de bit y salto</p> <p>Sintaxis: [label] BTFSS f,d  <b>Operandos:</b> <math>d \in [0,1]</math>, <math>0 \leq f \leq 127</math>  <b>Operación:</b> Salto si <math>(f &lt; b) = 1</math>  <b>Flags afectados:</b> Ninguno  <b>Código OP:</b> 01 11bb bfff ffff</p> <p><b>Descripción:</b> Si el bit b del registro f es 1, se salta una instrucción y se continúa con la ejecución. En caso de salto, ocupará dos ciclos de reloj.</p> <p><b>Ejemplo:</b> BTFSS REG,6  GOTO NO_ES_0  SI_ES_0 Instrucción  NO_ES_0 Instrucción</p>	<p><b>CALL</b> Salto a subrutina</p> <p>Sintaxis: [label] CALL k  <b>Operandos:</b> <math>0 \leq k \leq 2047</math>  <b>Operación:</b> <math>PC \Rightarrow Pila</math>; <math>k \Rightarrow PC</math>  <b>Flags afectados:</b> Ninguno  <b>Código OP:</b> 10 0kkk kkkk kkkk</p> <p><b>Descripción:</b> Salto a una subrutina. La parte baja de k se carga en PCL, y la alta en PCLATCH. Ocupa 2 ciclos de reloj.</p> <p><b>Ejemplo:</b> ORIGEN CALL DESTINO    Antes: PC = ORIGEN  Después: PC = DESTINO</p>
<p><b>CLRF</b> Borra un registro</p> <p>Sintaxis: [label] CLRF f  <b>Operandos:</b> <math>0 \leq f \leq 127</math>  <b>Operación:</b> : <math>0x00 \Rightarrow (f)</math>, <math>1 \Rightarrow Z</math>  <b>Flags afectados:</b> Z  <b>Código OP:</b> 00 0001 1fff ffff</p> <p><b>Descripción:</b> El registro f se carga con 0x00. El flag Z se activa.</p> <p><b>Ejemplo:</b> : CLRF REG    Antes: REG = 0x5A  Después: REG = 0x00, Z = 1</p>	<p><b>CLRW</b> Borra el registro W</p> <p>Sintaxis: [label] CLRW  <b>Operandos:</b> Ninguno  <b>Operación:</b> : <math>0x00 \Rightarrow W</math>, <math>1 \Rightarrow Z</math>  <b>Flags afectados:</b> Z  <b>Código OP:</b> 00 0001 0xxx xxxx</p> <p><b>Descripción:</b> El registro de trabajo W se carga con 0x00. El flag Z se activa.</p> <p><b>Ejemplo:</b> : CLRW    Antes: W = 0x5A  Después: W = 0x00, Z = 1</p>	<p><b>CLRWD</b> Borra el WDT</p> <p>Sintaxis: [label] CLRWD  <b>Operandos:</b> Ninguno  <b>Operación:</b> <math>0x00 \Rightarrow WDT</math>, <math>1 \Rightarrow /TO</math>  <math>1 \Rightarrow /PD</math>  <b>Flags afectados:</b> /TO, /PD  <b>Código OP:</b> 00 0000 0110 0100  <b>Descripción:</b> Esta instrucción borra tanto el WDT como su preescaler. Los bits /TO y /PD del registro de estado se ponen a 1.</p> <p><b>Ejemplo:</b> : CLRWD  Después: Contador WDT = 0,  Preescalas WDT = 0,  /TO = 1, /PD = 1</p>



# Juego de instrucciones (detallado)

<p><b>COMF</b> Complemento de f</p> <p>Sintaxis: [label] COMF f,d  <b>Operandos:</b> <math>d \in [0,1]</math>, <math>0 \leq f \leq 127</math>  <b>Operación:</b> <math>(/f), 1 \Rightarrow (\text{dest})</math>  <b>Flags afectados:</b> Z  <b>Código OP:</b> 00 1001 dfff ffff</p> <p><b>Descripción:</b> El registro f es complementado. El flag Z se activa si el resultado es 0. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p> <p><b>Ejemplo:</b> : COMF REG,0</p> <p>Antes: REG = 0x13          Después: REG = 0x13, W = 0XEC</p>	<p><b>DECF</b> Decremento de f</p> <p>Sintaxis: [label] DECF f,d  <b>Operandos:</b> <math>d \in [0,1]</math>, <math>0 \leq f \leq 127</math>  <b>Operación:</b> <math>(f) - 1 \Rightarrow (\text{dest})</math>  <b>Flags afectados:</b> Z  <b>Código OP:</b> 00 0011 dfff ffff</p> <p><b>Descripción:</b> Decrementa en 1 el contenido de f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p> <p><b>Ejemplo:</b> : DECF CONT,1</p> <p>Antes: CONT = 0x01, Z = 0          Después: CONT = 0x00, Z = 1</p>	<p><b>DECFSZ</b> Decremento y salto</p> <p>Sintaxis: [label] DECFSZ f,d  <b>Operandos:</b> <math>d \in [0,1]</math>, <math>0 \leq f \leq 127</math>  <b>Operación:</b> <math>(f) - 1 \Rightarrow d</math>; Salto si R=0  <b>Flags afectados:</b> Ninguno  <b>Código OP:</b> 00 1011 dfff ffff</p> <p><b>Descripción:</b> Decrementa el contenido del registro f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Si la resta es 0 salta la siguiente instrucción, en cuyo caso costaría 2 ciclos.</p> <p><b>Ejemplo:</b> DECFSZ REG,0          GOTO NO_ES_0          SI_ES_0 Instrucción          NO_ES_0 Salta instrucción anterior</p>
<p><b>GOTO</b> Salto incondicional</p> <p>Sintaxis: [label] GOTO k  <b>Operandos:</b> <math>0 \leq k \leq 2047</math>  <b>Operación:</b> <math>k \Rightarrow \text{PC} \langle 8:0 \rangle</math>  <b>Flags afectados:</b> Ninguno  <b>Código OP:</b> 10 1kkk kkkk kkkk</p> <p><b>Descripción:</b> Se trata de un salto incondicional. La parte baja de k se carga en PCL, y la alta en PCLATCH. Ocupa 2 ciclos de reloj.</p> <p><b>Ejemplo:</b> ORIGEN GOTO DESTINO</p> <p>Antes: PC = ORIGEN          Después: PC = DESTINO</p>	<p><b>INCF</b> Decremento de f</p> <p>Sintaxis: [label] INCF f,d  <b>Operandos:</b> <math>d \in [0,1]</math>, <math>0 \leq f \leq 127</math>  <b>Operación:</b> <math>(f) + 1 \Rightarrow (\text{dest})</math>  <b>Flags afectados:</b> Z  <b>Código OP:</b> 00 1010 dfff ffff</p> <p><b>Descripción:</b> Incrementa en 1 el contenido de f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p> <p><b>Ejemplo:</b> : INCF CONT,1</p> <p>Antes: CONT = 0xFF, Z = 0          Después: CONT = 0x00, Z = 1</p>	<p><b>INCFSZ</b> Incremento y salto</p> <p>Sintaxis: [label] INCFSZ f,d  <b>Operandos:</b> <math>d \in [0,1]</math>, <math>0 \leq f \leq 127</math>  <b>Operación:</b> <math>(f) + 1 \Rightarrow d</math>; Salto si R=0  <b>Flags afectados:</b> Ninguno  <b>Código OP:</b> 00 1111 dfff ffff</p> <p><b>Descripción:</b> Incrementa el contenido del registro f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Si la resta es 0 salta la siguiente instrucción, en cuyo caso costaría 2 ciclos.</p> <p><b>Ejemplo:</b> INCFSZ REG,0          GOTO NO_ES_0          SI_ES_0 Instrucción          NO_ES_0 Salta instrucción anterior</p>

# Juego de instrucciones (detallado)

<p><b>IORLW</b>    W OR literal</p> <p>Sintaxis: [label] IORLW k  <b>Operandos:</b> <math>0 \leq k \leq 255</math>  <b>Operación:</b> <math>(W) \text{ OR } (k) \Rightarrow (W)</math>  <b>Flags afectados:</b> Z  <b>Código OP:</b> 11 1000 kkkk kkkk</p> <p><b>Descripción:</b> Se realiza la operación lógica OR entre el contenido del registro W y k, guardando el resultado en W.</p> <p><b>Ejemplo:</b>        IORLW 0x35</p> <p>Antes: W = 0x9A  Después: W = 0xBF</p>	<p><b>IORWF</b>        W AND F</p> <p>Sintaxis: [label] IORWF f,d  <b>Operandos:</b> <math>d \in [0,1], 0 \leq f \leq 127</math>  <b>Operación:</b> <math>(W) \text{ AND } (f) \Rightarrow (\text{dest})</math>  <b>Flags afectados:</b> Z  <b>Código OP:</b> 00 0100 dfff ffff</p> <p><b>Descripción:</b> Realiza la operación lógica AND entre los registros W y f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p> <p><b>Ejemplo:</b>        IORWF REG,0</p> <p>Antes: W = 0x91, REG = 0x13  Después: W = 0x93, REG = 0x13</p>	<p><b>MOVLW</b> Cargar literal en W</p> <p>Sintaxis: [label] MOVLW f  <b>Operandos:</b> <math>0 \leq f \leq 255</math>  <b>Operación:</b> <math>(k) \Rightarrow (W)</math>  <b>Flags afectados:</b> Ninguno  <b>Código OP:</b> 11 00xx kkkk kkkk</p> <p><b>Descripción:</b> El literal k pasa al registro W.</p> <p><b>Ejemplo:</b>        MOVLW 0x5A</p> <p>Después: REG = 0x4F, W = 0x5A</p>
<p><b>MOVF</b>        Mover a f</p> <p>Sintaxis: [label] MOVF f,d  <b>Operandos:</b> <math>d \in [0,1], 0 \leq f \leq 127</math>  <b>Operación:</b> <math>(f) \Rightarrow (\text{dest})</math>  <b>Flags afectados:</b> Z  <b>Código OP:</b> 00 1000 dfff ffff</p> <p><b>Descripción:</b> El contenido del registro f se mueve al destino d. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Permite verificar el registro, puesto que afecta a Z.</p> <p><b>Ejemplo:</b>        MOVF REG,0</p> <p>Después: W = REG</p>	<p><b>MOVWF</b>        Mover a f</p> <p>Sintaxis: [label] MOVWF f  <b>Operandos:</b> <math>0 \leq f \leq 127</math>  <b>Operación:</b> <math>W \Rightarrow (f)</math>  <b>Flags afectados:</b> Ninguno  <b>Código OP:</b> 00 0000 1fff ffff</p> <p><b>Descripción:</b> El contenido del registro W pasa al registro f.</p> <p><b>Ejemplo:</b>        MOVWF REG,0</p> <p>Antes: REG = 0xFF, W = 0x4F  Después: REG = 0x4F, W = 0x4F</p>	<p><b>NOP</b>        No operar</p> <p>Sintaxis: [label] NOP  <b>Operandos:</b> Ninguno  <b>Operación:</b> No operar  <b>Flags afectados:</b> Ninguno  <b>Código OP:</b> 00 0000 0xx0 0000</p> <p><b>Descripción:</b> No realiza operación alguna. En realidad consume un ciclo de instrucción sin hacer nada.</p> <p><b>Ejemplo:</b>        CLRWD</p> <p>Después: Contador WDT = 0,  Preescalas WDT = 0,  /TO = 1, /PD = 1</p>

# Juego de instrucciones (detallado)

<p><b>RETFIE</b> Retorno de interrup.</p> <p>Sintaxis: [label] RETFIE  Operandos: Ninguno  Operación: : <math>1 \Rightarrow \text{GIE}</math>; <math>\text{TOS} \Rightarrow \text{PC}</math>  Flags afectados: Ninguno  Código OP: 00 0000 0000 1001</p> <p>Descripción: El PC se carga con el contenido de la cima de la pila (TOS): dirección de retorno. Consume 2 ciclos. Las interrupciones vuelven a ser habilitadas.</p> <p>Ejemplo: : RETFIE</p> <p>Después: PC = dirección de retorno  GIE = 1</p>	<p><b>RETLW</b> Retorno, carga W</p> <p>Sintaxis: [label] RETLW k  Operandos: <math>0 \leq k \leq 255</math>  Operación: : <math>(k) \Rightarrow (W)</math>; <math>\text{TOS} \Rightarrow \text{PC}</math>  Flags afectados: Ninguno  Código OP: 11 01xx kkkk kkkk</p> <p>Descripción: El registro W se carga con la constante k. El PC se carga con el contenido de la cima de la pila (TOS): dirección de retorno. Consume 2 ciclos.</p> <p>Ejemplo: : RETLW 0x37</p> <p>Después: PC = dirección de retorno  W = 0x37</p>	<p><b>RETURN</b> Retorno de rutina</p> <p>Sintaxis: [label] RETURN  Operandos: Ninguno  Operación: : <math>\text{TOS} \Rightarrow \text{PC}</math>  Flags afectados: Ninguno  Código OP: 00 0000 0000 1000</p> <p>Descripción: El PC se carga con el contenido de la cima de la pila (TOS): dirección de retorno. Consume 2 ciclos.</p> <p>Ejemplo: : RETURN</p> <p>Después: PC = dirección de retorno</p>
<p><b>RLF</b> Rota f a la izquierda</p> <p>Sintaxis: [label] RLF f,d  Operandos: <math>d \in [0,1]</math>, <math>0 \leq f \leq 127</math>  Operación: Rotación a la izquierda  Flags afectados: C  Código OP: 00 1101 dfff ffff</p> <p>Descripción: El contenido de f se rota a la izquierda. El bit de menos peso de f pasa al carry (C), y el carry se coloca en el de mayor peso. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p> <p>Ejemplo: RRF REG,0</p> <p>Antes: REG = 1110 0110, C = 0  Después: REG = 1110 0110,  W = 1100 1100, C = 1</p>	<p><b>RRF</b> Rota f a la derecha</p> <p>Sintaxis: [label] RRF f,d  Operandos: <math>d \in [0,1]</math>, <math>0 \leq f \leq 127</math>  Operación: Rotación a la derecha  Flags afectados: C  Código OP: 00 1100 dfff ffff</p> <p>Descripción: El contenido de f se rota a la derecha. El bit de menos peso de f pasa al carry (C), y el carry se coloca en el de mayor peso. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p> <p>Ejemplo: RRF REG,0</p> <p>Antes: REG = 1110 0110, C = 1  Después: REG = 1110 0110,  W = 01110 0011, C = 0</p>	<p><b>SLEEP</b> Modo bajo consumo</p> <p>Sintaxis: [label] SLEEP  Operandos: Ninguno  Operación: <math>0x00 \Rightarrow \text{WDT}</math>, <math>1 \Rightarrow / \text{TO}</math>  <math>0 \Rightarrow \text{WDT Preescaler}</math>, <math>0 \Rightarrow / \text{PD}</math>  Flags afectados: / PD, / TO  Código OP: 00 0000 0110 0011</p> <p>Descripción: El bit de energía se pone a 0, y a 1 el de descanso. El WDT y su preescaler se borran. El micro para el oscilador, llendo al modo "durmiente".</p> <p>Ejemplo: : SLEEP</p> <p>Preescalas WDT = 0,  /TO = 1, /PD = 1</p>

# Juego de instrucciones (detallado)

<p><b>SUBLW Resta Literal - W</b></p> <p><b>Sintaxis:</b> [label] SUBLW k  <b>Operandos:</b> <math>0 \leq k \leq 255</math>  <b>Operación:</b> <math>(k) - (W) \Rightarrow (W)</math>  <b>Flags afectados:</b> Z, C, DC  <b>Código OP:</b> 11 110x kkkk kkkk  <b>Descripción:</b> Mediante el método del complemento a dos el contenido de W es restado al literal. El resultado se almacena en W.</p>	<p><b>SUBWF Resta f - W</b></p> <p><b>Sintaxis:</b> [label] SUBWF f,d  <b>Operandos:</b> <math>d \in [0,1], 0 \leq f \leq 127</math>  <b>Operación:</b> <math>(f) - (W) \Rightarrow (dest)</math>  <b>Flags afectados:</b> C, DC, Z  <b>Código OP:</b> 00 0010 dfff ffff  <b>Descripción:</b> Mediante el método del complemento a dos el contenido de W es restado al de f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p>	<p><b>SWAPF Intercambio de f</b></p> <p><b>Sintaxis:</b> [label] SWAPF f,d  <b>Operandos:</b> <math>d \in [0,1], 0 \leq f \leq 127</math>  <b>Operación:</b> <math>(f &lt; 3: 0) \Leftrightarrow (f &lt; 7: 4)</math>  <b>Flags afectados:</b> Ninguno  <b>Código OP:</b> 00 1110 dfff ffff</p> <p><b>Descripción:</b> Los 4 bits de más peso y los 4 de menos son intercambiados. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p>
<p><b>Ejemplos:</b> SUBLW 0x02</p> <p>Antes: W=1, C=?. Después: W=1, C=1          Antes: W=2, C=?. Después: W=0, C=1          Antes: W=3, C=?. Después: W=FF, C=0          (El resultado es negativo)</p>	<p><b>Ejemplos:</b> SUBWF REG,1          Antes: REG = 0x03, W = 0x02, C = ?          Después: REG=0x01, W = 0x4F, C=1          Antes: REG = 0x02, W = 0x02, C = ?          Después: REG=0x00, W = 0x02, C= 1          Antes: REG= 0x01, W= 0x02, C= ?          Después: REG=0xFF, W=0x02, C= 0          (Resultado negativo)</p>	<p><b>Ejemplo:</b> : SWAPF REG,0</p> <p>Antes: REG = 0xA5          Después: REG = 0xA5, W = 0x5A</p>
<p><b>XORLW W OR literal</b></p> <p><b>Sintaxis:</b> [label] XORLW k  <b>Operandos:</b> <math>0 \leq k \leq 255</math>  <b>Operación:</b> <math>(W) \text{ XOR } (k) \Rightarrow (W)</math>  <b>Flags afectados:</b> Z  <b>Código OP:</b> 11 1010 kkkk kkkk</p> <p><b>Descripción:</b> Se realiza la operación lógica XOR entre el contenido del registro W y k, guardando el resultado en W.</p>	<p><b>XORWF W AND F</b></p> <p><b>Sintaxis:</b> [label] XORWF f,d  <b>Operandos:</b> <math>d \in [0,1], 0 \leq f \leq 127</math>  <b>Operación:</b> <math>(W) \text{ XOR } (f) \Rightarrow (dest)</math>  <b>Flags afectados:</b> Z  <b>Código OP:</b> 00 0110 dfff ffff</p> <p><b>Descripción:</b> Realiza la operación lógica XOR entre los registros W y f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p>	
<p><b>Ejemplo:</b> : XORLW 0xAF</p> <p>Antes: W = 0xB5          Después: W = 0x1A</p>	<p><b>Ejemplo:</b> : XORWF REG,0</p> <p>Antes: W = 0xB5, REG = 0xAF          Después: W = 0xB5, REG = 0x1A</p>	

# Juego de instrucciones

- ❑ La gama baja sólo tiene 31 de estas instrucciones anteriores:
  - Carece de las instrucciones ADDLW, RETFIE, RETURN y SUBLW.
- ❑ En cambio, y puesto que los registros OPTION y TRIS no son accesibles, se añaden las dos siguientes:

<p><b>OPTION</b> Carga del reg. option</p> <p><b>Sintaxis:</b> [label] OPTION  <b>Operandos:</b> Ninguno  <b>Operación:</b> (W) ⇒ OPTION  <b>Flags afectados:</b> Ninguno  <b>Código OP:</b> 0000 0000 0010</p> <p><b>Descripción:</b> El contenido del registro W se carga en el registro OPTION, registro de sólo lectura en el que se configura el funcionamiento del preescaler y el TMR0.</p>
<p><b>Ejemplo:</b> :                   OPTION</p> <p>Antes: W= 0x06, OPTION = 0x37          Después: W= 0x06, OPTION = 0x06</p>

<p><b>TRIS</b> Carga del registro TRIS</p> <p><b>Sintaxis:</b> [label] TRIS f  <b>Operandos:</b> <math>5 \leq f \leq 7</math>  <b>Operación:</b> (W) ⇒ Registro TRIF&lt;f&gt;  <b>Flags afectados:</b> Ninguno  <b>Código OP:</b> 0000 0000 0fff</p> <p><b>Descripción:</b> El contenido del registro W se carga en el registro TRISA, TRISB o TRISC, según el valor de f. Estos registros de sólo lectura configuran las patillas de un puerto como de entrada o salida.</p>
<p><b>Ejemplo:</b> :                   TRIS PORTA</p> <p>Antes: W=0xA5, TRISA=0x56          Después: W=0xA5, TRISA=0xA5</p>

- ❑ El código OP de las instrucciones de la gama baja sólo ocupan 12 bits, no correspondiéndose, por tanto, con el de la gama media.



## Directivas al compilador

❑ Se trata de especificarle al compilador ciertas cosas que debe tener en cuenta a la hora de generar el código HEX

❑ Las principales directivas al compilador son:

- **LIST**: Define el procesador a utilizar durante todos los procesos (ensamblado, emulación, grabación).
  - Ejemplo: LIST P = 16F877A
- **ORG**: seguida de una posición de memoria, indica al ensamblador dónde debe situar ese código en la memoria de programa del microcontrolador.
  - Se coloca antes de la primera instrucción.
- **END**: es imprescindible e indica al ensamblador el final del programa.
- **REM**: indica que lo que le sigue es un comentario.
- **EQU**: define una constante <label> equ <expr>
- **SET**: define una variable <label> set <expr>



# Directivas al compilador

❑ La lista completa de directivas al compilador es la siguiente:

## CONTROL

Directive	Description	Syntax
CONSTANT	Declare Symbol Constant	constant <label> [= <expr>,...,<label> [= <expr>] ]
#DEFINE	Define a Text Substitution Label	#define <name> [[(<arg>,...,<arg>)] <value>]
END	End Program Block	end
EQU	Define an Assembly Constant	<label> equ <expr>
#INCLUDE	Include Additional Source File	include <<include_file>> include "<include_file>"
ORG	Set Program Origin	<label> org <expr>
PROCESSOR	Set Processor Type	processor <processor_type>
RADIX	Specify Default Radix	radix <default_radix>
SET	Define an Assembler Variable	<label> set <expr>
#UNDEFINE	Delete a Substitution Label	#undefine <label>
VARIABLE	Declare Symbol Variable	variable <label> [= <expr>,..., <label> [= <expr>] ]

## CONDITIONAL ASSEMBLY

Directive	Description	Syntax
ELSE	Begin Alternative Assembly Block to IF	else
ENDIF	End Conditional Assembly Block	endif
ENDW	End a While Loop	endw
IF	Begin Conditionally Assembled Code Block	if <expr>
IFDEF	Execute If Symbol is Defined	ifdef <label>
IFNDEF	Execute If Symbol is Not Defined	ifndef <label>
WHILE	Perform Loop While Condition is True	while <expr>

# Directivas al compilador

## DATA

Directive	Description	Syntax
<a href="#">_ _BADRAM</a>	Specify invalid RAM locations	<code>_ _badram &lt;expr&gt;</code>
<a href="#">CBLOCK</a>	Define a Block of Constants	<code>cblock [&lt;expr&gt;]</code>
<a href="#">_ _CONFIG</a>	Set configuration fuses	<code>_ _config &lt;expr&gt; OR _ _config &lt;addr&gt;, &lt;expr&gt;</code>
<a href="#">DA</a>	Store Strings in Program Memory	<code>[&lt;label&gt;] da &lt;expr&gt; [, &lt;expr2&gt;, ..., &lt;exprn&gt;]</code>
<a href="#">DATA</a>	Create Numeric and Text Data	<code>data &lt;expr&gt; [, &lt;expr&gt;, ..., &lt;expr&gt;] data "&lt;text_string&gt;" [, "&lt;text_string&gt;", ...]</code>
<a href="#">DB</a>	Declare Data of One Byte	<code>db &lt;expr&gt; [, &lt;expr&gt;, ..., &lt;expr&gt;]</code>
<a href="#">DE</a>	Declare EEPROM Data	<code>de &lt;expr&gt; [, &lt;expr&gt;, ..., &lt;expr&gt;]</code>
<a href="#">DT</a>	Define Table	<code>dt &lt;expr&gt; [, &lt;expr&gt;, ..., &lt;expr&gt;]</code>
<a href="#">DW</a>	Declare Data of One Word	<code>dw &lt;expr&gt; [, &lt;expr&gt;, ..., &lt;expr&gt;]</code>
<a href="#">ENDC</a>	End an Automatic Constant Block	<code>endc</code>
<a href="#">FILL</a>	Specify Memory Fill Value	<code>fill &lt;expr&gt;, &lt;count&gt;</code>
<a href="#">RES</a>	Reserve Memory	<code>res &lt;mem_units&gt;</code>
<a href="#">IDLOCS</a>	Set ID locations	<code>idlocs &lt;expr&gt;</code>
<a href="#">MAXRAM</a>	Specify maximum RAM address	<code>maxram &lt;expr&gt;</code>

## LISTING

Directive	Description	Syntax
<a href="#">ERROR</a>	Issue an Error Message	<code>error "&lt;text_string&gt;"</code>
<a href="#">ERRORLEVEL</a>	Set Message Level	<code>errorlevel 0 1 2 [&lt;+&gt; &lt;msg&gt;]</code>
<a href="#">LIST</a>	Listing Options	<code>list [&lt;option&gt; [, ..., &lt;option&gt;]]</code>
<a href="#">MESSG</a>	Create User Defined Message	<code>messg "&lt;message_text&gt;"</code>
<a href="#">NOLIST</a>	Turn off Listing Output	<code>nolist</code>
<a href="#">PAGE</a>	Insert Listing Page Eject	<code>page</code>
<a href="#">SPACE</a>	Insert Blank Listing Lines	<code>space [&lt;expr&gt;]</code>
<a href="#">SUBTITLE</a>	Specify Program Subtitle	<code>subtitl "&lt;sub_text&gt;"</code>
<a href="#">TITLE</a>	Specify Program Title	<code>title "&lt;title_text&gt;"</code>

## MACRO

Directive	Description	Syntax
<a href="#">ENDM</a>	End a Macro Definition	<code>endm</code>
<a href="#">EXITM</a>	Exit from a Macro	<code>exitm</code>
<a href="#">EXPAND</a>	Expand Macro Listing	<code>expand</code>
<a href="#">LOCAL</a>	Declare Local Macro Variable	<code>local &lt;label&gt; [, &lt;label&gt;]</code>
<a href="#">MACRO</a>	Declare Macro Definition	<code>&lt;label&gt; macro [&lt;arg&gt; ..., &lt;arg&gt;]</code>
<a href="#">NOEXPAND</a>	Turn off Macro Expansion	<code>noexpand</code>

## OBJECT FILE

Directive	Description	Syntax
<a href="#">BANKISEL</a>	Generate RAM bank selecting code for indirect addressing	<code>bankisel &lt;label&gt;</code>
<a href="#">BANKSEL</a>	Generate RAM bank selecting code	<code>banksel &lt;label&gt;</code>
<a href="#">CODE</a>	Begins executable code section	<code>[&lt;name&gt;] code [&lt;address&gt;]</code>
<a href="#">EXTERN</a>	Declares an external label	<code>extern &lt;label&gt; [, &lt;label&gt;]</code>
<a href="#">GLOBAL</a>	Exports a defined label	<code>extern &lt;label&gt; [, &lt;label&gt;]</code>
<a href="#">IDATA</a>	Begins initialized data section	<code>[&lt;name&gt;] idata [&lt;address&gt;]</code>
<a href="#">PAGESEL</a>	Generate ROM page selecting code	<code>pagesel &lt;label&gt;</code>
<a href="#">UDATA</a>	Begins uninitialized data section	<code>[&lt;name&gt;] udata [&lt;address&gt;]</code>
<a href="#">UDATA_ACS</a>	Begins access uninitialized data section	<code>[&lt;name&gt;] udata_acs [&lt;address&gt;]</code>
<a href="#">UDATA_OVR</a>	Begins overlapped uninitialized data section	<code>[&lt;name&gt;] udata_ovr [&lt;address&gt;]</code>
<a href="#">UDATA_SHR</a>	Begins shared uninitialized data section	<code>[&lt;name&gt;] udata_shr [&lt;address&gt;]</code>



# Directivas al compilador (Ejemplo)

```

;*****
;* MPASM. Ejemplo de utilización de
;* directivas
;*
;*****

processor 16f84a      ;Establece que procesador se utilizará
radix dec             ;Numeros representados en decimal
                      ;(por defecto esta en hexadecimal)
                      ;uso radix hex|dec|oct

#include <p16f84a.inc> ;Incluye el fichero de texto
                      ;p16f84a.inc

DTEMP equ 0x20        ;Asigna expresiones a las etiquetas
DFLAG equ 0x21        ;correspondientes
DFL0 equ DTEMP+5

LARGO set 4           ;Define variables en ensamblador
ANCHO set 6           ;equivalente a EQU

AREA set ANCHO * LARGO

cblock 0x30
    dato1,dato2      ;Asigna valores desde el indicado
    resultado:0,resul_h,resul_l ;a las etiquetas que siguen, se
    tabla:5          ;puede establecer un incremento en
                      ;la asignacion.
    aux              ;dato1=0x30, dato2=0x31,resul_h=0x32
    W_temp,STATUS_temp ;resul_l=0x33, tabla=0x34, aux=0x39
endc                 ;W_temp=0x3A, STATUS_temp=0x3B

cblock 0x20
    DTEMP, DFLAG
endc

PUSH macro           ;definición de macro
    movwf W_temp
    swaf STATUS,W
    movwf STATUS_temp
ende

POP macro
    swaf STATUS_temp,W
    movwf STATUS
    swaf W_temp,f
    swaf W_temp,W
endm

#define banco1 bsf STATUS,RP0 ;define una cadena de sustitución
#define banco0 bcf STATUS,RP0 ;de texto.

org 0x00             ;Indica la posición de de la memoria

```

```

Inicio
    goto Inicio
    org 0x04
    goto ServInt
    org 0x06

    banco0
    clrf 0x05

    banco1
    clrf 0x05

    movlw 65
    movlw d'65'
    movlw h'41'
    movlw a'A'
    movlw b'01000001'
    movlw o'101'
    call Sub1
    ; resto
    ; del
    ; programa
    ; .....

    org 0x100

ServInt
    PUSH
    ; rutina...
    ; ....
    POP
    Retfie

Sub1
    PUSH
    ; rutina...
    ; ....
    POP
    Return

end ;fin de programa fuente

```

```

;de programa donde se ubicara el código
;siguiente
;Vector de reset

;Vector de interrupción

;Inicio del programa

;Selecciona el banco cero
;pone a cero el registro de dirección
;0x5(PORTA)
;Selecciona el banco1
;pone a cero el registro de dirección
;0x5(TRISA)
;base decimal por defecto: radix dec
;En las siguientes instrucciones carga
;el registro W con el mismo valor en
;diferentes bases

```

## Ejemplo

- ❑ Sumar dos valores inmediatos (7 y 8) y el resultado dejarlo en la posición de memoria 0x10:

```

;*****
;Programa E001.asm                      Fecha: 3 Diciembre 2004
;Este programa suma dos valores inmediatos (7+8) y el resultado
;lo deposita en la posición 0x10
;Revisión: 0.0                          Programa para PIC16F84
;Velocidad de reloj: 4 MHz               Instrucción: 1Mz=1 us
;Perro Guardián: deshabilitado           Tipo de Reloj: XT
;Protección de código: OFF
;*****
LIST      p=16F84 ;Tipo de PIC
;*****
RESULTADO EQU 0x10      ;Define la posición del resultado
;*****
ORG 0          ;Comando que indica al Ensamblador
               ;la dirección de la memoria de programa
               ;donde situar la siguiente instrucción
;*****
INICIO        movlw    0x07      ;Carga primer sumando en W
               addlw    0x08      ;Suma W con segundo sumando
               movwf    RESULTADO ;Almacena el resultado

               END              ;Fin del programa fuente

```

**Cabecera**

**Tipo de microcontrolador**

**Definición de constantes**

**Comentarios**

**Programa**

## Tareas a realizar

(sólo hay que entregar los ejercicios 5, 6 y 7)

**Entrega P1 (Moodle)**

**23/02/18 – 23:59**

1. Escribir un programa en lenguaje ensamblador para la MCU 16F877A que sume dos números de 16 bits almacenados en memoria y almacene el resultado también en memoria.
2. Escribir un programa en lenguaje ensamblador para la MCU 16F877A que reste dos números de 16 bits almacenados en memoria y almacene el resultado también en memoria.
3. Escribir un programa en lenguaje ensamblador para la MCU 16F877A que compare dos números de 8 bits y almacene en la posición de memoria RESULT el mayor de los dos o cero en caso de igualdad.
4. Escribir un programa en lenguaje ensamblador para la MCU 16F877A que divida dos números de 8 bits almacenados en memoria en las posiciones DIVIDOR y DIVIDENDO, dejando el resultado también en memoria en las posiciones COCIENTE y RESTO.
5. Escribir un programa en lenguaje ensamblador para la MCU 16F877A que multiplique dos números de 8 bits almacenados en memoria y almacene el resultado en memoria en la posición RESULTADO\_ALTA y RESULTADO\_BAJA.
6. Escribir un programa en lenguaje ensamblador para la MCU 16F877A que convierta un número de 8 bits en binario natural, localizado en la posición de memoria BINARIO, en su representación BCD natural de tres dígitos, cada uno de ellos almacenado, de menor a mayor peso, en las posiciones de memoria BCD\_1, BCD\_2 y BCD\_3.
7. Escribir en lenguaje ensamblador una rutina de retardo variable desde 1 mseg hasta 1 segundo, parametrizable mediante el registro W, para una MCU 16F877A con una frecuencia de reloj de 8 MHz.