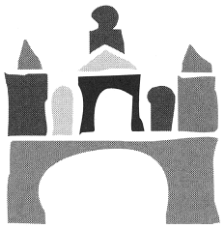
ESCUELA POLITÉCNICA SUPERIOR
UNIVERSIDAD DE BURGOS



HERRAMIENTA DE AUTOAPRENDIZAJE BASADO EN MICROCONTROLADOR OPERANDO EN LENGUAJE ENSAMBLADOR

ADAPTACIÓN AL GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

AUTORES:

Diego Bugallo Garrido

TUTORES:

Miguel Ángel Lozano

Pedro Luis Sánchez Ortega

JUNIO DE 2013

AGRADECIMIENTOS

Agradezco especialmente a mis tutores D. Miguel Ángel Lozano y Pedro Luis Sánchez Ortega, la inestimable ayuda en la elaboración de este Trabajo Fin de Grado.

También quiero agradecer a D. Carlos Germán de la Peña González, por su ayuda aportada en la estructuración del Trabajo fin de Grado.

Así mismo no puedo olvidar al departamento de electrónica de la Escuela Politécnica Superior en su conjunto, por los medios que pusieron en la realización de este trabajo.

ÍNDICE

1.- MEMORIA

1.1.- MEMORIA DESCRIPTIVA

1.1.1.- Antecedentes

El *Real Decreto 1393/2007, de 29 de octubre*, por el que se establece la ordenación de las enseñanzas universitarias oficiales indica que todas las enseñanzas oficiales de grado concluirán con la elaboración y defensa de un Trabajo Fin de Grado, que ha de formar parte del plan de estudios. El Trabajo Fin de Grado deberá realizarse en la fase final del plan de estudios y estar orientado a la evaluación de competencias asociadas al título.

El *Reglamento sobre trabajos fin de grado en la EPS de la Universidad de Burgos, de 26 de septiembre de 2011*, establece la reglamentación necesaria para la presentación y defensa del mismo.

1.1.2.- Objetivos

La finalidad de este proyecto es la realización de una serie de prácticas con el objetivo que unos alumnos puedan aprender **la programación de un microcontrolador**. Dichos alumnos pueden pertenecer a una universidad donde se imparte la asignatura de microcontroladores o bien alumnos que pertenezcan a un grado superior de formación profesional relacionado con la electrónica por ejemplo *técnico superior en desarrollo de productos electrónicos.*

Como objetivos secundarios, se busca el **manejo del programa MPLAB** capaz de realizar simulaciones, emulaciones y grabaciones de dispositivos físicos así como *comprender y ser capaz de montar circuitos donde intervengan microcontroladores.*

Por último también se busca que sean capaces, no solamente de seguir un guión preestablecido con las prácticas, sino la **autonomía personal**, mediante una serie de ejercicios propuesto que mejoraran la estructuración del pensamiento al tratarse de una programación estructural.

1.1.3.- Descripción de la solución adoptada

1.1.3.1.- El microcontrolador

En la programación de microcontroladores, existe una gran variedad de dispositivos de distintas marcas comerciales así como numerosas familias de cada una. Se ha optado por la elección de la marca microchip debido a:

- ✓ Gran variedad de PIC.
- ✓ Encajan perfectamente en el programa MPLAB.
- ✓ Uso generalizado.

1.1.3.2.- El PIC16F886

Dentro de los PICs existen numerosas familias, se ha optado por el PIC16F886, sus características principales se pueden observar en el apartado *otros anejos, el PIC16F886*:

- ✓ Suficientes entradas y salidas (24) para conectar a la vez varios dispositivos como un teclado numérico, un display y tener aún suficientes I/O para poder ampliar la programación.
- ✓ Posee tanto entradas y salidas analógicas como digitales.
- ✓ Dispone de tres temporizadores.
- ✓ Rango de frecuencia de trabajo elevado.
- ✓ Cuatro tipos de interrupciones distintas.
- ✓ Una memoria de datos suficiente para la programación.

- ✓ Sencillez en la programación en ensamblador, son sólo 35 instrucciones es capaz de programar.

1.1.3.3.- MPLAB

Para la selección del programa, se ha optado por el MPLAB que es capaz de simular, emular y grabar sobre dispositivos físicos. Se opta por este programa debido a:

- ✓ **Freeware**, se trata de un software gratuito, por lo que su uso está bastante extendido en los centros, y cualquier persona interesada en el aprendizaje de la programación puede aprender sin invertir dinero.

- ✓ El software se va actualizando con el tiempo, lo que permite corregir bugs e implementar el programa. Actualmente está disponible la versión X.

- ✓ Al ser un software creado por microchip, encaja perfectamente en el tipo de microcontrolador elegido.

1.1.3.4.- Lenguaje ensamblador

El lenguaje que va ser utilizado es el lenguaje ensamblador, pese a la diversidad de lenguajes de programación de un microcontrolador, se ha optado por dicho lenguaje debido a las siguientes ventajas:

- ✓ La programación a bajo nivel, permite una mejor compresión del funcionamiento del microcontrolador.

- ✓ Una programación en ensamblador tiene la ventaja a la hora de programar , del uso específico de registros, así como una mejor compactación del código, lo que permite usar menor memoria de programa.

- ✓ Esta programación es permite posteriormente expandirse a programaciones de alto nivel sin ningún tipo de problemas de comprensión, el paso contrario requiere más tiempo.

- ✓ La programación a bajo nivel está íntimamente relacionada con el grado de electrónica y automática.

1.1.3.5.- Laboratorio PIC'School

A la hora de realizar los montajes, se ha optado por el *"maletín"* Laboratorio PIC'School en vez de por el cableado clásico por diversas razones:

- ✓ Está preparado para usarse principalmente con los PICs y el MPLAB.
- ✓ Para realizar el montaje es mucho más cómodo y visual realizarlo, no perdiendo excesivamente tiempo en el montaje, lo que permite más tiempo para el objetivo principal.
- ✓ Su portabilidad es una gran ventaja.
- ✓ Dispone de librerías ya creadas para el manejo de elementos como el LCD, teclado, temporizadores.
- ✓ Dispone de una amplia documentación de su manejo.

1.1.3.6.- Prácticas

A la hora de la realización de las distintas prácticas, se ha estimado dividir las prácticas en dos grandes bloques,

- *Prácticas 1-3:* Consta de tres prácticas programadas únicamente para la simulación, no disponen de ninguna entrada física o salida por lo que no se realizará montaje alguno. Con el fin de adquirir los conocimientos del lenguaje de programación y sus técnicas. Al no realizar el montaje, contienen un **ejercicio adicional de ampliación** de conocimientos, profundizando sobre el tema de la práctica.
- *Prácticas 4-9:* Estás prácticas requieren además de la simulación **el montaje** de dichas prácticas. Al considerar importante en el aprendizaje, no solamente la realización de un código de programación, sino también visualizar su funcionamiento realmente, así como adquirir soltura a la hora de ensamblar dispositivos.

1.1.3.7.- Reparto horario

La distribución horaria de las 150 horas (ver anejo de datos de partida) se representa en la Tabla 1:

Tabla 1. *Distribución horaria de las prácticas.*

Práctica	Teoría	Ejercicio	Ejercicio ampliación	Montaje	Programas propuestos	TOTAL
1	4 horas	4 horas	4 horas	-	8 horas	*20 horas*
2	4 horas	4 horas	4 horas		8 horas	*20 horas*
3	4 horas	4 horas	4 horas		8 horas	*20 horas*
4	3 horas	4 horas	-	2 horas	6 horas	*15 horas*
5	3 horas	4 horas	-	2 horas	6 horas	*15 horas*
6	3 horas	4 horas	-	2 horas	6 horas	*15 horas*
7	3 horas	4 horas	-	2 horas	6 horas	*15 horas*
8	3 horas	4 horas	-	2 horas	6 horas	*15 horas*
9	3 horas	4 horas	-	2 horas	6 horas	*15 horas*
Todas	**30 horas**	**36 horas**	**12 horas**	**12 horas**	**60 horas**	***150 horas***

En cuanto a esta disposición se aprecia lo establecido en el anterior apartado 6 de la justificación de la solución adoptada (*Prácticas*) que indica la realización de un ejercicio de ampliación en las primeras tres prácticas no así en el resto de ellas debido al tiempo necesario para el montaje de dichas prácticas.

Se ha establecido un criterio de tiempos de tal forma que las primeras tres prácticas correspondan con el **40%** del tiempo empleado, mientras que el resto de las seis prácticas llenen el **60%.**

El objetivo de esta estructuración es la dedicación de una gran parte del tiempo al aprendizaje del lenguaje ensamblador aplicado a los microcontroladores durante las tres primeras prácticas. Una vez aprendido el lenguaje, tener una buena cimentación para encarar el resto de prácticas.

1.1.4.- Soluciones no adoptadas

1.1.4.1.- El PIC16F84

Dentro de los PICs existen numerosas familias, otro microcontrolador que fue analizado, a la hora de la elección fue el PIC16F84, es un microcontrolador con peores características aunque resulta algo más sencillo de programarlo, al contener menor número de registros, temporizadores, etc.

Sin embargo el número de entradas y salidas son muy limitadas (13), lo que hace imposible programar a la vez un display y un teclado.

1.1.4.2.- PROTEUS

A la hora de simular existen otros programas que realizan esta función, como por ejemplo el PROTEUS, sin embargo no se opto por las siguientes razones:

- ✓ Se requiere una licencia para su uso completo.
- ✓ Esta más pensado para la programación en alto nivel.
- ✓ Su uso no es tan extendido como el programa MPLAB.

1.1.4.3.- Lenguaje C

Otra opción a la hora de programar, fue la programación a alto nivel en lenguaje C, se desestimó esta opción debido a:

- ✓ La compresión del PIC a alto nivel es muy limitada.
- ✓ Ocupa más espacio de memoria de programa.
- ✓ La programación a alto nivel esta menos relacionada con el grado de electrónica y automática.

1.1.4.4.- Cableado

Una opción que se tuvo en cuenta a la hora del montaje fue el cableado simple de los circuitos, razones por las que se desecho está idea fueron las siguientes:

- ✓ Requiere un mayor tiempo de montaje.
- ✓ Su portabilidad puede llegar a ser incómoda
- ✓ Es más fácil realizar errores de conexión, y puede llevar más tiempo detectarlos.

1.1.4.5.- Prácticas

Otra posible distribución analizada de las prácticas fue la introducción de una práctica más de montaje en detrimento de una práctica de aprendizaje del lenguaje propiamente dicha:

- *Prácticas 1-2:* Prácticas programadas únicamente para la simulación, con el fin de adquirir los conocimientos del lengua de programación y sus técnicas.
- *Prácticas 3-9:* Estás prácticas requieren además de la simulación **el montaje** de dichas prácticas.

Sería una práctica menos de comprensión del lenguaje y una práctica más de montaje, como por ejemplo la introducción de una práctica que enseñe al manejo de un conversor analógico digital, sin embargo, se estima escaso el aprendizaje del lenguaje con únicamente dos prácticas, pudiendo arrastrar el problema a lo largo del resto de prácticas.

La carencia de una práctica del conversor analógico digital, no es tan importante ya que en la *práctica 4* se explica en el funcionamiento de los puertos, como programar una entrada analógica o digital y simplemente faltaría realizar el análisis del registro donde se almacena el dato, estudio sencillo debido a la gran cantidad de registros analizados en este trabajo.

1.1.5.- Prácticas con el PIC16F886

1.1.5.1.- Consideraciones previas

En este apartado se desarrollan una serie de prácticas para la programación del microcontrolador, estás prácticas se estructuran de la siguiente manera:

- Practica 1: Operaciones aritmético-lógicas.
- Practica 2: Bucles y subrutinas.
- Practica 3: Modos de direccionamiento y tablas.
- Practica 4: Puertos entrada y salida.
- Practica 5: Temporizadores y contadores.
- Practica 6: Interrupciones.
- Practica 7: Memoria EEPROM.
- Practica 8: La pantalla LCD.
- Practica 9: El teclado.

1.1.5.2.- El PIC16F886

La programación de las prácticas se realizan sobre el microcontrolador PIC16F886, se pueden estudiar sus características técnicas en *otros anejos a la memoria, el PIC16F886* y *anejo de instrucciones*.

Aunque el contenido íntegro se basa en este PIC, se puede extrapolar a todos los PICs existentes y sería sencilla la transición del PIC16F886 a otro microcontrolador, simplemente se tendría que obtener la hoja de características del microcontrolador y en ella están desglosados las posiciones de los registros del PIC, el código de instrucciones soportado, el número de temporizadores y demás características necesarias para la programación.

1.1.5.3.- El MPLAB

A la hora de realizar la simulación emulación y grabación del dispositivo, se emplea el programa MPLAB. En *otros anejos a la memoria, MPLAB* hay un tutorial que explica el funcionamiento del programa.

1.1.5.4.- Practicas

1.1.5.4.1.- Practica 1: Operaciones aritméticas y lógicas

1.1.5.4.1.1.- Objetivos

- ✓ Manejo del programa MPLAB.
- ✓ Iniciarse en la programación en ensamblador.
- ✓ Manejar las operaciones aritmético-lógicas.
- ✓ Editar, compilar y simular un programa.

1.1.5.4.1.2.- Enunciado

Sigue secuencialmente las instrucciones propuestas a continuación:

- ✓ Crea 3 constantes de valores 16 en decimal, 11 en hexadecimal y 101 en binario asignándolas el nombre de *cte1, cte2 y cte3* respectivamente.
- ✓ Crea 7 variables de nombre *var1, var2, var3, res1, res2 , res3 y resultado* en posiciones consecutivas de memoria a partir de la dirección 0x10.
- ✓ Asigna los valores de 5 en decimal, 12 en hexadecimal y 3 en binario a los registros *var1, var2 y var3* respectivamente.
- ✓ Realiza las siguientes operaciones aritméticas:
  - ➢ res1=var1+cte1+5
  - ➢ res2=var2+ct2-var3
  - ➢ resultado=res1+res2
  - ➢ resultado=resultado+1
  - ➢ res1=res1-1
- ✓ Realiza las siguientes operaciones lógicas:
  - ➢ Borra las variables res1, res3 y el registro de trabajo.
  - ➢ res1 = var3 AND cte3
  - ➢ res3= res1 OR b'00010110'
  - ➢ Borra los bits 0,3 y 5 de la variable res3 utilizando la función bsf
  - ➢ Realiza una operación al número b'10000000' que ponga a uno el flag de cero

1.1.5.4.1.3.- Ejercicio de ampliación

Sigue secuencialmente las instrucciones propuestas a continuación:

- ✓ Realiza la siguiente operación:
- ✓ res1=2(a-1)+2(b+1)-c-d / a=3, b=2, c=4, d=2 con c, d constantes.
- ✓ Introduce el valor de b'11011011' en una variable, borra el bit 0 y el bit 4 sin modificar el resto aplicando una máscara and.
- ✓ Al resultado anterior, intercambia de valor los bits, de la parte baja (0-3) sin modificar la parte alta, mediante una máscara XOR.
- ✓ Intercambia la parte baja por la alta del resultado anterior.
- ✓ Realiza una operación que ponga a uno el flag C (Carry) sin activar el flag DC (Digit Carry)
- ✓ Realiza una operación que ponga a uno el flag DC sin poner a uno el flag C.
- ✓ Realiza una operación con la función decremento que ponga a uno el flag de cero.

1.1.5.4.1.4.- Contenido teórico

1.1.5.4.1.4.1.- Instrucciones

Las instrucciones utilizadas por el PIC16F886 se encuentran resumidas en el anejo de instrucciones.

1.1.5.4.1.4.2.- Constantes y variables

Un registro de la memoria de datos ocupa una posición de memoria cuya dirección va desde la posición 00 a 1F0 [7], al programar en ensamblador para referirse a un registro concreto se puede indicar su posición con un número hexadecimal indicando la dirección correspondiente o asignar un nombre a esa posición y después referirse a él por dicho nombre.

La siguiente figura muestra un ejemplo a la hora de usar la memoria RAM, la diferencia entre *la dirección, el contenido y el nombre* del registro, (Figura 1):

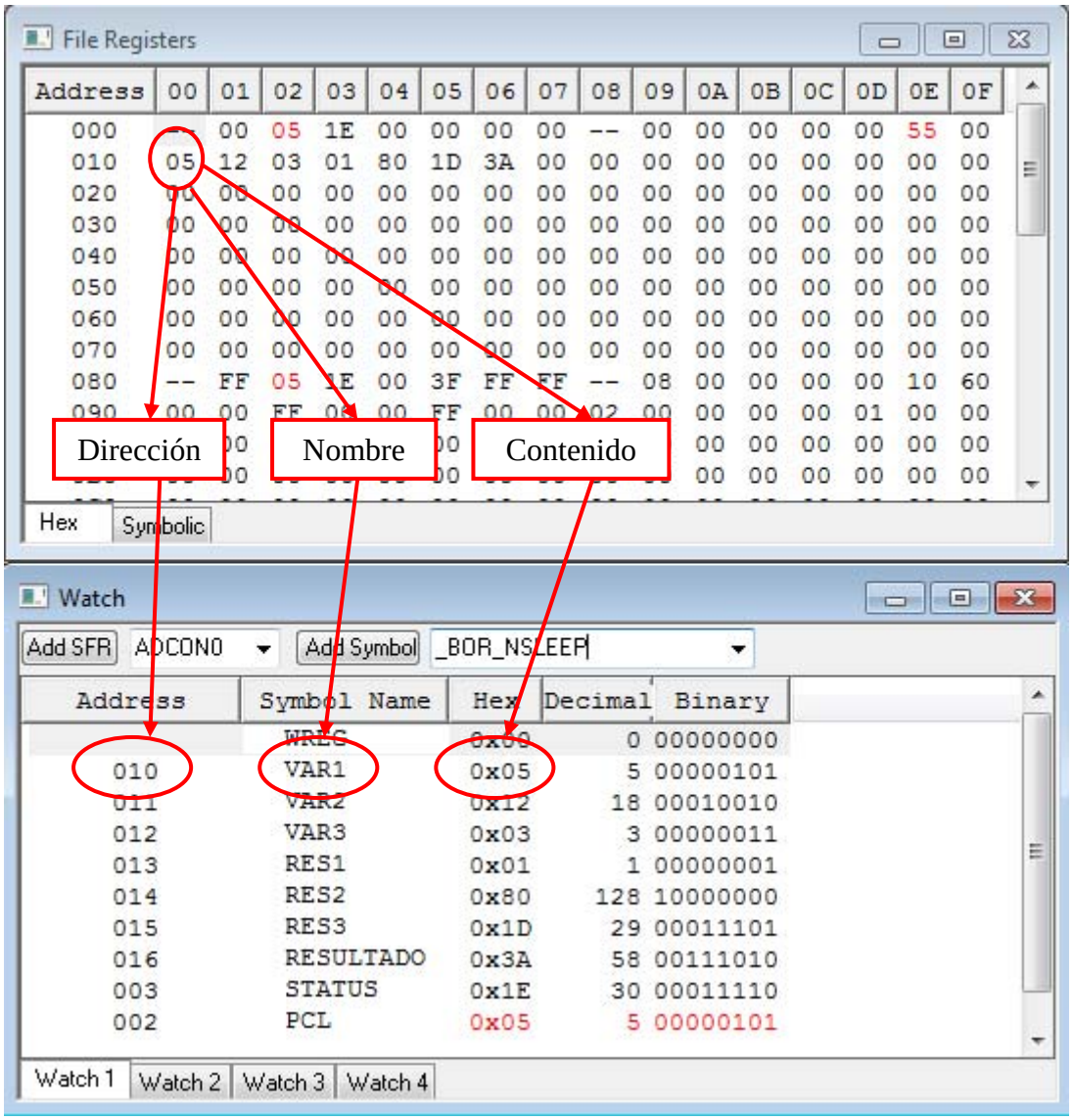


Figura 1. *Análisis de la memoria RAM.*

De tal forma que para acceder al dato marcado en la figura 1, 0x05h durante la programación puedo referirme a él como el registro 0x10h o como VAR1.

Por similitud con otros lenguajes de programación a los nombres de registros creados por el programador se denominarán **variables**, estas variables se pueden modificar en cualquier momento del programa.

Por el contrario, **las constantes**, no podrán ser modificadas en ningún momento del programa y no ocupan ningún espacio en la memoria de datos, simplemente es una asignación de un valor a una palabra que luego al compilar y crear el *.hex* cambiará esa palabra por el valor definido.

Todas las constantes y variables que se vayan a usar en el programa hay que declararlas previamente y siguen la siguiente estructura:

- ✓ `#define nombre_constante valor`
- ✓ `nombre_variable equ  valor_posición_memoria`

Las distintas formas de introducir un valor a un registro son:

- ✓ **Binario:** b'xxxxxxxx', siendo x 0 ó 1
- ✓ **Decimal:** Con un punto que precede al número (por ejemplo " .12 ")
- ✓ **Hexadecimal:** Existen distintas formas 0x3fh, 0x3f, 3fh ó 3f.
- ✓ **Constante:** Con el nombre de una constante previamente definida.

A continuación se muestran diferentes formas de declaraciones, (Figura 2)

```
    list p=16F84            ; Tipo de PIC
    #include "P16F84.INC"    ; Archivo que identifica los ppales regis

;-----------------------------------------------------
;---    DECLARACIÓN DE CTES. Y VARIABLE               -
;-----------------------------------------------------

    #define constante1 .10          ; Número decimal
    #define constante_2 0x10        ; Número hexadecimal
    #define constante3 b'00000010'  ; Número binario

    VARIABLE1 equ   0x20            ; Posición de memoria 20h
    VARIABLE2 equ   VARIABLE1+1     ; Posición de memoria 21h
    VARIABLE3 equ   constante_2     ; Posición de memoria 10h

;-----------------------------------------------------
;---    INICIO DEL PROGRAMA PRINCIPAL                 -
;-----------------------------------------------------

    org 0x00            ; Posición en la que comienza el programa
```

Figura 2. *Declaración de constantes y registros.*

A la hora de programar, como el MPLAB no distingue por colores las variables, constantes y etiquetas para facilitar el estudio del programa se escribirán *los nombres de las constantes en minúsculas, las variables en mayúsculas y las etiquetas con la primera letra en mayúsculas.*

1.1.5.4.1.4.3.- Registros importantes

- Contador de programa (PC)

Formado por dos registros, la parte baja o PCL (0x02h) y la parte alta o PCLACH (0x0Ah) [7] de los cuales únicamente intervienen los primeros cinco bits. Trabaja a modo de puntero conteniendo *la dirección de memoria de programa de la siguiente instrucción a ejecutar*.

- Registro Status (STATUS)

El registro status es un registro interno del microcontrolador ubicado en la dirección 0x03h de la memoria de datos, encargado de proporcionar información del estado del PIC y de las operaciones realizadas por la ALU. A continuación se muestra el contenido del registro status en la Tabla 2 :

Tabla 2. *El registro STATUS.*[7]

REGISTRO STATUS (0x03h)							
BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
IRP	RP1	RP0	TO	PD	Z	DC	C

Los bits más importantes usados en esta primera práctica son:

- ✓ **Z**: "Zero bit", se activa si el resultado de una operación es cero.
- ✓ **DC**: "Digit Carry", se activa si en las operaciones addlw, addwf, sublw o subwf hay acarreo en el cuarto bit de menos peso
- ✓ **C**: "Carry" de igual forma que con el flag DC se activa si hay acarreo en el bit más significativo, es usado en los desplazamientos a izquierdas o derechas.

- El registro de trabajo (W)

Este registro es un almacén temporal de datos, y se emplea a modo de acumulador en los PICs. Para la mayoría de las operaciones aritméticas o lógicas se usa el registro de trabajo, las únicas operaciones que no usan este registro son el *incremento, decremento, borrado, complemento, rotación, intercambio y borrado o seteo de un bit.*

Conviene hacer un clear del registro de trabajo en la primera instrucción del programa, asegurando que al empezar el programa su valor es cero, (Figura 3):

```
;-----------------------------------------------------
;---    INICIO DEL PROGRAMA PRINCIPAL                -
;-----------------------------------------------------

    org 0x00            ; Posición en la que comienza el programa
    goto Inicio         ; Salto a la etiqueta Inicio


    org 0x05            ; Posicion de la etiqueta Inicio

Inicio                  ; Etiqueta de comienzo de programa
      clrw              ; Pongo a cero el registro de trabajo
```

Figura 3. *Borrado inicial del registro de trabajo.*

1.1.5.4.1.4.4.- Manejo de datos

Para pasar un dato a una variable, es necesario el uso del registro de trabajo, a continuación se muestran algunos ejemplos:

- Dar un valor determinado a una variable,VAR1 = 10, (Figura 4).

```
movlw   .10     ; W = 10
movwf   VAR1    ; VAR1 = W  = 10
```

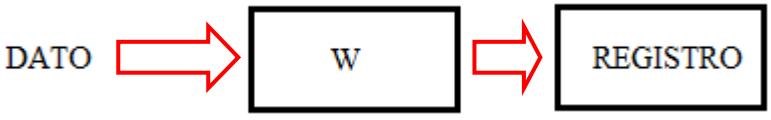


Figura 4. *Almacenar datos en variables (1).*

Primero se pasa el dato al registro de trabajo y a continuación de W a la variable deseada.

- Dar un valor determinado a una variable definido por una constante,VAR2 = cte., (Figura 5)

```
movlw   cte1    ; W = cte1 = 5
movwf   VAR2    ; VAR2 = W  = cte1 = 5
```

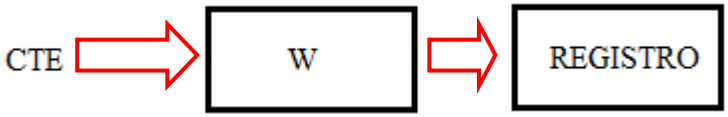


Figura 5. *Almacenar datos en variables (2).*

Sigue el mismo proceso que el anterior caso, pero previamente hay que definir el valor de la constante.

- Dar el valor de una variable a otra variable ,VAR3 = VAR1, (Figura 6)

```
movf    VAR1,W  ; W = VAR1 = 10
movwf   VAR3    ; VAR3 = W = 10
```

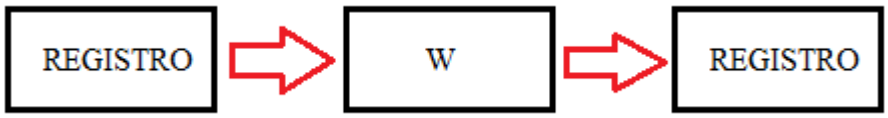


Figura 6. *Almacenar datos en variables (3).*

Primero se pasa el valor de la variable fuente al registro de trabajo y posteriormente se pasa el valor de W a la variable destino.

1.1.5.4.1.4.5.- Operaciones aritméticas

Las únicas operaciones aritméticas capaces de realizar por el microprocesador 16F84 son las suma y la resta.

- *Incremento / Decremento de registros*: Estás dos operaciones suman o restan una unidad a un registro, no usan el registro de trabajo, (Figura 7).

```
incf    VAR1    ; VAR1 <-- VAR1 + 1
decf    VAR1    ; VAR2 <-- VAR2 - 1
```

Figura 7. *Incremento y decremento.*

- *Suma y resta de dos números*, (Figura 8):

```
addlw   cte1    ; W = W + cte1
sublw   cte2    ; W = cte2 - W
addwf   VAR1,W  ; W = var1 + w
subwf   VAR2,VAR2 ; var2 = var2 - w
```

Figura 8. *Sumas y restas.*

Cada registro está compuesto de 8 bits, por lo que el **número máximo** que se puede obtener al sumar dos números sin que exista desbordamiento es de 255.

1.1.5.4.1.4.6.- Operaciones lógicas

Las operaciones lógicas que puede realizar son la operación *AND, OR XOR, complemento, rotación, intercambio y puesta a uno y a cero de un bit*. A continuación se muestran estos ejemplos, (Figura 9):

```
andlw   cte1    ; W = W & cte1
iorwf   VAR1,w  ; W = W OR VAR1
xorwf   VAR2,VAR2 ; var2 = W XOR VAR2
rlf     VAR2,W  ; almaceno en w la rotación a izq.
rrf     VAR2,VAR2 ; Almaceno en VAR2 la rotación a dcha.
swapf   VAR3,w  ; Intercambio la parte baja con la alta
bcf     VAR3,0  ; borro el bit 0 de VAR3
bsf     VAR3,7  ; Pongo a uno el bit 7 de VAR3
```

Figura 9. *Operaciones lógicas comunes.*

1.1.5.4.1.4.7.- La instrucción END

Esta instrucción debe colocarse al final del código, con ella se le indica al ensamblador que el programa a finalizado, es decir el ensamblador compilará hasta dicha instrucción.

1.1.5.4.1.4.8.- Contenido teórico de ampliación

- *El desplazamiento como multiplicación o división*

Cuando se realiza un desplazamiento, lo primero que hay que tener en cuenta es el estado del acarreo (C), ya que el desplazamiento se hace a través de él.

Si el desplazamiento es a izquierdas, y el bit C esta a '0' , e trata de añadir un cero a la derecha del número, es decir realizar una multiplicación por el valor 10. **Al estar trabajando en binario, está realmente multiplicando por dos en decimal**.

Por el contrario si el desplazamiento es a la derecha y el acarreo tiene el valor de cero, se **dividiría entre dos**, si el bit menos significativo es cero, sería una división pura, si vale '1' estaría truncando el resultado. El bit C después del desplazamiento informaría si el resultado es decimal (.5) o no.

- *Máscaras*

Las máscaras son instrumentos lógicos para obtener información de bits concretos, o modificar el estado de algún bit o bits. Las más usadas son la AND, OR y XOR.

- ✓ *AND*: Un uso bastante extendido es la puesta a cero de determinados bits de un puerto como se observa en la Tabla 3:

Tabla 3. *Máscara AND.*

PUERTO A INCIAL	1	0	1	1	0	1	1	0
MÁSCARA AND	0	0	0	0	1	1	1	1

PUERTO A FINAL	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**

Con esta máscara se consigue apagar las salidas RA7-RA4 y mantener como se encontraban las salidas RA3-RA0.

- ✓ *OR*: Se usa de forma similar a la máscara and, para setear salidas. (Tabla4):

Tabla 4. *Máscara OR.*

PUERTO A INCIAL	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**
MÁSCARA OR	1	1	1	1	0	0	0	0
PUERTO A FINAL	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**0**

Consiguiendo poner a uno las salidas RA7-RA4 y mantener como estaban las salidas RA3-RA0.

- ✓ XOR: Se usa para intercambiar el estado de las salidas.(Tabla 5)

Tabla 5. *Máscara XOR*

PUERTO A INCIAL	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**
MÁSCARA XOR	1	1	1	1	0	0	0	0
PUERTO A FINAL	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**0**

Consiguiendo cambiar de estado a las salidas ra7-ra4 y mantener como estaban las salidas ra3-ra0.

El uso de máscaras en la programación es bastante útil a la hora de testear el estado de un bit concreto, una vez realizada la máscara con instrucciones de control del flujo que se estudiarán en la siguiente práctica se puede regular el flujo del código en función de los bits deseados sin necesidad de modificar el resto de bits del registro en que se está trabajando.

1.1.5.4.1.5.- Ejercicio básico

1.1.5.4.1.5.1- Planteamiento y diagrama de flujo

En esta primera práctica, simplemente se van siguiendo las instrucciones una a continuación de otra, sin realizar ningún salto ni llamada, siguiendo los pasos indicados en el enunciado, por lo que no se considera necesaria la explicación del planteamiento ni la realización de un diagrama de flujo.

1.1.5.4.1.5.2- Programación y simulación

Lo primero que se pide es la declaración de constantes y variables que se van a utilizar, para ello seguimos los pasos indicados en el apartado *constantes y variables* de esta misma práctica, (Figura 10)

```
;-----------------------------------------------------
;---    DECLARACIÓN DE CTES. Y VARIABLE               -
;-----------------------------------------------------

    #define cte1 .16            ; Número decimal
    #define cte2 0x11           ; Número hexadecimal
    #define cte3 b'00000101'    ; Número binario

    VAR1 equ    0x10        ; Posición de memoria inicial 0x10h
    VAR2 equ    VAR1+1
    VAR3 equ    VAR1+2
    RES1 equ    VAR1+3
    RES2 equ    VAR1+4
    RES3 equ    VAR1+5
    RESULTADO equ   VAR1+6  ; Posición de memoria final 0x16h
```

Figura 10 . *Declaraciones.*

Al simular el programa, nada más comenzar, las variables son asignadas a sus posiciones de memoria, y el puntero se coloca en la posición 0x00 de la memoria de programa. Para poder observarlo (ver anejo de MPLAB) se puede ir a la ventana watch, donde se le indicará las variables a visualizar, o la ventana file registers que realiza una distribución tabulada de la memoria de datos al completo. (Figura 11)

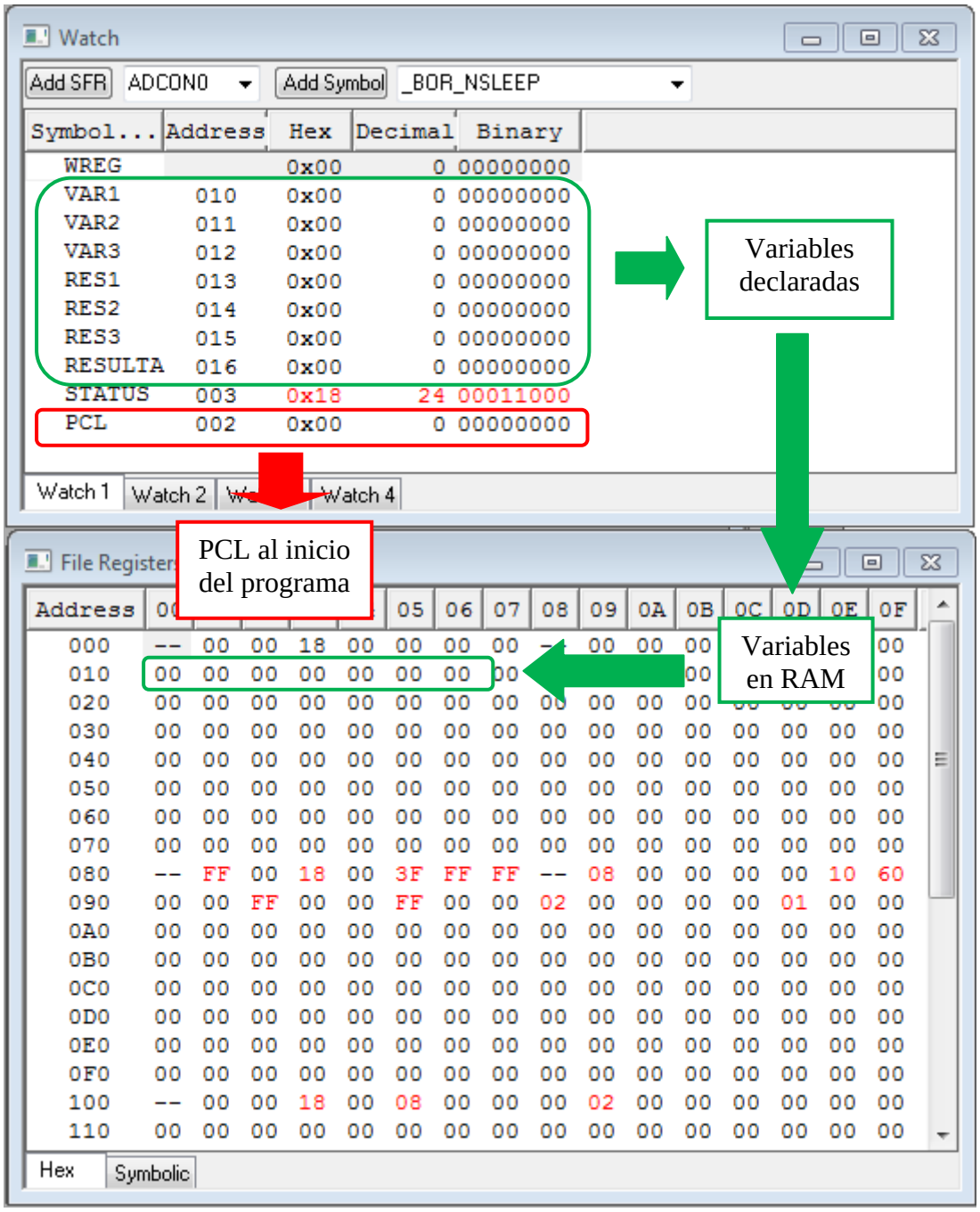


Figura 11.- *Estudio inicial de los registros.*

En adelante, salvo que sea necesario, se mostrara **solamente la ventana watch** donde se indicarán los registros deseados sin mostrar la venta de la memoria de datos.

A continuación se debe comenzar el programa, al no utilizar interrupciones (ver práctica 6) el inicio del programa sería el siguiente, (Figura 12).

```
;-------------------------------------------------------
;---    INICIO DEL PROGRAMA PRINCIPAL                  -
;-------------------------------------------------------

    org 0x00    ; Posición de memoria de prog. que comienza
    goto Inicio ; Salto a la etiqueta Inicio


    org 0x05    ; Posicion de memoria de prog. de la etiqueta Inicio

Inicio          ; Etiqueta de comienzo de programa
```

Figura 12 . *Código inicial de un programa sin interrupciones.*

Se observa como al ejecutar la primera instrucción salta de la posición inicial 0x00h a la posición Inicio a la que se le dio la dirección 0x05h con la instrucción org. El contador de programa (PC) apuntará a la siguiente dirección de instrucción a ejecutar (0x05h), en este caso la instrucción clrw. Hay que tener en cuenta que **una etiqueta no es una instrucción**, por esto clrw tiene la dirección 0x05h, (Figura 13).

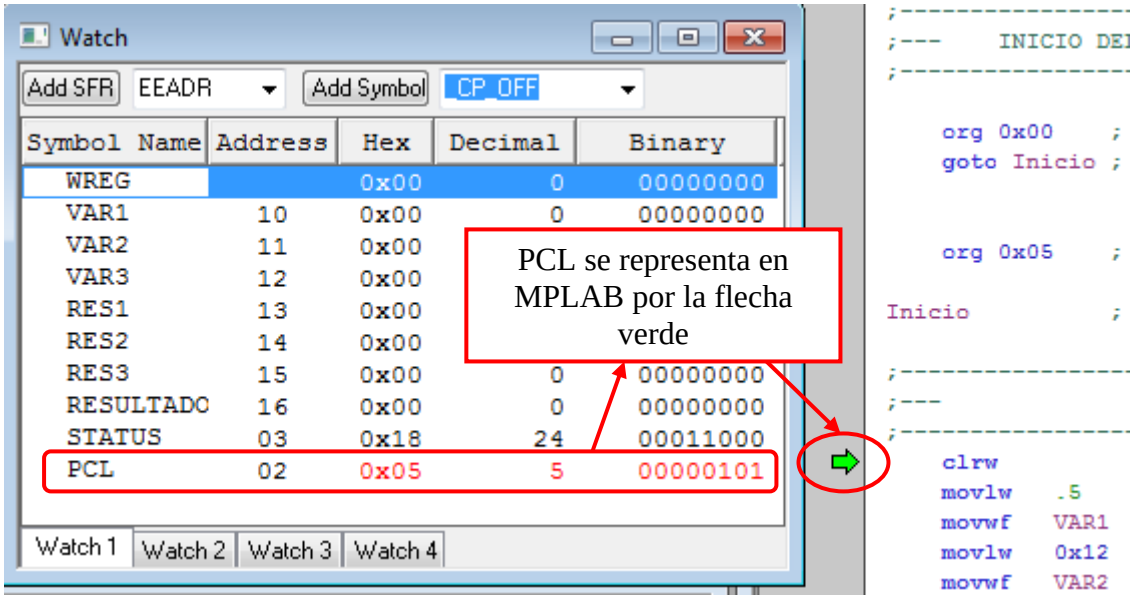


Figura 13. *Análisis del contador de programa.*

Después se pide que se inicien las variables a los siguientes valores:

- ✓ VAR1 =. 5
- ✓ VAR2 =12h
- ✓ VAR3=b'00000011'

Como se indica en la figura 5, Primero se introduce el valor al registro de trabajo, y desde W se pasa a la variable deseada. Para ello se emplea el siguiente código, (Figura 14):

```
;-----------------------------------------------------------
;---                VALORES INICIALES                      -
;-----------------------------------------------------------
    clrw                ; W = 0
    movlw   .5
    movwf   VAR1        ; VAR1 = 5
    movlw   0x12
    movwf   VAR2        ; VAR2 = 12h
    movlw   b'00000011'
    movwf   VAR3        ; VAR3 = b'00000011'
```

Figura 14. *Copia de datos a los registros del programa.*

Es conveniente nada más empezar el programa realizar un clear del registro de trabajo, esto se debe a que el programa a veces asigna valores erróneos a W, cuando se realiza un reset de las memorias y sin compilar se empieza el programa.

A la hora de introducir los datos, se ha usado la opción del número directamente en decimal, hexadecimal y binario. Otra opción hubiera sido la creación de tres constantes con la instrucción *define* e indicar el valor de dichas constantes, como se explica en el apartado *contenido teórico* de esta práctica.

A continuación se muestra la simulación de como obtiene la primera variable el valor cinco en decimal.(Figura 15)

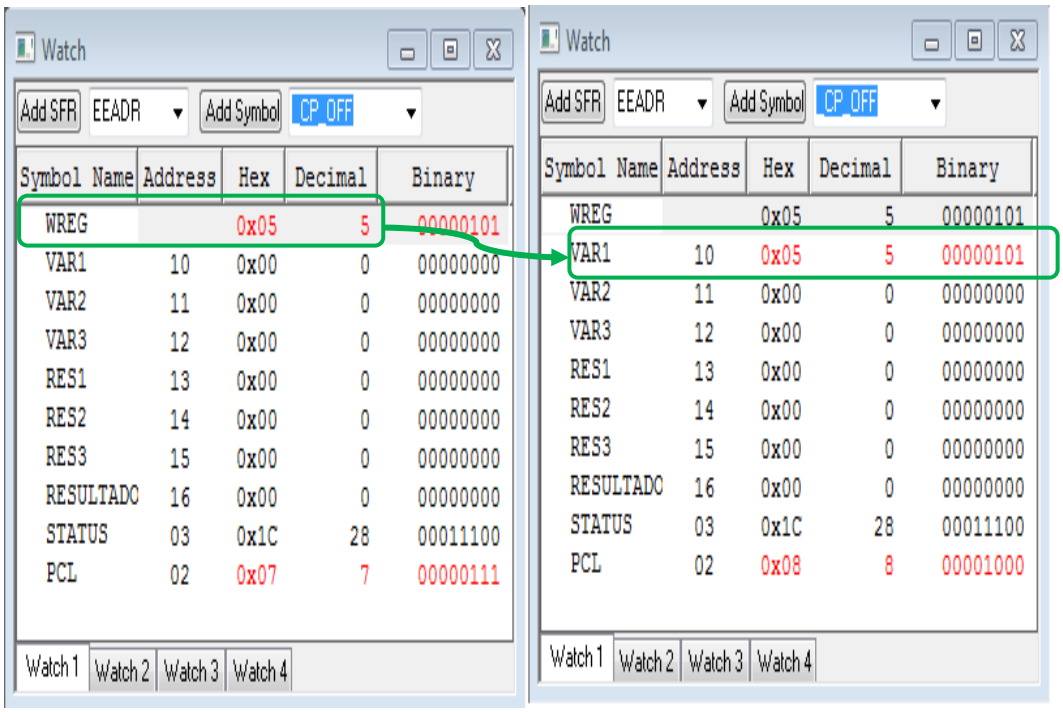


Figura 15. *Asignación de un valor al registro VAR1.*

Una vez asignado los tres valores a las variables, se obtendría, (Figura 16)

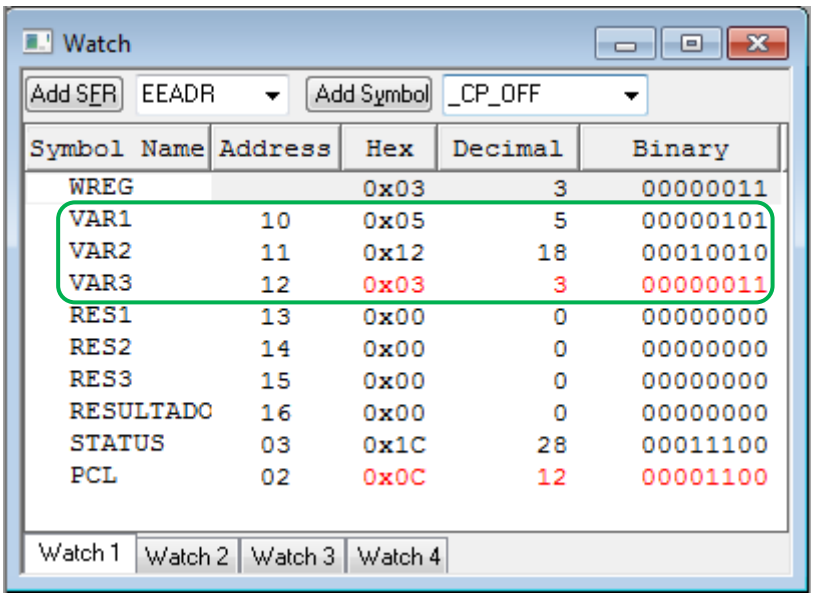


Figura 16. *Asignación de los tres valores del enunciado.*

A continuación se debe realizar la operación RES1 = VAR1 + cte1 + 5. El microcontrolador solamente es capaz de *realizar la suma de dos números siempre que uno de ellos se encuentre en el registro de trabajo*, por lo que para realizar la operación se van a ir acumulando los valores en RES1 como indica la figura 17:

```
;-------------------------------------------------------
;---                OPERACIONES ARITMÉTICAS            -
;-------------------------------------------------------

; RES1 = VAR1 + cte1 + 5
    movlw   .5
    movwf   RES1        ; RES1 = 5
    movlw   cte1
    addwf   RES1,RES1   ; RES1 = 5 + cte1
    movf    VAR1,W
    addwf   RES1,RES1   ; RES1 = 5 + cte1 + VAR1
```

Figura 17. *Código del ejercicio aritmético (1).*

Se observa en las siguientes figuras como vamos acumulando los valores en la variable resultado, hasta obtener el valor final, (Figura 18, 19 y 20):

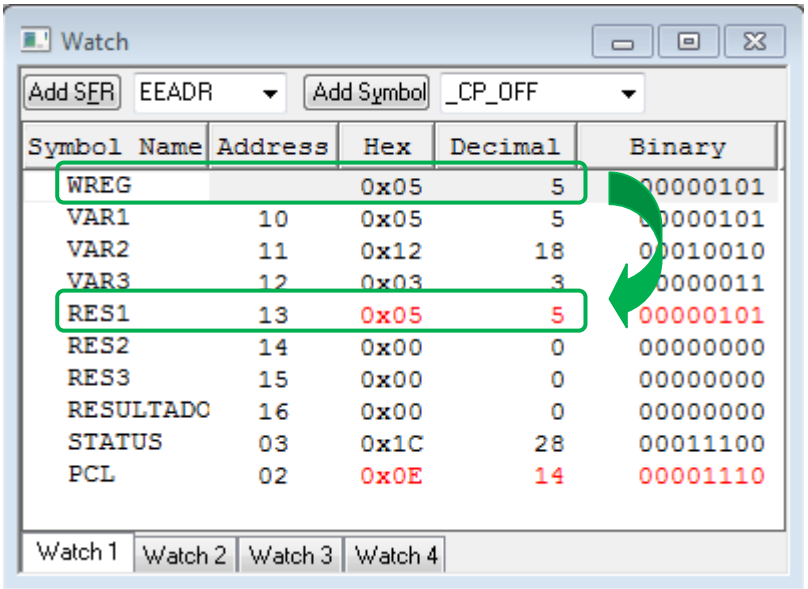


Figura 18. *Simulación del ejercicio aritmético (1).*

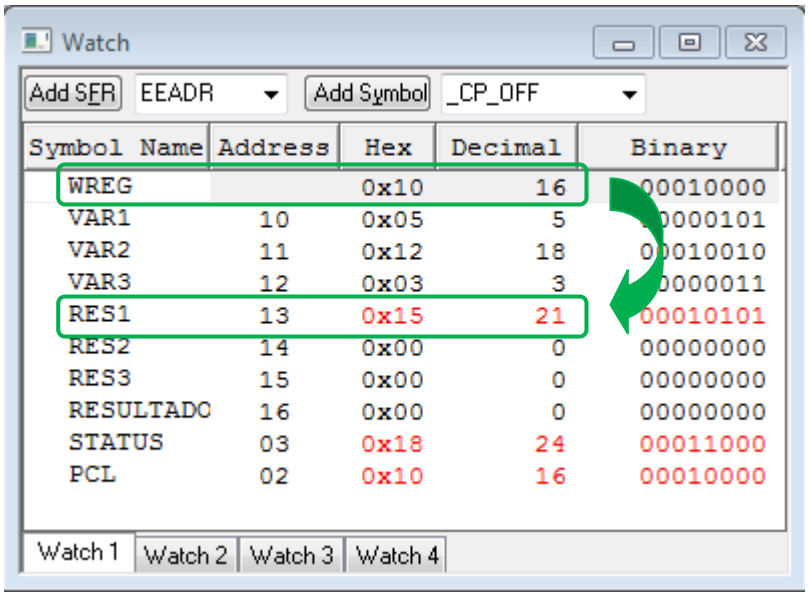


Figura 19. *Simulación del ejercicio aritmético (2).*

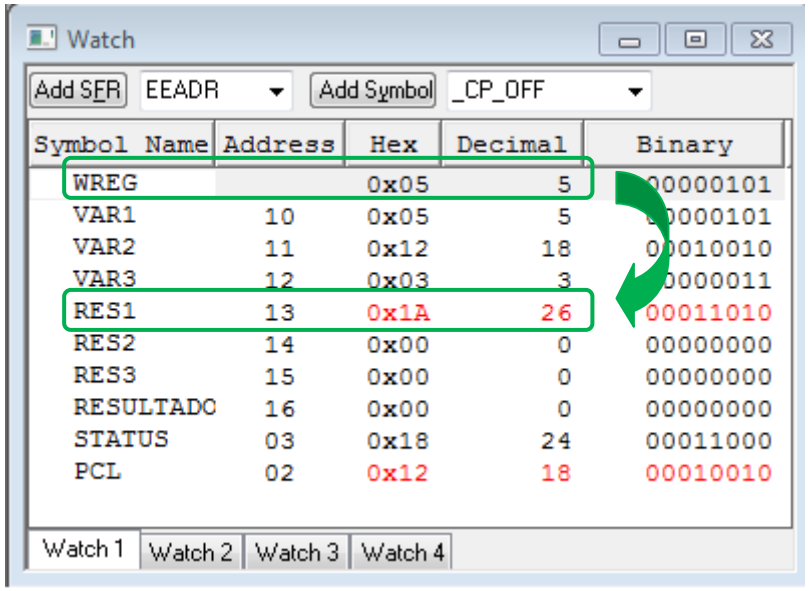


Figura 20 *Simulación del ejercicio aritmético (3).*

La siguiente operación a realizar es RES2 = VAR2 + cte2 - VAR3. Con la resta ocurre lo mismo, *hay que utilizar W, en el introduciremos el valor del sustraendo*. Para realizar la operación vamos ir acumulando los valores en RES2 y al final sustraer los valores deseados.

El código que realiza esta operación se ilustra en la figura 21:

```
; RES2 = VAR2 + cte2 - VAR3
    movf    VAR2,W
    movwf   RES2        ; RES2 = VAR2
    movlw   cte1
    addwf   RES2        ; RES2 = VAR2 + cte1
    movf    VAR3,W
    subwf   RES2        ; RES2 = VAR2 + cte1 - VAR3
```

Figura 21 . *Código ejercicio aritmético (2).*

El resultado final se muestra a continuación, (Figura 22):

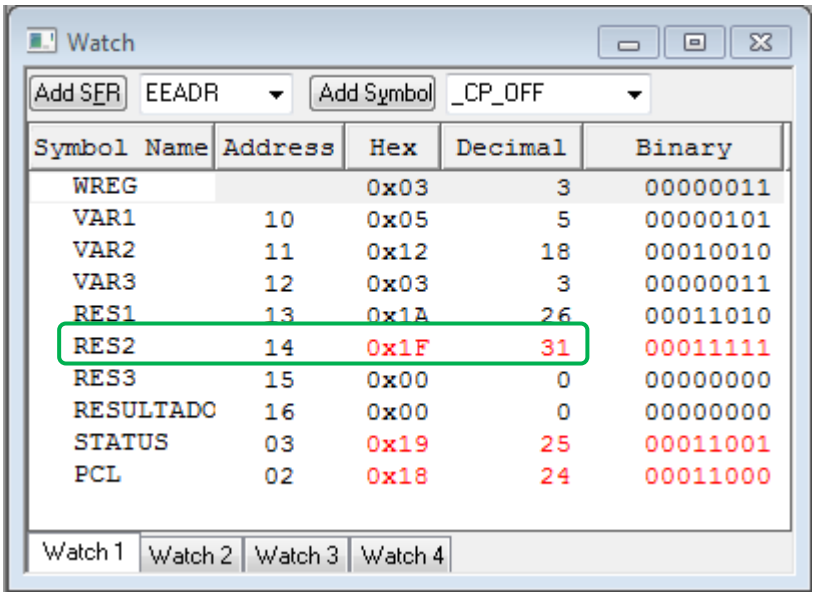


Figura 22. *Simulación del ejercicio aritmético (4).*

La siguiente operación es RESULTADO = RES1 + RES2. Para sumar estos dos números seguimos los pasos indicados en la figura 23:

```
; RESULTADO = RES1 + RES2
    movf    RES1,W
    movwf   RESULTADO   ; RESULTADO = RES1
    movf    RES2,W
    addwf   RESULTADO   ; RESULTADO = RES1 + RES2
```

Figura 23. *Código ejercicio aritmético (3).*

Y en resultado es, (Figura 24):

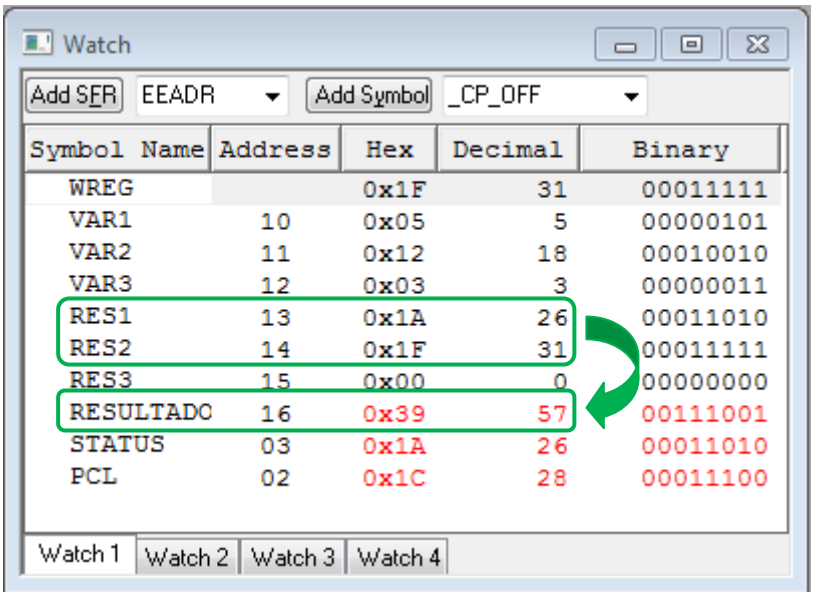


Figura 24. *Simulación del ejercicio aritmético (5).*

Por último hay que incrementar en uno el valor de resultado y decrementar en uno el valor de RES1. Se podría realizar la suma y la resta de la forma que se han realizado los ejercicios anteriores, sin embargo, la manera de **optimizar el código** es usar las instrucciones incf y decf que directamente suman y restan uno al valor de un registro, por lo que el código mas optimo sería, (Figura 25):

```
; Incrementar RESULTADO y decrementar RES1
    incf    RESULTADO   ; RESULTADO = RESULTADO + 1
    decf    RES1        ; RES1 = RES1 - 1
```

Figura 25. *Código ejercicio aritmético (4).*

La simulación resultante es la siguiente, (Figura 26):

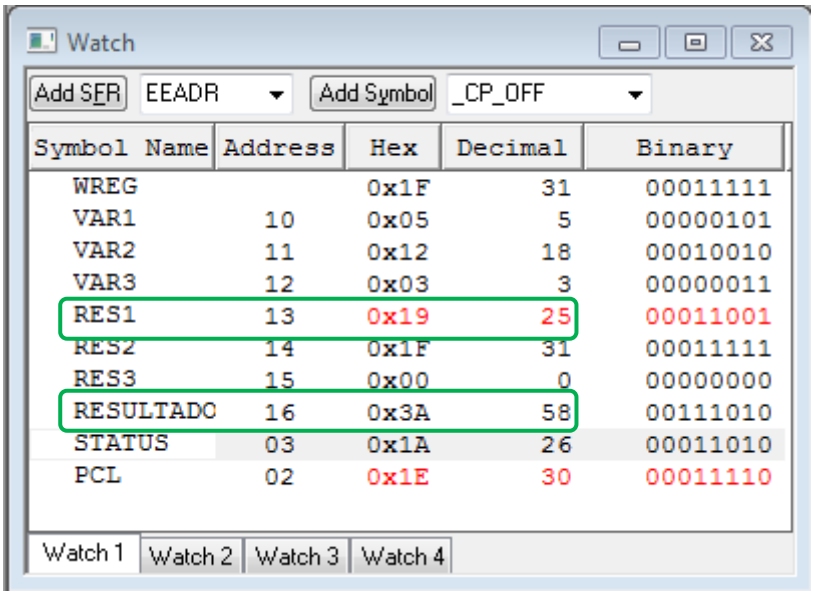


Figura 26. *Simulación del ejercicio aritmético (6).*

Lo último del ejercicio es la realización de las operaciones lógicas, la primera es la puesta a cero de los registros RES1, RES3, W. Se puede mover el valor cero al registro de trabajo y luego pasarlo a los registros RES1 y RES3, aunque el número de instrucciones en este caso concreto son las mismas que realizar tres *clears*, es mucho mas visual y más sencillo programarlo con la instrucción *clrf* o *clrw*, por ello se utiliza el siguiente código, (Figura 26):

```
;-------------------------------------------------------
;---                OPERACIONES LÓGICAS                -
;-------------------------------------------------------

; Puesta a cero de registros
    clrw
    clrf RES1
    clrf RES3
```

Figura 27. Código ejercicio aritmético (5).

Se muestran los tres registros borrados, (Figura 28):

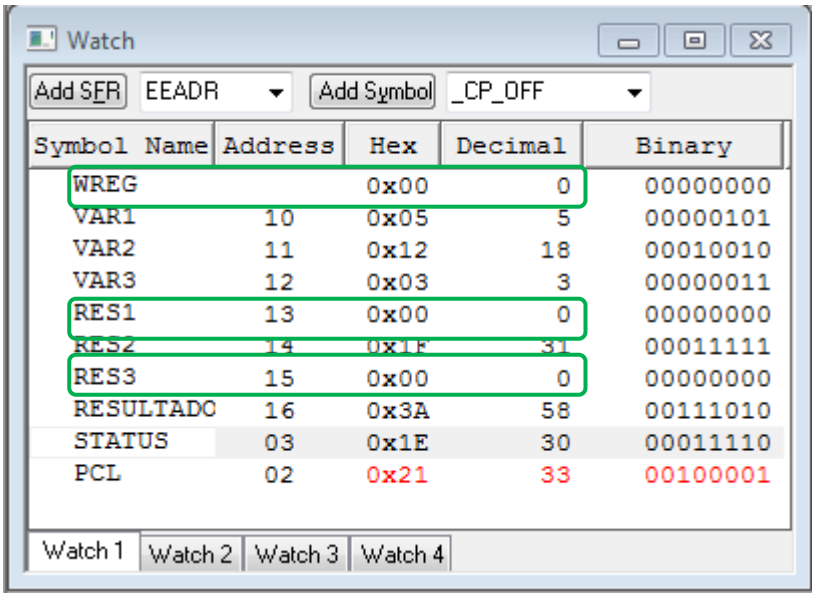


Figura 28. *Simulación del ejercicio aritmético (7).*

La siguiente operación lógica es RES1 = VAR3 AND cte3. Igual que en las operaciones aritméticas, solamente se puede realizar una operación lógica de dos números siempre que uno de ellos se encuentre en W.

Primero pasaremos un parámetro a RES1 y después el otro a W realizando el AND y almacenándolo en RES1,(Figura 29):

```
; RES1 = VAR3 & cte3
    movf    VAR3,W
    movwf   RES1        ; RES1= VAR3
    movlw   cte3
    andwf   RES1        ; RES1 = VAR3 & cte3
```

Figura 29. *Código ejercicio lógico(1).*

La operación AND da como resultado, (Figura30)

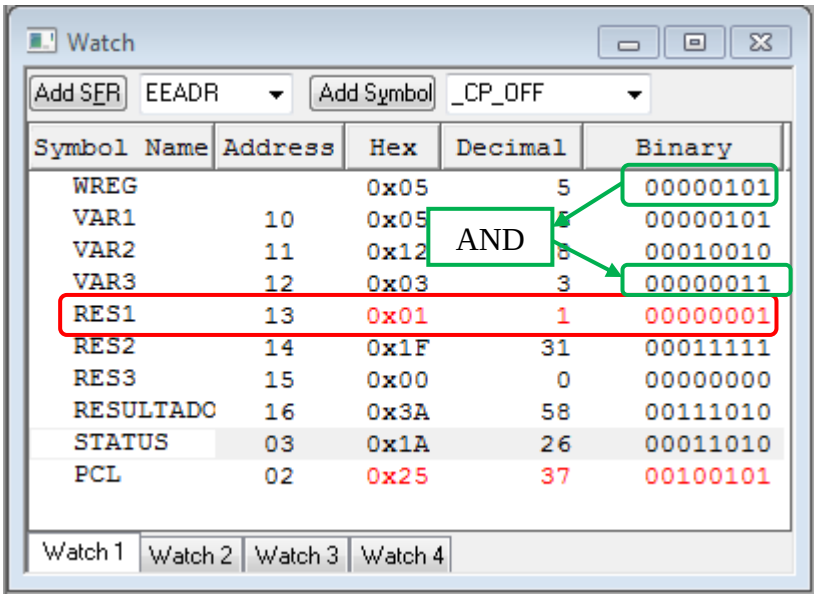


Figura 30. *Simulación ejercicio lógico(1).*

De forma análoga al anterior, para realizar la siguiente operación RES3 = RES1 OR b'00010110' se emplea el siguiente código, (Figura 31)

```
; RES3 = RES1 OR '00010110'
    movf    RES1,W
    movwf   RES3        ; RES3 = RES1
    movlw   b'00010110'
    iorwf   RES3        ; RES3 = RES1 OR '00010110'
```

Figura 31. *Código ejercicio lógico (2).*

El resultado de la operación OR se puede observar en la Figura 32.

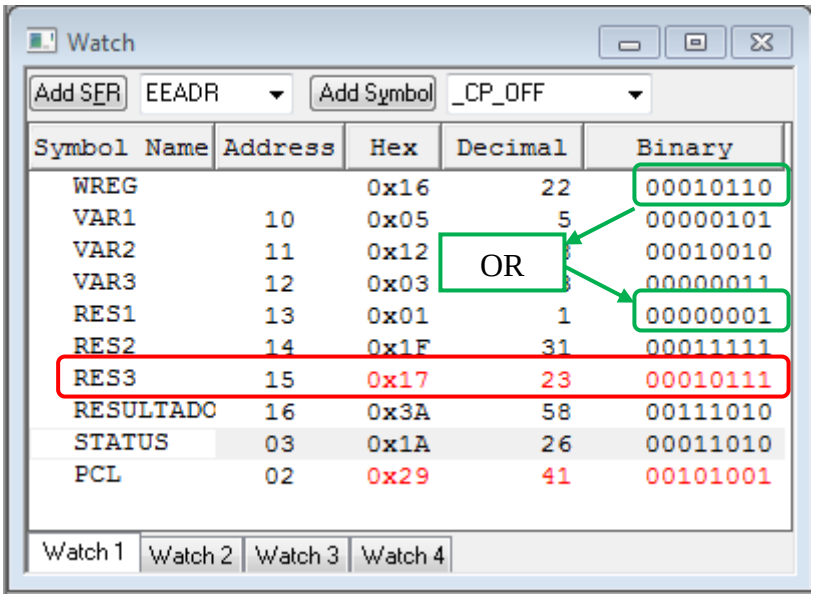


Figura 32. *Simulación ejercicio lógico(2).*

Para realizar la puesta a cero del bit 1 y puesta a uno del bit3 de RES3, la forma más sencilla de poner a cero o a uno un bit de un registro es usar las instrucciones *bcf* y *bsf* como se indica en la Figura 33:

```
; RES3 --> BIT1 = 0 Y BIT 3 =1
    bcf     RES3,1      ; Puesta a cero del bit1
    bsf     RES3,3      ; puesta a uno del bit3
```

Figura 33. *Código ejercicio lógico(3).*

Se observa el cambio del bit 1 y del bit 3 a continuación, (Figura 34):

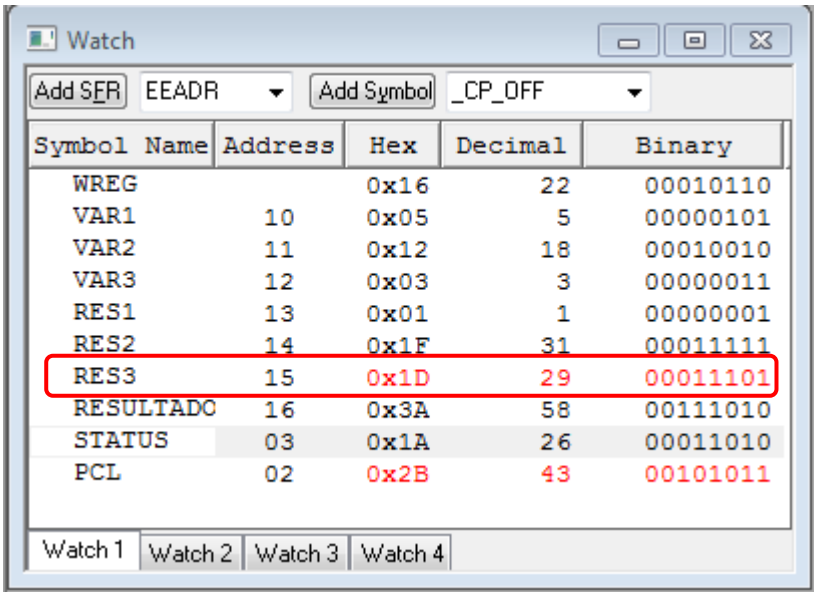


Figura 34. *Simulación ejercicio lógico(3).*

Por último para activar el flag de cero al realizar una operación con el número '10000000', existen numerosas formas, basta con que el resultado de una operación sea cero. No todas las instrucciones afectan al flag de cero (ver anejo de instrucciones), por ejemplo la puesta a cero de un bit no afecta.

Para este ejercicio se ha escogido realizar **la operación AND** del valor anterior con el valor b'00001111' (Figura 35).

```
; Operación a '10000000'    que ponga a uno Z
    movlw   b'10000000'
    movwf   RES2
    movlw   b'000011110'
    andwf   RES2,W      ; realizon un and y almaceno en W
```

Figura 35. *Código ejercicio lógico(4).*

El flag de cero (bit2) cambia a uno debido a la operación, (Figura 36).

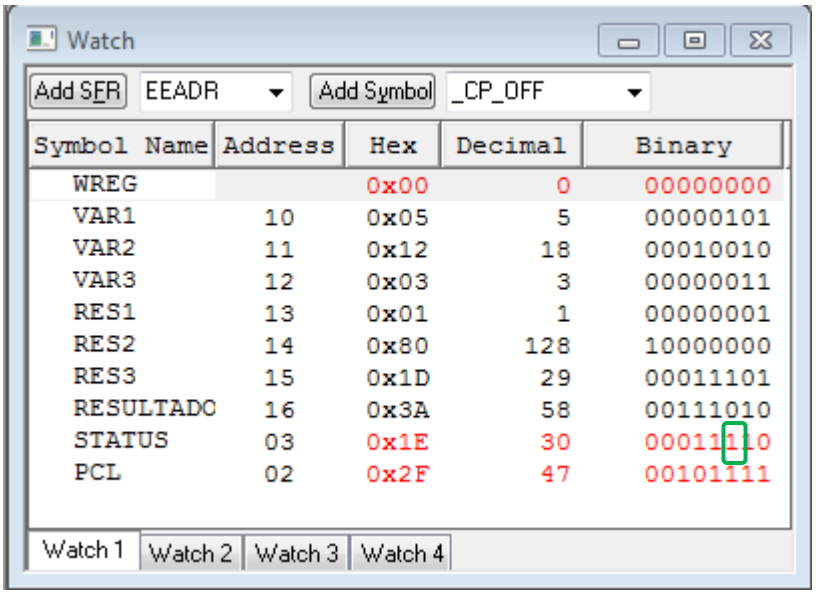


Figura 36. *Simulación ejercicio lógico(4).*

1.1.5.4.1.6.- Ejercicio de ampliación

1.1.5.4.1.6.1- Planteamiento y diagrama de flujo

En esta primera práctica, simplemente se van siguiendo las instrucciones una a continuación de otra, sin realizar ningún salto ni llamada, siguiendo los pasos indicados en el enunciado, por lo que no se considera necesaria la explicación del planteamiento ni la realización de un diagrama de flujo.

1.1.5.4.1.6.2- Programación y simulación

Una vez declaradas todas las constantes y variables, lo primero es realizar la parte aritmética, para ello se trata de operar RES1=2(a-1)+2(b+1)-c-d. El código empleado en la parte aritmética se muestra en la figura 37:

```
;-----------------------------------------------------------
;---                  OPERACIONES ARITMÉTICAS              -
;-----------------------------------------------------------

; RES1=2(a-1)+2(b+1)-c-d

    movlw   cte_a-1 ;   a=3
    movwf   RES1    ;   RES1 = a-1
    rlf     RES1    ;   RES1 = 2*(a-1)
    movlw   cte_b+1 ;   b=2
    addwf   RES1    ;   RES1 = 2*(a-1) + b+1
    addwf   RES1    ;   RES1 = 2*(a-1) + 2*(b+1)
    movlw   cte_c   ;   c=4
    subwf   RES1    ;   RES1 = 2*(a-1) + 2*(b+1)-c
    movlw   cte_d   ;   d=2
    subwf   RES1    ;   RES1 = 2*(a-1) + 2*(b+1)-c-d
```

Figura 37. *Código ejercicio ampliación, artmética.*

Se puede observar el uso del desplazamiento a izquierdas del registro para la multiplicación por dos del valor a-1, sin embargo para b+1 se ha optado por realizar una suma dos veces, esto se debe *al tener que usar una variable temporal para el desplazamiento ya que no se puede realizar sobre el registro de trabajo.*

Se observa como el desplazamiento del registro RES1 a izquierdas multiplica el valor por dos, (Figura 38)

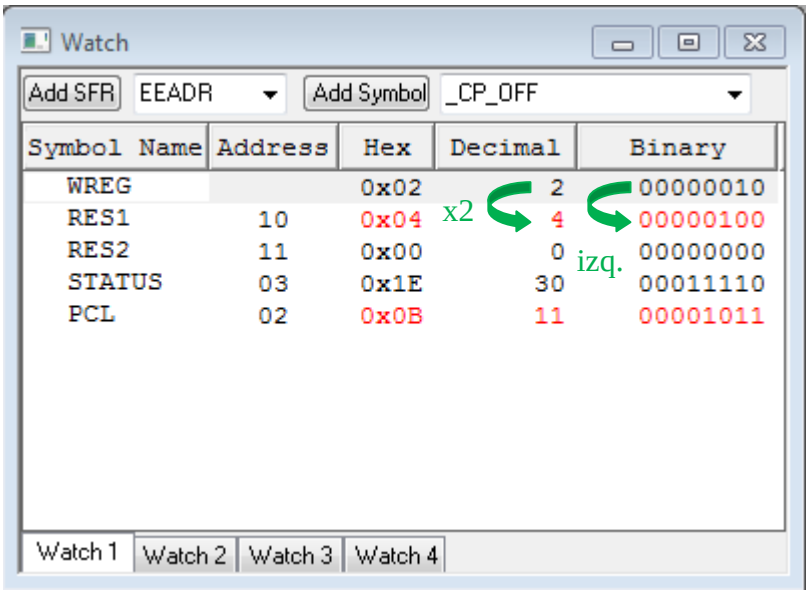


Figura 38. *Simulación ejercicio ampliación, aritmética (1)*

Después sumamos dos veces el valor de b+1,(Figura 39):

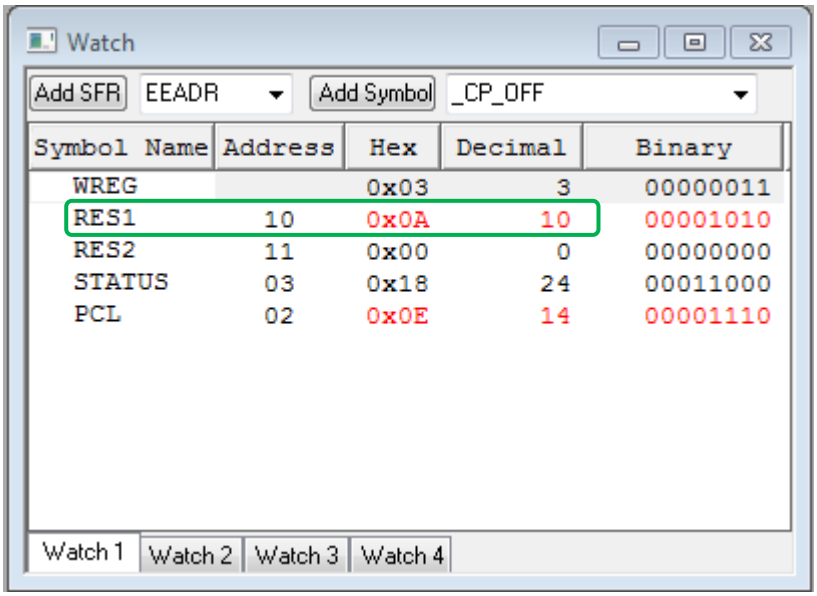


Figura 39. *Simulación ejercicio ampliación, aritmética (2).*

Para finalmente restarle los valores de c y d, (Figura 40)

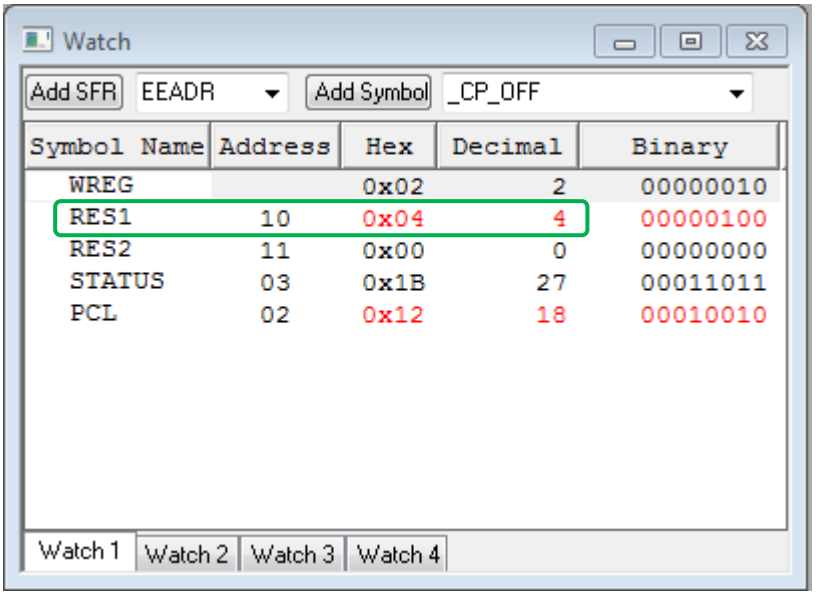


Figura 40. *Simulación ejercicio ampliación, aritmética (3).*

Las operaciones lógicas a realizar se muestran en el siguiente código, (Figura 41):

```
;-------------------------------------------------------
;---                OPERACIONES LÓGICAS                -
;-------------------------------------------------------

;  b'11011011' poner a cero los bits 0 y 4

    movlw   b'11011011'
    movwf   RES2
    movlw   b'11101110' ; Máscara AND
    andwf   RES2

;   b'11011011' intercambio de estado de b0-b3
    movlw   b'00001111' ; Máscara XOR
    xorwf   RES2

; Intercambio de parte alta y baja
    swapf   RES2
```

Figura 41. *Código ejercicio de ampliación lógico.*

La primera operación lógica para poner a cero el bit 0 y el bit 4 se usa una máscara and, dejando el resto de números en su estado inicial, (Figura 42):

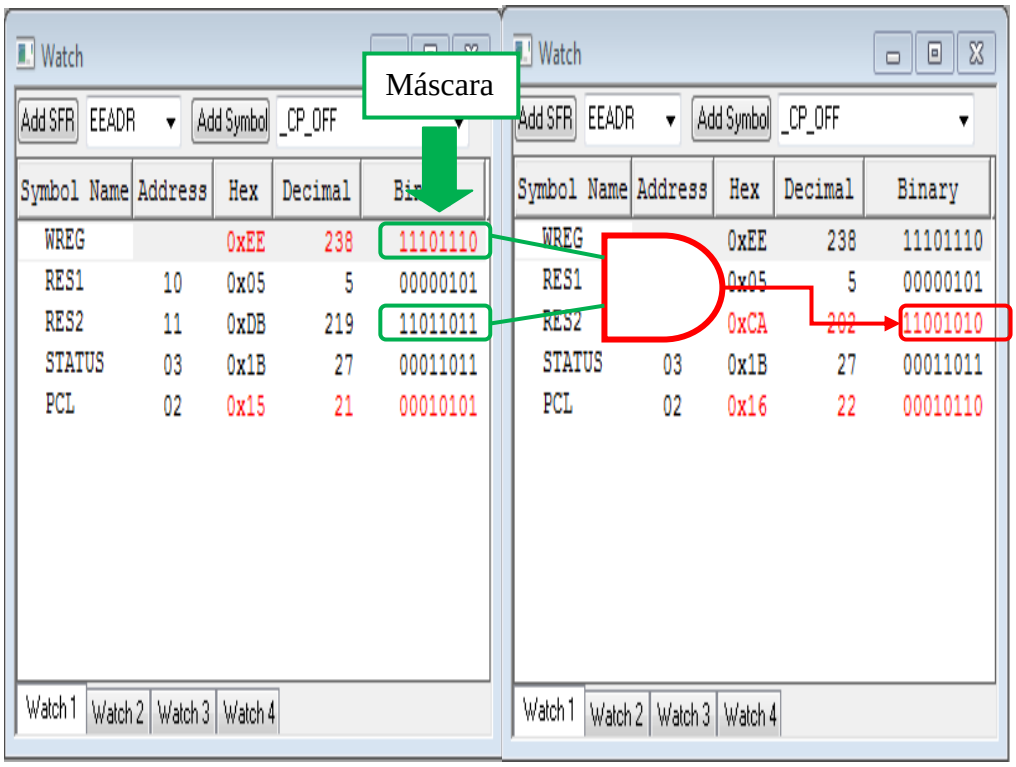


Figura 42. *Simulación ejercicio de ampliación lógico (1).*

Después se cambia el estado los 4 bits menos significativos,(Figura 43):

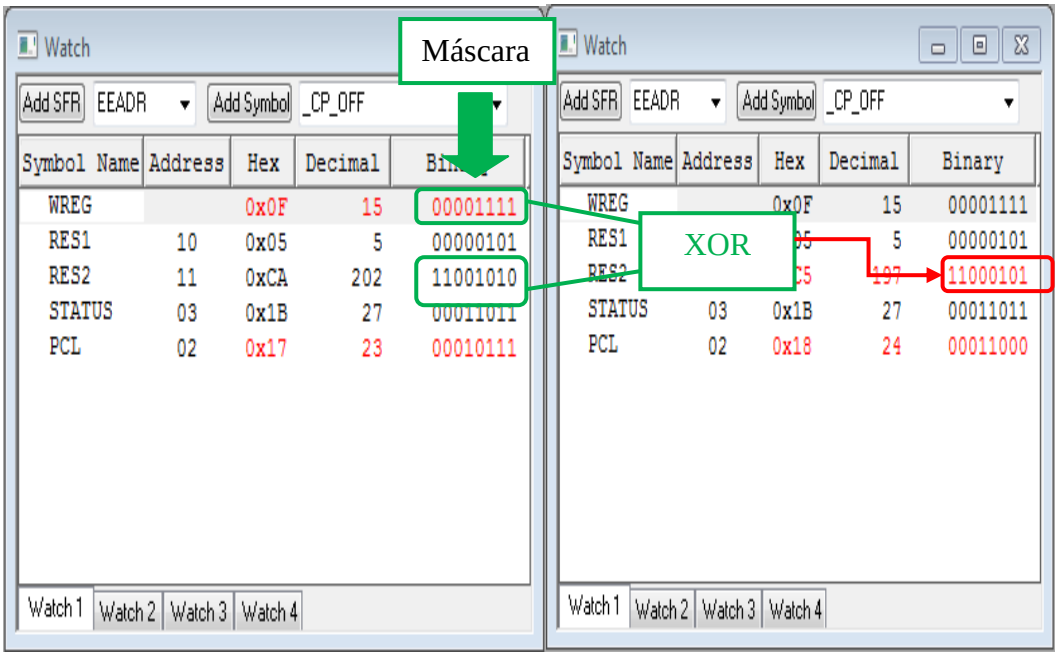


Figura 43. *Simulación ejercicio de ampliación lógico (2).*

Y por último se intercambia la parte alta con la baja, (Figura 44):

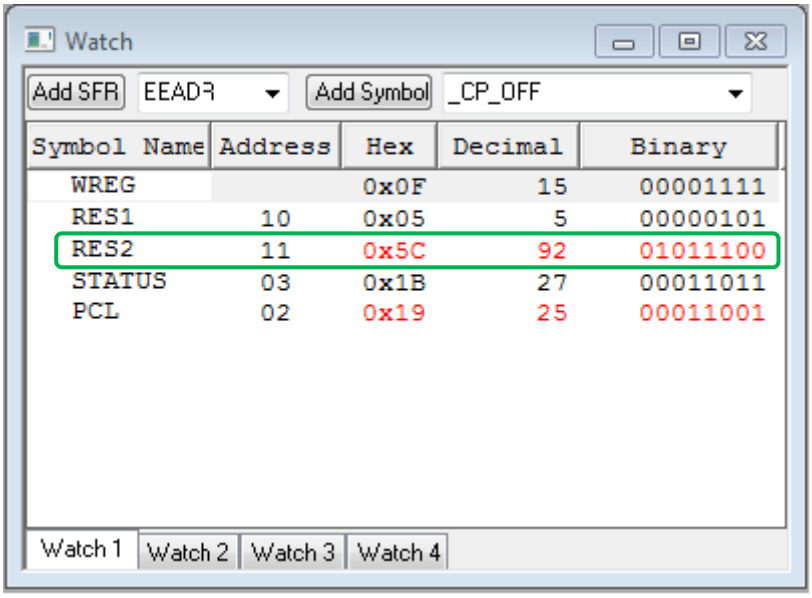


Figura 44. *Simulación ejercicio de ampliación lógico (3).*

En los ejercicios sobre los flags se empleó el siguiente código, (Figura 45):

```
;-----------------------------------------------------
;---                  FLAGS                          -
;-----------------------------------------------------


; Operación C=1 y DC =0

    movlw   0x80        ; b'10000000'
    movwf   RES1
    addwf   RES1

; Operación C=0 y DC = 1

    movlw   0x0F        ; b'00001111'
    movwf   RES1
    addwf   RES1

; Usando decremento activar Z

    movlw   0x01
    movwf   RES1
    decf    RES1
```

Figura 45. *Código ejercicio de ampliación flags.*

Se observa en la primera operación como cambia el flag de C pero no DC, también cambia el flag Z, ya que el resultado de la operación es cero, (Figura 46):

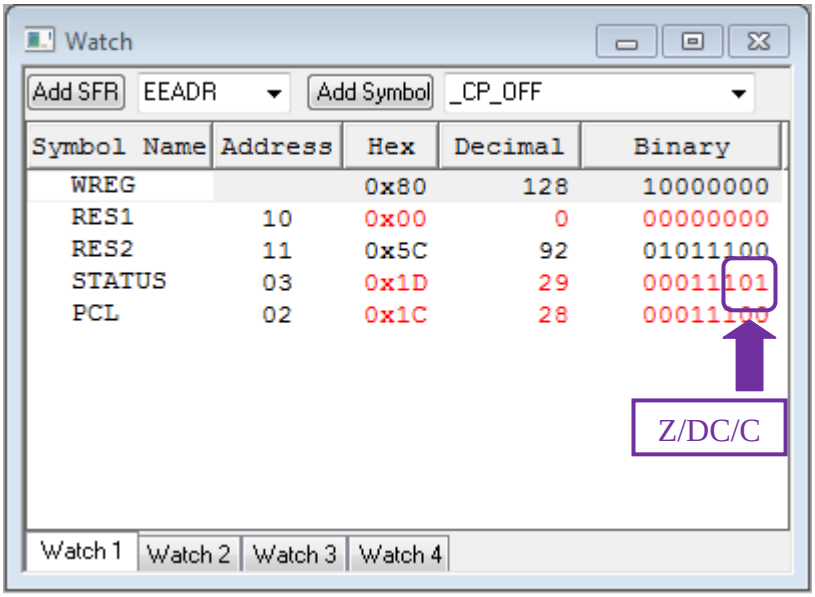


Figura 46. *Simulación ejercicio de ampliación flags (1).*

En el segundo caso cambia DC pero no C, (Figura 47):

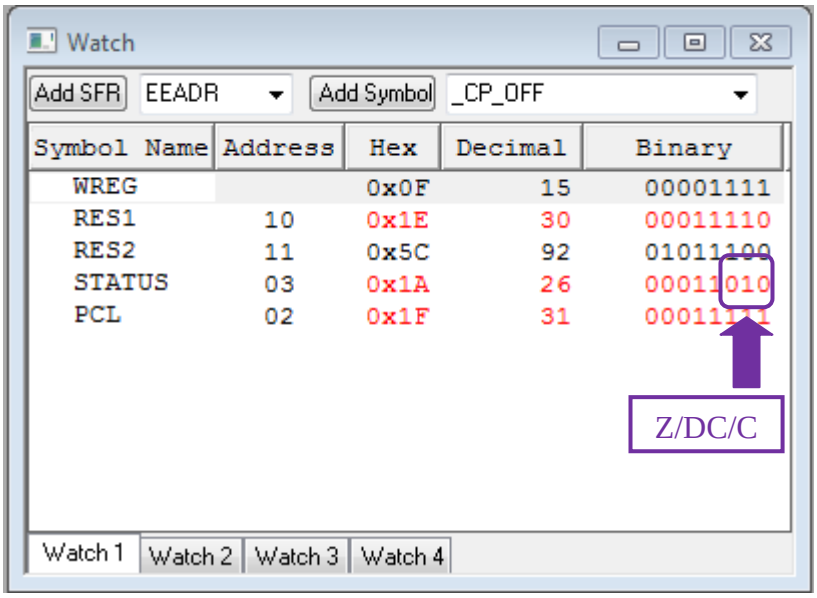


Figura 47. *Simulación ejercicio de ampliación flags(2).*

Y por último con la función decremento conseguimos cambiar el flag de Z, (Figura 46).

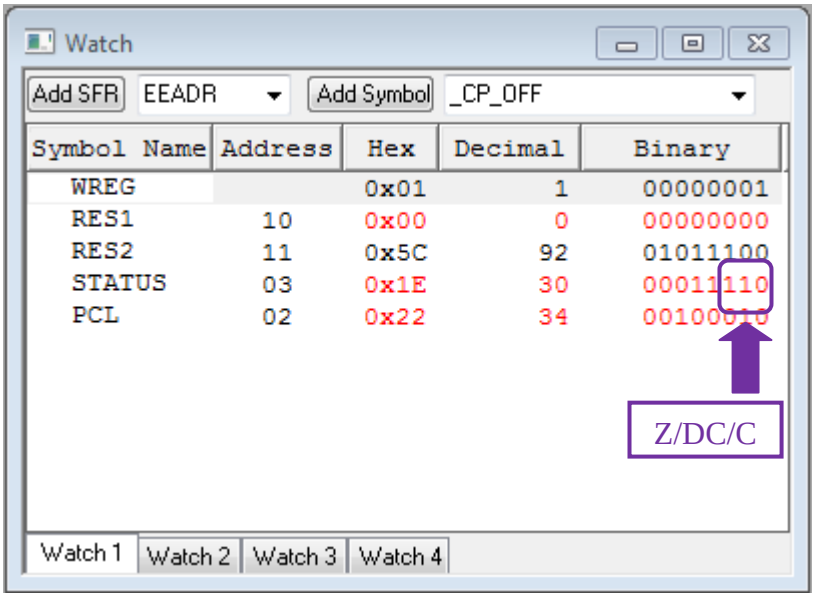


Figura 48. *Simulación ejercicio de ampliación flags(3).*

1.1.5.4.1.7.- Programas propuestos

Desarrollar un código que realice las siguientes operaciones aritméticas con a=5, b=10, c=12, d=2:

- *a+b-c-d*
- *(a+1)·2+(b+2)·2*
- *0.5c+0.5d-a*

Desarrollar un código que realice las siguientes operaciones lógicas, al valor b'11111111':

- *Intercambiar parte alta con parte baja.*
- *Máscara que ponga los bit7 y bit 6 a cero.*
- *Borrar el bit cero.*
- *Cambiar de estado los bit 7,6,5 y 4.*

1.1.5.4.2.- Practica 2: Bucles, Subrutinas e instrucciones de control

1.1.5.4.2.1.- Objetivos

- ✓ Estudio con diagramas de flujo.
- ✓ Manejo de las instrucciones de control del flujo del programa.
- ✓ Comprensión de las subrutinas y bucles.

1.1.5.4.2.2.- Enunciado

Realizar un programa que almacene en memoria el mayor valor de tres datos dados. Para ello se creará una subrutina capaz de distinguir el valor más alto de dos números.

- ✓ Realizar la simulación para los datos a1=10 , a2=5 a3 =25.
- ✓ Realizar la simulación para los datos a1=15, a2=125, a3=4.
- ✓ Realizar la simulación para los datos a1=56, a2=1, a3=0.

1.1.5.4.2.3.- Ejercicio de ampliación

Realizar un programa que multiplique dos números, para ello se usará dos registros, uno será la parte baja del dato y el otro la parte alta.

- ➢ Realizar la simulación para los datos a1 = 5, a2=30.
- ➢ Realizar la simulación para los datos a1=200, a2=3.

1.1.5.4.2.4.- Contenido teórico

1.1.5.4.2.4.1 - La instrucción *goto*

Esta instrucción realiza un *salto incondicional* a la posición de la memoria de programa indicada en el código, para indicar dicha posición se realiza mediante etiquetas (lo más habitual) o directamente poniendo el valor de la dirección de la memoria de programa, como se indica en la Figura 49.

```
        goto    Etiqueta
        ; ...

Etiqueta

        ;...
        goto    0x100
        ; ...

        org 0x100

        ; ...
```

Figura 49. *Instrucción GOTO*

Cuando se ejecuta la instrucción carga en el contador de programa (PC y PCLATH) el valor de la dirección de la siguiente instrucción a ejecutar (ver práctica 1).

Uno de sus principales usos en ensamblador es junto con la instrucción de salto condicional, para la realización de **bucles de control**.

1.1.5.4.2.4.2.- Subrutinas

Son pequeños programas dentro del programa principal, usan las instrucciones:

- *call*

Es la instrucción de llamada a la subrutina. Esta instrucción realiza un *salto incondicional* a la posición de memoria indicada en el código, al igual que la instrucción goto se le indica dicha dirección de memoria, sin embargo, *carga en la pila el valor de la dirección de memoria de la instrucción que se encuentra a continuación del call , permitiendo retornar al programa una vez finalizada la subrutina*.

- *return*

Realiza el *retorno de subrutina*, para ello coloca el valor de la pila en el contador de programa, es decir, retorna a la siguiente instrucción situada después de la llamada a la subrutina.

- *retlw*

Realiza el retorno de subrutina, para ello coloca el valor de la pila en el contador de programa, a diferencia de la instrucción return, también copia el valor indicado en el registro de trabajo.

Uno de los principales usos de la subrutinas es la creación de funciones, si se necesita realizar cierta operación muchas veces en el programa, lo más sencillo es crear una subrutina que realice la operación, y cada vez que se necesite realizarla se llamara a la subrutina.(Figura 50):

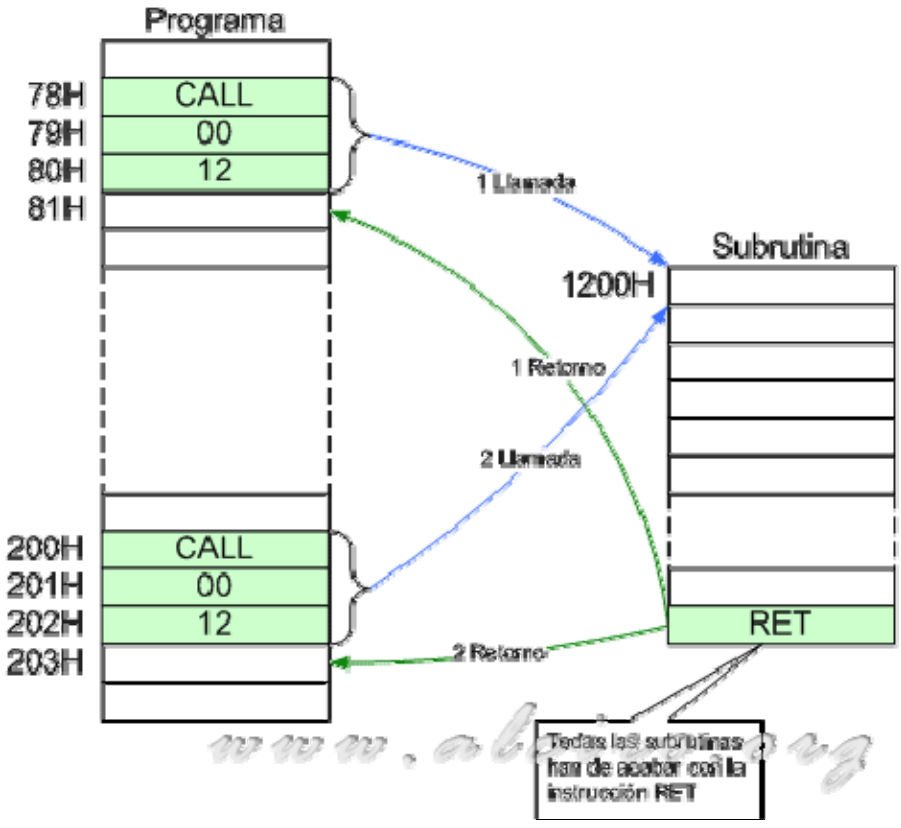


Figura 50.- *Funcionamiento de las subrutinas.* [8]

- Subrutinas dentro de subrutinas:

Es posible llamar a una subrutina dentro de otra subrutina, la pila es capaz de almacenar hasta **ocho direcciones de memoria**, por lo que está concatenación se podría realizar hasta ocho veces.

EL funcionamiento de la pila es tipo **LIFO**, es decir, la última dirección que se almacenó, será la primera dirección a la que se volverá. (Figura 51):

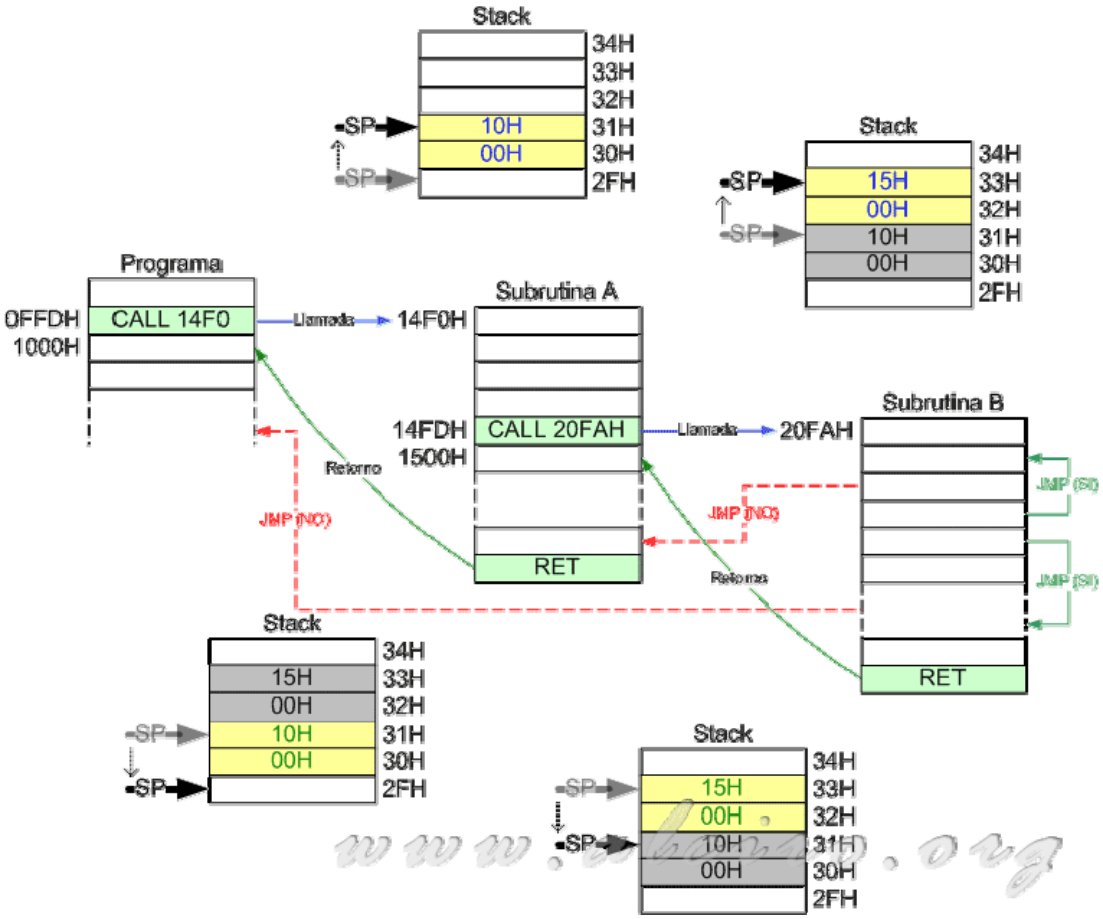


Figura 51. *Anidación de subrutinas.* [8]

Se debe tener cuidado con las variables usadas en dichas subrutinas, ya que se podría estar usando una misma variable para dos operaciones diferentes, falseando así el resultado de la operación.

1.1.5.4.2.4.3.- Saltos condicionales

Se requiere el cumplimiento de una condición para realizar el salto, **solamente realizan el salto de una instrucción**. Su funcionamiento es sumar uno al contador de programa si se cumple la condición. Las 4 instrucciones de salto son las siguientes:

- *btfsc* *REG*,5 -> salta si el bit 5 del registro REG está a cero
- *btfss* *REG*,0 -> salta si el bit 0 del registro REG está a uno
- *decfsz* *REG* -> Decremento REG, si es cero salta (si Z=1)
- *incfsz* *REG* -> Incremento REG, si es cero salta (si Z=1)

1.1.5.4.2.4.4.- Bucles

Un bucle es un conjunto de instrucciones que se van a repetir un número n determinado de veces, cada repetición del bucle se denomina *iteración*. Uno de sus principales aplicaciones en ensamblador es la creación de temporizadores y contadores como se explica en la práctica 5.

Existen muchas maneras de crear un bucle, una forma de implementarlo es la siguiente,(Figura 52):

```
    movlw   .10
    movwf   CONTADOR

Inicio_bucle
    ; ----
    ; ----
    decfsz  CONTADOR
    goto Inicio_bucle
    ; -----
    ; -----
```

Figura 52. *Estructura del bucle.*

La variable *CONTADOR* será la que controle el bucle. Antes de empezar el bucle es necesario asignarla un valor, si se diera ese valor dentro del bucle, en cada iteración volvería a su valor inicial lo que crearía un bucle infinito.

Al **finalizar el bucle se decrementa la variable de control**, si el valor es distinto de cero no salta la siguiente instrucción y realiza el *'goto Inicio_bucle'* pero si ha llegado a cero, la instrucción '*decfsz*' salta la siguiente instrucción (goto Inicio_bucle) dando por finalizado el bucle.

No todos los bucles terminan cuando se realiza un determinado número de veces, también pueden realizarse hasta que se cumpla una condición determinada, por ejemplo, un bucle que se ejecuta mientras el puerto A no tenga el valor de 1.

1.1.5.4.2.5.- Ejercicio básico

1.1.5.4.2.5.1.- Planteamiento

En este ejercicio se trata de obtener el mayor valor de tres números. Para realizarlo se va a crear una función que compare dos números, a la que se llamará dos veces y así obtener el mayor de los tres datos.

En el enunciado, se indica guardar en la memoria el valor del mayor número, por lo que es indiferente si algún numero es igual que otro al seguirse almacenado el valor del mayor número.

La forma de comparar dos números es usar la resta de dos números, si el valor de la resta a-b es positivo el numero a será mayor (o igual se es cero) que b y si la resta es negativo el mayor será b. [1]

La operación resta , en realidad se trata de una suma de dos números a+b ' siendo b' el complemento a2 de b , si hay desbordamiento en esa suma, C tiene el valor de uno (el resultado es negativo) y si no es positivo.(Tabla 6)

Tabla 6. *Resta de dos números.*

a-b	Carry	Significado
b>a	C=1	Negativo
a≥b	C=0	Positivo

1.1.5.4.2.5.2.- Diagrama de flujo

En el programa principal DATOA y DATOB son los valores a calcular su máximo, la función Máximo devuelve el valor mayor de esos dos número en W. Se realizan dos llamadas a la función con los tres valores obteniendo el valor máximo.(Figura 53):

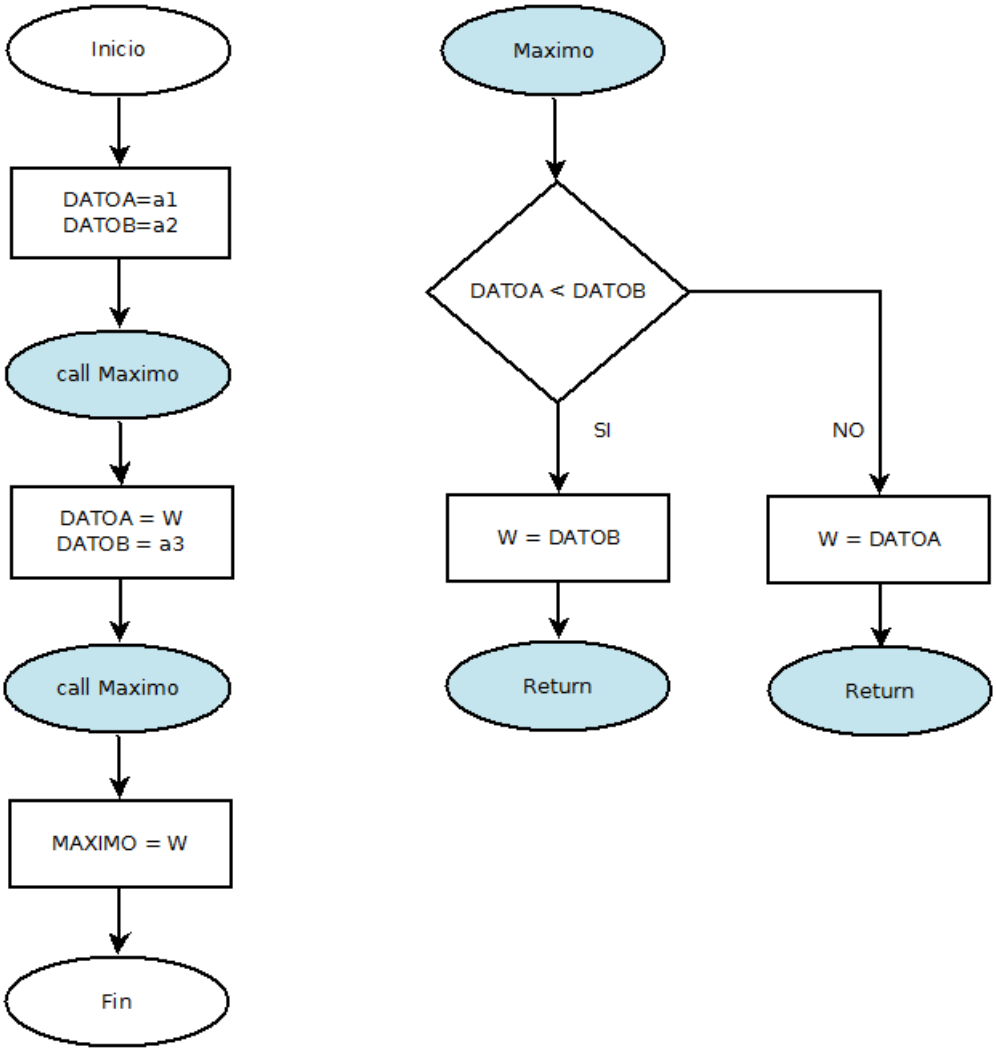


Figura 53. *Máximo de tres números.*

La subrutina máximo compara los valores almacenados en DATOA y DATOB, copiando en el registro de trabajo el mayor de los 3.(Figura 53).

1.1.5.4.2.5.3.- Programación y simulación

Al no indicar nada el enunciado, se almacena solo el mayor dato. (Figura 54):

```
;-----------------------------------------------------
;---    DECLARACIÓN DE CTES. Y VARIABLE               -
;-----------------------------------------------------

    #define a1 .15          ; Datos a comparar
    #define a2 .125
    #define a3 .4

    MAX     equ 0x13        ; Registro del valor máximo
    DATOA   equ 0x14        ; Variables auxiliares para la subrutina
    DATOB   equ 0x15
```

Figura 54. *Registros y constantes del programa.*

- *a1,a2,a3* son los datos a introducir.
- *DATOA y DATOB:* parámetros de entrada a la función máximo.
- *MAX* es el registro donde se almacenará el mayor número de los tres.

1.1.5.4.2.5.3.1- Programa principal

- *Código*

```
Inicio          ; Etiqueta de comienzo de programa

;-----------------------------------------------------------
;---                PROGRAMA PRINCIPAL                    -
;-----------------------------------------------------------

    clrw            ; Reset del registro de trabajo

    movlw   a1      ; Parámetros a pasar al bucle
    movwf   DATOA
    movlw   a2
    movwf   DATOB

    call Maximo     ; W = MAXIMO (a1,a2)

    movwf   DATOA   ; DATOA = MAXIMO (a1,a2)
    movlw   a3
    movwf   DATOB   ; DATOB = a3

    call Maximo     ; W = MAXIMO(MAXIMO(a1,a2),a3))
    movwf MAX       ; Almaceno el valor máximo

Fin goto Inicio
```

Figura 55. *Código del programa principal.*

Al comenzar el programa se realiza un clear del registro de trabajo (ver práctica 1), y se guardan los valores de los dos primeros números a comparar en DATOA y DATOB , que son los *parámetros a pasar a la subrutina máximo* y se llama a la función.

Esta función como se mostrará más adelante devuelve en w el máximo de dos números, por lo que ese máximo lo copiamos en DATOA como un nuevo parámetro y DATAB con el valor del tercer número.

Se vuelve a llamar a la función escribiendo en el registro de trabajo *el valor máximo de los tres números que se almacenará en el registro MAX*.(Figura 55)

- *Simulación*

Los valores usados en esta simulación son a1=10 , a2=5 a3 =25, el programa comienza en la dirección 0x05 y los valores iniciales de los registros más importantes son, (Figura 56)

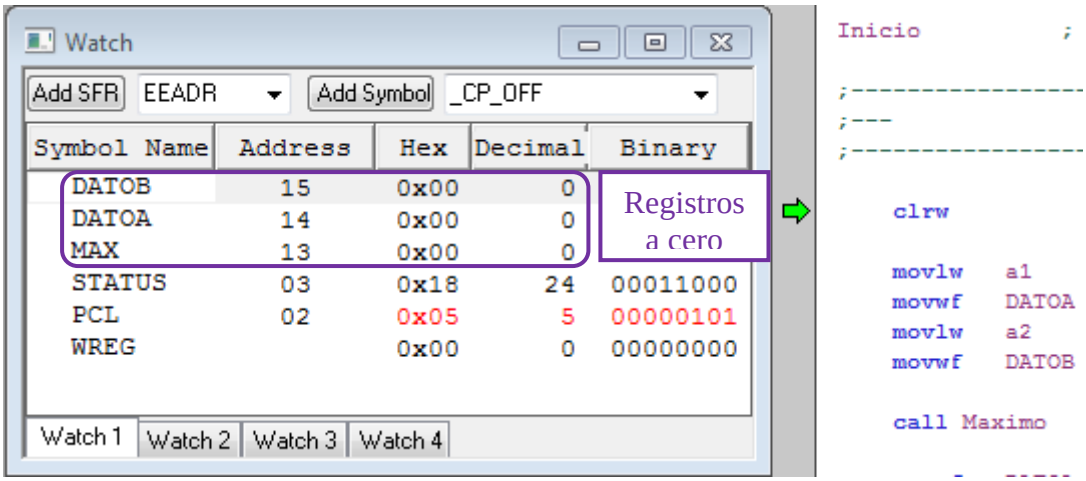


Figura 56. *Valores iniciales de los registros.*

Cargamos los parámetros de los dos primeros números a comparar antes de entrar a la función, en los registros DATOA y DATOB, (Figura 57):

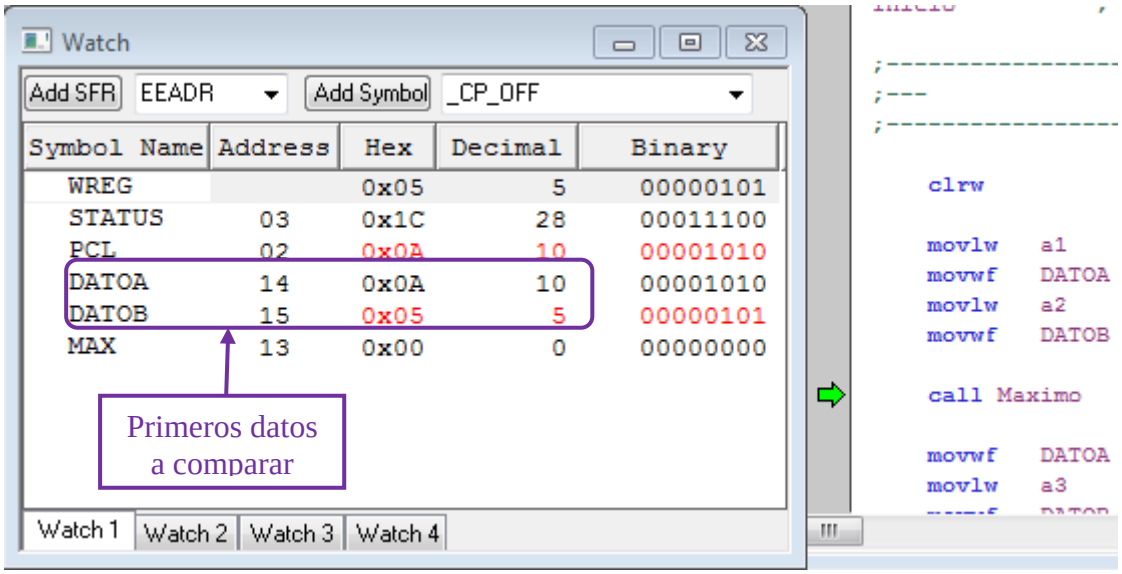


Figura 57. *Parámetros de entrada a la subrutina.*

La función máximo se analizará más adelante, ahora simplemente simulamos el resultado de la llamada, se observa como el máximo de 10 y 5 queda almacenado en W,(Figura 58) y dentro de la función nos *ha modificado DATOA, pero no DATOB*, en este programa nos da igual que nos lo modifique al solo interesar el valor máximo, Si se quisiera conservar por el motivo que fuera DATOA se debería que programar la función de otra forma.

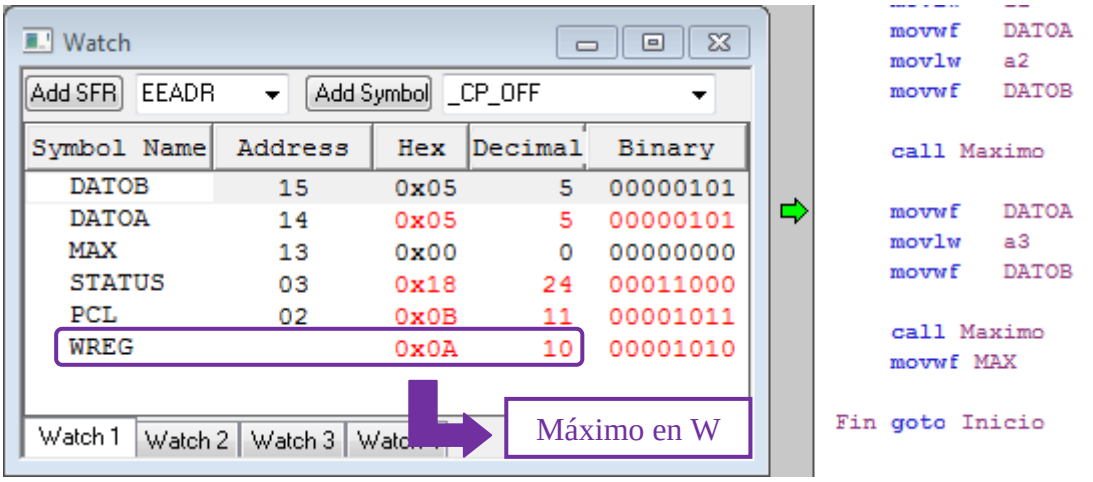


Figura 58. *Salida de la subrutina.*

A continuación se vuelve a llamar a la función usando como parámetros el máximo anterior y el tercer número, antes de llamar a la subrutina los registros están de la siguiente forma, (Figura 59):

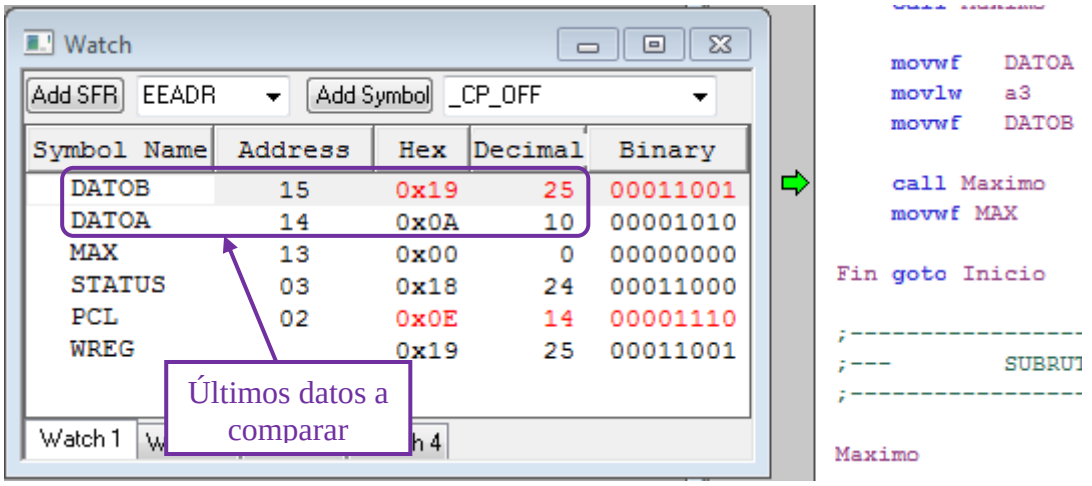


Figura 59. *Parámetros de entrada, segunda llamada.*

Y el resultado de la subrutina es el siguiente, (Figura 60)

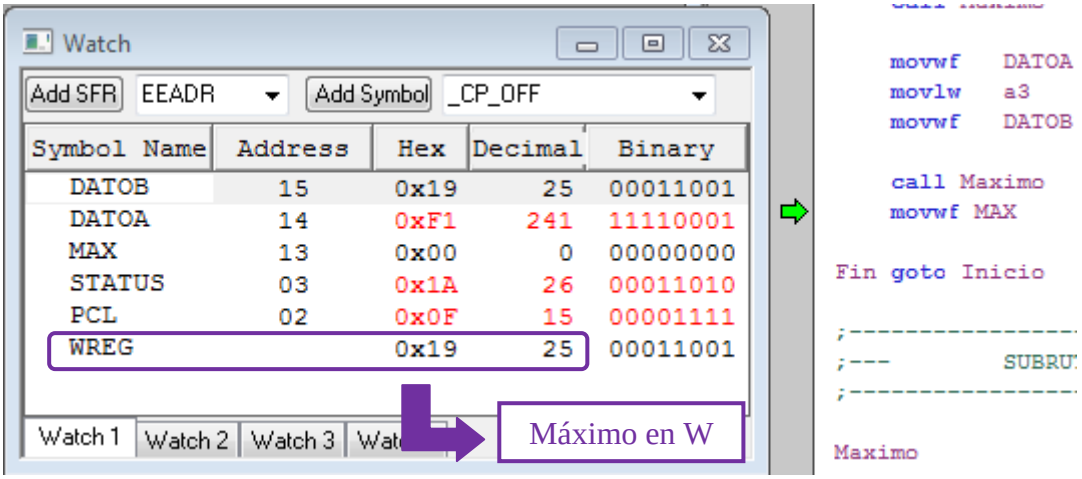


Figura 60. *Salida de la subrutina, segunda llamada.*

Solo falta la ultima instrucción de almacenamiento en el registro máximo, (Figura 61):

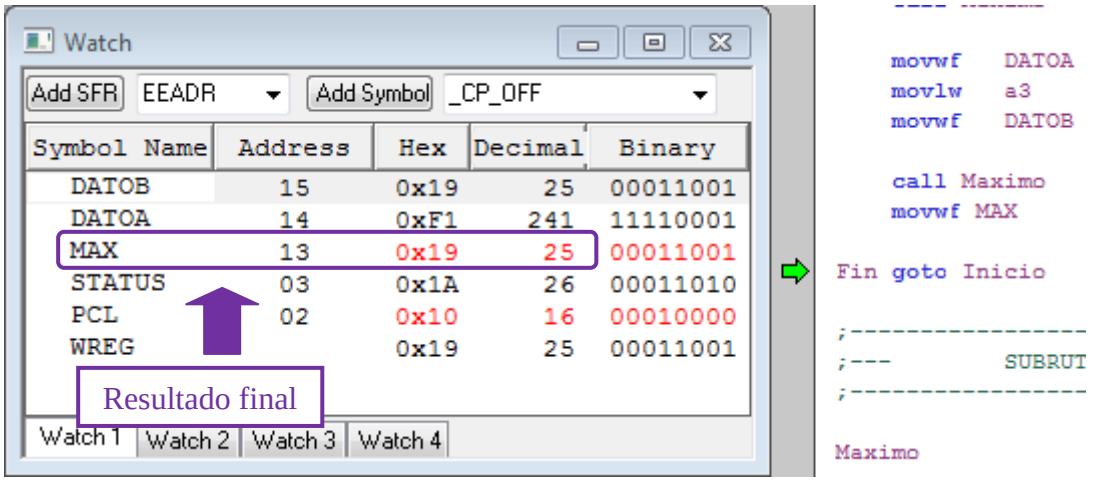


Figura 61. *Máximo de tres números.*

A continuación se muestran los tres casos del enunciando(Figura 62, 63 y 64)

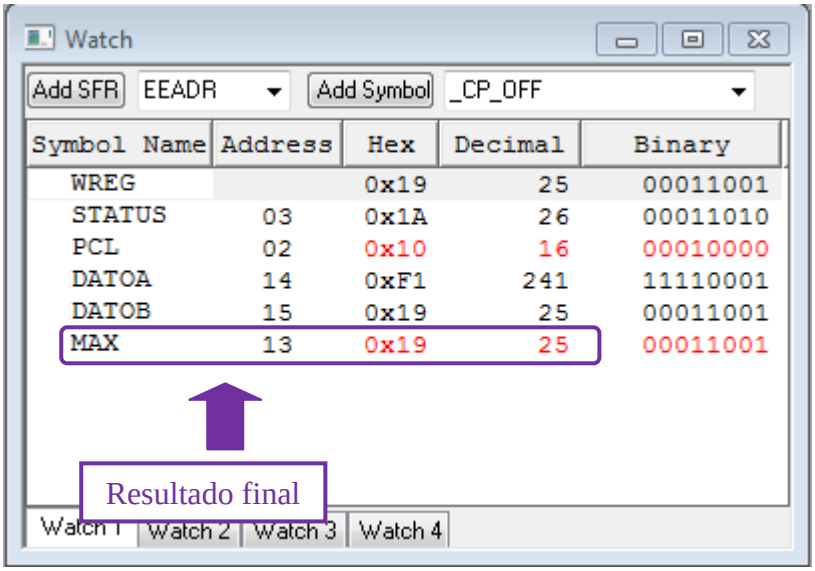


Figura 62. *Resolución del primer caso.*

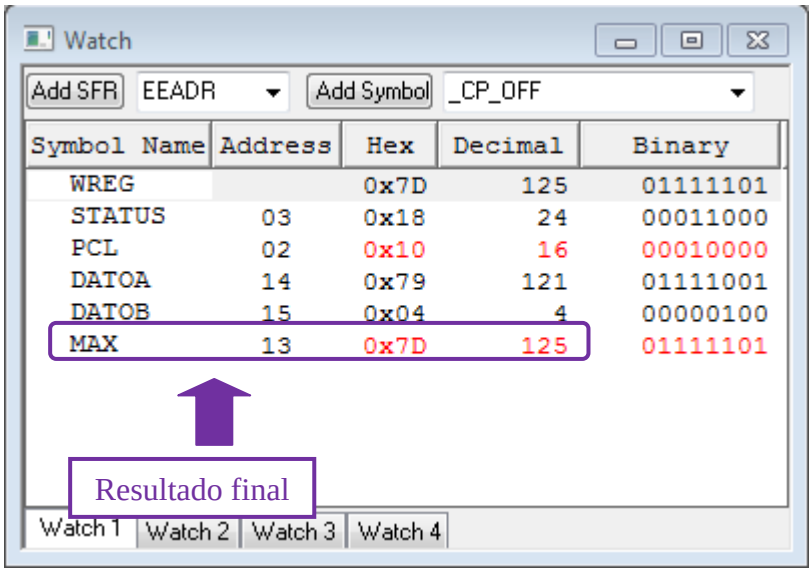


Figura 63. *Resolución del segundo caso.*

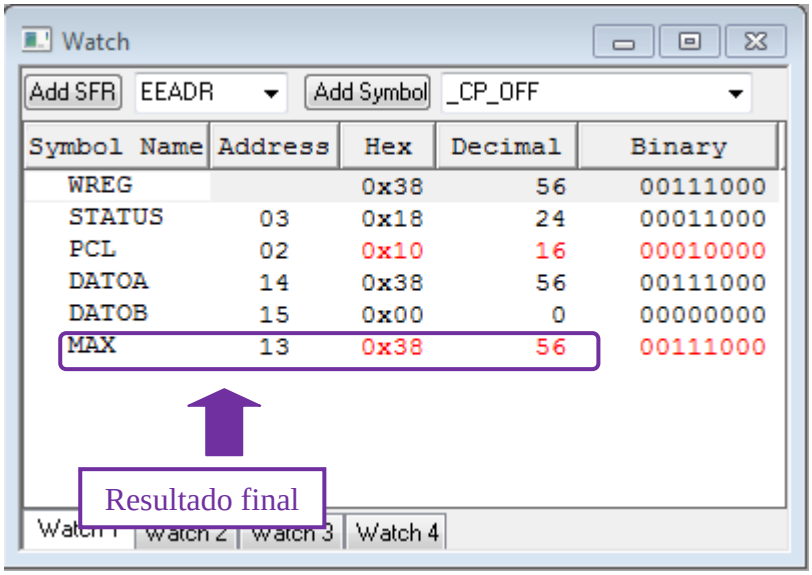


Figura 64. *Resolución del tercer caso.*

1.1.5.4.2.5.3.2.- Subrutina máximo

↳ *Código*

```
;-------------------------------------------------------
;---        SUBRUTINA COMPARA    W=MAX(DATOA,DATOB)      -
;-------------------------------------------------------

Maximo

    movf    DATOB,W         ; W = DATOB
    subwf   DATOA           ; DATOA' = DATOA - DATOB
    btfss   STATUS,C        ; Si la resta es negativa salta
    return                  ; b>a W=DATOB
    addwf   DATOA,W         ; a>=b W=DATOB + DATOA' = DATOA
    return
```

Figura 65. *Subrutina Máximo.*

Cuando se entra en la subrutina los registros *DATOA y DATOB contienen los valores que van a ser comparados*, lo primero que realiza es la copia del valor de DATOB al W para realizar la resta DATOA - W. Si el resultado es negativo se activa el flag C, por lo que testeamos con la función btfss ese flag. (Figura 65)

Si el *flag está a uno*, significa que el DATOB es el mayor y como ya se encuentra en W retorna al programa principal, si por el contrario DATOA es el mayor se salta la instrucción de retorno y sumamos a W el nuevo valor de DATOA para después retornar (2):

DATOA' =DATOA -W = DATOA - DATOB (1)

W= DATOB + DATOA' = DATOB + DATOA - DATOB = DATOA (2)

↳ *Simulación*

Antes de la primera llamada, DATOA y DATOB tiene los valores de los dos primeros números, el puntero apunta a la siguiente instrucción (call Máximo) 0x0Ah,(Figura 66)

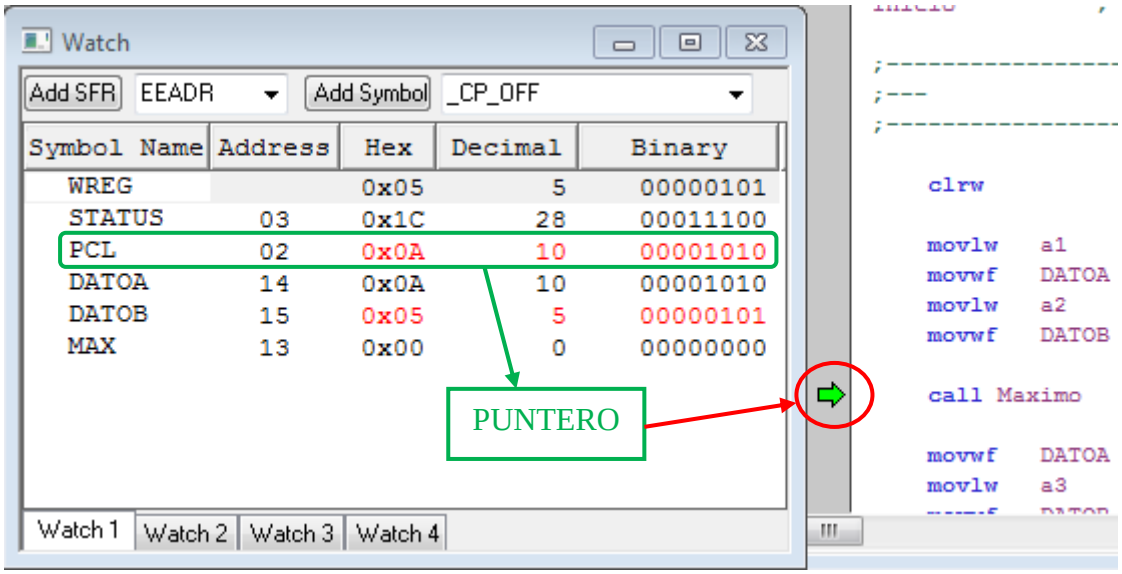


Figura 66. *Posición del PC antes de entrar en la subrutina.*

Antes de llamar a la subrutina, se aprecia en la Figura 67, como se encuentra en el nivel cero, correspondiente con el nivel del programa principal.

El resto de *stack* se encuentra sin dirección de retorno, al no haber realizado llamada alguna a subrutinas y/o interrupciones.

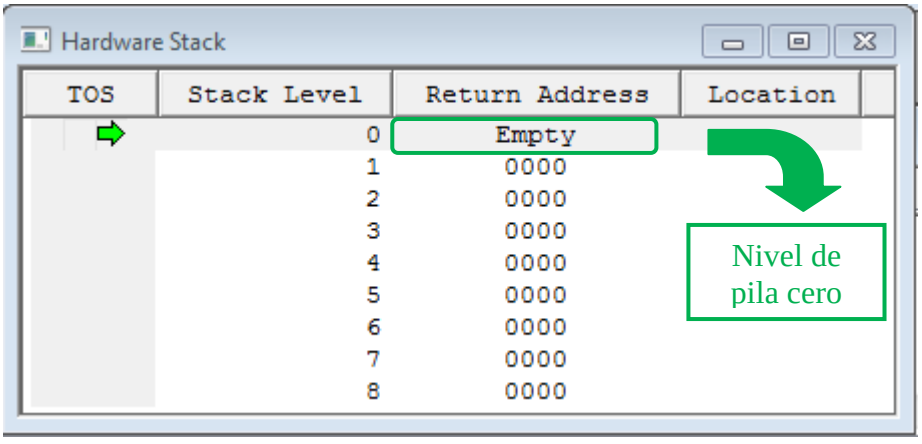


Figura 67. *Nivel de stack.*

Al entrar en la subrutina , observamos el cambio de valor del contador de programa, (Figura 68):

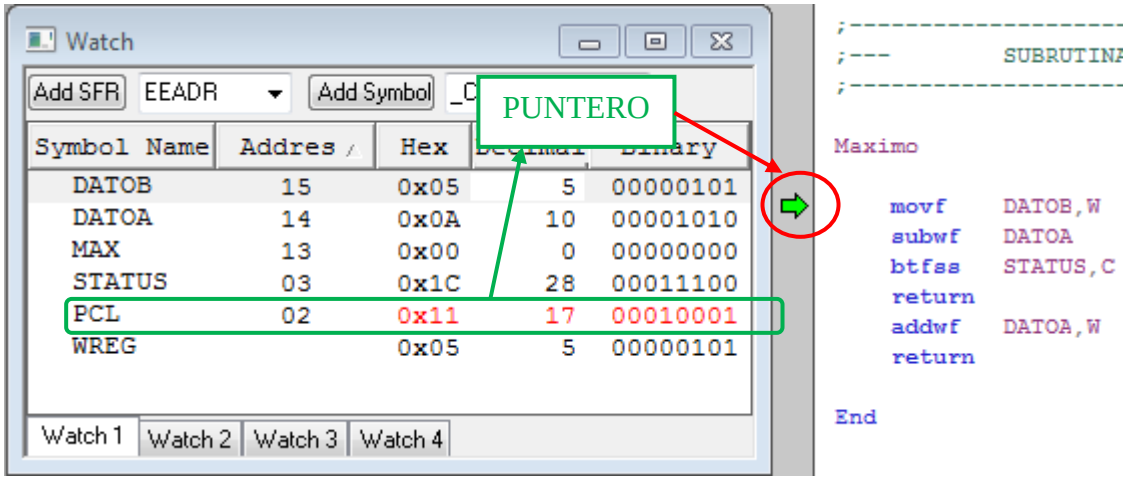


Figura 68. *Posición del PC al entrar en la subrutina.*

Y En la pila se carga la posición de la siguiente instrucción a ejecutar al terminar la subrutina, indicando en qué nivel de pila nos encontramos, (Figura 69):

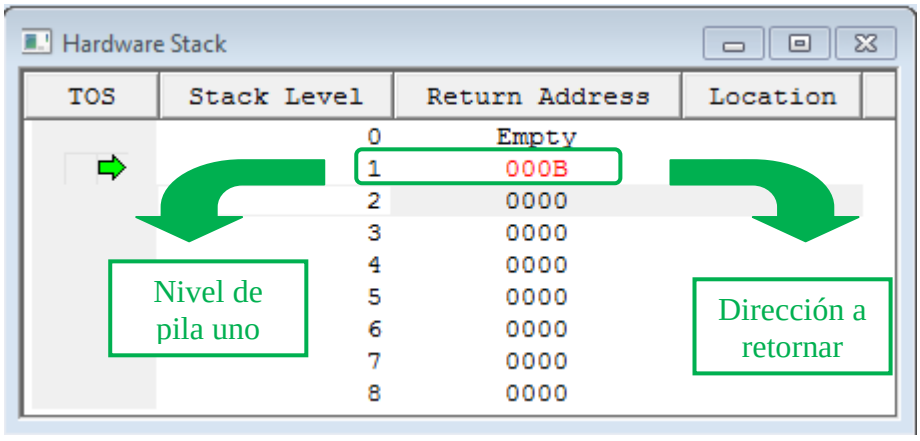


Figura 69. *Nueva posición de pila.*

Se copia DATOB en W y se realiza la resta, Al ser mayor el DATOA se pone a uno el flag C y en DATOA queda el valor de DATOA - W, (Figura 70):

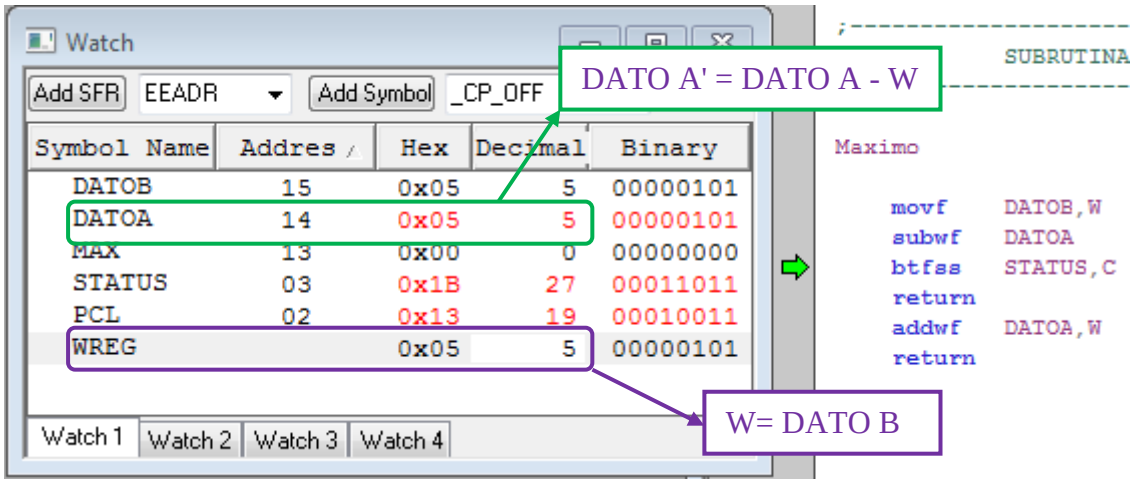


Figura 70. *Comparación de dos números.*

Testea el flag C y como está a uno se salta la siguiente instrucción, (Figura 71)

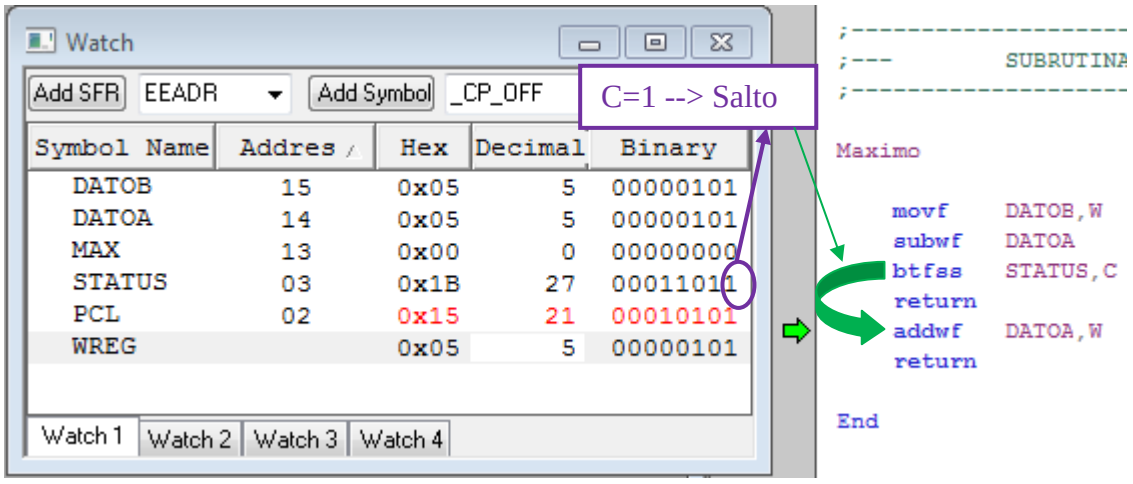


Figura 71. *Activación del bit de carry.*

Y por ultimo pasa a W el mayor dato mediante la suma explicada anteriormente en las ecuaciones (1) y (2) mostrado en la figura 72:

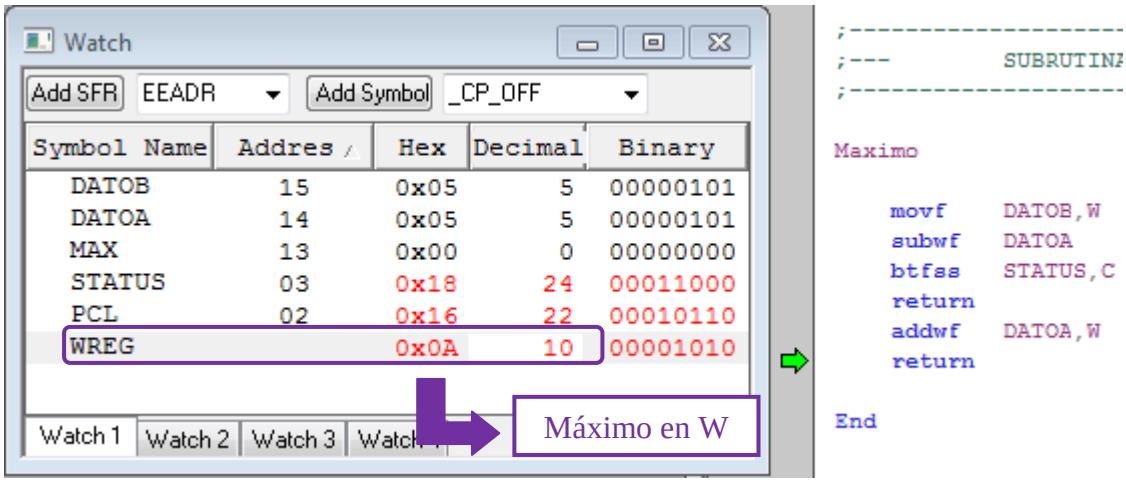


Figura 72. *El valor máximo se copia en W.*

Retorna a la instrucción indicada en la pila 0x0Bh, (Figura 73):

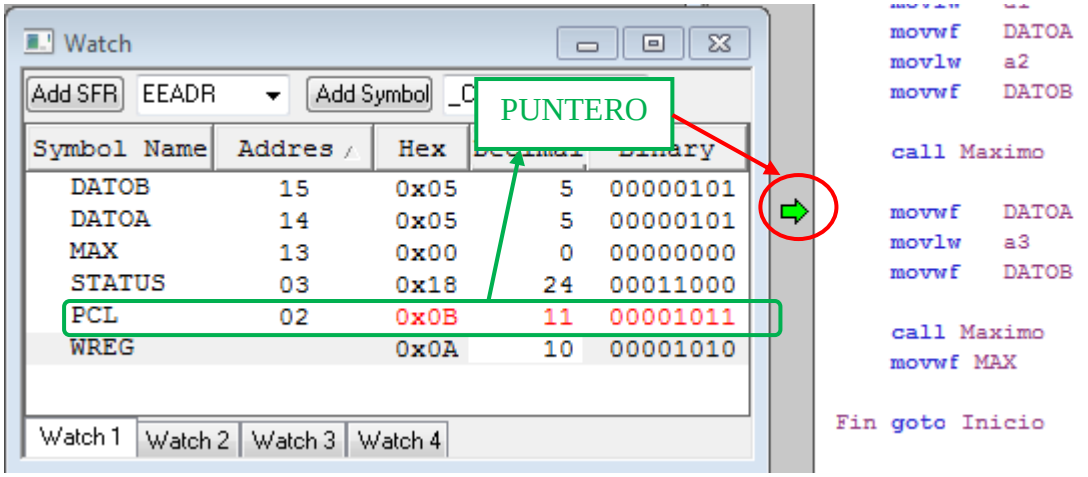


Figura 73. *Retorno de subrutina.*

Y la pila baja un nivel, (Figura 74):

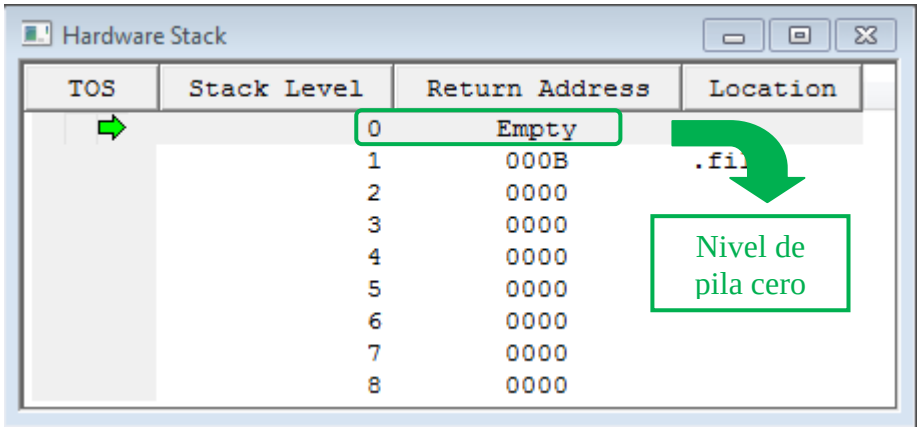


Figura 74. *Nivel de pila al retornar de subrutina.*

1.1.5.4.2.6.- Ejercicio de ampliación

1.1.5.4.2.6.1. - Planteamiento

Se trata de multiplicar dos números dados, el código ensamblador *no es capaz de realizar la multiplicación directamente a modo de una única instrucción.*

Es necesario realizar varias sumas para multiplicar dos números acorde con la definición de multiplicar (3):

$$a \cdot b = \sum_{i=0}^{b} a \tag{3}$$

En el código se creará un bucle que se repita b veces, que acumule en valor de a en la variable resultado. Al trabajar con registros de 8 bits el máximo valor que se puede obtener es (4):

$$2^8 - 1 = 255 \tag{4}$$

Por lo que se crearán dos variables, una para la parte baja del número y otra para la parte alta, la máxima suma de dos números de 8 bits es (5)

$$FFh + FFh = 1FE \tag{5}$$

Lo que significa que la parte alta sólo puede incrementarse en 1 como máximo, cada vez que se desborde.

Por último, como el bucle se va a repetir b veces, es interesante que b sea el número de menor valor de los dos a multiplicar, optimizando así el tiempo de ejecución del programa.

Se puede aprovechar el ejemplo anterior para obtener el máximo, con una ligera modificación del código para obtener también el mínimo.

1.1.5.4.2.6.2. - Diagrama de flujo

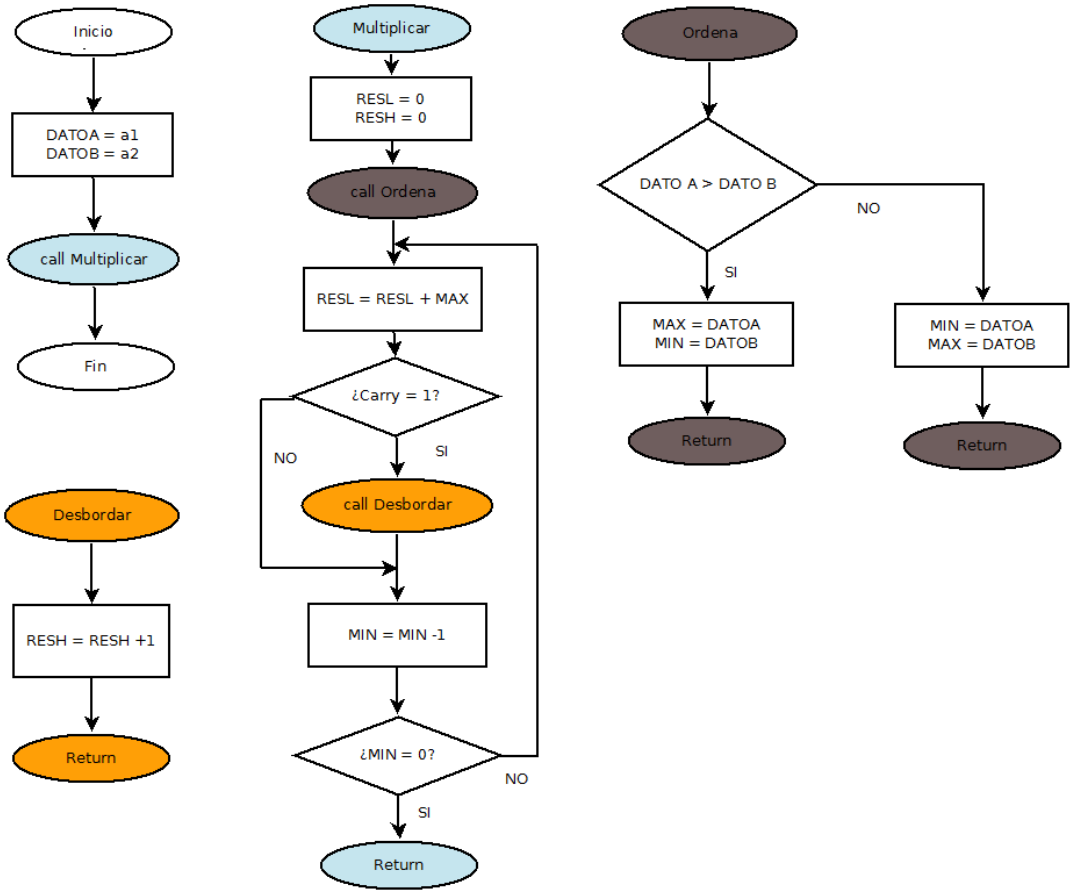


Figura 75. *Multiplicación de dos números.*

El *programa principal* comienza cargando los valores a pasar la subrutina multiplicación (DATOA, DATOB), llama a la función Multiplicar y una vez terminada, acaba el programa. (Figura 75).

La *subrutina multiplicar* comienza borrando las variables RESL y RESH y llama a una nueva *subrutina ordenar* que coloca el mayor en MAX y el menor en MIN.

A continuación sumará en la parte baja de resultado el valor máximo, si sobrepasa el valor de 255 llama a la *función desbordar* para incrementar en uno la parte alta.

Por último decrementa el valor mínimo que actúa a modo de contador del bucle, realizando tantas iteraciones como indique el valor mínimo, cuando llegue a cero regresa al programa principal con el **resultado almacenado es RESH y RESL**.

1.1.5.4.2.6.2. - Programación y simulación

1.1.5.4.2.6.2.1.- Programa principal

- *Código*

```
;-----------------------------------------------------
;---                PROGRAMA PRINCIPAL                -
;-----------------------------------------------------

    clrw            ; Inicialización de registros

    movlw   a1      ; Parámetros a multiplicar
    movwf   DATOA
    movlw   a2
    movwf   DATOB

    call Multiplicar    ; Almacena el resultado en RESL y RESH

Fin goto Inicio
```

Figura 76. *Programa principal de multiplicación.*

El código del programa principal es bastante simple, prácticamente el núcleo del programa se encuentra en las subrutinas. Carga los parámetros en DATOA y DATOB para llamar a la función Multiplicar. (Figura 76)

En el caso que se volviera a usar la función multiplicar, *no es necesario realizar un clear de los registros RESH y RESL* , ya que el la subrutina Multiplicar los inicializa a cero.

- *Simulación*

Para el caso a1 = 5 y a2 = 30, antes de llamar a Multiplicar se observa en la figura 77 los registros más importantes:

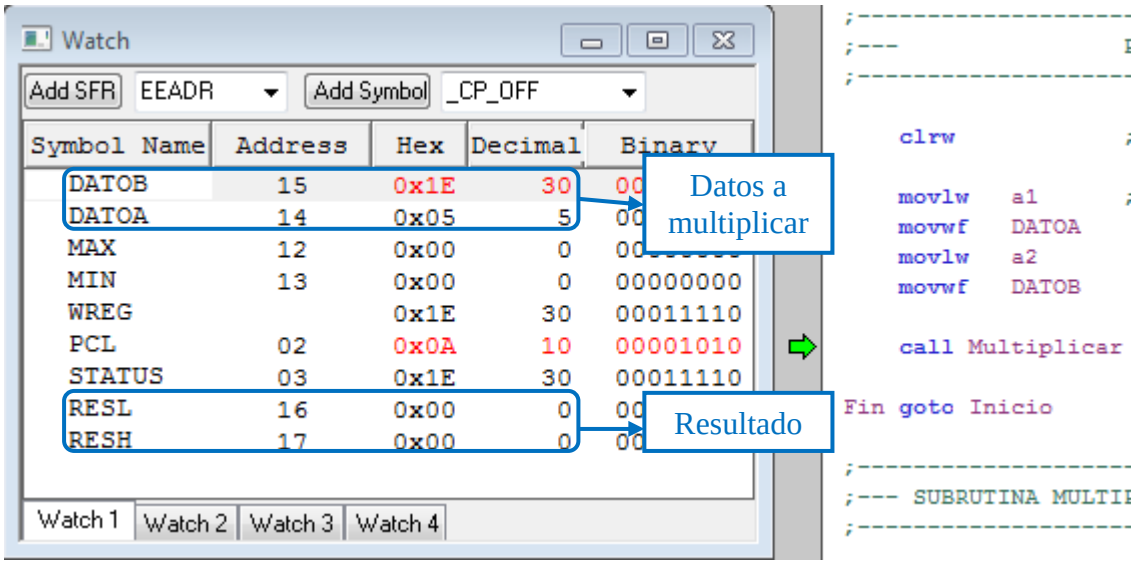


Figura 77. *Registros principales antes de entrar en la subrutina.*

Los resultados obtenidos para los casos a la salida de la subrutina de multiplicación son, (figura 78 y 79)

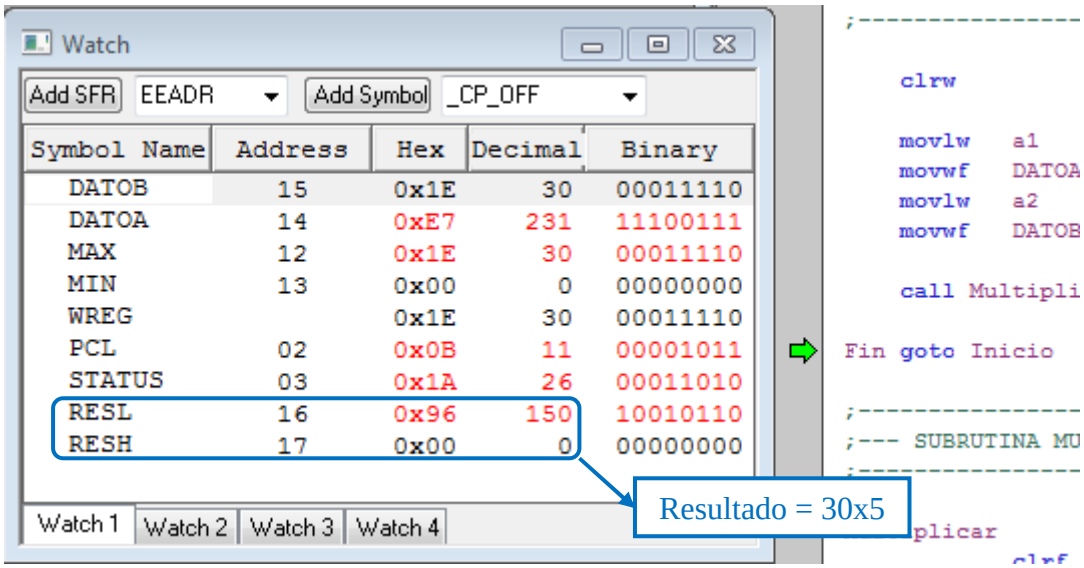


Figura 78. *Multiplicación primer ejemplo.*

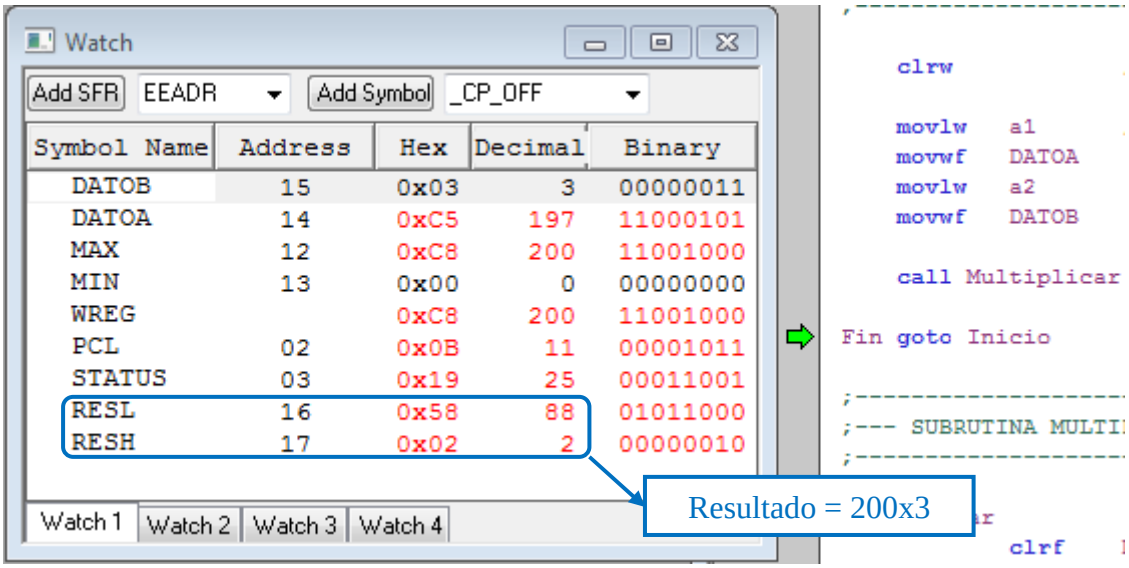


Figura 79. *Multiplicación segundo ejemplo.*

Comprobamos el valor total de resultado(6)

$$resulado = resh \cdot 255 + resl = 2x256 + 88 = 600 \qquad (6)$$

1.1.5.4.2.6.2.2.- Subrutina

- *Código*

```
;-------------------------------------------------------
;--- SUBRUTINA MULTIPLICAR    RESH & RESL =DATOA*DATOB  -
;-------------------------------------------------------

Multiplicar
            clrf    RESH     ; Puesta a cero de los resultados
            clrf    RESL
            call Ordena ; (MAX,MIN) = ORDENA(DATOA,DATOB)
Multiplica
            movf    MAX,W        ; MAX multiplicando
            addwf   RESL         ; RESL = RESL + MAX
            btfsc   STATUS,C     ; Si no hay desbordamiento salta
            call    Desbordar    ; SI C=1
            decfsz  MIN          ; Decremento el multiplicador
            goto    Multiplica   ; Si MIN > 0
            return
```

Figura 80. *Subrutina Multiplicar.*

Al empezar el bucle se ejecuta un clear de RESH y RESL para asegurarse que no contienen valor alguno, si no es la primera vez a ejecutar, tendrían el resultado de una multiplicación previa. Y Llama a la subrutina Ordena. (Figura 81)

El bucle consiste en pasar el valor máximo al registro de trabajo (un sumando debe ser W, ver **practica 1**), y sumar a resultado su contenido más el valor máximo.

Si el valor acumulado de resultado supera 255, el flag C se activará, en ese caso lo testeamos con btfsc, si no se desborda salta la siguiente instrucción, si se desborda llama la subrutina desbordar, que consiste en incrementar uno la parte alta de resultado, (Figura 81)

```
;-------------------------------------------------------
;--- SUBRUTINA DESBORDAR    RESH = RESH + 1             -
;-------------------------------------------------------

Desbordar   incf    RESH        ; RESH = RESH + 1
            return
```

Figura 81. *Subrutina Desbordar.*

Decrementa en uno con salto el registro MIN , si MIN = 0 activa el flag Z que chequea la instrucción saltando el *'goto Multiplica'*, finalizando la subrutina. Si no llega cero realiza una nueva iteración con el valor min decrementado.(Figura 32).

```
;-------------------------------------------------------
;--- SUBRUTINA ORDENA    (MAX,MIN)= ORDENA(DATOA,DATOB) -
;-------------------------------------------------------
Ordena
    movf    DATOA,W
    movwf   MIN
    movf    DATOB,W         ; W = DATOB
    movwf   MAX
    subwf   DATOA           ; DATOA' = DATOA - DATOB
    btfss   STATUS,C        ; Si la resta es negativa salta
    return                  ; b>a W=DATOB
    addwf   DATOA,W         ; a>=b W=DATOB + DATOA' = DATOA
    movwf   MAX
    movf    DATOB,W
    movwf   MIN
    return
```

Figura 82. *Subrutina Ordena*

Esta subrutina es muy similar al ejercicio básico, la cual se le ha añadido al inicio el supuesto que DATOB sea mayor, guardando B en MAX y A en MIN, después lo comprueba, si es cierto retorna y si es falso intercambia MAX y MIN,(Figura 82).

↳ *Simulación*

Para la simulación de la subrutinas se han tomado los datos a1=200 y a2=3. Los valores antes de entrar son, (Figura 83).

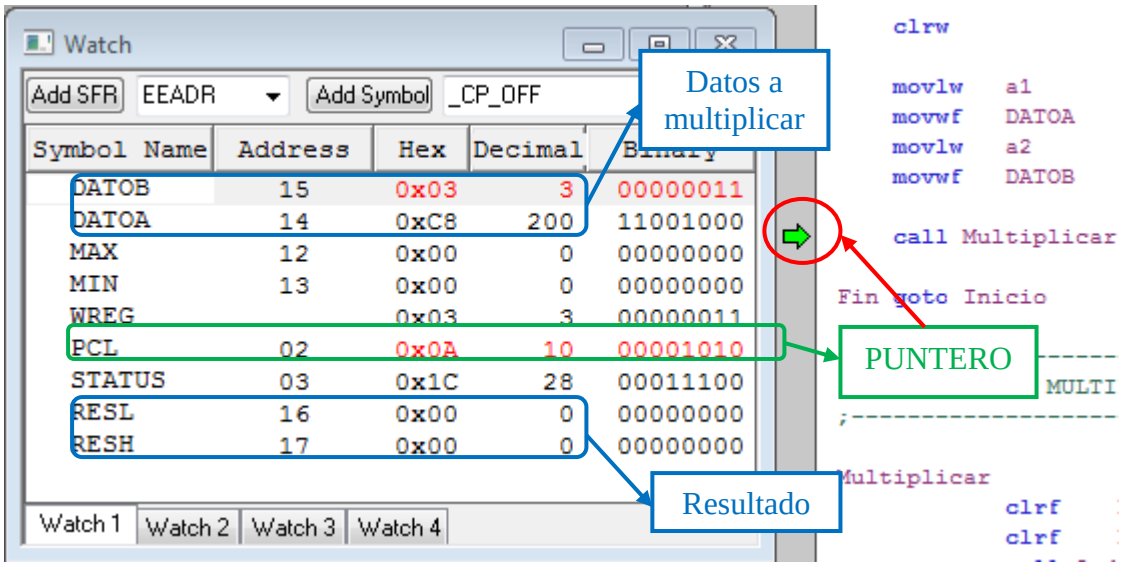


Figura 83. *Registros a la entrada de la subrutina.*

La pila inicialmente se encuentra vacía, (Figura 84):

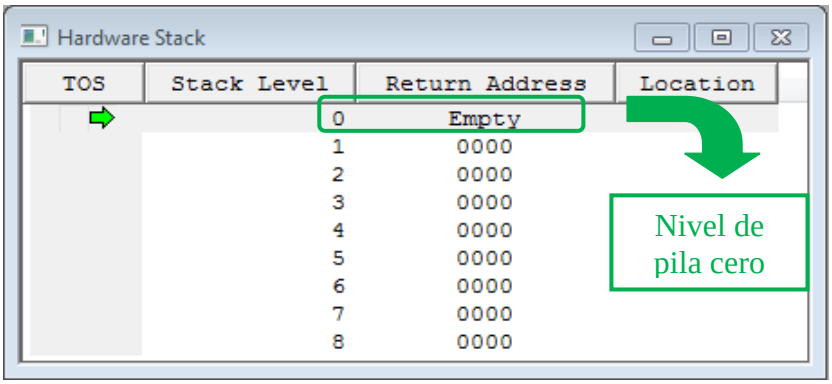


Figura 84. *Nivel de pila antes de entrar en la subrutina.*

Al entrar en la subrutina cambia al nivel 1 de la pila indicando la dirección de la siguiente instrucción a ejecutar cuando retorne, (Figura 85):

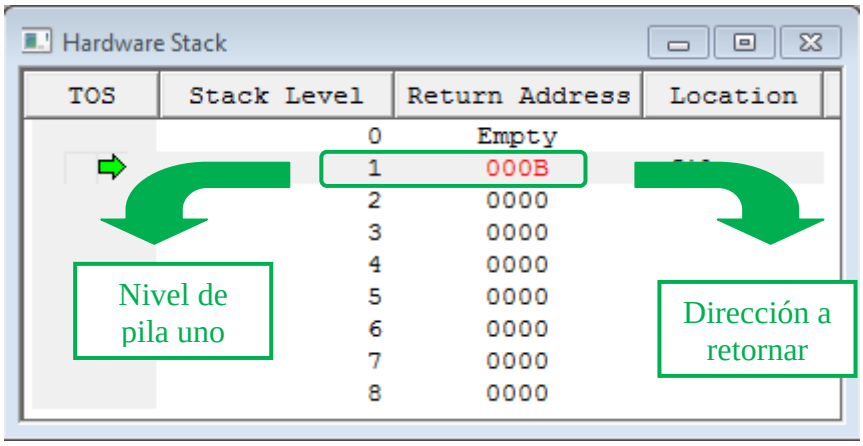


Figura 85. *Nivel de pila al entrar en la subrutina.*

Al entrar en la función Ordena , cambia al nivel 2 , almacenando la dirección a la que tiene que regresar el programa al acabar la subrutina ordena,(Figura 86):

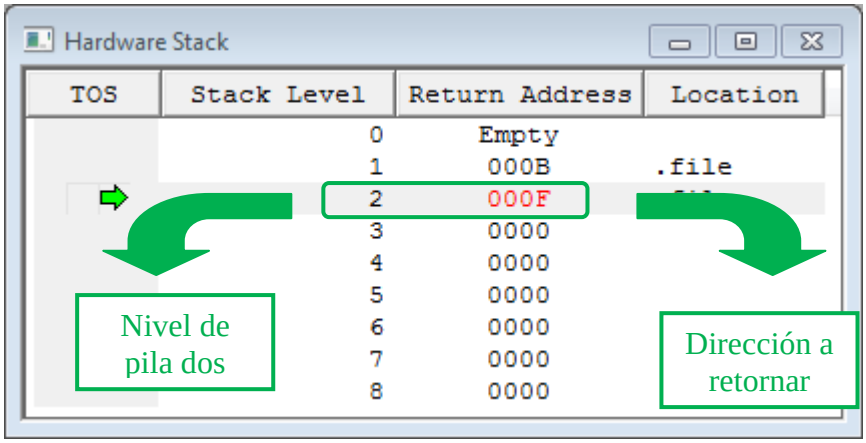


Figura 86. *Nivel de pila en la subrutina ordena.*

Almacena el supuesto MAX=DATOB y MIN=DATOA y después los compara, (Figura 87):

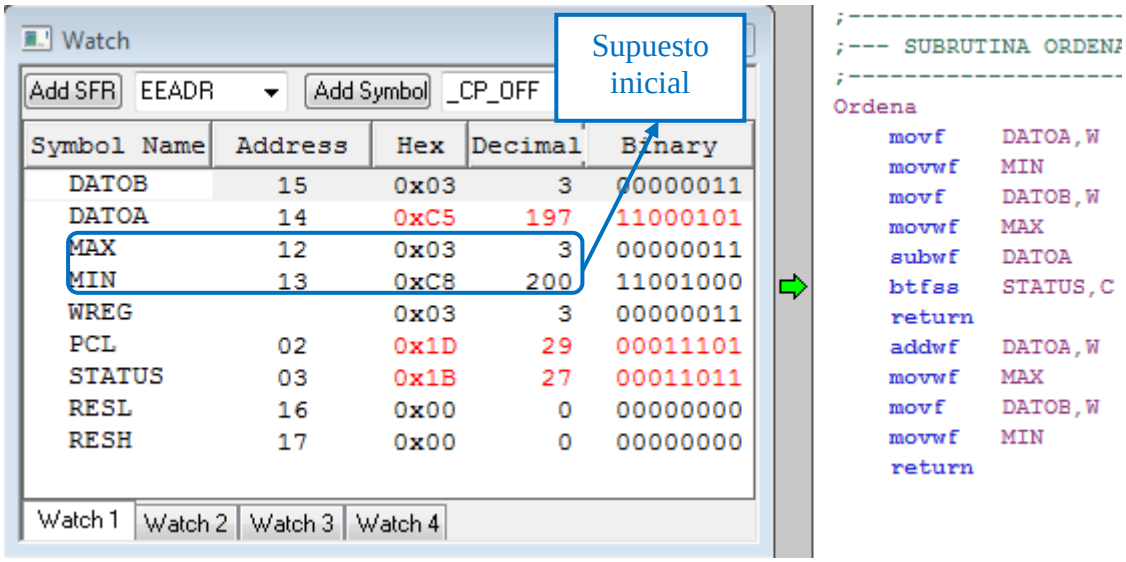


Figura 87. *Comparación de MAX y MIN.*

Se activa el flag C , lo que significa que el supuesto es erróneo, por lo que salta la instrucción de retorno e intercambia MAX y MIN, (Figura 88):

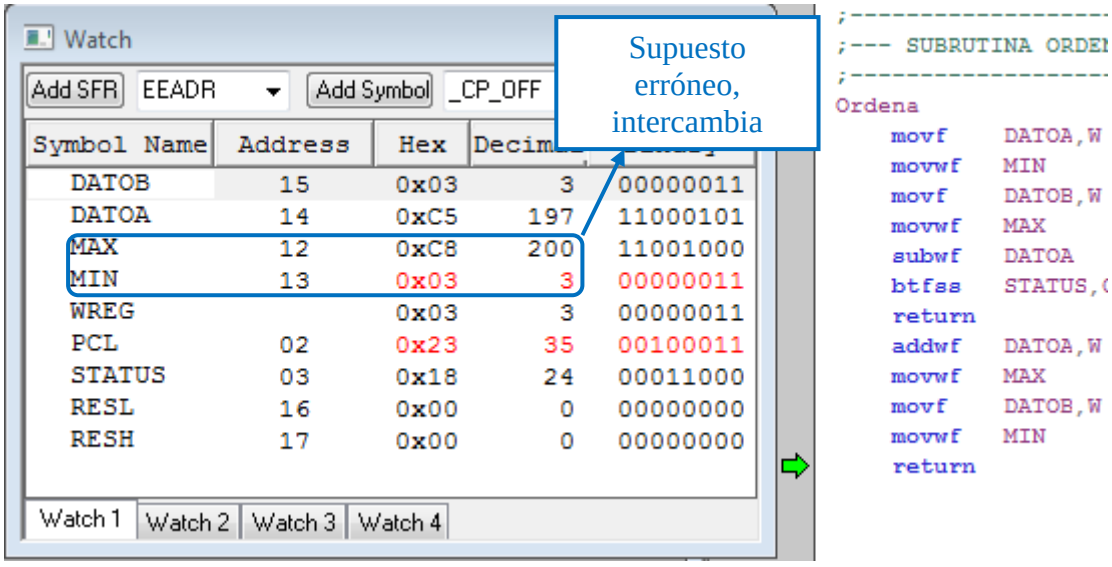


Figura 88. *Corrección del supuesto.*

Al retornar regresa a Multiplicar, y se observa que la pila desciende un nivel, retornando a la instrucción indicada por la pila 0x0F,(Figura 89 y 90):

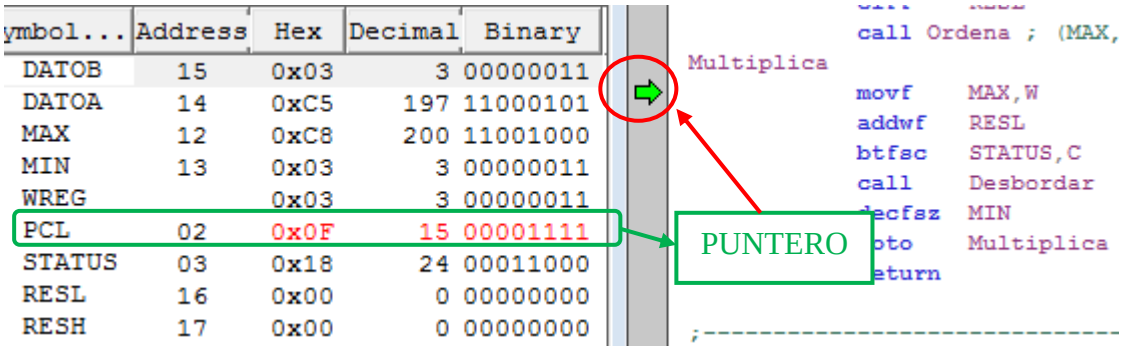


Figura 89. *Retorno a subrutina Multiplicar.*

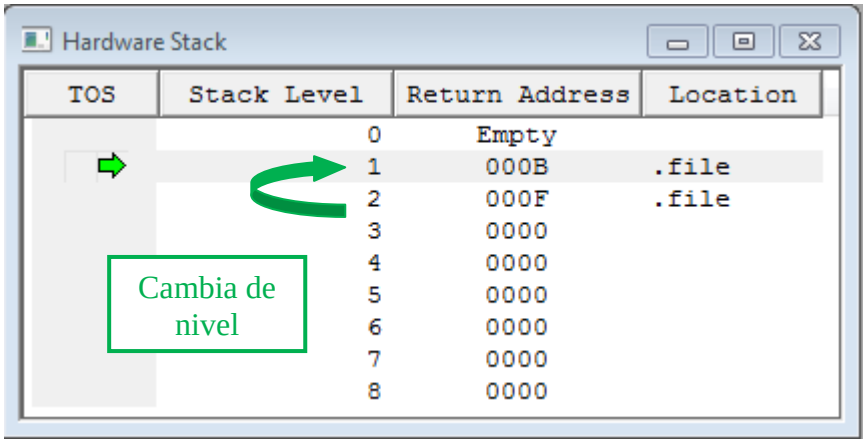


Figura 90. *Cambio de nivel de pila.*

Suma a RESL el valor de MAX, (Figura 91):

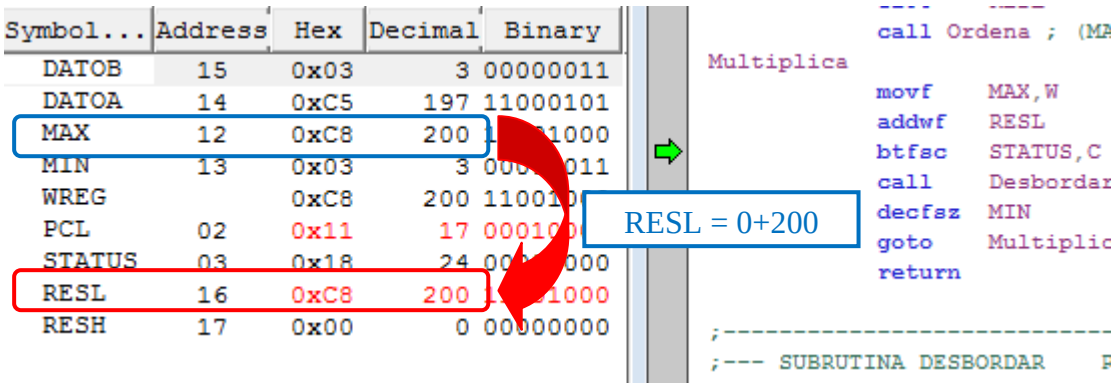


Figura 91. *Primera iteración.*

Al no desbordarse salta, y decrementa MIN comprobando si llega a cero, (Figura 92):

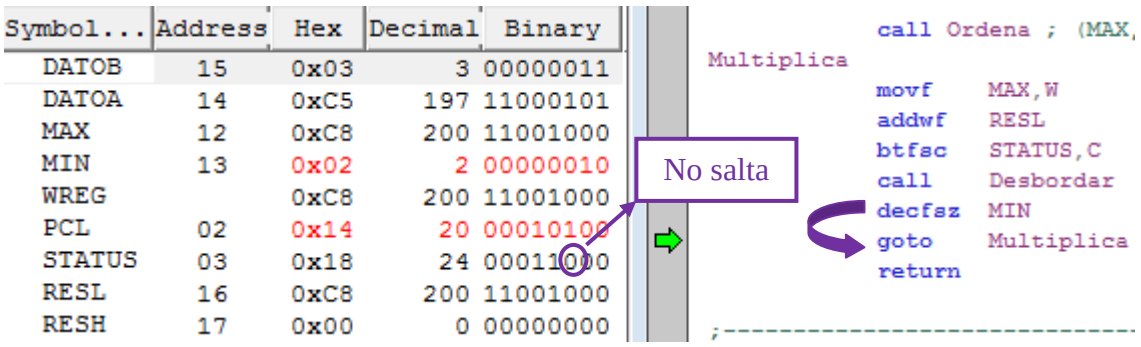


Figura 92. *Fin de la primera iteración.*

Se ejecuta una nueva iteración, produciéndose desbordamiento, (Figura 93):

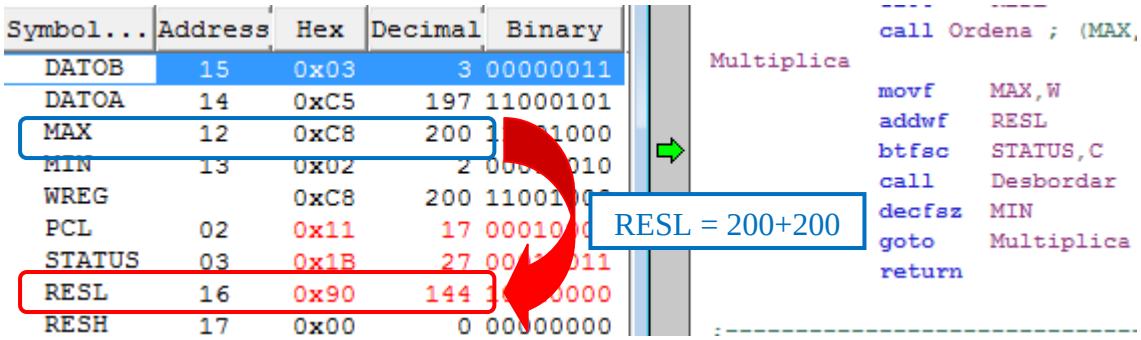


Figura 93. *Nueva iteración.*

Por lo que al testear el bit C no se saltara la siguiente instrucción, (Figura 94):

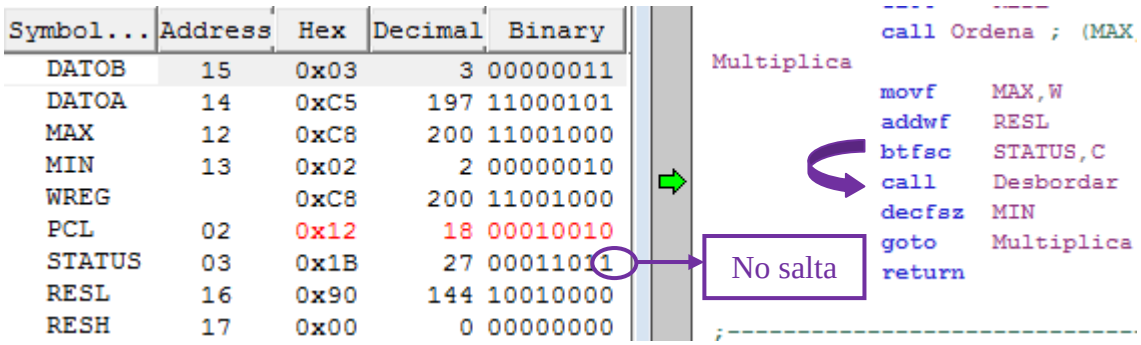


Figura 94. *Llamada a la subrutina Desbordar.*

Llama a la subrutina Desbordar ascendiendo un nivel en la pila, (Figuras 95 y 96):

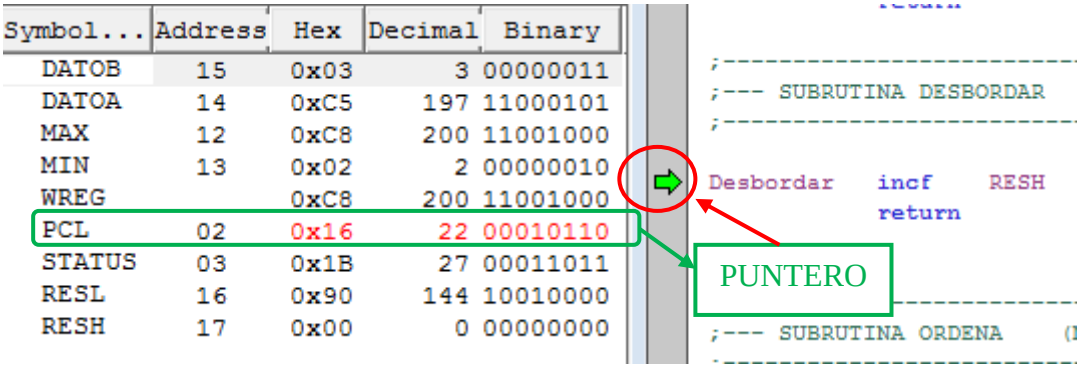


Figura 95. *Inicio de la subrutina Desbordar.*

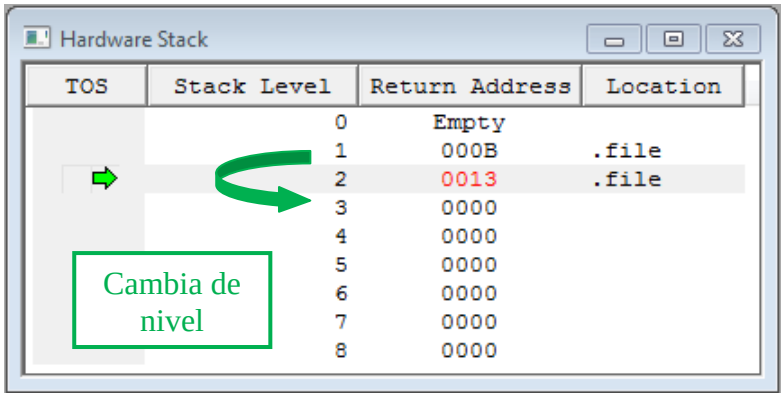


Figura 96. *Nuevo nivel de pila.*

En la cual incrementa en uno el valor de RESH y retorna, (Figura 49):

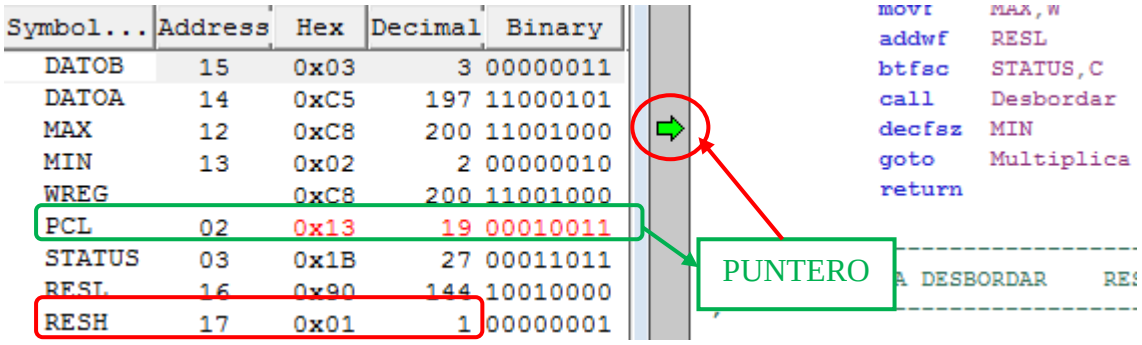


Figura 97. *Retorno de subrutina Desbordar.*

Descendiendo en el nivel de la pila,(Figura 98):

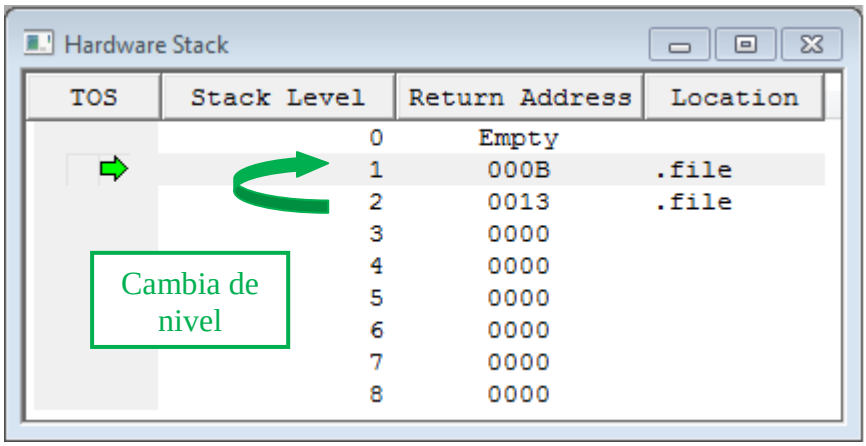


Figura 98. *Cambio de nivel de pila.*

Continúan las iteraciones hasta que mínimo llegue a cero, en ese momento se activará el flag Z y mediante la instrucción *decfsz*, de decremento con salto, el contador de programa pasara a la instrucción *return*. (Figura 99):

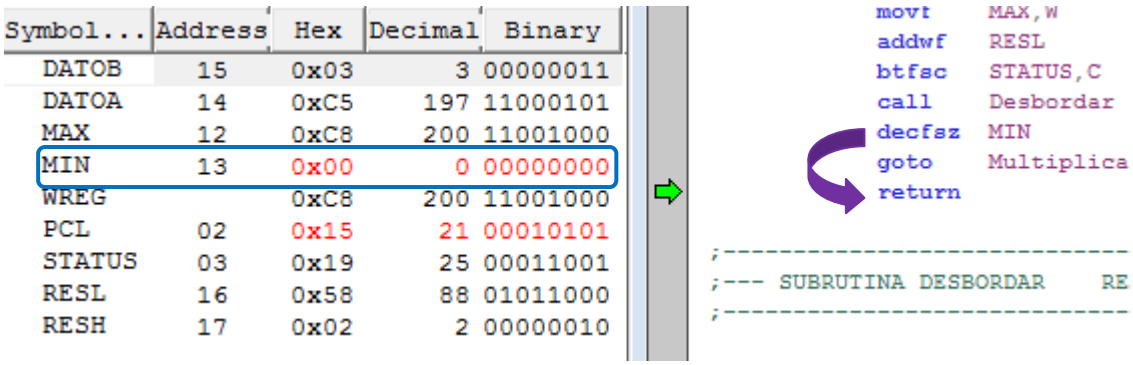


Figura 99. *Fin del bucle.*

Y regresa al programa principal, descendiendo al nivel mínimo de pila, el resultado de la multiplicación quedará almacenado en los registros RESL y RESH (Figura 100 y 101):

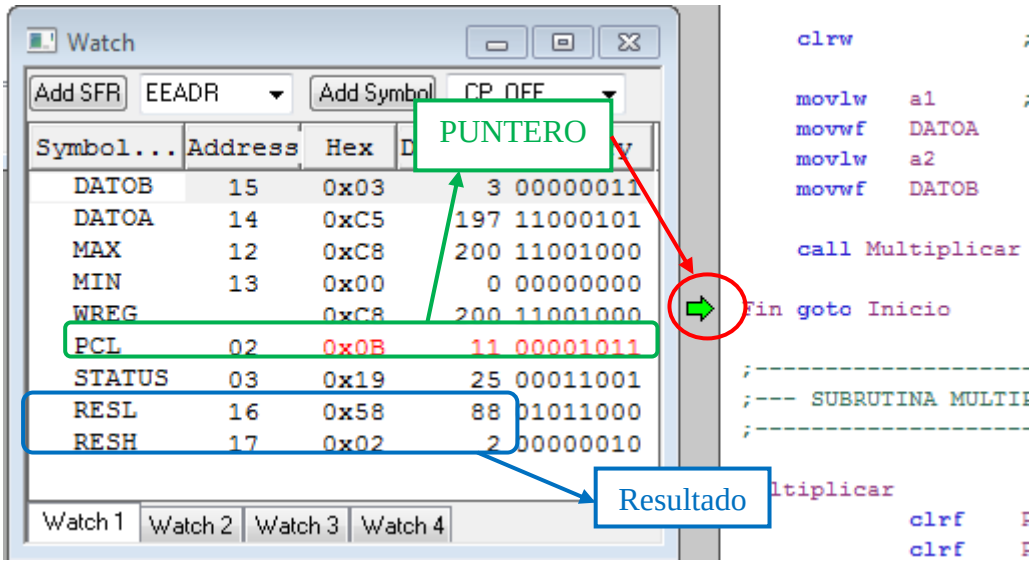


Figura 100. *Retorno al programa principal.*

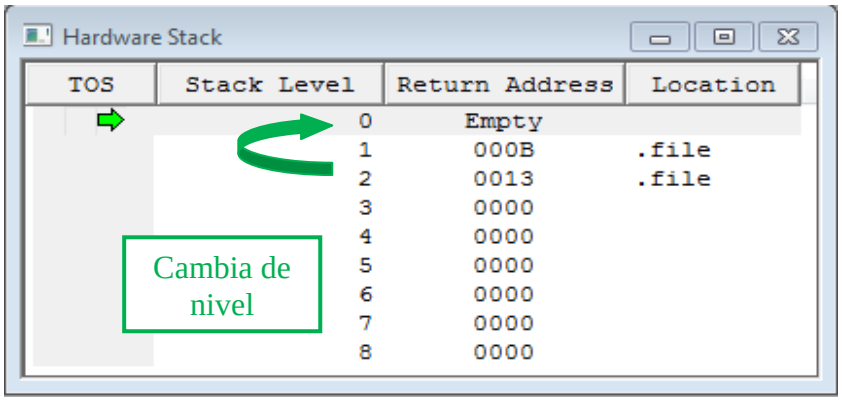


Figura 101. *Nivel inicial de pila.*

1.1.5.4.2.7.- Programas propuestos

Realizar un programa que ordene de menor a mayor tres números dados, aprovechando la función ordena del ejercicio de ampliación.

Realizar un programa capaz de contar desde un valor inicial dado (inicio = 39) hasta un valor final dado (fin =255), cada vez que llegue a fin se incrementará en uno un registro. El programa debe terminar cuando el registro alcance un valor de 7. En ese momento activar el bit 5 de un registro denominado salida.

1.1.5.4.3.- Práctica 3 - Modos de direccionamiento. y tablas

1.1.5.4.3.1.- Objetivo

- ✓ Uso del contador del programa para recorrer tablas.
- ✓ Comprensión y aplicación de los distintos modos de direccionamiento.
- ✓ Entender el funcionamiento de las tablas en lenguaje ensamblador

1.1.5.4.3.2.- Enunciado

Realizar un programa que transforme el valor de 5, 0 y 9 en un valor binario correspondiente al encendido de un display de 7 segmentos.

Realizar un programa que escriba en memoria la secuencia de datos 20, 19, 18, 17, ...,11 desde la posición 0x30 hasta la posición 0x39 respectivamente.

1.1.5.4.3.3.- Ejercicio de ampliación

Realizar un programa que muestre en un display de 7 segmentos un valor introducido, además ese valor será almacenado en la memoria desde la posición 0x30 hasta la 0x3F.

Si el valor introducido es mayor que nueve en vez de almacenarle realizará un borrado de memoria desde la posición 0x30 hasta la posición 0x3Fy en el display aparecerá la letra E.

Nota: Crear dos REGISTROS llamados PUERTOA y PUERTOB para simular los puertos del PIC, al no conocer todavía el manejo de los puertos.

1.1.5.4.3.4.- Contenido teórico

1.1.5.4.3.4.1- Modos de direccionamiento

En programación existen varias formas de indicarle al microcontrolador el dato que tiene que utilizar, las distintas maneras de indicarlo se denominan direccionamiento.

- *Direccionamiento inmediato*

El **dato usado por en la instrucción se encuentra contenido en la propia instrucción**, en este caso se denomina literal [1], un ejemplo de direccionamiento es el siguiente:

```
MOVLW   0x12
```

- *Direccionamiento directo*

En esta forma de direccionamiento de la memoria de datos se trata de comunicar al PIC la **posición de la memoria donde se encuentra el dato**, en vez de decirle el dato propiamente dicho. Para ello se puede indicar la posición con el valor de la dirección donde se encuentra el dato o generalmente con el nombre asignado a esa dirección:

- NOMBRE equ 0x10 ; Se indica que nombre hace referencia a la dirección 0x10 de la memoria de datos, (ver declaración de variables y constantes en la práctica 1)

- MOVF NOMBRE,W ; Se le indica que queremos mover el contenido de la dirección a la que hace referencia nombre.

No se debe confundir el contenido de una dirección con la propia dirección, por ejemplo en este caso no se copia el valor 0x10 (dirección de la memoria de datos) sino se copia el contenido de esa dirección, previamente programado.

La memoria RAM se encuentra ordenada en bancos, concretamente el PIC16F886 estructura la memoria en cuatro bancos llamados banco 0, banco 1, banco 2 y banco 3.

Cuando se trabaja con este modo de direccionamiento hay que asegurarse haber seleccionado correctamente el banco en el que se encuentra el dato a buscar, en el apartado *otros anejos, el PIC16F886* se muestra un esquema de la organización de la memoria RAM.

Para seleccionar del banco se usa el registro status, a continuación se muestra en la Tabla 7 el registro STATUS y se analizan los bits usados en el direccionamiento de datos.

Tabla 7. *Registro STATUS.*

REGISTRO STATUS (0x03h)							
BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
IRP	RP1	RP0	TO	PD	Z	DC	C

★ IRP: Selección de banco en modo de *direccionamiento indirecto* junto con el bit de mayor peso del registro FSR.

★ RP0,RP1 --> Selección de banco en *direccionamiento directo*, como se muestra en la tabla 8:

Tabla 8. *Selección de banco.*

RP1	PR0	Banco
0	0	Banco 0
0	1	Banco 1
1	0	Banco 2
1	1	Banco 3

↳ *Direccionamiento indirecto*

Este modo de direccionamiento de la memoria RAM consiste en el uso de un puntero, para ello se usan los registros el FSR (0x04 , 0x84) y el INDF (0x00 y 0x80):

★ El registro FSR **contiene una dirección de la memoria de datos**, el bit más significativo junto con el IRP del registro STATUS indican el banco

★ El registro INDF no está implementado físicamente, y **el contenido es el contenido de la dirección de memoria de FSR**.

En la figura 102 se muestra su utilización:

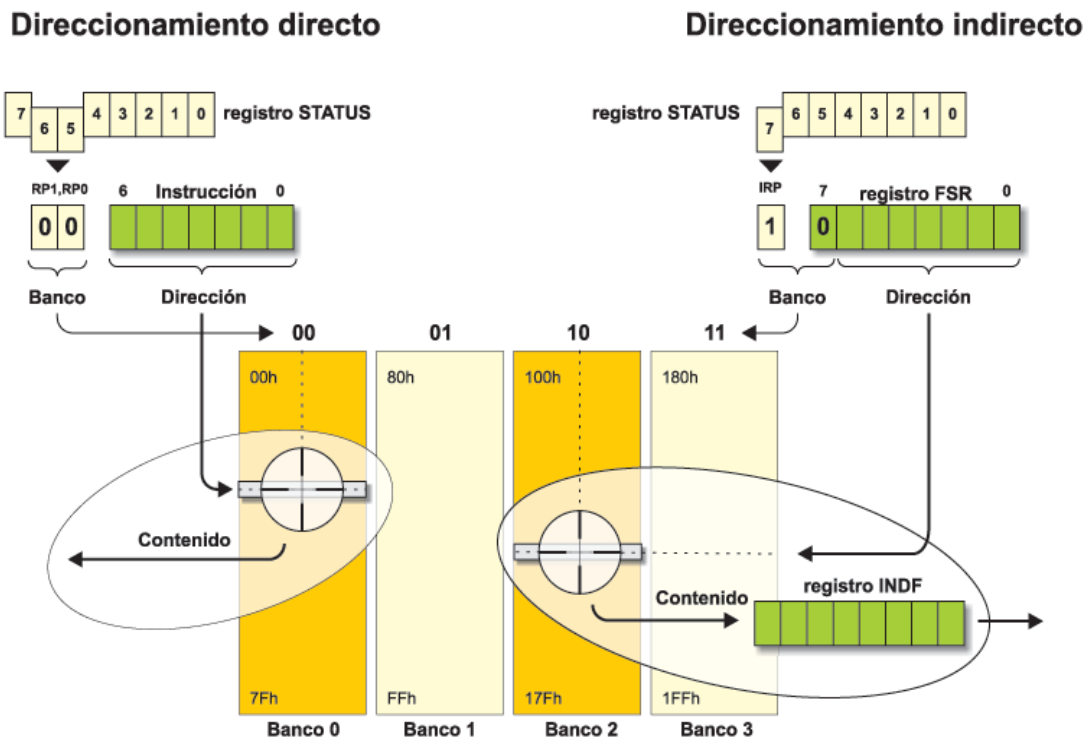


Figura 102. *Modos de direccionamiento de la memoria de datos.*[8]

Este modo de direccionamiento es muy útil, con un bucle que incremente en 1 el puntero FSR recorreremos toda la memoria de datos pudiendo así borrarla, extraer datos o modificar datos.

1.1.5.4.3.4.2- Tablas

El uso de tablas en los microcontroladores es frecuente en programación. Sus mayores usos son la *codificación de un display de 7 segmentos, codificación de un teclado matricial, selección de mensajes predefinidos*, etc.

El funcionamiento de una tabla es simple, cada instrucción de la tabla realizará un retorno del tipo *'retlw'* (retorno con un valor cargado en W) y mediante una suma al contador de programa iremos a la instrucción deseada de dicha tabla.

En la figura 103, se muestra un ejemplo de una tabla. Al ejecutarse la instrucción *'addwf PCL'* el contador de programa se incrementa en uno (igual que cada vez que se ejecuta una instrucción, llegando a la instrucción *'retlw a1'*) pero además hay que sumarle el contenido de w, en este caso 2 colocándose en la instrucción retlw a3.

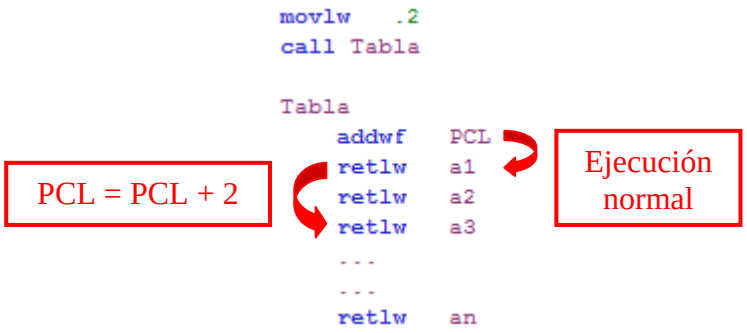


Figura 103. *Funcionamiento de las tablas.*

Es importante tener en cuenta que el contador de programa no puede exceder de un valor, al estar sumando en la parte baja **como máximo podríamos hacer una tabla de 256 entradas**, sin embargo si la tabla se ubica al final, el contador de programa ya tendrá un valor y por lo tanto la tabla será menor. **Suele ser conveniente colocar las tablas al principio del programa para evitar este problema**.[3]

También hay que vigilar que el valor de *w no exceda de la cantidad de entradas de la tabla*, lo que provocaría un PCL no deseado, produciendo errores en el programa.

1.1.5.4.3.5.- Ejercicio básico

1.1.5.4.3.5.1.- Planteamiento

Para la primera parte del ejemplo se ha elegido un display de catado común, lo que significa que para encender cada led es necesario enviar un 1 al LED, (Figura 3)

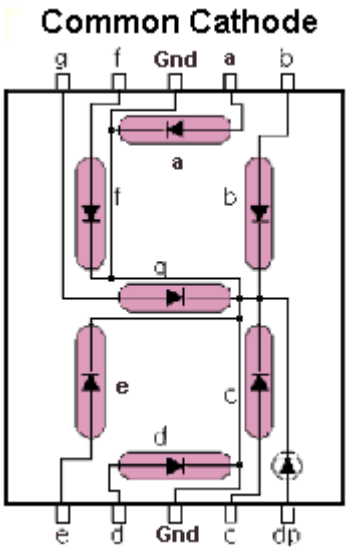


Figura 104.- *Cátodo común.*

En la tabla 9 se muestra en valor de salida en binario para cada digito:

Tabla 9. *Codificación 7 segmentos en cátodo común.*

DÍGITO	dp	G	F	E	D	C	B	A
0	0	0	1	1	1	1	1	1
1	0	0	0	0	0	1	1	0
2	0	1	0	1	1	0	1	1
3	0	1	0	0	1	1	1	1
4	0	1	1	0	0	1	1	0
5	0	1	1	0	1	1	0	1
6	0	1	1	1	1	1	0	1
7	0	0	0	0	0	1	1	1
8	0	1	1	1	1	1	1	1
9	0	1	1	0	1	1	1	1
E	0	1	1	1	1	0	0	1

Se ha implementado la codificación de la letra 'E' para indicar si hay alguno error en el programa que aparezca dicha letra.

Antes de realizar la llamada a la tabla será necesario comprobar que no se sobrepase la tabla al sumar el valor del registro del trabajo al contador de programa. [3]

En la segunda parte se usará el modo de direccionamiento indirecto explicado anteriormente, mediante una variable contador cuyo valor inicial contendrá el numero de posiciones de memoria a grabar.

Esta variable contador se decrementa en el paso de cada iteración hasta llegar al valor de 0, momento en el que finalizará la subrutina.

En cada iteración se incrementará el puntero en uno para que apunte a la siguiente dirección a guardar y se grabará el dato.

Para grabar el dato por direccionamiento indirecto se copia en el registro INDF, automáticamente ese dato se almacenará también en el registro apuntado por el puntero INDF.

1.1.5.4.3.5.2.- Diagrama de flujo

- *Primera parte*

En el programa principal se almacena el primer número en la variable dígito y posteriormente se llama a la subrutina TABLA.

El retorno de la subrutina será el valor a sacar por el display que se almacenará en una variable llamada DIGITO_DISPLAY.

La función tabla empieza comparando si el digito está entre cero y nueve, sino es así *hay un error en el digito por lo que tomara el valor 10, correspondiente en la tabla con la codificación de 'E'.*

Se añade al contador del programa el valor del digito recorriendo la tabla hasta dicho valor y por último se obtendrá la codificación del valor binario al valor del display en el registro de trabajo.(Figura 105)

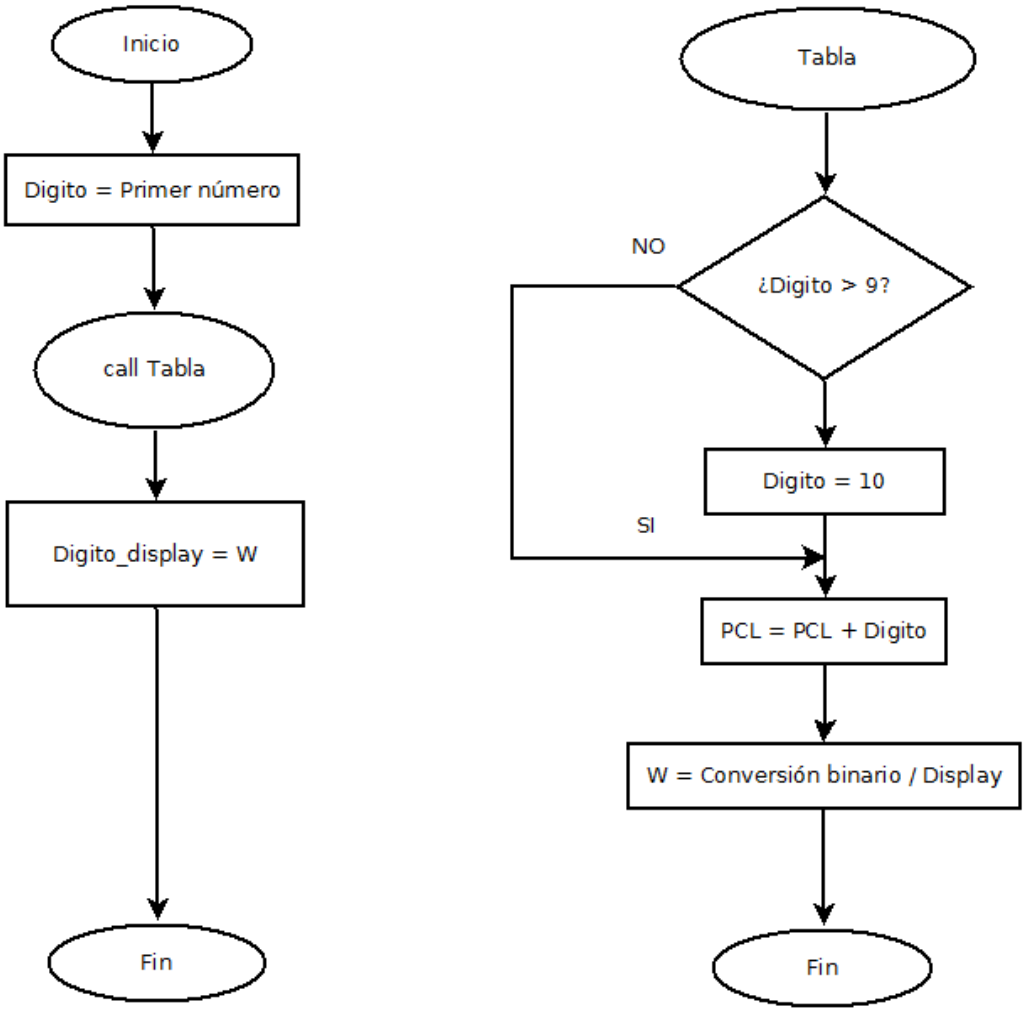


Figura 105. *Visualización en un display.*

↳ *Segunda parte*

La variable contador determinará el numero de posiciones a almacenar en la memoria a partir de una posición denominada inicial.

Se comienza iniciando los registro, cabe destacar que el puntero (FSR) se le inicializa a una posición anterior de memoria a almacenar en la RAM, esto se debe a que nada más comenzar el bucle se incrementa en uno (**preincremento**).

Otra opción que se verá en el ejercicio de ampliación es incrementarle al final del bucle, en ese caso tomaría el valor de la posición inicial y no la anterior.

Posteriormente se pasa el valor del dato a almacenar al registro INDF y simultáneamente el valor de INDF será el valor del registro al que apunta FSR.

Por último se decrementa el numero a almacenar y el contador hasta que llegue a cero, entonces terminará el programa.(Figura 106):

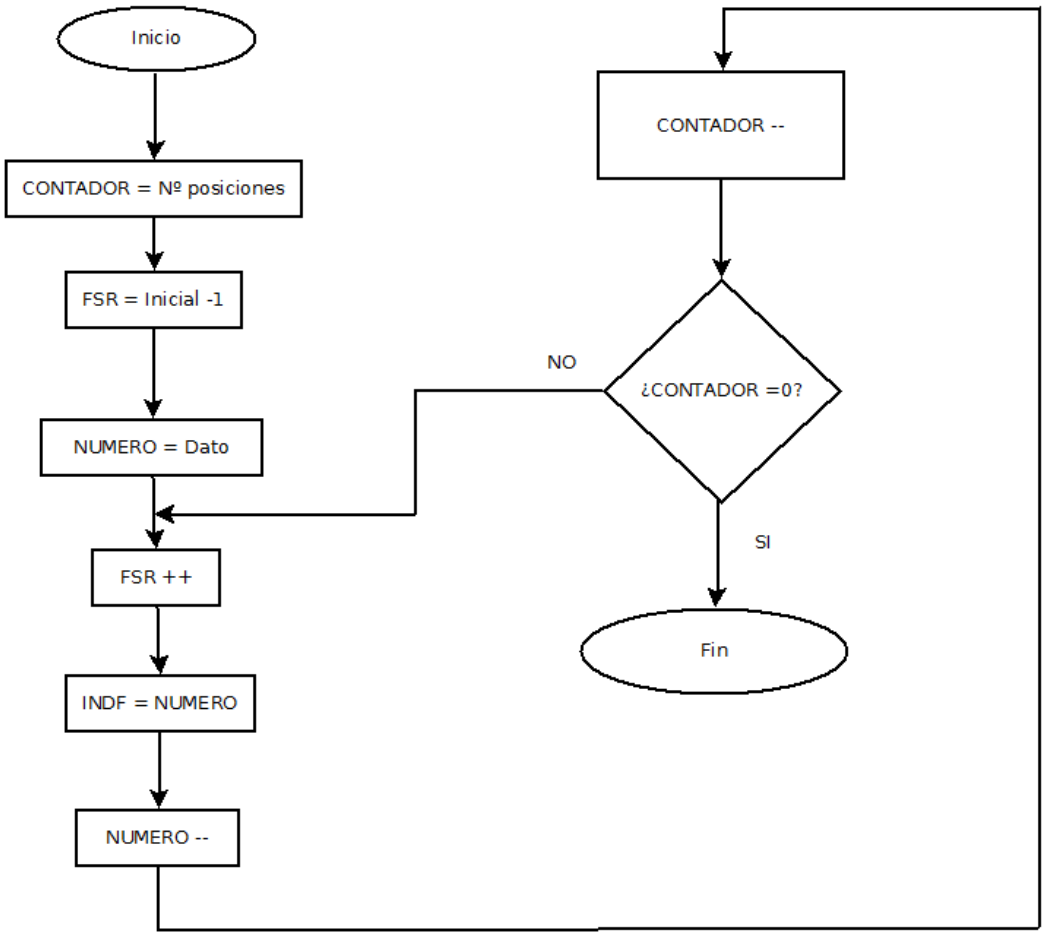


Figura 106. *Llenado de memoria.*

1.1.5.4.3.5.3.- Programación y simulación

1.1.5.4.3.5.3.1- Primera parte

Las variables y constantes usadas en esta parte se muestran en figura 107:

```
--------- DISPLAY 7 SEGMENTOS -----------------------

 #define valor1  .5      ; Valores a codificar
 #define valor2  .0
 #define valor3  .9
 #define maximo  .9      ; Valor máximo para comparar en la tabla

- Display   ---

 #define cero    b'00111111'
 #define uno     b'00000110'
 #define dos     b'01011011'
 #define tres    b'01001111'
 #define cuatro  b'01100110'
 #define cinco   b'01101101'
 #define seis    b'01111101'
 #define siete   b'00000111'
 #define ocho    b'01111111'
 #define nueve   b'01101111'
 #define fallo   b'01111001'     ; Letra E

 DIGITO equ 0x20             ; Variable donde se almacena el número a convertir
 DIGITO_DISPLAY  equ 0x21    ; Valor del display
 TEMP    equ 0x22            ; Variable auxiliar
```

Figura 107. *Registros y constantes.*

- ★ *Valor1,2 y 3* son los datos en binario de los que se quiere obtener su codificación en un display de 7 segmentos.

- ★ *máximo* es una constante que permite ver si el digito introducido está entre 0 y 9 mediante una comparación.

- ★ *cero,..., fallo* valores de cada número con los leds a activar en el display.

- ★ *DIGITO* es una variable que contendrá el dígito en binario

★ *DIGITO_DISPLAY*: variable que almacena la codificación en 7 segmentos.

1.1.5.4.3.5.3.1.1 - Programa principal

↳ *Código*

```
Inicio                  ; Etiqueta de comienzo de programa

    ;---------- PARTE A CODIFICACIÓN DISPLAY --------------

    clrw
    movlw   valor1      ; Paso a DIGITO el valor binario a convertir
    movwf   DIGITO
    call    Tabla       ; Tabla saca a W la conversión
    movwf   DIGITO_DISPLAY  ; Paso a DIGITO_DISPLAY el valor del display
    movlw   valor2      ;
    movwf   DIGITO
    call    Tabla
    movwf   DIGITO_DISPLAY
    movlw   valor3      ;
    movwf   DIGITO
    call    Tabla
    movwf   DIGITO_DISPLAY
```

Figura 108. *Programa principal.*

Se trata de un programa lineal, que cargará en el *registro DIGITO el valor del número en binario*, ese valor será el argumento que se pasará a la función Tabla y en su retorno *devolverá el digito codificado en el registro de trabajo* para la activación de un display de siete segmentos.

El *valor codificado se guardará en el registro DIGITO_DISPLAY*, repitiéndose el proceso las tres veces indicadas del enunciado de la práctica. (Figura 108)

↳ *Simulación*

Se carga el primer valor, cinco en el registro DIGITO, (Figura 109):

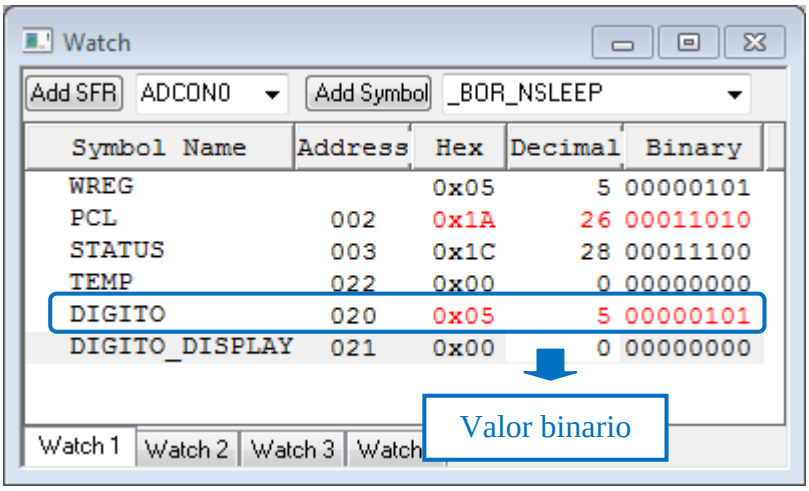


Figura 109. *Primer valor a codificar.*

Se llama a la función tabla que devuelve en el registro de trabajo el valor de 5 codificado para un display de 7 segmentos '01101101', (Figura 110):

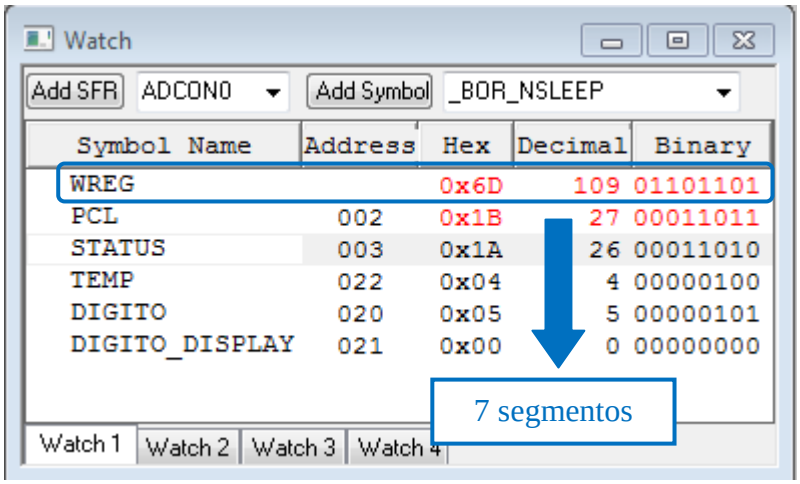


Figura 110. *Retorno de la subrutina Tabla.*

Por último se guarda ese valor en el registro DIGITO_DISPLAY, (Figura 111):

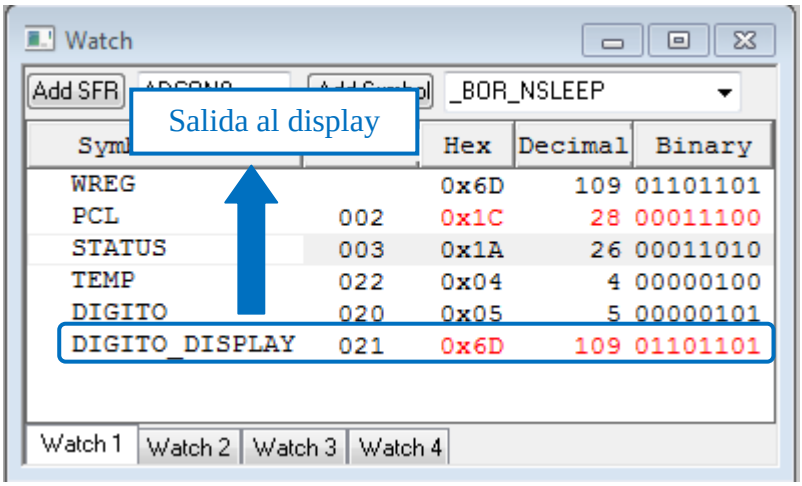


Figura 111. *Fin de la simulación del primer valor.*

Este proceso se repite dos veces más, codificando los valores 0 y 9 y los resultados obtenidos son los siguientes. Para el valor cero, (Figura 112):

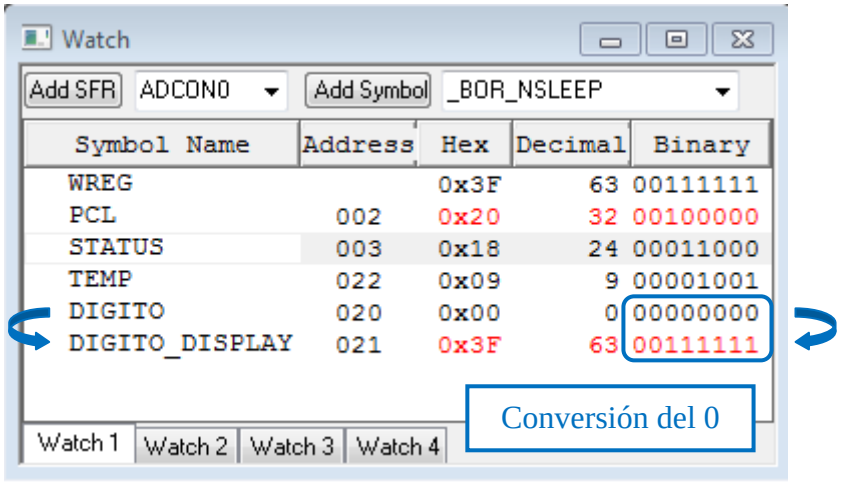


Figura 112. *Segundo dígito a codificar.*

El tercer dígito, (figura 113):

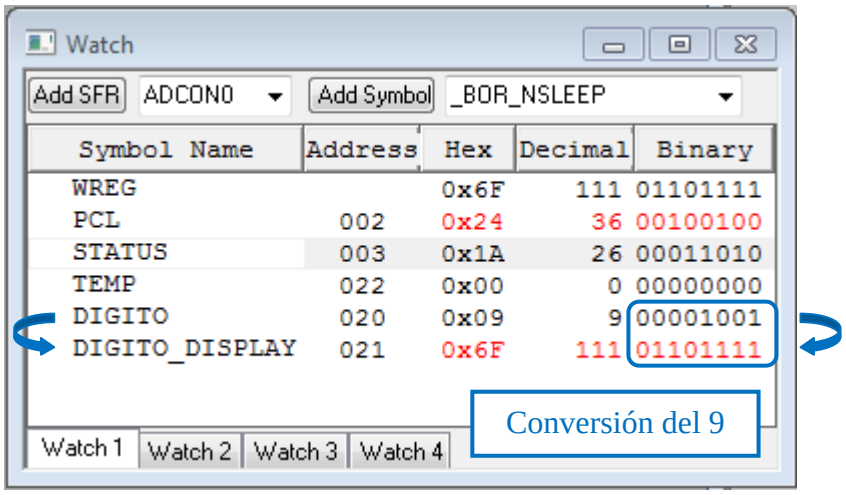


Figura 113. *Último dígito.*

1.1.5.4.3.5.3.1.2 - Subrutina tabla

↳ *Código*

```
Tabla
    movlw   maximo
    movwf   TEMP
    movf    DIGITO,W    ; Copio el contenido del puerto a W
    subwf   TEMP
    btfss   STATUS,C    ; Compruebo si la entrada es 0-9
    movlw   maximo+1    ; Si no es el número 0-9 --> W = .10
    addwf   PCL
    retlw   cero        ; LED's activados para el cero en un display
    retlw   uno         ; LED's activados para el uno en un display
    retlw   dos         ; LED's activados para el dos en un display
    retlw   tres        ; LED's activados para el tres en un display
    retlw   cuatro      ; LED's activados para el cuatro en un display
    retlw   cinco       ; LED's activados para el cinco en un display
    retlw   seis        ; LED's activados para el seis en un display
    retlw   siete       ; LED's activados para el siete en un display
    retlw   ocho        ; LED's activados para el ocho en un display
    retlw   nueve       ; LED's activados para el nueve en un display
    retlw   fallo       ; LED's activados para la E en un display
```

Figura 114. *Subrutina Tabla.*

Primero mueve a una variable auxiliar TEMP, el valor de máximo para poder comparar con el dígito y observar si el digito se encuentra entre 0 y 9, esta comparación se realiza mediante la resta subwf, (Figura 114)

TEMP = máximo
W = DIGITO
W = TEMP - W

Si *la resta es negativa*, significa que el digito es mayor que el máximo, el bit C =0 y ejecuta la siguiente instrucción de asignar a W el valor de 10, correspondiente con la posición fallo de la tabla activando una E en el display. Si la *resta es positiva* , el digito estará comprendido entre 0 y 9 por lo que C=1 saltándose la siguiente instrucción, no modificando el valor del registro de trabajo.

Por último *se añade al contador de programa el valor del digito* (10 si hay un error) ejecutando la instrucción correspondiente al digito a codificar.

- *Simulación*

Antes de entrar en la subrutina los valores de los registro más importantes se pueden observar en la figura 109. Después se pasa a la variable auxiliar el valor máximo del digito y al registro de trabajo el valor del dígito, (Figura 115):

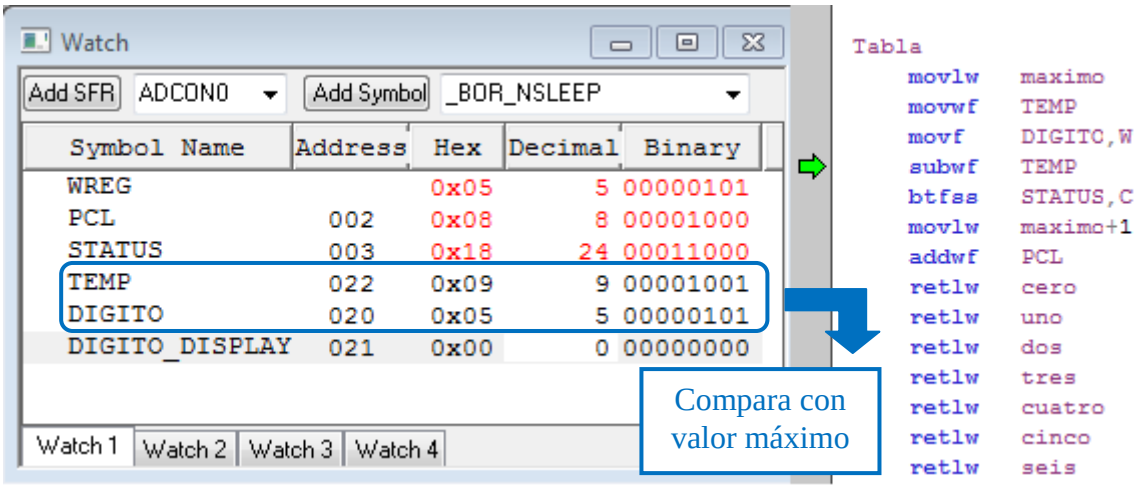


Figura 115. *Comprobación para no salirse de la tabla.*

En la figura 116 se observa la comparación mediante la instrucción sub, cual es el mayor de ambos, como el digito está entre cero y nueve, la resta es positiva por lo que se activa el bit de carry (**ver practica 1**)

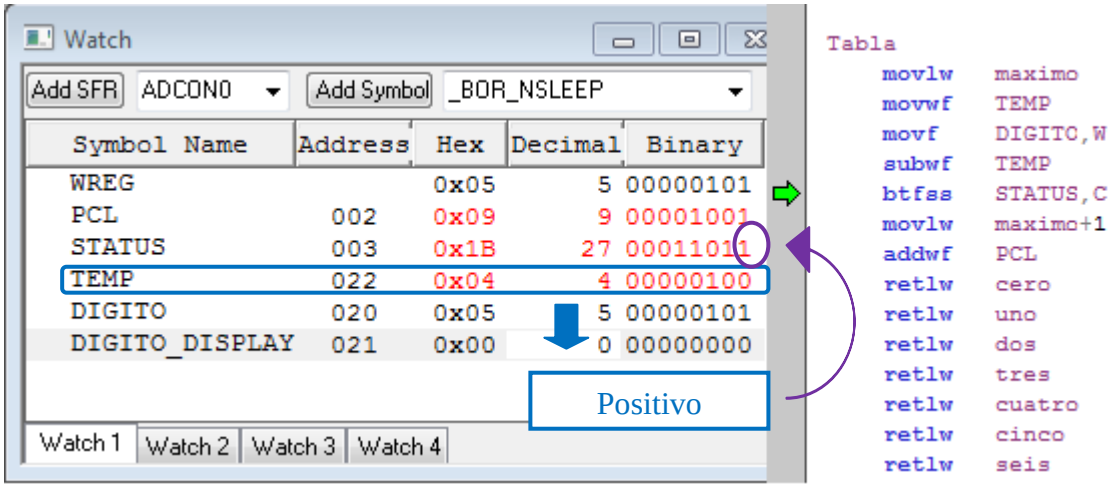


Figura 116. *Comparación correcta.*

Se testea el bit de carry, como se encuentra a uno saltara la siguiente instrucción, se puede observar como en el salto el contador de programa se ha incrementado en 2 y no en uno como es habitual, (Figura 117):

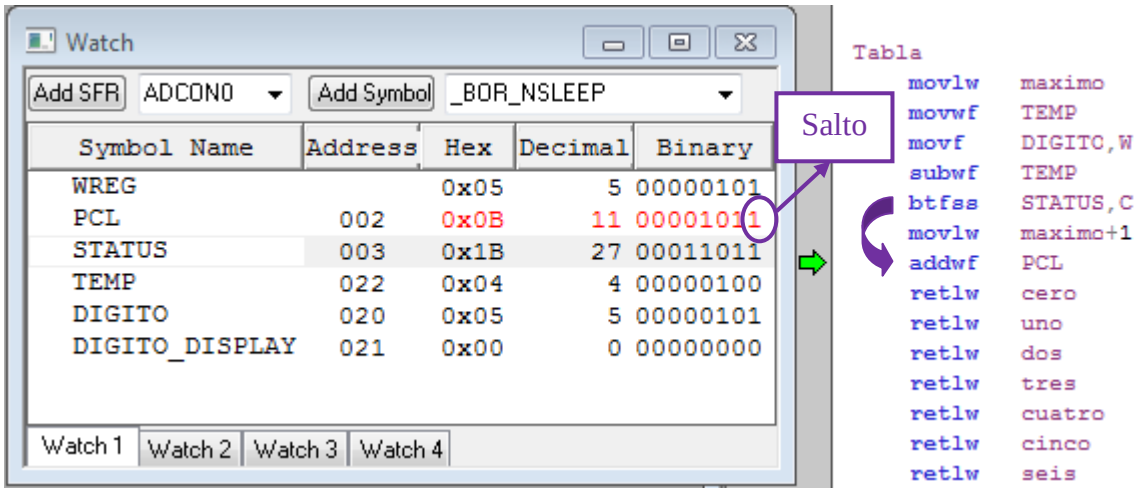


Figura 117. *Actualización del contador de programa.*

Se añade el valor del dígito al contador del programa, colocándose en la instrucción correspondiente, en este caso en la que devuelve el valor de 5 codificado en el registro de trabajo.(Figura 118):

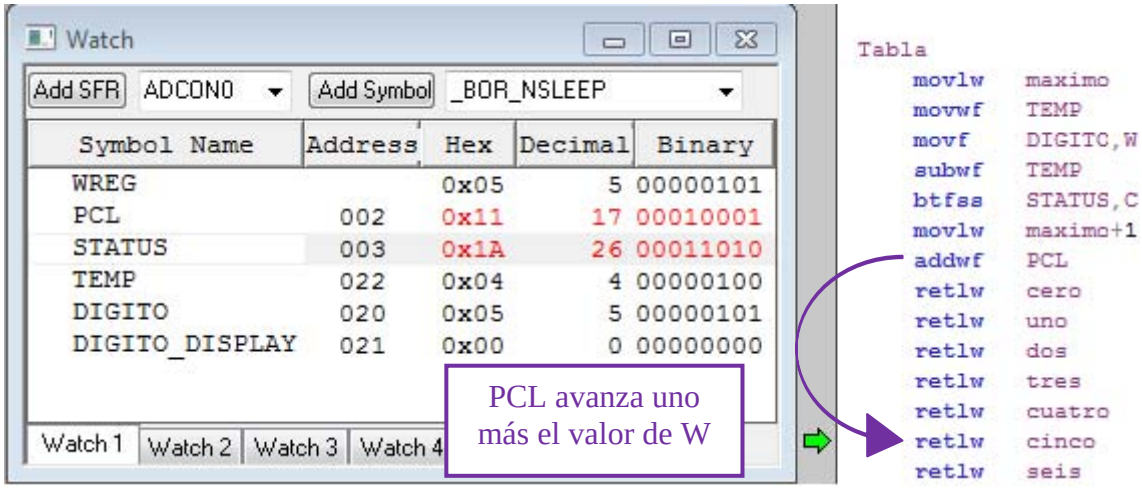


Figura 118. *Obtención de la codificación deseada.*

Y retorna al programa principal, con el valore en W (Figura 110)

1.1.5.4.3.5.3.1- Segunda parte

Las variables y constantes empleadas en este ejercicio son las siguientes, (Figura 18):

```
--------- GRABACIÓN MEMORIA RAM ----------------------

 #define dato     .20      ; Valor inicial a almacenar en RAM
 #define posiciones .10    ; Número de posiciones a ocupar en la RAM
 #define inicial 0x30      ; Primera dirección a almacenar


 CONTADOR equ 0x23           ; Variable para contar las posiciones de la RAM
 NUMERO  equ 0x24            ; variable que almacena el numero a guardar
```

Figura 119. *Registros y constantes.*

- ✓ *Dato* corresponde con el valor inicial a almacenar en memoria.
- ✓ *Posiciones* es el numero de registros que se van a ir almacenando los datos
- ✓ *Inicial* es la posición de memoria inicial donde se almacenará el primer dato

- ✓ *CONTADOR*, variable creada que indicará el número de veces a repetir el bucle tendrá inicial mente el valor de posiciones
- ✓ *NUMERO*, variable creada que almacenará el dato a guardar, inicialmente se cargara con dato y posteriormente ira decrementándose.

↳ *Código*

En este caso no existen subrutina , solamente el programa principal, (Figura 120)

```
    ;---------- PARTE B ESCRITURA EN RAM --------------

    movlw posiciones
    movwf CONTADOR      ; Cargo en CONTADOR el numero de posiciones
    movlw inicial-1
    movwf FSR           ; Cargo en el puntero la posición anterior a la inicial
    movlw dato          ; Cargo en NUMERO el dato inicial a guardar
    movwf NUMERO        ;

Ram incf FSR            ; Apunta a la siguiente posición
    movwf INDF          ; Dato --> INDF --> Dato se carga en dirección que apun
    decf NUMERO
    movf NUMERO,W       ; W = W - 1
    decf CONTADOR       ; Decremento el contador y compruebo si llega a cero
    btfss STATUS,Z      ; Si llega  a cero salta la siguiente instrucción
    goto Ram

Fin goto    Inicio      ; Fin de programa cíclico
```

Figura 120. *Escritura en RAM.*

El programa comienza cargando en CONTADOR el número de iteraciones del bucle, cargando en el puntero la posición anterior a almacenar en memoria (bucle tipo preincremento) y por último cargando en NUMERO el dato que se guardará en la primera posción de memoria.

Al comenzar el bucle se incrementa el puntero, **si es la primera iteración apuntará a la primera posición a guardar la dato**. Copiaremos el dato en el registro INDF, lo que automáticamente provocará que se copie también en la dirección a la que apunta FSR.

Después se decrementa el numero a guardar (ver enunciado) y el contador. Cuando el contador llega a cero, habrá guardado en todas las posiciones saltándose la siguiente instrucción correspondiente a realizar un nuevo paso por el bucle.

↳ *Simulación*

Antes de entrar al bucle los valores de CONTADOR, NUMERO y FSR (Figura 121):

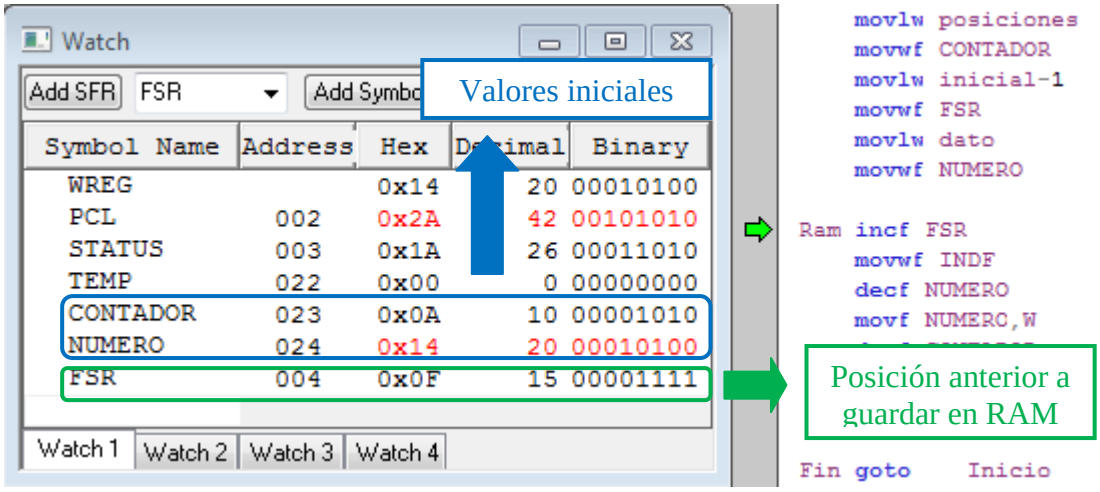


Figura 121. *Valores de los registros principales antes de entrar al bucle.*

Y la memoria RAM se encuentra vacía desde la posición 0x30 hasta la posición 0x39, (Figura 122)

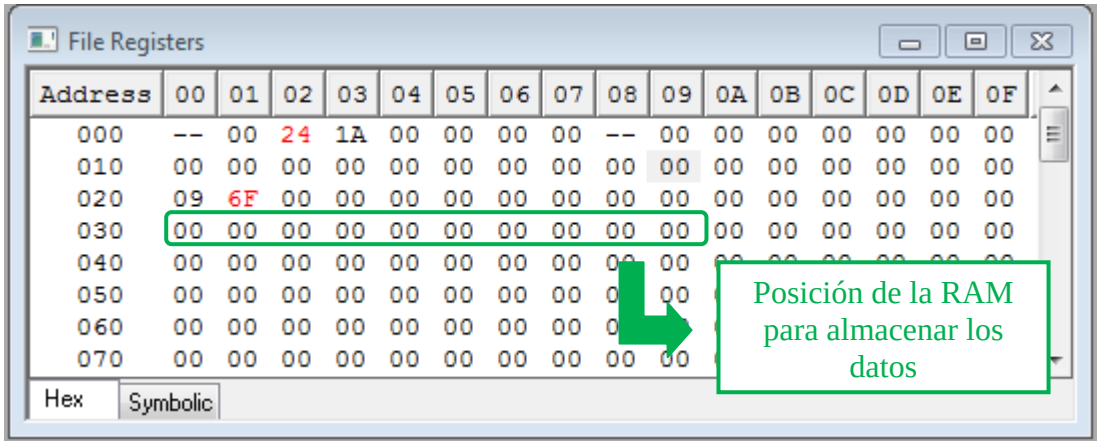


Figura 122. *Memoria RAM antes de empezar el bucle.*

Al entrar al bucle se incrementa el puntero, por lo que la dirección inicial será la 0x30, (figura 123)

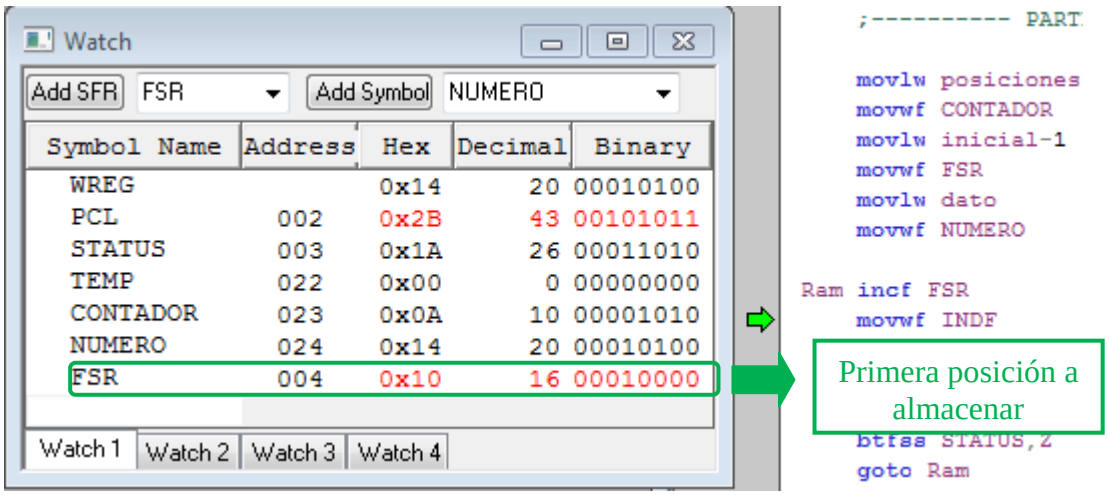


Figura 123. *Preincremento.*

En la figura 124 se observa la copia del dato, inicialmente 20 decimal, que se encuentra en W al registro INDF cambiando automáticamente el registro 0x10 al valor de 20 decimal (0x14h)

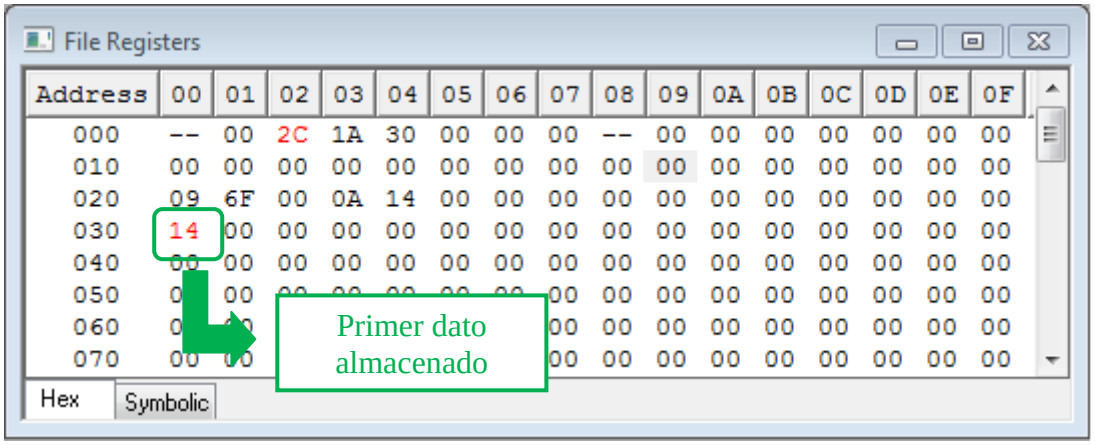


Figura 124. *Primer dato almacenado en RAM*

Se decrementa NUMERO (enunciado) y se decrementa CONTADOR, como no ha llegado a cero el bit Z se mantiene a cero, (figura 125):

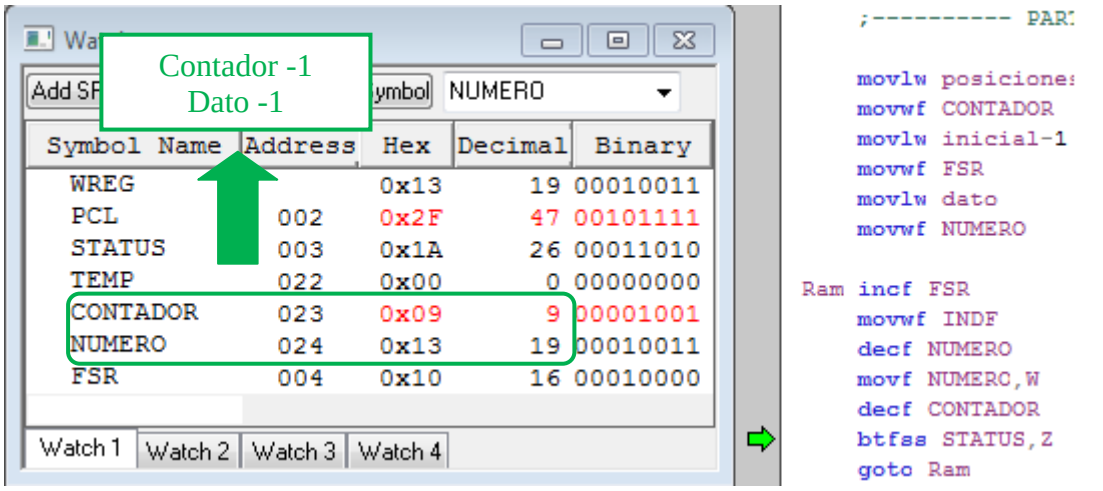


Figura 125. *Actualización de valores.*

Al testear el bit Z como se encuentra a '0' el programa no salta la siguiente instrucción volviéndose a ejecutar el bucle, almacenando el próximo valor en la siguiente dirección de la memoria de datos. (Figura 126):

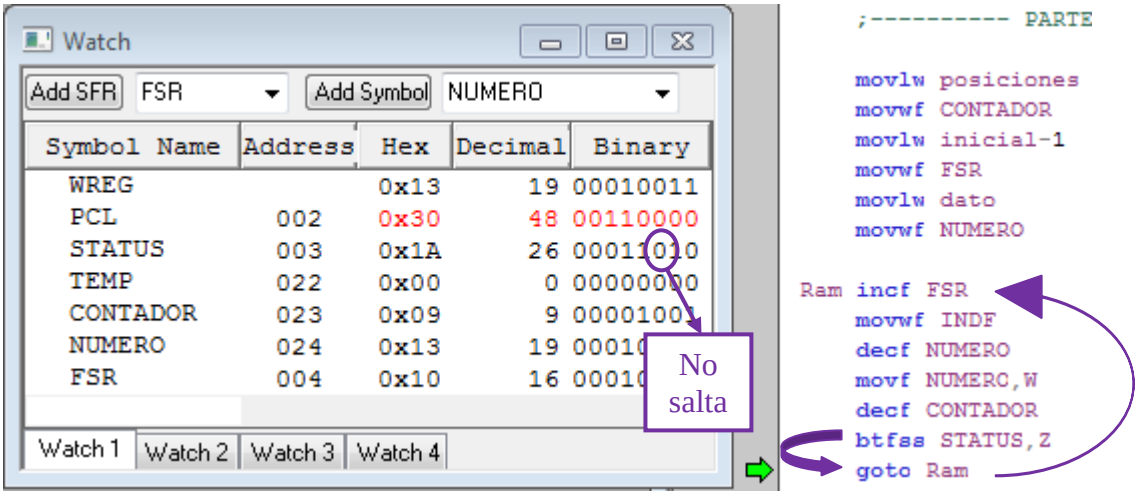


Figura 126. *Nueva iteración.*

Después de realizar las 9 iteraciones restantes la memoria de datos queda de la siguiente forma, (Figura 127):

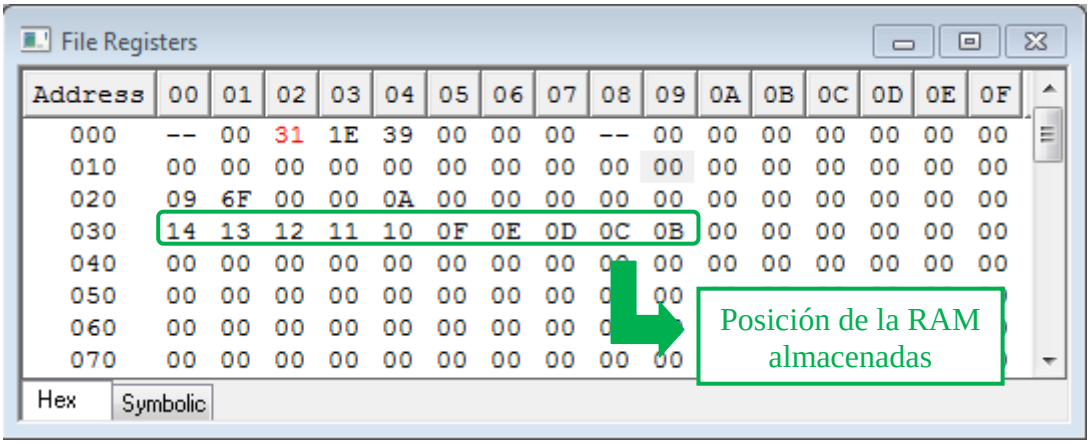


Figura 127. *Memoria RAM al finalizar el programa.*

Y las variables, (Figura 128):

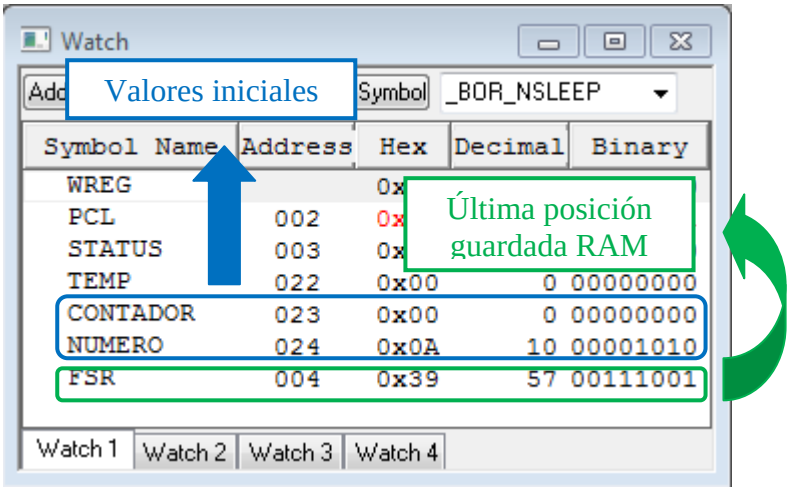


Figura 128. *Registros del programa al finalizar.*

1.1.5.4.3.6.- Ejercicio de ampliación

1.1.5.4.3.6.1- Planteamiento

Como se especifica en el enunciado crearemos dos registros, el primero se llamará PUERTOA que será el valor introducido, a la hora de programar se simulará la entrada con la instrucción movlw . El otro registro será el PUERTOB que simulará la salida a un display de 7 segmentos.

Nota: No usar la notación PORTA o PORTB al ser registros existentes referenciados a los puertos a y b.

La primera parte del programa es hacer que el valor del PUERTOA aparezca en un display a través del PUERTOB. Para ello será necesario la creación de una tabla que relacione la entrada con la salida, es decir los leds que se necesitan encender, para cada número.

Por último se creara una función capaz de modificar las posiciones de memoria desde 0x30 hasta 0x3F en función del dato introducido, esto se realizara mediante direccionamiento indirecto , explicado en el contenido teórico.

1.1.5.4.3.6.2- Diagrama de flujo

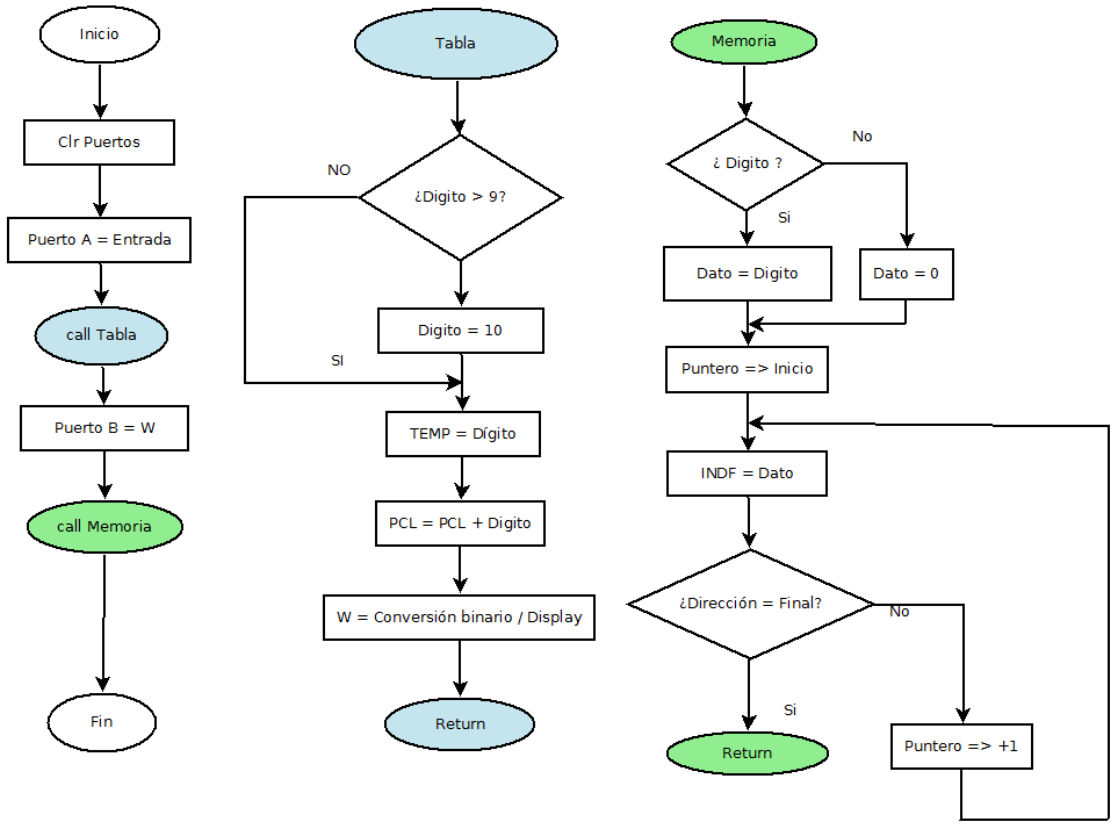


Figura 129.- *Diagrama de flujo del ejercicio de ampliación.*

El programa comienza limpiando los puertos, a modo de simulación (Figura 129). Después se copia el valor de una entrada al registro PUERTO A. Si el puerto A sería un puerto normal, este paso se realizaría automáticamente, mediante una entrada externa sin tener que programarlo,

Una vez que se tiene la entrada en el puerto, se llama a la función tabla, la única diferencia con el diagrama anterior (Figura 4), es la *introducción de una variable auxiliar TEMP, que será usada en la subrutina Memoria para poder comprobar si la entrada es un dígito o no.*

Al regresar al programa principal , la salida de la subrutina tabla se copia al puerto B, teniendo en el puerto el número codificado para activar un display de siete segmentos.

Por último se llama a la función memoria.

Lo primero que realiza la subrutina memoria es chequear si la entrada es un *dígito del cero al nueve,* en ese caso el valor a almacenar será el propio dígito, sino se realizara un borrado de la memoria para ello simplemente el dato a almacenar será un cero.

Después se ejecutará el mismo bucle que en el anterior ejercicio (figura 5) con la salvedad que en vez de realizar un preincremento del puntero, **se realiza un postincremento**, luego la primera dirección del puntero será la inicial y no la anterior a la inicial. Y en vez de usar un contador para saber cuántas direcciones se han grabado en la memoria RAM, se compara la dirección actual con la final y si coinciden se acaba el bucle.

En el diagrama de la figura 28, El programa principal solo tiene una entrada, habría que repetirlo para la segunda entrada.

1.1.5.4.3.6.2- Programación y simulación

Las variables y constantes empleadas en este programa se muestran en la figura 130:

```
#define entrada1    .2      ; Primer valor introducido en el puerto A
#define entrada2    .15     ; Segundo valor introducido en el puerto A

#define inicio      0x30    ; Primera dirección a almacenar
#define fin         0x3F    ; Ultima dirección a borrar
#define maximo      .9      ; Valor máximo para comparar en la tabla

- Display siete segmentos  ---

#define cero    b'00111111'
#define uno     b'00000110'
#define dos     b'01011011'
#define tres    b'01001111'
#define cuatro  b'01100110'
#define cinco   b'01101101'
#define seis    b'01111101'
#define siete   b'00000111'
#define ocho    b'01111111'
#define nueve   b'01101111'
#define fallo   b'01111001'     ; Letra E

PUERTOA equ 0x10    ; Puerto de entrada
PUERTOB equ 0x11    ; Puerto de salida
TEMP    equ 0x12    ; Registro temporal
```

Figura 130. *Registros y variables.*

- ✓ *Entrada 1, entrada 2*: Los valores de los dígitos.
- ✓ *Inicio, Fin*: Dirección inicial y final a almacenar en la memoria de datos.
- ✓ *máximo*: utilizada para comprobar si es un digito.
- ✓ *Cero,...,fallo*: codificaciones del display
- ✓ *PUERTOA, PUERTO B*: Registros que simulan los puertos.
- ✓ *TEMP*: Variable auxiliar usada en la subrutina de escritura en la RAM

↳ *Código*

A continuación (Figura 131) se explicará el programa principal, tanto la subrutina tabla como la subrutina Memoria son similares a los realizados en el ejercicio básico de esta práctica.

```
Inicio                  ; Etiqueta de comienzo de programa

    clrw
    clrf    PUERTOA     ;Inicializo los puertos
    clrf    PUERTOB

    movlw   entrada1    ; Simulación en código de una entrada
    movwf   PUERTOA     ; No sería necesario con puertos reales

    call    Tabla       ; Conversión de la entrada al valor del display
    movwf   PUERTOB     ; Valor de salida al display
    call    Memoria     ; Escritura en la RAM mediante el puntero

    movlw   entrada2    ; Simulación de una nueva entrada
    movwf   PUERTOA

    call    Tabla       ; Conversión de la entrada al valor del display
    movwf   PUERTOB     ; Valor de salida al display
    call    Memoria     ; Escritura en la RAM mediante el puntero

Fin goto    Fin     ; Fin de programa cíclico
```

Figura 131. *Programa principal.*

Se comienza inicializando los registros a cero. Después se copia el primer dígito al registro PUERTO A, se trata de una simulación. **Si fuera el puerto A verdadero este paso sería innecesario**.

Con el valor del puerto A se entra en la subrutina Tabla que devolverá en el registro de trabajo el dígito codificado en 7 segmentos, y se copiará al registro PUERTO B.

Después se entra en la subrutina memoria que grabará desde la posición 0x30h hasta la posición 0x3F el digito del puerto A.

Posteriormente entra un segundo dato y se repetirá el mismo proceso, como el dato no es un dígito (15) en el display aparecerá una E de error y realizará un borrado de las posiciones de memoria 0x30 hasta la 0x3F.

Antes de entrar a la función tabla los valores de los registros más importantes son los siguientes, (Figura 132)

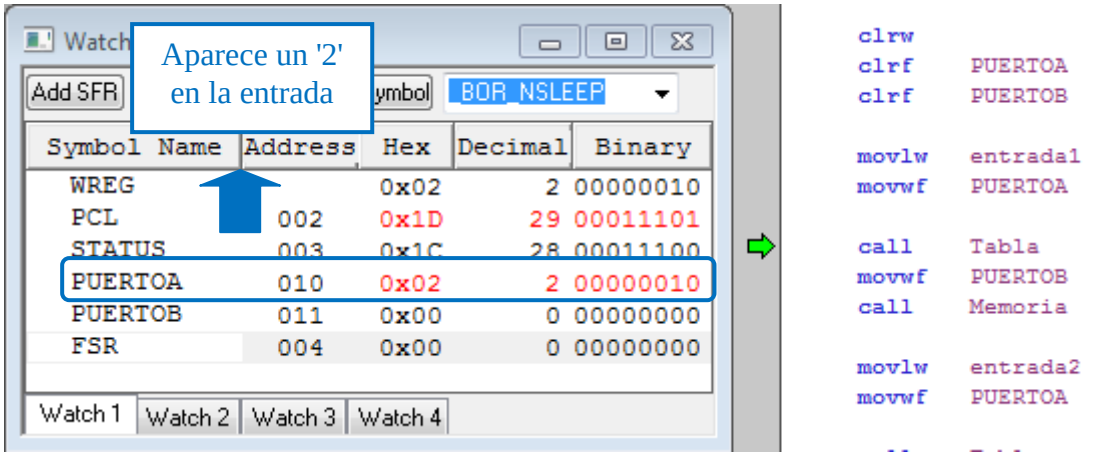


Figura 132. *Valores de los registros antes de entrar en la subrutina.*

Y la memoria RAM en la figura 133:

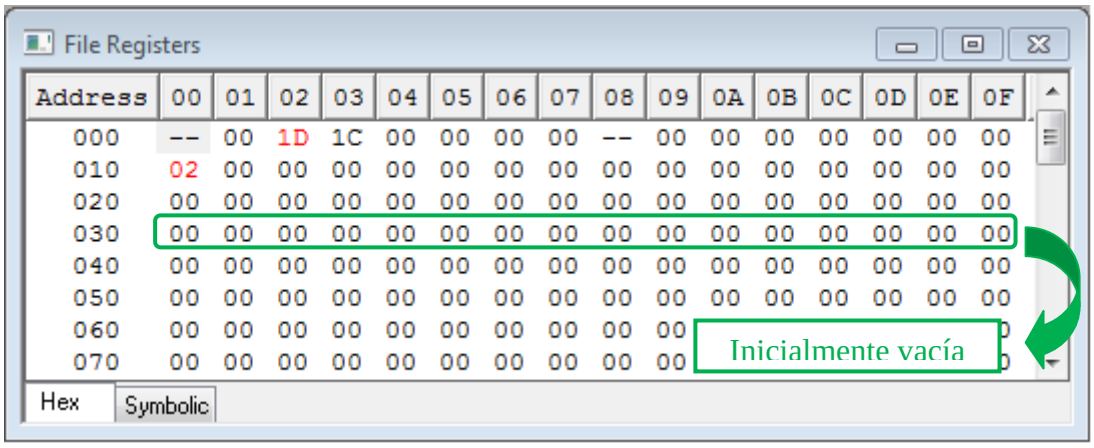


Figura 133. *Memoria RAM antes de entrar en la subrutina.*

En la salida de la subrutina Tabla se escribe en el registro de trabajo la codificación a 7 segmentos del dígito 2, y se copiara en PUERTO B, (figura 134):

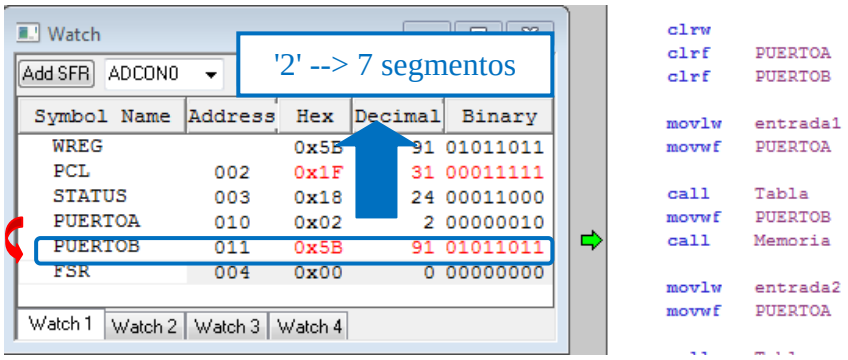


Figura 134. *Retorno de subrutina Tabla.*

Después de ejecutarse la subrutina memoria se grabara el dígito 2, (figura 135):

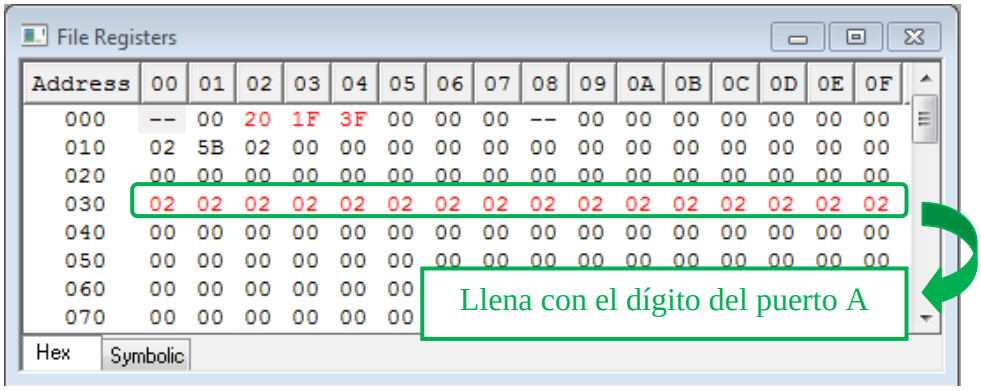


Figura 135. *Retorno de subrutina Memoria.*

La segunda entrada (15), al no ser un dígito borra la RAM. (Figura 136, y 137):

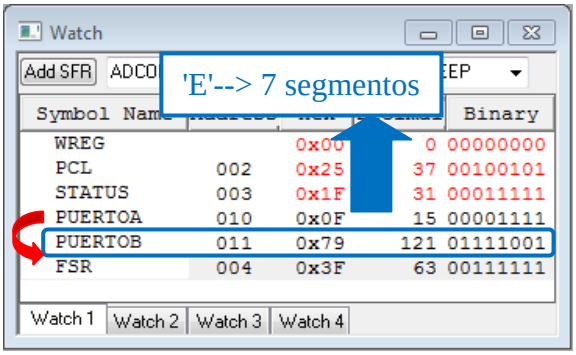


Figura 136. *Retorno de subrutina Tabla.*

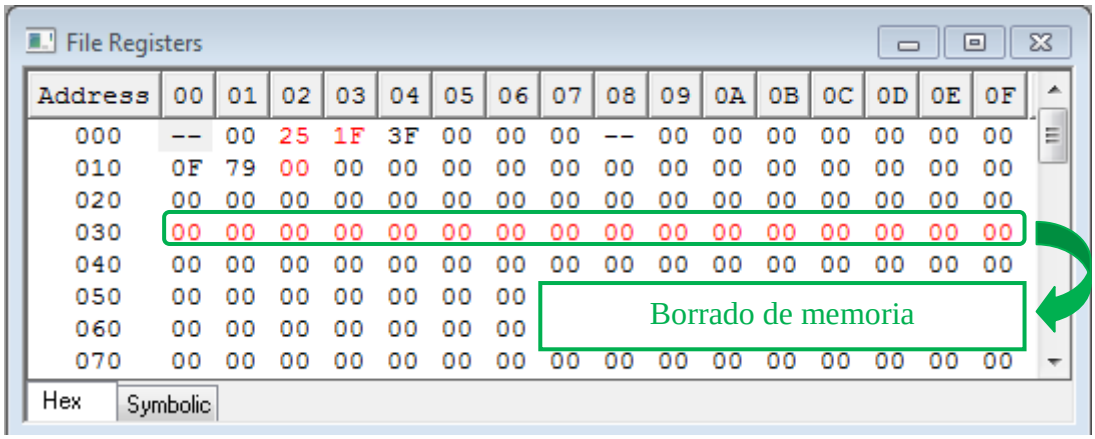


Figura 137. *Retorno de subrutina Memoria.*

1.1.5.4.3.7.- Programas propuestos

Diseñar un programa que borre las posiciones de la memoria RAM desde la 0x30 hasta la 0x4F mediante direccionamiento indirecto.

Crear una subrutina que codifique las 4 primeras letras del abecedario en un display de 7 segmentos, si la entrada vale 0 aparecerá una A en el display y así sucesivamente según la siguiente tabla.

Tabla 10. *Relación entrada/salida.*

Display	*Entrada*
A	*00h*
B	*01h*
C	*02h*
D	*03h*
8	*Resto*

1.1.5.4.4. - Práctica 4: Puertos I/O.

1.1.5.4.4.1.- Objetivo

- ✓ Manejo de los puertos del PIC16F886 en modo entrada y modo salida.
- ✓ Realización de prácticas en el laboratorio USB-PIC'School.
- ✓ Comprender las diferencias entre los registros PORT y TRIS.
- ✓ Manejar los registros ANSEL y ANSELH.

1.1.5.4.4.2.- Enunciado

Realizar un programa, que genere una salida en el puerto c (RC0-RC7) en función de dos señales de entrada, RA0 y RA1, y el estado del puerto b (RB0-RB7), según se indica en la tabla 11:

Tabla 11. *Relación entradas/salidas.*

IN		OUT
RA1	RA0	PORT C
0	0	PUERTO C = 00h
0	1	PUERTO C = FFh
1	0	PUERTO C = PUERTO B
1	1	PUERTO C = NOT(PUERTOB)

1.1.5.4.4.3.- Contenido teórico

↳ *Registros TRIS*

El pic16f886 dispone de 4 puertos,(ver *otros anejos, PIC16F886*) de los cuales tres están completos (8 bits) y el cuarto sólo posee un bit:

- ★ Puerto A : RA0-RA7
- ★ Puerto B: RB0-RB7
- ★ Puerto C: RC0-RC7
- ★ Puerto E: RE3

Cada pin puede actuar como entrada o como salida según se programa, luego se dispone de un total de 25 entradas o salidas para usar.

Para poder usar los puertos es necesario programarles primero, indicando en cada pin si se comportará como entrada o como salida. Para ello se utilizarán los registros *TRISA(0x85h), TRISB(0x86h), TRISC(0x87h), y TRISE(0x89h)* cada bit de estos registros configura a sus puertos correspondientes de la siguiente forma:

- ★ '1' : El pin correspondiente actúa como salida.
- ★ '0' : El pin correspondiente actúa como entrada.

No hay que olvidarse que **los registros TRIS se encuentran en el banco 1** para lo cual los bits 5 y 6 del registro STATUS para poder programar los registros deberán estar:

- ★ RP0 = 1
- ★ RP1 = 0

En el siguiente ejemplo se muestra como programar el puerto C de tal forma que los bits RC7-RC4 actúen como salida y el resto como entrada. (Figura 138):

```
bsf STATUS,RP0
movlw   b'11110000'
movwf   TRISC
bcf STATUS,RP0
```

Figura 138. *Configuración de puertos.*

Es importante a la hora de programar los puertos tener en cuenta que los pines **RA7 y RA6 no se pueden programar como entrada.** [5]

Si se está en el banco 1 y en vez de usar el registro TRISA, B, C o E se usan los registros PORTA,PORTB,PORTC o PORTE el PIC entiende que se están

configurando los puertos como entradas o salidas, es decir, **manipular un puerto en el banco 1 implica manipular el registro TRIS de dicho puerto**.

Los registros PORTA(0x05h), PORTB(0x06h), PORTC(0x07h) y PORTE(0x09h) indican el contenido de los puertos y pueden ser leídos en cualquier momento y escritos si se trata de puertos de salidas con las mismas instrucciones que operan el resto de registros, *mov*, *btf*, etc.

- *Registro ANSEL y ANSELH*

El PIC16F886 contiene entradas y salidas analógicas en algunos de los pines del puerto A y puerto B (Ver otros *anejos, PIC16F886*), por lo que si se van a usar dichas entradas es necesario indicar si dichos pines funcionarán como entradas digitales o analógicas.

En este caso será entradas digitales por lo que es necesario poner los registros ANSEL(0x188h) y ANSELH(0x189h) a cero. Estos registros se encuentran situados en el banco 3, por lo que los bit RP0 y RP1 del registro STATUS deberán estar a '1' para trabajar en el banco 3, (Figura 139):

```
bsf STATUS,RP0
bsf STATUS,RP1      ; Banco 3
clrf    ANSEL       ; Entradas digitales
clrf    ANSELH
```

Figura 139. *Configuración entradas digitales.*

1.1.5.4.4.4.- Ejercicio

1.1.5.4.4.4.1.- Planteamiento

Según el estado de dos entradas conectadas al puerto A (RA0 y RA1) el puerto C será borrado, puesto a uno, será igual que el puerto B o su complementario.

Para ello será necesario programar los registros TRISA, TRISB Y TRISC indicando si se comportan como salida o como entrada:

- ★ PUERTO A (TRIS A) : Entrada
- ★ PUERTO B(TRIS B): Entrada
- ★ PUERTO C(TRIS C): Salida

Las entradas y salidas *son entradas digitales*, por lo que será necesario programarlo realizando un clear de los registros ANSEL y ANSELH.

Para la programación del ejercicio se ha optado por realizar una *máscara del puerto A con el b'00000011' consiguiendo poner a cero todos los bits salvo RA0 y RA1* que mantendrán su estado, como se explica en la Tabla 12.

Tabla 12. *Máscara AND.*

PORT A	X	X	X	X	X	X	X	X
MÁSCARA	0	0	0	0	0	0	1	1
RESULTADO	0	0	0	0	0	0	X	X

Por lo que se obtendrá un valor entre 0 y 3 que se usará para añadir al contador del programa de forma similar al manejo de las tablas (Ver práctica 3).

Otra manera de resolver este ejercicio, es realizar 4 máscaras, correspondientes a cada posible valor de RA0 y RA1, testear si se activa el flag Z en cada una de ellas y mediante un "goto" programar cada uno de los casos a modo de *"case"*.

1.1.5.4.4.4.2.- Diagrama de flujo

El programa principal comienza borrando los puertos, asegurándonos que no tiene valores no deseados, " *borra los latch de salida del puerto.*"

Después se *configurar los puertos* indicando si son de salida o de entrada y puertos digitales. (Figura 140)

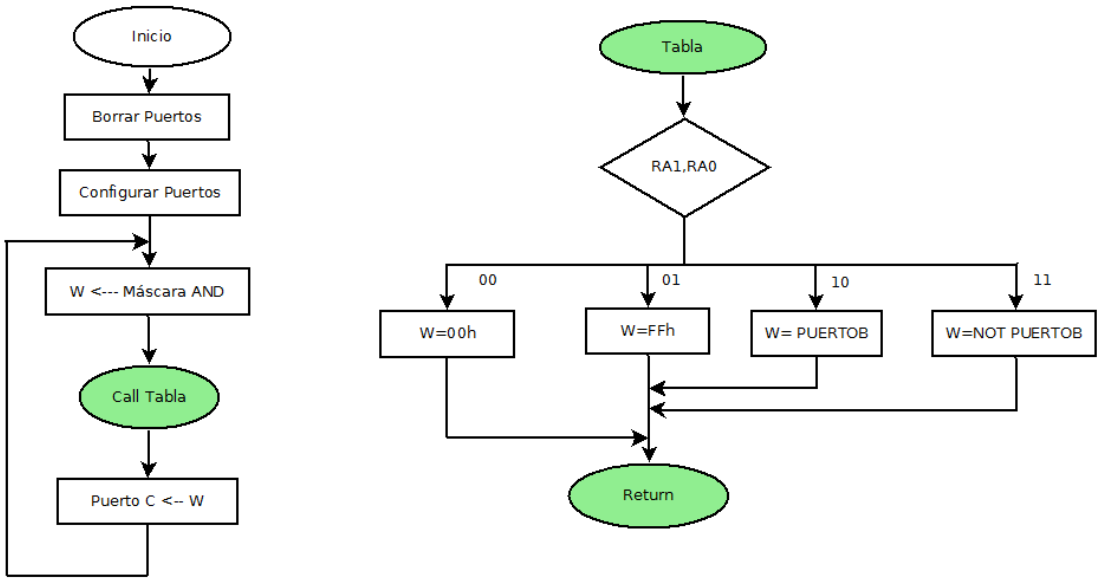


Figura 140. *Diagrama de flujo.*

Se realiza una *máscara con el puerto A* que se será almacenada en el registro de trabajo, llamando posteriormente a la subrutina Tabla.

La subrutina en función del resultado de la máscara devolverá en el registro de trabajo el valor deseado para el puerto C, por lo que al retornar de la función simplemente basta con copiar el registro de trabajo en el puerto C que funciona como salida.

Por último se trata de un **programa cíclico** por lo que se volverá al punto del programa donde se realiza la máscara para mantener constantemente actualizado el puerto C.

1.1.5.4.4.4.3.- Programación y simulación

Para la realización del programa no ha sido necesario el uso de ningún registro adicional, simplemente se usa el registro de trabajo, y los registros de configuración de los puertos y contenidos de los mismo. Por ello no es necesario declarar ninguna constante ni variable.

1.1.5.4.4.4.3.1 - Programa principal

↳ *Código*

```
Inicio                  ; Etiqueta de comienzo de programa

;--- Configuración de los puertos ---
    clrw
    clrf    PORTA       ; Se borran los latch de salida del puerto
    clrf    PORTB
    clrf    PORTC
    bsf STATUS,RP0
    bsf STATUS,RP1      ; Banco 3
    clrf    ANSEL       ; Entradas digitales
    clrf    ANSELH
    bcf STATUS,RP1      ; Banco 1
    movlw   0xFF        ;
    movwf   PORTA       ; Puerto A (In)
    movwf   PORTB       ; Puerto B (In)
    clrf    PORTC       ; Puerto C (Out)
    bcf STATUS,RP0      ; Banco 0

;--- Programa ---

Leer    movf    PORTA,W
        andlw   b'00000011' ; W = 0000 00XX
        call    Tabla       ; Devuelve en W el estado del puerto C
        movwf   PORTC       ; Copio en el Puerto C el registro W
        Goto    Leer
```

Figura 141. Código del programa principal.

La primera parte del programa sirve para configurar los puertos A,B y C explicada en el contenido teórico de esta práctica.

El programa propiamente dicho es un bucle infinito que continuamente *está realizando una máscara con el puerto A* , el valor de la máscara será almacenado en W usado en la subrutina Tabla para indicar que dato debe sacar el puerto C, ese dato es devuelto en el registro de trabajo y se copiará en el puerto C. Después se vuelve a realizar el bucle manteniendo así actualizado el puerto C.(Figura 141)

Aunque en este programa los bit RA7-RA2 deberían estar a cero, se usa la máscara en vez de copiar el puerto A al registro de trabajo, para asegurarnos siempre que son cero.

↳ *Simulación*

Para la simulación de esta práctica se opta por usar la ventana *Stimulus* (ver *otros anejos, MPLAB*) colocando cada una de las entradas usadas de los puertos (Figura 142):

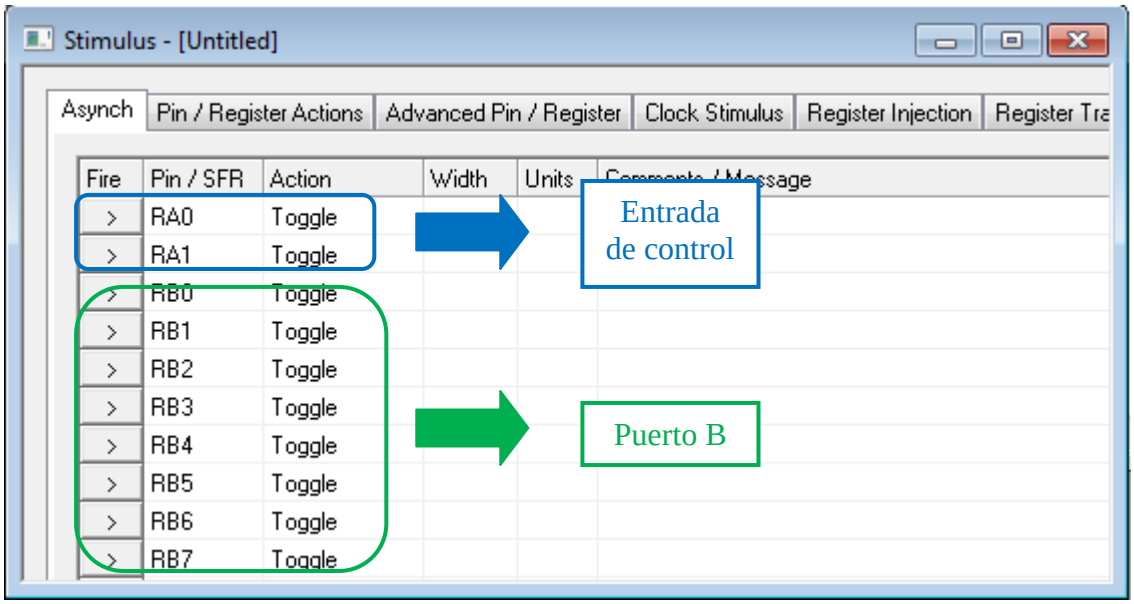


Figura 142. *Ventana Stimulus.*

Mediante el *Stimulus* en la simulación se le va asignar al puerto B un valor constante de b'10101100' y se va ir cambiando las entradas del puerto A para observar los 4 casos posibles.

Para modificar los puertos basta con hacer click en la columna Fire en el pin que se desee modificar, la forma más sencilla de programar esta ventana es usando la acción "*toogle*" de tal forma que cada pin esté conectado a un interruptor.

También se puede configurar el pin como un pulsador, indicándole en la columna Width y en la columna Units el tiempo que estará activo el pulsador.

Después de la configuración de los puertos los registros deberían estar, (Figura 143)

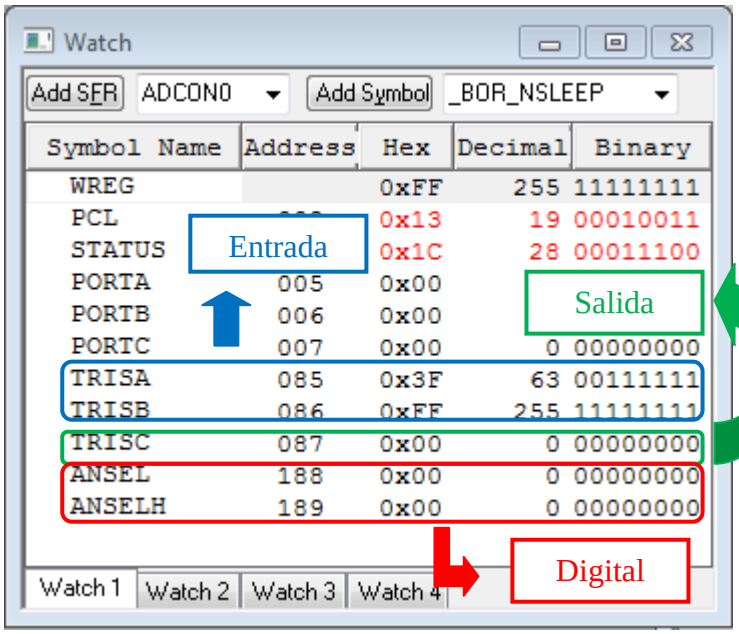


Figura 143. *Configuración de puertos.*

Realizada la máscara antes de entrar en la subrutina Tabla y mediante Stimulus dando el valor al puerto B, (figura 144):

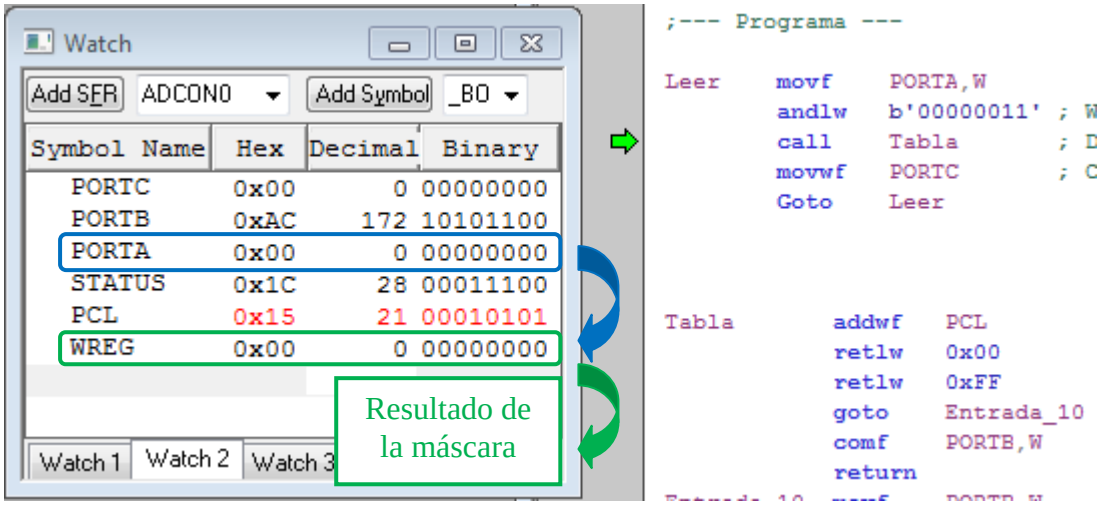


Figura 144. *Registros antes de entrar en la subrutina Tabla.*

A continuación se muestran los resultados obtenidos de la simulación para las distintas entradas del puerto a. (Figura 144,145,146 y 147):

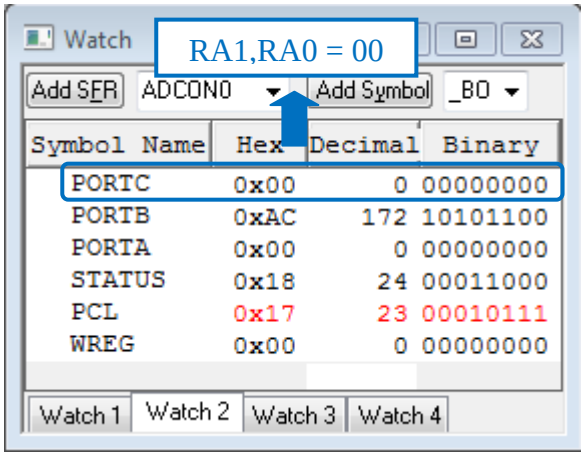


Figura 145. *Caso 00.*

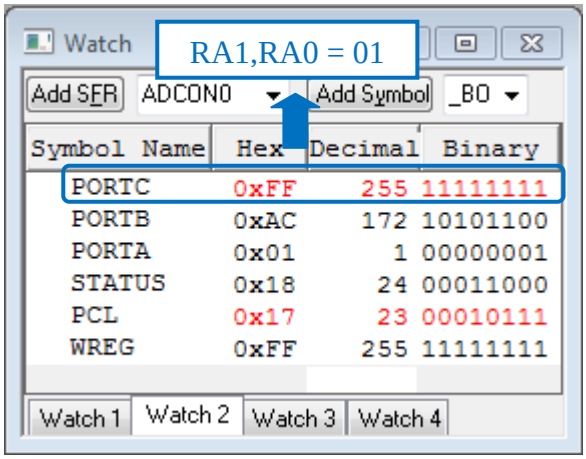


Figura 146. *Caso 01.*

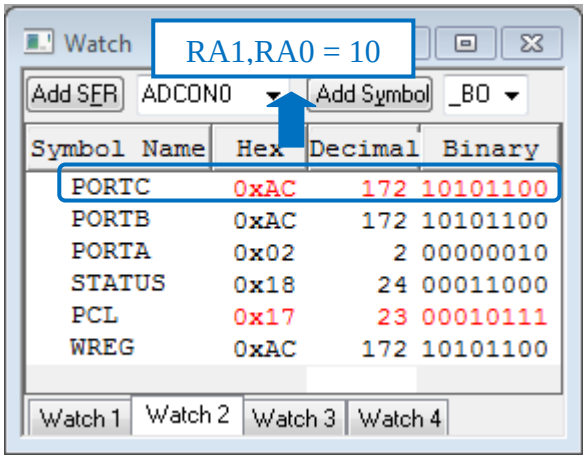


Figura 147. *Caso 10.*

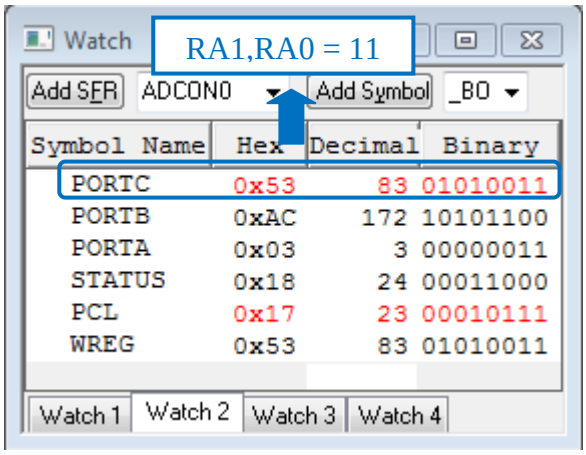


Figura 148. *Caso 11.*

1.1.5.4.4.4.3.2 - Subrutina tabla

- *Código*

```
Tabla       addwf   PCL         ; Añado al contador del Programa la máscara
            retlw   0x00        ; 00 --> W = 00h
            retlw   0xFF        ; 01 --> W = FFh
            goto    Entrada_10  ; 10
            comf    PORTB,W     ; 11 --> W = NOT (PORTB)
            return
Entrada_10  movf    PORTB,W     ; 10 --> W = PORTB
            return
```

Figura 149. *Subrutina Tabla.*

Al entrar en Tabla, W tiene el contenido de la máscara con el puerto A, según el estado que se encuentren RA0 y RA1 tendrá el valor de 0,1,2 ó 3. Ese valor se añadirá el contador del programa (PCL) para así devolver en w el valor deseado. (Figura 149)

En el caso de que W = 2, no se puede usar la instrucción 'retlw' para retornar en w con el valor de un registro, así que es necesario usar un goto para copiar el puerto B en W (al requerir 2 instrucciones si W = 3 sin el goto iría a la instrucción return y no a la instrucción comf).

- *Simulación*

En la **práctica 3** se explica se explica detalladamente cómo funciona la simulación de una tabla programada en ensamblador y los resultados de la subrutina se encuentran detallados en la simulación del programa principal.

1.1.5.4.4.4.4.- Montaje

El USB-PIC'School dispone de 8 entradas digitales de las cuales 4 son interruptores (E0-E3) y 4 son pulsadores (E4-E7).(Figura 150)

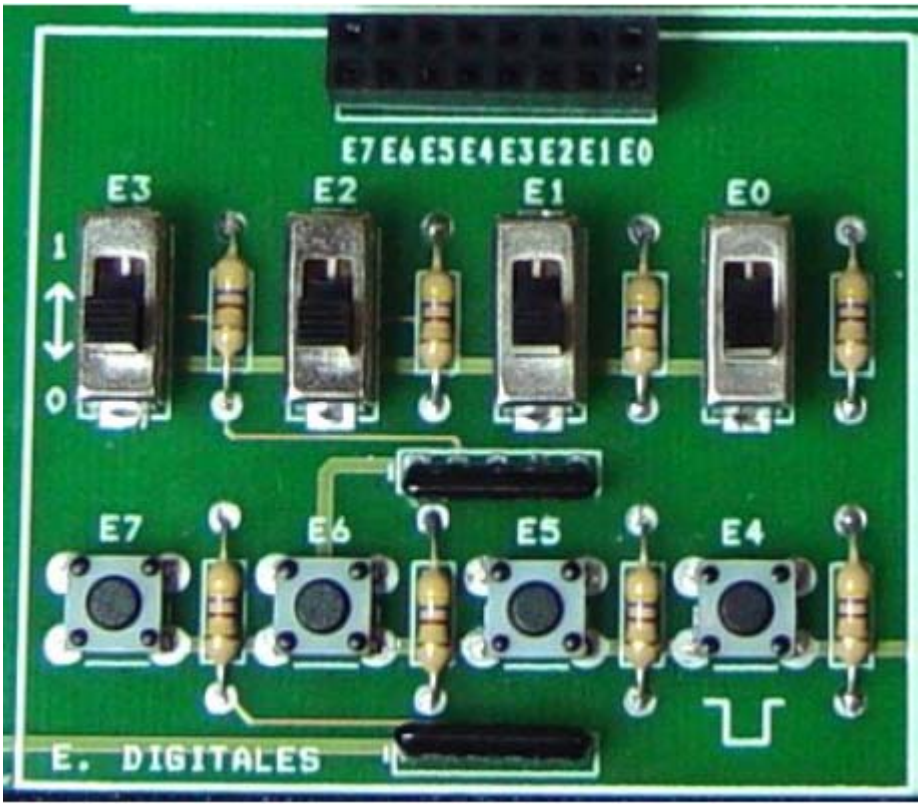


Figura 150. *Entradas del PIC'School. [5]*

Como salidas dispone de 8 Leds correspondientes a S0-S7, (Figura 151)

Figura 151. *Salidas del PIC'School. [5]*

Sin embargo en el programa es necesario disponer de 10 entradas, 8 en el puerto B y 2 en el puerto A. Por ello simplemente se va a implementar el puerto B como un puerto de 6 bits desde el RB7-RB2, dejando las entradas RB0, RB1 sin conectar, (Figura 152)

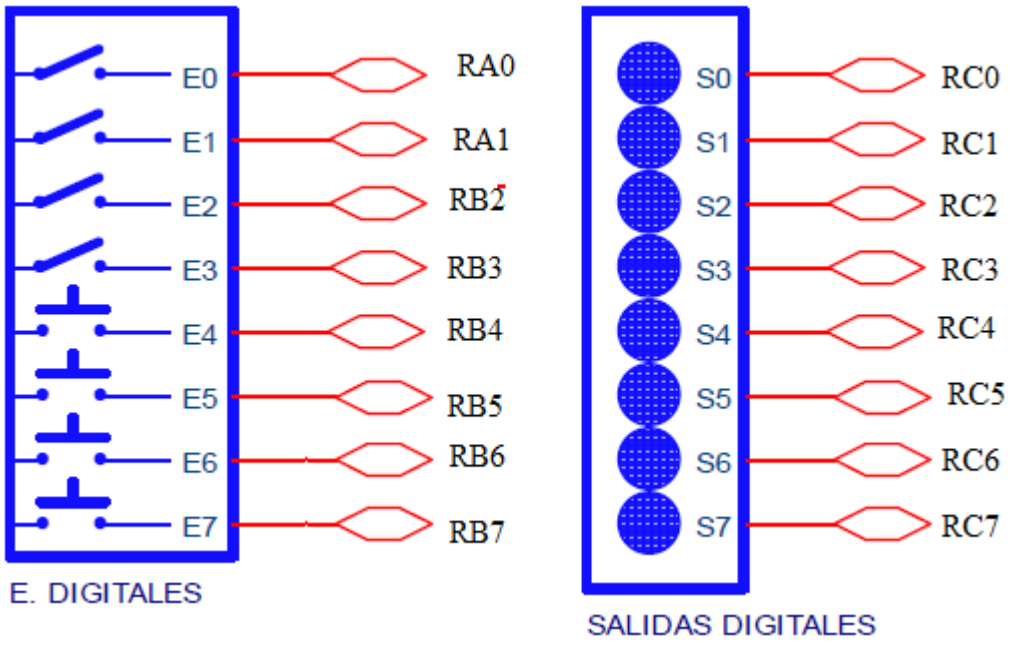


Figura 152. *Esquema de montaje.*[5]

1.1.5.4.4.5.- Programas propuestos

Realizar un programa contador de piezas, en una cinta transportadora hasta llenar una caja. Se colocará un sensor óptico(RA0) que irá generando pulsos según vayan llegando piezas, cuando el número de piezas llegue a 24, un robot(RB0) moverá la caja hasta que se active un final de carrera(RA1). En ese momento otro robot(RB1) colocará un nueva caja hasta activar un nuevo final de carrera(RA2). En ese momento la cinta transportadora se pondrá en marcha(RB2). Se incorporará un botón de paro de emergencia(RA3).

1.1.5.4.5.- Práctica 5: Temporizadores y contadores

1.1.5.4.5.1.- Objetivos

- ✓ Comprender y programar los distintos temporizadores del PIC16F886.
- ✓ Estudiar los temporizadores a modo de contadores.
- ✓ Comprender el funcionamiento de los registros que intervienen en la temporización.
- ✓ Calcular el valor inicial del temporizador interno para programar una cuenta determinada.

1.1.5.4.5.2.- Enunciado

Programar un juego de luces de 8 leds. Se asignarán los leds al puerto B, el programa comenzará luciendo el led RB0 y cada segundo en función de la entrada RA0 lucirá el led situado a la izquierda del que lució si está a nivel alto o el de la derecha si esta a cero.

1.1.5.4.5.3.- Contenido teórico

El PIC16F886, dispone de tres temporizadores denominados TMR0, TMR1 y TMR2. A continuación se explican cada uno de los temporizadores:

- ↳ *Timer 0*

El TMR0 es un temporizador/contador de 8 bits con las siguientes características:

- ✓ Registro 8 Bits (TMR0).
- ✓ Prescalado de 8 bits compartido con el WDT.
- ✓ Fuente de reloj externa o interna programable.
- ✓ Genera interrupción en el desbordamiento.
- ✓ Selección del flanco externo de reloj programable.

El funcionamiento de este temporizador se explica en el siguiente diagrama, (Figura 153):

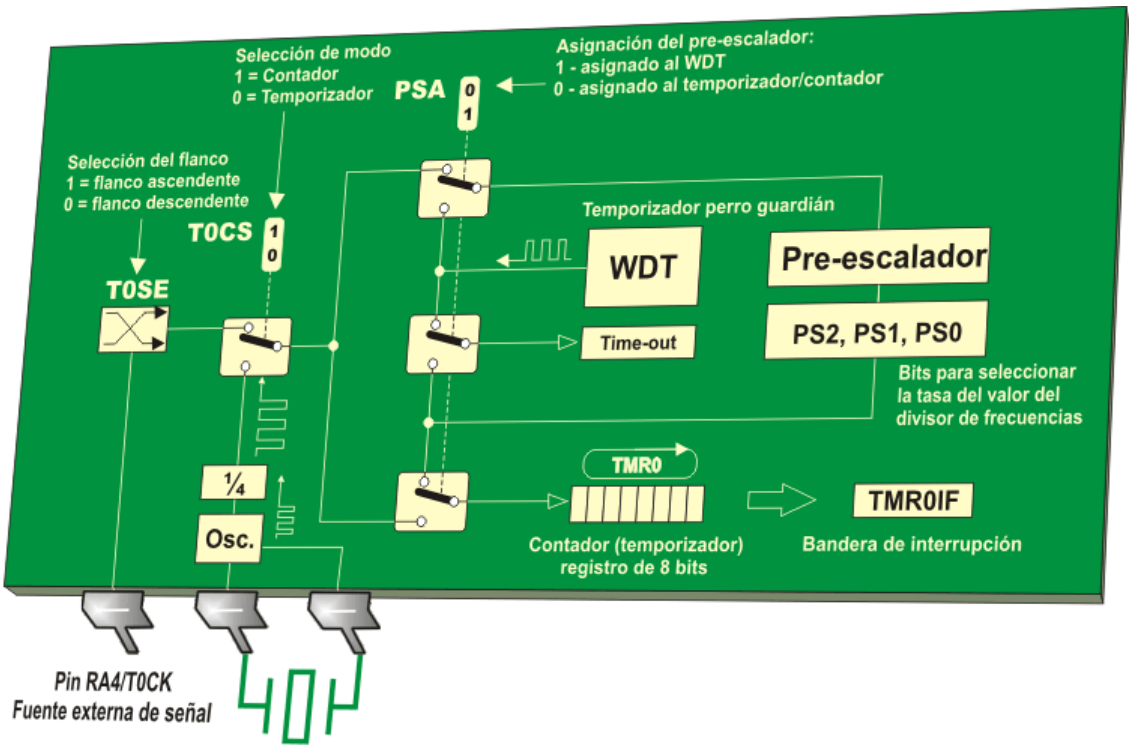


Figura 153. *TMR0.* [6]

El registro OPTION_REG es el encargado de gobernar el TMR0, su funcionamiento se explica en la Tabla 13:

Tabla 13. *Registro OPTION.* [7]

REGISTRO OPTION							
BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

★ RBPU: Resistencias de polarización del puerto B, activo a nivel bajo
- ✓ 1 = Resistencias deshabilitadas.
- ✓ 0 = Resistencias habilitadas.

★ INTEDG: Interrupción por flanco.

- ✓ 1 = Flanco de subida.
- ✓ 0 = Flanco de bajada.

★ T0CS: Selección de modo

- ✓ 1 = Contador
- ✓ 0 = Temporizador

★ T0SE: Selección del flanco

- ✓ 1 = Flanco ascendente
- ✓ 0 = Flanco descendente

★ PSA: Asignación del preescaler

- ✓ 1 = Preescalado al WDT
- ✓ 0 = Preescalado al timer

★ PS2-PS0: Son los bits que indican el valor de preescalado, su valor es distinto si el prescalado es asignado al temporizador o al watchdog, como se puede observar en la Tabla 14:

Tabla 14. *Preescalado.*

PS2,PS1,PS0	Timer	WDT
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

Cuando el bit PSA se encuentra a 0, el prescalado se le asigna al TMR0, y su funcionamiento se rige por la Figura 154:

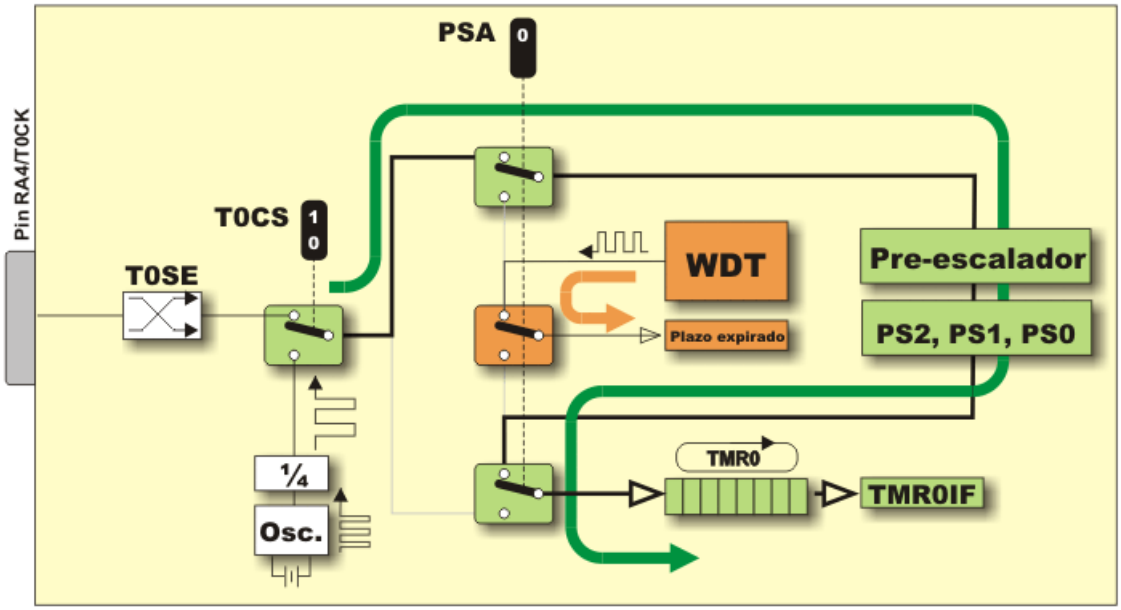


Figura 154. *Preescalado al temporizador.* [6]

Por el contrario si se encuentra a uno, se asigna el preescalado al WDT, (Figura 155):

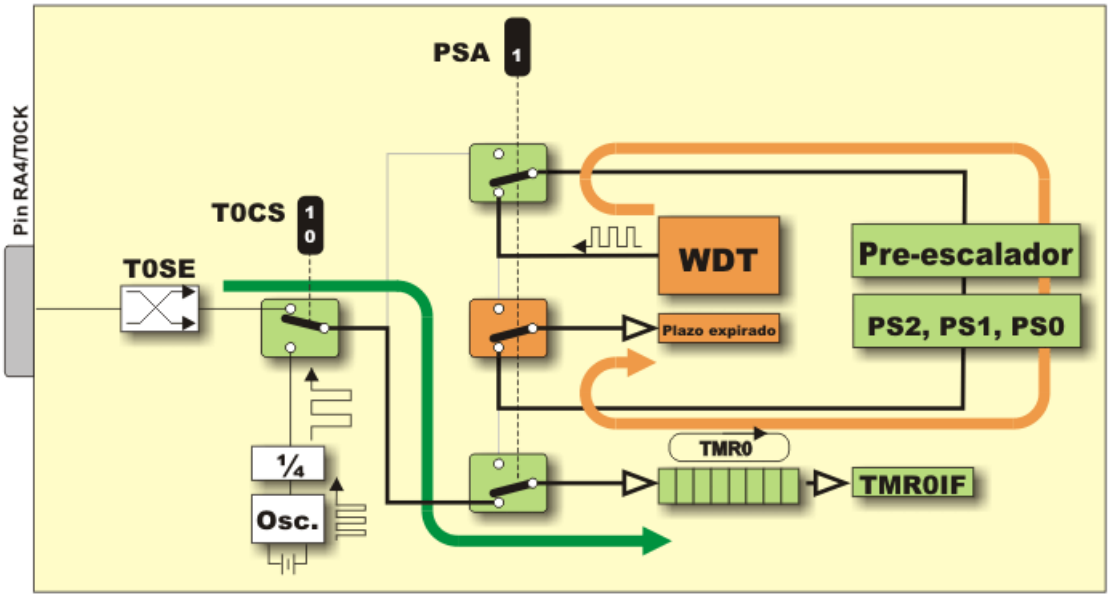


Figura 155. *Preescalado al Watchdog.* [6]

Si el TOCS se encuentra a cero, utiliza la entrada RA4/T0CK como fuente de pulsos funcionando como contador, si por el contrario se encuentra a uno utiliza el reloj interno. Este reloj interno u oscilador, se puede programar a distintas frecuencias como indica la figura 4:

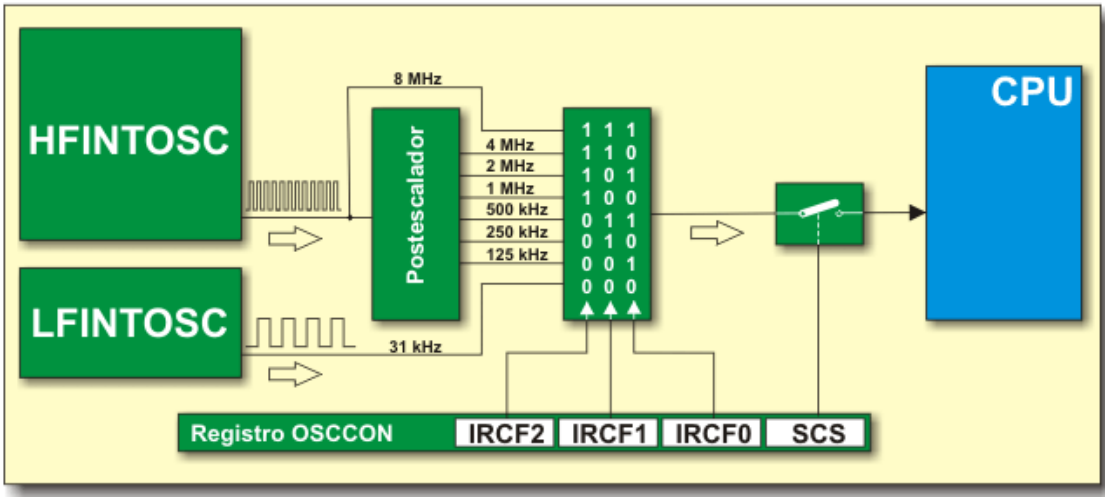


Figura 156. *Frecuencias del reloj interno.* [6]

El registro para el gobierno del oscilador es el OSCCON y se puede observar en la Tabla 15 :

Tabla 15. *Registro OSCCON.* [7]

REGISTRO OSCCON							
BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
--	IRCF2	IRCF1	IRCF0	OSTS	HTS	LTS	SCS

Los bits más interesantes de este registro para la práctica son el IRCF2, IRCF1, IRCF0 que son los encargados de seleccionar el escalado de la frecuencia interna, (figura 156)

- *Tmr1*

El temporizador Timer1 tiene las siguientes características:

- ✓ Temporizador/contador de 16 bits compuesto por un par de registros.
- ✓ Fuente de reloj interna o externa programable.
- ✓ Pre-escalador de 3 bits.
- ✓ Oscilador LP opcional.
- ✓ Funcionamiento síncrono o asíncrono.
- ✓ Compuerta para controlar el temporizador Timer1 (conteo habilitado) por medio del comparador o por el pin T1G.
- ✓ Interrupción por desbordamiento.
- ✓ "Despierta" al microcontrolador (salida del modo de reposo) por desbordamiento (reloj externo).
- ✓ Fuente de reloj para la función de Captura/Comparación.

El funcionamiento de este temporizador se puede apreciar en la figura 157.

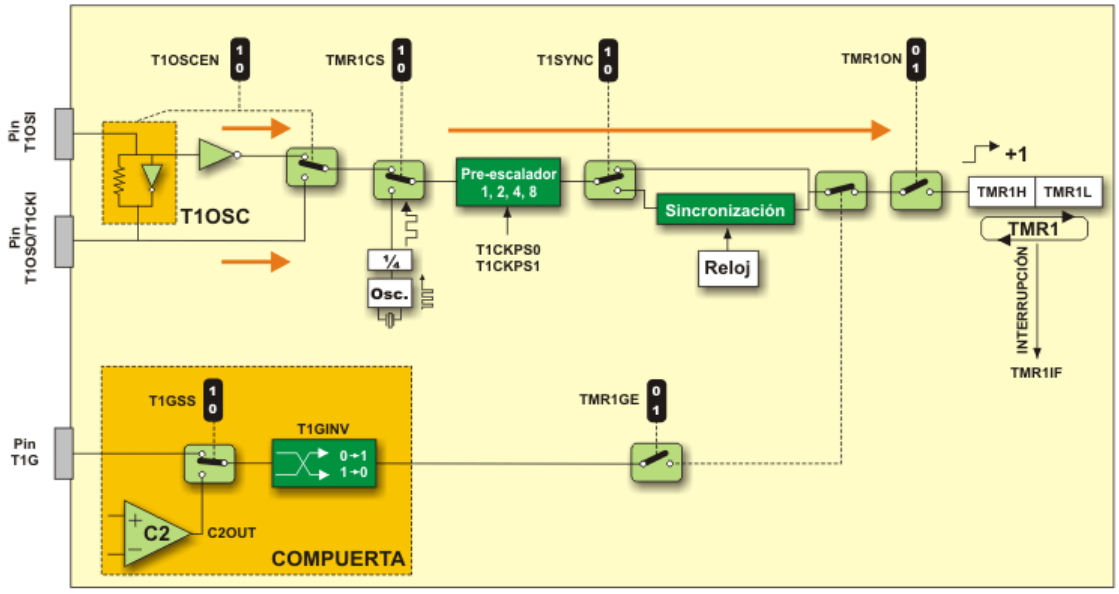


Figura 157. *TMR1.* [6]

Este temporizador es gobernado por el registro T1CON, la composición del registro se observa en la Tabla 16:

Tabla 16. *Registro T1CON.* [7]

REGISTRO T1CON							
BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
T1GINV	TMR1GE	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

Los bits más importantes son los que activan el tmr1 como temporizador o como contador que veremos a continuación.

Para seleccionar el modo temporizador es necesario poner a cero el bit TMR1CS, y el funcionamiento sería, (Figura 158):

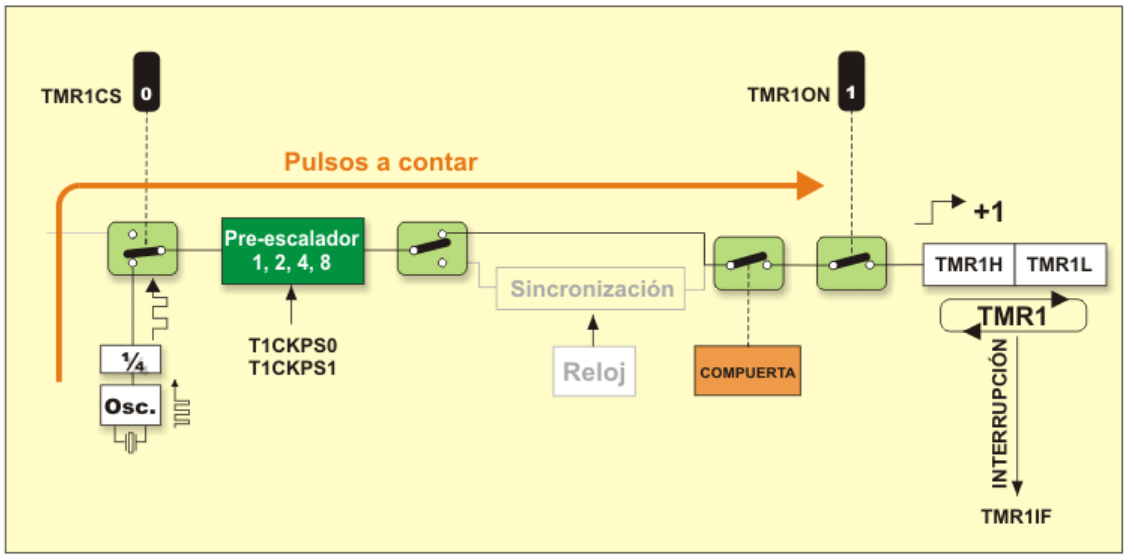


Figura 158. *TMR1 modo temporizador.* [6]

En modo contador sería necesario poner a uno el bit TMR1CS, (Figura 159):

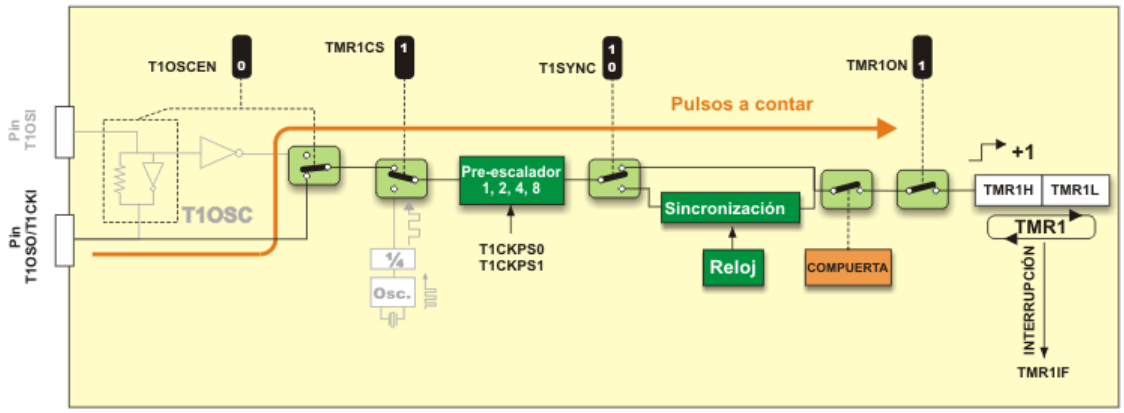


Figura 159. *TMR1 modo contador.* [6]

↳ *Tmr2*

El TMR2 se caracteriza por ser un temporizador de 8 bits, en la figura 160 se explica el funcionamiento:

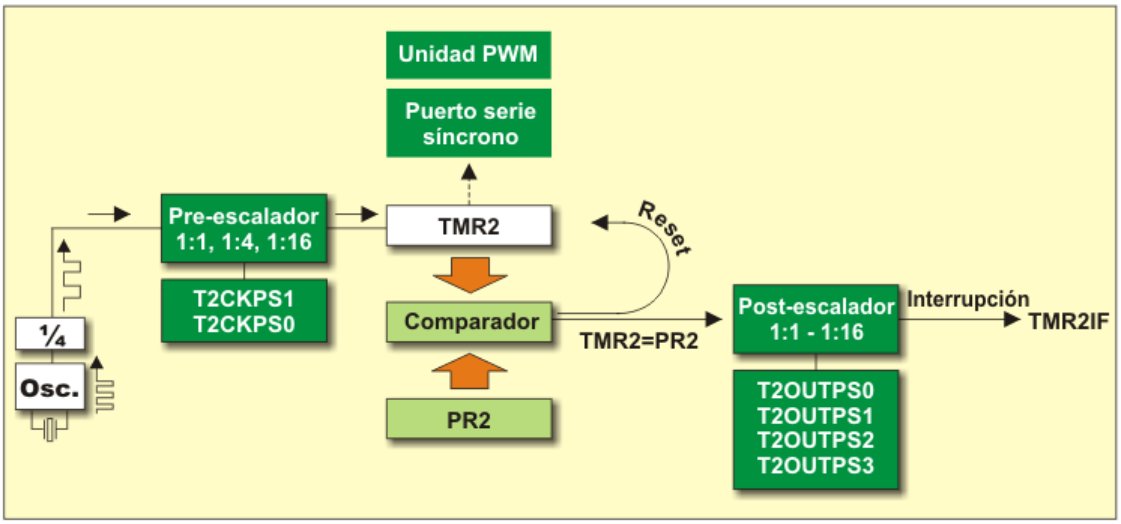


Figura 160. *TMR2.* [6]

El registro que lo gobierna es el T2CON, y se muestra en la Tabla 17:

Tabla 17. *Registro T2CON.* [7]

REGISTRO T2CON							
BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0

- bit 6-3: Selección del post escalado:
  - 0000 = 1:1
  - 0001 = 1:2
  - 0010 = 1:3
  - 0011 = 1:4
  - 0100 = 1:5
  - 0101 = 1:6
  - 0110 = 1:7
  - 0111 = 1:8
  - 1000 = 1:9
  - 1001 = 1:10
  - 1010 = 1:11
  - 1011 = 1:12
  - 1100 = 1:13
  - 1101 = 1:14
  - 1110 = 1:15
  - 1111 = 1:16
- TMR2ON: Activa el temporizador.
- bit 1-0: Selección del preescaler:
  - 00: Preescalado 1
  - 01: Preescalado 4
  - 1x : Preescalado 16

- *Librerías*

Para programar el temporizador se puede ir ajustando mediante el preescalado la frecuencia deseada y así obtener el tiempo deseado. O bien se puede incorporar la macro Delay de temporización contenida en el archivo "MSE_delay.inc" **creada por Microsystems Engineer de Mikel Etxebarria [5]**. Si se usa la macro es necesario incorporar en el programa las siguientes directivas:

- ★ *#define Fosc 4000000*: Para definir la frecuencia de trabajo
- ★ *MSE_Delay_V equ 0x73* : Variables temporales de uso de la MACRO
- ★ *include "MSE_Delay.inc*": Localiza la macro para el uso del programa.
- ★ *Delay XXXX Milis*: El nombre de la macro que permite temporizar XXXX milisegundos.

Cuando se usan entradas digitales, se puede producir el llamado **efecto rebote**, este efecto consiste en que al pulsar una entrada se produzcan varios rebotes produciéndose varias pulsaciones en una, por ello es conveniente esperar un tiempo suficiente para que se estabilice la señal, generalmente se realiza una espera de 25 ms programados con la macro delay [5].

1.1.5.4.5.4.- Ejercicio

1.1.5.4.5.4.1.- Planteamiento

Se trata de realizar una secuencia de luces, inicialmente lucirá el led conectado al pin RB0 y se realizará un desplazamiento a derechas o a izquierdas en función de la entrada RA0:

- ★ RA0 = 0 desplazamiento a derechas.

- ★ RA1 = 1 desplazamiento a izquierdas.

Para la temporización de un segundo se empleará la macro Delay comentada en el apartado **contenido teórico**.

1.1.5.4.5.4.2.- Diagrama de flujo

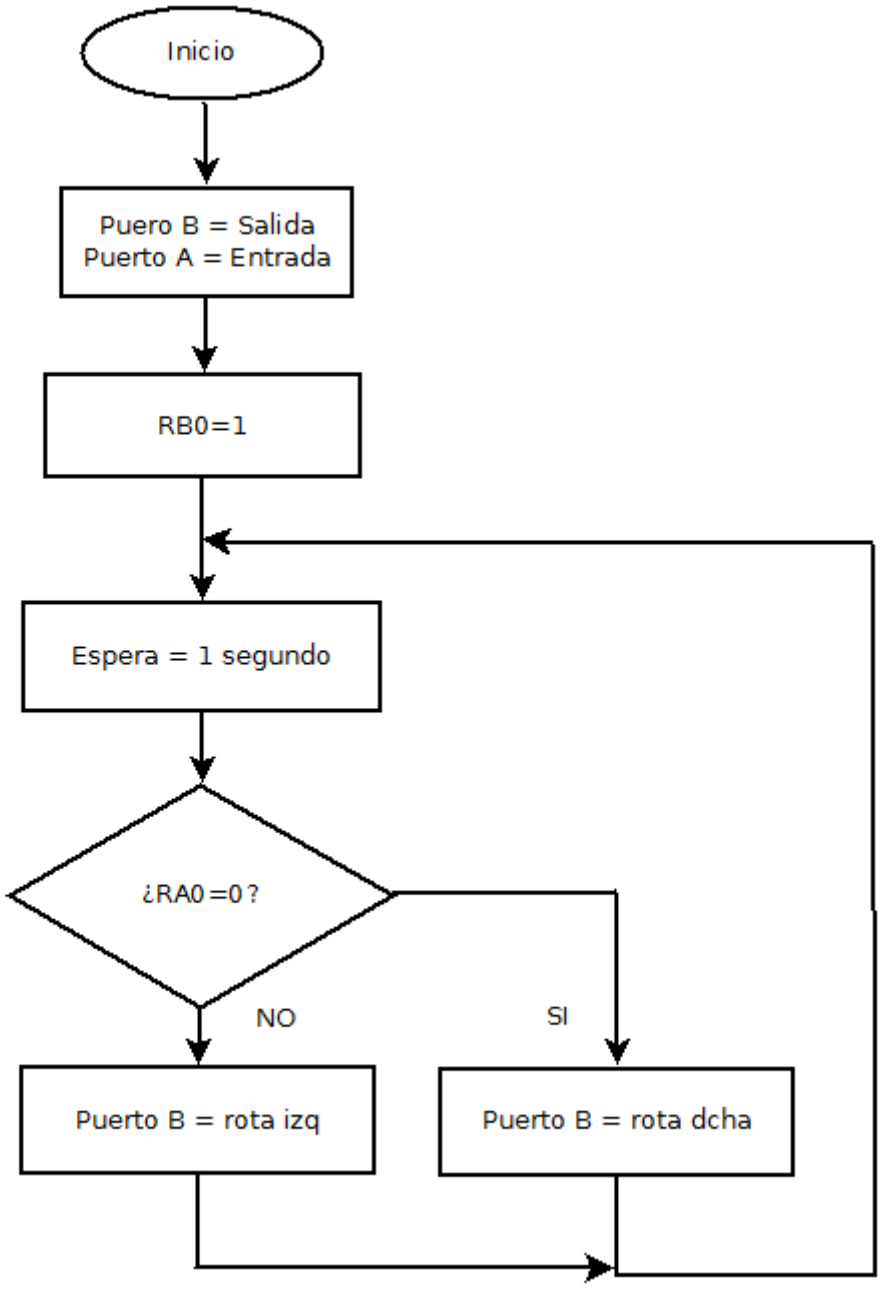


Figura 161. *Diagrama de flujo*

Se configura el puerto B como salida y el puerto A como entrada e inicialmente se activa el LED conectado al pin RB0, se entra en un bucle infinito y comienza esperando 1 segundo y , después chequea el estado del puerto a, si está RA0 a '1' rotara a izquierdas , si está a cero rotará a derechas. (Figura 161).

1.1.5.4.5.4.3.-Programación y simulación

Para la programación del dispositivo, no ha sido necesario el uso de ninguna variable salvo las necesarias en el temporizador, comentadas en el contenido teórico, por lo que la única variable que se declara es, (Figura 162):

```
-----------------------------------------------------
-    DECLARACIÓN DE CTES. Y VARIABLE                -
-----------------------------------------------------

 MSE_Delay_V     equ     0x73    ;Variables (3) empleadas por las macros
```

Figura 162. *Registros y variables.*

↳ *Código*

```
Inicio                  ; Etiqueta de comienzo de programa

;--- Configuración de los puertos ---
    clrw
    clrf    PORTA       ; Se borran los latch de salida del puerto
    clrf    PORTB
    bsf STATUS,RP0
    bsf STATUS,RP1      ; Banco 3
    clrf    ANSEL       ; Entradas digitales
    clrf    ANSELH
    bcf STATUS,RP1      ; Banco 1
    movlw   0xFF        ;
    movwf   PORTA       ; Puerto A (In)
    clrf    PORTB       ; Puerto B (Out)
    bcf STATUS,RP0      ; Banco 0

;--- Programa ---
    movlw   0x01        ; Posición inicial de las luces
    movwf   PORTB

Luces   Delay 1000 Milis    ; Tiempo de espera hasta rotación
        btfss   PORTA,0     ; RA0=1?
        goto    rota_dcha   ; RA0=0 --> rota a dererechas
        rlf PORTB           ; RA0=1 --> rota a izquierdas
        goto Luces
rota_dcha   rrf PORTB
        goto Luces
```

Figura 163. *Código del programa.*

El programa comienza declarando el puerto A como entrada digital y el puerto B como salida digital (**ver práctica 4**).

Se comienza inicializando el puerto B a b'00000001' y se pasa al bucle, mediante el comando **Delay** llamamos a la macro y le indicamos el tiempo que tiene que estar en espera

Una vez pasado el segundo, testea RA0 si se encuentra a uno realiza un desplazamiento a izquierdas y si se encuentra a cero el desplazamiento será a derechas, (Figura 163).

↳ Simulación

Mediante el logic analycer (**MPLAB**) podemos ir observando los diferentes estados de los pin, (Figura 164):

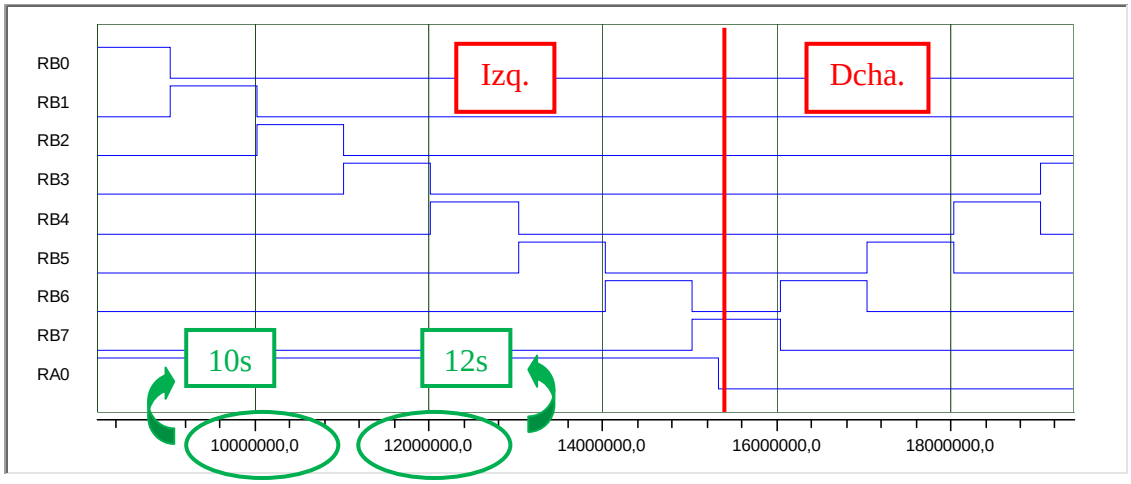


Figura 164. *Simulación del programa.*

Mientras la entrada RA0 se encuentre a nivel alto, se observa cómo se produce el *desplazamiento hacia la izquierda*, cuando la entrada se pone a cero una vez terminado el tiempo de espera de un segundo se produce la *rotación a derechas*.

1.1.5.4.5.4.4.- Montaje

Conectar el interruptor E0 con la entrada RA0 y los leds S0-S7 a las líneas de salida B0-B7 como se indica en la siguiente figura:

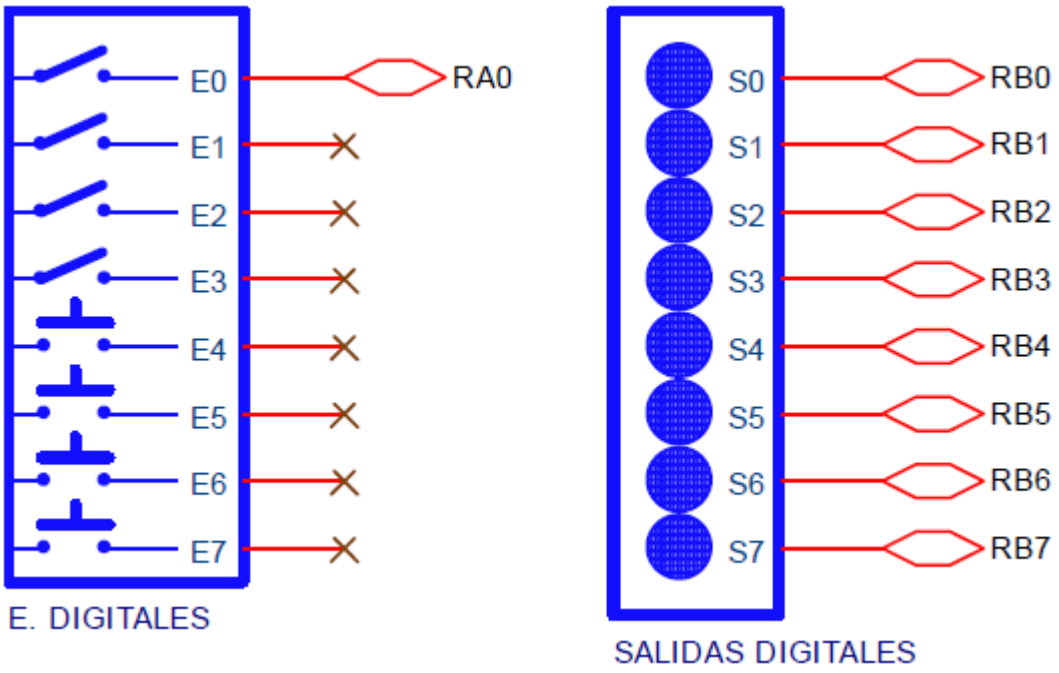


Figura 165. *Esquema de montaje.* [6]

1.1.5.4.5.5.- Programas propuestos

Realizar un programa que controle 3 leds simulando un semáforo. El semáforo en verde estará 5 segundos, en ámbar 1 segundo y en rojo 3 segundos. Además existirá un semáforo para peatones verde y rojo.

Realizar un programa que cuente las monedas que se introducen en una máquina, para ello simplificar el programa de tal forma que sólo se introduzca un tipo de moneda, usar el tmr0 en modo contador. Crear un registro que vaya contando los desbordamientos del tmr0 y cuando llegue a 1000 monedas activar un led indicador de lleno, y que funcione hasta que no se vacíe.

1.1.5.4.6.- Practica 6: Interrupciones.

1.1.5.4.6.1.- Objetivo

- ✓ Comprender el funcionamiento de las diferentes interrupciones provocadas en el PIC16F886.
- ✓ Entender los registros relacionados con las interrupciones.
- ✓ Ser capaz de programar las interrupciones del PIC.

1.1.5.4.6.2.- Enunciado

Programar un led que parpadee durante 0,25 segundos usando la interrupción por temporización.

1.1.5.4.6.3.- Contenido teórico

Una **interrupción es una subrutina que no es llamada de forma convencional**, para poder llamarla es necesaria que se den ciertas condiciones.

Dependiendo de dichas condiciones se obtienen los tres tipos generales de interrupciones:

- *Por temporizador*: ocurre si se desborda el TMR0.
- *Externa*: Si se activa RB0, debe ser configurada como entrada.
- *Puerto B*: Si hay un cambio de estado en el puerto B.

Independientemente del tipo de interrupción. su funcionamiento se observa en la siguiente figura, (Figura 166):

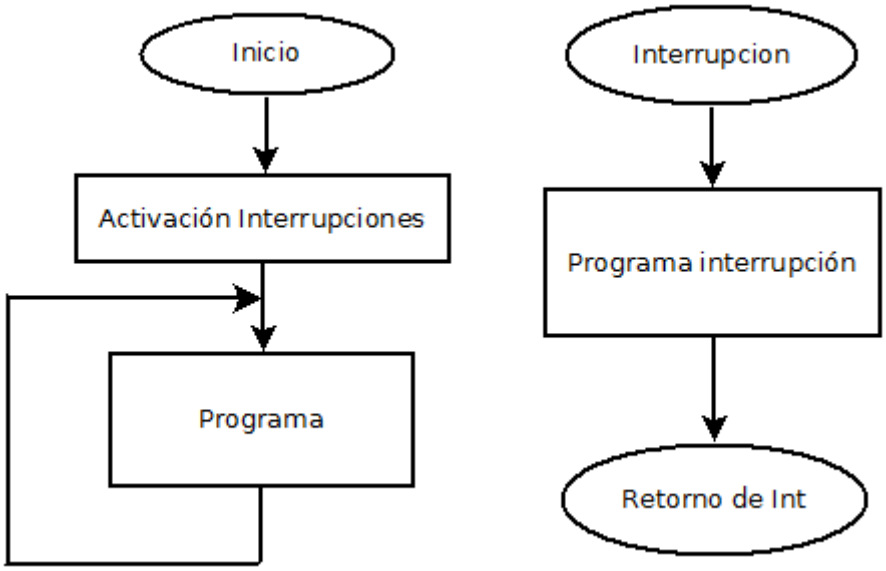


Figura 166. *Interrupciones.*

Para que se produzca una interrupción es necesario que previamente se hallan habilitado, después el programa continua cíclicamente hasta que se produce alguna de las condiciones de interrupción, cuando eso ocurre el programa salta automáticamente a la subrutina de interrupción y cuando se dan las condiciones necesarias para terminar la subrutina vuelve al programa principal.

El registro que gobierna las interrupciones es el registro INTCON, y se muestra en la Tabla 18:

Tabla 18. *Registro INTCON***.** [7]

REGISTRO INTCON							
BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

Todos los bits indican activación a nivel alto y en el caso de los flag se ponen a uno cuando ocurre la interrupción:

★ GIE: habilitación de interrupciones.

- ★ PEIE: habilitación de interrupciones periféricas.
- ★ T0IE: habilitación de interrupción por temporizador
- ★ INTE: habilitación de interrupción externa.
- ★ RBIE: habilitación de interrupción del puerto b.
- ★ T0IF: Flag de interrupción por temporizador.
- ★ INTF: Flag de interrupción externa.
- ★ RBIF: Flag de interrupción del puerto b.

Cuando se produce una interrupción el programa salta a la posición 0x04h, en esa posición es necesario programar el salto a la interrupción, (Figura 167):

```
org 0x00    ; Posición de memoria de prog. que comienza
goto Inicio ; Salto a la etiqueta Inicio

org 0x04    ; Posición del vector Interrupción
goto Interrupcion
org 0x05    ; Posición de memoria de prog. de la tabla
```

Figura 167. *Vector de interrupciones.*

Cuando se usan a la vez más de una interrupción es necesario realizar una programación para identificar el tipo de interrupción y una vez identificada borrar el flag de interrupción correspondiente como se indica en la figura 168:

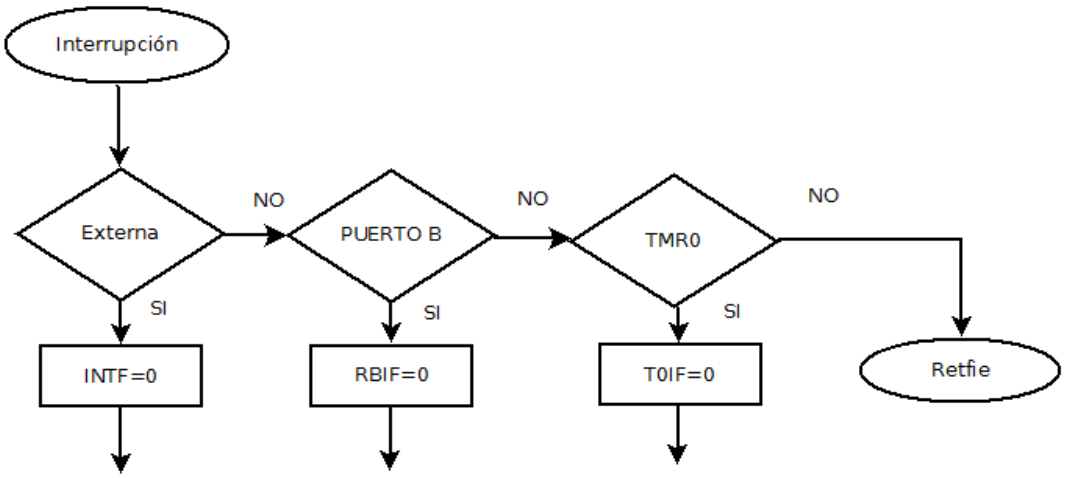


Figura 168. *Comprobación del tipo de interrupción.*

Según el programa se puede cambiar la prioridad si saltaran dos interrupciones al mismo tiempo, en este caso la prioridad principal la tiene la interrupción externa, en otros programas puede ser distinta esta configuración.

Nada más saltar la interrupción, se habilita el flag indicando que interrupción fue, y solamente se puede borrar por medio de software.

Además al saltar a la interrupción se deshabilitan las interrupciones (GIE =0), por lo que dentro de la interrupción sino se programa **no pueden saltar nuevas interrupciones** y con la instrucción *retfie* se retorna igual que un return , sin embargo, activa de nuevo las interrupciones (GIE = 1).

Las interrupciones pueden ocurrir en cualquier momento del programa, por esta razón en algunos programas lo primero que se debe hacer es **almacenar el contenido de los registros importantes** (p.e. Status o W) nada más empezar la interrupción en unos registros de uso temporal.

Antes de que vaya a finalizar la interrupción, se cargará la información de dichos registros importantes, almacenada en los registros temporales creados previamente.(Figura 169)

```
Interrupcion
    movwf   TEMP_W      ; guardo W en un registro temporal
    swapf   STATUS,W    ; almaceno STATUS en W sin afectar a Z
    movwf   TEMP_STATUS ; guardo STATUS en un registro temporal
;---
;... Código de la interrupción
;---
    swapf   TEMP_STATUS,W   ;
    movwf   STATUS          ; Cargo en status el valor anterior al salto
    swapf   TEMP_W,TEMP_W   ;
    swapf   TEMP_W,W        ; Cargo en W el valor anterior
```

Figura 169. *Copia de los registros importantes*

En W, es necesario realizar un swap sobre sí mismo, ya que el swap cambia el orden de los bits, así con dos swap se obtiene el registro original.

1.1.5.4.6.4.- Ejercicio

1.1.5.4.6.4.1 - Planteamiento

Se desea hacer a un led parpadear cada 250ms mediante la interrupción por temporizador, para ello se habilitará exclusivamente dicha interrupción y cada vez que el TMR0 pase de 255 a 0 saltará la interrupción.

Para calcular los 250 ms es necesario tener en cuenta la frecuencia del temporizador (ver **práctica 5**) La frecuencia del temporizador (8) y el tiempo de cada ciclo (9)

$$F_{TMR0} = \frac{F_{INT}}{4} \quad (7)$$

$$F_{TEMP} = \frac{F_{TMR0}}{escalado} \quad (8)$$

$$T_{TEMP} = \frac{escalado}{F_{TMR0}} \quad (9)$$

Siendo:

- ✓ F_{INT}: frecuencia interna
- ✓ F_{TMR0}: frecuencia del tmr0 sin escalar
- ✓ F_{TEMP}: frecuencia del tmr0 escalada
- ✓ escalado: factor escala seleccionado por el registro OPTION

Cada Ttemp realizará una cuenta, para calcular el tiempo que tarda en desbordarse (10)

$$t_{desb} = (256 - VI) \cdot T_{TEMP} \quad (10)$$

Con:

- ✓ t_{desb}: tiempo que tarda en desbordarse el temporizador
- ✓ VI: Valor inicial del temporizador

Despejando la valor inicial obtenemos (11):

$$VI = 256 - \frac{t_{desb}}{T_{TEMP}} = 256 - t_{desb} \cdot F_{TEMP} \tag{11}$$

$$VI = 256 - t_{desb} \cdot \frac{F_{INT}}{4 \cdot escalado} \tag{12}$$

En el programa se busca un tiempo de 10ms, lo que implicaría que se desbordara 25 veces. El valor inicial que se debe dar al temporizador será (13)

$$VI = 256 - \frac{0.01s \cdot 4 \cdot 10^6 s^{-1}}{4 \cdot 256} = 216.93 \rightarrow 217 \tag{13}$$

Como se trata de un parpadeo de un led no es necesario ser muy preciso en el cálculo de la cuenta, sino se debería buscar una cuenta exacta ya que el error cometido se calcula de la siguiente forma (15)

$$t_{desb} = (256 - 217) \cdot \frac{256}{10^6} = 9.984 \cdot 10^{-3} \tag{14}$$

$$\varepsilon(\%) = \left(1 - \frac{9.984}{10}\right) \cdot 100 = 0.16\% \tag{15}$$

En este programa no es necesario guardar el registro status ni W al no intervenir en la ejecución.

1.1.5.4.6.4.2. - Diagrama de flujo

Primero se configuran los puertos, el preescaler del temporizador y se habilita la interrupción por TMR0. A continuación el programa entra en un bucle *testeando en cada momento si la cuenta a llegado a 50*, cuando la cuenta llegue a 50, la pone a cero y cambia el estado del led.(Figura 170).

Cuanto el *TMR0 se desborda*, se pasa a la subrutina de interrupción en donde se desactiva el flag, se incrementa en uno la variable cuenta y se actualiza el temporizador.

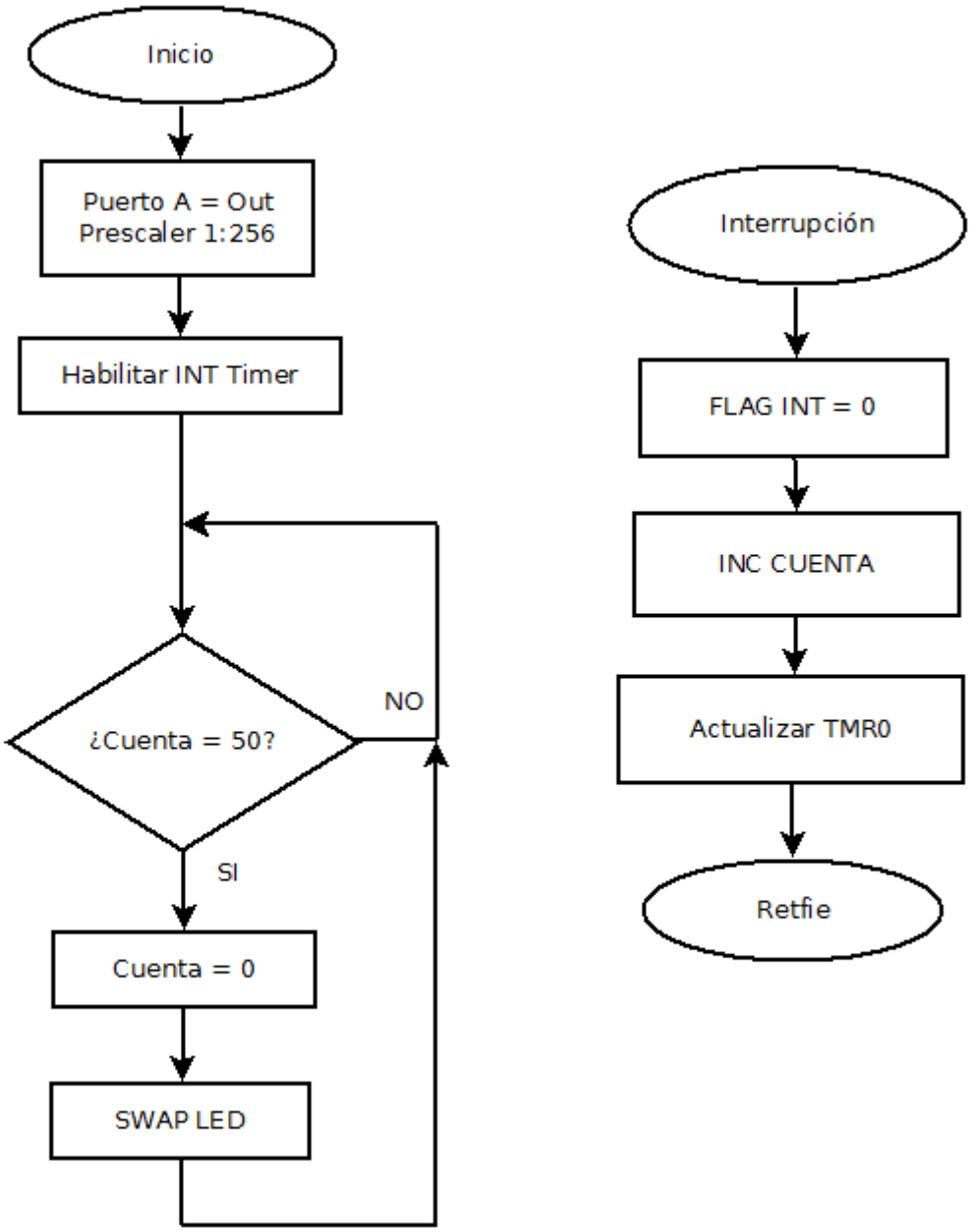


Figura 170. *Diagrama de flujo.*

1.1.5.4.6.4.3. - Programación y simulación

Para la programación del ejercicio se emplearon las siguientes constantes y variables, (Figura 171):

```
;-----------------------------------------------------
;---    DECLARACIÓN DE CTES. Y VARIABLE              -
;-----------------------------------------------------
    #define cuenta1 .217
    #define cuenta2 .25

    CONTADOR equ    0x30
```

Figura 171. *Registros y constantes.*

- ✓ *cuenta1*: El valor con el que tiene que inicial el TMR0
- ✓ *cuenta2*: Indica las veces que se tiene que desbordar el TMR0 para llegar a los 250ms.
- ✓ *CONTADOR*: Variable que va contando el número de veces que se desborda el TMR0.

1.1.5.4.6.4.3.1 - Programa principal

- ↳ *Código*

La configuración del registro **OPTION_REG se tiene que realizar en el banco 1**, los tres últimos bits indican el tipo de preescalado, en nuestro caso 1:256. (Figura 172).

Lo último que se habilita es la interrupción por temporizador, *se suele habilitar al final para que no salte un momentos indeseados.*

En el programa principal simplemente se trata de un bucle que testea mediante una resta la variable contador con cuenta2, si no son iguales el flag de Z no saltará y volverá al inicio de bucle.

Si CONTADOR ha llegado a 50, significa que se ha llegado a los 250ms por lo que se cambia el estado del LED con la función XOR (ver practica 1) y se pone a cero el CONTADOR.

```
Inicio                  ; Etiqueta de comienzo de programa

;--- Configuración de los puertos ---
    clrw
    clrf    PORTA
    bsf STATUS,RP0
    bsf STATUS,RP1      ; Banco 3
    clrf    ANSEL       ; Entradas digitales
    clrf    ANSELH
    bcf STATUS,RP1      ; Banco 1
    clrf    PORTA       ; Puerto A (Out)

;--- Configuración del TMR0
    movlw b'0000111' ; modo timer escala 1:256 flanco ascendente
    movwf OPTION_REG
    bcf STATUS,RP0      ; Banco 0


;--- Programa ---

    movlw   cuenta1     ; carga inicial del TMR0
    movwf   TMR0
    movlw   b'10100000' ; Habilito solo int TMR0
    movwf   INTCON

Led movlw   cuenta2
    subwf   CONTADOR,W
    btfss   STATUS,Z    ; CONTADOR = 50 ?
    goto    Led         ; NO
    movlw   0x01        ; SI
    xorwf   PORTA       ; RA0 cambia de estado
    clrf    CONTADOR    ; Pongo a cero contador
    goto    Led
```

Figura 172. *Código del programa.*

↳ *Simulación*

Para simular este programa es necesario desactivar el WDT desde la pestaña configure (configuration bits) o introduciendo en el código la expresión clrwdt para que no se desborde. Obteniendo así la figura 173:

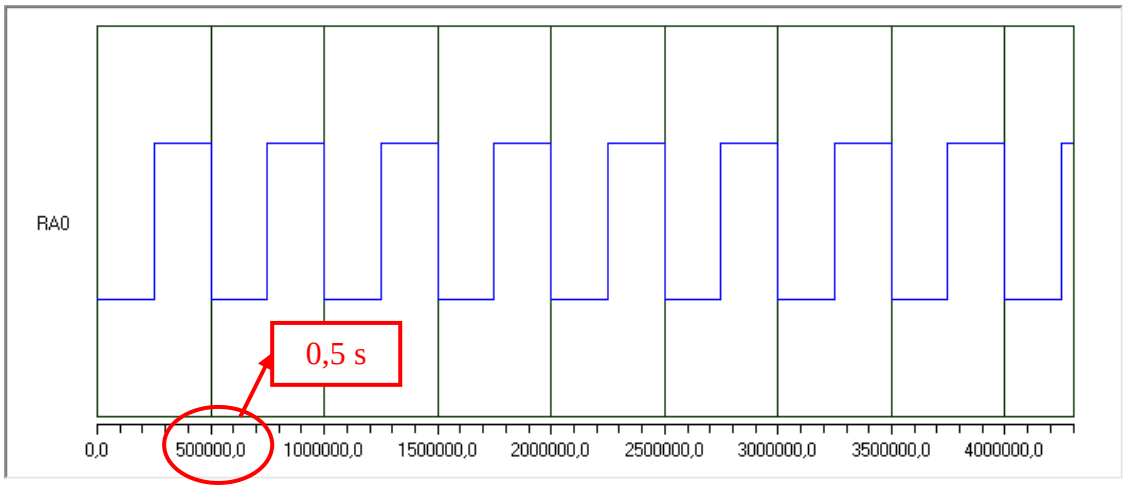


Figura 173. *Parpadeo del led.*

Se puede apreciar como **cada 0.25s se apaga o se enciende el LED** conectado a RA0.

1.1.5.4.6.4.3.2 - Interrupción

↳ *Código*

```
Interrupcion
    bcf INTCON,T0IF     ; desactivo el flag
    incf    CONTADOR    ; Aumento contador
    movlw   cuenta1     ; actualizo el TMR0
    movwf   TMR0        ;
    retfie
```

Figura 174. *Subrutina de interrupción.*

Dentro de la interrupción *se actualiza el TMR0* con el valor calculado en el contenido teórico, definido por la constante cuenta1.(Figura 174).

↳ *Simulación*

Antes de entrar en la interrupción los valores de los registros más importantes son, (Figura 175):

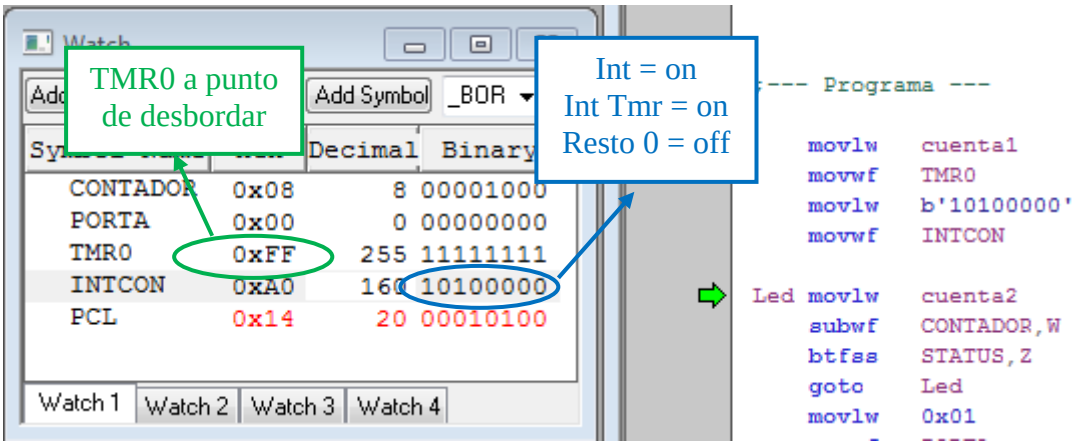


Figura 175. *Valores anteriores al desbordamiento del temporizador.*

Al entrar en la interrupción se observa cómo se desborda el TMR0, se pasa a la posición 0x04, se deshabilita las interrupciones y se activa el flag de interrupción por temporizador, (figura 176):

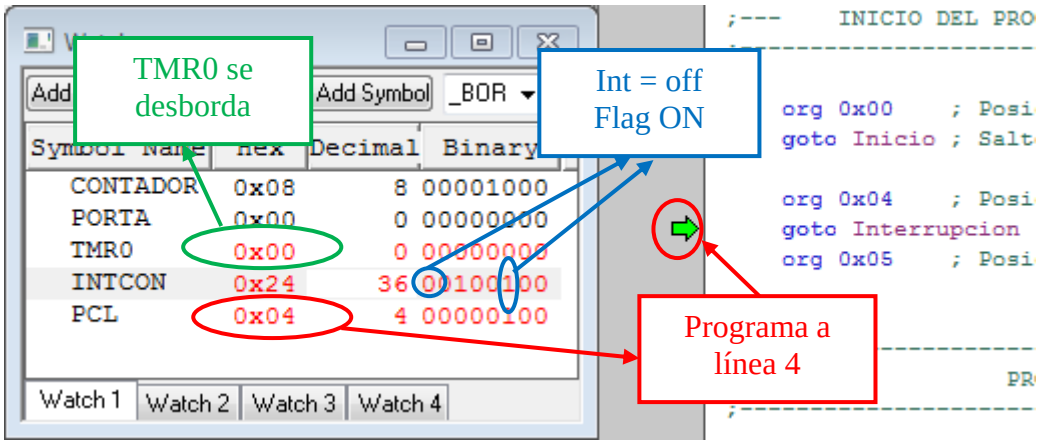


Figura 176. *Activación de la interrupción.*

Dentro de la interrupción primero se borra el flag, para ello es necesario programarlo por software mediante la instrucción *bcf* (figura 177):

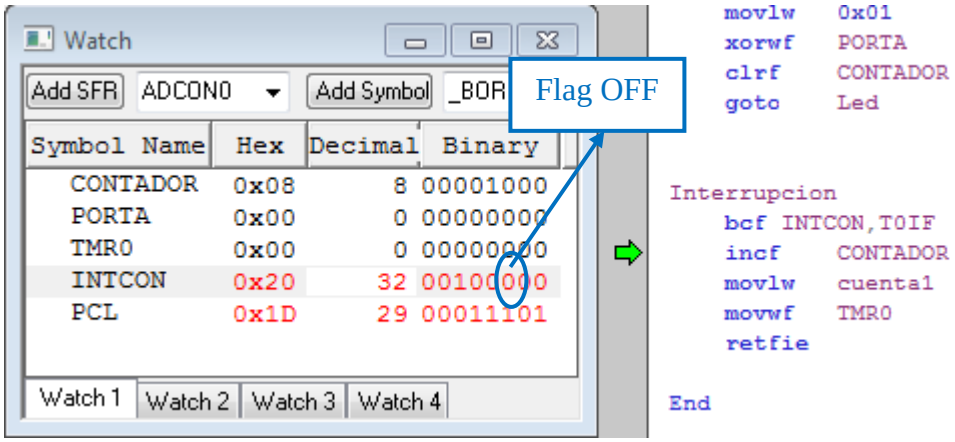


Figura 177. *Borrado del Flag por software.*

Se incrementa el contador, (figura 178)

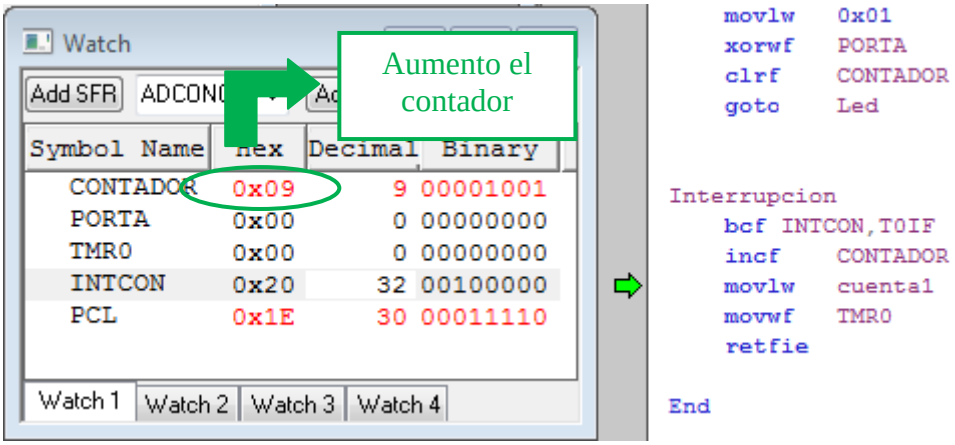


Figura 178. *Incremento del contador*.

Y por último se actualiza el contador del timer, con el valor de 217 para que vuelva a realizar otra cuanta de 10 milisegundos. (Figura 179):

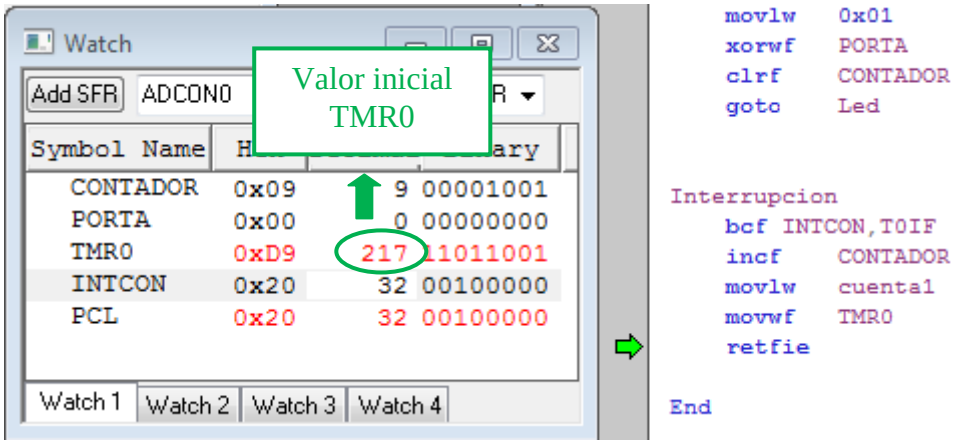


Figura 179. *Carga del valor del TMR0.*

Al retornar se observa como gracias a la instrucción retfie se vuelven a habilitar las interrupciones, el bit que habilita las interrupciones de forma general pasa a nivel alto, INTCON,GIE = '1' (Figura 180):

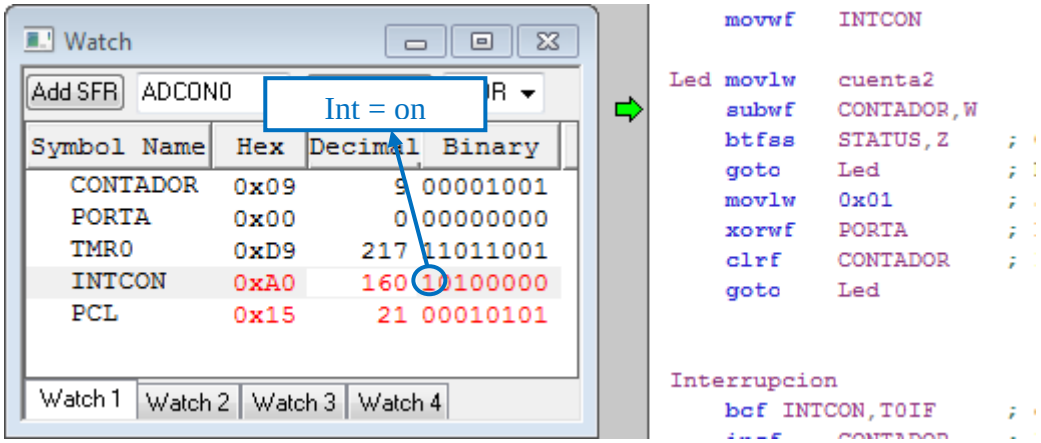


Figura 180. *Retorno de interrupción.*

Y se observa como el contador del programa ejecuta la siguiente instrucción, donde se quedó al ser llamada una interrupción, (Figura 175)

1.1.5.4.6.4.4. - Montaje

En esta práctica se tiene una única salida RA0 y se conectará al led S0, (Figura 181):

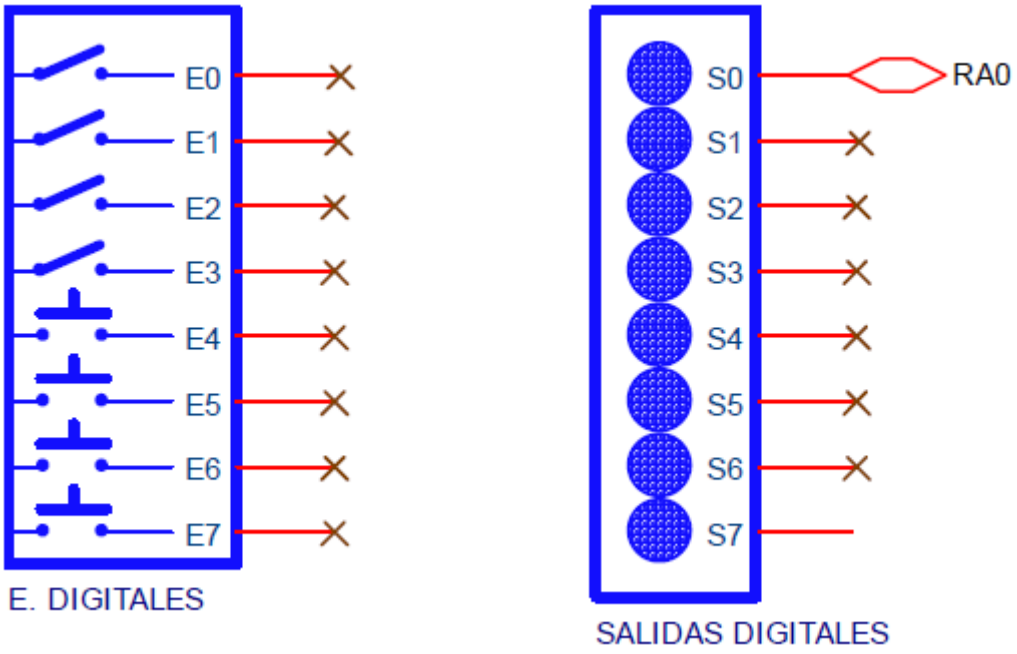


Figura 181. *Esquema de montaje.* [6]

1.1.5.4.6.5.- Programas propuestos

Realizar un programa que contiene un interruptor de activación (RA0) y dos entradas RA2 y RA3 que según su configuración las salidas (RB1 – RB4) son:

RA3	*RA2*	*RB4*	*RB3*	*RB2*	*RB1*
0	*0*	*0*	*1*	*0*	*1*
0	*1*	*0*	*0*	*1*	*1*
1	*0*	*1*	*1*	*0*	*0*
1	*1*	*0*	*0*	*0*	*0*

Hay dos pulsadores un de paro (RB0) programado como interrupción externa, y uno de activación (RA3) de tal forma que si se activa el botón de paro hay que activar el pulsador activación para que continúe.

Realizar un programa, que active una señal de emergencia al activar el pulsador de paro y ponga todas las salidas (puerto B) a cero. Para volver a arrancar el sistema es necesario activar un pulsador, Además de un interruptor general . Mientras el sistema funciona.

1.1.5.4.7.- Practica 7: Memoria EEPROM

1.1.5.4.7.1 - Objetivo

- ✓ Estudiar la programación de la memoria eeprom.
- ✓ Comprender el funcionamiento de lectura de la eeprom
- ✓ Comprender el funcionamiento de escritura de la eeprom.
- ✓ Analizar los registros que intervienen en la programación de la ROM.

1.1.5.4.7.2 - Enunciado

Realizar contador de 0 a 9 del turno de una tienda de supermercado, se dispondrá de un pulsador en RA0 que irá actualizando el turno actual, cada turno será grabado en la memoria eeprom por si se apagará la corriente y así recordar en que turno se encuentra.

.

1.1.5.4.7.3 - Contenido teórico

La memoria EEPROM del PIC 16F886 empieza en la dirección 0x00h y termina en la 0xFFh, es decir, almacena 256 datos de 8bits por lo que tiene una capacidad de 256 bytes.

Los registros que intervienen en la programación de la EEPROM son en el banco 2 EEADR, EEDATA y en el banco 3 EECON1, EECON2.

- *EEADR:* Es el puntero de la EEPROM, contiene la dirección de memoria cuyo dato va a ser grabado o leído. Al contener 256 bytes sólo los primeros 6 bits sirven para direccionar toda la memoria ROM

- *EEDATA*: Guarda el contenido de la posición de EEADR antes de su escritura o después de su lectora.

- *EECON1:* Este registro controla la escritura y lectura de la EEPROM.(Tabla 19)

Tabla 19. *Registro EECON1*

REGISTRO EECON1							
BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
EEPGD	-	-	-	WRERR	WREN	WR	RD

★ EEPGD: Selecciona la memoria
'1' = Memoria de programa.
'0' = Memoria de datos.

★ WREER: Flag de error

★ WREN: Si esta activo permite la escritura.
'1' = Permite escritura en la ROM.
'0' = No permite escribir en la ROM.

★ WR: ciclo de escritura.
'1' = Comienzo de escritura.
'0' = Escritura off.

★ RD: el ciclo de lectura.
'1'= Comienzo de lectura
'0' = Lectura off.

↳ *EECON2*: Este registro no está implementado físicamente, solo sirve para la protección de escritura, para que no se pueda modificar la eeprom solo trabajando con EECON1.

La memoria ROM es una memoria de acceso lento, requiere aproximadamente 10ms grabar o un dato en ella.

Inicialmente toda la memoria se encuentra a uno, cada dirección de memoria contiene 0xFFh.

El proceso para copiar un dato en la eeprom sigue el siguiente esquema, (Figura 182):

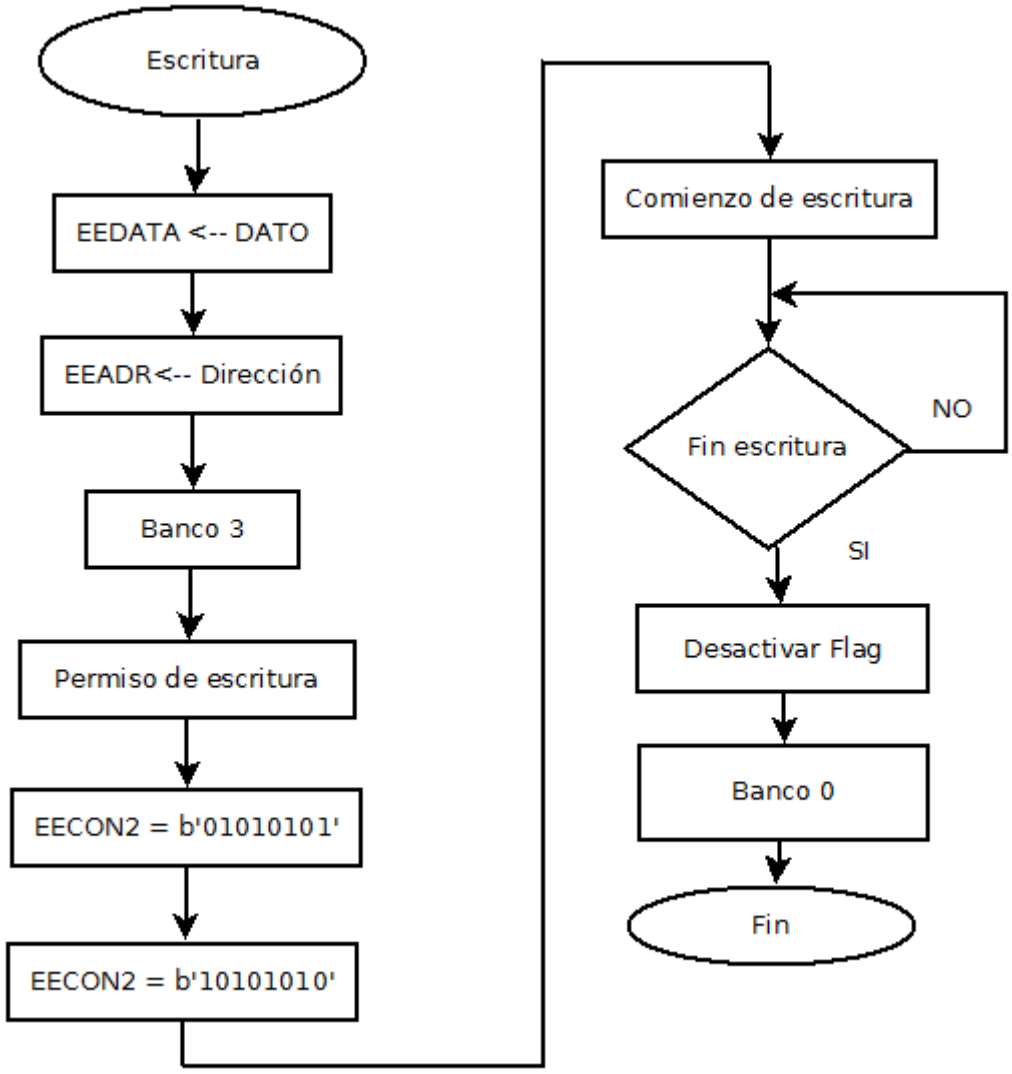


Figura 182. *Escritura en la EEPROM.*

Se copia el dato a almacenar en el registro EEDATA y la dirección de la eeprom en el registro EEADR. Los registros que manejan el funcionamiento se encuentran en el banco 3, una vez cambiado de banco se ordena el permiso de escritura, se introducen los valores de b'01010101' y b'10101010' al registro EECON2 y ya se puede dar la orden de comienzo de escritura. Cuando se active el flag de fin de escritura el proceso habrá terminado por lo que desactivamos el flag y volvemos al banco 0.

Para leer un dato de la EEPROM el proceso es el siguiente, (Figura 183):

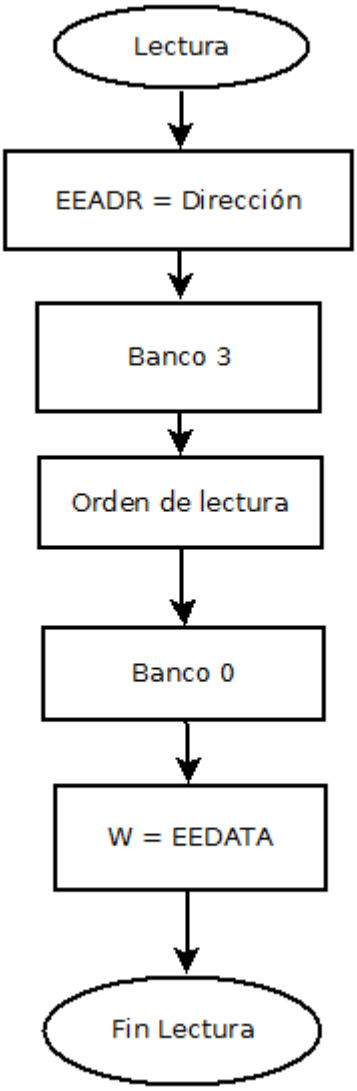


Figura 183. *Lectura de la EEPROM.*

Para el proceso de lectura, se indica la dirección del dato a leer en el registro EEADR, se pasa al banco 3 dando la orden de lectura, el dato se copiara en el registro EEDATA, por lo que se vuelve al banco cero y el dato se pasa por ejemplo al registro de trabajo.

1.1.5.4.7.4 - Ejercicio

1.1.5.4.7.4.1 - Planteamiento

La única entrada es un pulsador que conectado a RA0. Las salidas serán el puerto B que será conectado al display. Se utilizará una variable contador para ir contando el número de clientes, esa variable será codificada para un display de siete segmentos (**práctica 3**) y su valor máximo será 9. Cada vez que se accione el pulsador sumará uno la variable contador, se visualizará en el display y se almacenará en la eeprom.

1.1.5.4.7.4.2 - Diagrama de flujo

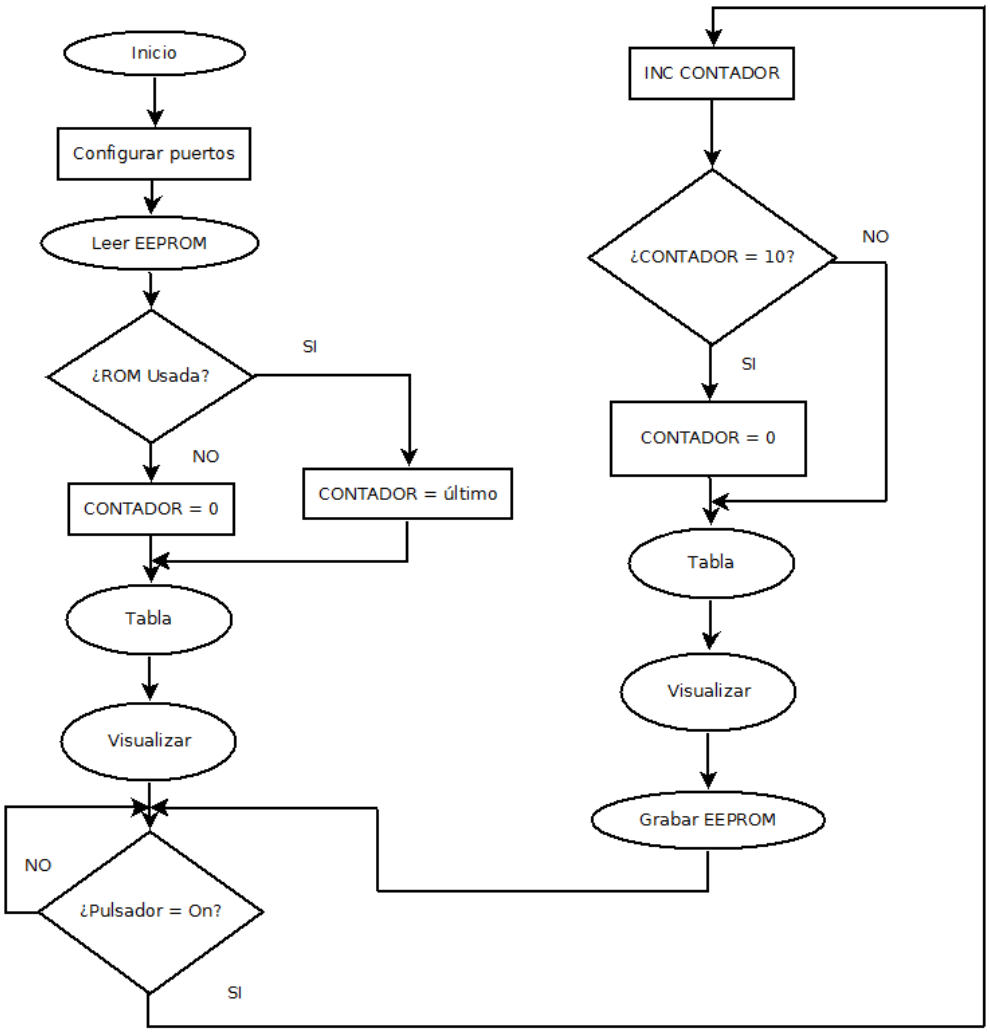


Figura 184. *Diagrama de flujo.*

El programa comienza configurado los puertos , a continuación lee el dato de la memoria ROM, si ha sido grabado algún dato con anterioridad, carga ese dato, y si la memoria ROM está limpia pone el contador a cero.(Figura 184)

Mediante las funciones **Tabla y visualiza**r, transforma el contador en binario a una codificación de siete segmentos para mostrar en el display. Una vez visualizado el primer número entra en un bucle testeando si se ha pulsado el botón del turno, en cuanto es pulsado sale del bucle y aumenta el contador. Comprueba si el contador ha llegado a diez y si se da el caso le pone a cero, después visualiza en el display el valor del contador y guarda el número en la EEPROM, comenzando de nuevo el bucle.

1.1.5.4.7.4.3 - Programación y simulación

En esta práctica se vuelve a incluir la librería MSE_Delay.INC usada para la generación de retrasos. Las constantes y variables empleadas en el programa son, (Figura 185):

```
;-----------------------------------------------------
;---    DECLARACIÓN DE CTES. Y VARIABLE               -
;-----------------------------------------------------
    MSE_Delay_V equ 0x73    ;Variables (3) de temporización
    CUENTA      equ 0x30    ;Variable para contar el turno

    #define maximo  .9      ; Máximo de número a representar
;--- Display   ---

    #define cero    b'00111111'
    #define uno     b'00000110'
    #define dos     b'01011011'
    #define tres    b'01001111'
    #define cuatro  b'01100110'
    #define cinco   b'01101101'
    #define seis    b'01111101'
    #define siete   b'00000111'
    #define ocho    b'01111111'
    #define nueve   b'01101111'
    #define fallo   b'01111001'
```

Figura 185. *Registros y constantes.*

- *MSE_Delay_V:* Corresponden a tres registros reservados para la librería MSE_Delay

- *CUENTA:* Registro que almacena el turno en binario.
- *máximo:* Índica el máximo número que puede alcanzar el registro CUENTA
- *Display:* Todas las codificaciones de los dígitos convertidos a 7 segmentos.

1.1.5.4.7.4.3.1 - Programa principal

- *Código*

El programa principal se desglosa en tres grandes partes, la configuración de puertos, la carga inicial de los registros y el bucle principal.

- *Configuración de puertos*

De forma análoga a otras prácticas **(**ver practicas 4,5 y 6**)** se configura el puerto B como salida y el puerto A como entrada, ambos digitales, (Figura 186)

```
Inicio                  ; Etiqueta de comienzo de programa

;--- Configuración de los puertos ---
    clrw
    clrf    PORTA
    clrf    PORTB       ; Borra los latch de salida
    bsf STATUS,RP0
    bsf STATUS,RP1      ; Banco 3
    clrf    ANSEL       ; Entradas digitales
    clrf    ANSELH
    bcf STATUS,RP1      ; Banco 1
    clrf    PORTB       ; Puerto B (Out)
    movlw   0xFF        ; Puerto A (In)
    movwf   PORTA
    bcf STATUS,RP0      ; Banco 0
```

Figura 186. *Configuración de puertos.*

- *Carga inicial*

```
;--- Programa carga inicial ---

    bsf     STATUS,RP1  ; Banco 2
    clrf    EEADR       ; Posición de ROM donde se guarda el dato
    call    Lee_eeprom  ; Lee el dato de EEADR
    bsf     STATUS,RP1  ; Banco 2
    movlw   0xFF
    subwf   EEDATA,W
    bcf     STATUS,RP1  ; banco 0
    btfss   STATUS,Z    ; ¿ MEMORIA SIN USAR ?
    goto    Dato_inicial
    clrf    CUENTA      ; SI --> CONTADOR = 0
```

Figura 187. *Lectura de la memoria.*

Una vez configuraos los puertos , se indica la dirección de la EEPROM que tiene que ser cargada, en nuestro caso 0x00h, en el registro EEADR (Banco 2), a continuación se llama a la subrutina encargada de leer el dato, y en el retorno se tiene el dato en el registro EEDATA también del banco 2,

Se le compara con el valor 0xFFh (valor inicial de la ROM), si coincide significa que la memoria está sin usar, por lo que se borra el registro cuenta. Antes de realizar la instrucción de test del bit Z, se pasa al banco 0 ya que el registro CUENTA se encuentra en el banco 0 y la instrucción bcf no afecta al flag Z. (Figura 187)

```
Dato_inicial    bsf     STATUS,RP1
                movf    EEDATA,W    ; NO --> CONTADOR = DATO GUARDADO
                bcf     STATUS,RP1  ; Banco 0
                movwf   CUENTA      ;
                goto    Display
```

Figura 188. *Dato de salida.*

En el caso de que el valor de la dirección 0x00 de la ROM sea distinto de 0xFFh, entrará en el banco 2 para copiar el valor del registro en CUENTA, hay que tener cuidado a lo hora de programar el uso correcto de los banco, ya que CUENTA se encuentra en el banco 0 y EEDATA en el banco 2.(Figura 188)

- *Bucle principal*

```
Display     movf    CUENTA,W
            call    Tabla
            movwf   PORTB       ; Visualizo el contador en el puerto B
            btfss   PORTA,0     ; Miro si se activa el pulsador
            goto    Display
            Delay   10 Milis    ; Para evitar rebotes
            incf    CUENTA      ; Aumento en uno contador
            movlw   maximo+1
            subwf   CUENTA,W
            btfsc   STATUS,Z    ; Contador = 10 ?
            clrf    CUENTA
            movf    CUENTA,W    ; Paso contador a W
            bsf     STATUS,RP1  ; Banco 2
            movwf   EEDATA      ; EEDAT = CONTADOR
            clrf    EEADR       ; EEADR = 0x00h
            call    Escribe_eeprom
            goto    Display
```

Figura 189. *Programa contador.*

Una vez cargado el valor de inicial de CUENTA, se llamará a la función Tabla que devuelve en W la conversión del dígito en binario a un display de siete segmentos (ver **práctica 3**) y será enviado al puerto B.(Figura 189).

Luego se comprueba si ha sido pulsado la entrada RA0, mientras no se halla pulsado refrescará el display, cuando se pulse, mediante la librería Delay (**ver práctica 5**) se introduce una temporización de 10 ms para evitar los rebotes del pulsador.

Después se incrementa CUENTA, y se compara con el valor máximo de un dígito más uno (10) mediante una resta, si son iguales significa que el contador se ha desbordado y se resetea, si no son iguales se deja como está.

Por último se guarda el dato en la ROM, para lo cual se indica la dirección a guardar de la ROM (0x00h) en el registro EEADR y el dato a guardar (CUENTA) en el registro EEDATA, ambos registros se encuentran en el banco2.

Una vez que los registros tienen los parámetros se llama a la subrutina Escribe_eeprom que almacenará el dato en la ROM.

- *Simulación*

La configuración de la EEPROM es la siguiente, (Figura 190):

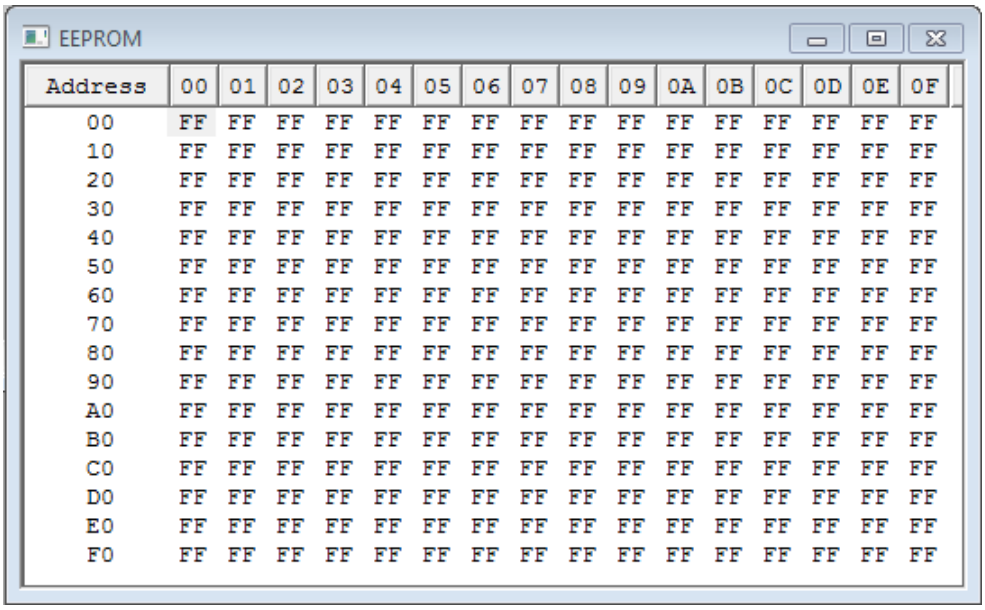


Figura 190.- *EEPROM Inicial.*

Antes de la subrutina, se indica el valor de EEADR del banco 2, (Figura 191):

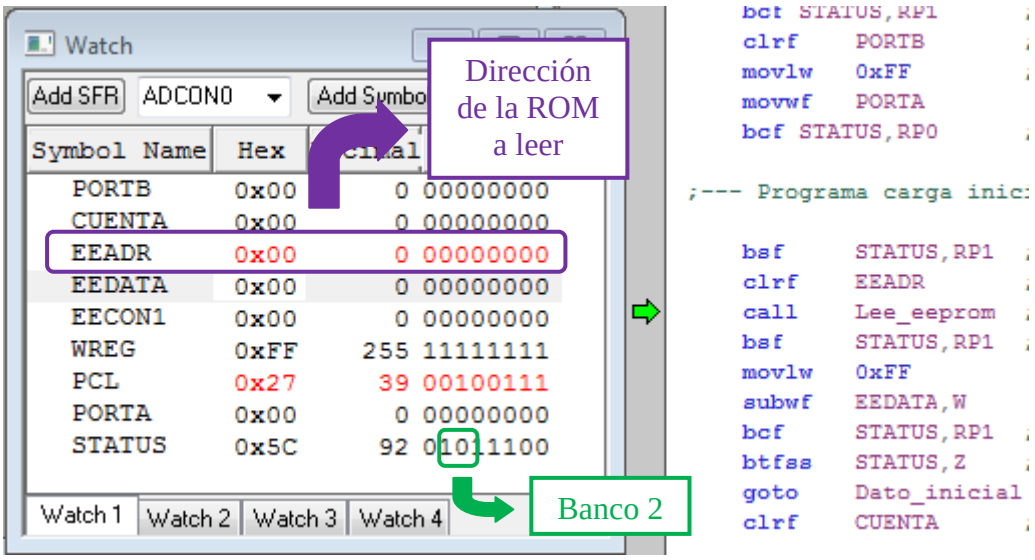


Figura 191. *Registros antes de leer la eeprom.*

A la salida de la subrutina se tiene el dato de la ROM en el registro EEDATA después hay que pasar al banco cero, (figura 192)

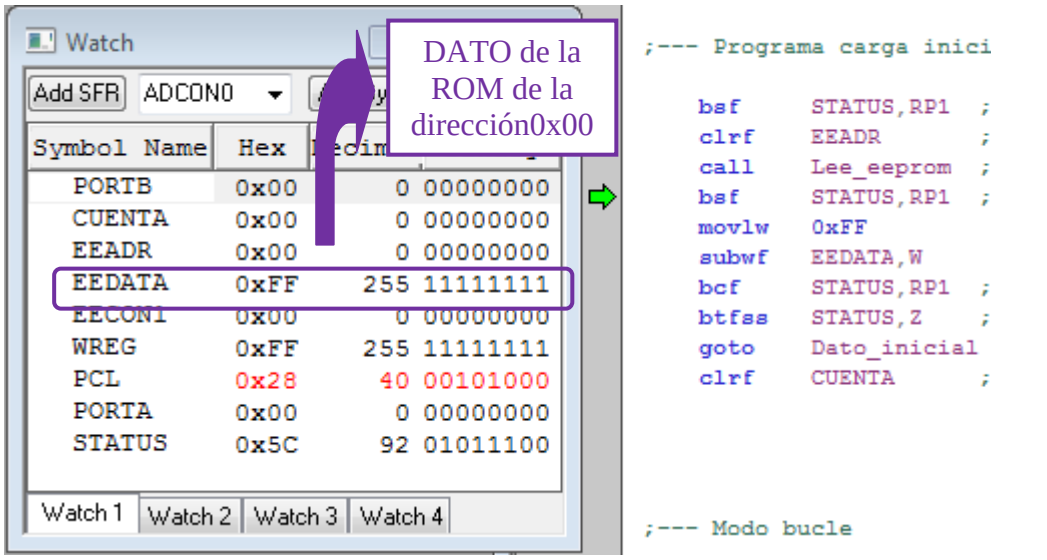


Figura 192. *Salida de la subrutina de lectura.*

Como el dato es FF, al compararlo saltará el flag de cero , y pondremos la cuenta a cero, (Figura 193)

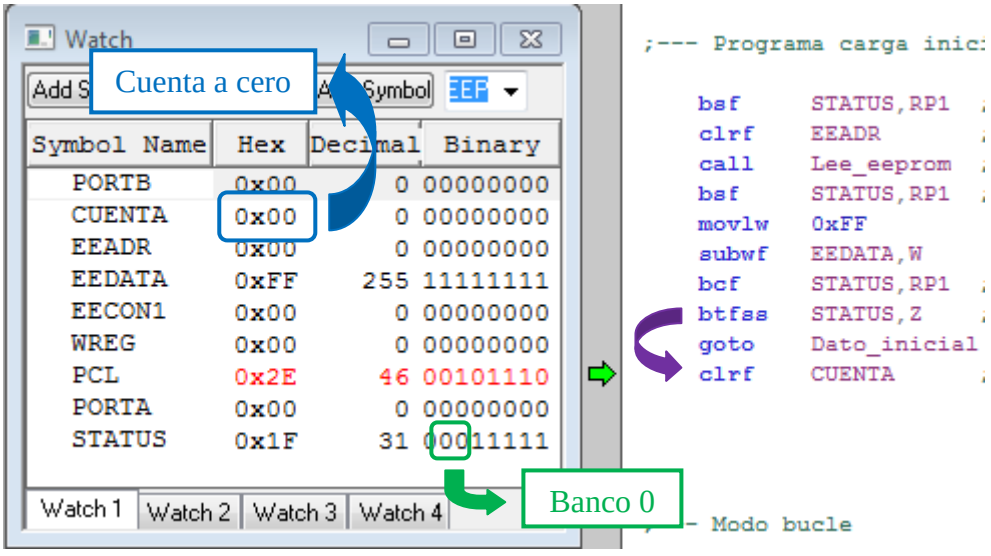


Figura 193. *Cuenta a cero.*

A continuación se muestra el bucle para una cuenta que estuviera en 4, antes de entrar en la subrutina tabla los valores de los registros más importantes son, (Figura 194).

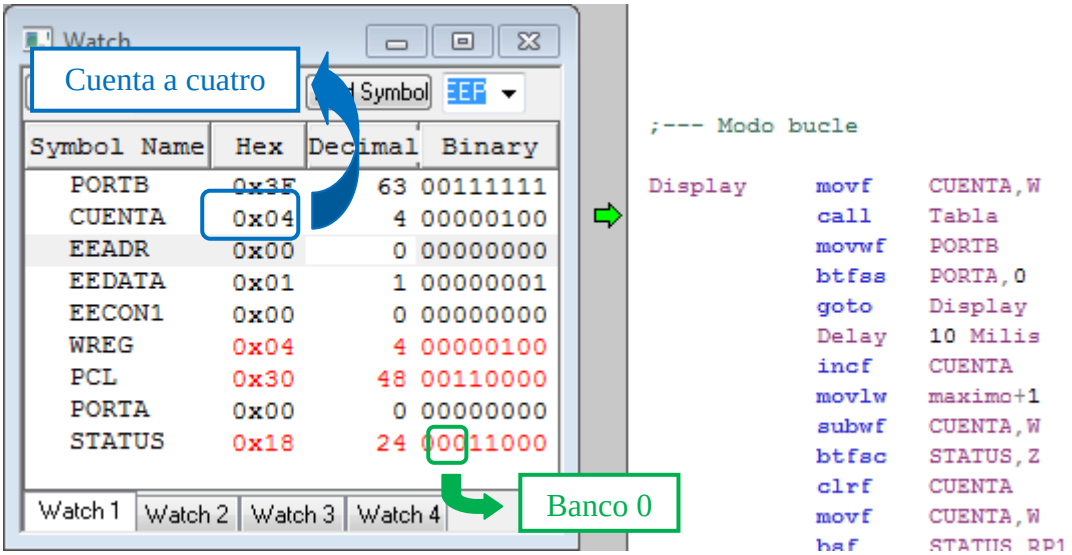


Figura 194. *Iteración cuatro.*

Y en la memoria EEPROM se encuentra almacenado el anterior número, (figura 195):

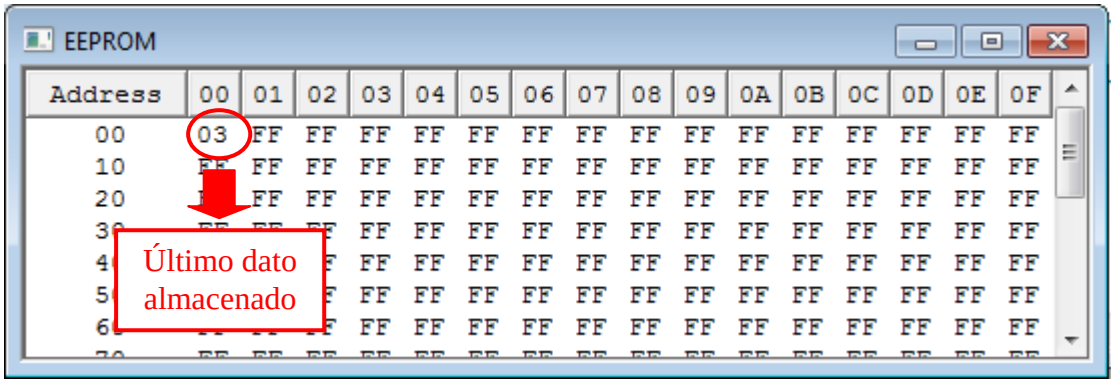


Figura 195. *EEPROM.*

La subrutina devuelve el valor a pasar al display en el registro de trabajo y posteriormente se modifica el display, (figura 196)

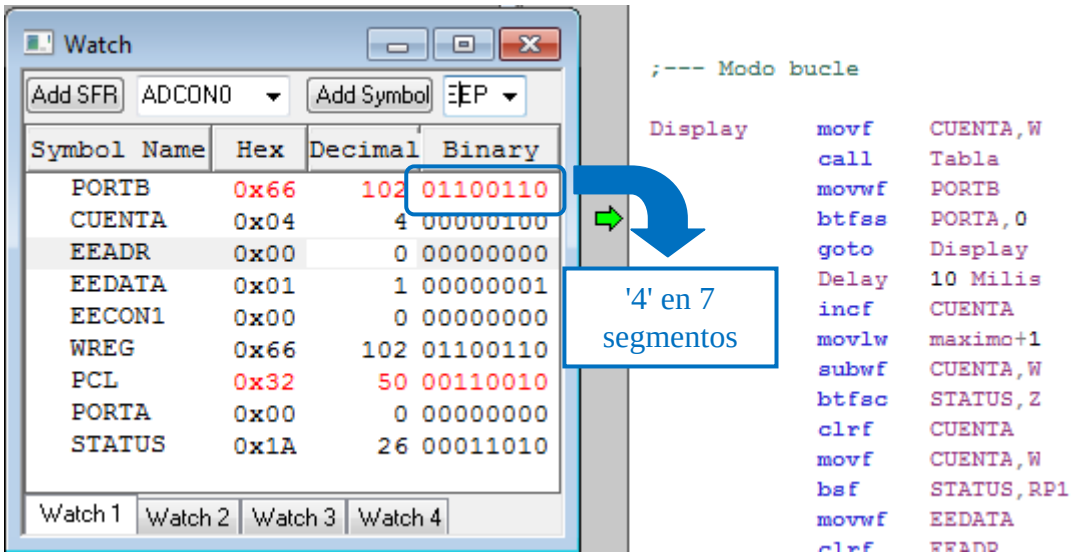


Figura 196. *Salida al display.*

Una vez activado el pulsador y esperado el tiempo suficiente para eliminar los rebotes, la cuenta se incrementa, como no llega a 10, se carga el valor de 5 en el EEDATA, (Figura 197):

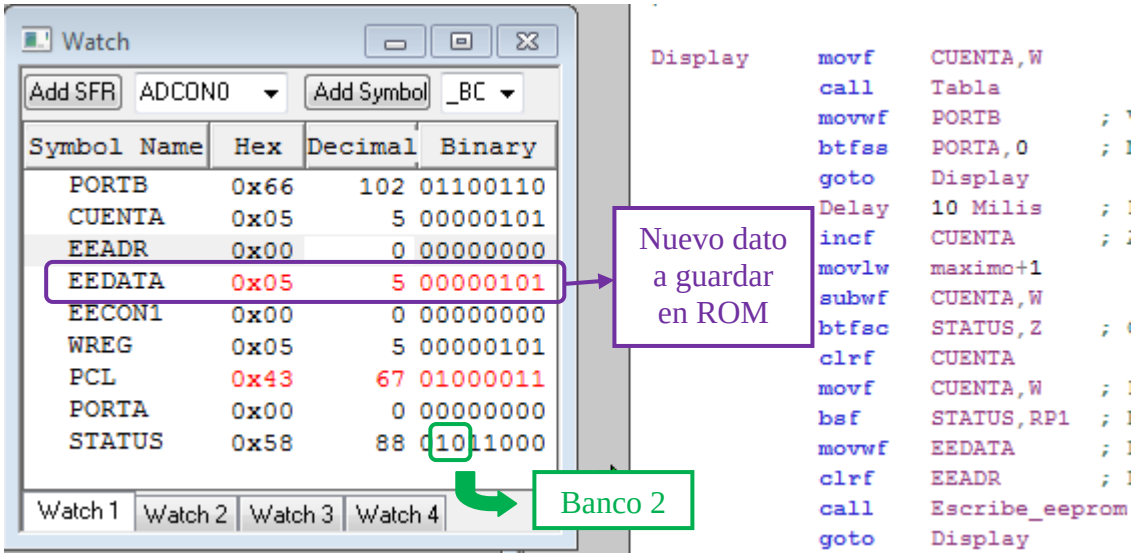


Figura 197. *Grabación en EEPROM.*

Una vez que se sale de la subrutina se ha modificado la ROM, (Figura 198 y 199):

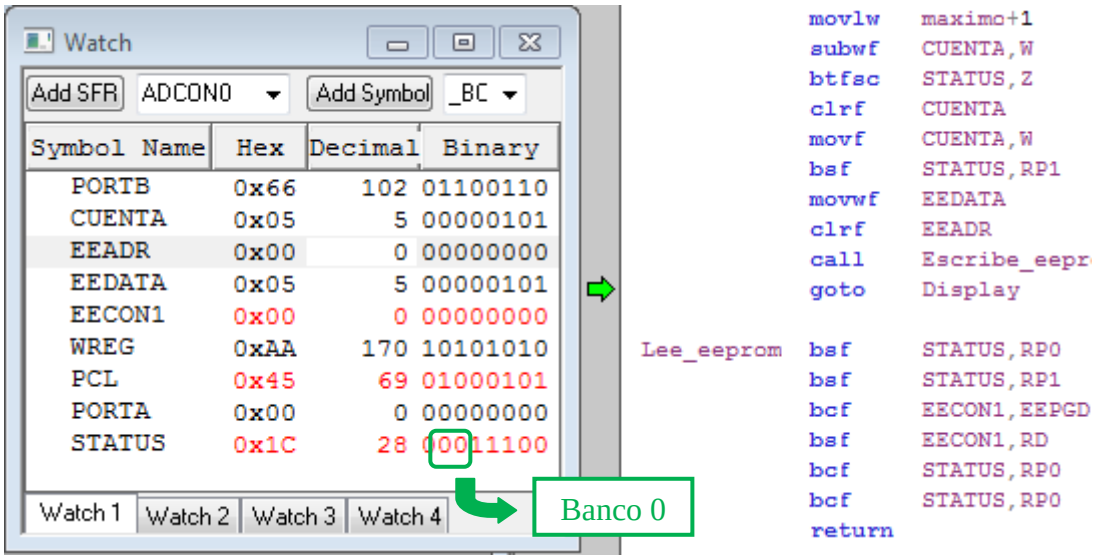


Figura 198. *Retorno de subrutina.*

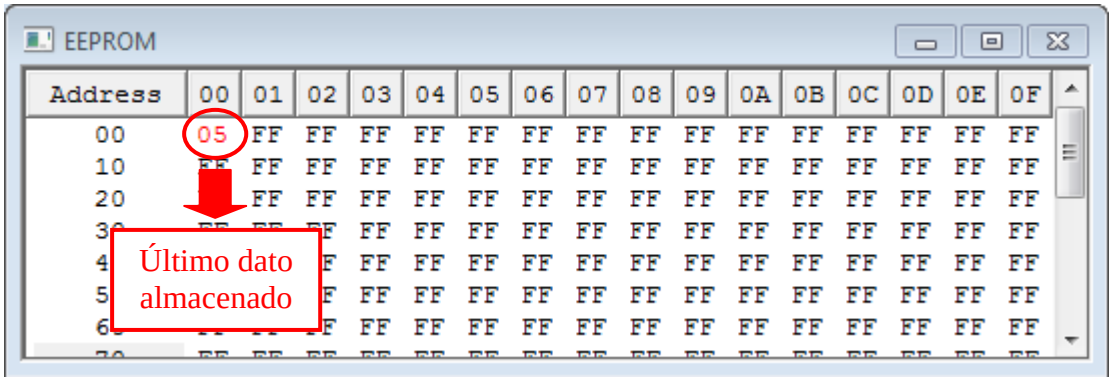


Figura 199. *EEPROM.*

1.1.5.4.7.4.3.2 - Subrutina tabla

La subrutina tabla devuelve en el registro de trabajo la codificación de un display de 7 segmentos, (Figura 200):

```
;-------------------------------------------------------
;---            TABLA DEL DISPLAY                      -
;-------------------------------------------------------

Tabla
    addwf   PCL
    retlw   cero        ; LED's activados para el cero en un display
    retlw   uno         ; LED's activados para el uno en un display
    retlw   dos         ; LED's activados para el dos en un display
    retlw   tres        ; LED's activados para el tres en un display
    retlw   cuatro      ; LED's activados para el cuatro en un display
    retlw   cinco       ; LED's activados para el cinco en un display
    retlw   seis        ; LED's activados para el seis en un display
    retlw   siete       ; LED's activados para el siete en un display
    retlw   ocho        ; LED's activados para el ocho en un display
    retlw   nueve       ; LED's activados para el nueve en un display
    retlw   fallo       ; LED's activados para la E en un display
```

Figura 200. *Código de subrutina Tabla.*

Mediante el valor del contador del programa se selecciona la línea de programa deseada (**ver práctica 3**)

1.1.5.4.7.4.3.3 - Subrutina Lee_eeprom

- *Código*

```
Lee_eeprom  bsf     STATUS,RP0
            bsf     STATUS,RP1  ; Banco 3
            bcf     EECON1,EEPGD    ;EEPROM de datos
            bsf     EECON1,RD       ;Orden de lectura
            bcf     STATUS,RP0
            bcf     STATUS,RP0      ;Selección de banco 0
            return
```

Figura 201. *Código de subrutina Lee_eprom.*

Antes de llamar a la subrutina se le pasa el parámetro EEADR, indicando la dirección de la ROM a leer

Una vez dentro se da la orden de lectura borrando el bit EEPGD y seteando el bit RD ambos contenidos en el registro EECON1 del banco 3. (Figura 201).

↳ *Simulación*

Antes de entrar a esta subrutina los valores de los registros más importantes se pueden observar en la (figura 9)**.** Primero se pasa al banco 3 para poder acceder a los registros de control de la ROM y se borra el bit que indica el tipo de memoria y se activa el bit de permiso de escritura, (Figura 202):

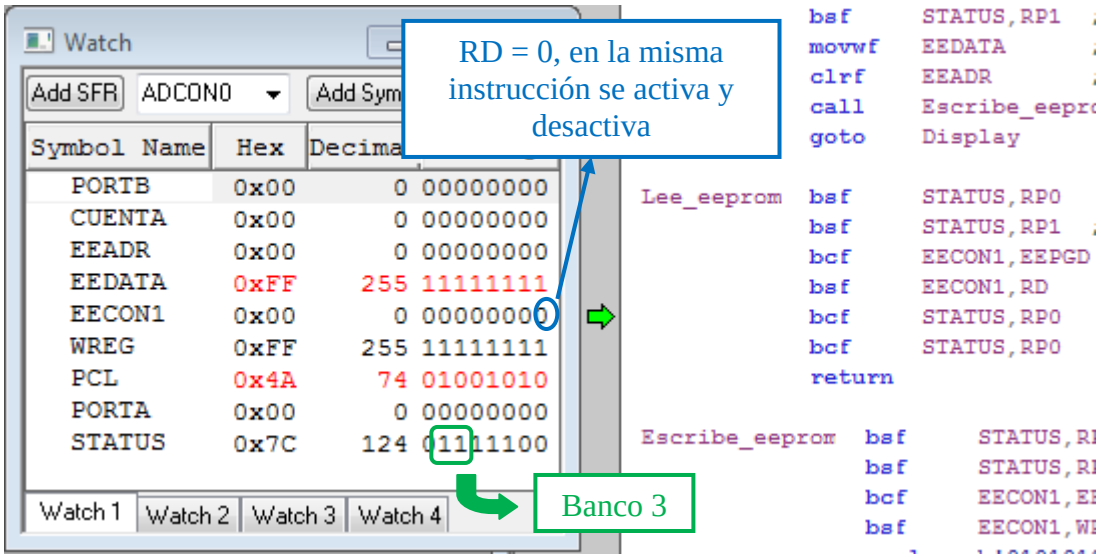


Figura 202. *Subrutina de lectura.*

1.1.5.4.7.4.3.4 - Subrutina Escribe_eeprom

↳ *Código*

```
Escribe_eeprom  bsf     STATUS,RP0
                bsf     STATUS,RP1      ;banco 3
                bcf     EECON1,EEPGD    ;EEPROM de datos
                bsf     EECON1,WREN     ;Permiso de escritura
                movlw   b'01010101'
                movwf   EECON2
                movlw   b'10101010'
                movwf   EECON2          ;Secuencia establecida por Microchip
                bsf     EECON1,WR       ;Orden de escritura
Espera          btfsc   EECON1,WR       ;Testear flag de fin de escritura
                goto    Espera
                bcf     EECON1,WREN     ;Fin permiso de escritura
                bcf     EECON1,EEIF     ;Borra el flag de fin de escritura
                bcf     STATUS,RP0
                bcf     STATUS,RP1      ;Selecciona banco 0
                return
```

Figura 203. *Subrutina de escritura.*

Antes de entrar en la subrutina deben estar cargados los **registros EEADR indicando la dirección a escribir y EEDAT que contiene el dato a escribir**. A continuación se indica al PIC que se va a escribir en la EEPROM mediante los registros EECON1 y EECON2 pertenecientes al banco 3. En el EECON1 se activan los bits EEPGD y WREN y en la EECON2 se introducen la secuencia descrita por microchip, primero b'01010101' y a continuación b'10101010'. (Figura 203)

Una vez establecido que se puede grabar en la ROM hay que dar la orden de escritura activando el bit WR, el bit WR ese pone a cero cuando acaba la escritura, por lo que se realiza un bucle testeando ese bit hasta que se pone a cero. Al acabar de escribir en la eeprom se debe borrar el bit de permiso de escritura (WREN) y el flag de fin de escritura (EEIF).

- *Simulación:*

Antes de entrar a la subrutina de escritura los valores de los registros más importantes se encuentran en la (figura 204). Dentro de la subrutina se activan todos los bits necesarios de los registros EECON1 y EECON2:

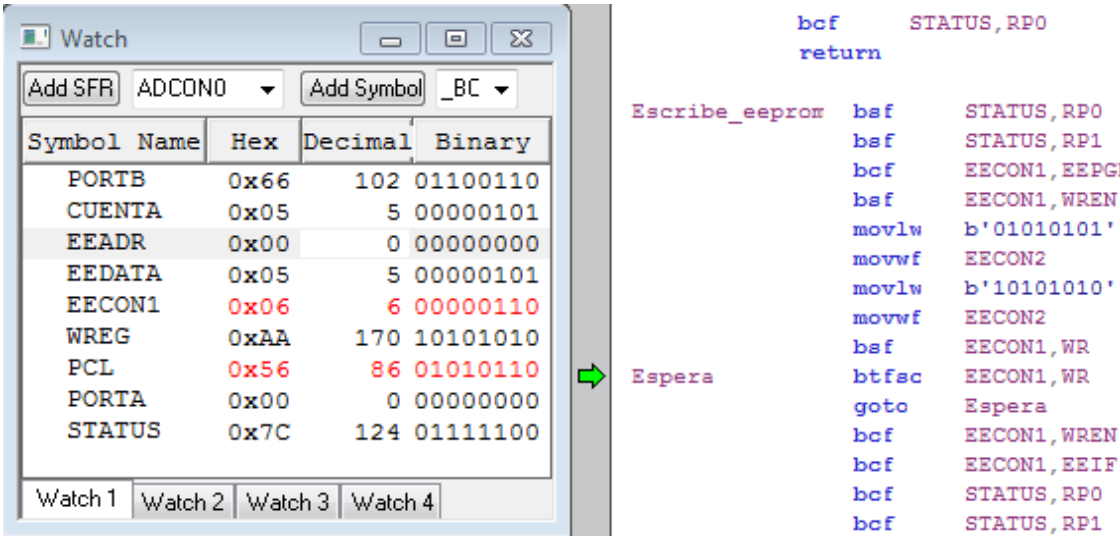


Figura 204. *Subrutina de escritura.*

Cuando se acaba de escribir, el bit WR se pone a cero y la ROM se modifica (ver figura 198).

1.1.5.4.7.4.3.4.- Montaje

Para la realización de esta práctica es necesario el uso del display de 7 segmentos (Figura 205):

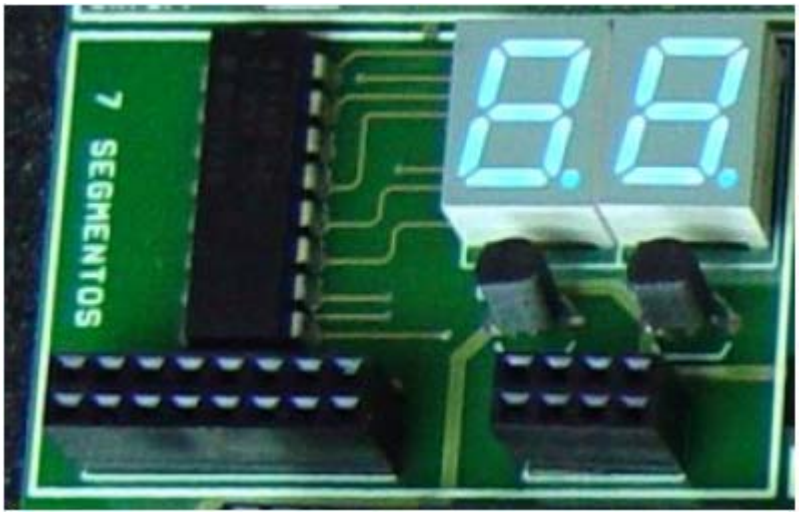


Figura 205. *PIC'School Display.* [5]

El puerto B será conectado al display y el pin RA0 al pulsador E4 como indica la figura, (Figura 206):

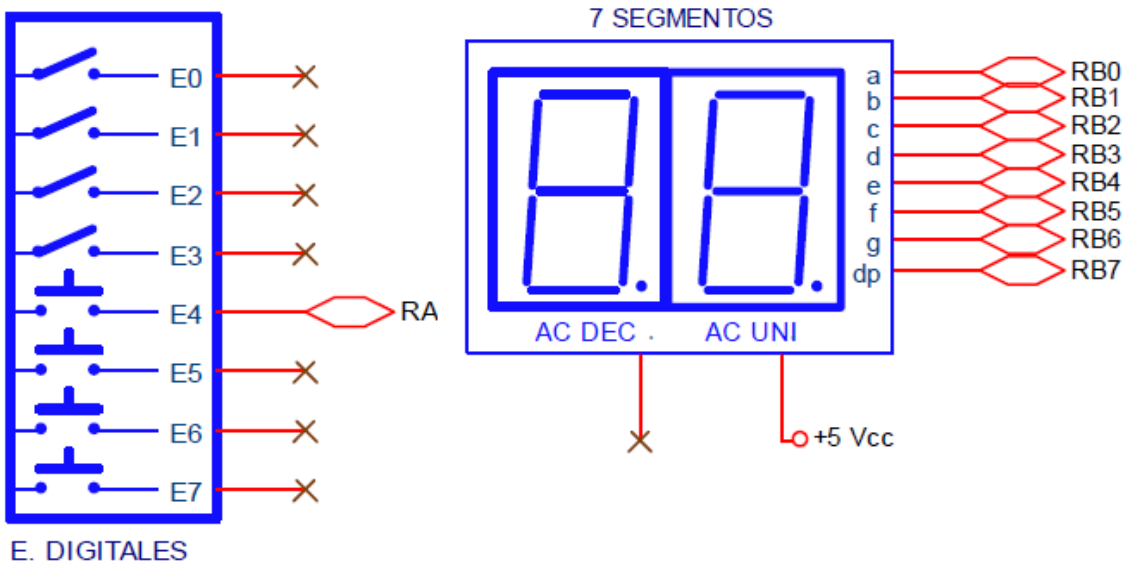


Figura 206. *Esquema de montaje.[5]*

1.1.5.4.7.5.- Programas propuestos

Realizar un programa de una simulación de pantalla en blanco(0) y negro(1), la pantalla contendrá 2x2 pixeles que serán simulados por las entradas RB0-RB3 y usaran las salidas RA0-RA3. Tendrá un botón de apagado (RA4) , cada vez que se apague se podrán las salidas a cero, se almacenará en la EEPROM y al encender cargará la configuración de la EEPROM.

1.1.5.4.8.- Práctica 8: LCD

1.1.5.4.8.1.- Objetivos

- ✓ Comprender el funcionamiento de la pantalla LCD
- ✓ Entender el funcionamiento de la librería LCD4bitsPIC16.INC

1.1.5.4.8.2.- Enunciado

Realizar un programa que saque en la pantalla LCD el mensaje 'Hola'.

1.1.5.4.8.3.- Contenido teórico

La pantalla LCD (Liquid Crystal Display) es un dispositivo para la visualización de caracteres y símbolos. Es capaz de visualizar *dos filas de 16 caracteres* cada una, siendo cada carácter una matriz de *pixeles de 7x5*. Se usa en el PIC school la LCD HD44780 . Los pines correspondientes a la LCD se representan en la Tabla 20:

Tabla 20. *Patillas de la LCD.* [2]

PIN	SÍMBOLO	DESCRIPCIÓN
1	GND	0V
2	Vcc	5V
3	Vo	Control de contraste
4	RS	0 = registro de control
		1 = registro de datos
5	R/W	0 = Escritura
		1 = Lectura
6	E	0 = LCD off
		1 = LCD on
7-14	D0-D7	Bus de datos
15	LED+	5V
16	LED-	0V

El esquema de conexión de la LCD sería el siguiente, (Figura 1):

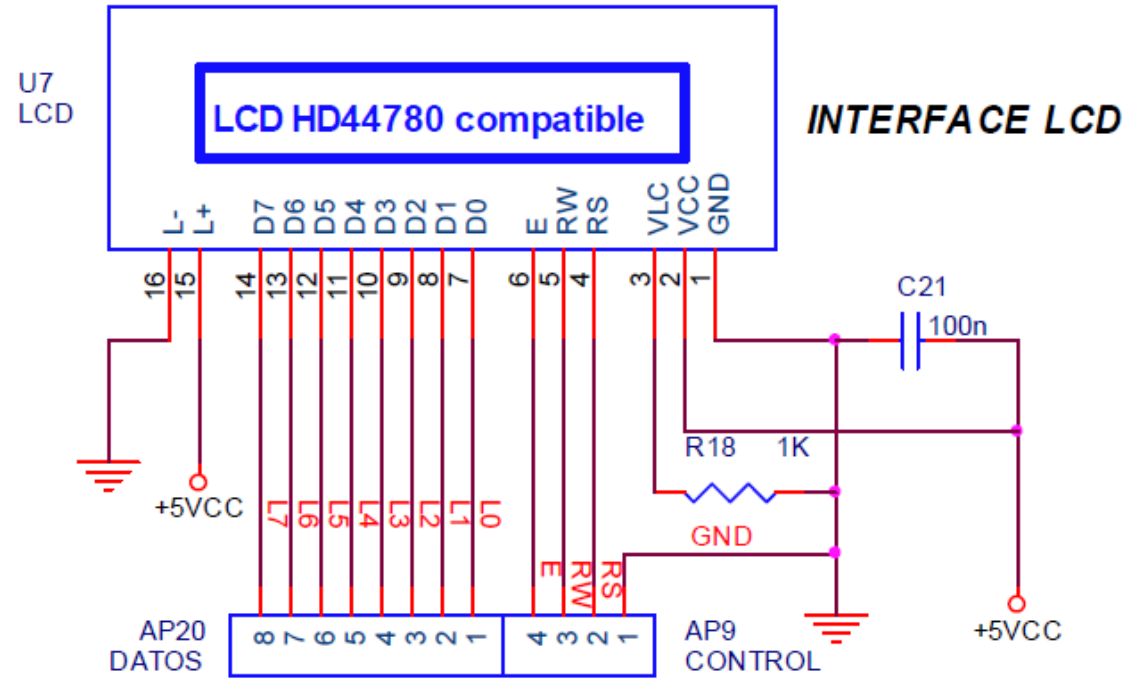


Figura 207. *Conexión del LCD.*[5]

Los instrucciones más utilizadas se muestran en la tabla 22:

Tabla 21. *Instrucciones de la LCD.* [2]

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Descripción
0	0	0	0	0	0	0	0	0	0	Clear
0	0	0	0	0	0	0	0	1	-	Cursor inicio.
0	0	0	0	0	0	0	1	I/D	S	Dirección movimiento
0	0	0	0	0	0	1	D	C	B	Control
0	0	0	0	0	1	S/C	R/L	-	-	Mueve cursor
0	0	0	0	1	DL	N	F	-	-	Control
0	0	0	1	X	X	X	X	X	X	CGRAM adress
0	0	1	X	X	X	X	X	X	X	DDRAM adress
0	1	BF	X	X	X	X	X	X	X	Busy flag
1	0	X	X	X	X	X	X	X	X	Dato a escribir
1	1	X	X	X	X	X	X	X	X	Dato a leer

- I/D = 1: Incremento del cursor
- I/D = 0: Decremento del cursor
- S = 1: Desplaza la visualización cada vez que se escribe un dato.
- S/C = 1: Desplaza el display.
- S/C = 0: Mueve el cursor.
- R/L = 1: Desplazamiento a la derecha.
- R/L = 0: Desplazamiento a la izquierda.
- DL = 1: 8 bits, DL = 0: 4 bits
- N = 1: 2 líneas LCD, N = 0: 1 línea LCD
- F = 5 x10 píxeles , F = 5 x7 píxeles
- BF = 1: Ocupado
- BF = 0: Libre
- B=1 parpadeo on
- C=1 cursor on
- D=1 display ON

La secuencia de inicio del LCD se explica en el siguiente diagrama, (Figura 2):

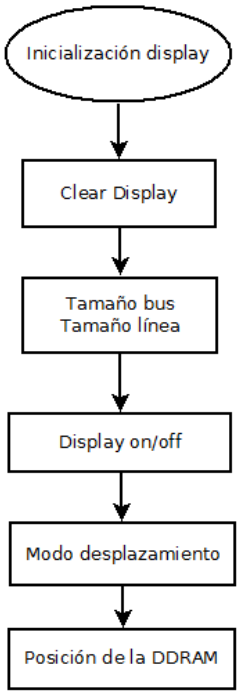


Figura 208. *Secuencia de inicio del LCD.*

Para el uso de la pantalla LCD se recomienda el uso de librerías, al igual que ocurre con la temporización. En nuestro caso se usará la librería "LCD4bitsPIC16.INC" programada por **Microsystems Engineer de Mikel Etxebarria.** De la misma forma que el temporizador existen una serie de instrucciones necesarias para el uso de la librería:

- Lcd_var equ 0x70: Registros temporales (3) reservados para el uso de la librería.
- include "LCD4bitsPIC16.INC": Incorpora la librería a nuestro programa.
- call nombre_rutina: Es la llamada desde el programa principal, en la siguiente tabla se resumen las rutinas más importantes de la librería:

Tabla 22. *Rutinas del programa. [5]*

NOMBRE	IN	OUT	DESCRIPCIÓN
UP_LCD	-	-	Configura las E/S para adaptarlas a la LCD
LCD_INI	-	-	Rutina de inicio del LCD
LCD_DATO	W = Carácter	-	Envía a la pantalla el dato a visualizar
LCD_REG	W = Comando	-	Envía el comando a ejecutar

Esta librería se encuentra en el anejo de códigos, y en ella se pueden observar los valores a introducir al LCD, (Figura 209)

```
;Se supone un interface de 4 bits. RB0-RB3 es la puerta de datos.
;RA1-RA3 son las señales de control.
;Estas conexiones se pueden modificar según se necesite

#define ENABLE       bsf PORTA,1     ;Activa señal E (RA1)
#define DISABLE       bcf PORTA,1     ;Desactiva señal E (RA1)
#define LEER         bsf PORTA,2     ;Pone LCD en Modo RD (RA2)
#define ESCRIBIR     bcf PORTA,2     ;Pone LCD en Modo WR (RA2)
#define LCD_COMANDO   bcf PORTA,3     ;Desactiva RS (modo comando)(RA3)
#define LCD_DATOS   bsf PORTA,3   ;Activa RS (modo dato)(RA3)
#define LCD_C_PORT PORTA ;Puerta de control del LCD (RA1-RA3)
#define LCD_C_TRIS TRISA ;Control de la puerta de control del LCD
#define LCD_PORT PORTB ;Puerta de datos del LCD (RB0-RB3)
#define LCD_TRIS TRISB ;Control de la puerta de datos del LCD
```

Figura 209. *Constantes del programa LCD4bitsPIC16.INC*

Así por ejemplo :

- Si se quiere activar la LCD se pondrá a uno la entrada RA1.
- Si se quiere escribir en la LCD RA2 debe estar a cero.
- Si se trata de un dato RA3 debe estar a 1.

1.1.5.4.8.4.- Ejercicio

1.1.5.4.8.4.1.- Planteamiento

Se trata de visualizar la palabra *Hola* en la pantalla LCD, para ello se hará uso de la librería *LCD4bitsPIC16.INC* y deberá ser incluida en el programa mediante software con la directiva *include*.

Para escribir un carácter en la pantalla basta con mover al registro de trabajo el carácter deseado y llamar a la subrutina LCD_DATO.

1.1.5.4.8.4.2.-Diagrama de flujo

Lo primero que se debe hacer es configurar los puertos, después se llama a la librería para que configure los puertos usados en la LCD.

A continuación la inicialización de la LCD usando la librería. Se le pasa al LCD las instrucciones deseadas: parpadeo, cursor y LCD, según se indica en la siguiente figura, (Figura 210).

Una vez que está listo el LCD para escribir, se pone en el registro de trabajo el carácter a visualizar, y se llama a la función encargada de visualizar el dato que está en W.

Hay que realzarlo para cada letra del mensaje, se pueden definir los mensajes como posiciones de memoria de datos, para ello el primer carácter al asignar la posición se introduce el carácter '$', lo que indica que el resto de caracteres irán en posiciones consecutivas de memorias.

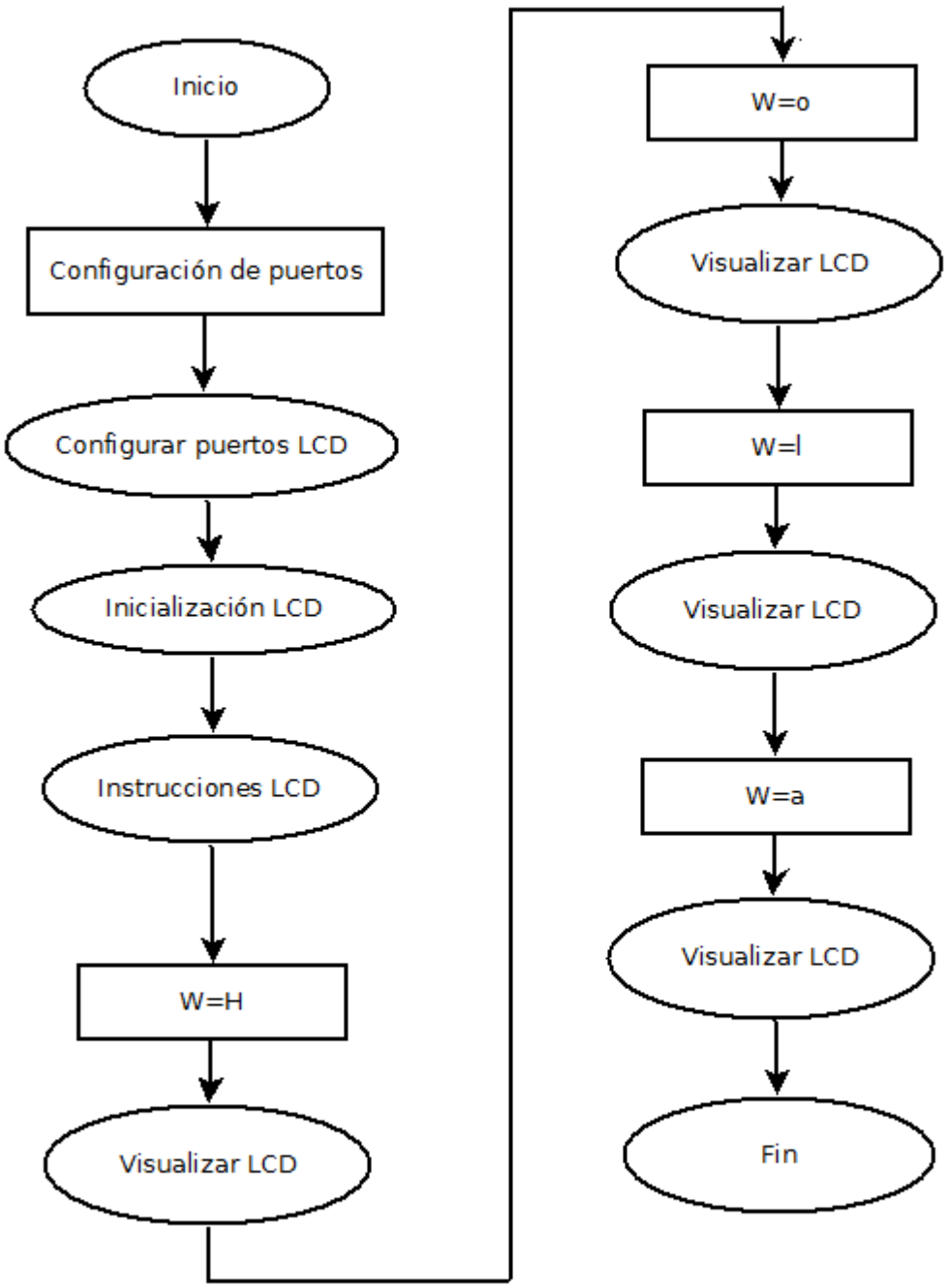


Figura 210. *Diagrama de flujo.*

1.1.5.4.8.4.3.-Programación y simulación

Para este programa solo es necesario reservar 3 espacios para los registros temporales usados en la librería que controla el LCD.(Figura 211)

```
;-----------------------------------------------------
;---    DECLARACIÓN DE CTES. Y VARIABLE               -
;-----------------------------------------------------
    Lcd_var     equ 0x70    ;Variables (3) del LCD
```

Figura 211. *Registros y variables.*

↳ *Código*

```
Inicio                  ; Etiqueta de comienzo de programa

;--- Configuración de los puertos ---
    clrw
    clrf    PORTA
    clrf    PORTB   ; Borra los latch de salida
    bsf STATUS,RP0
    bsf STATUS,RP1  ; Banco 3
    clrf    ANSEL   ; Entradas digitales
    clrf    ANSELH
    bcf STATUS,RP1  ; Banco 1
    clrf    PORTB   ; Puerto B (Out)
    clrf    PORTA   ; Puerto A (Out)
    bcf STATUS,RP0  ; Banco 0

;--- Programa LCD ---
    call    UP_LCD      ; Puertos LCD
    call    LCD_INI     ; Inicialización del LCD
    movlw   b'00001111' ; LCD=On, Cursor = on, Parpadeo = On
    call    LCD_REG
    movlw   'H'
    call    LCD_DATO    ; Visualizo H
    movlw   'o'
    call    LCD_DATO    ; Visualizo o
    movlw   'l'
    call    LCD_DATO    ; Visualizo l
    movlw   'a'
    call    LCD_DATO    ; Visualizo a

Fin goto    Fin
End
```

Figura 212. *Programa principal.*

La primera parte del código configura los puertos a y b como salidas digitales (**ver práctica 4**). Después llama la librería de la LCD con la subrutina UP_LCD que configura los puertos de la pantalla (Figura 6). A continuación inicializa la LCD con la subrutina LCD_INI, esta se encarga de limpiar la pantalla, activarla, y decir el número de líneas (2 en nuestro caso) los pixeles por carácter etc.

La siguiente instrucción sirve para indicar como queremos el cursor, en este caso se ha optado por visualizar el cursor y que parpadee (ver **tabla XX**). Por último para escribir la palabra deseada simplemente letra a letra la vamos copiando al registro de trabajo y llamamos a la subrutina LCD_DATO para que pase el dato a la LCD y lo visualice por pantalla uno a continuación de otro. (Figura 212).

- *Simulación:*

La configuración de los puertos queda de la siguiente forma, (Figura 213)

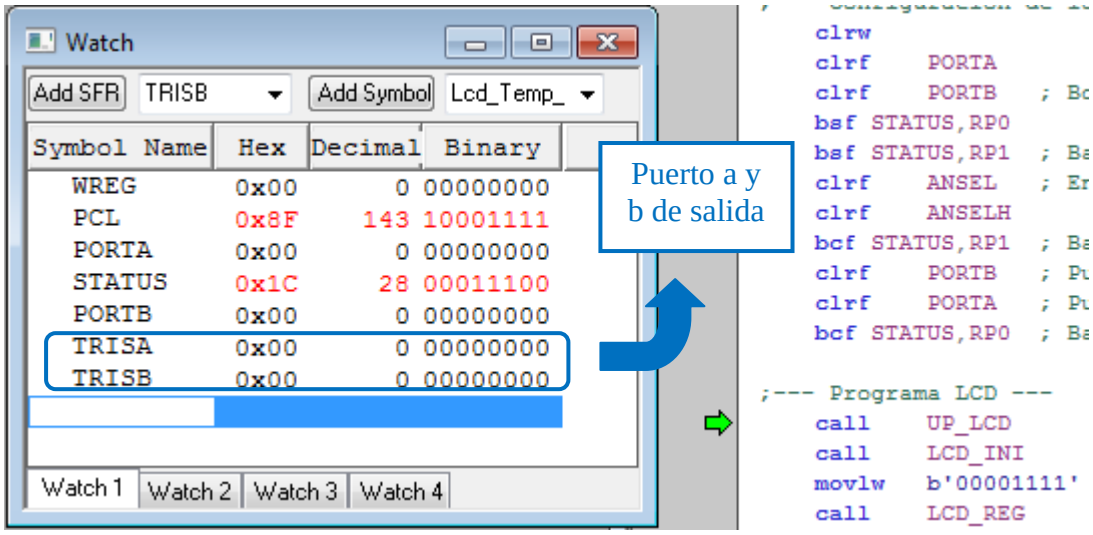


Figura 213. *Configuración de puertos.*

El primer valor a visualizar H corresponde con la codificación 0x72h, (Figura 214):

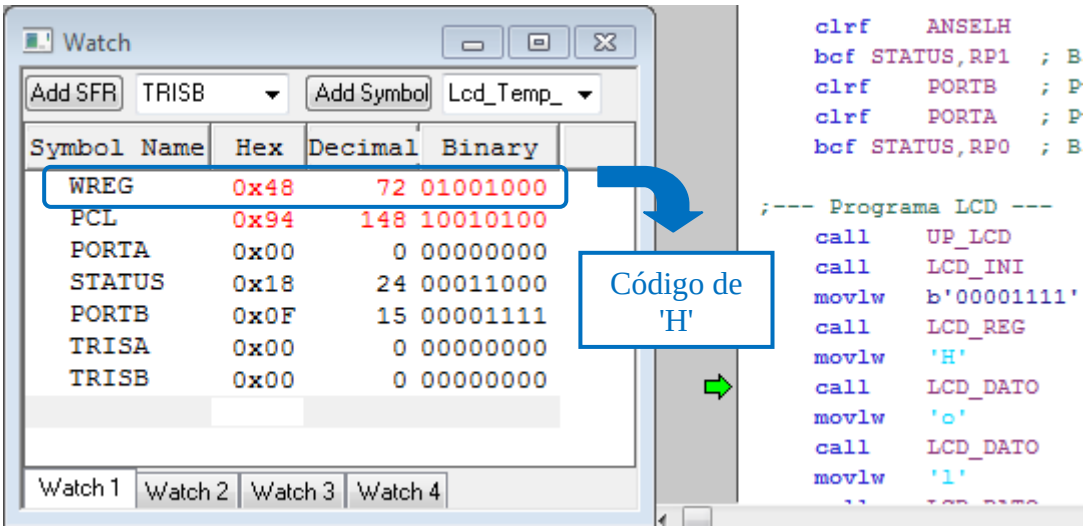


Figura 214. *Letra H.*

Para el segundo dato, (Figura 215):

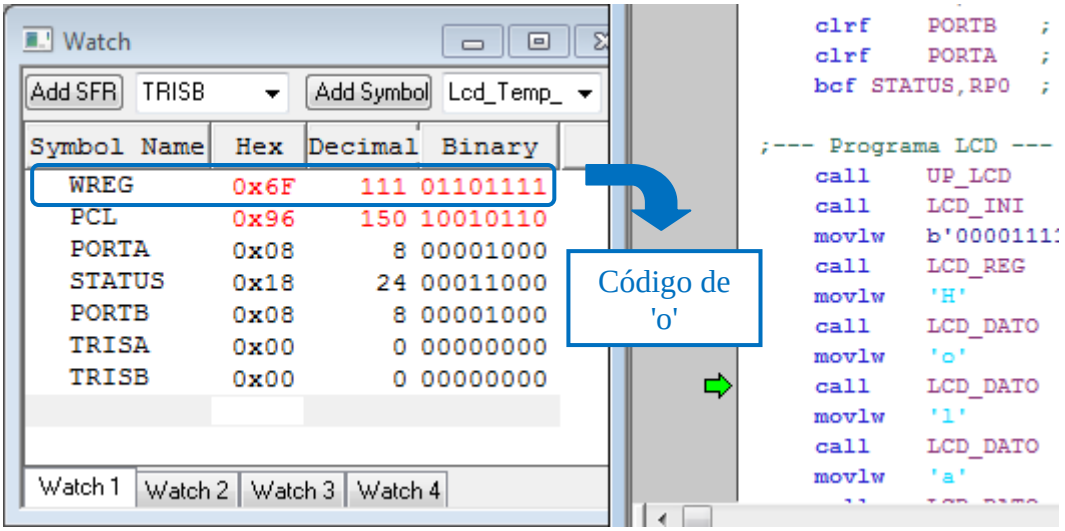


Figura 215. *Letra o.*

Para el tercer dato, (Figura 216):

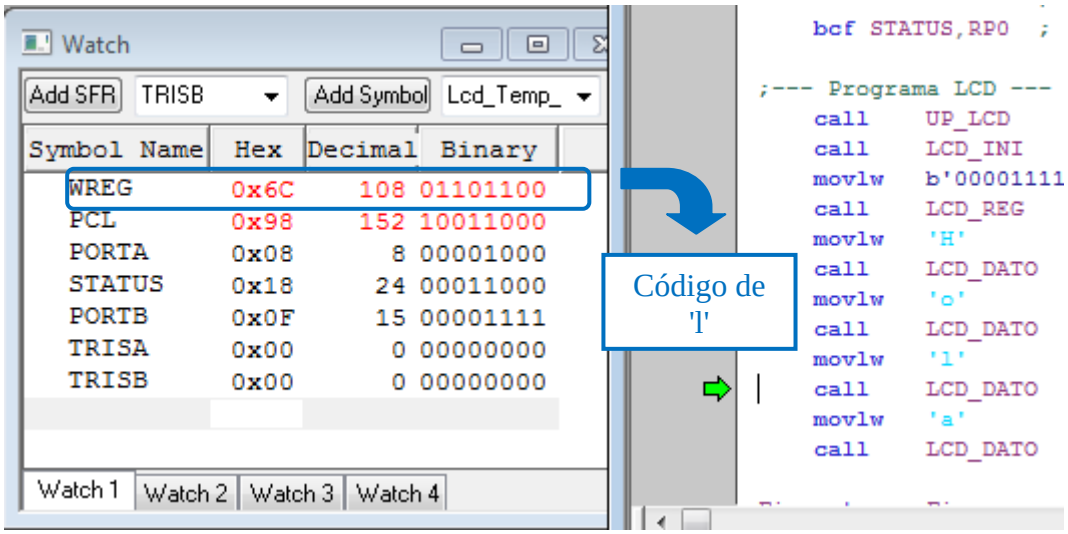


Figura 216. *Letra l.*

Y por último la letra a, (Figura 217):

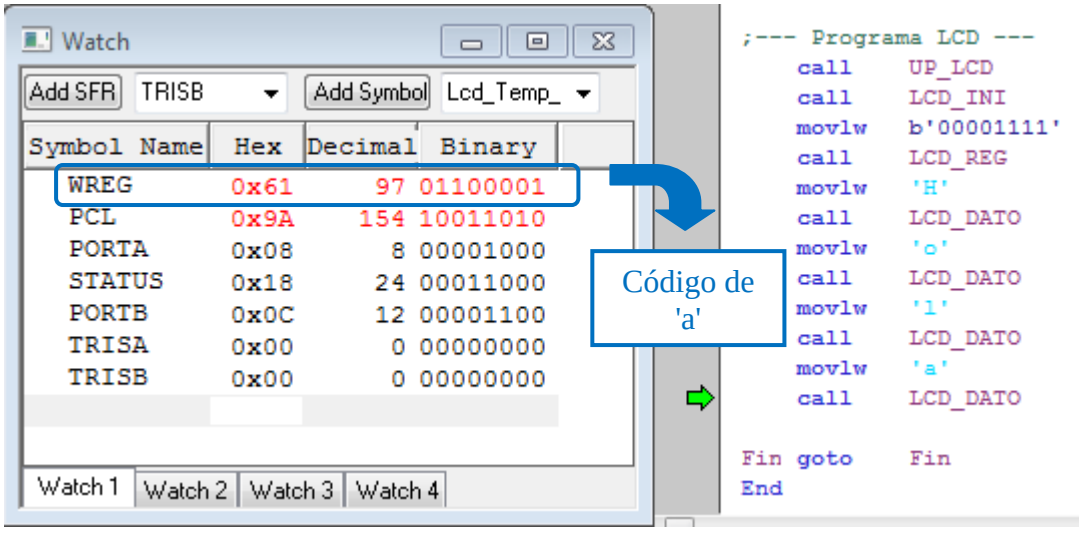


Figura 217. *Letra a.*

Finalizando así la visualización de la palabra Hola.

1.1.5.4.8.4.4.- Montaje

Para la conexión de este ejercicio las líneas del puerto b RB0-RB3 se conectan a las líneas de datos de la pantalla D4-D7, y las salidas del puerto a RA1-RA3 al control del dispositivo como se indica en la figura 12.

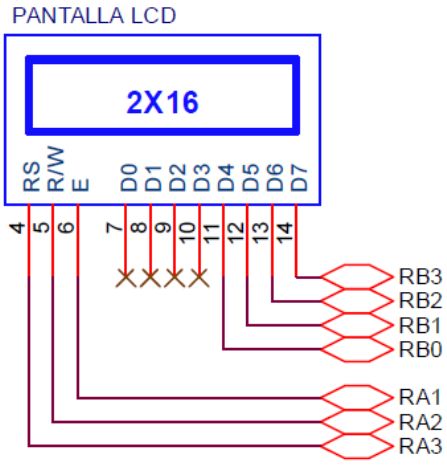


Figura 218. *Esquema de conexiones.* [5]

Obteniendo así el resultado deseado, (Figura 13):

Figura 219. *Resultado.* [5]

1.1.5.4..5.- Programas propuestos

Realizar un programa que visualice por pantalla tres mensajes cualesquiera, para usar estos mensajes, en vez de ir letra a letra crear una tabla para cada mensaje.

1.1.5.4.9.- Practica 9 - Teclado matricial

1.1.5.4.9.1- Objetivos

- ✓ Comprender el funcionamiento de un teclado matricial.
- ✓ Entender el funcionamiento de la librería TECLADO.INC

1.1.5.4.9.2- Enunciado

Programar un teclado matricial 4x4, de tal forma que al pulsar 3 teclas si coinciden con una contraseña guardada en el ROM se active un led.

1.1.5.4.9.3.- Contenido teórico

El teclado matricial funciona de la siguiente manera, (Figura 220):

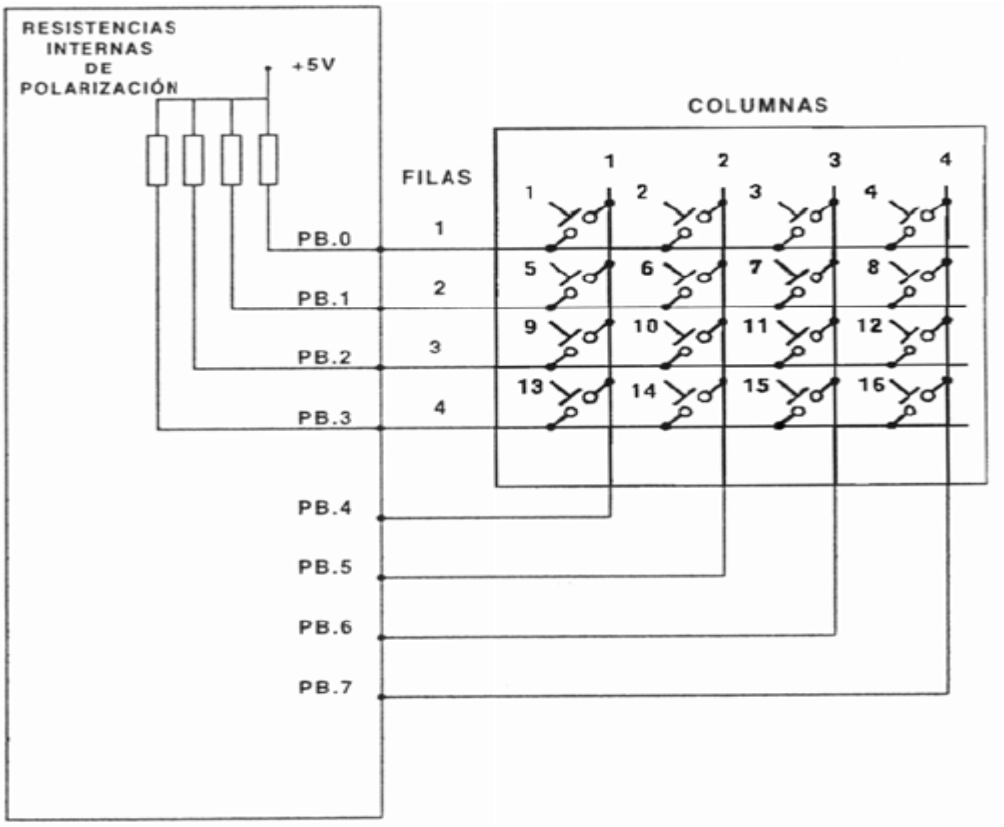


Figura 220. *Teclado matricial*.[4]

Las filas del teclado se conectan a las patillas de salidas mediante las **resistencias internas de polarización del PIC**, y las columnas están conectadas a las patillas de entrada. La obtención de la tecla pulsada se basa en un barrido de filas.

Los 4 bits de salida (filas) comienzan con un valor '1110', es decir un cero en la primera fila y el resto unos. Si la tecla pulsada se encuentra en la primera fila, "puenteará" ese cero con la columna correspondiente y aparecerá un cero lógico en la entrada de esa columna.

Por lo que bastará con ir poniendo a cero las distintas filas mientras el resto esté a cero para obtener la tecla pulsada.

Existen librerías creadas para el manejo del teclado, usaremos la librería TECLADO.INC programada por **Microsystems Engineer de Mikel Etxebarria**. Esta librería devuelve el código en BCD de la tecla pulsada y 0x80h si no ha sido pulsada ninguna tecla. Para usar esta librería al igual que las anteriores se deben usar las siguientes instrucciones [5]:

- Key_var equ 0x73h: seis variables reservadas para el uso del teclado.
- include "teclado.inc": fichero que contiene la librería
- call Key_scan: llamada a la subrutina.

1.1.5.4.9.4.- Ejercicio

1.1.5.4.9.4.1- Planteamiento

El programa cargará inicialmente la contraseña que se encuentra ubicada en la ROM en 3 registros de la RAM, después cada vez que se pulse una tecla se almacenará en otros tres registros.

Cuando se hallan pulsado 3 teclas se comparan con la contraseña si es cierta se enciende el led y si es falsa se borran los registros de las tres teclas pulsadas. *Para la captura de las teclas se usará la librería TECLADO.INC.*

El código de los dígitos de salida de la función TECLADO.INC, (Tabla 23):

Tabla 23. *Salida del teclado matricial.*

TECLA	CÓDIGO
0	01111101 = 0x7D
1	11101110 = 0xEE
2	11101101 = 0xED
3	11101011 = 0xEB
4	11011110 = 0xDE
5	11011101 = 0xDD
6	11011011 = 0xDB
7	10111110 = 0xBE
8	10111101 = 0xBD
9	10111011 = 0xBB
A	01111110 = 0x7E
B	01111011 = 0x7B
C	01110111 = 0x77
D	10110111 = 0xB7
E	11010111 = 0xD7
F	11100111 = 0xE7

1.1.5.4.9.4.2- Diagrama de flujo

Primero se configuran los puertos, a continuación se realiza un proceso de lectura de la EEPROM , copiando su contenido en la RAM (password).

Después se espera hasta que se hallan pulsado tres teclas y se comprueba que coinciden con el password.

Si coincide se enciende una luz y finaliza el programa, sino coincide se borran las teclas pulsadas a la espera de tres nuevas teclas.(Figura 221)

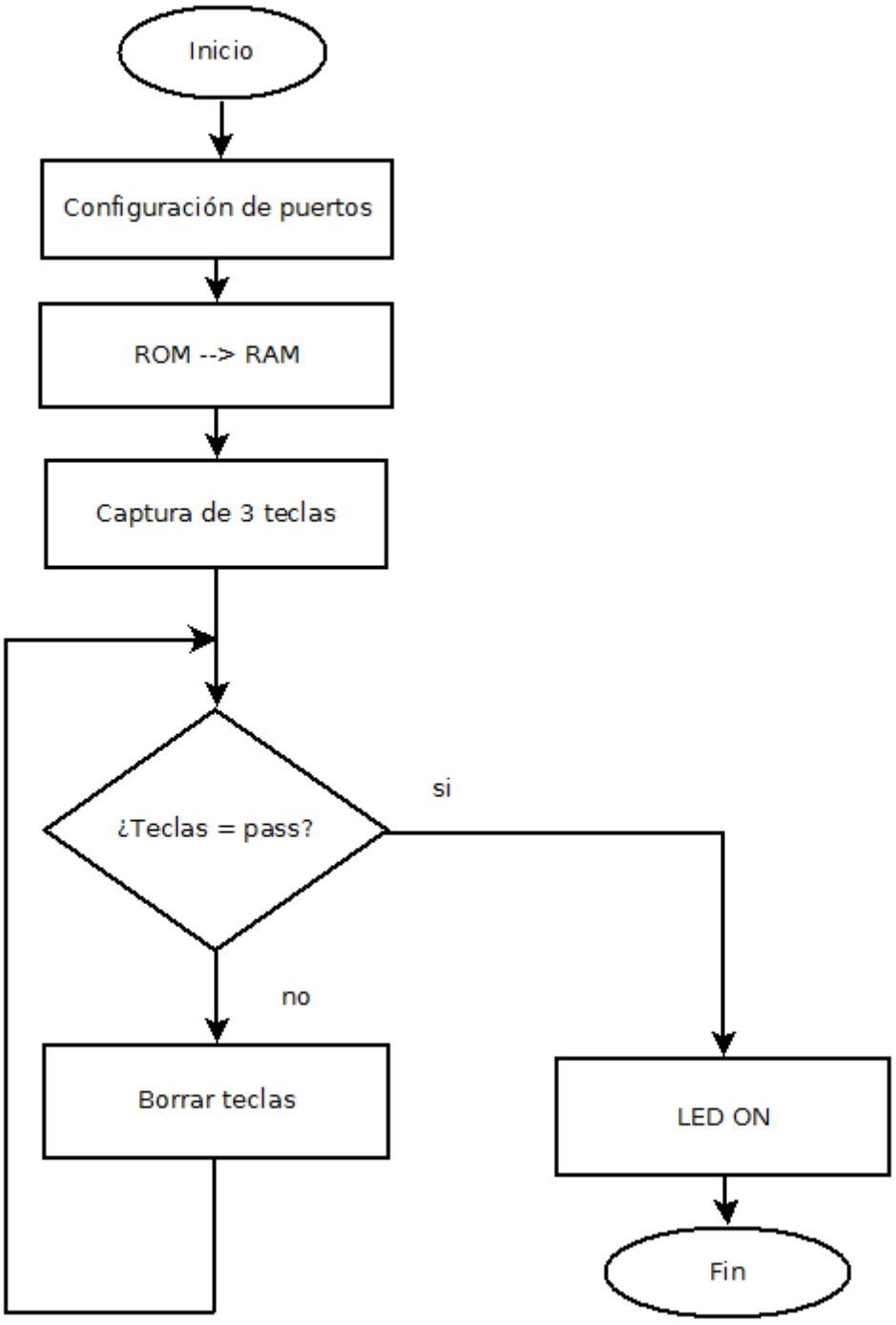


Figura 221. *Diagrama de flujo.*

1.1.5.4.9.4.3- Programación y simulación

Las constantes y variables usadas en el programa son las siguientes, (Figura 222):

```
;-----------------------------------------------------
;---    DECLARACIÓN DE CTES. Y VARIABLE               -
;-----------------------------------------------------
    Key_var     equ 0x73    ;Variables (6) del teclado
    PASS1   equ 0x30
    PASS2   equ 0x31
    PASS3   equ 0x32
    TECLA1  equ 0x40
    TECLA2  equ 0x41
    TECLA3  equ 0x42
```

Figura 222. *Registros y constantes.*

- *PASS1,PASS2 y PASS3* contiene la contraseña necesaria para activar el LED
- *TECLA1,TECLA2,TECLA3* son registros que almacenan las teclas pulsadas
- *Key_var*: registros usados por la librería del teclado

↳ *Código*

El programa principal se va a dividir en cuatro partes, configuración de puertos, carga de contraseña, obtención de las teclas y comprobación de password.

❖ *Configuración de puertos*

```
Inicio                  ; Etiqueta de comienzo de programa

;--- Configuración de los puertos ---
    clrw
    clrf    PORTA
    clrf    PORTB       ; Borra los latch de salida
    bsf STATUS,RP0
    bsf STATUS,RP1      ; Banco 3
    clrf    ANSEL       ; Entradas digitales
    clrf    ANSELH
    bcf STATUS,RP1      ; Banco 1
    movlw   0x0F        ; RB0-RB3 = In
    movwf   PORTB       ; RB4-RB7 = Out
    clrf    PORTA       ; PORT A (Out)
    bcf STATUS,RP0      ; Banco 0
```

Figura 223. *Configuración de puertos.*

De forma análoga a anteriores puertos se configuran RB0-RB3 como entrada digital y RB4-RB7 y el puerto A como salida digital.(Figura 223).

- *Carga de contraseña*

```
;--- Programa carga inicial ---

    bsf     STATUS,RP1  ; Banco 2
    clrf    EEADR       ; Posición de ROM donde se guarda el dato
    call    Lee_eeprom  ; Lee el dato de EEADR
    bsf     STATUS,RP1  ; Banco 2
    movf    EEDATA,W
    bcf     STATUS,RP1  ; Banco 0
    movwf   PASS1       ; ROM --> RAM
    bsf     STATUS,RP1  ; Banco 2
    incf    EEADR       ; Siguiente dígito
    call    Lee_eeprom
    bsf     STATUS,RP1  ; Banco 2
    movf    EEDATA,W
    bcf     STATUS,RP1  ; Banco 0
    movwf   PASS2       ; ROM --> RAM
    bsf     STATUS,RP1  ; Banco 2
    incf    EEADR       ; Siguiente dígito
    call    Lee_eeprom
    bsf     STATUS,RP1  ; Banco 2
    movf    EEDATA,W
    bcf     STATUS,RP1  ; Banco 0
    movwf   PASS3       ; ROM --> RAM
```

Figura 224. *Carga de contraseña.*

Para la lectura de la EEPROM (ver **practica 7**) se indica la dirección del dato a leer en el registro EEADR y se obtiene el dato en el registro EEDATA, ambos registros pertenecientes al banco 2. Por lo que introduciendo las tres direcciones de la contraseña y pasándolas a la memoria RAM obtendremos los tres dígitos del password. (Figura 224)

- *Captura de tecla*

```
; --- CAPTURA DE TECLAS
Tecla1  call    Key_Scan    ; capturo tecla
        movf    Tecla,W
        movwf   TECLA1
        movlw   0x80        ; compruebo si se ha pulsado alguna
        subwf   TECLA1,W
        btfsc   STATUS,Z    ; ¿Se pulso tecla?
        goto    Tecla1      ; NO --> Esperar captura
Tecla2  call    Key_Scan    ; capturo tecla 2
        movf    Tecla,W
        movwf   TECLA2
        movlw   0x80        ; compruebo si se ha pulsado alguna
        subwf   TECLA2,W
        btfsc   STATUS,Z    ; ¿Se pulso tecla?
        goto    Tecla2      ; NO --> Esperar captura
Tecla3  call    Key_Scan    ; capturo tecla
        movf    Tecla,W
        movwf   TECLA3
        movlw   0x80        ; compruebo si se ha pulsado alguna
        subwf   TECLA3,W
        btfsc   STATUS,Z    ; ¿Se pulso tecla?
        goto    Tecla3      ; NO --> Esperar captura
```

Figura 225. *Capturas de teclas.*

La librería TECLADO.INC devuelve en el registro Tecla, la tecla que se ha pulsado y si no fue pulsado ninguna tecla retorna con el valor 0x80. Por lo que para cada tecla basta con comparar el valor que retorna de la función, si es 0x80 tiene que volver a captura, si tiene otro valor, pasa a capturar la siguiente tecla, hasta que se tengan los tres dígitos que pasará a la verificación de contraseña.(Figura 225).

- *Verificación de contraseña*

```
; --- COMPROBACIÓN DE PASS
        movf    TECLA1,W
        subwf   PASS1,W     ; Primer dígito
        btfss   STATUS,Z    ; ¿Digito ok?
        goto    Tecla1      ; No --> Nuevas teclas
        movf    TECLA2,W    ; Si --> siguiente tecla
        subwf   PASS2,W     ; Primer dígito
        btfss   STATUS,Z    ; ¿Digito ok?
        goto    Tecla1      ; No --> Nuevas teclas
        movf    TECLA3,W    ; Si --> siguiente tecla
        subwf   PASS3,W     ; Primer dígito
        btfss   STATUS,Z    ; ¿Digito ok?
        goto    Tecla1      ; No --> Nuevas teclas
        bsf     PORTA,0     ; Si --> Activo led
```

Figura 226. *Comprobación de password.*

En esta parte del programa tiene que comprobar que la contraseña sea correcta, para ello va comprobando dígito a dígito si coincide con el password. En el momento que un dígito no coincide vuelve a buscar otras tres nuevas teclas.(Figura 226)

Si las tres teclas pulsadas coincide, activará el led y finalizará el programa.

- *Simulación*

Para la simulación del teclado hay que tener en cuenta que la librería está programada de tal forma que la tecla pulsada es la que se encuentra a nivel bajo. Se cargará una contraseña inicial no programada haciendo click en la pantalla de la memoria, si se quiere programar hay que realizar la subrutina de escritura en la eeprom.

Inicialmente se carga la contraseña 'A0B' que corresponde los valores según la tabla el 0x7Eh,0x7D,0x7B, (Figura 227)

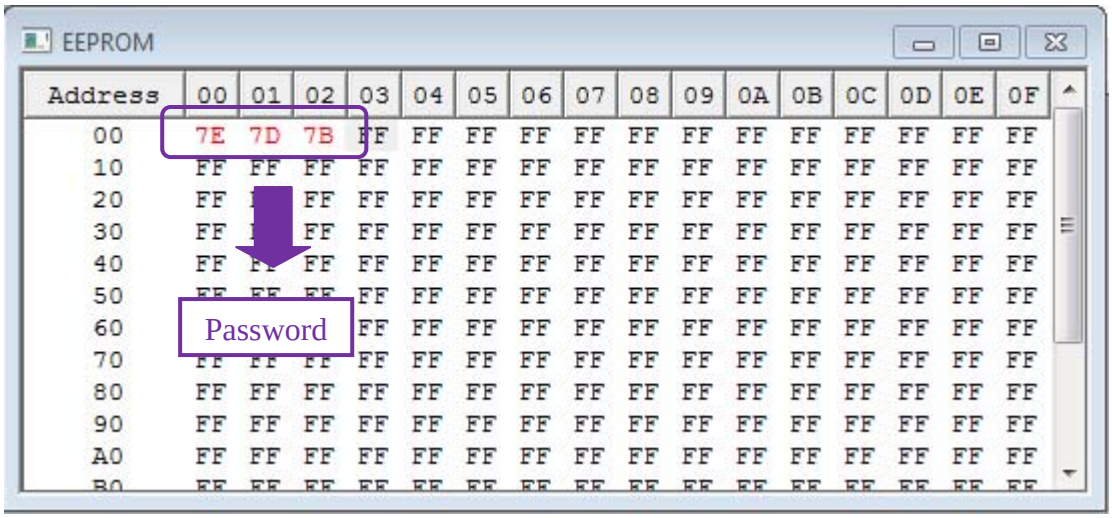


Figura 227. *EEPROM.*

Después de la configuración de puertos los registros más importantes se observan en la figura 228.

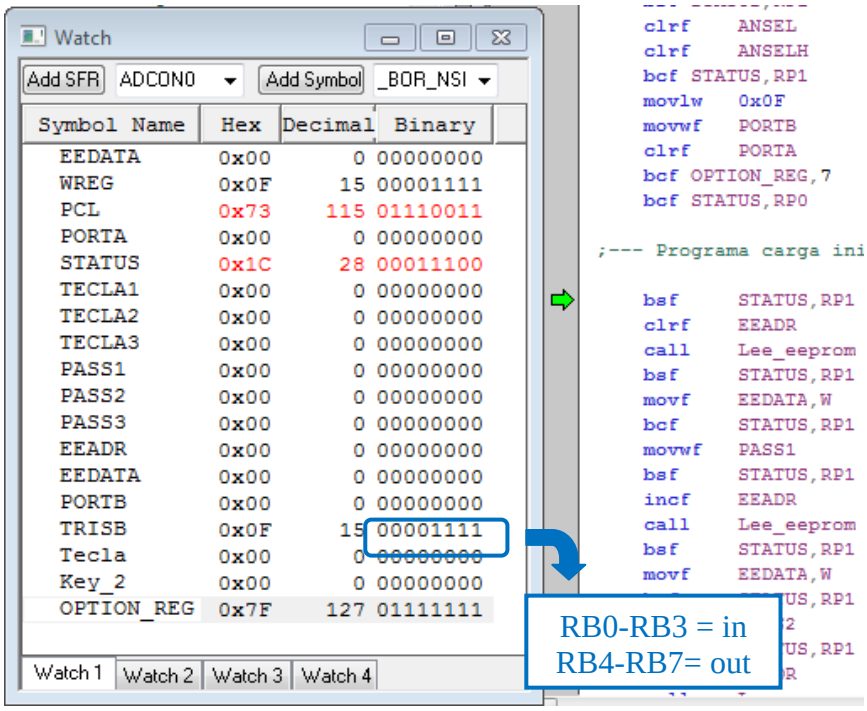


Figura 228. *Configuración de puertos.*

Una vez cargadas las contraseñas, y colocando las 4 entradas de las teclas (RB0-RB3) activas a nivel alto, es decir, sin pulsar una tecla, (Figura 229):

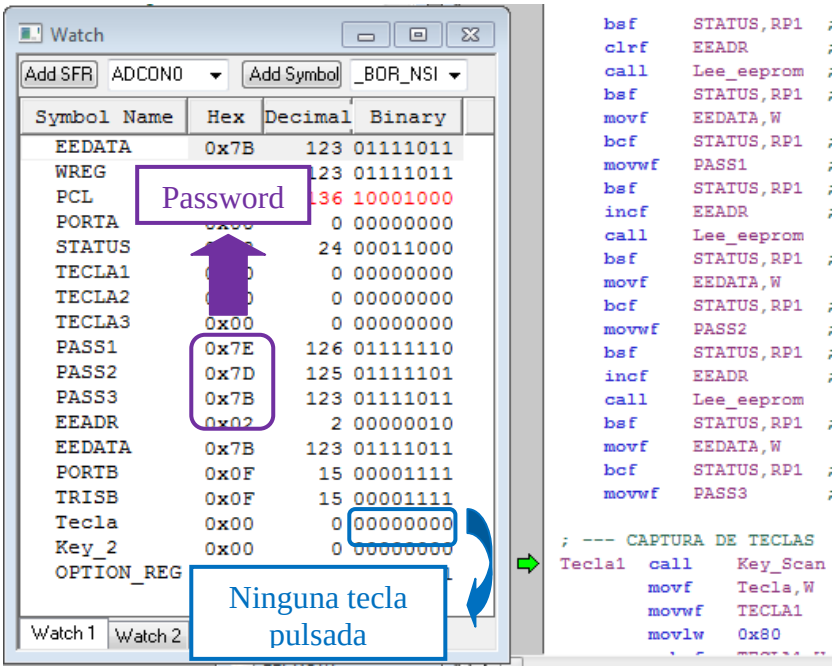


Figura 229. *Carga del password.*

Si no es pulsada ninguna tecla la librería TECLADO.INC devuelve el valor 0x80h en el registro Tecla, ese valor será el empleado para comparar y realizar un bucle mientras no varíe el registro tecla de 0x80h, (Figura 230):

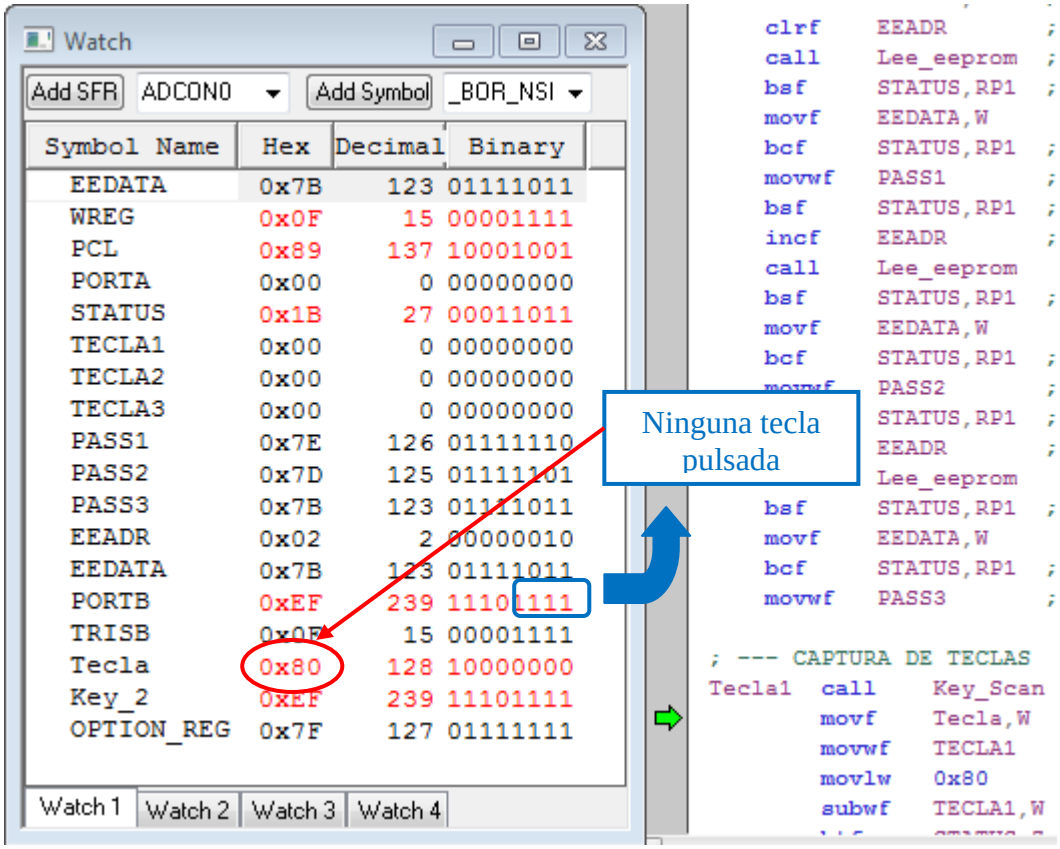


Figura 230. *Teclado sin pulsar.*

Cuando se pulsa una tecla, el valor del registro Tecla cambia, y deja de valer 0x80h, por lo que saltará el bucle y comenzará a pedir la nueva tecla.

En nuestro caso se pulsa el valor de la tecla A, para la primera tecla capturada (Figura 231):

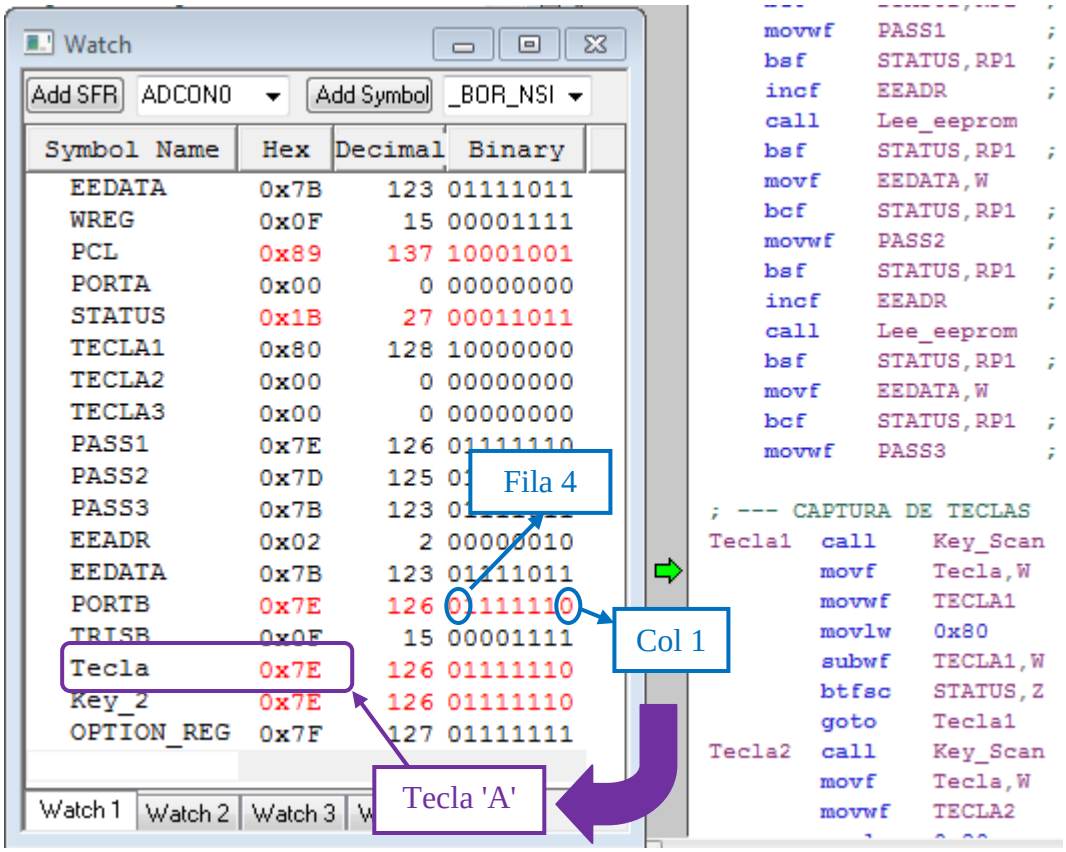


Figura 231. *Primera tecla pulsada.*

Para la segunda tecla se pulsa la tecla 0, esta tecla corresponde a la fila 4 columna dos del teclado numérico.(Figura 220).

Después de entrar a la librería TECLADO.INC, el retorno de subrutina saca el valor codificado de la tecla, en el registro Tecla con el valor correspondiente según la Tabla 23.

Como es un valor distinto de 0x80h, es decir que sí se presionó alguna tecla, el programa continuará saliéndose del bucle, (Figura 232).

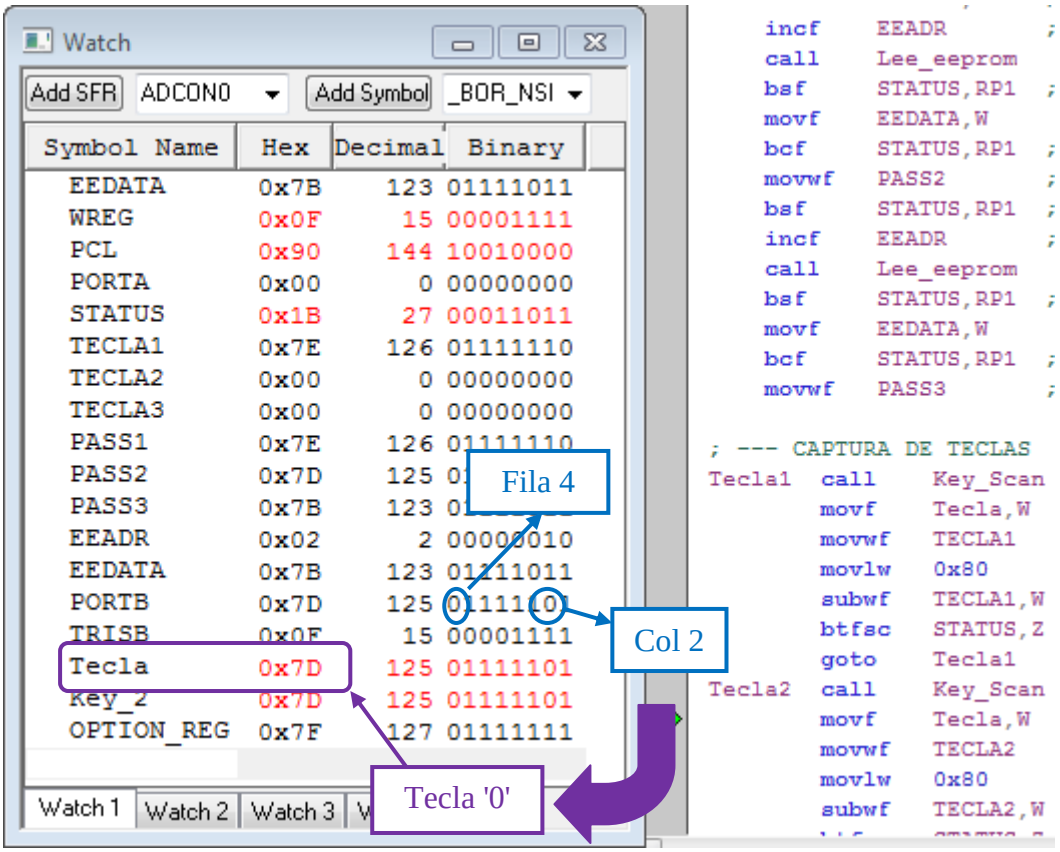


Figura 232. *Segunda tecla pulsada.*

Para la tercera tecla se pulsa la tecla B, esta tecla corresponde a la fila cuatro, columna tres del teclado numérico.(Figura 220).

Después de entrar a la librería TECLADO.INC, el retorno de subrutina saca el valor codificado de la tecla, en el registro Tecla con el valor correspondiente según la Tabla 23.

Como es un valor distinto de 0x80h, es decir que sí se presionó alguna tecla, el programa continuará saliéndose del bucle, (Figura 233).

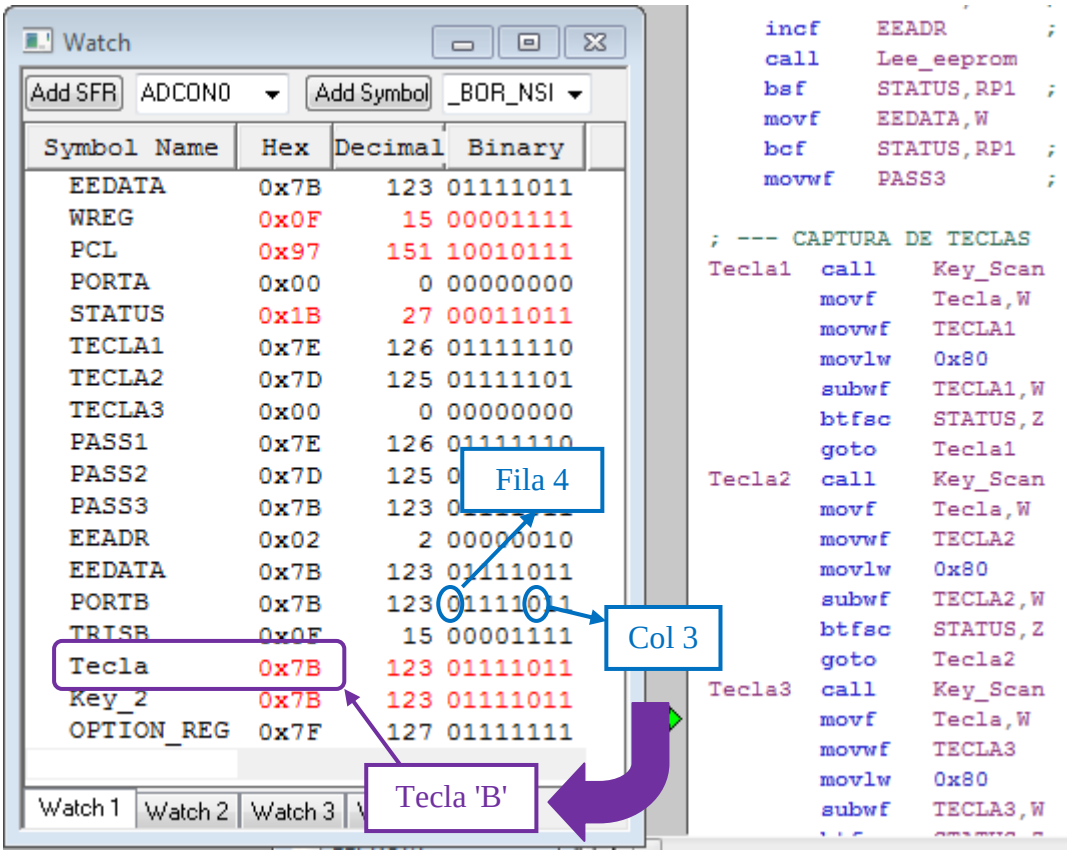


Figura 233. *Tercera tecla pulsada.*

Por último verifica la contraseña, para ello comparará las tres teclas pulsadas mediante el teclado matricial con la contraseña cargada de la EEPROM.

Si alguno de los dígitos no coincide en el orden indicado volverá al punto de partida para pedir tres nuevos dígitos y comprobar si coinciden. Esta acción se repetirá hasta el momento que las teclas pulsadas coincidan con el password que se tenía almacenado en la memoria EEPROM.

En este caso al ser la contraseña correcta, pasará a la última parte del prgroama donde encenderá la luz del puerto A, (Figura 234):

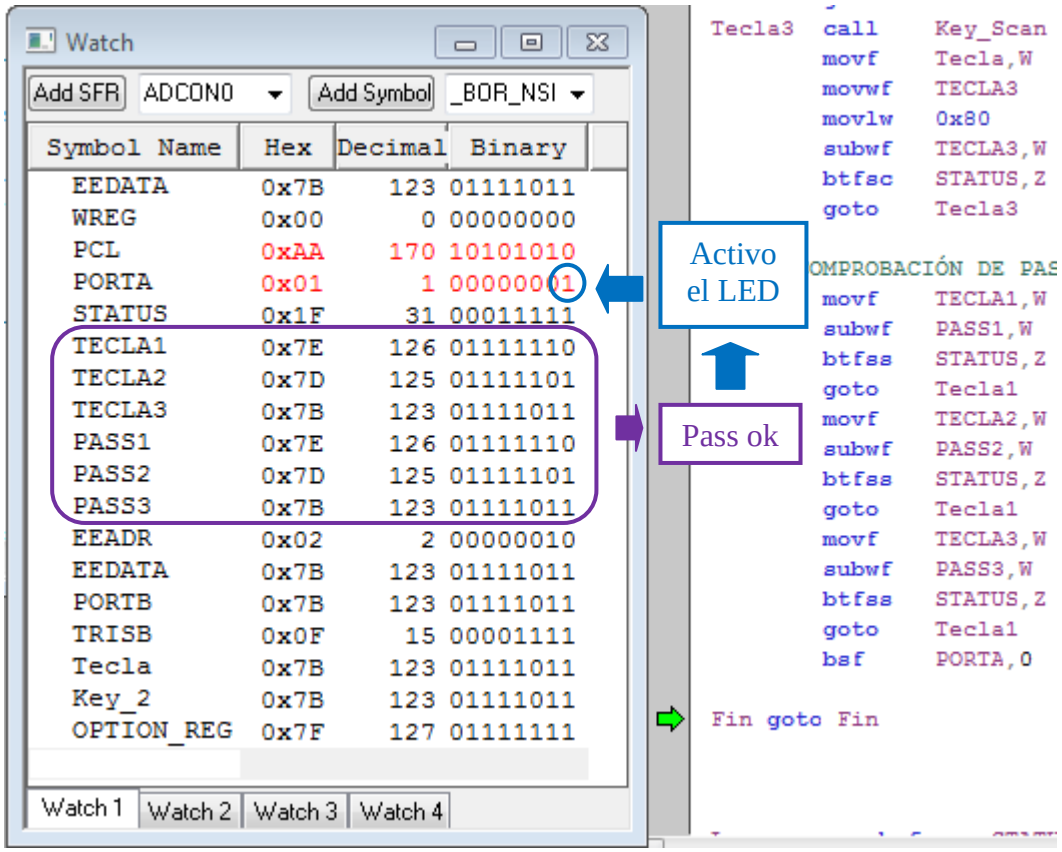


Figura 234. *Contraseña correcta.*

1.1.5.4.9.4.4.- Montaje

El puerto B se conecta con RB7-RB4 con las líneas F3-F0 (filas) y RB3-RB0 con C3-C0 (columnas).

Además el puerto que actúa como salida conecta RA0 al led S0 que se activará cuando la contraseña sea la correcta .(Figuras 235 y 236)

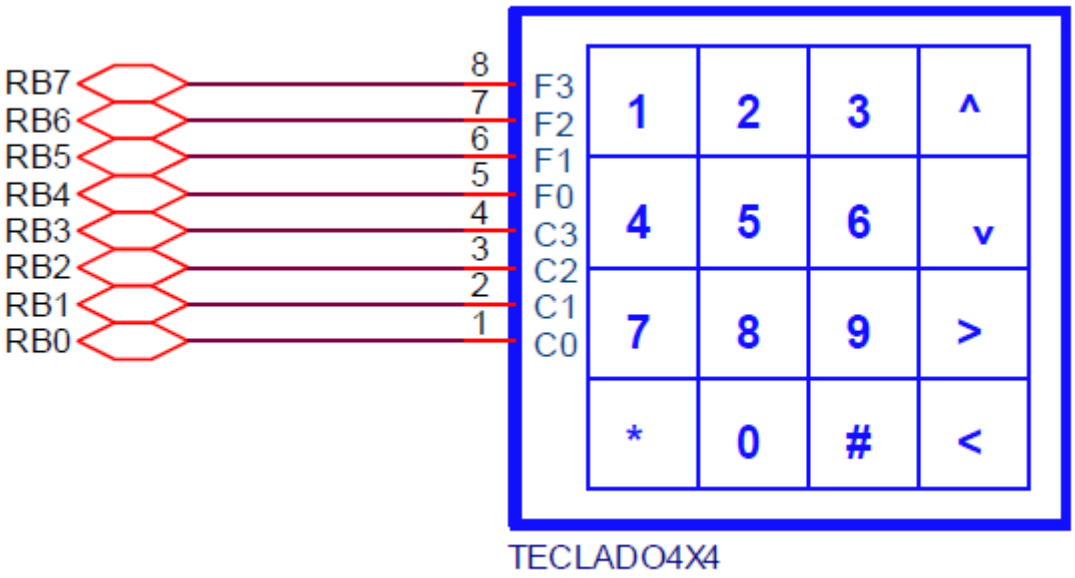


Figura 235. *Esquema de conexiones del teclado matricial.* [5]

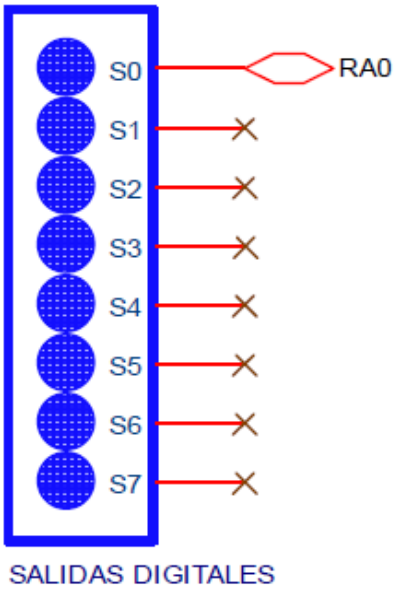


Figura 236. *Esquema de conexiones de los leds.*

1.1.5.4.9.5.- Programas propuestos

Implementar el programa del teclado, de tal forma que la detección de teclas se realice por una interrupción por el puerto B.

Realizar un programa que visualice en un display la tecla pulsada por el teclado.

1.1.6.- Distribución del tiempo empleado

Durante la realización del tiempo empleado, se han empleado 18 semanas con una media semanal de 25 horas, en total las horas empleadas en el TFG son las siguientes (16):

$$19\ semanas \cdot 25 \frac{horas}{semana} = 475\ horas\ de\ realización \tag{16}$$

Atendiendo a los criterios temporales estipulados para la ECTS (***European credit transfer system***) se cumplen las expectativas para un trabajo de 18 créditos.

La distribución de las horas empleadas en los distintos apartados se observa en la tabla 24:

Tabla 24. *Distribución de tiempos.*

Apartado	Horas dedicadas	Total
Prácticas	1,5 semanas/prácticas	337,5 horas
Tutorial MPLAB	1,5 semanas	37.5 horas
PIC 16F886	1 semana	25 horas
Realización de la memoria	3 semanas	75 horas

1.2.- ANEJOS A LA MEMORIA

1.2.1.- Anejo de datos de partida

A la hora de realizar el trabajo fin de grado, se busca generar una programación de un curso de microcontroladores, en distintas enseñanzas de grado superior de formación profesional relacionadas con la automática o la electrónica o bien en enseñanzas universitarias relacionadas.

Se parte de un curso de 150 horas o 6 créditos, de los cuales 60 horas son presenciales y 90 horas son de trabajo personal, la distribución final se puede ver en el apartado 7 de *justificación de la solución adoptada*.

1.2.2.- Anejo de cálculos

1.2.2.1.- Cálculo del valor máximo

1.2.2.1.1.- Registros sin signo

Los registros que utiliza el PIC16F886, son registros de 8 bits, tanto en la memoria RAM como en la EEPROM, por esto el máximo valor que se puede obtener de un registro sin signo se calcula de la siguiente forma (18):

$$max = 2^n - 1 \tag{17}$$

$$max = 2^8 - 1 = 255 \tag{18}$$

Siendo n el número de bits del registro.

1.2.2.1.2.- Registros con signo

Para obtener el valor máximo en un registro que usa el bit más significativo como signo del número, en este caso el número de bits disponibles serían 7, por lo tanto el valor máximo se calcula en la ecuación (19):

$$max = 2^7 - 1 = 127 \quad (19)$$

1.2.2.1.3.- Varios registros

En operaciones como la multiplicación, a veces, es necesario el uso de varios registros a la hora de almacenar un dato. El valor máximo se obtendría de la siguiente expresión, (2):

$$max = (2^n)^m - 1 \quad (20)$$

Siendo m el número de registros empleados para almacenar el dato. Generalmente se suelen agrupar dos registros, en este caso (5):

$$max = (2^8)^2 - 1 = 2^{16} - 1 = 65535 \quad (21)$$

1.2.2.- Cálculo del temporizador

Para calcular un temporizador es necesario tener en cuenta la frecuencia del temporizador dicha frecuencia se calcula de la siguiente forma, (23) y el tiempo de ejecución de cada ciclo (24)

$$F_{TMR0} = \frac{F_{INT}}{4} \quad (22)$$

$$F_{TEMP} = \frac{F_{TMR0}}{escalado} \quad (23)$$

$$T_{TEMP} = \frac{escalado}{F_{TMR0}} \quad (24)$$

Siendo:

- ✓ F_{INT}: frecuencia interna
- ✓ F_{TMR0}: frecuencia del tmr0 sin escalar
- ✓ F_{TEMP}: frecuencia del tmr0 escalada
- ✓ escalado: factor escala seleccionado por el registro OPTION

Cada Ttemp realizará una cuenta, para calcular el tiempo que tarda en desbordarse(25):

$$t_{desb} = (256 - VI) \cdot T_{TEMP} \tag{25}$$

Con:

- ✓ t_{desb}: tiempo que tarda en desbordarse el temporizador.
- ✓ VI: Valor inicial del temporizador

Despejando el valor inicial obtenemos (27):

$$VI = 256 - \frac{t_{desb}}{T_{TEMP}} = 256 - t_{desb} \cdot F_{TEMP} \tag{26}$$

$$VI = 256 - t_{desb} \cdot \frac{F_{INT}}{4 \cdot escalado} \tag{27}$$

En el programa se busca un tiempo de 10ms, lo que implicaría que se desbordara 25 veces. El valor inicial que se debe dar al temporizador será (28)

$$VI = 256 - \frac{0.01s \cdot 4 \cdot 10^6 s^{-1}}{4 \cdot 256} = 216.93 \rightarrow 217 \tag{28}$$

El error cometido de aproximar a una cuenta inicial de 217 se calcula de la siguiente forma (30)

$$t_{desb} = (256 - 217) \cdot \frac{256}{10^6} = 9.984 \cdot 10^{-3} \tag{29}$$

$$\varepsilon(\%) = \left(1 - \frac{9.984}{10}\right) \cdot 100 = 0.16\% \tag{30}$$

Siendo ε el error cometido en tanto por cierto, de tomar como valor inicial de carga del temporizador como 217.

1.2.3.- Anejo de programas

Se muestra a continuación los programas sugeridos para la realización de las distintas prácticas con el PIC16F886:

1.2.3.1.- Práctica 1

1.2.3.1.1.- Ejercicio

```
;*******************************************************
;*                      PRACTICA 1                     *
;*          OPERACIONES ARITMÉTICO LÓGICAS             *
;*                   EJERCICIO BÁSICO                  *
;*******************************************************
;* Autor: Diego Bugallo Garrido                        *
;* Controlador: PIC16F886                              *
;* Última revisión: 4 - Marzo - 2013                   *
;*******************************************************

      list p=16F886                ; Tipo de PIC
      #include "P16F886.INC"  ; Identifica los ppales registros

;-------------------------------------------------------
;---  DECLARACIÓN DE CTES. Y VARIABLE                  -
;-------------------------------------------------------

      #define cte1 .16             ; Número decimal
      #define cte2 0x11            ; Número hexadecimal
      #define cte3 b'00000101'     ; Número binario

      VAR1 equ    0x10        ; Posición de memoria inicial 0x10h
      VAR2 equ    VAR1+1
      VAR3 equ    VAR1+2
      RES1 equ    VAR1+3
      RES2 equ    VAR1+4
      RES3 equ    VAR1+5
      RESULTADO equ     VAR1+6      ; Posición de memoria final 0x16h

;-------------------------------------------------------
;---  INICIO DEL PROGRAMA PRINCIPAL                    -
;-------------------------------------------------------
```

```
      org 0x00    ; Posición de memoria de prog. que comienza
      goto Inicio ; Salto a la etiqueta Inicio
      org 0x05    ; Posición de memoria de la etiqueta Inicio

Inicio            ; Etiqueta de comienzo de programa

;-----------------------------------------------------------
;---                    VALORES INICIALES                -
;-----------------------------------------------------------
      clrw              ; W = 0
      movlw .5
      movwf VAR1        ; VAR1 = 5
      movlw 0x12
      movwf VAR2        ; VAR2 = 12h
      movlw b'00000011'
      movwf VAR3        ; VAR3 = b'00000011'

;-----------------------------------------------------------
;---                    OPERACIONES ARITMÉTICAS          -
;-----------------------------------------------------------

; RES1 = VAR1 + cte1 + 5
      movlw       .5
      movwf RES1        ; RES1 = 5
      movlw cte1
      addwf RES1,RES1   ; RES1 = 5 + cte1
      movf  VAR1,W
      addwf RES1,RES1   ; RES1 = 5 + cte1 + VAR1

; RES2 = VAR2 + cte2 - VAR3
      movf  VAR2,W
      movwf RES2        ; RES2 = VAR2
      movlw       cte1
      addwf RES2        ; RES2 = VAR2 + cte1
      movf  VAR3,W
      subwf RES2        ; RES2 = VAR2 + cte1 - VAR3

; RESULTADO = RES1 + RES2
      movf  RES1,W
      movwf RESULTADO   ; RESULTADO = RES1
      movf  RES2,W
      addwf RESULTADO   ; RESULTADO = RES1 + RES2
```

```
; Incrementar RESULTADO y decrementar RES1
      incf  RESULTADO   ; RESULTADO = RESULTADO + 1
      decf  RES1        ; RES1 = RES1 - 1

;-------------------------------------------------------
;---                   OPERACIONES LÓGICAS             -
;-------------------------------------------------------

; Puesta a cero de registros
      clrw
      clrf RES1
      clrf RES3

; RES1 = VAR3 & cte3
      movf  VAR3,W
      movwf RES1        ; RES1= VAR3
      movlw cte3
      andwf RES1        ; RES1 = VAR3 & cte3

; RES3 = RES1 OR '00010110'
      movf  RES1,W
      movwf RES3        ; RES3 = RES1
      movlw b'00010110'
      iorwf RES3        ; RES3 = RES1 OR '00010110'

; RES3 --> BIT1 = 0 Y BIT 3 =1
      bcf         RES3,1            ; Puesta a cero del bit1
      bsf         RES3,3            ; puesta a uno del bit3

; Operación a '10000000'      que ponga a uno Z
      movlw       b'10000000'
      movwf       RES2
      movlw       b'000011110'
      andwf RES2,W            ; realizo un and y almaceno en W

Fin   goto Inicio

End
```

1.2.3.1.2.- Ejercicio de ampliación

```
;*******************************************************
;*          PRACTICA 1                                 *
```

```
;*    OPERACIONES ARITMÉTICO LÓGICAS                    *
;*          EJERCICIO AMPLIACIÓN                        *
;*****************************************************
;* Autor: Diego Bugallo Garrido                        *
;* Controlador: PIC16F886                              *
;* Ultima revisión: 10 - Marzo - 2013                  *
;*****************************************************


      list p=16F886                ; Tipo de PIC
      #include "P16F886.INC"  ; identifica los ppales registros

;-----------------------------------------------------
;---  DECLARACIÓN DE CTES. Y VARIABLE                 -
;-----------------------------------------------------

      #define cte_a .3
      #define cte_b .2
      #define cte_c .4
      #define cte_d .2

      RES1 equ    0x10        ; Posición de memoria inicial 0x10h
      RES2 equ    0x11
      RES3 equ    0x12
;-----------------------------------------------------
;---  INICIO DEL PROGRAMA PRINCIPAL                   -
;-----------------------------------------------------

      org 0x00    ; Posición de memoria de prog. que comienza
      goto Inicio ; Salto a la etiqueta Inicio


      org 0x05    ; Posición de memoria de la etiqueta Inicio

Inicio                  ; Etiqueta de comienzo de programa

;-----------------------------------------------------
;---                    VALORES INICIALES             -
;-----------------------------------------------------
      clrw                    ; W = 0
      clrf RES1               ; RES1 = 0
      clrf RES2               ; RES2 = 0

;-----------------------------------------------------
```

```
;---                    OPERACIONES ARITMÉTICAS          -
;-----------------------------------------------------------

; RES1=2(a-1)+2(b+1)-c-d

      movlw cte_a-1     ;     a=3
      movwf RES1  ;     RES1 = a-1
      rlf   RES1  ;     RES1 = 2*(a-1)
      movlw cte_b+1 ;   b=2
      addwf RES1  ;     RES1 = 2*(a-1) + b+1
      addwf RES1  ;     RES1 = 2*(a-1) + 2*(b+1)
      movlw cte_c ;     c=4
      subwf RES1  ;     RES1 = 2*(a-1) + 2*(b+1)-c
      movlw cte_d ;     d=2
      subwf RES1  ;     RES1 = 2*(a-1) + 2*(b+1)-c-d


;-----------------------------------------------------------
;---  OPERACIONES LÓGICAS                                  -
;-----------------------------------------------------------

;  b'11011011' poner a cero los bits 0 y 4

      movlw b'11011011'
      movwf RES2
      movlw b'11101110' ; Máscara AND
      andwf RES2

;     b'11011011' intercambio de estado de b0-b3
      movlw b'00001111' ; Máscara XOR
      xorwf RES2

; Intercambio de parte alta y baja
      swapf RES2

;-----------------------------------------------------------
;---  FLAGS                                                -
;-----------------------------------------------------------


; Operación C=1 y DC =0

      movlw 0x80        ; b'10000000'
      movwf RES1
```

```
      addwf RES1

; Operación C=0 y DC = 1

      movlw 0x0F        ; b'00001111'
      movwf RES1
      addwf RES1

; Usando decremento activar Z

      movlw 0x01
      movwf RES1
      decf  RES1

Fin   goto Inicio

End
```

1.2.3.2.- Práctica 2

1.2.3.2.1.- Ejercicio

```
;*********************************************************
;*          PRACTICA 2                                   *
;*    BUCLES, SUBRUTINAS E INSTRUCIONES DE CONTRO        *
;*          EJERCICIO BÁSICO                             *
;*********************************************************
;* Autor: Diego Bugallo Garrido                          *
;* Controlador: PIC16F886                                *
;* Versión: 1.0                                          *
;* Ultima revisión: 18 - Marzo - 2013                    *
;*********************************************************
;* Descripción: Calcular el valor máximo de 3 números    *
;* Subrutina: Valor máximo de dos número almacenados en  *
;* DATOA y DATOB máximo en W ----> W=max(DATOA,DATOB)    *
;*********************************************************


      list p=16F886           ; Tipo de PIC
      #include "P16F886.INC"  ; identifica los ppales registros

;-------------------------------------------------------
;---  DECLARACIÓN DE CTES. Y VARIABLE                   -
```

```
;-------------------------------------------------------

      #define a1 .10               ; Datos a comparar
      #define a2 .5
      #define a3 .25

      MAX   equ 0x13          ; Registro del valor máximo
      DATOA equ   0x14  ; Variables auxiliares para la subrutina
      DATOB equ 0x15

;-------------------------------------------------------
;---  INICIO DEL PROGRAMA PRINCIPAL                    -
;-------------------------------------------------------

      org 0x00    ; Posición de memoria de prog. que comienza
      goto Inicio ; Salto a la etiqueta Inicio


      org 0x05    ; Posición de memoria de la etiqueta Inicio

Inicio            ; Etiqueta de comienzo de programa

;-------------------------------------------------------
;---                    PROGRAMA PRINCIPAL             -
;-------------------------------------------------------

      clrw              ; Reset del registro de trabajo

      movlw a1          ; Parámetros a pasar al bucle
      movwf DATOA
      movlw a2
      movwf DATOB

      call Máximo       ; W = MAXIMO (a1,a2)

      movwf DATOA       ; DATOA = MAXIMO (a1,a2)
      movlw a3
      movwf DATOB       ; DATOB = a3

      call Máximo       ; W = MAXIMO(MAXIMO(a1,a2),a3))
      movwf MAX         ; Almaceno el valor máximo

Fin   goto Inicio
```

```
;-------------------------------------------------------
;---        SUBRUTINA COMPARA   W=MAX(DATOA,DATOB)      -
;-------------------------------------------------------

Máximo

      movf  DATOB,W           ; W = DATOB
      subwf DATOA             ; DATOA' = DATOA - DATOB
      btfss STATUS,C          ; Si la resta es negativa salta

      return                  ; b>a W=DATOB
      addwf DATOA,W           ; a>=b W=DATOB + DATOA' = DATOA
      return

End
```

1.2.3.2.2.- Ejercicio ampliación

```
;*******************************************************
;*          PRACTICA 2                                 *
;*    BUCLES, SUBRUTINAS E INSTRUCIONES DE CONTRO      *
;*          EJERCICIO AVANZADO                         *
;*******************************************************
;* Autor: Diego Bugallo Garrido                        *
;* Controlador: PIC16F886                              *
;* Ultima revisión: 20 - Marzo - 2013                  *
;*******************************************************
;* Descripción: Multiplicación de dos números          *
;* Máximo:  (MAX,MIN)=Ordena(DATOA,DATOB)              *
;* Multiplicar: RESH & RESH   = Multiplica(DATOA,DATOB)  *
;* Nota: Ordena los números reduciendo las iteraciones  *
;*******************************************************


      list p=16F886           ; Tipo de PIC
      #include "P16F886.INC"  ; identifica los ppales registros

;-------------------------------------------------------
;---  DECLARACIÓN DE CTES. Y VARIABLE                  -
;-------------------------------------------------------

      #define a1 .200              ; Datos a multiplicar
      #define a2 .3
```

```
      MAX   equ 0x12          ; Multiplicando
      MIN   equ 0x13          ; Multiplicador
      DATOA equ   0x14        ; Variables de la subrutina
      DATOB equ 0x15
      RESL equ 0x16           ; Parte baja del resultado
      RESH equ 0x17           ; Parte alta del resultado

;-----------------------------------------------------
;---  INICIO DEL PROGRAMA PRINCIPAL                   -
;-----------------------------------------------------

      org 0x00    ; Posición de memoria de prog. que comienza
      goto Inicio ; Salto a la etiqueta Inicio


      org 0x05    ; Posición de memoria de la etiqueta Inicio

Inicio            ; Etiqueta de comienzo de programa

;-----------------------------------------------------
;---              PROGRAMA PRINCIPAL                 -
;-----------------------------------------------------

      clrw              ; Inicialización de registros

      movlw a1          ; Parámetros a multiplicar
      movwf DATOA
      movlw a2
      movwf DATOB

      call Multiplicar  ; Almacena el resultado en RESL y RESH

Fin   goto Inicio

;-----------------------------------------------------
;--- SUBRUTINA MULTIPLICAR    RESH & RESL =DATOA*DATOB  -
;-----------------------------------------------------

Multiplicar
                  clrf  RESH  ; Puesta a cero de los resultados
                  clrf  RESL
                  call Ordena ; (MAX,MIN) = ORDENA(DATOA,DATOB)
Multiplica
```

```
                movf  MAX,W       ; MAX multiplicando
                addwf RESL        ; RESL = RESL + MAX
                btfsc STATUS,C    ; Si no hay desbordamiento salta
                call  Desbordar   ; SI C=1
                decfsz      MIN   ; Decremento el multiplicador
                goto  Multiplica  ; Si MIN > 0
                return

;-----------------------------------------------------
;--- SUBRUTINA DESBORDAR    RESH = RESH + 1            -
;-----------------------------------------------------

Desbordar   incf  RESH        ; RESH = RESH + 1
                return


;-----------------------------------------------------
;--- SUBRUTINA ORDENA    (MAX,MIN)= ORDENA(DATOA,DATOB) -
;-----------------------------------------------------
Ordena
      movf  DATOA,W
      movwf MIN
      movf  DATOB,W           ; W = DATOB
      movwf MAX
      subwf DATOA             ; DATOA' = DATOA - DATOB
      btfss STATUS,C          ; Si la resta es negativa salta

      return                  ; b>a W=DATOB
      addwf DATOA,W           ; a>=b W=DATOB + DATOA' = DATOA
      movwf MAX
      movf  DATOB,W
      movwf MIN
      return

End
```

1.2.3.3.- Práctica 3

1.2.3.3.1.- Ejercicio

```
;*******************************************************
;*          PRACTICA 3                                 *
;*    DIRECCIONAMIENTO Y TABLAS                        *
```

```
;*          EJERCICIO BÁSICO                                  *
;*************************************************************
;*  Autor: Diego Bugallo Garrido                              *
;*  Controlador: PIC16F886                                    *
;*    Ultima revisión: 2 - Abril - 2013                       *
;*************************************************************
;* Descripción:  A)Programar un display de 7 segmentos        *
;* B) escribir en RAM 20-19-18-...-11                        *
;*************************************************************


      list p=16F886          ; Tipo de PIC
      #include "P16F886.INC"  ; identifica los ppales registros

;-------------------------------------------------------------
;---  DECLARACIÓN DE CTES. Y VARIABLE                        -
;-------------------------------------------------------------

;----------- DISPLAY 7 SEGMENTOS ----------------------

      #define     valor1      .5    ; Valores a codificar
      #define     valor2      .0
      #define     valor3      .9
      #define     máximo      .9    ; Valor máximo de la tabla

;--- Display   ---

      #define cero      b'00111111'
      #define uno       b'00000110'
      #define dos       b'01011011'
      #define tres      b'01001111'
      #define cuatro    b'01100110'
      #define cinco     b'01101101'
      #define seis      b'01111101'
      #define     siete b'00000111'
      #define ocho      b'01111111'
      #define nueve     b'01101111'
      #define fallo     b'01111001'       ; Letra E

      DIGITO equ 0x20   ; se almacena el número a convertir
      DIGITO_DISPLAY    equ 0x21    ; Valor del display
      TEMP  equ 0x22                ; Variable auxiliar
```

```
;----------- GRABACIÓN MEMORIA RAM ----------------------

      #define dato      .20   ; Valor inicial a almacenar en RAM
      #define posiciones .10  ; posiciones a ocupar en la RAM
      #define inicial 0x30    ; Primera dirección a almacenar



      CONTADOR equ 0x23 ; cuenta las posiciones de la RAM
      NUMERO equ 0x24   ; variable que almacena el numero a guardar




;-------------------------------------------------------
;---  INICIO DEL PROGRAMA PRINCIPAL                    -
;-------------------------------------------------------

      org 0x00    ; Posición de memoria de prog. que comienza
      goto Inicio ; Salto a la etiqueta Inicio


      org 0x05    ; Posición de memoria de prog. de la tabla

;-------------------------------------------------------
;---              TABLA DEL DISPLAY                    -
;-------------------------------------------------------

Tabla
      movlw máximo
      movwf TEMP
      movf  DIGITO,W    ; Copio el contenido del puerto a W
      subwf TEMP
      btfss STATUS,C    ; Compruebo si la entrada es 0-9
      movlw maximo+1    ; Si no es el número 0-9 --> W = .10
      addwf PCL
      retlw cero      ; LED's activados para el cero en un display
      retlw uno       ; LED's activados para el uno en un display
      retlw dos       ; LED's activados para el dos en un display
      retlw tres      ; LED's activados para el tres en un display
      retlw cuatro    ; LED's activados para el cuatro en un display
      retlw cinco     ; LED's activados para el cinco en un display
      retlw seis      ; LED's activados para el seis en un display
      retlw siete     ; LED's activados para el siete en un display
```

```
      retlw ocho      ; LED's activados para el ocho en un display
      retlw nueve     ; LED's activados para el nueve en un display
      retlw fallo     ; LED's activados para la E en un display



;-----------------------------------------------------
;---                    PROGRAMA PRINCIPAL            -
;-----------------------------------------------------


Inicio      ; Etiqueta de comienzo de programa

      ;---------- PARTE A CODIFICACIÓN DISPLAY --------------

      clrw
      movlw valor1      ; Paso a DIGITO el valor binario a convertir
      movwf DIGITO
      call  Tabla       ; Tabla saca a W la conversión
      movwf DIGITO_DISPLAY    ; DIGITO_DISPLAY  valor del display
      movlw valor2
      movwf DIGITO
      call  Tabla
      movwf DIGITO_DISPLAY
      movlw valor3
      movwf DIGITO
      call  Tabla
      movwf DIGITO_DISPLAY

      ;---------- PARTE B ESCRITURA EN RAM --------------

      movlw posiciones
      movwf CONTADOR    ; Cargo en CONTADOR el numero de posiciones
      movlw inicial-1
      movwf FSR         ; puntero = posición anterior a la inicial
      movlw dato        ; Cargo en NUMERO el dato inicial a guardar
      movwf NUMERO

Ram   incf FSR          ; Apunta a la siguiente posición
      movwf INDF        ; Dato --> INDF
      decf NUMERO
      movf NUMERO,W     ; W = W - 1
      decf CONTADOR     ; Decremento el contador
      btfss STATUS,Z    ; Si llega  a cero salta
```

```
      goto Ram

Fin   goto  Fin         ; Fin de programa cíclico

End
```

1.2.3.3.2.- Ejercicio ampliación

```
;*********************************************************
;*          PRACTICA 3                                   *
;*    DIRECCIONAMIENTO Y TABLAS                          *
;*    EJERCICIO AVANZADO                                 *
;*********************************************************
;* Autor: Diego Bugallo Garrido                         *
;*  Controlador: PIC16F84                               *
;*    Ultima revisión: 22 - Marzo - 2013                *
;*********************************************************
;* Descripción: Programar un display de 7 segmentos y   *
;* escribir en RAM mediante direccionamiento indirecto  *
;* Tabla: relaciona un valor en hexadecimal con el      *
;* valor mostrado en un display                         *
;* Memoria: Función para copiar un numero en RAM desde  *
;* una posición inicial a una final dada                *
;*********************************************************


      list p=16F886           ; Tipo de PIC
      #include "P16F886.INC"  ; identifica los ppales registros

;---------------------------------------------------------
;---  DECLARACIÓN DE CTES. Y VARIABLE                    -
;---------------------------------------------------------

      #define     entrada1    .2    ; Primer valor    #define
      entrada2    .15               ; Segundo valor
      #define inicio          0x30  ; Primera dirección a almacenar
      #define fin             0x3F  ; Ultima dirección a borrar
      #define máximo .9 ; Valor máximo para comparar en la tabla

;--- Display siete segmentos  ---

      #define cero      b'00111111'
      #define uno       b'00000110'
```

```
      #define dos       b'01011011'
      #define tres      b'01001111'
      #define cuatro    b'01100110'
      #define cinco     b'01101101'
      #define seis      b'01111101'
      #define siete     b'00000111'
      #define ocho      b'01111111'
      #define nueve     b'01101111'
      #define fallo     b'01111001'       ; Letra E

      PUERTOA equ 0x10  ; Puerto de entrada
      PUERTOB equ 0x11  ; Puerto de salida
      TEMP  equ   0x12  ; Registro temporal

;-----------------------------------------------------------
;---  INICIO DEL PROGRAMA PRINCIPAL                        -
;-----------------------------------------------------------

      org 0x00    ; Posición de memoria de prog. que comienza
      goto Inicio ; Salto a la etiqueta Inicio


      org 0x05    ; Posición de memoria de prog. de la tabla

;-----------------------------------------------------------
;---              TABLA DEL DISPLAY                       -
;-----------------------------------------------------------

Tabla
      movlw máximo
      movwf TEMP
      movf  PUERTOA,W   ; Copio el contenido del puerto a W
      subwf TEMP
      btfss STATUS,C    ; Compruebo si la entrada es 0-9
      movlw maximo+1    ; Si no es el número 0-9 --> W = .10
      movwf TEMP        ; Registro para la escritura en la RAM
      addwf PCL
      retlw cero        ; LED's activados para el cero en un display
      retlw uno         ; LED's activados para el uno en un display
      retlw dos         ; LED's activados para el dos en un display
      retlw tres        ; LED's activados para el tres en un display
      retlw cuatro      ; LED's activados para el cuatro
      retlw cinco       ; LED's activados para el cinco
      retlw seis        ; LED's activados para el seis en un display
```

```
      retlw siete       ; LED's activados para el siete
      retlw ocho        ; LED's activados para el ocho en un display
      retlw nueve       ; LED's activados para el nueve
      retlw fallo       ; LED's activados para la E en un display



;-----------------------------------------------------------
;---                    PROGRAMA PRINCIPAL                -
;-----------------------------------------------------------


Inicio      ; Etiqueta de comienzo de programa

      clrw
      clrf  PUERTOA           ;Inicializo los puertos
      clrf  PUERTOB

      movlw entrada1    ; Simulación en código de una entrada
      movwf PUERTOA     ; No sería necesario con puertos reales

      call  Tabla       ; Conversión de la entrada al display
      movwf PUERTOB     ; Valor de salida al display
      call  Memoria     ; Escritura en la RAM mediante el puntero

      movlw entrada2    ; Simulación de una nueva entrada
      movwf PUERTOA

      call  Tabla       ; Conversión de la entrada al display
      movwf PUERTOB     ; Valor de salida al display
      call  Memoria     ; Escritura en la RAM mediante el puntero

Fin   goto  Fin         ; Fin de programa cíclico


;-----------------------------------------------------------
;---        SUBRUTINA DE ESCRITURA EN RAM                -
;-----------------------------------------------------------


Memoria
      movlw maximo+1
      subwf TEMP,W      ; Compruebo si la entrada es un digito
      btfsc STATUS,Z    ; Si era un digito 0-9 temp = digito
```

```
      clrf  TEMP        ; Si no era un digito temp = 0
      movlw inicio
      movwf FSR         ; el puntero apunta a la dirección inicial
Borrado
      movf  TEMP,W      ; valor a escribir en la RAM
      movwf INDF        ; Copio W en la dirección que apunta FSR
      movlw fin         ; Compruebo si llego a la última dirección
      subwf FSR,W
      btfsc STATUS,Z
      return            ; retorno si FSR => fin
      incf  FSR         ; Si FRS!= fin FSR apunta a la siguiente
      goto  Borrado

End
```

1.2.3.4.- Práctica 4

```
;*********************************************************
;*          PRACTICA 4                                   *
;*                MANEJO DE ENTRADAS Y SALIDAS           *
;*********************************************************
;*    Autor: Diego Bugallo Garrido                       *
;*  Controlador: PIC16F886                               *
;*    Ultima revisión: 10 - Abril - 2013                 *
;*********************************************************
;* Descripción: Programar los puertos A,B como salida    *
;* C como entrada de tal forma que:                       *
;*    IN          OUT                                    *
;*    RA0   RA1   PORT C                                 *
;*    0     0     PORT C = OOh                           *
;*    0     1     PORT C = FFh                           *
;*    1     0     PORT C = PORT B                        *
;*    1     1     PORT C = NOT(PORT B)                   *
;*********************************************************


      list p=16F886                ; Tipo de PIC
      #include "P16F886.INC"  ; identifica los ppales registros

;---------------------------------------------------------
;---  DECLARACIÓN DE CTES. Y VARIABLE                    -
;---------------------------------------------------------
```

```
;-----------------------------------------------------
;---  INICIO DEL PROGRAMA PRINCIPAL                   -
;-----------------------------------------------------

      org 0x00    ; Posición de memoria de prog. que comienza
      goto Inicio ; Salto a la etiqueta Inicio


      org 0x05    ; Posición de memoria de prog. de la tabla



;-----------------------------------------------------
;---                   PROGRAMA PRINCIPAL            -
;-----------------------------------------------------


Inicio            ; Etiqueta de comienzo de programa

;--- Configuración de los puertos ---
      clrw
      clrf  PORTA       ; Se borran los latch de salida del puerto
      clrf  PORTB
      clrf  PORTC
      bsf   STATUS,RP0
      bsf STATUS,RP1    ; Banco 3
      clrf  ANSEL       ; Entradas digitales
      clrf  ANSELH
      bcf STATUS,RP1    ; Banco 1
      movlw 0xFF        ;
      movwf PORTA       ; Puerto A (In)
      movwf PORTB       ; Puerto B (In)
      clrf  PORTC       ; Puerto C (Out)
      bcf   STATUS,RP0  ; Banco 0

;--- Programa ---

Leer  movf  PORTA,W
            andlw b'00000011' ; W = 0000 00XX
            call  Tabla       ; Devuelve en W el estado del puerto C
```

```
            movwf PORTC       ; Copio en el Puerto C el registro W
            Goto  Leer



Tabla       addwf PCL               ; Añado al contador del Programa
la máscara
                  retlw 0x00        ; 00 --> W = 00h
                  retlw 0xFF        ; 01 --> W = FFh
                  goto  Entrada_10  ; 10
                  comf  PORTB,W     ; 11 --> W = NOT (PORTB)
                  return
Entrada_10  movf  PORTB,W           ; 10 --> W = PORTB
                  return


Fin   goto  Fin         ; Fin de programa cíclico



End
```

1.2.3.5.- Práctica 5

```
;*********************************************************
;*                                  PRACTICA 5           *
;*                TEMPORIZADOR / CONTADOR                *
;*********************************************************
;*    Autor: Diego Bugallo Garrido                       *
;*  Controlador: PIC16F886                               *
;*    Ultima revisión: 18 - Abril - 2013                 *
;*********************************************************
;* Descripción: Programar un juego de 8 luces de tal     *
;* forma que cada 0,5 ms rote a izquierdas o derechas    *
;* según el estado de RA0                                *
;*********************************************************


      list p=16F886                ; Tipo de PIC
      #include "P16F886.INC"   ; identifica los ppales registros
      #define Fosc 4000000     ;Velocidad de trabajo

;-------------------------------------------------------
```

```
;---  DECLARACIÓN DE CTES. Y VARIABLE                  -
;-----------------------------------------------------------

      MSE_Delay_V equ 0x73    ;Variables (3) de la librería



;-----------------------------------------------------------
;---  INICIO DEL PROGRAMA PRINCIPAL                    -
;-----------------------------------------------------------

      org 0x00    ; Posición de memoria de prog. que comienza
      goto Inicio ; Salto a la etiqueta Inicio


      org 0x05    ; Posición de memoria de prog. de la tabla

      include     "MSE_Delay.inc"   ;Incluir rutinas de temporización

;-----------------------------------------------------------
;---                    PROGRAMA PRINCIPAL               -
;-----------------------------------------------------------


Inicio      ; Etiqueta de comienzo de programa

;--- Configuración de los puertos ---
      clrw
      clrf  PORTA       ; Se borran los latch de salida del puerto
      clrf  PORTB
      bsf   STATUS,RP0
      bsf STATUS,RP1    ; Banco 3
      clrf  ANSEL       ; Entradas digitales
      clrf  ANSELH
      bcf STATUS,RP1    ; Banco 1
      movlw 0xFF        ;
      movwf PORTA       ; Puerto A (In)
      clrf  PORTB       ; Puerto B (Out)
      bcf   STATUS,RP0  ; Banco 0

;--- Programa ---
      movlw 0x01        ; Posición inicial de las luces
      movwf PORTB
```

```
Luces Delay 1000 Milis        ; Tiempo de espera hasta rotación
            btfss PORTA,0     ; RA0=1?
            goto  rota_dcha   ; RA0=0 --> rota a dererechas
            rlf   PORTB       ; RA0=1 --> rota a izquierdas
            goto Luces
rota_dcha   rrf PORTB
            goto Luces

End
```

1.2.3.6.- Práctica 6

```
;*******************************************************
;*          PRACTICA 6  - A                            *
;*    INTERRPCIÓN POR TEMPORIZADOR                     *
;*******************************************************
;* Autor: Diego Bugallo Garrido                        *
;*  Controlador: PIC16F886                             *
;*    Ultima revisión: 25 - Abril - 2013               *
;*******************************************************
;* Descripción: LED que parpadee cada 0.25s            *
;*******************************************************


      list p=16F886                ; Tipo de PIC
      #include "P16F886.INC"   ; identifica los ppales registros


;-------------------------------------------------------
;---  DECLARACIÓN DE CTES. Y VARIABLE                  -
;-------------------------------------------------------
      #define cuenta1 .217
      #define cuenta2 .25

      CONTADOR equ      0x30
;-------------------------------------------------------
;---  INICIO DEL PROGRAMA PRINCIPAL                      -
;-------------------------------------------------------

      org 0x00    ; Posición de memoria de prog. que comienza
      goto Inicio ; Salto a la etiqueta Inicio

      org 0x04    ; Posición del vector Interrupción
```

```
      goto Interrupcion
      org 0x05    ; Posición de memoria de prog. de la tabla


;-----------------------------------------------------------
;---                    PROGRAMA PRINCIPAL                -
;-----------------------------------------------------------


Inicio            ; Etiqueta de comienzo de programa

;--- Configuración de los puertos ---
      clrw
      clrf  PORTA
      bsf   STATUS,RP0
      bsf STATUS,RP1          ; Banco 3
      clrf  ANSEL       ; Entradas digitales
      clrf  ANSELH
      bcf STATUS,RP1          ; Banco 1
      clrf  PORTA       ; Puerto A (Out)

;--- Configuración del TMR0
      movlw b'0000111' ; modo timer escala 1:256 flanco ascendente
      movwf OPTION_REG
      bcf   STATUS,RP0        ; Banco 0


;--- Programa ---

      movlw cuenta1     ; carga inicial del TMR0
      movwf TMR0
      movlw b'10100000' ; Habilito solo int TMR0
      movwf INTCON

Led   movlw cuenta2
      subwf CONTADOR,W
      btfss STATUS,Z    ; CONTADOR = 50 ?
      goto  Led               ; NO
      movlw 0x01        ; SI
      xorwf PORTA       ; RA0 cambia de estado
      clrf  CONTADOR    ; Pongo a cero contador
      goto  Led
```

```
Interrupcion
      bcf   INTCON,T0IF ; desactivo el flag
      incf  CONTADOR    ; Aumento contador
      movlw cuenta1     ; actualizo el TMRO
      movwf TMR0        ;
      retfie

End
```

1.2.3.7.- Práctica 7

```
;*********************************************************
;*    PRACTICA 7                                         *
;*    MEMORIA EEPROM                                     *
;*********************************************************
;*    Autor: Diego Bugallo Garrido                       *
;*  Controlador: PIC16F886                               *
;*    Ultima revisión: 15 - Mayo - 2013                  *
;*********************************************************
;* Descripción: Contador desde 0 a9 representado en un   *
;* display, se incrementa mediante un pulsador en RA0    *
;* y cada vez que se pulsa se guarda el dato en la ROM   *
;*********************************************************


      list p=16F886            ; Tipo de PIC
      #include "P16F886.INC"   ;identifica los ppales registros
      #define Fosc 4000000     ;Velocidad de trabajo

;---------------------------------------------------------
;---  DECLARACIÓN DE CTES. Y VARIABLE                   -
;---------------------------------------------------------
      MSE_Delay_V equ   0x73  ;Variables (3) de temporización
      CUENTA equ 0x30         ;Variable para contar el turno

      #define máximo  .9      ; Máximo de número a representar
;--- Display   ---

      #define cero      b'00111111'
      #define uno       b'00000110'
      #define dos       b'01011011'
      #define tres      b'01001111'
      #define cuatro    b'01100110'
```

```
      #define cinco     b'01101101'
      #define seis      b'01111101'
      #define     siete b'00000111'
      #define ocho      b'01111111'
      #define nueve     b'01101111'
      #define fallo     b'01111001'


;-----------------------------------------------------
;---  INICIO DEL PROGRAMA PRINCIPAL                  -
;-----------------------------------------------------

      org 0x00    ; Posición de memoria de prog. que comienza
      goto Inicio ; Salto a la etiqueta Inicio

      org 0x05    ; Posición de memoria de prog. de la tabla
      include     "MSE_Delay.inc"   ;Incluir rutinas de temporización

;-----------------------------------------------------
;---              TABLA DEL DISPLAY                  -
;-----------------------------------------------------

Tabla
      addwf PCL
      retlw cero  ; LED's activados para el cero en un display
      retlw uno   ; LED's activados para el uno en un display
      retlw dos   ; LED's activados para el dos en un display
      retlw tres  ; LED's activados para el tres en un display
      retlw cuatro; LED's activados para el cuatro en un display
      retlw cinco ; LED's activados para el cinco en un display
      retlw seis  ; LED's activados para el seis en un display
      retlw siete ; LED's activados para el siete en un display
      retlw ocho  ; LED's activados para el ocho en un display
      retlw nueve ; LED's activados para el nueve en un display
      retlw fallo ; LED's activados para la E en un display

;-----------------------------------------------------
;---                    PROGRAMA PRINCIPAL           -
;-----------------------------------------------------


Inicio      ; Etiqueta de comienzo de programa

;--- Configuración de los puertos ---
```

```
      clrw
      clrf  PORTA
      clrf  PORTB       ; Borra los latch de salida
      bsf   STATUS,RP0
      bsf STATUS,RP1    ; Banco 3
      clrf  ANSEL       ; Entradas digitales
      clrf  ANSELH
      bcf STATUS,RP1    ; Banco 1
      clrf  PORTB       ; Puerto B (Out)
      movlw 0xFF        ; Puerto A (In)
      movwf PORTA
      bcf   STATUS,RP0  ; Banco 0

;--- Programa carga inicial ---

      bsf         STATUS,RP1  ; Banco 2
      clrf  EEADR       ; Posición de ROM donde se guarda el dato
      call  Lee_eeprom  ; Lee el dato de EEADR
      bsf   STATUS,RP1  ; Banco 2
      movlw 0xFF
      subwf EEDATA,W
      bcf   STATUS,RP1  ; banco 0
      btfss STATUS,Z    ; ¿ MEMORIA SIN USAR ?
      goto  Dato_inicial
      clrf  CUENTA      ; SI --> CONTADOR = 0


;--- Modo bucle

Display           movf  CUENTA,W
                  call  Tabla
                  movwf PORTB       ; Visualizo en el puerto B
                  btfss PORTA,0     ; Miro si se activa el pulsador
                  goto  Display
                  Delay 10 Milis    ; Para evitar rebotes
                  incf  CUENTA      ; Aumento en uno contador
                  movlw maximo+1
                  subwf CUENTA,W
                  btfsc STATUS,Z    ; Contador = 10 ?
                  clrf  CUENTA
                  movf  CUENTA,W    ; Paso contador a W
                  bsf   STATUS,RP1  ; Banco 2
                  movwf EEDATA      ; EEDAT = CONTADOR
                  clrf  EEADR       ; EEADR = 0x00h
```

```
                call  Escribe_eeprom
                goto  Display

Lee_eeprom      bsf         STATUS,RP0
                bsf         STATUS,RP1        ; Banco 3
                bcf         EECON1,EEPGD      ;EEPROM de datos
                bsf         EECON1,RD         ;Orden de lectura
                bcf         STATUS,RP0
                bcf         STATUS,RP0        ;Selección de banco 0
           return

Escribe_eeprom  bsf         STATUS,RP0
                bsf         STATUS,RP1        ;banco 3
                bcf         EECON1,EEPGD      ;EEPROM de datos
                bsf         EECON1,WREN       ;Permiso de escritura
                movlw       b'01010101'
                movwf       EECON2
                movlw       b'10101010'
                movwf       EECON2          ;Secuencia de Microchip
                bsf         EECON1,WR   ;Orden de escritura
Espera          btfsc   EECON1,WR   ;Testear flag de fin de escritura
               goto     Espera
                bcf      EECON1,WREN      ;Fin permiso de escritura
                bcf      EECON1,EEIF      ;Borra el flag
                bcf      STATUS,RP0
                bcf   STATUS,RP1        ;Selecciona banco 0
                      return

Dato_inicial    bsf   STATUS,RP1
                movf  EEDATA,W    ; NO --> CONTADOR = DATO GUARDADO
                bcf   STATUS,RP1  ; Banco 0
                movwf CUENTA      ;
                goto  Display
End
```

1.2.3.8.- Práctica 8

```
;*******************************************************
;*          PRACTICA 8                                 *
;*   MANEJO DE LCD                                     *
;*******************************************************
;*    Autor: Diego Bugallo Garrido                     *
;*  Controlador: PIC16F886                             *
```

```
;*    Ultima revisión: 20 - Mayo - 2013                *
;*****************************************************
;* Descripción: Realizar un programa quie visualice por *
;* pantalla la palabra Hola                            *
;*****************************************************


      list p=16F886                 ; Tipo de PIC
      #include "P16F886.INC"   ; identifica los ppales registros


;-----------------------------------------------------
;---  DECLARACIÓN DE CTES. Y VARIABLE                -
;-----------------------------------------------------
      Lcd_var     equ   0x70  ;Variables (3) del LCD


;-----------------------------------------------------
;---  INICIO DEL PROGRAMA PRINCIPAL                    -
;-----------------------------------------------------

      org 0x00    ; Posición de memoria de prog. que comienza
      goto Inicio ; Salto a la etiqueta Inicio

      org 0x05    ; Posición de memoria de prog. de la tabla
      include     "LCD4bitsPIC16.inc"     ;Libreria LCD


;-----------------------------------------------------
;---                    PROGRAMA PRINCIPAL           -
;-----------------------------------------------------


Inicio            ; Etiqueta de comienzo de programa

;--- Configuración de los puertos ---
      clrw
      clrf  PORTA
      clrf  PORTB ; Borra los latch de salida
      bsf   STATUS,RP0
      bsf STATUS,RP1    ; Banco 3
      clrf  ANSEL ; Entradas digitales
      clrf  ANSELH
      bcf STATUS,RP1    ; Banco 1
```

```
      clrf  PORTB ; Puerto B (Out)
      clrf  PORTA ; Puerto A (Out)
      bcf   STATUS,RP0  ; Banco 0

;--- Programa LCD ---
      call  UP_LCD            ; Puertos LCD
      call  LCD_INI           ; Inicialización del LCD
      movlw b'00001111' ; LCD=On, Cursor = on, Parpadeo = On
      call  LCD_REG
      movlw 'H'
      call  LCD_DATO    ; Visualizo H
      movlw 'o'
      call  LCD_DATO    ; Visualizo o
      movlw 'l'
      call  LCD_DATO    ; Visualizo l
      movlw 'a'
      call  LCD_DATO    ; Visualizo a

Fin goto    Fin
End
```

1.2.3.9.- Práctica 9

```
;*******************************************************
;*          PRACTICA 9                                 *
;*        TECLADO MATRICIAL                            *
;*******************************************************
;*    Autor: Diego Bugallo Garrido                     *
;*  Controlador: PIC16F886                             *
;*    Ultima revisión: 5 - Junio - 2013                *
;*******************************************************
;* Descripción: Comparar un password introducido por un *
;* teclado, con un número almacenado en la ROM, si es   *
;* correcto se encenderá un led. El passwor tiene 3     *
;* dígitos                                              *
;*******************************************************


      list p=16F886           ; Tipo de PIC
      #include "P16F886.INC"   ; identifica los ppales registros
      #define Fosc 4000000     ;Velocidad de trabajo

;-------------------------------------------------------
```

```
;---  DECLARACIÓN DE CTES. Y VARIABLE                  -
;-----------------------------------------------------------
      Key_var     equ   0x73  ;Variables (6) del teclado
      PASS1 equ   0x30
      PASS2 equ 0x31
      PASS3 equ 0x32
      TECLA1 equ 0x40
      TECLA2 equ 0x41
      TECLA3 equ 0x42



;-----------------------------------------------------------
;---  INICIO DEL PROGRAMA PRINCIPAL                    -
;-----------------------------------------------------------

      org 0x00    ; Posición de memoria de prog. que comienza
      goto Inicio ; Salto a la etiqueta Inicio

      org 0x05    ; Posición de memoria de prog. de la tabla
      include     "teclado.inc"     ;Incluir rutinas de temporización


;-----------------------------------------------------------
;---                    PROGRAMA PRINCIPAL               -
;-----------------------------------------------------------


Inicio      ; Etiqueta de comienzo de programa

;--- Configuración de los puertos ---
      clrw
      clrf  PORTA
      clrf  PORTB       ; Borra los latch de salida
      bsf   STATUS,RP0
      bsf STATUS,RP1    ; Banco 3
      clrf  ANSEL       ; Entradas digitales
      clrf  ANSELH
      bcf STATUS,RP1    ; Banco 1
      movlw 0x0F        ; RB0-RB3 = In
      movwf PORTB       ; RB4-RB7 = Out
      clrf  PORTA       ; PORT A (Out)
      bcf   OPTION_REG,7
      bcf   STATUS,RP0        ; Banco 0
```

```
;--- Programa carga inicial ---

      bsf   STATUS,RP1  ; Banco 2
      clrf  EEADR       ; Posición de ROM donde se guarda el dato
      call  Lee_eeprom  ; Lee el dato de EEADR
      bsf   STATUS,RP1  ; Banco 2
      movf  EEDATA,W
      bcf   STATUS,RP1  ; Banco 0
      movwf PASS1       ; ROM --> RAM
      bsf   STATUS,RP1  ; Banco 2
      incf  EEADR       ; Siguiente dígito
      call  Lee_eeprom
      bsf   STATUS,RP1  ; Banco 2
      movf  EEDATA,W
      bcf   STATUS,RP1  ; Banco 0
      movwf PASS2       ; ROM --> RAM
      bsf   STATUS,RP1  ; Banco 2
      incf  EEADR       ; Siguiente dígito
      call  Lee_eeprom
      bsf   STATUS,RP1  ; Banco 2
      movf  EEDATA,W
      bcf   STATUS,RP1  ; Banco 0
      movwf PASS3       ; ROM --> RAM

; --- CAPTURA DE TECLAS
Tecla1      call  Key_Scan    ; capturo tecla
            movf  Tecla,W
            movwf TECLA1
            movlw 0x80        ; compruebo si se ha pulsado alguna
            subwf TECLA1,W
            btfsc STATUS,Z    ; ¿Se pulsó tecla?
            goto  Tecla1           ; NO --> Esperar captura
Tecla2      call  Key_Scan    ; capturo tecla 2
            movf  Tecla,W
            movwf TECLA2
            movlw 0x80        ; compruebo si se ha pulsado alguna
            subwf TECLA2,W
            btfsc STATUS,Z    ; ¿Se pulso tecla?
            goto  Tecla2           ; NO --> Esperar captura
Tecla3      call  Key_Scan    ; capturo tecla
            movf  Tecla,W
            movwf TECLA3
            movlw 0x80        ; compruebo si se ha pulsado alguna
```

```
            subwf TECLA3,W
            btfsc STATUS,Z    ; ¿Se pulso tecla?
            goto  Tecla3            ; NO --> Esperar captura

; --- COMPROBACIÓN DE PASS
            movf  TECLA1,W
            subwf PASS1,W           ; Primer dígito
            btfss STATUS,Z    ; ¿Dígito ok?
            goto  Tecla1            ; No --> Nuevas teclas
            movf  TECLA2,W    ; Si --> siguiente tecla
            subwf PASS2,W           ; Primer dígito
            btfss STATUS,Z    ; ¿Dígito ok?
            goto  Tecla1            ; No --> Nuevas teclas
            movf  TECLA3,W    ; Si --> siguiente tecla
            subwf PASS3,W           ; Primer dígito
            btfss STATUS,Z    ; ¿Dígito ok?
            goto  Tecla1            ; No --> Nuevas teclas
            bsf         PORTA,0           ; Si --> Activo led

Fin   goto Fin



Lee_eeprom  bsf         STATUS,RP0
            bsf         STATUS,RP1  ; Banco 3
            bcf         EECON1,EEPGD      ;EEPROM de datos
            bsf         EECON1,RD         ;Orden de lectura
            bcf         STATUS,RP0
            bcf         STATUS,RP0        ;Selección de banco 0
            return

End
```

1.2.4.- Otros anejos

1.2.4.1.- PIC16F886

1.2.4.1.1.- Introducción

El PIC 16F886 pertenece a la familia de microcontroladores 16F88X de microchip que trabajan hasta una frecuencia de reloj de 20MHz, utilizan 35 instrucciones de programación y es capaz de almacenar 8 niveles de pila.

1.2.4.1.2.- Características

El PIC 16F886 contiene una memoria de programa de 8K, una memoria de datos de 368 bytes y una memoria EEPROM de 256 bytes. Consta de 28 pines de los cuales 24 pueden funcionar como I/O.

Posee entre otros elementos un comparador analógico, un conversor A/D, 2 temporizadores de 8bits y un temporizador de 16 bits con preescalado.

A continuación se muestra una tabla con las diferencias entre los PICs de la familia 16F88X, (Figura 237)

PIC16F882/883/884/886/887 Family Types

Device	Program Memory	Data Memory		I/O	10-bit A/D (ch)	ECCP/ CCP	EUSART	MSSP	Comparators	Timers 8/16-bit
	Flash (words)	SRAM (bytes)	EEPROM (bytes)							
PIC16F882	2048	128	128	24	11	1/1	1	1	2	2/1
PIC16F883	4096	256	256	24	11	1/1	1	1	2	2/1
PIC16F884	4096	256	256	35	14	1/1	1	1	2	2/1
PIC16F886	8192	368	256	24	11	1/1	1	1	2	2/1
PIC16F887	8192	368	256	35	14	1/1	1	1	2	2/1

Figura 237. *Familia PIC16F88X.* [7]

1.2.4.1.3.- Diagrama de pines

Los 28 pines del microcontrolador se muestran en el siguiente diagrama, (figura238):

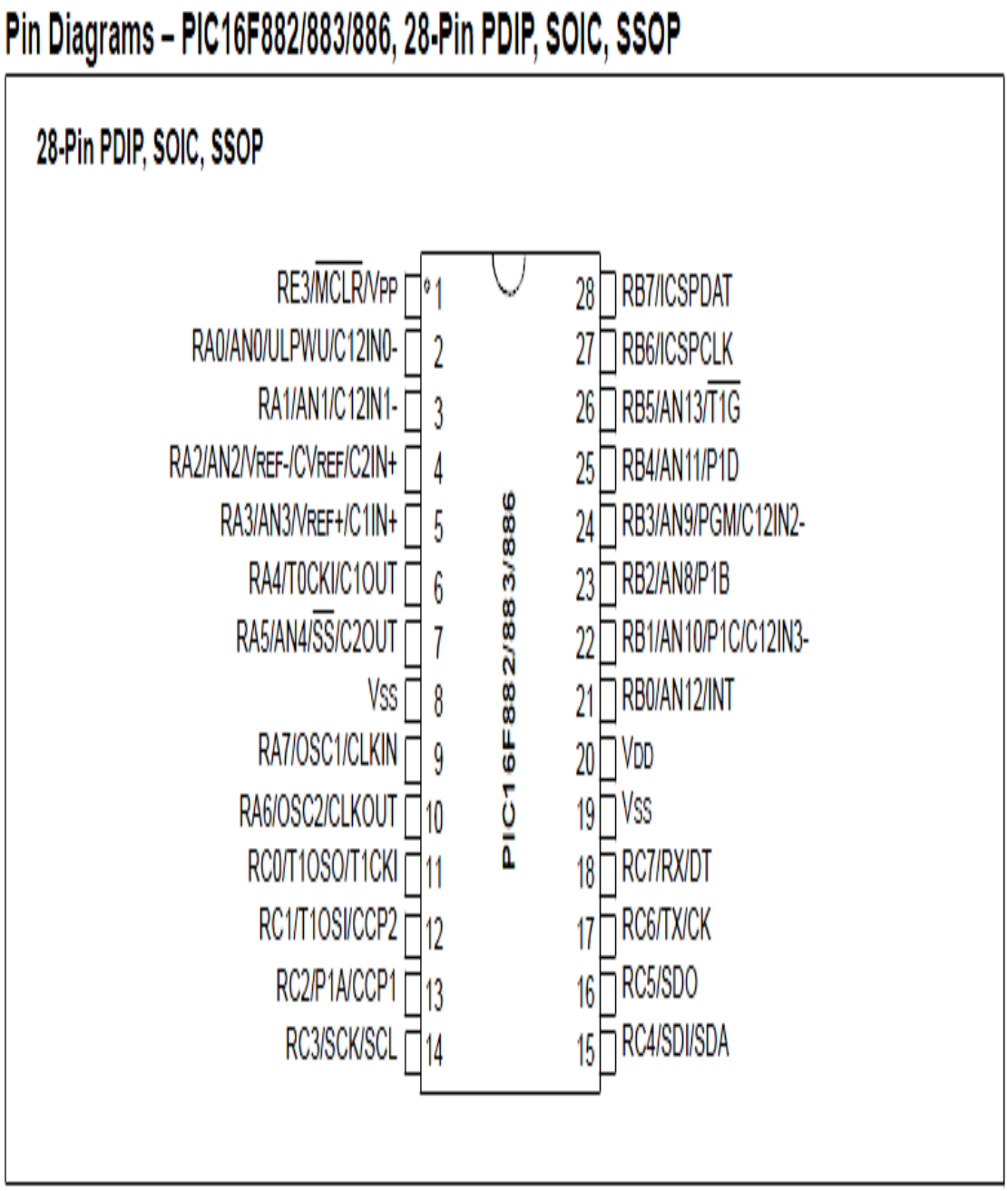


Figura 238. *Diagrama de pin.*[7]

Cada pin puede tener uno o varios usos, en la siguiente tabla se observa el desglose de cada pin en función de su uso, (Figura 239):

PIC16F882/883/884/886/887

TABLE 1: 28-PIN PDIP, SOIC, SSOP ALLOCATION TABLE (PIC16F882/883/886)

I/O	28-Pin PDIP/SOIC/SSOP	Analog	Comparators	Timers	ECCP	EUSART	MSSP	Interrupt	Pull-up	Basic
RA0	2	AN0/ULPWU	C12IN0-	—	—	—	—	—	—	—
RA1	3	AN1	C12IN1-	—	—	—	—	—	—	—
RA2	4	AN2	C2IN+	—	—	—	—	—	—	VREF-/CVREF
RA3	5	AN3	C1IN+	—	—	—	—	—	—	VREF+
RA4	6	—	C1OUT	T0CKI	—	—	—	—	—	—
RA5	7	AN4	C2OUT	—	—	—	$\overline{SS}$	—	—	—
RA6	10	—	—	—	—	—	—	—	—	OSC2/CLKOUT
RA7	9	—	—	—	—	—	—	—	—	OSC1/CLKIN
RB0	21	AN12	—	—	—	—	—	IOC/INT	Y	—
RB1	22	AN10	C12IN3-	—	P1C	—	—	IOC	Y	—
RB2	23	AN8	—	—	P1B	—	—	IOC	Y	—
RB3	24	AN9	C12IN2-	—	—	—	—	IOC	Y	PGM
RB4	25	AN11	—	—	P1D	—	—	IOC	Y	—
RB5	26	AN13	—	$\overline{T1G}$	—	—	—	IOC	Y	—
RB6	27	—	—	—	—	—	—	IOC	Y	ICSPCLK
RB7	28	—	—	—	—	—	—	IOC	Y	ICSPDAT
RC0	11	—	—	T1OSO/T1CKI	—	—	—	—	—	—
RC1	12	—	—	T1OSI	CCP2	—	—	—	—	—
RC2	13	—	—	—	CCP1/P1A	—	—	—	—	—
RC3	14	—	—	—	—	—	SCK/SCL	—	—	—
RC4	15	—	—	—	—	—	SDI/SDA	—	—	—
RC5	16	—	—	—	—	—	SDO	—	—	—
RC6	17	—	—	—	—	TX/CK	—	—	—	—
RC7	18	—	—	—	—	RX/DT	—	—	—	—
RE3	1	—	—	—	—	—	—	—	Y(1)	$\overline{MCLR}$/VPP
—	20	—	—	—	—	—	—	—	—	VDD
—	8	—	—	—	—	—	—	—	—	VSS
—	19	—	—	—	—	—	—	—	—	VSS

Note 1: Pull-up activated only with external $\overline{MCLR}$ configuration.

Figura 239. *Localización de pines.*[7]

- ✓ MSSP: Master Synchronous Serial Port
- ✓ ECCP: Captura, comparación y PWM

- ✓ EUSART: Transmisor/Receptor Universal Síncrono/Asíncrono. Mejorado

1.2.4.1.4.- Diagrama del PIC16F886

El PIC 16F886 está estructurado internamente de la siguiente manera,(Figura 4):

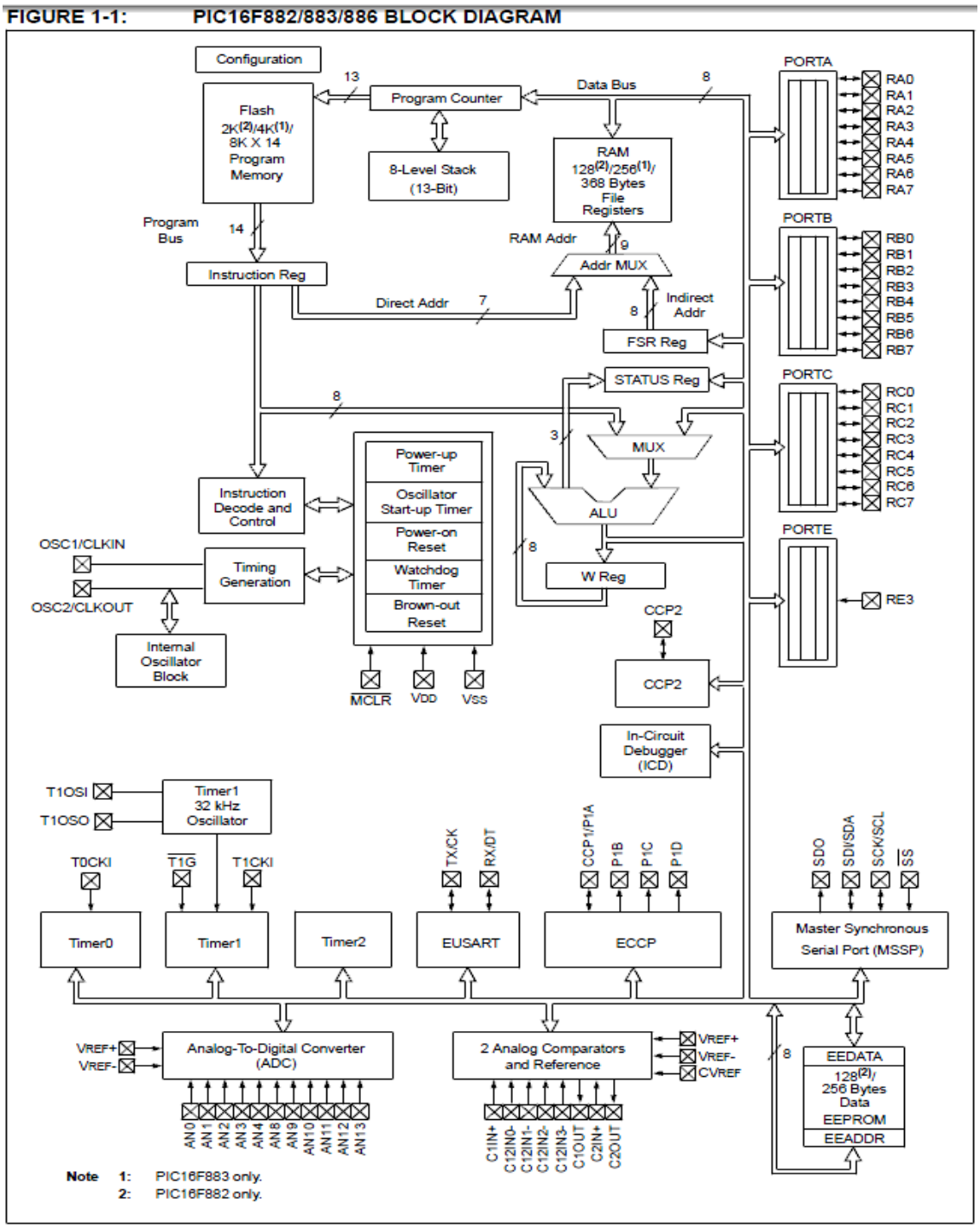


Figura 240. *Diagrama de bloques.*[7]

1.2.4.1.5.- Organización de la memoria

Posee un contador de programa de 13 bits y una memoria de programa de 8K desde la dirección 0000h hasta la 1FFFh, el vector de reset se encuentra en la dirección 0000h y el de interrupciones en la 0004h.

La pila es de 8 niveles por lo que es capaz de almacenar 8 direcciones de retorno de subrutina. (Figura 5):

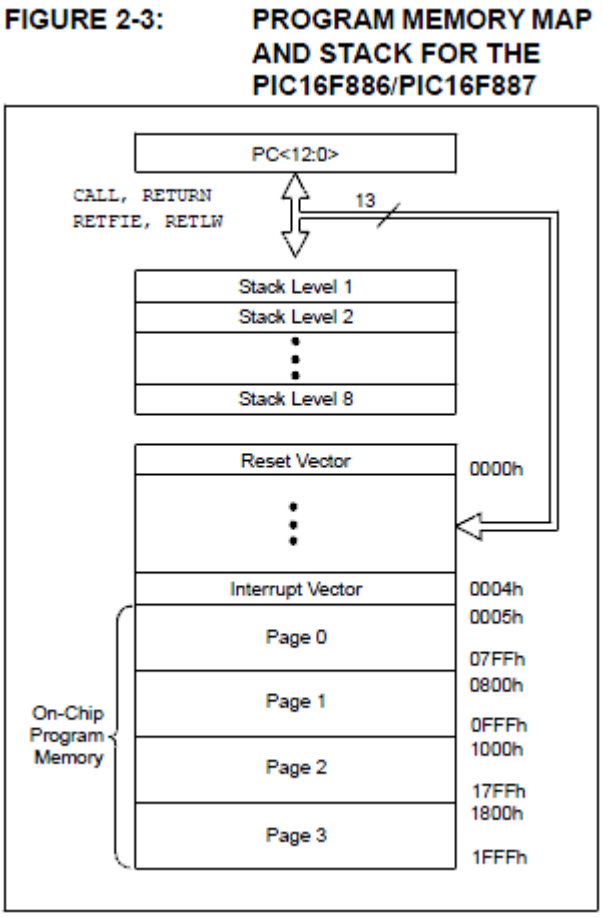


Figura 241. *Organización de la memoria.* [7]

Su memoria RAM se encuentra dividida en 4 bancos y se accede a ellos mediante el registro STATUS, en ella se encuentran los registros especiales, el mapa de la memoria de datos es el siguiente,(Figura 242):

FIGURE 2-6: PIC16F886/PIC16F887 SPECIAL FUNCTION REGISTERS

	File Address		File Address		File Address		File Address
Indirect addr. (1)	00h	Indirect addr. (1)	80h	Indirect addr. (1)	100h	Indirect addr. (1)	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h	WDTCON	105h	SRCON	185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h	CM1CON0	107h	BAUDCTL	187h
PORTD(2)	08h	TRISD(2)	88h	CM2CON0	108h	ANSEL	188h
PORTE	09h	TRISE	89h	CM2CON1	109h	ANSELH	189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDAT	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2(1)	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Reserved	18Eh
TMR1H	0Fh	OSCCON	8Fh	EEADRH	10Fh	Reserved	18Fh
T1CON	10h	OSCTUNE	90h		110h		190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h	WPUB	95h		115h		195h
CCPR1H	16h	IOCB	96h	General Purpose Registers	116h	General Purpose Registers	196h
CCP1CON	17h	VRCON	97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h	16 Bytes	119h	16 Bytes	199h
RCREG	1Ah	SPBRGH	9Ah		11Ah		19Ah
CCPR2L	1Bh	PWM1CON	9Bh		11Bh		19Bh
CCPR2H	1Ch	ECCPAS	9Ch		11Ch		19Ch
CCP2CON	1Dh	PSTRCON	9Dh		11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
General Purpose Registers 96 Bytes	20h – 6Fh	General Purpose Registers 80 Bytes	A0h – EFh	General Purpose Registers 80 Bytes	120h – 16Fh	General Purpose Registers 80 Bytes	1A0h – 1EFh
	70h – 7Fh	accesses 70h-7Fh	F0h – FFh	accesses 70h-7Fh	170h – 17Fh	accesses 70h-7Fh	1F0h – 1FFh
Bank 0		Bank 1		Bank 2		Bank 3	

Unimplemented data memory locations, read as '0'.

Note 1: Not a physical register.

2: PIC16F887 only.

Figura 242. *Bancos y sus registros.* [7]

1.2.4.2.- MPLAB

1.2.4.2.1- Introducción

MPLAB es una herramienta gratuita creada por Microchip, simulación, emulación e incluso capaz de grabar físicamente en un PIC, que permite la realización de proyectos en distintos tipos de lenguajes de programación, entre ellos en ensamblador.

1.2.4.2.2- Creación de un proyecto

Para crear un proyecto nuevo se puede usar el asistente desde el menú *proyect -> Proyect Wizard* como se indica en la figura 243:

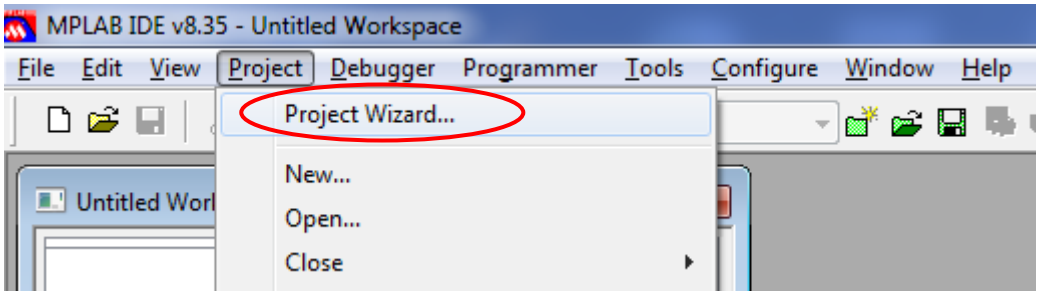


Figura 243. *Creación de un proyecto nuevo.*

A continuación se selecciona el PIC con el que se va a trabajar, (Figura 244)

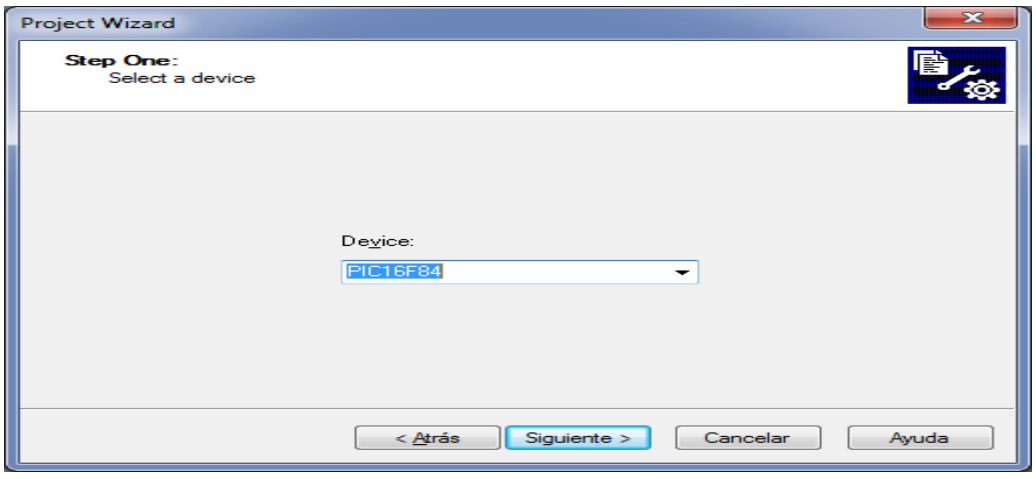


Figura 244. *Selección del dispositivo.*

Después se escoge la herramienta de compilación, *Microsoft MPASM Toolsiute* para compilar en ensamblador,(Figura 245):

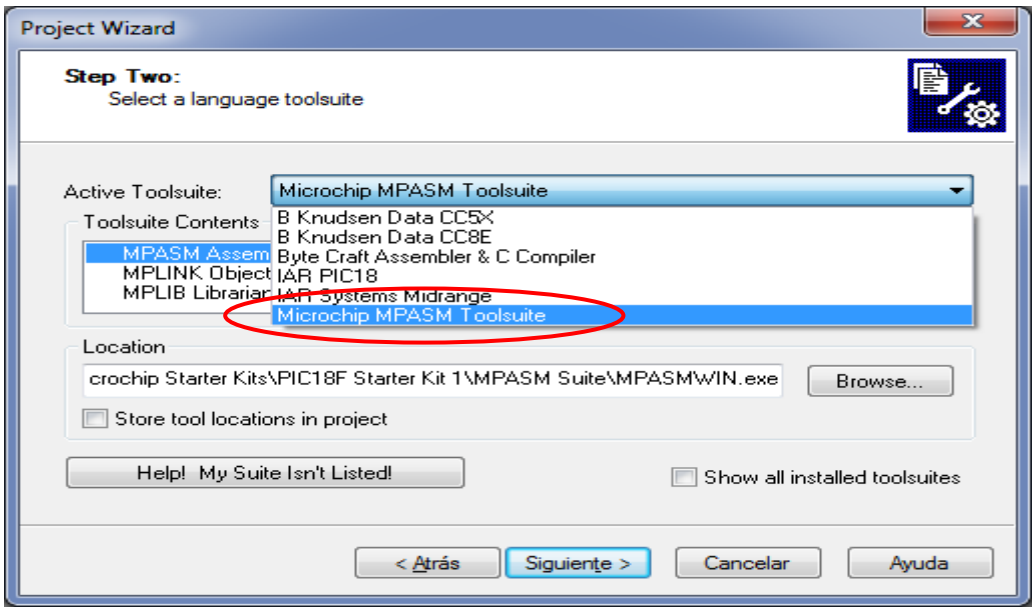


Figura 245. *Selección del compilador.*

Luego se introduce el nombre del nuevo proyecto y se elige su ubicación,(Figura 246):

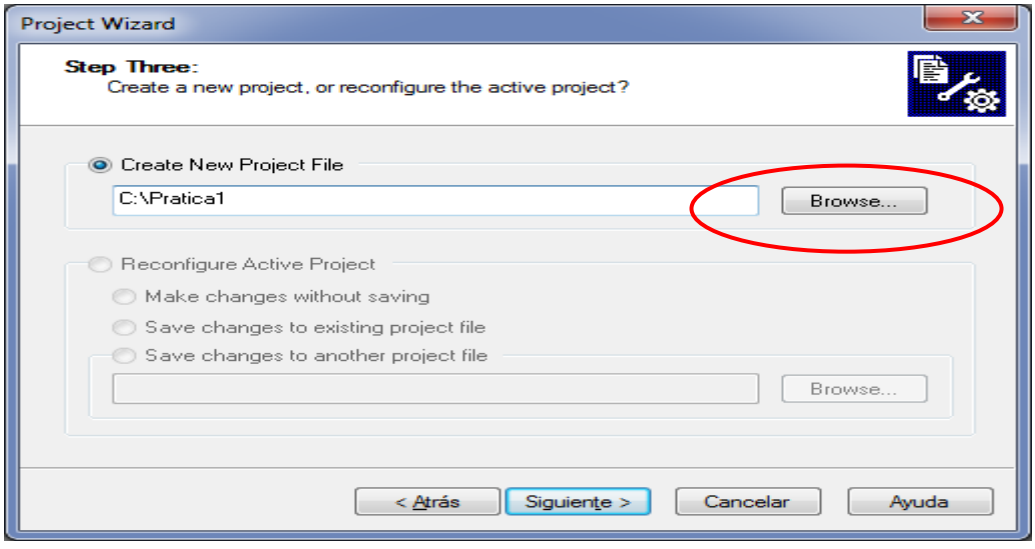


Figura 246. *Selección del proyecto.*

Por último se puede añadir archivos que puedan ser útiles al proyecto,(Figura 247):

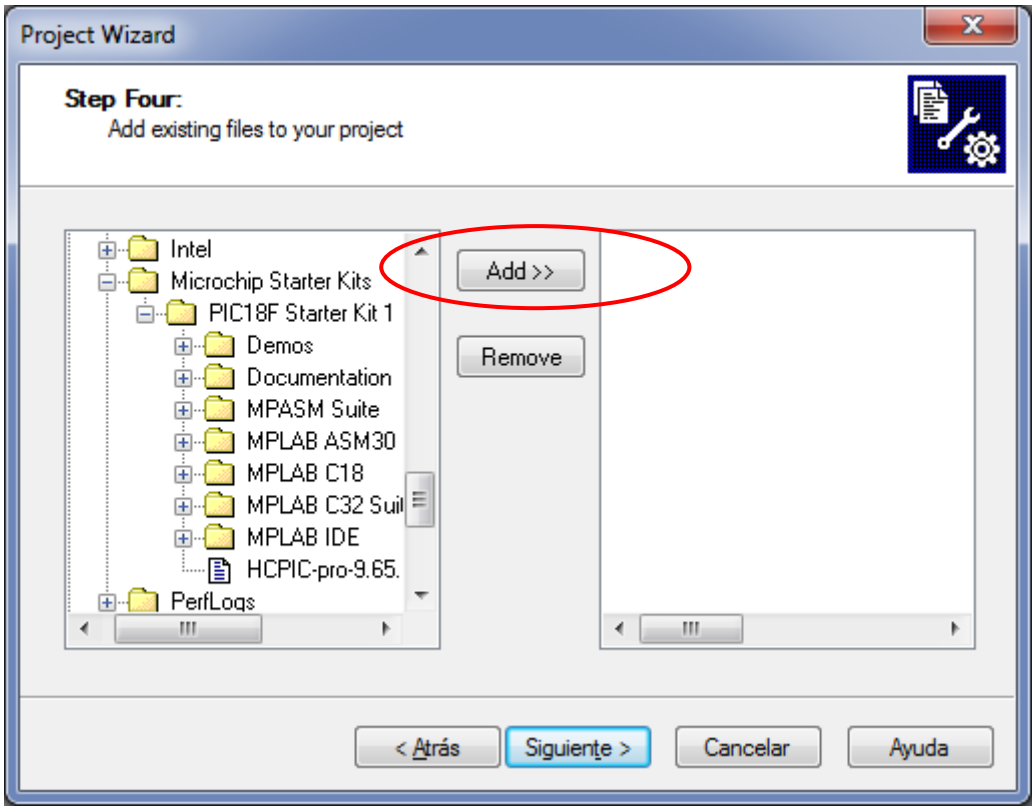


Figura 247. *Añadir archivos existentes.*

1.2.4.2.3- Entorno de trabajo

En la figura 248 se muestras las distintas zonas de trabajo con sus ventanas principales a la hora de realizar un proyecto.

- ✓ Archivos.
- ✓ Registros
- ✓ Variables.
- ✓ Entradas
- ✓ Código
- ✓ ROM
- ✓ RAM
- ✓ Compilación.

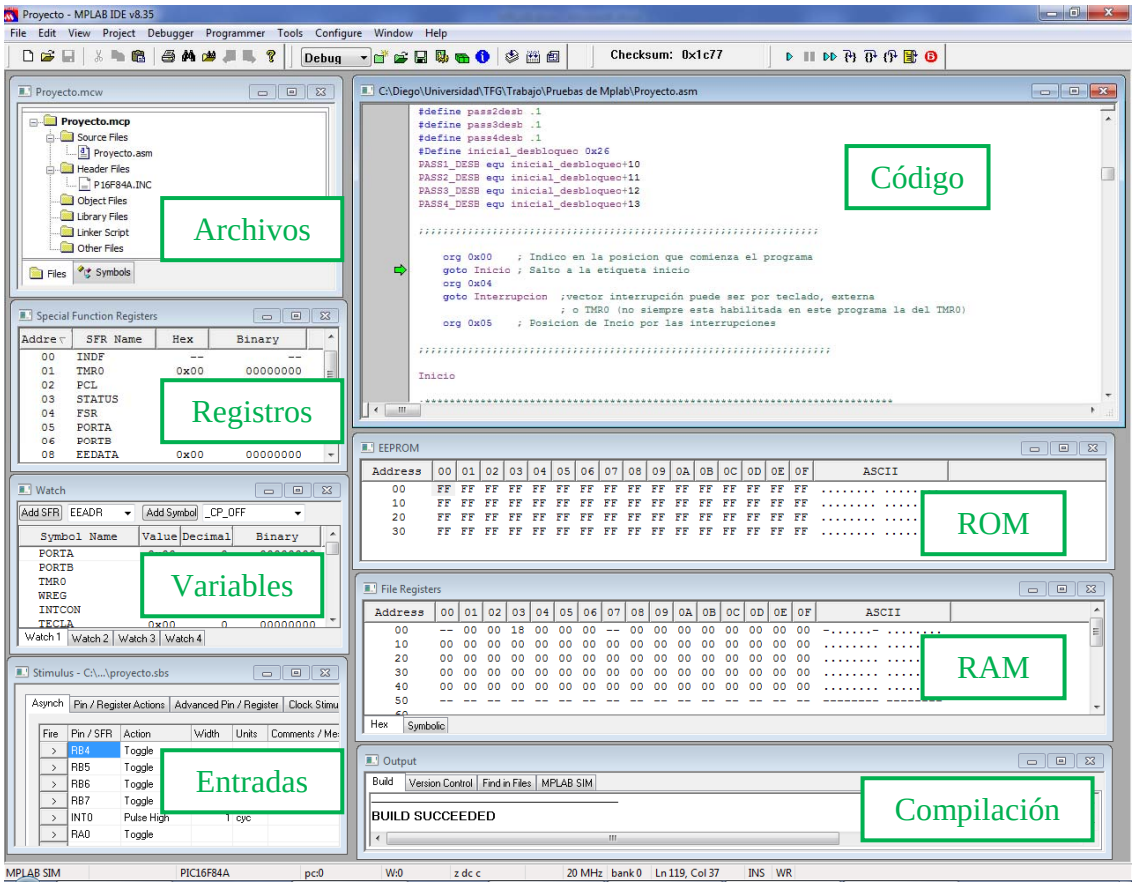


Figura 248. *Estructura del MPLAB.*

Los iconos que se encuentran en la barra de herramienta son accesos rápidos a comandos que se encuentran en el menú de control.

Se destacan los siguientes iconos.(Figura 249):

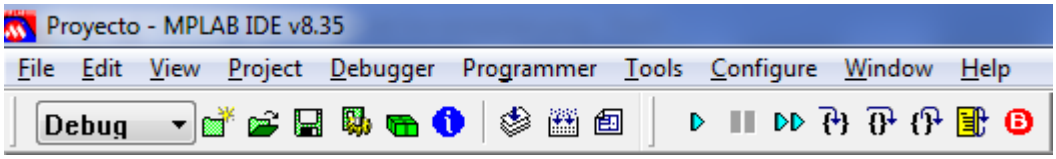


Figura 249. *Barra de Herramientas.*

1.2.4.2.4- Inicio del proyecto

Es necesario seleccionar el debugger que se va a utilizar, para la simulación se utilizará MPLAB SIM, (Figura 250):

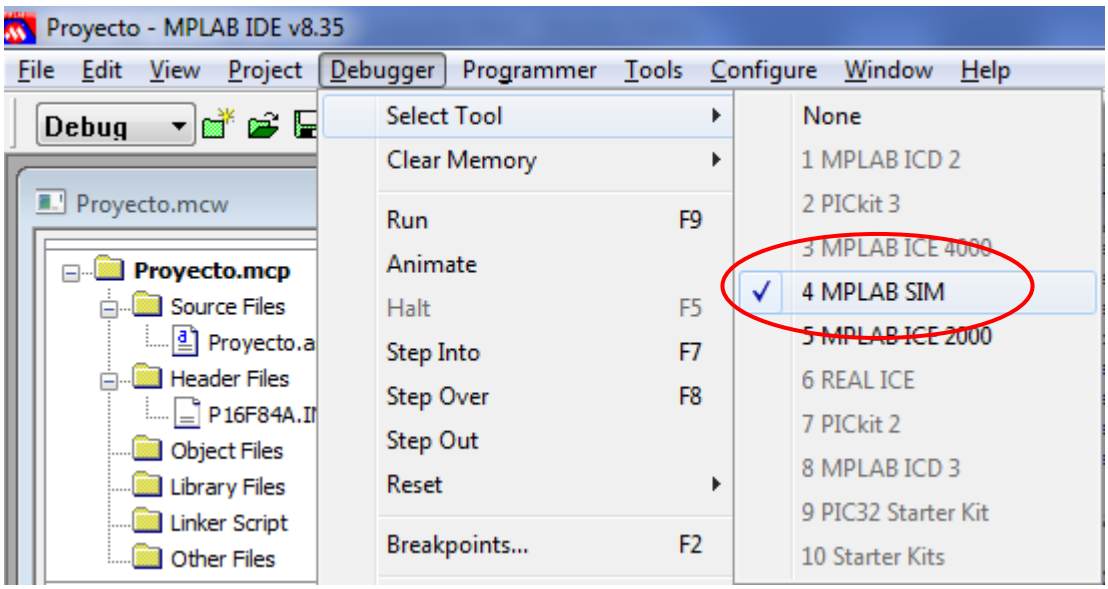


Figura 250. *Selección del debugger.*

Para comenzar con la práctica, una vez creado el nuevo proyecto, desde el menú file pulsamos en new o directamente ctrl+n. También se puede añadir un archivo existente.(Figura 251):

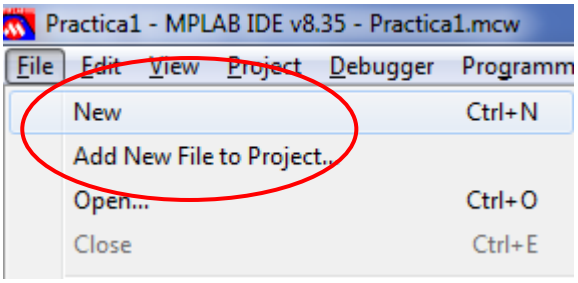


Figura 251. *Añadir un archivo al proyecto.*

Así se abrirá la ventana del editor (Figura 252):

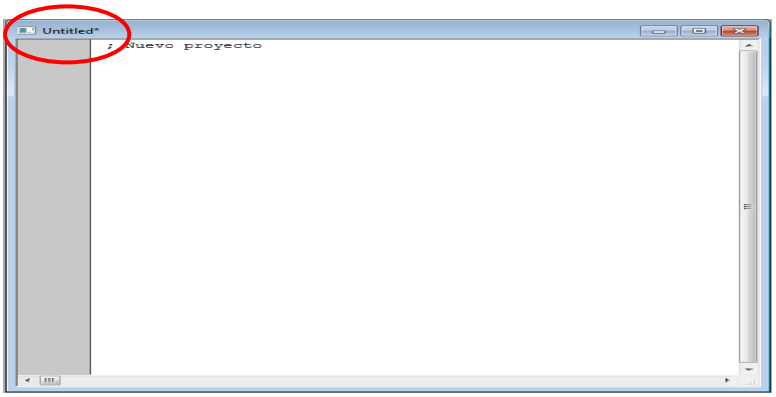


Figura 252. *Nuevo archivo.*

Sin embargo, aunque se comience a escribir el código, no está asociado al proyecto, para ello se debe primero grabarlo desde el menú *file--> save* con el nombre del programa y con extensión asm.(Figura 253):

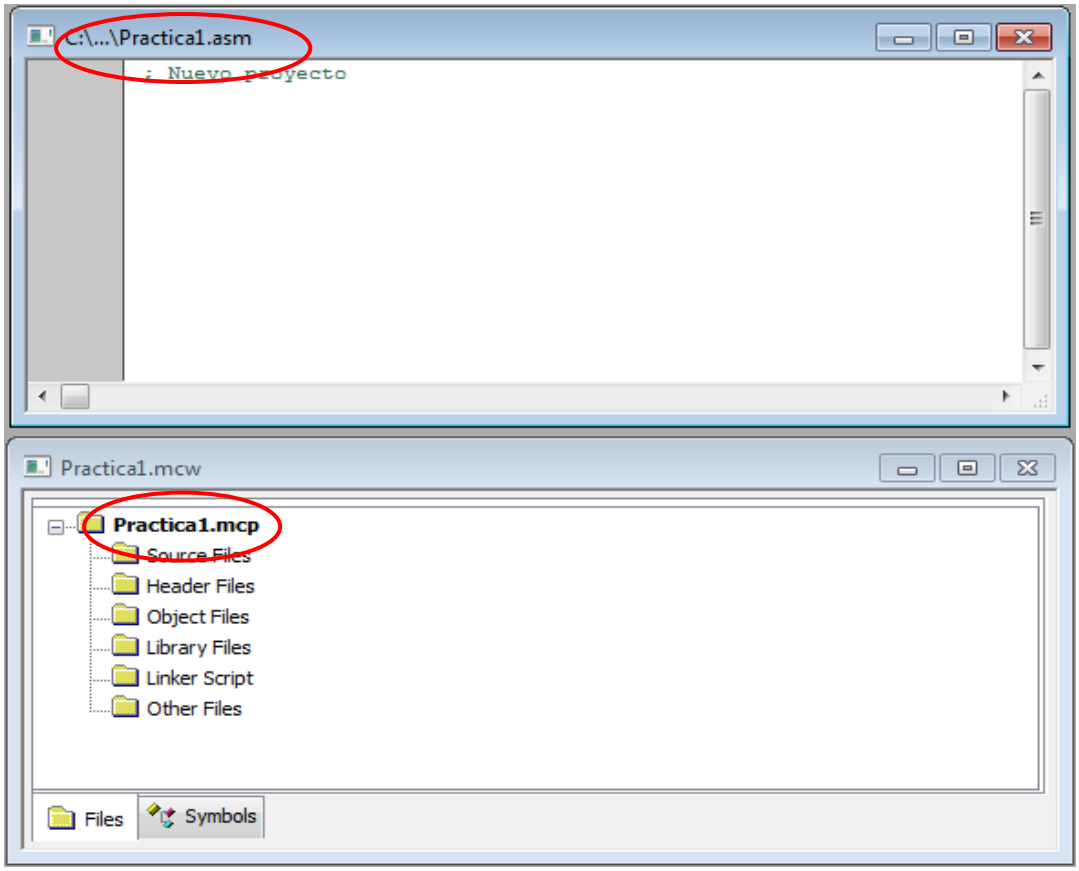


Figura 253. *Adjuntar el archivo al proyecto.*

Y se añade al proyecto bien desde *file* --> Add new file to proyect, o bien pulsando el botón derecho en Source Files --> Add file en la ventana Project,(Figura 254):

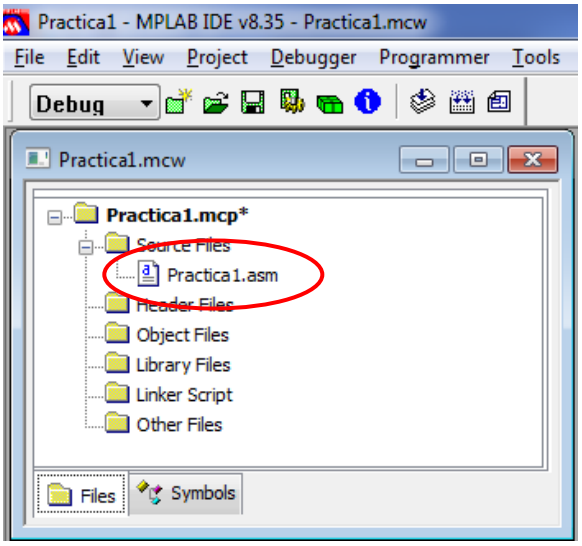


Figura 254. *Fin de añadir archivo al proyecto.*

Se añade el micro utilizado en la ventana de proyectos, pulsamos con el botón derecho en Header Files --> Add file y selecciona el PIC con el que se va a realizar el proyecto. Este archivo identifica los registros principales con nombres, facilitando la programación.(Figura 255)

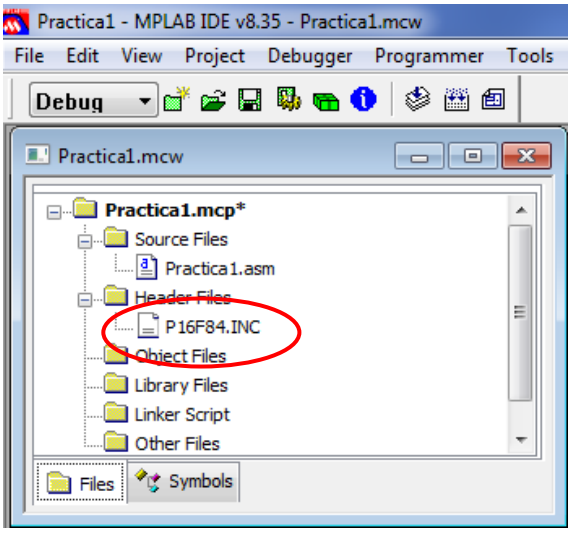


Figura 255. *Adjuntar PIC.*

1.2.4.2.5- Editor de programa

Es el lugar donde se escribe el código de programa, (Figura 256):

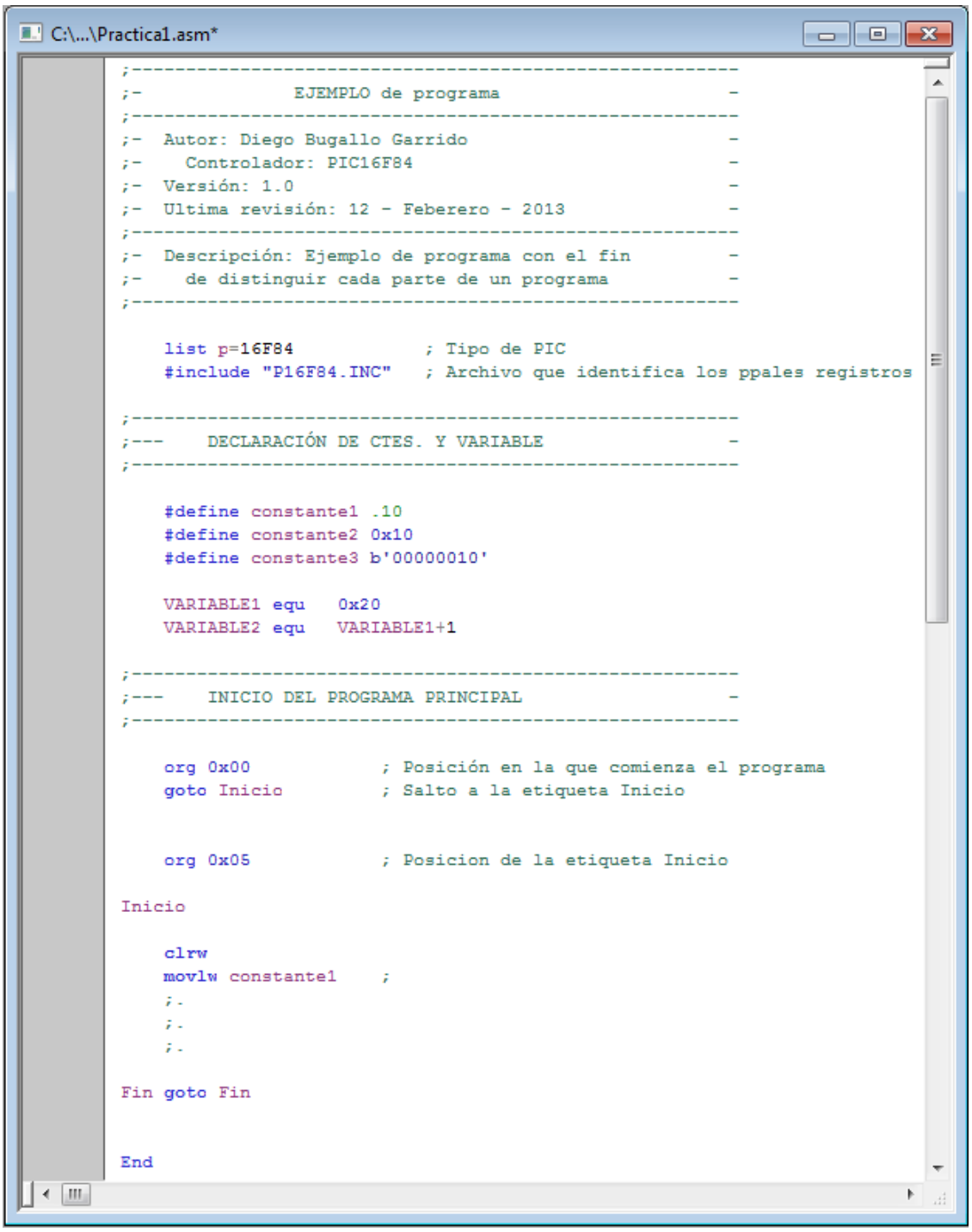


Figura 256. *Editor de programa.*

Cada línea de código sigue la siguiente estructura,(Figura 257):

```
Inicio  movlw   0x01        ; W=1h
        movwf   REGISTRO    ; REGISTRO=W
        movlw   dato        ; W=dato
```

Figura 257. *Estructura de programación.* [1]

- *Etiquetas:*
  - ★ Se sitúan en la primera columna.
  - ★ Solamente contienen caracteres alfanuméricos y los caracteres " _ " o " ? "
  - ★ El máximo número de caracteres es de 31.
- *Operación*:
  - ★ Se sitúan las instrucciones.
- *Operandos*:
  - ★ Puede contener un dato, una dirección de memoria.
- *Comentarios*:
  - ★ Precedidos del carácter " ; " indica al compilador que ignore lo que se encuentra a la derecha del punto y coma.

Los colores empleados por el simulador son los siguientes:

- *Verde*: Comentarios o un valor en decimal
- *Azul*: Instrucción o un valor en hexadecimal o binario
- *Rosa:* Nombres de etiqueta, registros o constantes. Cuando se escribe mal una instrucción está aparecerá de este color ya que lo considera como uno de los tres tipos anteriores.

- *Negro*: Indica un numero, solamente aparece este color cuando al escribir un número no le indicamos el tipo de número que es (binario, hexadecimal, etc.)

1.2.4.2.6- El compilador

Su función es traducir el código en ensamblador a un código que pueda interpretar el microcontrolador denominado código maquina, se puede compilar desde el menú proyect --> make o proyect --> build all, pulsado la tecla F10 o desde su correspondiente icono. (Figura 258):

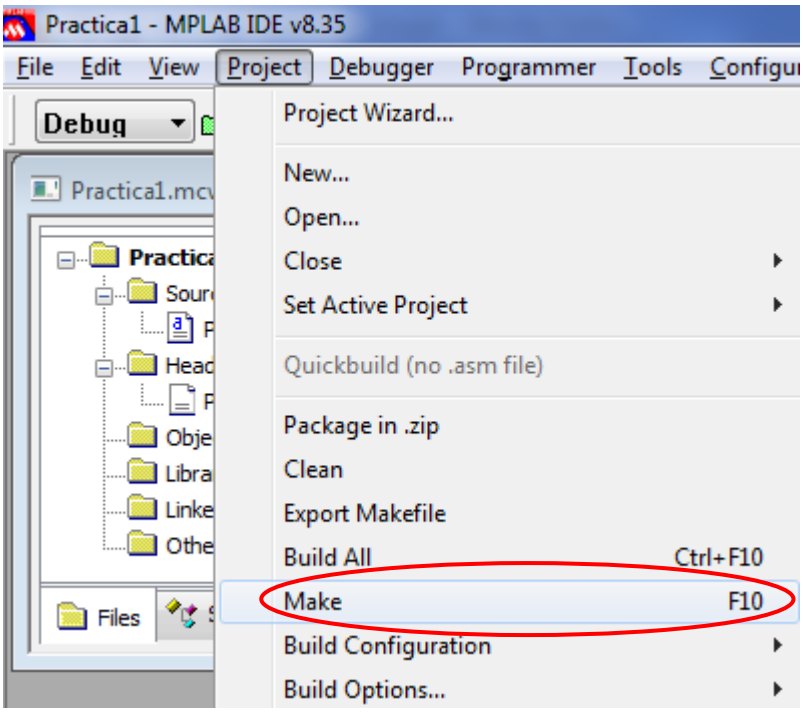


Figura 258. *Selección del compilador.*

En la ventana output se observa el resultado de la compilación, si contiene errores aparecerá un mensaje similar al siguiente, (Figura 259):

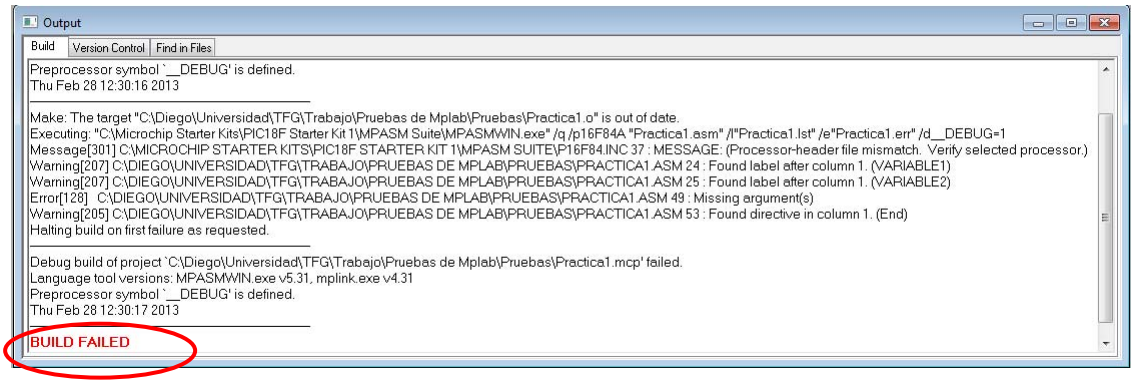


Figura 259. *Compilación fallida.*

Se indica que la compilación no ha satisfactoria, indicando el tipo de error, la línea donde se produce el error y una breve descripción del error. Si esto ocurre se debe revisar el código hasta solucionar todos los errores y sus derivados al cambiar el programa.

Una vez corregidos los errores aparecerá un mensaje similar al siguiente,(Figura 260):

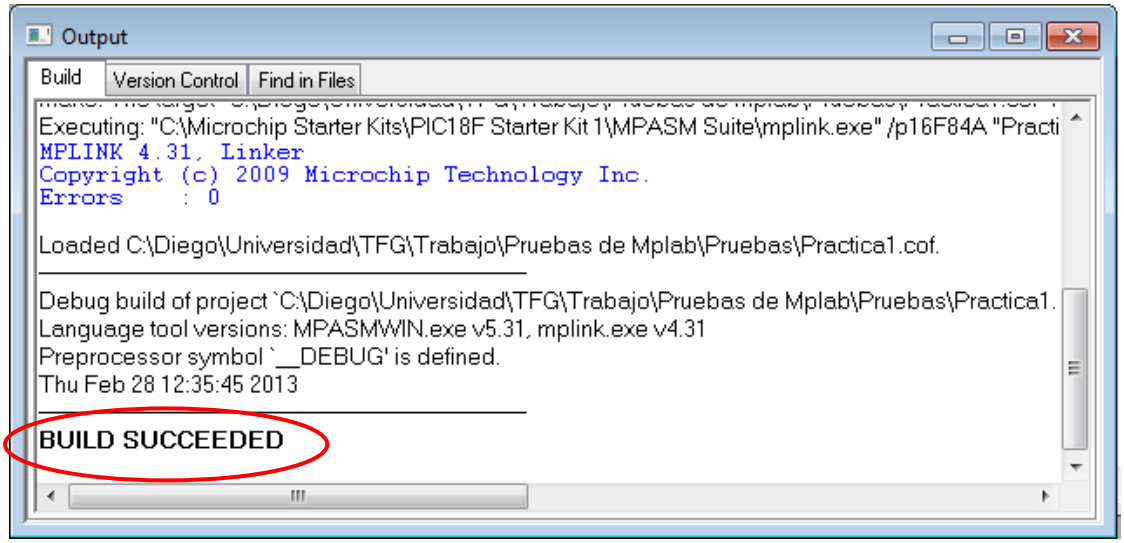


Figura 260. *Compilación correcta.*

Indicando una compilación correcta, lo que generará el archivo de extensión .hex, listo para ser interpretado por el microcontrolador.

Siempre que se modifique al menos un carácter del código es necesario volver a compilar.

1.2.4.2.7- La simulación

Una vez que se ha compilado correctamente, el siguiente paso es simular el programa, con el objetivo de observar si dicho programa funciona acorde a las necesidades del programador. Es probable que el programa no tenga ningún error en el lenguaje" sin embargo no funciona de la manera deseada.

Para poder simular es necesario haber seleccionado el simulador, explicado anteriormente en la figura 250. Una vez seleccionado se describe a continuación las distintas acciones posibles:

 Ejecución continua del programa a tiempo real

 Pausa

 Ejecución continua del programa aumentando notablemente el tiempo de ejecución de cada instrucción.

 Ejecución paso a paso

 Ejecución paso a paso realizando las subrutinas y bucles como un paso

 Estando dentro de un bucle o subrutina, ejecutar hasta la primera instrucción fuera del mismo.

1.2.4.2.7.1.- Ventanas de la simulación

Para poder observar los distintos resultados de los programas, existen distintas ventanas, para activar dichas ventanas desde el menú view,(Figura 261):

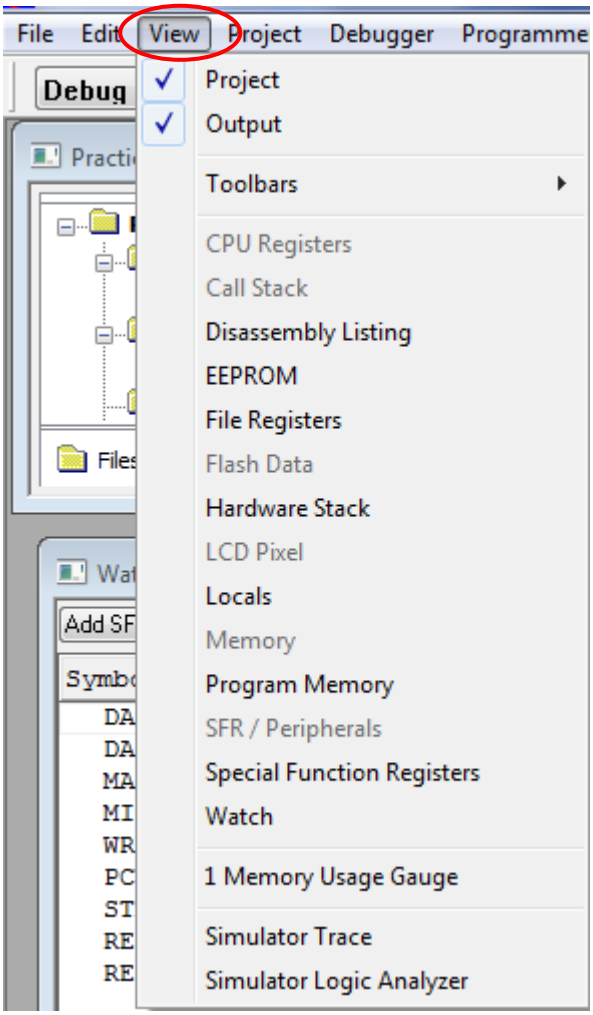


Figura 261. *Ventanas disponibles.*

A continuación se muestran las ventanas más importantes:

- *EEPROM:*

Esta ventana estructura el EEPROM del PIC en forma matricial desde la dirección 0x00h hasta la dirección 0x3Fh, inicialmente todas las direcciones contienen el valor 0xFF,(Figura 262):

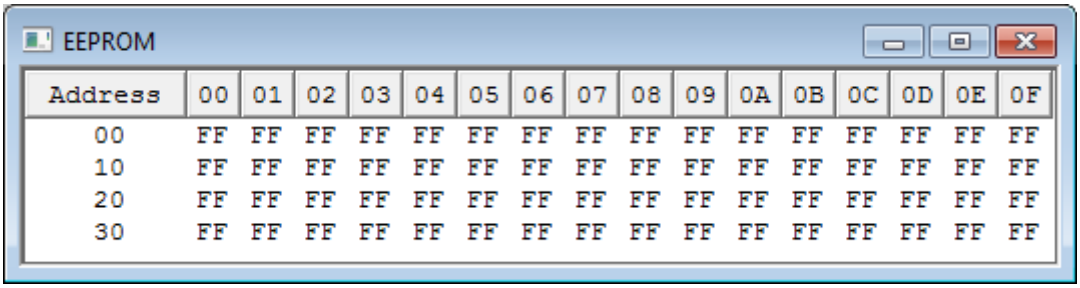


Figura 262. *EEPROM vacía.*

En el momento en que se escriba en la eeprom, cambiaría el contenido de la dirección modificada,(Figura 263):

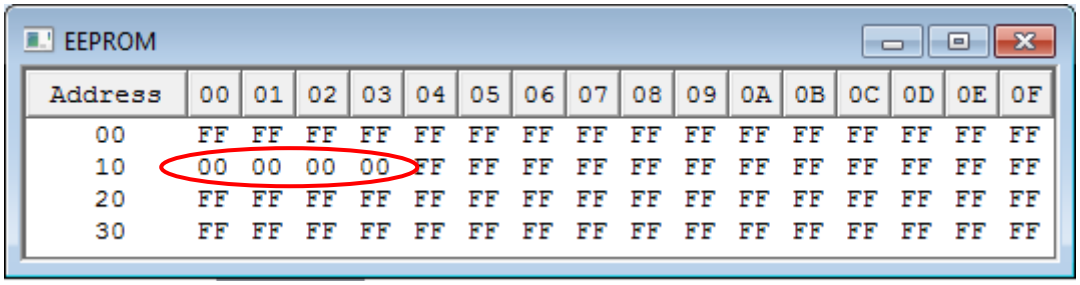
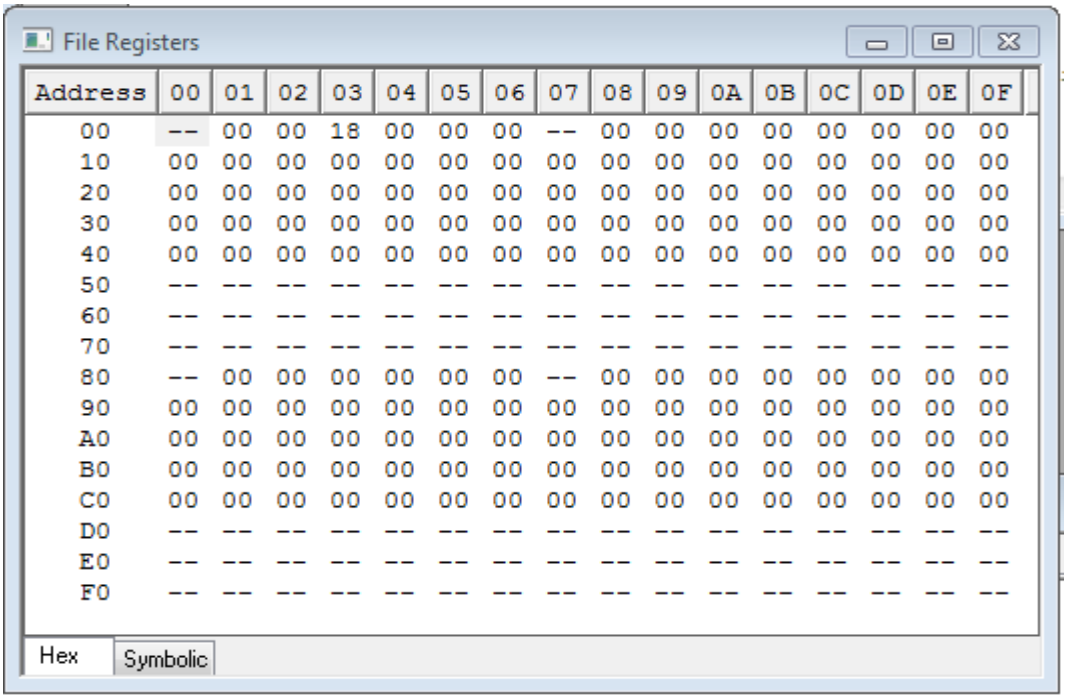


Figura 263. *EEPROM llena.*

- *File Register*

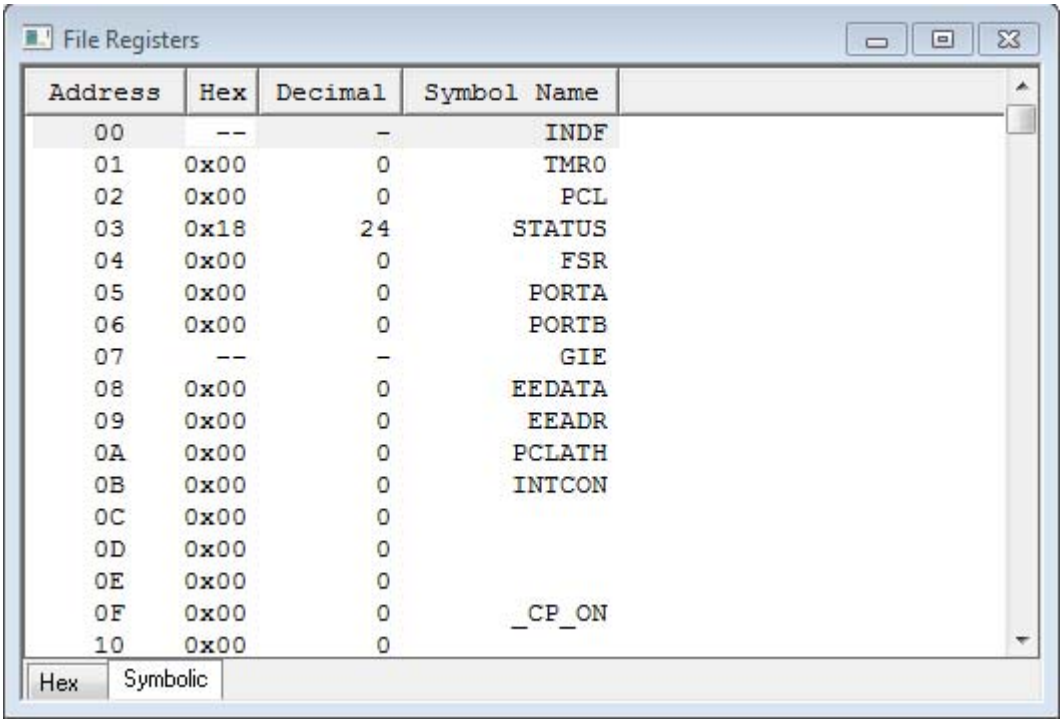
En esta ventana se muestra la memoria de datos del PIC, en el se encuentran todos los registros, es una memoria tipo RAM, se puede observar de forma matricial,(Figura 264):



Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	--	00	00	18	00	00	00	--	00	00	00	00	00	00	00	00
10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
50	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
60	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
70	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
80	--	00	00	00	00	00	00	--	00	00	00	00	00	00	00	00
90	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
D0	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
E0	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
F0	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Figura 264. *Memoria RAM.*

O de forma simbólica,(Figura 265):



Address	Hex	Decimal	Symbol Name
00	--	-	INDF
01	0x00	0	TMR0
02	0x00	0	PCL
03	0x18	24	STATUS
04	0x00	0	FSR
05	0x00	0	PORTA
06	0x00	0	PORTB
07	--	-	GIE
08	0x00	0	EEDATA
09	0x00	0	EEADR
0A	0x00	0	PCLATH
0B	0x00	0	INTCON
0C	0x00	0	
0D	0x00	0	
0E	0x00	0	
0F	0x00	0	_CP_ON
10	0x00	0	

Figura 265. *Memoria RAM, simbólica.*

- *Hardware Stack*

Esta ventana nos indica en el nivel o subnivel de programa en el que nos encontramos y almacena direcciones de retorno (**Practica 2**), en el siguiente caso estaríamos en el programa principal y sin ninguna subrutina ejecutada,(Figura 266):

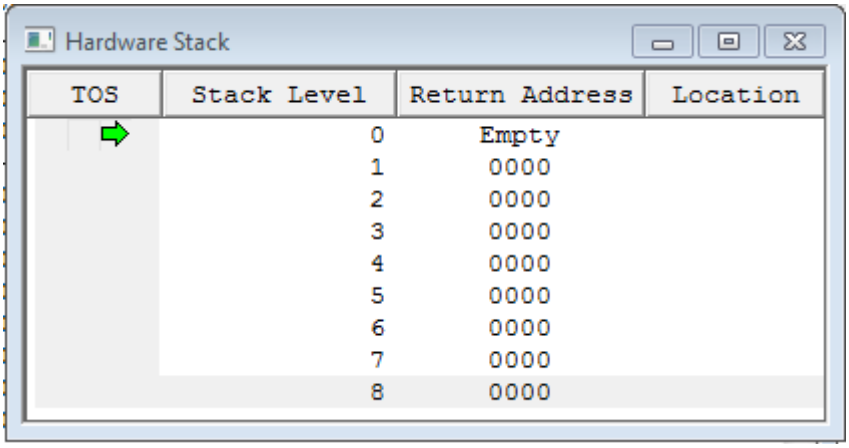


Figura 266. *Venta de pila vacía.*

En este ejemplo el programa se encontraría dentro de una subrutina de *nivel 2*,(Figura 267):

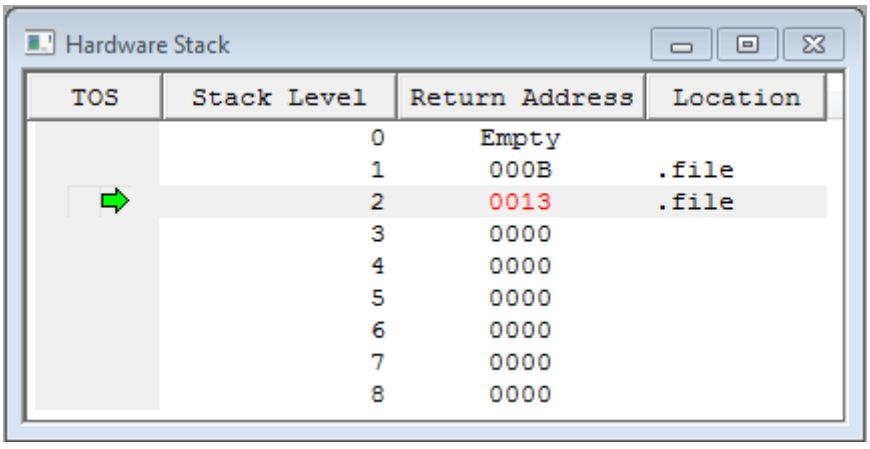


Figura 267. *Pila con varias subrutinas.*

- *Special Function Registers*

Son los registros más importantes, que se encuentran en la memoria de datos, estos registros son los encargados de gobernar el PIC y de indicar el estado del mismo, aunque también se pueden observar desde la ventana FILE REGISTER.(Figura 268):

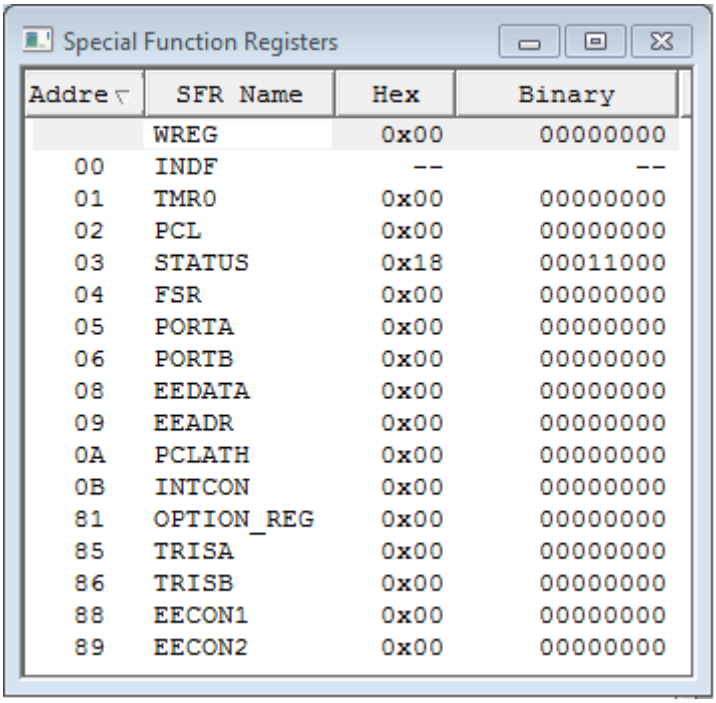


Figura 268. *Ventana de los registro más importantes.*

Haciendo click derecho dentro de la ventana se activa,(Figura 37):

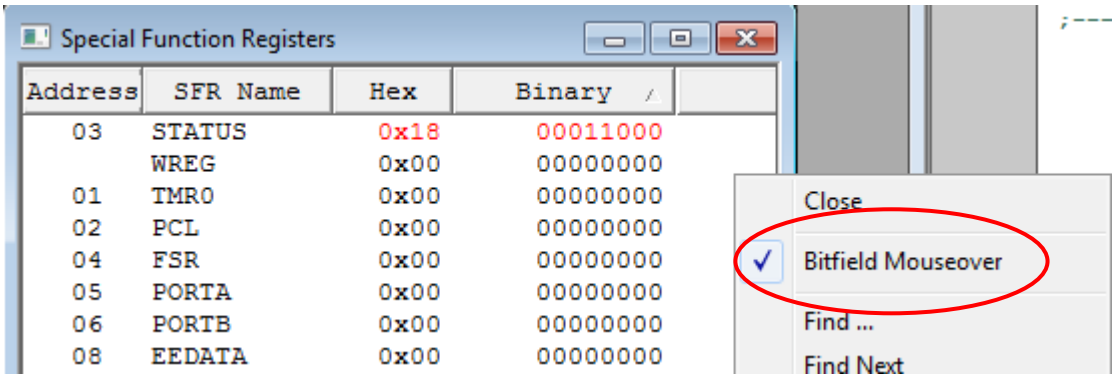


Figura 269. *Desglosar registro en bits.*

Que nos permite ver que es cada bit si pasamos por encima con el ratón,(Figura 270):

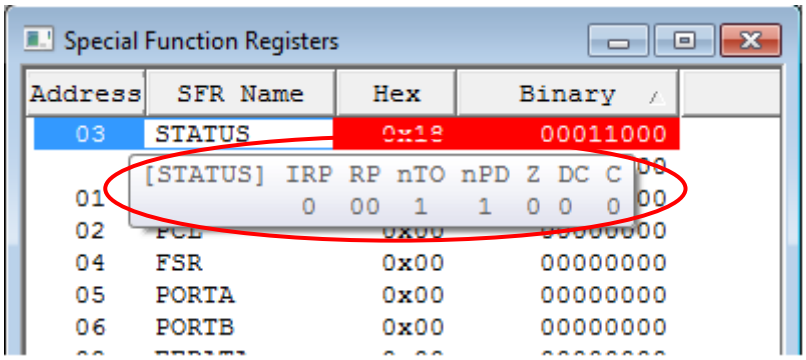


Figura 270. *Contenido de los registros.*

- *Watch*

Con esta ventana seleccionamos los registros más importantes para un programa concreto a controlar, también se pueden observar desde la ventana file register.(Figura 271):

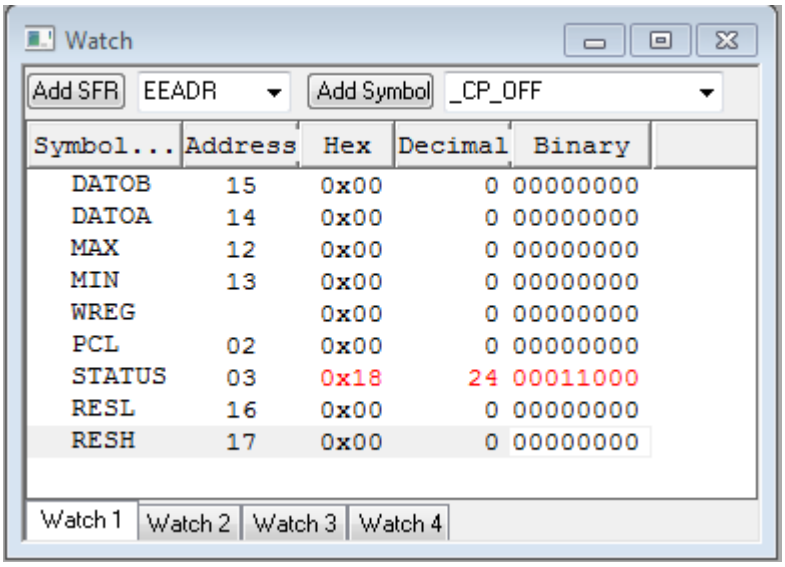


Figura 271. *Ventana Watch.*

Con el botón derecho del ratón se le indica los distintos valores de cada registro que queremos observar,(Figura 272):

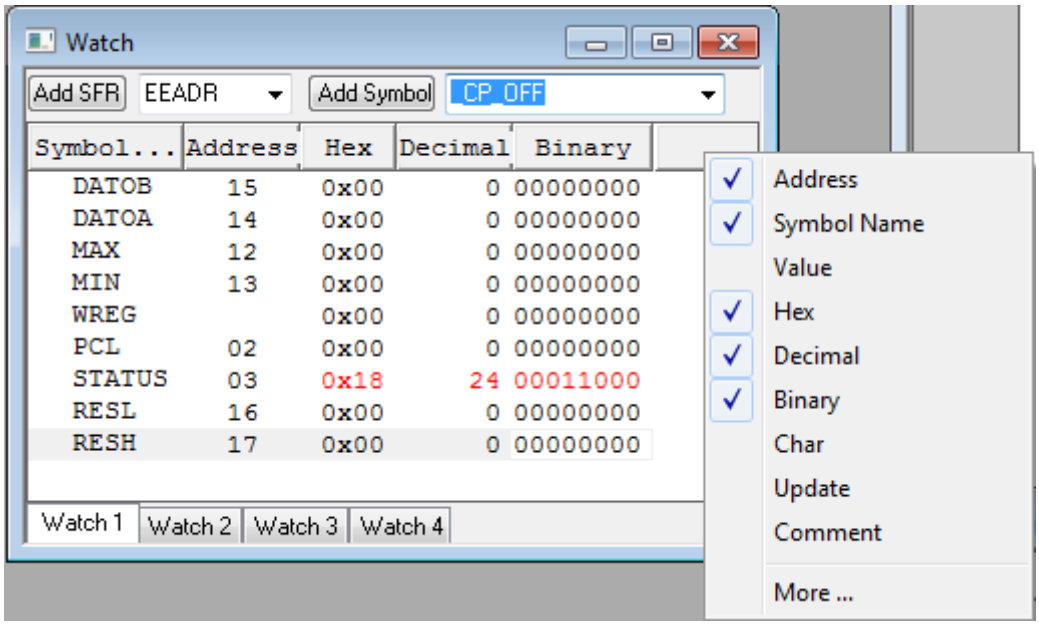


Figura 272. *Pestañas disponibles.*

↳ *Simulación de entradas:*

Una forma de simular entradas es el uso del Stimulus,(Figura 273):

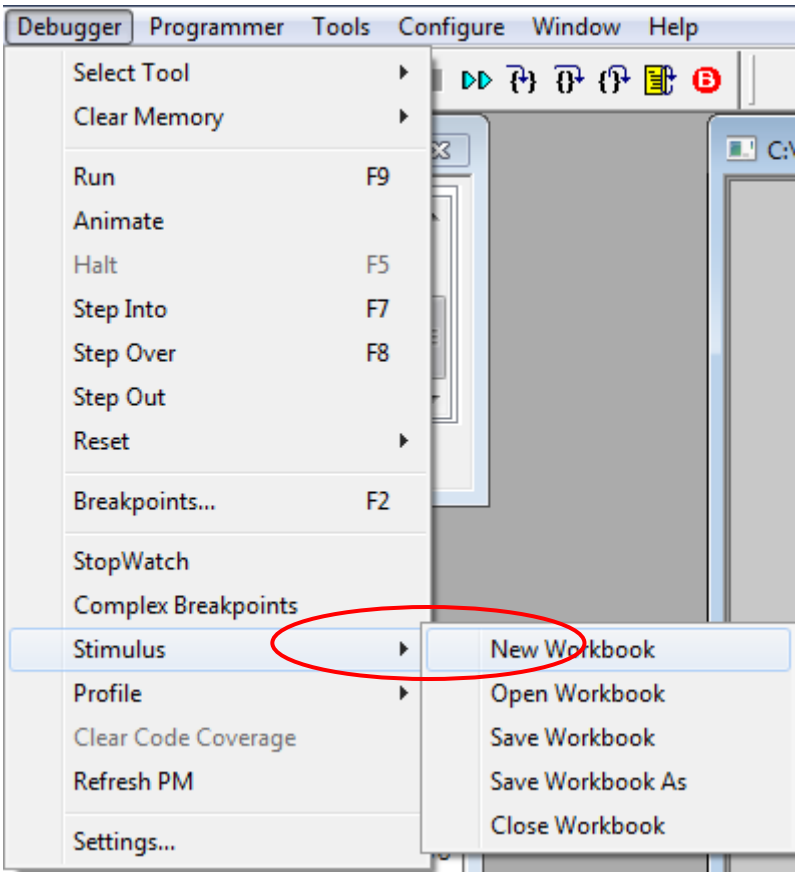


Figura 273. *Selección de la ventana Stimulus.*

En el cual vamos colocando las entradas deseadas, el tipo de entrada (pulsador, interruptor, etc.) Y cada vez que se quiere activar la entrada se debe hacer un click en la columna fire,(Figura 274):

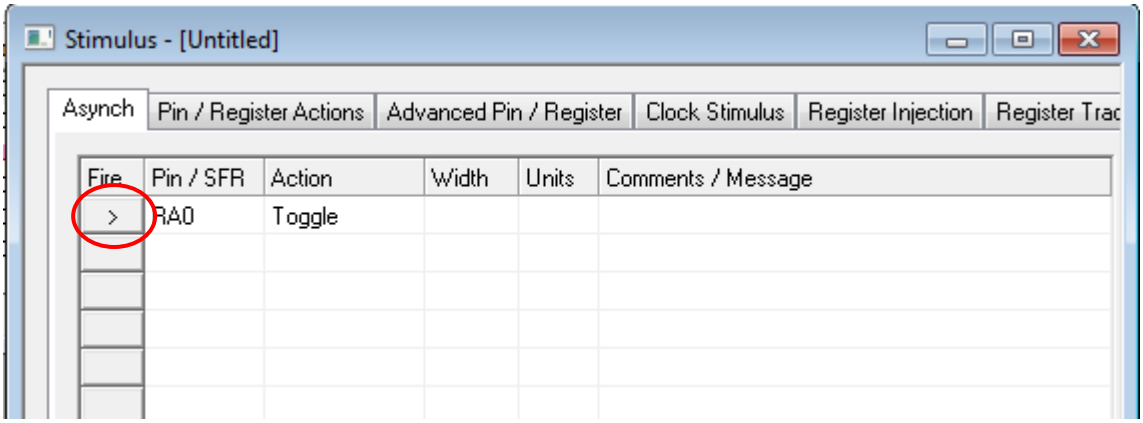


Figura 274. *Ventana Stimulus.*

1.2.4.3.- Instrucciones

Todas las instrucciones soportadas por el PIC16F886, empleadas en la programación en ensamblador se muestran a continuación (Figura 2765):

TABLE 15-2: PIC16F882/883/884/886/887 INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb	LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS							
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff		C, DC, Z	1, 2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff		Z	1, 2
CLRF	f	Clear f	1	00 0001 lfff ffff		Z	2
CLRW	–	Clear W	1	00 0001 0xxx xxxx		Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff		Z	1, 2
DECF	f, d	Decrement f	1	00 0011 dfff ffff		Z	1, 2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff			1, 2, 3
INCF	f, d	Increment f	1	00 1010 dfff ffff		Z	1, 2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff			1, 2, 3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff		Z	1, 2
MOVF	f, d	Move f	1	00 1000 dfff ffff		Z	1, 2
MOVWF	f	Move W to f	1	00 0000 lfff ffff			
NOP	–	No Operation	1	00 0000 0xx0 0000			
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff		C	1, 2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff		C	1, 2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff		C, DC, Z	1, 2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff			1, 2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff		Z	1, 2
BIT-ORIENTED FILE REGISTER OPERATIONS							
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff			1, 2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff			1, 2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff			3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff			3
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	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF GPIO, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 module.

3: If the Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

Figura 275. *Instrucciones del PIC16F886.*[7]

1.3.- REFERENCIAS

[1] EMILIO TOBOSO CALVO, Dispositivos lógicos microprogramables, Versión 3.2 , 2013 [en línea] <http://perso.wanadoo.es/pictob/indicemicroprg.htm>

[2] HD44780U (LCD-II) datasheet, HITACHI, Ltd, 1998

[3] JOEL OSWALDO CAMPOS PÉREZ, Curso básico de PIC tablas, Inventrónica, 23 Marzo 2013 [en línea] < http://es.scribd.com/doc/22595158/Crear-tablas-en-ensamblador-para-PIC >.

[4] JOSE MARIA ANGULO, Microcontroladores PIC, MCGRAW-HILL / INTERAMERICANA DE ESPAÑA, S.A., 2007

[5] MIKEL ETXEBARRIA, Laboratorio USB PIC'School: Manual de usuario con tutorial y ejemplos para PIC16F88X, INGENIERIA DE MICROSISTEMAS PROGRAMADOS S.L.,2010.

[6] MILAN VERLE, PIC Microcontrollers - Programming in C, MIKROELEKTRONIKA; 1st edition (2009) [en línea] < http://www.mikroe.com/products/view/285/book-pic-microcontrollers-programming-in-c/ >

[7] PIC16F882/883/884/886/887 Data Sheet, MICROCHIP TECHNOLOGY INC. 2007

[8] RAFAEL ARANDA AMEZCUA, Introducción a la programación de los microcontroladores 8051 (MCS-51), 2013 [en línea] < http://www.alciro.org/alciro/microcontroladores-8051_24 >.