INSTITUTO TECNOLÓGICO SUPERIOR AERONÁUTICO

CARRERA DE AVIÓNICA

"DISEÑO Y CONSTRUCCIÓN DE UN MARCADOR PROGRAMABLE DE TIEMPO PARA EL CONTROL DE HORAS ACADÉMICAS PARA EL COLEGIO JAN AMÓS COMENIUS DE LA CIUDAD DE LATACUNGA"

POR:

BERMEO HEREDIA FAUSTO FERNANDO

Proyecto de Grado presentado como requisito parcial para la obtención del Título de:

TECNÓLOGO EN AVIÓNICA

2007

CERTIFICACIÓN

Certifico que el presente trabajo fue realizado en su totalidad por la Sr. Bermeo Heredia Fausto Bermeo, como requerimiento parcial a la obtención de TECNÓLOGO EN AVIÓNICA.

_____

Ing. Marco Navas

Latacunga a 26 de febrero de 2007.

DEDICATORIA

El presente trabajo lo dedico a mi familia que en buenos y malos momentos me supieron apoyar, a todas las personas que con su respaldo incondicional supieron guiarme, y darme la fortaleza para seguir creyendo en la educación y especialmente lo dedico a las personas que me inculcaron el don de la superación que son mis padres.

Fausto Fernando Bermeo Heredia

AGRADECIMIENTO

Al finalizar esta etapa de mis estudios quiero dejar constancia de mi sincero agradecimiento al Instituto Tecnológico Superior Aeronáutico y a sus maestros que con sus sabios conocimientos impartidos ayudaron a culminar esta etapa.

Un sincero agradecimiento a la Ing. Martha Castellanos que gracias a su ayuda me dio la oportunidad para seguir con mis estudios y poder culminar esta etapa.

Un especial agradecimiento al Ing. Marco Navas quien me guió en el desarrollo del presente trabajo.

Y a todas aquellas personas que colaboraron incondicionalmente para la finalización de este trabajo.

Finalmente agradezco a Dios por su sabia bondad.

Fausto Fernando Bermeo Heredia

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El presente trabajo es un marcador de tiempo programable que se ha diseñado utilizando un microcontrolador PIC16F84, con una capacidad de guardar diez tiempos de programación en la memoria EEPROM que es una pequeña área de memoria de datos de lectura y escritura no volátil, gracias a la cual, un corte del suministro de la alimentación no ocasionará la pérdida de la información, que estará disponible al reinicializarse el programa marcador de tiempo.

Gracias a sus excelentes características: bajo precio, reducido consumo, pequeño tamaño, gran calidad, fiabilidad y abundancia de información, al PIC16F84 lo convierten en cómodo y rápido de utilizar en el marcador de tiempo programable.

El trabajo de investigación cuenta con un marco teórico el mismo que ha servido de apoyo para conocer con mayor claridad los conceptos básicos y modo de funcionamiento del PIC16F84.

El colegio Jan Amós Comenius en la actualidad dispone de un timbre eléctrico para la realización del cambio de horas académicas, lo cual induce a errores involuntarios, producidos por razones como olvidos, relojes no sincronizados de las personas que tocan el timbre, etc. Ante esta situación se crea la necesidad de automatizar el cambio de horas mediante un sistema electrónico, el mismo que se pone en consideración en el presente trabajo.

Con la propuesta de implementar este marcador de tiempo programable en el Colegio particular bilingüe “Jan Amós Comenius” de la ciudad de Latacunga automatizaríamos el control de horas académicas, dando paso a la implementación de un sistema que permita realizar esta actividad de manera autónoma y con la gran ventaja de cada cambio de hora se realice con total exactitud.

Con el marcador de tiempo programable se evitará que cada maestro se tome un tiempo adicional de su materia esto para evitar los inconvenientes con el siguiente maestro que viene a impartir su hora clase.

INTRODUCCIÓN

En la actualidad nos desarrollamos en un mundo competitivo en el cual nos exige siempre estar a la par del desarrollo tecnológico, en donde se deberá estar cambiando constantemente, caso contrario se deja de ser competitivo.

Los microcontroladores se utilizan en circuitos electrónicos comerciales desde hace muchos años de forma masiva, debido a que permiten reducir el tamaño y el precio de los equipos. Un ejemplo de éstos son los teléfonos móviles, las cámaras de video, la transmisión por satélite y otros.

En los últimos años se ha facilitado enormemente el trabajo con los microcontroladores al bajar los precios, aumentar las prestaciones y simplificar los montajes.

Diversos fabricantes ofrecen amplias gamas de microcontroladores para todas las necesidades. Pero sin duda, hoy en día los microcontroladores más aceptados para diseños son los microcontroladores PIC.

Entre los microcontroladores PIC se destaca el PIC16F84 cuya simplicidad, prestaciones, facilidad de uso y precio lo han convertido en el más popular de los microcontroladores, siendo un chip ordinario de 18 patillas, cuya estructura contiene mucha de la tecnología que se necesita conocer para entender los sistemas de control con microprocesadores.

PLANTEAMIENTO DEL PROBLEMA:

El Colegio particular bilingüe “Jan Amós Comenius” de la ciudad de Latacunga crea la necesidad de automatizar el control de inicio y fin de las horas académicas, dando esto lugar a la implementación de un sistema que permita realizar esta actividad de modo que no se requiera la intervención humana, y que además sea de una precisión adecuada.

Por otro lado es necesario que el sistema en mención tenga la posibilidad de modificar los inicios y fines de hora programados sin la intervención de técnicos en sistemas electrónicos o especialistas en programación.

Surge entonces la posibilidad de llevar a cabo este proyecto mediante la realización de un proyecto de grado que se espera sea beneficioso para todos los involucrados en el mismo.

JUSTIFICACIÓN:

En la actualidad se hace accesible inclusive en las tareas más cotidianas la automatización de ciertas actividades y procesos, debido a los bajos costos especialmente de los circuitos integrados digitales, y por supuesto la alta tecnología con la que estos se construyen.

Hace apenas pocos años no era posible pensar en un pequeño microcomputador casi completo e integrado monolíticamente en un solo circuito de 18 patillas, y que además1 disponga de las mejores características de varios de los complejos sistemas computacionales de entonces, y a costos muy accesibles.

Con estas facilidades a nuestro alcance, resulta plenamente justificado liberar a una persona de la rutinaria tarea de mirar el reloj para avisar mediante un timbrado del inicio o culminación de una hora académica.

Con la automatización de esta actividad además se logrará mejorar la precisión con la que se realizarán estos avisos y de este modo no perjudicar el tiempo que tienen los alumnos para el estudio, así como su tiempo libre (recreo).

OBJETIVOS:

OBJETIVO GENERAL:

Implementar un sistema automático para el control de las horas académicas en el Colegio particular bilingüe “Jan Amós Comenius”.

OBJETIVOS ESPECÍFICOS:

- Solucionar un problema concreto en la mencionada Institución Educativa.
- Familiarizar con el microcontrolador PIC16F84.
- Aprender a usar los microcontroladores en la implementación de pequeños sistemas de control automático.
- Conseguir tiempos de programación en el cual se pueda ingresar datos de acuerdo a la necesidad.
- Efectuar un sistema audible que permita a los docentes informar el inicio y fin de su hora clase.
- Elaborar un manual de operación que permita usar el sistema desarrollado a una persona que no tenga conocimiento de electrónica y concretamente de sistemas digitales.

ALCANCE:

Con el presente proyecto se pretende solucionar un problema concreto como es la automatización del control de las horas académicas en el Colegio particular bilingüe “Jan Amós Comenius” mediante la utilización de un microcontrolador como es el PIC16F84.

El presente trabajo servirá como fuente de información y consulta para los futuros alumnos de la carrera de Aviónica, así como de todos los interesados en el tema.

Además se elaborará un manual de operación para que cualquier persona pueda usarlo.

CAPITULO I

MARCO TEÓRICO

1.1. REVISIÓN DE LOS MICROCONTROLADORES

Los microcontroladores están conquistando el mundo. Están presentes en el trabajo, en la casa y en la vida, en general. Se pueden encontrar controlando el funcionamiento de los ratones y teclados de los computadores, en los teléfonos, en los hornos microondas y los televisores del hogar. Estos diminutos computadores, que gobernarán la mayor parte de los aparatos se fabrican y usamos los humanos.

Un microcontrolador es un circuito integrado de alta escala de integración que incorpora la mayor parte de los elementos que configuran un controlador.

1.1.1. MICROCONTROLADORES PIC

Un microcontrolador es un circuito integrado programable que contiene todos los componentes necesarios para controlar el funcionamiento de una tarea determinada, como un marcador de tiempo, un sistema de alarmas, etc.

El microcontrolador consta de una memoria donde almacena el programa que gobierna el funcionamiento del mismo ya una vez programado solo sirve para realizar la tarea asignada.

Cada tipo de microcontrolador sirve para una serie de casos y es el diseñador del sistema quien debe decidir cual es el microcontrolador más idóneo para cada uso.

Los PIC son una familia de microcontroladores que han tenido una gran aceptación y desarrollo en los últimos años gracias a que sus buenas características, bajo precio, reducido consumo, pequeño tamaño, gran calidad,

fiabilidad y abundancia de información, lo convierten en muy fácil, cómodo y rápido de utilizar.

La frecuencia de trabajo del microcontrolador puede llegar hasta los 20 MHz con su nueva versión PIC 16F84A-20.

1.1.2. ALIMENTACIÓN DE UN PIC16F84

Un microcontrolador PIC16F84 se alimenta con 5 voltios aplicados entre los pines **VDD** y **Vss** que son respectivamente, la alimentación y la masa del chip.

El consumo de corriente para el funcionamiento del microcontrolador depende de la tensión de alimentación, de la frecuencia de trabajo y de la carga que soporta sus salidas, siendo del orden de unos pocos miliamperios.

El circuito de alimentación del microcontrolador debe tratarse como el de cualquier otro dispositivo digital, debiendo conectar un condensador de desacoplo de unos 100 nF lo más cerca posible de los pines de alimentación.

1.1.3. PUERTOS DE ENTRADA/SALIDA

El microcontrolador se comunica con el mundo exterior a través de los puertos.

Los puertos se pueden configurar como entrada para recibir datos o como salida para gobernar dispositivos externos (figura 1.1)

El PIC16F84 tiene dos puertos.

- El Puerto A con 5 líneas, pines RA0 a RA4.
- El Puerto B con 8 líneas, pines RB0 a RB7.

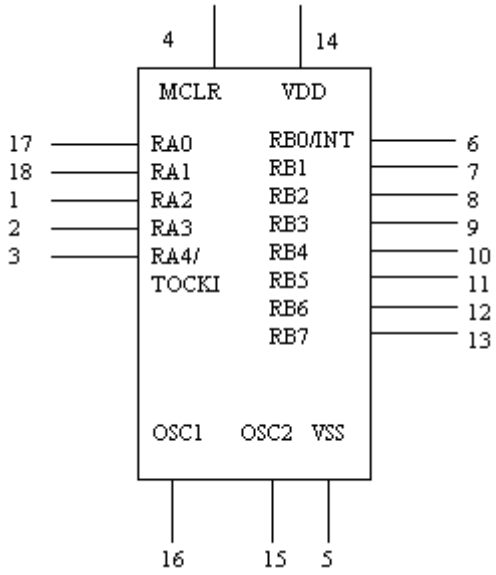


Figura 1.1 Distribución de pines en el PIC16F84

Cada línea puede ser configurada como entrada o salida, independiente una de otra, según se programe.

Las líneas son capaces de entregar niveles TTL cuando la tensión de alimentación aplicada en VDD es de 5V. La máxima capacidad de corriente de cada una de ellas es:

25 mA, cuando el pin está a nivel bajo, es decir, cuando consume corriente. Sin embargo, las sumas de las intensidades por las 5 líneas del Puerto A no puede exceder de 80 mA, ni la suma de las 8 líneas del Puerto B puede exceder de 150mA.

20 mA, cuando el pin está a nivel alto, es decir, cuando proporciona corriente. Sin embargo, la suma de la intensidad por las 5 líneas del Puerto A no puede exceder de 50 mA, ni la suma de los 8 líneas del Puerto B puede exceder de 100mA.

1.1.4. OSCILADOR

Todo microcontrolador requiere de un circuito que le indica la velocidad de trabajo a este se le conoce como oscilador o reloj.

Este oscilador genera una onda cuadrada de alta frecuencia que se utiliza como señal para sincronizar todas las operaciones del sistema, este circuito es simple pero de vital importancia para el buen funcionamiento del sistema.

Generalmente todos los componentes de reloj se encuentran integrados en el propio microcontrolador y tan solo se requiere unos pocos componentes externos, como un cristal de cuarzo o una red RC, para decir la frecuencia de trabajo.

Los pines **OSC1** y **OSC2** son las líneas utilizadas para conectar el oscilador, se permite cinco tipos de osciladores para definir la frecuencia de funcionamiento:

- **XT**. Cristal de cuarzo.
- **RC**. Oscilador con resistencia y condensador.
- **HS**. Cristal de alta velocidad.
- **LP**. Cristal para baja frecuencia y bajo consumo de potencia.
- **Externa**. Cuando se aplica una señal de reloj externa.

Para el proyecto a realizar se ha estimado conveniente utilizar el oscilador **XT**, por la facilidad de haberlo utilizado antes y está basado en el oscilador a cristal de cuarzo.

Es un oscilador estándar que permite una frecuencia de reloj muy estable comprendida entre 100KHz Y 4 MHz.

El condensador debe ir acompañado de dos condensadores entre 15 y 33 pF como se ilustra en la figura 1.2.

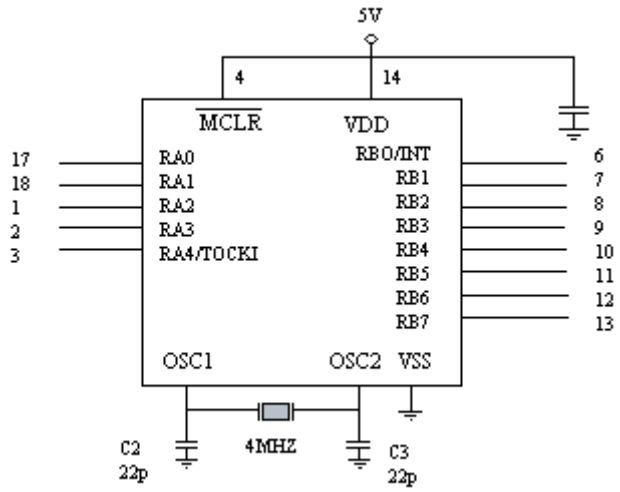


Figura 1.2 Configuración de un oscilador XT

Si se comprueba con un osciloscopio la señal en el pin **OSC2,** se debe visualizar un tren de pulsos de igual frecuencia que la del cristal utilizado.

1.2. CARACTERÍSTICAS DEL PIC16F84

A continuación se explicará las principales características del PIC16F84, donde ayudará a comprender de mejor manera estos dispositivos.

1.2.1. CARACTERÍSTICAS DE LA CPU RISC

- El juego de instrucciones tiene 35 instrucciones de una sola palabra.
- Todas las instrucciones duran un ciclo máquina, excepto las de salto que duran dos.
- Velocidad de operación:

  DC-20 MHz, para la frecuencia de reloj de entrada.

  DC-200 ns, para la duración del ciclo máquina.

- Frecuencia máxima de funcionamiento de 4 MHz para el PIC16F84A ó 20 MHz para el PIC16F84A.
- Memoria de programa tipo Flash de 1024 posiciones.
- Memoria RAM de datos de 68 bytes.
- Memoria EEPROM de datos de 64 bytes.

- Instrucciones con una longitud de 14 bits.
- Los datos tienen una longitud de 1 bytes (8bits).
- Dispone de 15 registros de funcionamiento específico.
- La pila tiene 8 niveles de profundidad.
- Dispone de cuatro fuentes de interrupción, las cuales pueden ser habilitadas o deshabilitadas independientemente por software:
    1. Externa por el pin RB0/INT.
    2. Por desbordamiento del timer 0.
    3. Por cambio en las líneas PORTB<7:4>.
    4. Por finalización de escritura de la memoria EEPROM de datos.

1.2.2. CARATERÍSTICAS DE LOS PERIFÉRICOS.

- Dispone de 13 líneas de entrada/ salida con control individual de dirección.
- Alta capacidad de corriente por terminal. Proporciona suficiente corriente para gobernar un LED.

    1. Consume 25 mA por pin cuando está a nivel bajo.
    2. Proporciona 20 mA por pin cuando está a nivel alto.

- Dispone de un Temporizador/ Contador de 8 bits (TMR0) con división de frecuencia programable.

1.2.3. CARACTERÍSTICAS ESPECIALES DEL MICROCONTROLADOR

- La memoria Flash de programa admite hasta 1.000 ciclos de borrado y escritura.
- La memoria EEPROM de datos admite hasta 1.000.000 de ciclos de borrado y escritura.
- Garantiza una retención de datos para la memoria EEPROM de datos superior a los 40 años.
- Se puede programar en el círculo vía serie de dos pines, ICSP (In Circuit Serial Programming)

- Dispone de un temporizador Watchdog (WDT) con su propio oscilador RC para un funcionamiento fiable.
- Protección de código de programa mediante la activación de un bit de protección.
- Modo de bajo consumo SLEEP.
- Tipo de oscilador seleccionable.

1.2.4. CARACTERÍSTICAS ELÉCTRICAS MÁXIMAS ADMISIBLES.

- Tensión de cualquier pin respecto de Vss
  (Excepto V_{DD}, MCLR y RA4) ---------- 0.3 V a (V_{DD}+0.3V)
- Tensión en V_{DD} respecto de Vss ---------- -0,3 a + 7,5V
- Tensión en MCLR respecto de Vss ---------- -0,3 a + 14V
- Tensión en RA4 respecto de Vss ---------- -0,3 a + 8,5V
- Potencia de disipación total ----------800 mW
- Máxima corriente por el pin Vss ---------- 150 mA
- Máxima corriente por el pin V_{DD} ---------- 100 mA
- Máxima corriente de salida en bajo por cualquier pin I/O ---------- 25 mA
- Máxima corriente de salida en alto por cualquier pin I/O ---------- 20 mA
- Máxima corriente de salida en bajo por el conjunto del Puerto A--- 80 mA
- Máxima corriente de salida en alto por el conjunto del Puerto A---- 50 mA
- Máxima corriente de salida en bajo por el conjunto del Puerto B- 150 mA
- Máxima corriente de salida en alto por el conjunto del Puerto B-- 100 mA

1.3. VENTAJAS DEL PIC16F84

Los productos que para su regulación incorporan un microcontrolador disponen de las siguientes ventajas:

1.3.1 AUMENTO DE PRESTACIONES

Un mayor control sobre un determinado elemento representa una mejora considerable en el mismo.

1.3.2 AUMENTO DE LA FIABILIDAD

Al reemplazar el microcontrolador por un elevado número de elementos disminuye el riesgo de averías y se precisan menos ajustes.

1.3.3 REDUCCIÓN DEL TAMAÑO EN EL PRODUCTO ACABADO

La integración del microcontrolador en un chip disminuye el volumen, la mano de obra y los stocks.

1.3.4 MAYOR FLEXIBILIDAD

Las características de control están programadas por lo que su modificación sólo necesita cambios en el programa de instrucciones.

El microcontrolador es en definitiva un circuito integrado que incluye todos los componentes de un computador. Debido a su reducido tamaño es posible montar el controlador en el propio dispositivo al que gobierna. En este caso el controlador recibe el nombre de controlador empotrado (embedded controller).

1.4 ORGANIZACIÓN DE LA MEMORIA

1.4.1 MEMORIA DE DATOS RAM.

En esta memoria se destinan a guardar las variables y los datos que se manejan en el programa, estos datos varían continuamente por lo que esta memoria es de escritura y lectura su memoria es de tipo volátil.

Cuenta con dos bancos de memoria, Banco 0 y Banco 1.

Esta memoria está dividida en dos partes:

1.4.1.1 REGISTRO DE FUNCIONES ESPECIALES SFR

Son registros que cumplen un propósito especial en el control del microcontrolador. Este registro está agrupado entre las direcciones 00h a 0Bh para el Banco 0 y entre las direcciones 80h hasta 8Bh para el Banco 1. Algunos de los registros SFR se encuentran duplicados en la misma dirección en los dos bancos, con el objetivo de simplificar su acceso.

1.4.1.2 REGISTRO DE PROPÓSITO GENERAL GPR.

Se puede usar para guardar los datos temporales del programa que se esté ejecutando. Tiene 68 posiciones de memoria, ya que sólo son operativas las del Banco 0, porque las del Banco 1 se mapean sobre el Banco 0. Es decir, cuando se apunta a un registro de propósito general del Banco 1, realmente se accede al mismo registro del Banco 0.

Para seleccionar el banco a acceder hay que configurar el bit 5 del registro STATUS. Con RP0 = 0 se accede al Banco 0 y con RP0 = 1 se accede al Banco 1. El Banco 0 es seleccionado automáticamente después de un reset.

1.4.2 MEMORIA EEPROM DE DATOS.

Es una pequeña área de memoria de datos de lectura y escritura no volátil, en el PIC16F84 dispone de una zona con 64 bytes de memoria EEPROM para almacenar datos que no se pierdan al desconectar la alimentación.

Al escribir en una posición de memoria ya ocupada, automáticamente se borra el contenido que había y se introduce el nuevo dato.

1.4.3 MEMORIA DE PROGRAMA

El microcontrolador está diseñado para que en su memoria de programa se almacenen todas las instrucciones del programa de control.

El programa a ejecutar siempre es el mismo, por tanto, debe estar grabado de forma permanente.

Esta característica de no volatilidad garantiza que la memoria mantenga su contenido aún sin alimentación, de forma que el programa no necesite volver a ser cargado en el sistema cada vez que se utilice.

La información contenida en esta memoria debe ser grabada previamente mediante un equipo físico denominado programador o grabador.

El PIC16F84A es un microcontrolador con un tipo memoria de programa no volátil denominado ROM flash, que permite una grabación muy sencilla, rápida y cómoda, lo que representa gran facilidad en el desarrollo de diseños.

Las memorias de programa del PIC16F84A tiene la capacidad de 1K (1024 posiciones) y está organizada en palabras de 14 bits.

Así pues, la memoria de programa comienza en la posición 000h (posición inicial de reset) y llega hasta la 3FFh como se indica en la figura 1.3.

El PIC16F84 admite unas 1000 grabaciones, y el fabricante garantiza que la información permanezca inalterable durante varias decenas de años.

1.4.4 EL CONTADOR DE PROGARAMA

Un programa está compuesto por instrucciones que generalmente se ejecutan de forma secuencial. En el PIC16F84A cada una de estas instrucciones ocupa una posición de memoria y programa.

El contador de programa PC es un registro interno que se utiliza para direccional las instrucciones del programa de control que están almacenadas en la memoria de programa como se observa en la figura 1.3.

El registro contiene la dirección de la aproxima instrucción y se incrementa automáticamente de manera que la secuencia natural de ejecución de programa es lineal.

El microcontrolador PIC16F84A dispone de un contador de programa que le permite direccional los 1K x 14 bits de memoria de programa implementada, desde la posición 000h hasta la 3FFh.

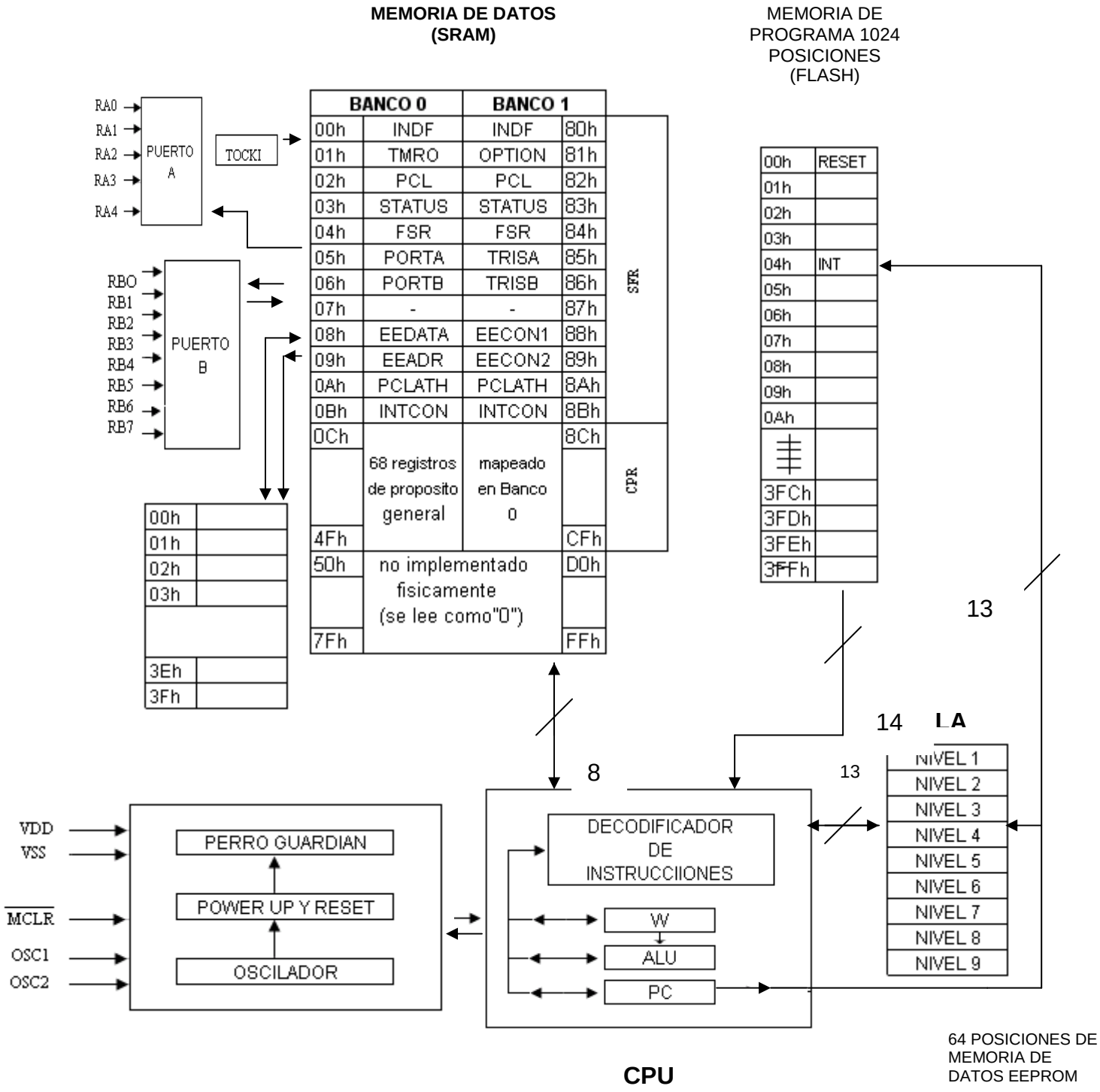


Figura 1.3 Arquitectura interna del PIC16F84

1.4.5 REGISTRO DEL SFR

En la tabla 1.1.Se detallan los registros de funciones especiales **SFR**

Tabla 1.1 Registro del SFR

Dirección	Nombre	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
BANCO 0									
00h	INDF	Registro utilizado en el direccionamiento indirecto (no es un registro físico)							
01h	TMR0	Timer / Contador de 8 bit							
02h	PCL	Registro con los 8 bit más bajos del Contador del Programa							
03h	STATUS	IRP	RP1	RP0	/TO	/PD	Z	DC	C
04h	FSR	Registro utlizado como puntero en el direccionamiento indirecto							
05h	PORTA				RA4/T0CKI	RA3	RA2	RA1	RA0
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT
07h		Posición no implementada, se lee como 0							
08h	EEDATA	Registros de datos EEPROM							
09h	EEADR	Registro de direcciones EEPROM							
0Ah	PCLATH				Buffer escrito con los 5 bit más altos del PC				
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
	BANCO 1								
80h	INDF	Registro utilizado en el direccionamiento indirecto (no es un registro físico)							
81h	OPTION	/RBPU	INTEDG	T0SC	T0SE	PSA	PS2	PS1	PS0
82h	PCL	Registro con los 8 bit más bajos del Contador del Programa							
83h	STATUS	IRP	RP1	RP0	/TO	/PD	Z	DC	C
84h	FSR	Registro utlizado como puntero en el direccionamiento indirecto							
85h	TRISA				Reg. De configuración de las líneas del Puerto a				
86h	TRISB	Registro de configuración de las líneas del Puerto B							
87h		Posición no implementada, se lee como 0							
88h	EECON1				EEIF	WRERR	WREN	WR	RD
89h	EECON2	Reg. De control para grabar en la EEPROM de datos (no es un registro fisico)							
8Ah	PCLATH	Buffer escrito con los 5 bit más altos del PC							
8Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

1.4.5.1 INDF

Registro para el direccionamiento indirecto de datos. Éste no es un registro disponible físicamente. Utiliza el contenido del SFR para seleccionar indirectamente la memoria de datos o RAM del usuario; la instrucción determinará lo que se debe hacer con el registro señalado.

El direccionamiento indirecto es un recurso de los microcontroladores PIC16F84A.

Para el direccionamiento indirecto se utiliza los registros INDF y FSR ubicados dentro de la zona de registros especiales, concretamente en la posición 00h y 04h del Banco 0, aunque el registro INDF no está implementado físicamente.

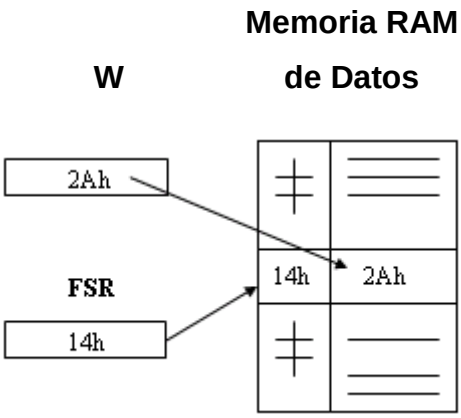
El registro FSR sirve como **puntero** para el direccionamiento indirecto. Si en cualquier instrucción se opera con el registro INDF, en realidad se está operando con la dirección donde se apunte el contenido del registro **FSR.**

Por ejemplo si el **FSR** contiene el valor 14h una instrucción que opera sobre INDF, opera en realidad sobre la dirección 14h.

Se puede decir en este ejemplo que la posición 14h de memoria fue direccionada en forma indirecta a través del puntero FSR.

Si FSR =14h, la instrucción movwf INDF carga el contenido de registro W en la posición 14h de la RAM, tal como se muestra en la figura 1.4. Simbólicamente se representa:

(W) $\longrightarrow$ ((FSR)), en este ejemplo, (W) $\longrightarrow$ (14h)

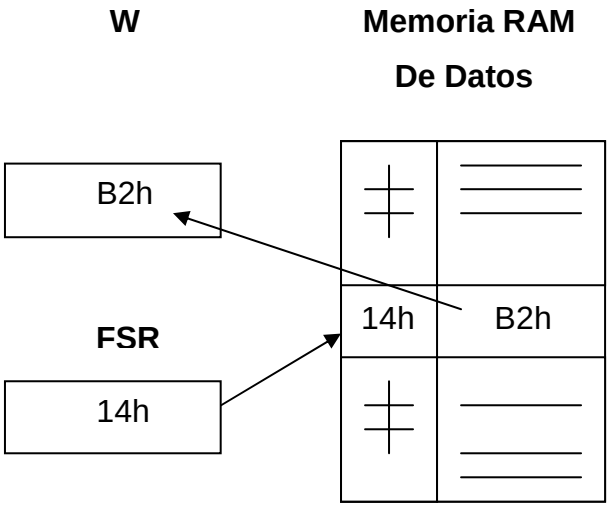


A) Al ejecutar movwf INDF

Figura 1.4. Funcionamiento de las instrucciones de Direccionamiento Indirecto

Siguiendo con otro ejemplo, si FSR=14h, la instrucción *movf* INDF,W carga el contenido de la posición 14h de la RAM en el registro de trabajo W, tal como se muestra en la figura 1.5. Simbólicamente se representa:

((FSR)) → (W), en este ejemplo, (14h) → (W)



B) Al ejecutar movf INDF,W

Figura 1.5 . Funcionamiento de las instrucciones de Direccionamiento Indirecto

El direccionamiento indirecto es muy útil para el procedimiento de posiciones consecutivas de memoria RAM de datos.

1.4.5.2 **TMR0** (timer 0).

Temporizador/contador de 8 bits. Se incrementa con una señal externa aplicada al pin RA4/TOCKI o de acuerdo a una señal interna proveniente del reloj del microcontrolador.

El PIC16F84A dispone de un timer principal denominado Timer 0 o TMR0 que es un contador ascendente de 8 bits. El TMR0 se inicializa con un valor, que se incrementa con cada impulso de entrada hasta su valor máximo b”11111111” ; con el siguiente impulso de entrada el contador desborda pasando a valer b”00000000”, circunstancias que se advierte mediante la activación del flag del contaje localizado en el registro INTCON.

Cuando el TMR0 trabaja como temporizador cuenta los impulsos de fosc/4. Se usa para determinar intervalos de tiempo concreto. Estos impulsos tienen una duración conocida de un ciclo máquina que es en cuatro veces el periodo de la señal de reloj.

Para una frecuencia de reloj igual a 4MHz el TMR0 se incrementa cada 1 µs.

Como se trata de un contador ascendente el TMR0 debe ser cargada con el valor de los impulsos que se desean contar restados de 256 que es el valor de desbordamiento, por ejemplo para contar cuatro impulsos se carga al TMR0 con 256 – 4 = 252.

Número de pulsos a contar: 4_{10} = b”00000100”

Número a cargar: $256_{10} = 252_{10}$ = b”11111100”

Incrementar a cada ciclo de instrucción: b”11111100” , b”11111101” , b”11111110” , b”11111111” ; aquí se desborda pasando a b”0000000” y activando al flan T0IF.

De esta manera, con la llegada de cuatro impulsos, el timer se ha desbordado alcanzando el valor b”00000000” que determina el tiempo de

temporización, en este caso 4 µs si los impulsos se hubieran aplicado cada microsegundo.

1.4.5.3 **PCL** (Program Counter Low byte).

Byte bajo del contador de programa, figura 1.6, el PIC16F84A dispone de un contador de programa de 13 bits. Sus bits de menor peso corresponden a los 8 bits del registro PCL , implementado en la posición de memoria RAM02h y duplicada en la posición 82h del banco1 por lo que puede ser leído o escrito directamente.

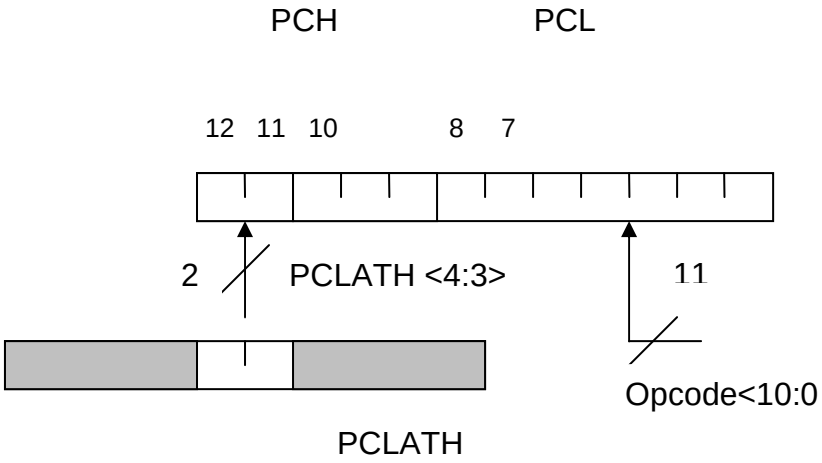


Figura 1.6 Composición del PC en instrucciones "call" y "goto"

Los cinco bits de mayor peso del PC corresponden con los del registro PCH que no puede ser leído ni escrito directamente figura 1.7.

Al conectar la alimentación se inicializa a (PCL)=b"00000000" y (PCH)=b"00000"

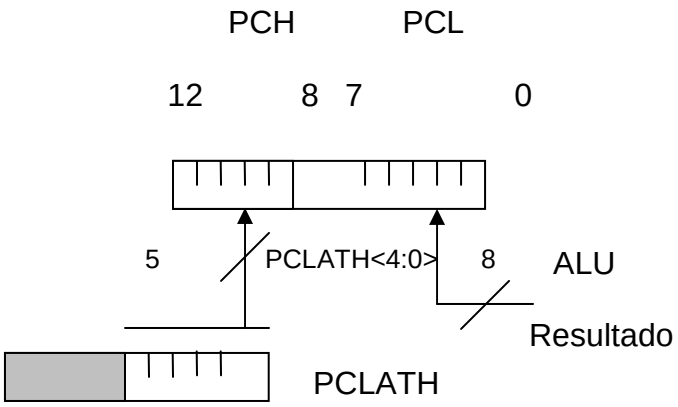


Figura 1.7. Composición del PC en instrucciones con PCL como destinatario

1.4.5.4 STATUS

El registro de estado o STATUS indicada el estado de la última operación aritmética o lógica realizada, la causa de reset y los bits de selección de banco para la memoria de datos.

A los bits de registro de estado se les suele denominar flags. Al conectar la alimentación los contenidos es (STATUS)=b"00011xxx".

Tabla 1.2 Registro de estado o STATUS

IRP	RP1	RP0	/TO	/PD	Z	DC	C
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

1.4.5.4.1 **C** (Carry bit).

Flag de acarreo en el octavo bit. Esta instrucción de suma aritmética se activa cuando se presenta acarreo desde el bit más significativo de resultado, lo que indica que el resultado ha desbordado la capacidad del registro sobre el que trabaja, es decir, el resultado de la operación ha superado el valor 11111111_2, (255_{10}), que es el máximo valor que se puede representar con 8 bits.

C=0. En la suma significa que no habido acarreo y en la resta que el resultado ha sido negativo.

C=1. En la suma significa que habido acarreo y en la resta que el resultado a sido positivo.

1.4.5.4.2 **DC** (Digital carry).

Flag de acarreo en el cuarto bit de menos peso. En operaciones aritméticas se activa cuando hay un acarreo entre el bit 3 y 4, es decir, cuando hay acarreo entre los nibbles (medios bytes) de menor y mayor peso.

1.4.5.4.3 **Z** (Zero).

Flag de cero. Se activa a “1” cuando el resultado de una operación aritmética o lógica es cero.

Z=0. El resultado de la última operación ha sido distinto de cero.

Z=1. El resultado de la última operación ha sido cero.

1.4.5.4.4 **/PD** (Power Down).

Flag de bajo consumo. Es un bit sólo de lectura, no puede ser escrito por el usuario. Sirve para detectar el modo de bajo consumo.

/PD=0. Al ejecutar la instrucción sleep y entra en reposo.

/PD=1. Tras conectar la alimentación V_{DD} o al ejecutar clrwdt.

1.4.5.4.5 **/TO** (Timer Out).

Flag indicador de fin de temporización del Watchdog. Es un bit de sólo lectura, no puede ser escrito por el usuario. Se activa en “0” cuando el circuito de vigilancia Watchdog finaliza la temporización. Sirve para detectar si una condición de reset fue producida por el Watchdog Timer.

/TO=0. Al desbordar el temporizador del Watchdog Timer.

/TO=1. Tras conectar VDD (funcionamiento normal) o al ejecutar las instrucciones clrwdt o sleep.

1.4.5.4.6 **RPO** (Register Bank Select bit).

Selección del banco para el direccionamiento directo. Señala el banco de memoria de datos seleccionado.

RPO=0. Selecciona el Banco 0.

RPO=1. Selecciona el Banco 1.

1.4.5.5 FSR

Selector de registro para direccionamiento indirecto. En asociación con el registro INDF se utiliza para seleccionar indirectamente los otros registros disponibles.

Al conectar su alimentación su contenido es desconocido, (FSR)=b”xxxxxxxx”.

1.4.5.6 PORTA

Puerto de entrada/salida de 5 bits (pines RB0 a RB4). El puerto A puede leerse o escribirse como si se tratara de un registro cualquiera.

El registro que controla el sentido (entrada/salida) de sus pines se llama TRISA y está localizada en la dirección 85h del Banco1. Su pin RA4/TOCKI también puede servir de entrada al Timer 0. Al conectar la alimentación queda configurada como entrada.

1.4.5.7 PORTB

Puerto de entrada/salida de 8 bits . El puerto B puede leerse o escribirse como si se tratara de un registro cualquiera. El registro que controla el sentido

(entrada/salida) de sus pines se llama TRISB y está localizada en la dirección 86h del Banco 1. Alguno de sus pines tiene función alterna en la generación de interrupción. Al conectar la alimentación queda configurada como entrada.

1.4.5.8 **EEDATA** (EEPROM Data register).

Contiene los bytes que se van a escribir o que se han leído de la EEPROM de datos. Al conectar la alimentación su contenido es desconocido, (EEDATA)=b"xxxxxxxx"

1.4.5.9 **EEADR** (EEPROM Address Register).

Contiene la dirección de la EEPROM de datos a la que accede para leer o escribir. Al conectar la alimentación su contenido es desconocido, (EEADR)=b"xxxxxxxx".

1.4.5.10 **PCLATH** (PC Latch Hight).

Registro que permite acceder de forma indirecta a la parte alta del contador de programa en alguna instrucción. Al conectar la alimentación se resetea (PCLATH)=b"---00000".

1.4.5.11 **INTCON** (Interrupts Control Register).

Registro para el control de las interrupciones. Es el encargado del manejo de las interrupciones. Contiene los 8 bits que se muestra en la tabla 1.3 , de los cuales unos actúan como flags señaladotes del estado de la interrupción y otro como bits de permiso o autorización para que se pueda producir la interrupción. Al conectar la alimentación su contenido es (INTCON)=b"0000000x".

Tabla 1.3 Registro de control de las interrupciones INTCON

GIE	EEIE	TOIE	INTE	RBIE	T0IF	INTF	RBIF
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

1.4.5.11.1 **RBIF** (RB port change Interrupt Flag).

Flag de estado de interrupción RBI.

Indica que se ha producido una interrupción por cambio de estado de cualquiera de las líneas RB4 a RB7.

RBIF=0. Ninguna de las entradas RB7 a RB4 ha cambiado de estado.

RBIF=0. Cualquiera de las líneas RB7a RB4 del puerto B ha cambiado.

1.4.5.11.2 **INTF** (External Interrupt Flag bit).

Flag de estado de la interrupción externa INT. Indica que ha ocurrido una interrupción a través del pin RB0/INT. (Debe borrarse por software).

1.4.5.11.3 **T0IF** (TMR0 Overflow Interrupt Flat Bit).

Flat de estado de la interrupción producida por un TMR0. Indica que se producido una interrupción por desbordamiento del Timer 0, es decir, que a pasado de b”11111111” (FFh) a b”00000000” (00h).

T0IF=0. El TMR0 no se ha desbordado.

T0IF=1. El TMR0 se ha desbordado. (Debe borrarse por software).

1.4.5.11.4 **RBIE** (RB Port Change Interrupt Enable).

Habilitación de la interrupción RBI. Flag que autoriza la interrupción por cambio de estado de las líneas RB7:RB4 del puerto B.

RBIE=0. Interrupción RBI deshabilitada.

RBIE=1. Interrupción RBI habilitada.

1.4.5.11.5 **INTE** (External INT Enable bit).

Habilitación de la interrupción externa INT. Flag que autoriza la interrupción externa a través del pin RB0/INT.

INTE=0. Interrupción INT deshabilitada.

INTE=1. Interrupción INT habilitada.

1.4.5.11.6 **T0IE** (TMR0 Interrupt Enable bit).

Habilitación de la interrupción T0I. Flag que autoriza la interrupción por desbordamiento del Timer 0.

T0IE=0. Interrupción T0I deshabilitada.

T0IE=1. Interrupción T0I habilitada.

1.4.5.11.7 **EEIE** (EEPROM Write Complete Interrupt Enable).

Habilitación de la interrupción EEI. Flag que autoriza la interrupción por escritura completa de un byte en la EEPROM de datos del PIC (el flag EEIF se encuentra en el registro EECON1).

EEIE=0. Interrupción EEI deshabilitada.

EEIE=1. Interrupción EEI habilitada.

1.4.5.11.8 **GIE** (Global Interrupt Enable).

Flag de habilitación global del permiso de interrupción. Se borra automáticamente cuando se reconoce una interrupción para evitar que se produzca otra mientras se está atendiendo a la primera. Al retorno de la interrupción con una instrucción *retfie,* el bit GIE se vuelve a activar poniéndose a “1”.

GIE=0. No autoriza interrupción de ningún tipo.

GIE=1. Autoriza cualquier tipo de interrupción. Se pone a “1” automáticamente con la instrucción *retfie.*

1.4.5.12 OPTION

Registro de configuración múltiple, aunque su misión principal es gobernar el comportamiento del TMR0. Algunos microcontroladores PIC tienen una instrucción denominada también *option,* por ello, el fabricante Microchip recomienda darle otro nombre a este registro. Así en el fichero de definición de etiqueta P16F84.

INC se le nombra como OPTION_REG. Al conectar la alimentación todos sus bits se pone a “1”, OPTION_REG = “b11111111”.

Tabla 1.4. Registro OPTION_REG

/RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

1.4.5.12.1 PS2:PS0

(Prescaler Rate Select bits). Bits para seleccionar los valores del Prescaler o rango con el que actúa el divisor de frecuencia, según la tabla 1.5.

Tabla 1.5 Selección del rango del divisor de frecuencia

PS2 PS1 PS0	Divisor del TMR0	Divisor del 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

1.4.5.12.1 **PSA** (Prescaler Assignment bit).

Asignación del divisor de frecuencia. El Prescaler es compartido entre el TMR0 y el WDT; su asignación es mutuamente excluyente ya que solamente a uno de ellos se puede aplicar el divisor de frecuencia a la vez.

PSA=0. El divisor de la frecuencia se asigna al TMR0.

PSA=1. El divisor de la frecuencia se asigna al Watchdog.

1.4.5.12.2 **T0SE** (TMR0 Source Edge Select bit).

Selecciona flanco de la señal al TMR0.

T0SE=0. TMR0 se incrementa en cada flanco ascendente de la señal aplicada al pin RA4/TOCKI.

T0SE=1. TMR0 se incrementa en cada flanco descendente de la señal aplicada al pin RA4/TOCKI.

1.4.5.12.4 **T0CS** (TMR0 Clock Source Select bit).

Selecciona la fuente de la señal TMR0.

T0CS=0. Pulso de reloj interno Fosc/4 (TMR0 como temporizador).

TOCS=1. Pulsos introducidos a través del pin RA4/TOCKI (TMRO como contador).

1.4.5.12.5 **INTEDG** (Interrupt Edge Select bit).

Selector de flanco de la interrupción INT.

INTEDG=0. Interrupción por flanco descendente del pin RB0/INT.

INTEDG=1. Interrupción por flanco ascendente del pin RB0/INT.

1.4.5.12.6 **/RBPU** (Resistor Port B Pull-Up Enable bit).

Habilitación de las resistencias de Pull-Up del Puerto B.

/RBPU=0. Habilita las resistencias del Pull-Up del puerto B.

/RBPU=1. Deshabilita las resistencias del Pull-Up del puerto B.

1.4.5.13 TRISA

Registro de configuración de las líneas del Puerto A. Es el registro de control para el Puerto A.

Un “0” en el bit correspondiente al pin lo configura como salida, mientras que un “1” lo hace como entrada. Al igual que el puerto A, sólo dispone de 5 bits. Al conectar la alimentación todos sus bits se ponen a “1” ,

(TRISA) = b”---11111”.

1.4.5.14 TRISB

Registro de configuración de las líneas del puerto B. Es el registro de control para el puerto A.

Un “0” en el bit correspondiente al pin lo configura como salida, mientras que un “1” lo hace como entrada. Al conectar la alimentación todos sus bits se pone a “1”, (TRISB) = b”11111111”.

1.4.5.15 **EECON1** (EEPROM Control Register 1).

Registro para el control de la memoria EEPROM de datos. Al conectar la alimentación su contenido (EECON1) = b”0x000”.

Tabla 1.6 Registro de control de la EEPROM de datos INTCON

			EEIF	WRERR	WREN	WR	RD
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

1.4.5.14.1 **RD** (Read Control Bit).

Bit de control de lectura en la EEPROM. Al ponerlo en “1” se inicia la lectura de un byte en la EEPROM de datos. Este bit se limpia (se pone a “0”) por hardware automáticamente al finalizar la lectura de la posición EEPROM.

RD=0. No inicializa la lectura de la EEPROM o la misma ha terminado.

RD=1. Inicializa la lectura de la EEPROM. Se borra por hardware.

1.4.5.14.2 **WR** (Write Control Bit).

Bit de control de escritura en la EEPROM. Al ponerlo en “1” se inicia una escritura de un byte en la EEPROM . Este bit se limpia (se pone en “0”) por hardware automáticamente una vez la escritura de la EEPROM ha terminado.

WR=0. No inicia la escritura de la EEPROM o la misma ha terminado.

WR=1. Inicia la escritura de la EEPROM. Se borra por hardware.

1.4.5.14.3 **WRERR** (EEPROM Write Error Flag Bit).

Flag de error en la escritura. Se posiciona a “1” cuando la operación de escritura termina prematuramente debido a cualquier condición de reset.

WRERR=0. La operación de escritura se ha completado correctamente.

WRERR=1. La operación de escritura ha terminado prematuramente.

1.4.5.14.4 **EEIF** (EEPROM Write Operation Interrupt Flag Bit).

Flag de estado de la operación de escritura de un byte en la EEPROM.

EEIF=0. La operación de escritura de la EEPROM no ha terminado o no comenzó.

EEIF=1. La operación de escritura de la EEPROM ha terminado. Debe borrarse por software.

1.4.5.14.5 **Bits 5,6 y 7** (Unimplemented).

No implementados físicamente. Se leen “0”.

1.4.5.15 **EECON2** (EEPROM Control Register 2).

Este registro no está implementado físicamente, por lo que es imposible leerlo (si se intenta leer, todos sus bits se leen como cero). Se emplea como dispositivo de seguridad durante el proceso de escritura de la EEPROM, para evitar las interferencias en el largo intervalo de tiempo que precisa su desarrollo.

1.4.5.16 REGISTROS DE CONFIGURACIÓN

El PIC16F84 dispone de una palabra de configuración de 14 bits que se escribe durante el proceso de grabación del microcontrolador y que no se puede modificar durante la ejecución de un programa.

Dicho bits ocupan la posición reservada de memoria de programa 2007h

Tabla 1.7 Registro de configuración (Configuración Word)

			CP	PWRTE	WDTE	FOSC1	FOSC0
Bit 13	……	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

1.4.5.16.1 FOSC<1:0>

(Flag Oscilador Selection). Selección del tipo de oscilador:

FOSC = 00. Oscilador de bajo consumo LP (32KHz - 200KHz)

FOSC = 01. Oscilador estándar XT (100KHz – 4MHz)

FOSC =10. Oscilador de alta velocidad HS (4MHz – 20MHz)

FOSC = 11. Oscilador de bajo corte RC.

1.4.5.16.2 **WDTE** (Watchdog Enable).

Bit de habilitación del Watchdog.

WDTE = 0. Watchdog deshabilitado.

WDTE = 1. Watchdog habilitado.

1.4.5.16.3 **PWRTE** (Power-up Timer Enable).

Activación del temporizador Power-up.

PWRTE = 0. Temporizador Power-up deshabilitado.

PWRTE = 1. Temporizador Power-up habilitado.

1.4.5.16.4 **CP** (Code Protection bit).

Bit de protección de código.

CP = 0. Toda la memoria de programa está protegida contra lecturas indeseables.

CP = 1. La memoria de programa se puede leer. No está protegida.

1.5 SUBRUTINAS

Las subrutinas son técnicas para realizar códigos de programa más cortos y eficaces.

1.5.1 DETALLE DE LAS SUBRUTINAS

Algunas veces el mismo grupo de instrucciones es ejecutado en diferentes partes de un programa. Según el procedimiento planteado hasta el presente, cada vez que dicho tramo de programa es requerido deberá insertar dentro del programa principal tantas veces como esto sea necesario. Esta puede parecer la forma más directa de atacar el problema pero es ineficiente

ya que requiere mayor extensión del programa y por lo consiguiente mayor utilización de la memoria ROM.

El flujo de programa resulta secuencial tal como se muestra en la figura 1.8.

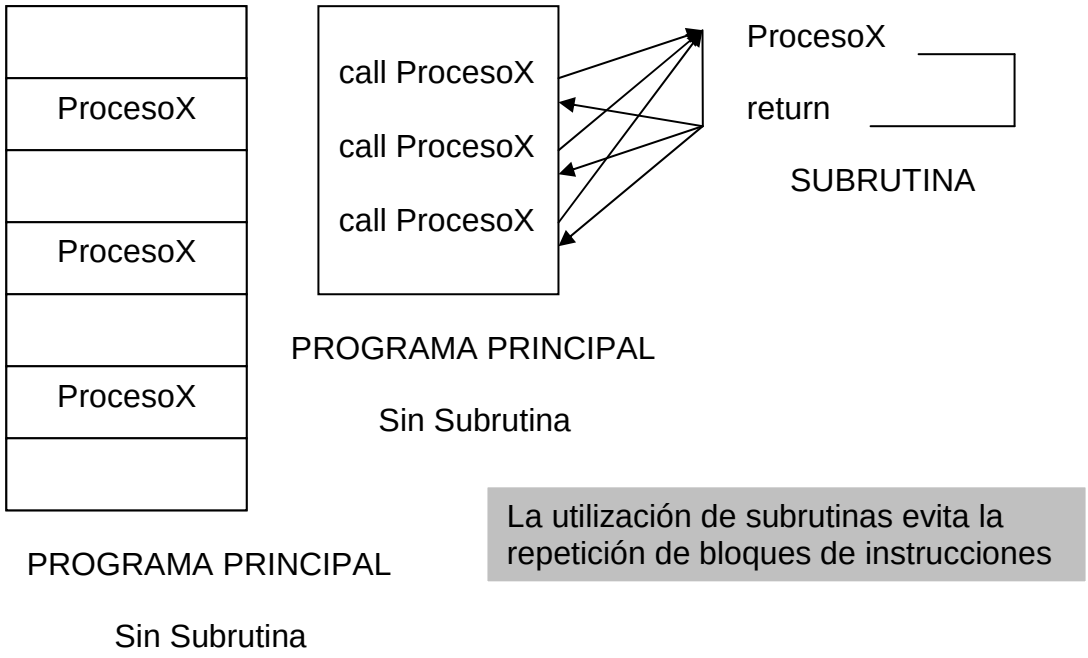


Figura 1.8 Utilización de las subrutinas

La solución más efectiva en términos de ahorro de memoria, se obtiene si el grupo de instrucciones que se repita aparezca una sola vez en el programa, pero con capacidad para ser ejecutado desde todos los puntos en el que aquel se pida, para implementar esta solución se utiliza la subrutina.

Una subrutina es un conjunto de instrucciones al que se tiene acceso desde cualquier punto del programa principal. Es decir, es un subprograma que se ejecuta cada vez que el programa principal lo necesita.

Como una subrutina conceptualmente queda fuera del flujo secuencial del programa principal, son necesarios ciertos mecanismos para poder llegar a

ella, una vez que se ha ejecutado las instrucciones que la componen, debe ser posible regresar al punto donde se quedó la ejecución del programa.

La acción de pasar del programa principal a la subrutina se denomina “llamar al subrutina” y se realiza con la instrucción *call.* Que se debe intercalar en el programa principal, como se muestra en la figura 1.8.

La acción de volver al programa principal después de llevar a cabo las tareas determinadas por la subrutina se llama “retorno de la subrutina” y se realiza con la instrucción *return* es un “salto inteligente” que sabe que tiene que volver a la instrucción inmediatamente después del *call* que llamó a la subrutina.

La figura 1.8 ilustra el procedimiento de ejecución del programa con subrutinas.

La instrucción del programa principal son ejecutadas sucesivamente hasta que se encuentra la primera instrucción *call procesos,* después de lo cual, la subrutina ProcesosX se ejecuta como cualquier otra sección del programa. La ultima instrucción de la subrutina es *return* que causa el regreso de la secuencia de ejecución al programa principal.

El programa con subrutina de la figura 1.8 podría tener el siguiente formato:

```
Principal                    ; Comienzo del programa principal.
…
call    ProcesoX             ; El control del programa pasa a la subrutina
…                            ; “Procesos”. Después de ejecutar la
…                            ; la subrutina la ejecución del programa
…                            ; vuelve a la instrucción después del “call”
…
call   ProcesoX              ; El control del programa pasa a la subrutina
…                            ; “Procesos”. Después de ejecutar la
…                            ; la subrutina la ejecución del programa
…                            ; vuelve a la instrucción después del “call”
;subrutina “procesoX”-------------------------------------------------------------------------
Procesos
…                    ;Aquí la instrucción de la subrutina.
Return               ;La subrutina devuelve con “return” el
…                    ;control al programa inicial.
END                  ;directiva “END” al final del programa.
```

La principal ventaja de las subrutinas es que la extensión de los programas se hace mucho más corta. No obstante las subrutinas presentan una desventaja, provoca una ejecución más lenta debido a que se tiene que ejecutar dos instrucciones extras *call* y *return* cada vez que se realiza una llamada y el obligado retorno de subrutina.

1.5.2 SUBRUTINAS ANIDADAS

Cuando una subrutina llama a otra subrutina se produce la situación conocida como **anidamiento de subrutinas,** es decir, hay subrutinas anidadas dentro de otra.

Cada *call* sucesivo sin que intervenga un *return* crea un nivel de anidamiento adicional. Esto se ilustra en la figura 1.9.

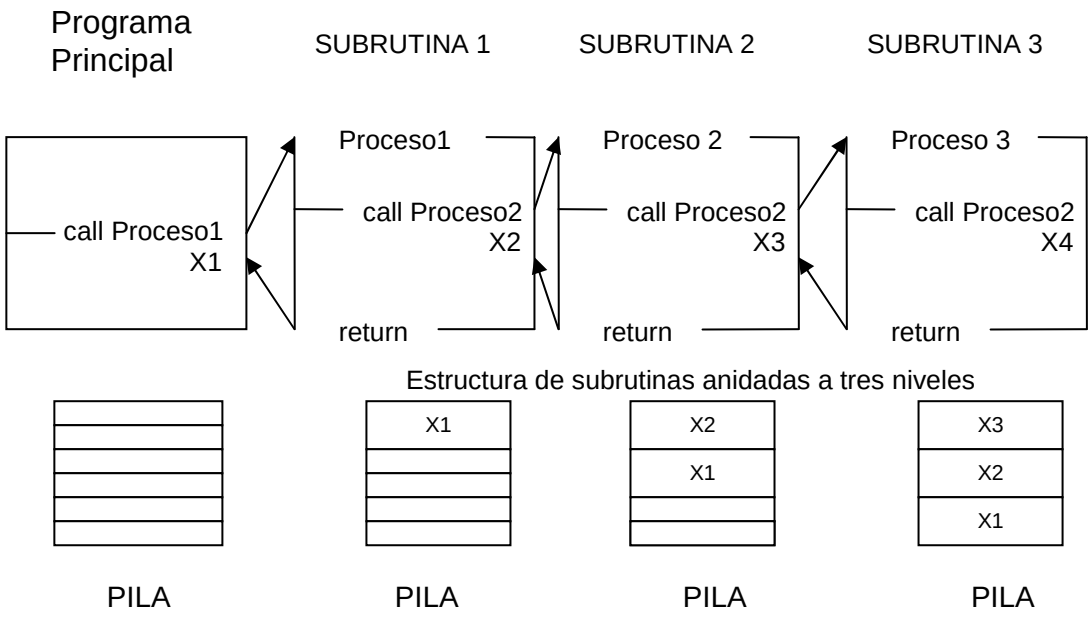


Figura 1.9 Subrutinas anidadas

La figura 1.9 puede tener el siguiente formato:

```
            …
Principal                   ; Comienzo del programa principal
          …
          call    Proceso1  ; El control del programa pasa a la subrutina “Proceso1”
          …                 ; Después de ejecutar la subrutina “Proceso1” la
          …                 ; ejecución del programa vuelve a la instrucción
          …                 ; inmediata después de “call”.
          …                 ; Aquí la instrucción final del programa principal.

; Subrutina “Proceso1”-----------------------------------------------------------------------
;....
Proceso1
          …                 ; Aquí las instrucciones de la subrutina “Proceso1”.
          …
          call    Proceso2  ; El control del programa pasa a la subrutina “Proceso2”
          …                 ; Después de ejecutar la subrutina “Proceso2” la
          …                 ; la ejecución del programa vuelve a la instrucción
          …                 ; inmediata después del call.

          return            ; Esta subrutina devuelve el control al programa principal

; Subrutina “Proceso2”-----------------------------------------------------------------------
;....
Proceso2
          …                 ; Aquí las instrucciones de la subrutina “Proceso2”.
          …
          call    Proceso2  ; El control del programa pasa a la subrutina “Proceso3
          …                 ; Después de ejecutar la subrutina “Proceso3” la
          …                 ; la ejecución del programa vuelve a la instrucción
          …                 ; inmediata después del call.

          return

; Subrutina “Proceso3”-----------------------------------------------------------------------
;....
Proceso3
          …                 ; Aquí las instrucciones de la subrutina “Proceso3”.
          …
          return
          …
          …
          END               ; La directiva “END” se pondrá al final del programa.
```

El nivel de anidamiento está limitado para cada microcontrolador y en el microcontrolador PIC16F84 es de 8 niveles. Es decir, para un PIC16F84 no puede haber más de ocho subrutinas anidadas como las esquematizadas en la figura 1.9.

1.5.3 LA PILA

La pila es una zona de memoria que se encuentra separada tanto de la memoria de programa como de la de datos dentro del microcontrolador.

Su estructura es de tipo LIFO (Last In First Out) por lo que el último dato que se guarda es el primero que sale.

El PIC16F84 dispone de una pila con ocho niveles o registros de una longitud de 13 bits cada uno de ellos como se lo puede interpretar en la figura 1.10.

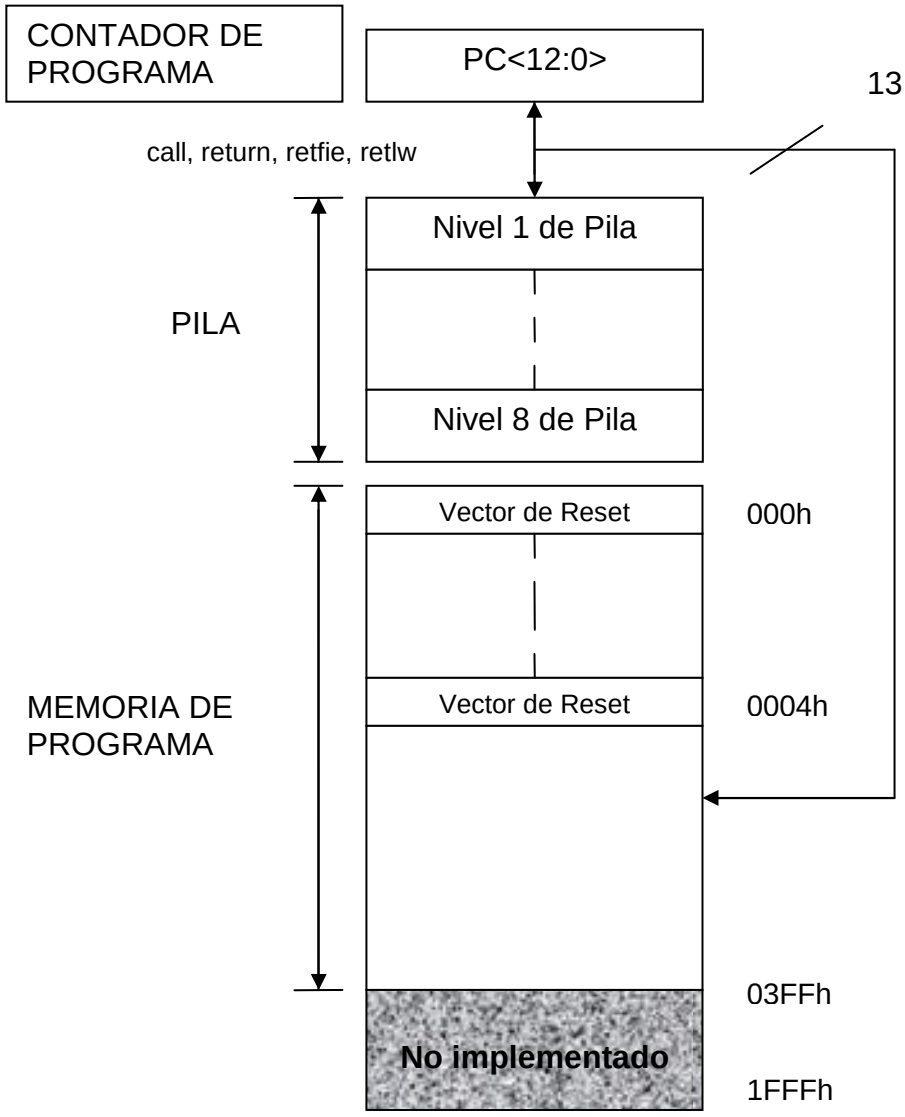


Figura 1.10 Estructura de la pila y memoria de programa del PIC16F84

La manera de cargar la pila es a través de la llamada a subrutina con la instrucción *call* , que almacena el contenido del contador de programa (PC) en la posición superior de la pila. Para recuperar el contenido de la pila en el PC, hay que ejecutar la instrucción de retorno de subrutina *return.*

1.5.4 INSTRUCCIONES “CALL” Y “RETURN”

La localización de una subrutina se identifica por la dirección de su primera instrucción.

El efecto de la instrucción *call* es provocar que la que la ejecución se transfiera a la subrutina. De esto se desprende que la mencionada instrucción contenga la dirección de la primera posición de memoria ocupada por la subrutina, como se lo detalla en la figura 10.11.

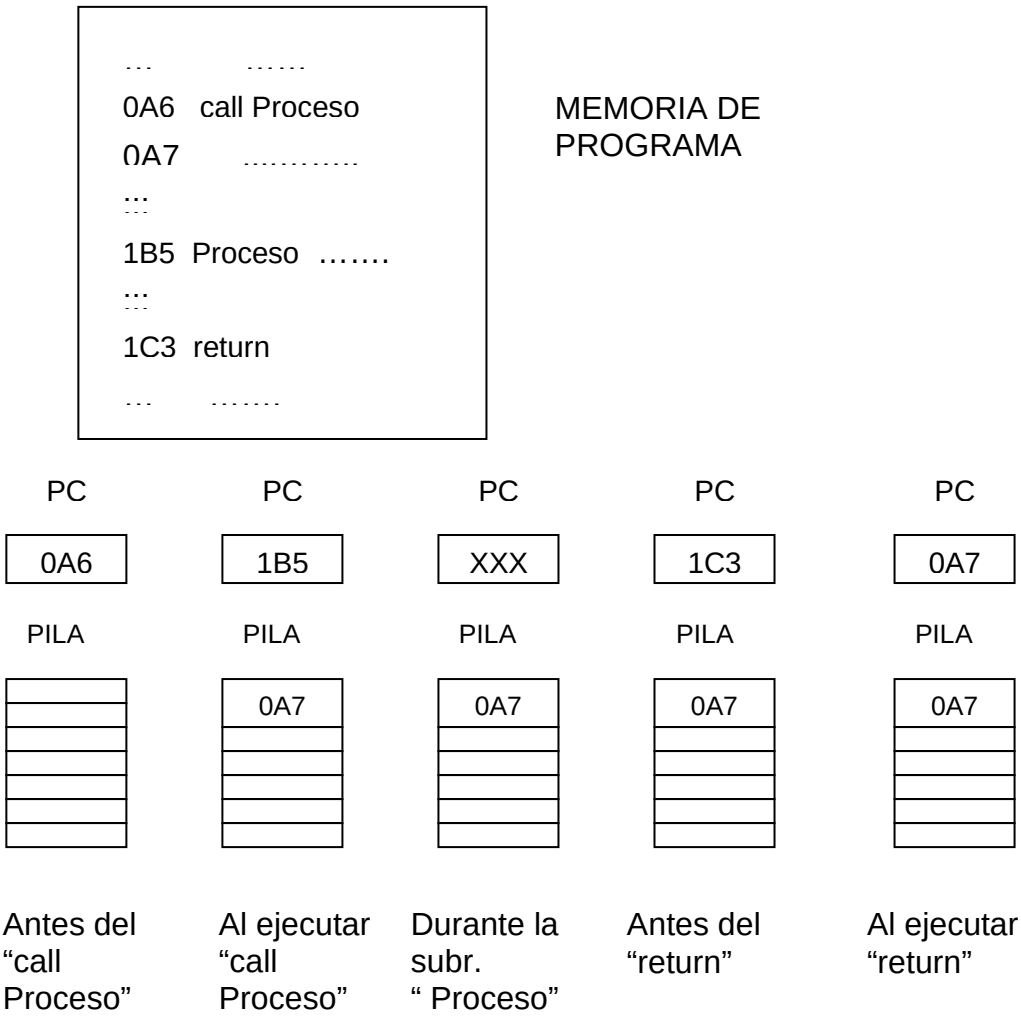


Figura 1.11 Mecanismo de funcionamiento de una subrutina.

Por otro lado, la instrucción *return* que provoca el retorno al programa principal, debe "recordar" la localización de la instrucción que sigue al *call*. Esto es posible sólo si la dirección de esa instrucción ha sido preservada en una zona de memoria, que no es otra que la pila.

Antes de transferir el control a una subrutina, la instrucción *call* se encarga de almacenar en la pila la dirección de la instrucción que sigue. Esto es así porque el contador de programa se incremente automáticamente cada vez que el microprocesador ejecuta una instrucción el PC ya se encuentra apuntando a la dirección de la siguiente instrucción.

Por lo tanto, cuando la instrucción *call* deposita en la pila el contenido del PC, lo que efectivamente está preservando es la dirección del punto de retorno. La figura 1.11 ilustra este mecanismo.

La transferencia de la secuencia de ejecución a la subrutina se logra cargando el PC con la dirección contenida en la instrucción *call* . De esta forma, la siguiente instrucción que se obtiene de la memoria proviene de la subrutina.

La ejecución de la instrucción *return* extrae la dirección almacenada en la pila y la transfiere al PC. Así, la próxima instrucción que se ejecuta después del *return* es precisamente la siguiente al *call* (que es la que se había guardado previamente en la pila), regresando el control al programa principal.

En las subrutinas anidadas, el efecto de cada instrucción *call* es guardar en la pila la dirección apropiada para el retorno. La instrucción *return* y sus efectos en la pila aseguran que el control del programa eventualmente regresa a la instrucción siguiente al primer *call*, como se observa en la figura 1.9.

El número de subrutinas anidadas está limitado por el tamaño de la pila de cada microcontrolador. En el PIC16F84 la pila tiene un tamaño de 8 posiciones, por lo tanto el anidamiento máximo de 8 niveles. No hay ningún mecanismo hardware que indique desbordamiento de la pila, debe ser el diseñador del programa quien debe procurar que esto no se produzca.

1.5.5 VENTAJAS DE LAS SUBRUTINAS

- Se puede escribir como subrutinas secciones de código y ser empleadas en muchos programas.
- Dan a los programas un carácter modular. Se puede codificar diferentes módulos para usarlos en cualquier programa obteniendo así una librería de subrutinas.
- Reduce notablemente el tiempo de programación y la detección de errores.

- El código es más fácil de interpretar, dado que las instrucciones de las subrutinas no aparecen en el programa principal. Sólo figuran las llamadas *call.*

1.5.6 LIBRERÍA DE SUBRUTINAS

Es frecuente necesitar más de una subrutina en los programas. También es habitual que algunas subrutinas se utilicen en varios programas. En estos casos es conveniente disponer de bibliotecas de subrutinas denominadas **librerías.**

En cada programa se cargan las subrutinas que se precisen. El ensamblador MPASM dispone de una directiva denominada **INCLUDE** que realiza esta función “pegando” el fichero de referencia en el programa. dicho fichero se inserta en el código durante el proceso de ensamblado.

1.5.7 DIRECTIVA “INCLUDE”

El formato de la directiva INCLUDE es:

INCLUDE <*include_file*>

El fichero especificado por <include_file> es leído como un fichero fuente. El efecto es el mismo que si el texto entero del <include_file> hubiera sido insertado dentro del fichero origen en la localización donde esta directiva se encuentra.

Es importante tener en cuenta que este fichero <include_file> no debe finalizar con la directiva END, la cual debe ir situada en el programa principal.

Esta directiva se usa cuando se incluye al principio de los programas

```
INCLUDE <P16F84A.INC
```

Con la directiva anterior lo que se hace es añadir al programa la definición de los registros de SFR y de sus bits.

Si es especificada la trayectoria completa del fichero <include_file>, sólo en ese trayecto será buscado. En caso contrario el orden de búsqueda será: directorio actual de trabajo, directorio del fichero fuente y, por último, directorio del fichero ejecutable MPASM.EXE.

La extensión "*.INC" no es obligatorio, aunque se recomienda su utilización para diferenciarlos de los programas propiamente dichos con extensión "*.asm".

Los ficheros *include* a su vez pueden tener otras directivas INCLUDE, pero no es recomendable ya que puede provocar confusión. De todas formas este anidamiento tampoco es ilimitado, puesto que el MPASM sólo permite hasta seis niveles.

1.5.8 SIMULACIÓN DE SUBRUTINAS EN MPLAB

En el simulador del MPLAB, con el comando Debugger > Step Into se ejecutarían paso a paso todas y cada una de las instrucciones que conforman la subrutina.

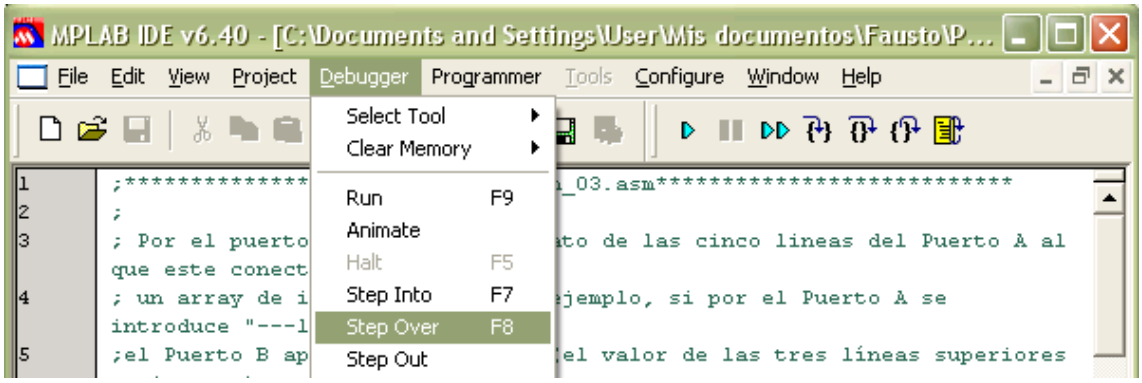


Figura 1.12 Las subrutinas se pueden ejecutar con el comando Step Over

Sin embargo, a veces en la simulación interesa ejecutar la subrutina sin tener que ejecutar una por una todas sus instrucciones. Para esto se utiliza el comando Debugger > Step Into, pero cuando llega a la ejecución de una subrutina (instrucción *call*) ésta se ejecuta de igual forma que si fuera una sola instrucción.

El contenido de la pila puede visualizarse seleccionado la opción View > Hardware Snack, como se alustra en la figura 1.13.

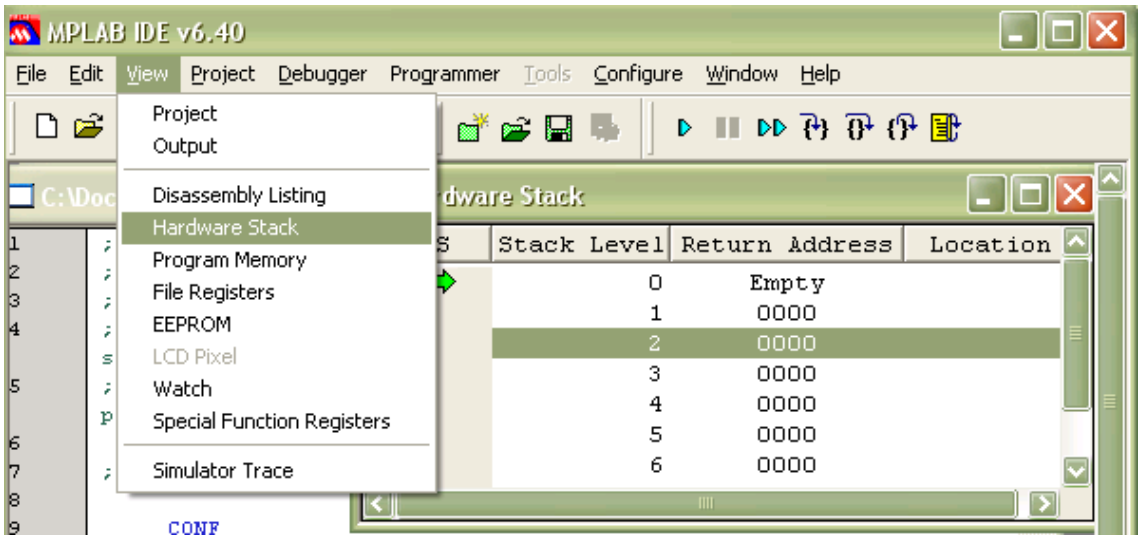


Figura 1.13 Visualización de la pila en el MPLAB

1.6 SUBRUTINA DE RETARDO

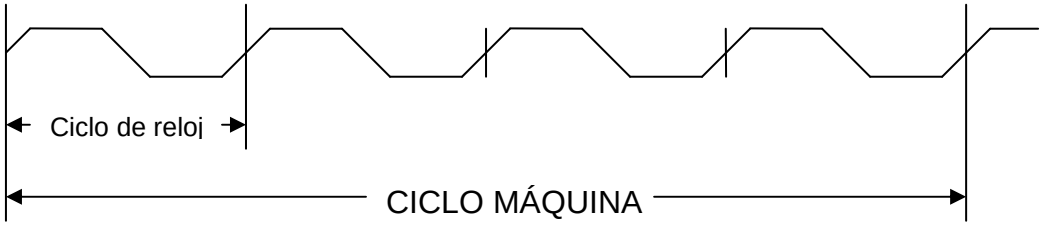
Controla el tiempo que tarda en ejecutarse algunas acciones.

1.6.1 CICLO MÁQUINA

El tiempo que tarda en ejecutar un programa depende de la frecuencia del oscilador conectado al microcontrolador y del número de ciclo máquina ejecutados.

Se lo define como un ciclo máquina como la unidad básica de tiempo que utiliza el microcontrolador. Para el PIC16F84 el ciclo máquina equivale a 4

ciclos de reloj, por lo tanto, el tiempo que tarda en producirse un ciclo máquina es igual a cuatro veces el periodo del oscilador, como se lo puede observar en la figura 1.14.



1 CICLO MÁQUINA = 4 CICLOS DE RELOJ

Figura 1.14 Ciclo máquina para el PIC16F84

Las instrucciones en el microcontrolador PIC16F84 necesitan 1 ciclo máquina para ejecutarse, excepto las de salto *(goto, call, btfsc, return, etc)* que necesitan de dos ciclos máquina.

El tiempo que tarda el microcontrolador en ejecutar una tarea viene fijado por la siguiente fórmula:

Tiempo = 4(1/f)cm

Siendo:

f = la frecuencia del oscilador.

cm = el número de ciclos máquina que tardan en ejecutar la tarea.

A continuación se presenta un ejemplo practico para entenderlo de una mejor manera.

Ejemplo: Calcular la duración de 1 ciclo máquina para un PIC16F84 que utiliza un cristal de cuarzo de 4 MHz.

Solución:

$$\text{Tiempo} = 4(1/f)cm = 4(1/4MHz)1 = 1\mu s.$$

1.6.2 MEDIR TIEMPOS CON MAPLAB

Para calcular el tiempo de ejecución de un programa o de una subrutina, se puede contar el número de instrucciones que se realizan y multiplicarlos por 4 veces la frecuencia de la señal de reloj o por 8 en el caso de que las instrucciones sean de salto.

El MPLAB dispone de una opción de cronómetro denominada Stopwatch que permite medir el tiempo de ejecución de las instrucciones de los programas.

El cronómetro Stopwatch calcula el tiempo basándose en la frecuencia del reloj del microcontrolador PIC que se está simulando. Es necesario fijar previamente la frecuencia del oscilador empleado, para eso, se activa desde el menú *Debugger* > *Setting* > *Clock* tal como se muestra en la figura 1.15. Inmediatamente se abre un cuadro de diálogo donde se fija la frecuencia de reloj a 4 MHz.

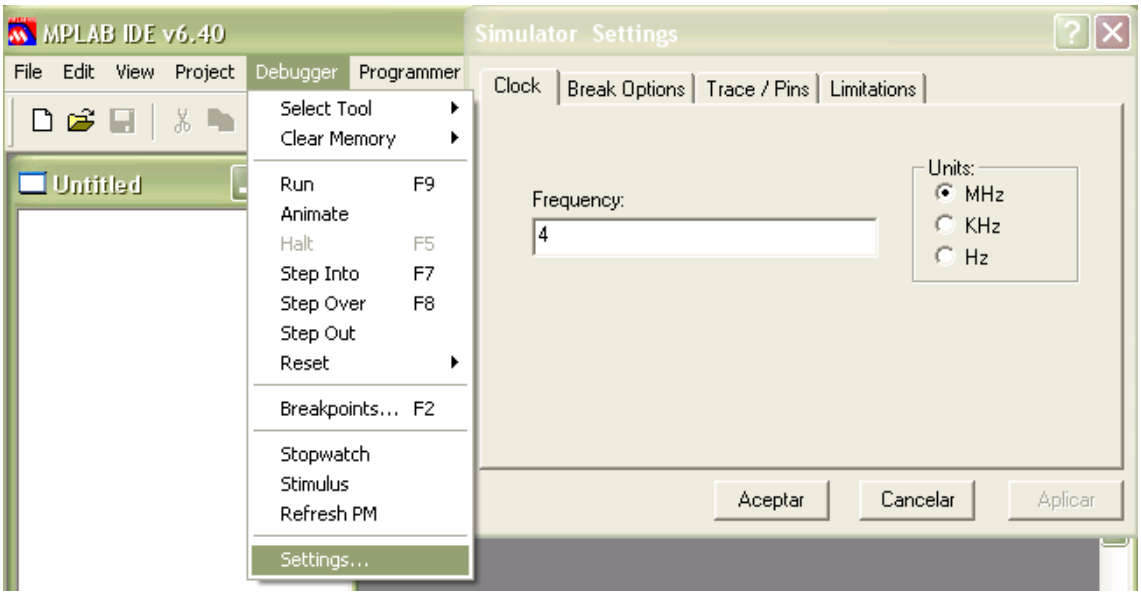


Figura 1.15 Selección de la frecuencia de simulación en el MPLAB SIN

Después se activa la opción *Debugger > Stopwatch,* con esto se consigue abrir la ventana que muestra el tiempo transcurrido y los ciclos máquina empleados en la ejecución de cada instrucción, como puede apreciarse en la figura 1.16.

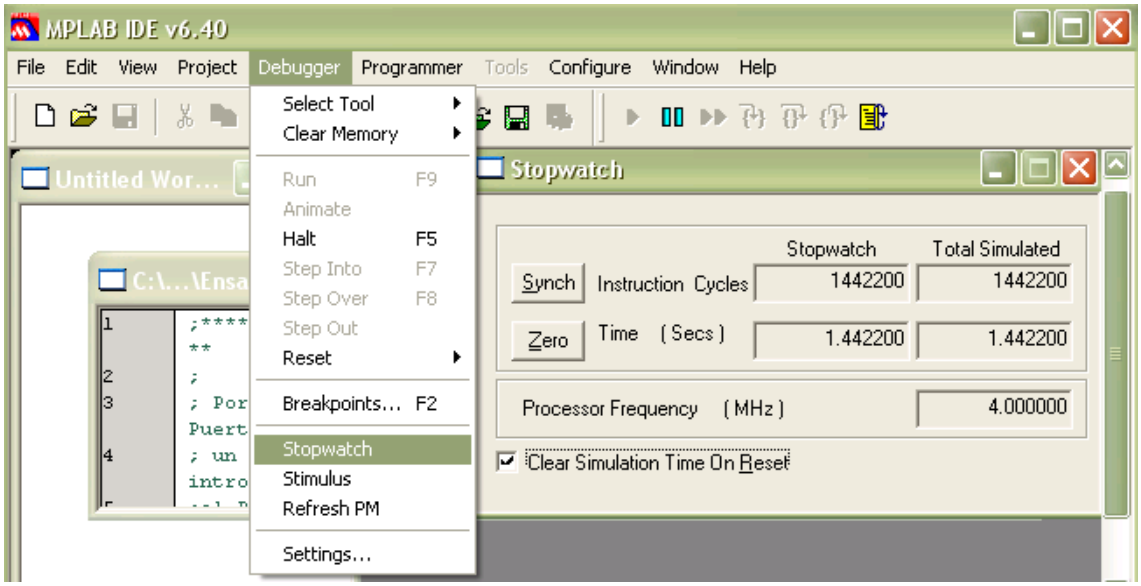


Figura 1.16 Ventana con el contenido del tiempo transcurrido

Es importante recordar que los simuladores de software no trabajan en tiempo real. Hay que tener en cuenta que el cronómetro del MPLAB trabaja mucho más lento que la realidad. Ahora bien, el tiempo que muestra será el tiempo real que tardan en ejecutarse las instrucciones.

1.6.3 INSTRUCCIÓNES “NOP”

Las instrucciones Nop (No Operation) no realiza operación alguna. En realidad, consume un ciclo máquina sin hacer nada.

La instrucción *nop* se utiliza para hacer gastar tiempo al microprocesador sin alterar el estado de los registros ni de los flags. Esta instrucción tarda 1 ciclo máquina en ejecutarse. Así, para un sistema con oscilador a cristal de cuarzo de 4MHz, tendrá una duración de 1µs (4 ciclos de reloj).

1.6.4 RETARDO MEDIANTE LAZO SIMPLE

En muchas aplicaciones con microcontroladores resulta necesario generar tiempos de espera conocidos como **tiempos de retardo**. Estos intervalos pueden conseguirse mediante una **subrutina de retardo**, basado en un lazo simple de algunas instrucciones que se repiten tantas veces como sea necesario, hasta conseguir el retardo pretendido, figura 1.17. Como el tiempo de ejecución de cada instrucción es conocido lo único que hay que hacer es calcular el valor inicial que debe tener el registro *R_ContA*, que actúa como el contador del número de interacción en el lazo, para obtener el tiempo de retardo deseado.

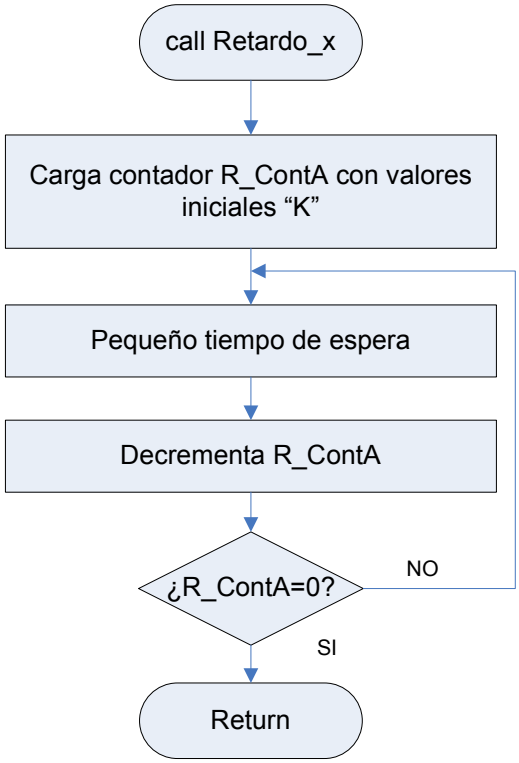


Figura 1.17 Estructura de una subrutina de retardo con un lazo simple

Un ejemplo típico de subrutina de retardo puede ser siguiente fragmento de programa, donde “cm” significa “numero de ciclos máquina” y en la figura 1.17 se muestra su diagrama de flujo.

```
Retardo 1ms                     ; La llamada “call” aporta 2 ciclos máquina.
     movlw   d”249”             ; Aporta 1 ciclo máquina. Éste es el valor de k.
     movwf  R_ContA             ; Aporta un ciclo máquina.
R 1ms_Bucle interno
     nop                        ; Aporta Kx1 ciclo máquina.
     decfsz  R_ContA,F          ; (K-1)x1cm (cuando no salta) +2cm (al saltar).
     goto     R1ms_BucleInterno  ; Aporta (K-1)x2 ciclos máquina.
     return                     ; El salto de retorno aporta 2 ciclos maquina.

; Em total esta subrutina tarda:
; 2+1+1+kx1+(K-1)x1+(K-1)x2+2=5+4K=1001 ciclos máquina (para K=249).
; 1001 ciclos máquinas suponen 1 milisegundo (1001us) para cristal de 4MHz.
```

En el ejemplo el bucle de las instrucciones *decfsz* R_ContA,F y *goto* R1ms_BucleInterno se repite tantas veces como determine el valor con el que

se carga R_ContA, en este caso 249. Cuanto mayor sea esta constante, mayor será el tiempo de retardo.

Es fácil deducir que el valor de la constante “K” con el que se ha cargado inicialmente el contador R_ContA vendrá dado por la siguiente ecuación, donde el tiempo viene expresado en µs:

$$Tiempo = 5+4k \qquad K= Tiempo-5/4$$

Para mayor entendimiento de esta fórmula se citá el siguiente ejemplo:

Calcular el valor de la constante K, para obtener una subrutina de retardo de 500µs.

Aplicando la ecuación se obtiene:

$$K = Tiempo – 5 / 4 \quad = \quad 500 – 5 / 4 \quad = \quad 123,7$$

Así pues se elige K=123, obteniendo un tiempo de retardo real de:

$$Tiempo = 5 + 4K \ = 5 +4 \ . \ 123 = 497µs$$

El ajuste fino para los 500us exacto se conseguiría añadiendo 3 instrucciones nop al principio de la subrutina de retardo.

1.7 MPLAB

1.7.1 PASOS CON MPLAB IDE

1. Con el explorador de Windows acceder a la carpeta de Mis Documentos y dentro de ella crear una carpeta que se llame “PIC16F84”, donde se irán guardando todos los programas que se vayan diseñando. La trayectoria absoluta o *path* del fichero no puede superar la longitud máxima de 62 caracteres, esto es importante tenerlo en cuenta si se trabaja en Windows 2000 o XP donde los *path* absolutos suelen ser bastante largos. Así pues,

el sub directorio donde se guardarán los ejercicios será del tipo “C:/Misdocumentos/PIC16F84” o similar.

2. Inicializar el programa actuando sobre el icono correspondiente a MPLAB situado en el escritorio. Se entrará en una pantalla similar a la figura 1.18.

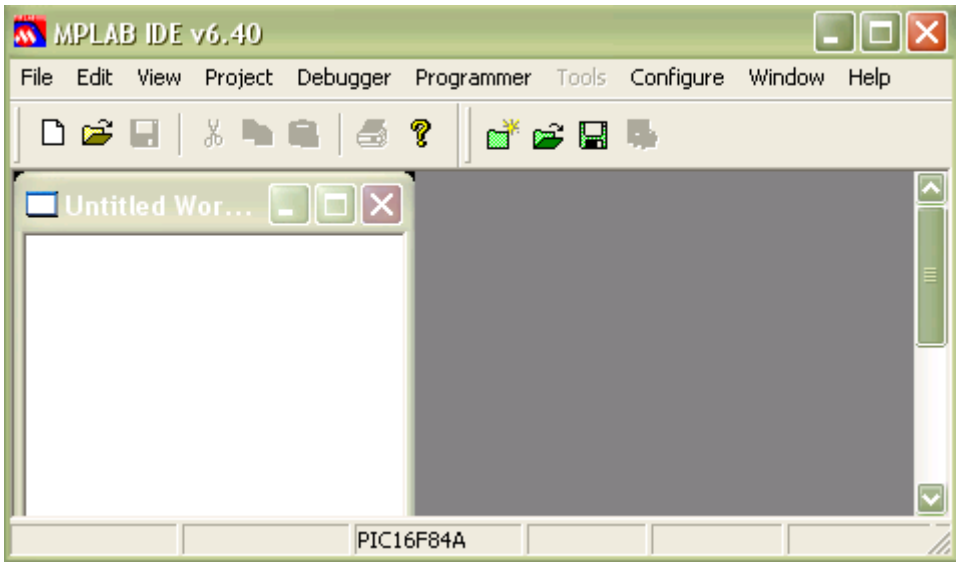


Figura 1.18 Pantalla de inicio del MPLAB IDE

A continuación se entra en la pantalla de edición, que al maximizar la hoja de trabajo queda como la figura 1.18.

3. Elegir el tipo de microcontrolador. Para ello acceder al menú *Configure* > *Select Device* y seleccionar PIC16F84, tal como se muestra en la figura 1.20.
4. A continuación es conveniente seleccionar el simulador, para ello activar el menú *Debugger* > *select Tool* > MPLAB SIM, tal como se muestra en la figura 1.21.
5. La frecuencia de trabajo del MPLAB SIM debe coincidir con la del circuito a simular. Para seleccionarlo acceder a *Debugger* > *Setting* > *Clock* y después comprobar que está a 4 MHz, tal como se muestra en la figura 1.22.
6. Crear un nuevo archivo fuente, para ello ir al menú *File* > *New* (ver en la figura 1.19). Se entrará en la pantalla de edición en blanco donde se puede escribir el primer programa.

7. A continuación se da el nombre al fichero fuente accediendo al menú *File > Save As…* Aparecerá un cuadro de diálogo que solicita el nombre del archivo. Se puede nombrar por ejemplo “Ensam_03.asm” y se guardará en la carpeta “C:/Misdocumentos/PIC16F84” creada anteriormente.

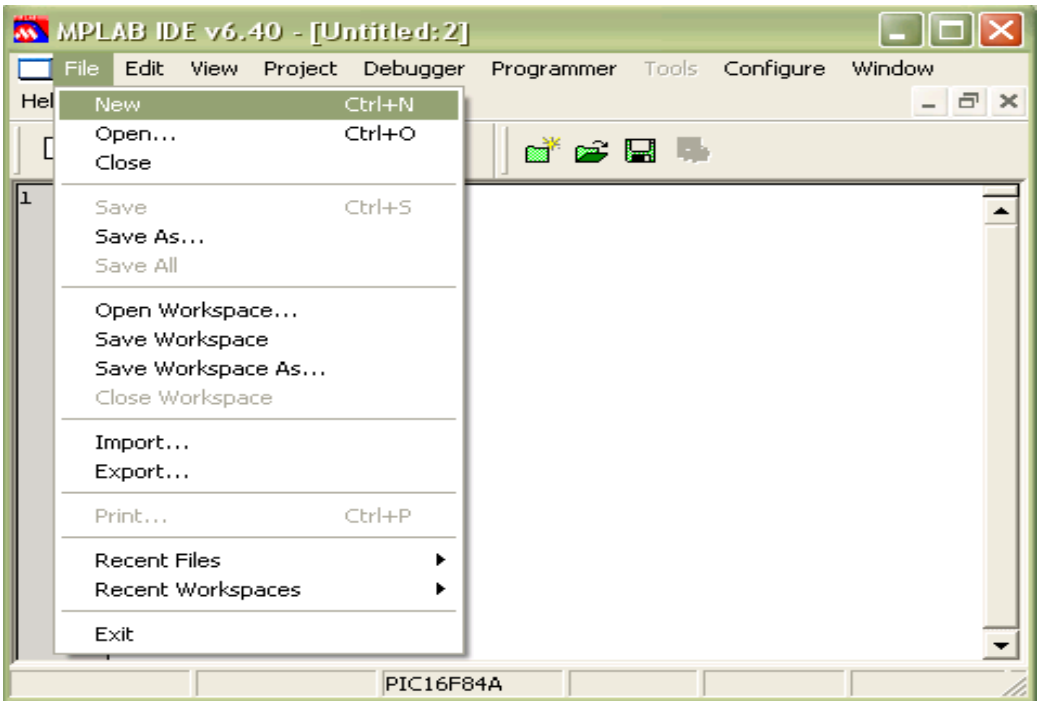


Figura 1.19 Pantalla de edición de programa en el MPLAB IDE

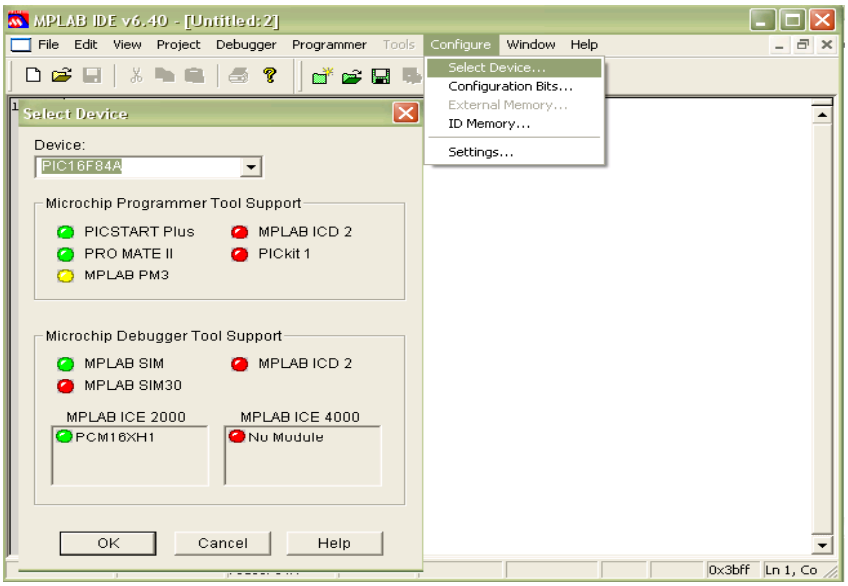


Figura 1.20 Selección del microcontrolador.

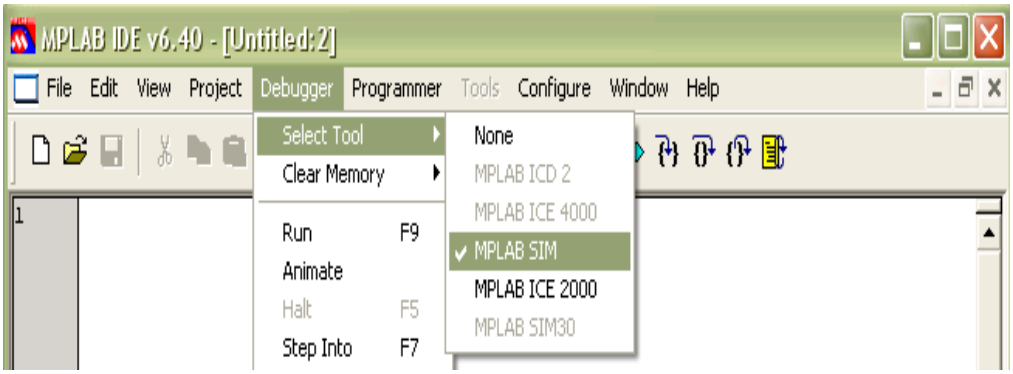


Figura 1.21 Selección del simulador

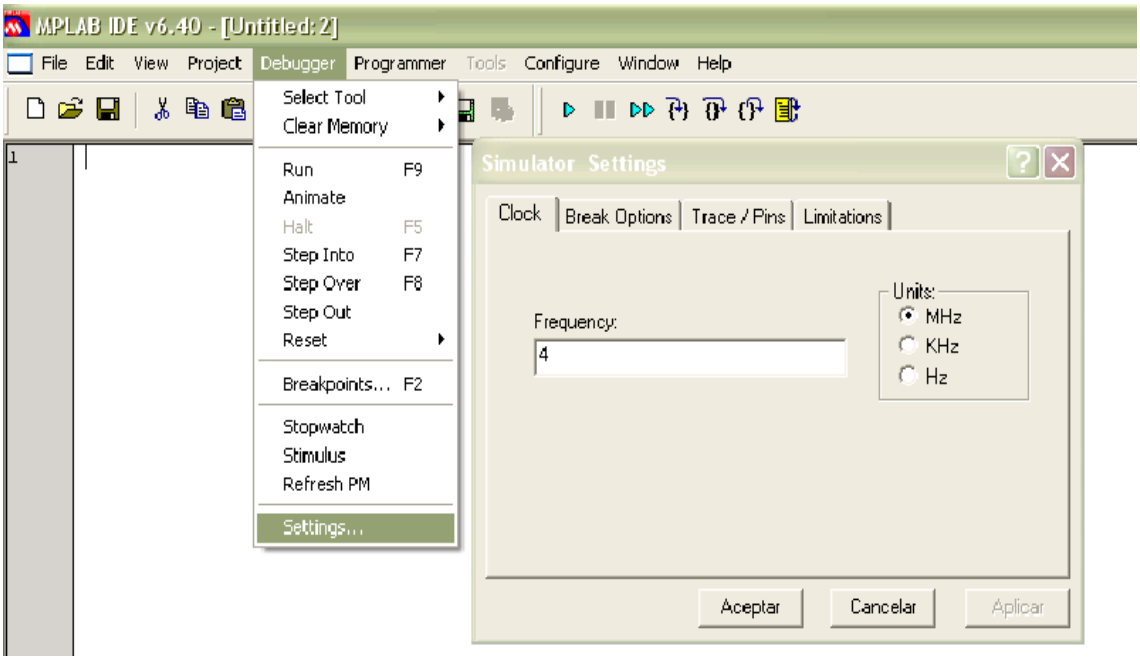


Figura 1.22 Selección de la frecuencia de simulación para el MPLAB SIM

8. Para trabajar con mayor comodidad es conveniente visualizar el número de cada línea. Para ello seleccionar el menú *Edit > Properties.* Dentro de la ventana *Editor Options* y pestaña *Editor* se activan las opciones que se indica figura 1.23.

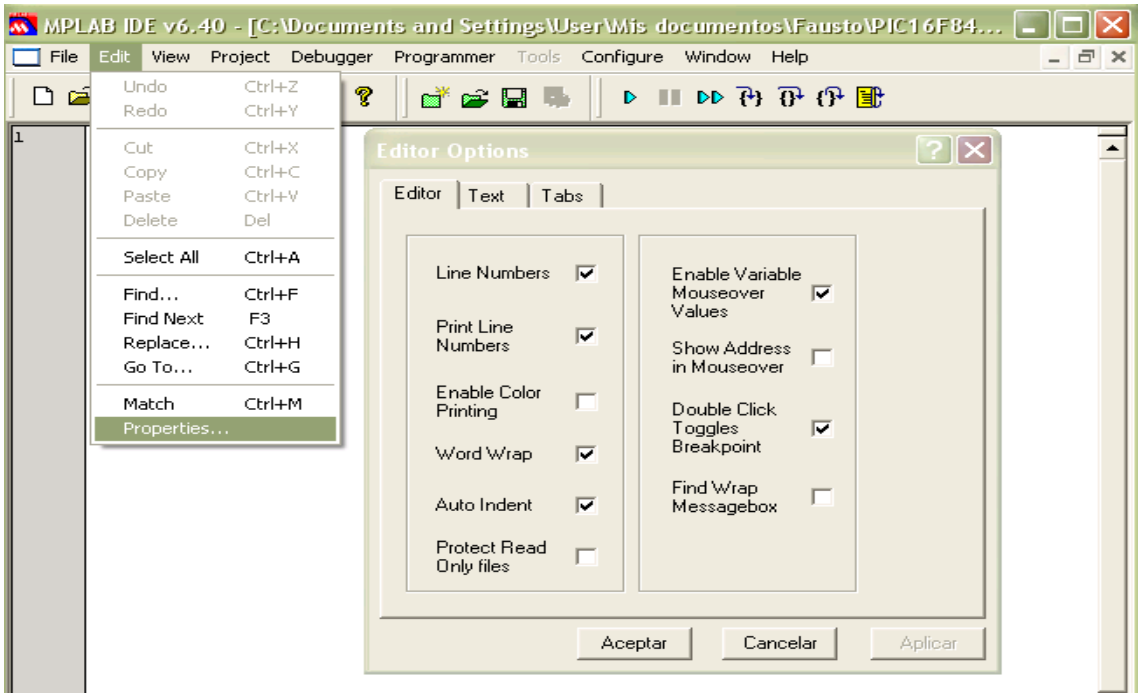


Figura 1.23 Propiedades de la pantalla de edición.

9. Elegir el tipo de letra, activado en el menú *Editor > Properties. Dentro de la ventana Editor Option* se activa la pestaña *Text* y se elige el tipo de letra seleccionada en la figura 1.24 o aquél otro tipo que al lector le resulte cómodo para su forma de trabajar.

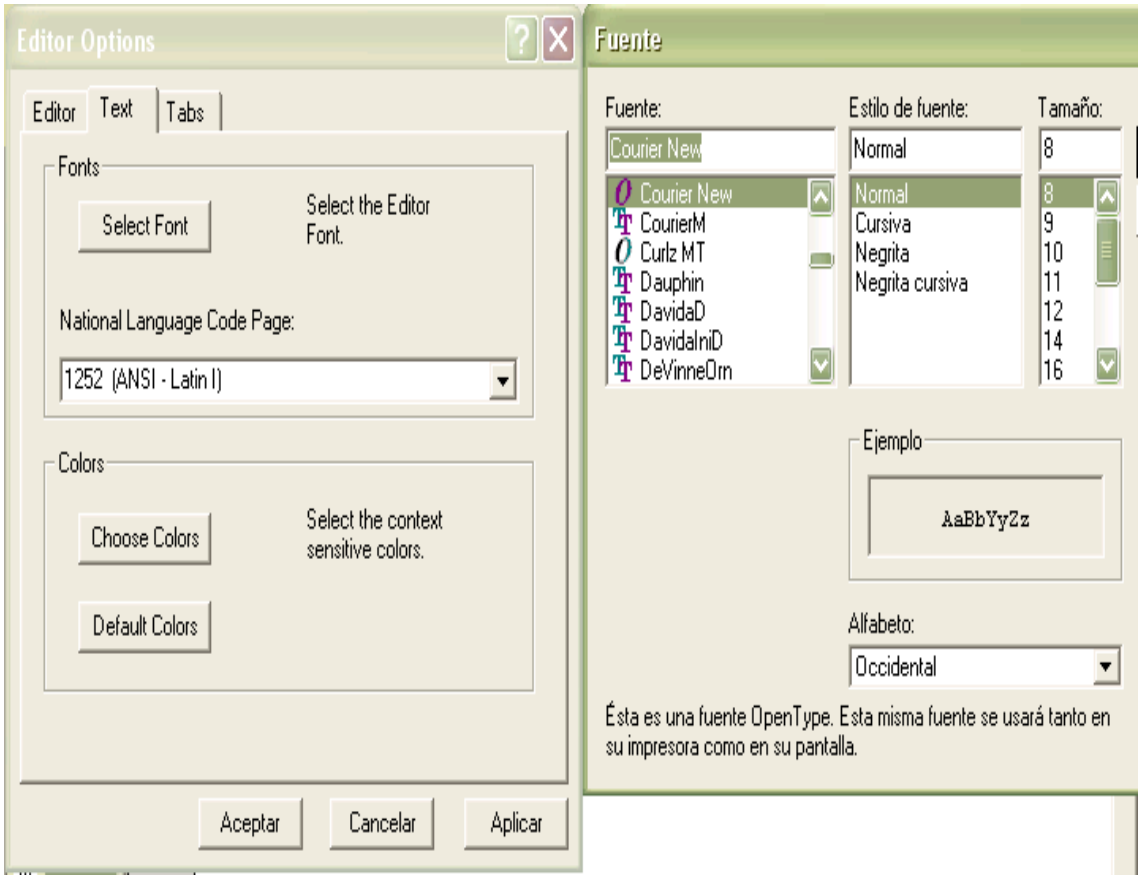


Figura 1.24 Elección del tipo de letra.

10. A continuación hay que seleccionar la tabulación deseada, activando el menú *Edit > Properties. Dentro de la ventana Editor Options se activa la pestaña Tabs* y se elige el valor indicado en la figura 1.25 o aquel que se crea conveniente.

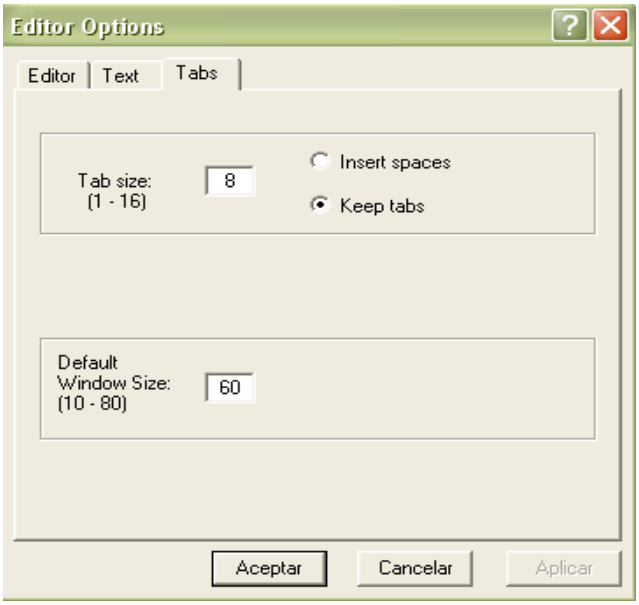


Figura 1.25 Elección de la tabulación.

11. Es conveniente dejar los colores de los caracteres configurados por defecto. Dentro de la ventana *Editor* > *Properties.* Dentro de la venta *Editor Option* se activa la pestaña *Text* y el botón *Default Colors* (figura 1.24).

12. Es cómodo que cada vez que se abra el MPLAB aparezca el último programa con el que ha trabajado. Para ello, hay que activar el menú *Configure* > *Setting* > Workspace y activar la casilla *Reload last workspace at startup* (figura 1.26).

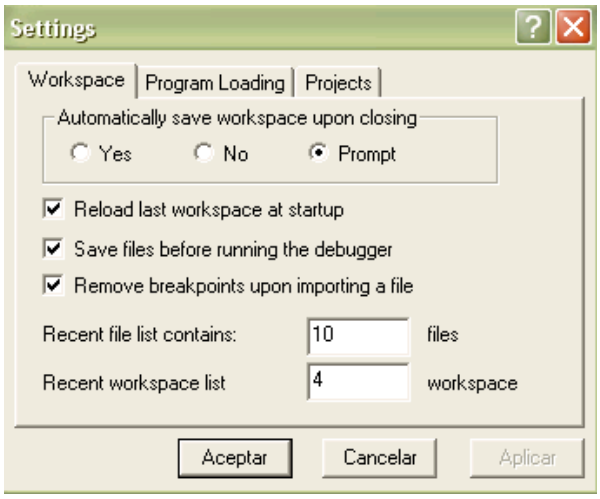


Figura 1.26 Configurar la recarga del último trabajo realizado al abrir el MPLAB.

13. En la pantalla de edición se procede a escribir el programa.
    Es importante recordar que "_ _ *CONFIG"* se inicia con dos subrayados (guiones bajos), no con uno.
14. A continuación el programa se ensambla y simula tal como se explica en los próximos apartados.
15. Una vez simulado el programa y corregido todos los errores se puede salir del MPLAB por el método habitual en Windows, activando para ello el menú *File > Exit.*

1.7.2 ENSAMBLADO DEL PROGRAMA

Una vez terminado de editar el programa hay que proceder a ensamblar el archivo fuente Ensam_03.asm. Para ello, hay que seleccionar el menú:

Project > Quickbuild Ensam_03.asm,

o mejor combinar con la combinación de teclas Alt+F10 (figura 1.27).En esta etapa se realiza en forma automática el ensamblado del archivo en forma automática el ensamblado del archivo fuente y el traspaso de éste a la memoria de simulación.

Figura 1.27 Ensamblar el archivo fuente

Momentáneamente aparecerá una ventana indicando el proceso de ensamblado (figura 1.28).

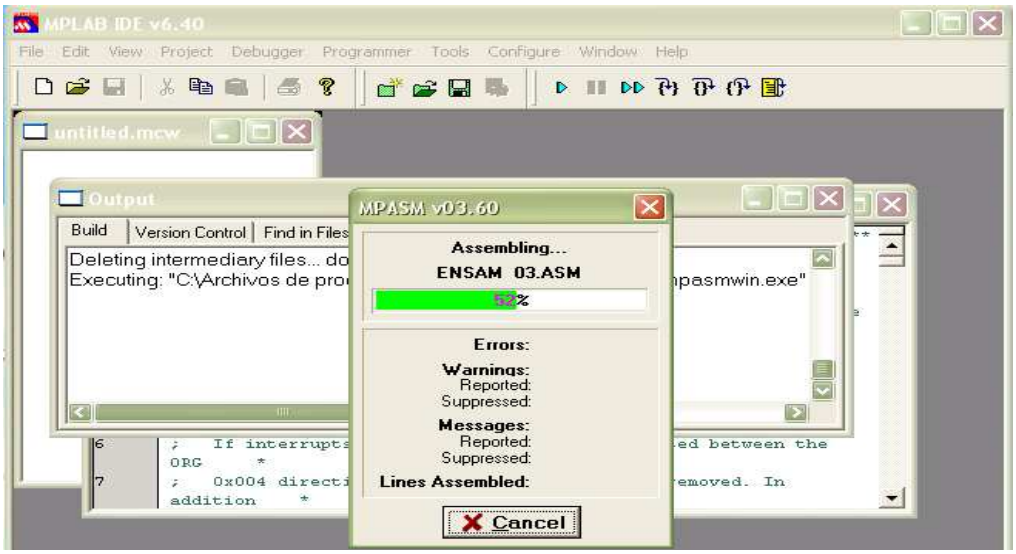


Figura 1.28 Proceso de ensamblado

Al finalizar el ensamblado, aparecerá una pantalla MPLAB – *Output* tal como muestra la figura 1.29, en la que indicará la ocurrencia de uno de estos dos casos:

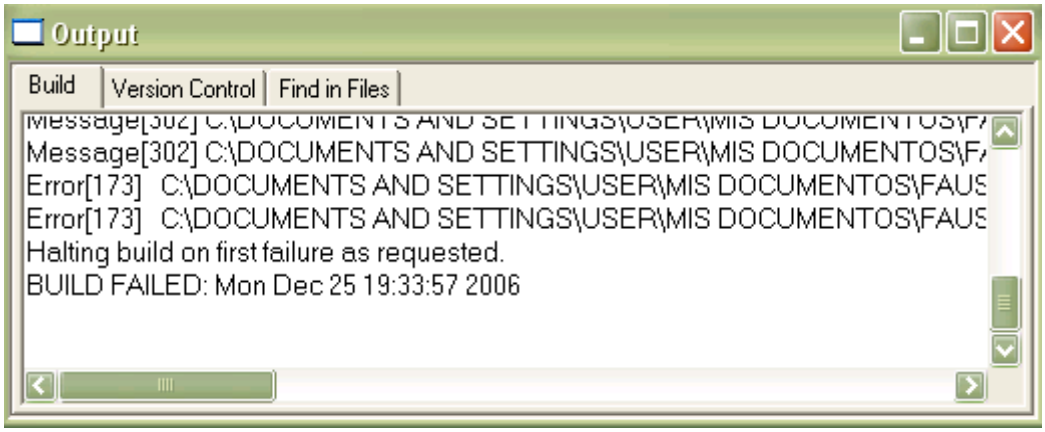


Figura 1.29 Pantalla final de proceso de ensamblado con éxito

- Si al final de esta pantalla indica “BUILD SUCCEDED” se confirma que el ensamblado se ha producido con éxito. Por lo tanto, ya se está en condiciones de pasar a la simulación. En esta pantalla pueden aparecer algunos mensajes de aviso “*Message*” , que llaman la situación sobre situaciones a tener en cuenta y que podría ocasionar un error en el programa pero que no impide el correcto ensamblado.

- Si al final de esta pantalla indica *"BUILD FAILED"* se advierte de la ocurrencia de errores. El proceso de ensamblado ha generado un archivo de error con descripción de los mismos. Si se hace doble clic sobre la línea que muestra el error el cursor saltará directamente a la línea de código donde se encuentra éste. Una vez subsanados los errores hay que volver a ensamblar el archivo fuente.

1.7.3 FICHERO HEXADECIMAL RESULTANTE

El proceso de ensamblado produce un fichero ejecutable con extensión (*.hex) que será el que posteriormente se grabará en la memoria de programa del PIC mediante el grabador. Este fichero puede ser analizado seleccionando el menú *File > Open* y dentro de los tipos de archivo *All Files (*.*)* se ha de elegir el *Ensam_03.HEX,* tal como se muestra en la figura 1.30.

Figura 1.30 Abrir el archivo resultado del ensamblado en código maquina.Hex*

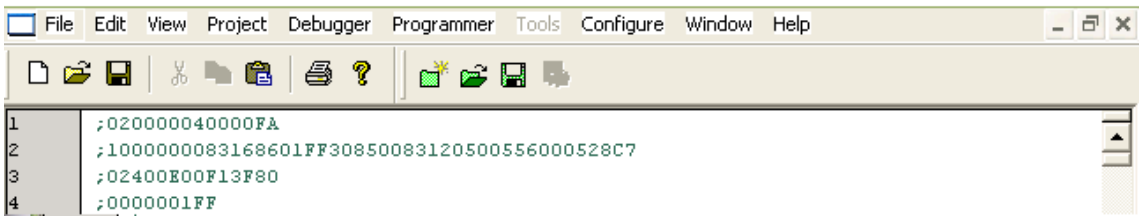


Figura 1.31 Contenido del archivo resultado del ensamblado del código maquina

En este archivo Ensam_03.hex únicamente contiene números hexadecimales, que es la forma de representar los ceros y unos binarios de la

información que se grabará posteriormente en la memoria de programa del microcontrolador.

1.7.4 VENTANA DE VISUALIZACIÓN

Una vez ensamblado el código fuente el entorno ya está preparado la simulación del programa. Para que este trabajo sea más eficaz conviene activar las ventanas que indica el estado de todas las memorias y registros del sistema. Las principales ventanas de simulación se encuentra dentro del menú VIEW y son:

- *Disassembly Listin.* Código máquina y archivo fuente.
- *File Registers.* Memoria de programa.
- *Program Memory.* Memoria de programa.
- *Special Function Registers.* Registro del SFR.
- *Watch.* Ventana personalizada.

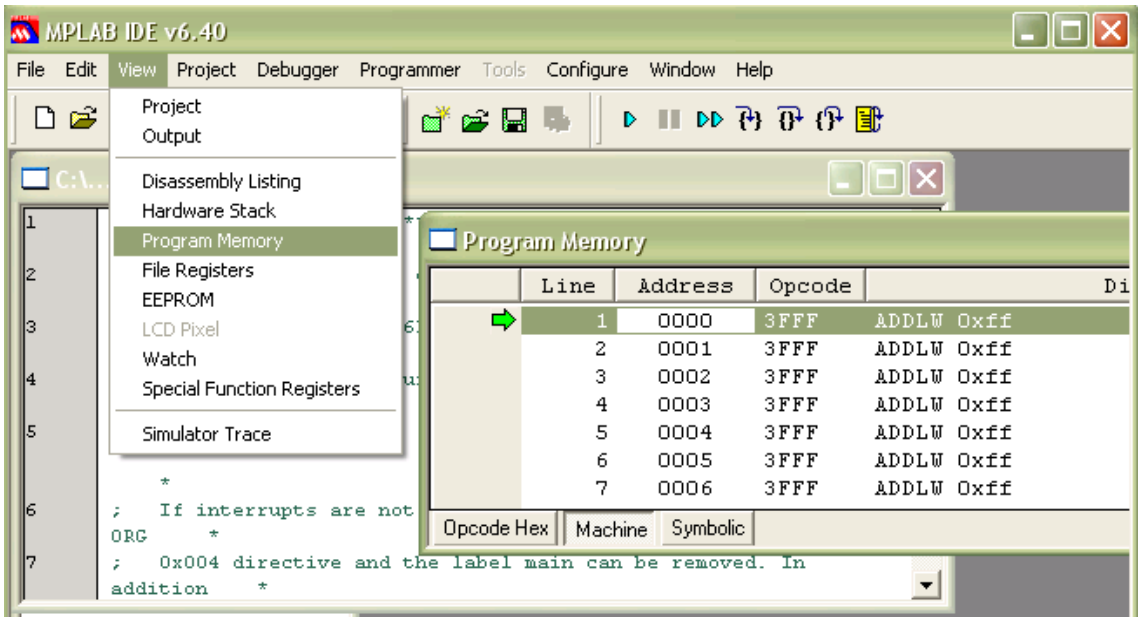


Figura 1.32 Ventana de visualización de la memoria de programa.

1.7.5 VENTANA DE VISUALIZACIÓN DE LA MEMORIA DE PROGRAMA

En esta ventana se aprecian las posiciones de memoria que ocupa cada una de las instrucciones, el código de operación de cada instrucción y la dirección de memoria de programa que se le ha asignado a cada etiqueta (figura 1.32). Se encuentra en ella actividad el menú *Viuw > Program Memory.*

1.7.6 VENTANA DISASSEMBLY

Esta ventana incluye el archivo fuente. Se entra en ella activando el menú *View> Disassembly Listing,* como se ve en la figura 1.33.

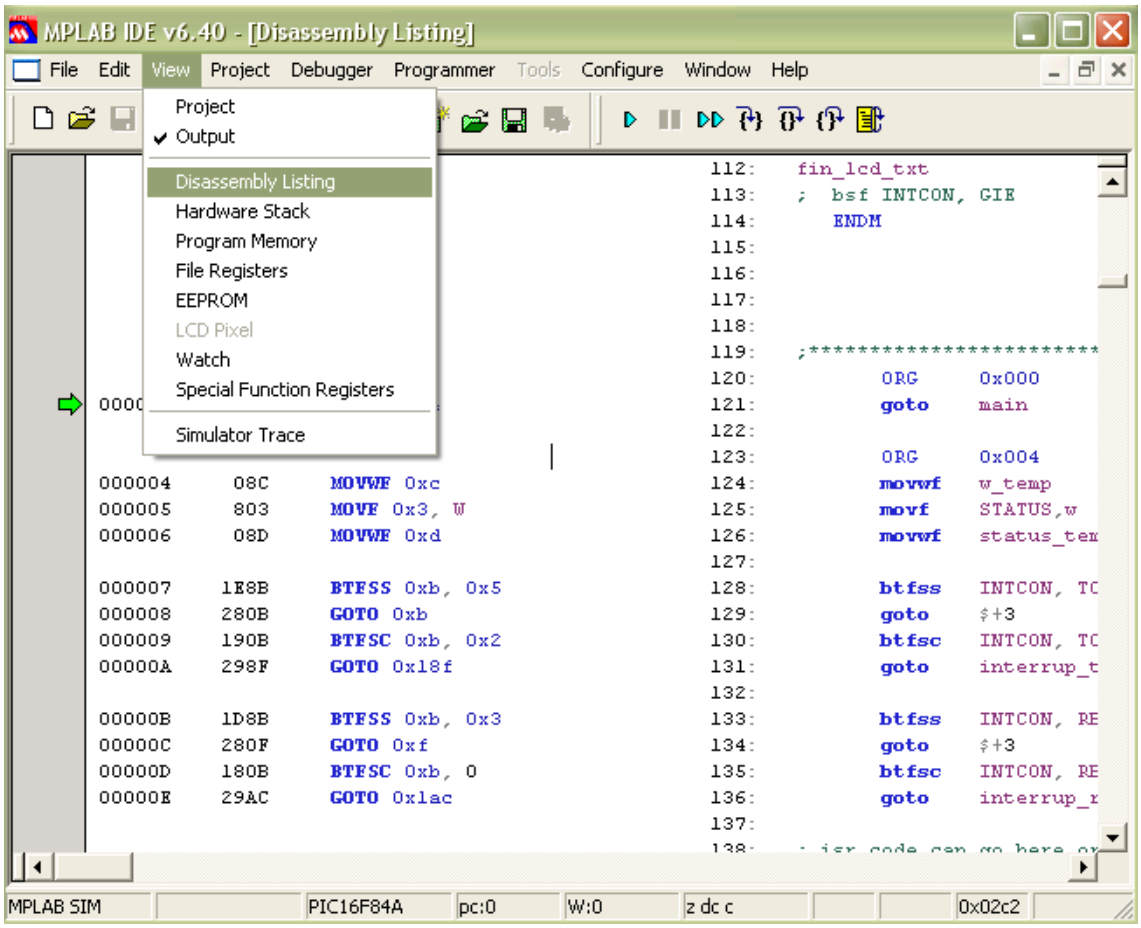


Figura 1.33 Ventana Disassembly

1.7.7 VENTANA DE VISUALIZACIÓN DE LOS REGISTROS DEL SFR

Para visualizar los registros especiales del SFR se activa *View* > *Special Function register* (figura 1.33).

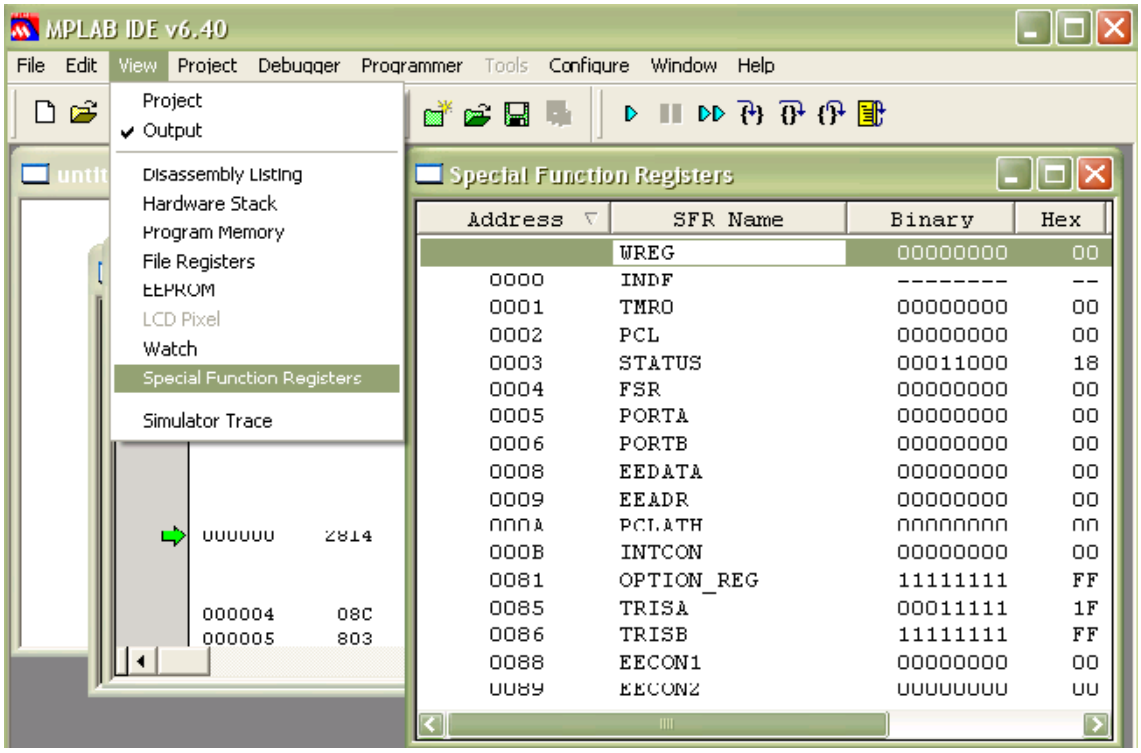


Figura 1.34 Ventana de visualización de los registros especiales

Para modificar manualmente uno de estos registros hay que hacer doble clic en la fila del registro correspondiente sobre alguna de las columnas *Hex, Decima, Binary,* o *Char* y modificarlo. Esto no es válido para los puertos que actúen como entrada, en cuyo caso hay que utilizar, dentro del menú, *Debugger,* la opción *Stimulus*.

En esta ventana suele ser interesante situar la columna *Binary* a continuación de la columna *SFR Name.* Para ello, pulsar el botón derecho del ratón y elegir la opción *Properties,* saldrá una ventana como la que se muestra en la figura 1.35. Señalando la casilla *Binary* y pulsado sobre el botón *Move Up,* la columna *Binary* se desplazará hasta situarse en la posición deseada tal como se explica en la figura 1.35.

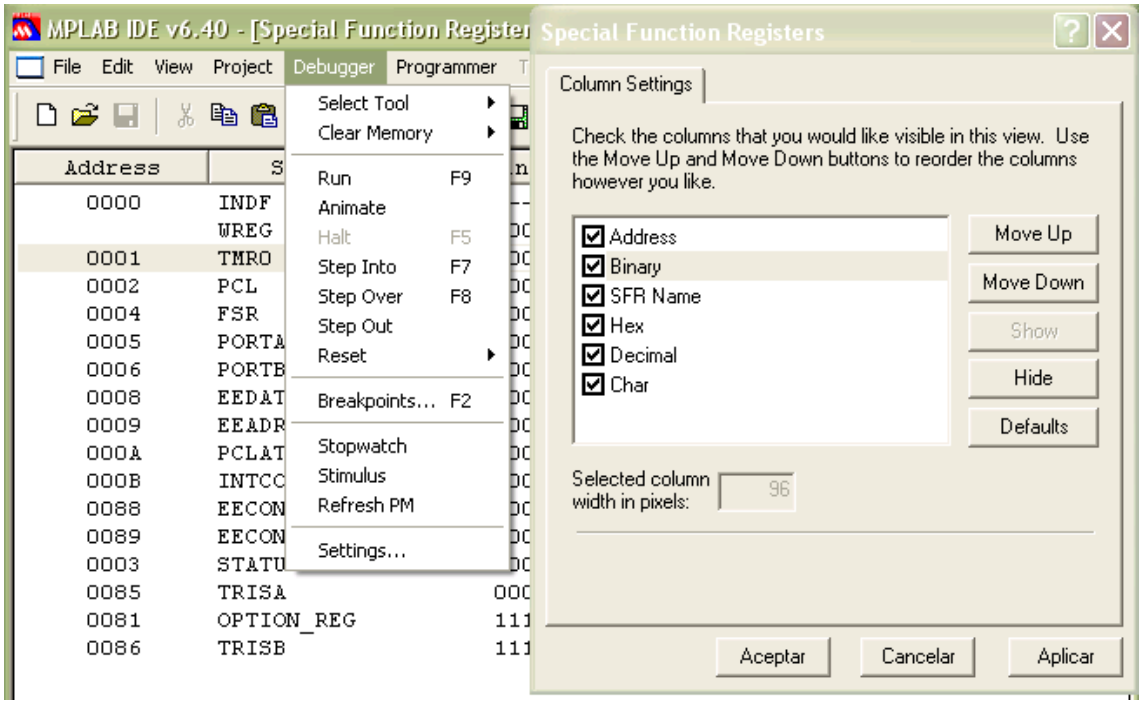


Figura 1.35 Desplazar la columna Binary dentro ventana de visualización del SFR

1.7.8 VENTANA DE CONTENIDO DE LA MEMORIA RAM

Esta ventana presenta una lista con todos los registros generales del microcontrolador simulado figura 1.36. Para visualizar la ventana de contenido de la memoria RAM de datos hay que seleccionar *View > File Register.*

Activando el botón inferior *Symbolic* se puede visualizar el nombre simbólico que le ha dado el programador a las diferentes posiciones de memoria RAM de usuario.

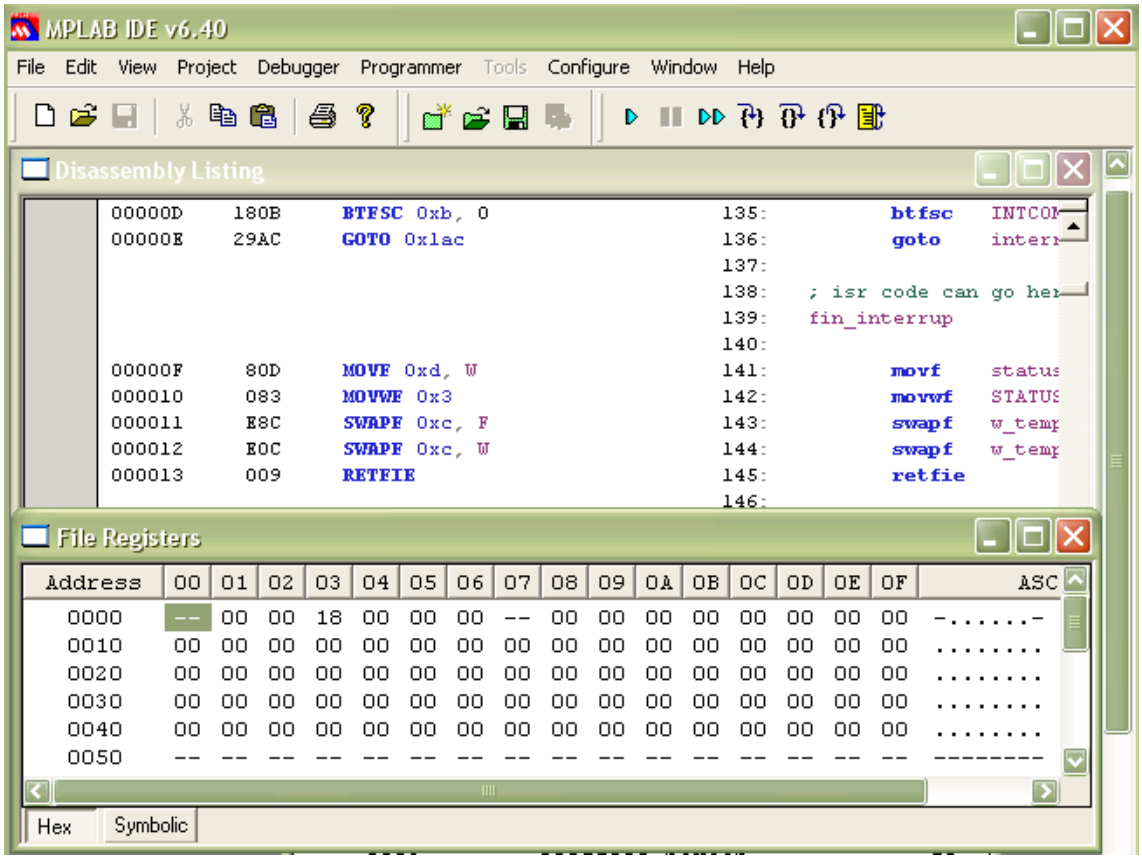


Figura 1.36 Ventana con el contenido de la memoria RAM de datos

1.7.9 SIMULACIÓN BÁSICA

Tras el proceso de ensamblado se procede a la simulación del programa. Mientras se ejecuta la simulación del programa es interesante visualizar el contenido de las ventanas explicadas antes y comprobar el efecto de cada una de ellas.

Es conveniente antes de nada, comprobar que está cargado correctamente el MPLAB SIM.

Los cinco comandos más importantes para la simulación se localizan dentro del menú *Debbuger* y se muestra en la figura 1.37.

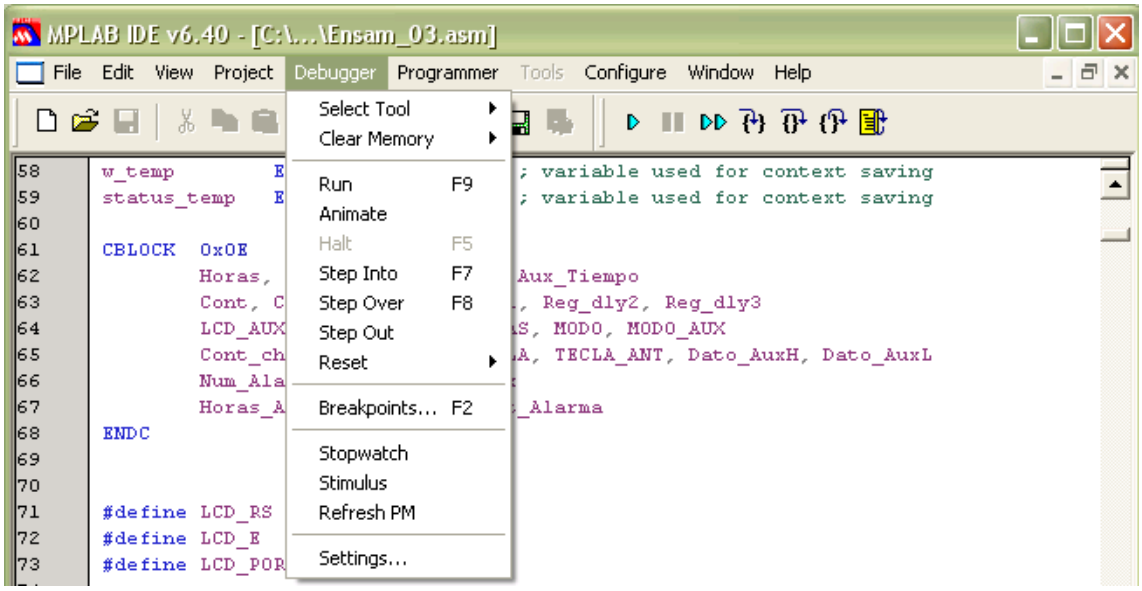


Figura 1.37 Menú para entrar en los comandos básicos de simulación

- *Run.* Modo de **ejecución continua.** Ejecuta el programa constantemente. Las ventanas abiertas en el paso anterior no se actualiza hasta que no se produce una parada. Es la forma más rápida de simular el programa, pero no se "ve" ni como evoluciona la memoria ni los registros. En este modo se entra seleccionando *Debugger* > *Run* o pulsando la tecla F9, también al pulsar sobre el icono correspondiente de la barra de herramientas (flecha azul).

- *ANIMATE* (o tecla ctrl.+F9). Modo de **ejecución animada.** Ejecuta el programa de forma continua pero actualizando todas las ventanas cada vez que se ejecuta una instrucción . Es más lento que el modo "Run" pero permite ver como va cambiando los registros. Tal vez sea el modo de ejecución más útil y recomendable. Se entra en este modo seleccionando *Debugger* > *Animate,* o también al pulsar sobre el icono correspondiente de la barra de herramientas (doble flecha).

- *HALT.* **Paro.** Para la ejecución del programa y actualiza todas las ventanas. Se consigue seleccionando *Debugger* > *Run* o pulsando la tecla F5. También se entra en este modo al activar el icono correspondiente de la barra de herramientas (dos barras verticales azulez).

- *STEP INTO.* Ejecución paso a paso. Ejecuta una sola instrucción del programa cada vez actualizando los valores de la ventana. Es la forma más lenta de simulación pero se comprueba fácilmente como van evolucionando todos los registros y memorias, siendo muy fácil detener los posibles errores. En este modo se entra seleccionando *Debugger* > *Step Into* o pulsando la tecla F7. También pulsando sobre el icono correspondiente de la barra de herramientas.

- *RESET.* Equivale a un **reset** por activación del pin **MCLD.** En este modo se entra seleccionando *Debugger* > *Reset* o pulsando la tecla F6. También si se pulsa sobre el icono correspondiente de la barra de herramientas.

- *RUN TO CURSOR.* **Ejecución hasta la posición actual** del cursor. Para entrar en este modo de simulación, el cursor debe situarse en la línea donde ésta la instrucción hasta donde quiere simular el programa, pulsar el botón derecho del ratón y activar la operación *Run to cursor* tal como indica la figura 1.38.

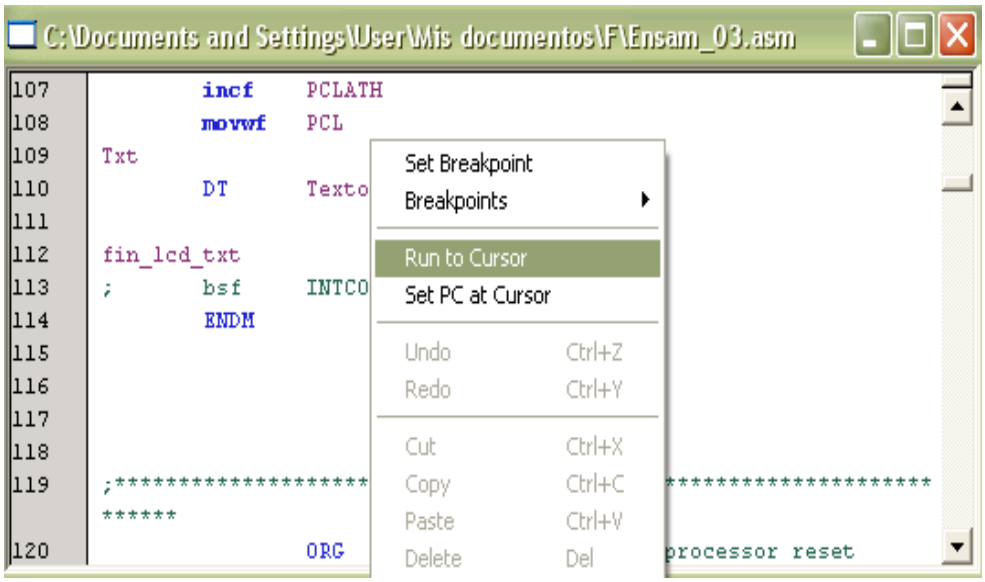


Figura 1.38 Modo de simulación "Run to Cursor"

1.7.10 SIMULACIÓN MEDIANTE BREAKPOINTS Y TRAZA

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

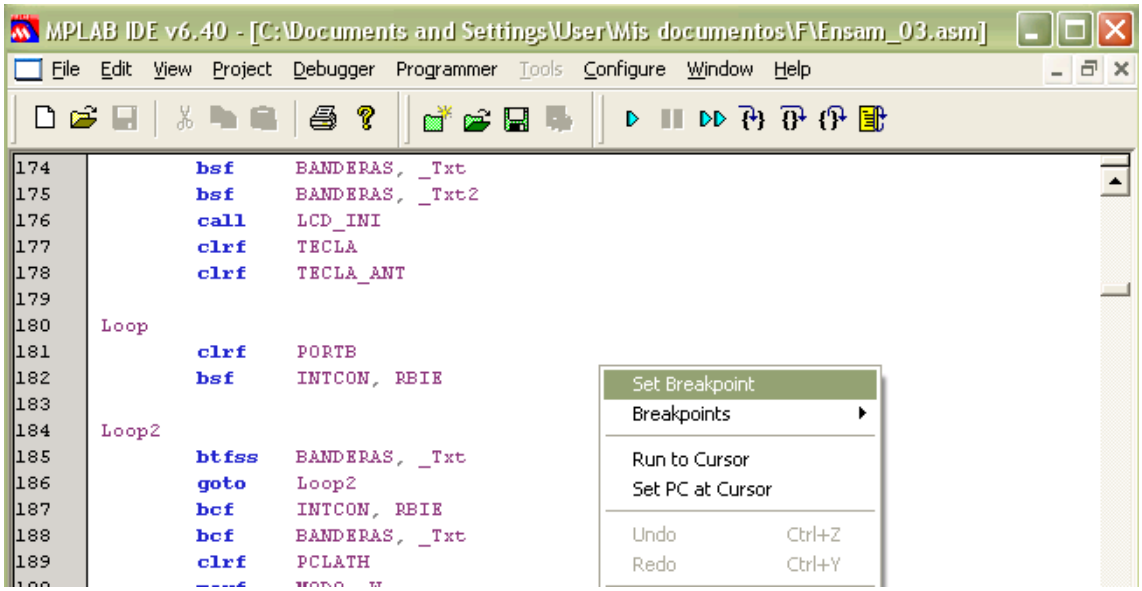


Figura 1.39 Situar un *Breakpoint*

Para situar un *Breakpoint* sobre una línea señalada por el cursor se pulsa el botón derecho del ratón, de manera que aparece el menú desplegable, como en la figura 1.39.

Selecciona *Set Breakpoint* y aparecerá sobre el programa una “B” en rojo en la posición donde ha situado el punto de paro. Otra forma de situar o eliminar un *Breakpoint* es realizando una doble posición con el ratón sobre el número de línea donde se quiere situar el punto de parada.

La ventana de **memoria de traza** es una herramienta que ayuda a simular los programas (figura 1.40). El *Simulate Trace* toma “una instantánea” de la ejecución del programa. En el simulador de buffer de traza o memoria de traza es útil para visualizar un registro a lo largo de la ejecución del programa, de manera que se puede registrar por dónde pasa el programa y después

analizarlo. El simulador toma datos desde la última vez que se pulsó *Run* o *Animate* hasta que se detiene la simulación del programa (normalmente hasta un *Breakpoint*).

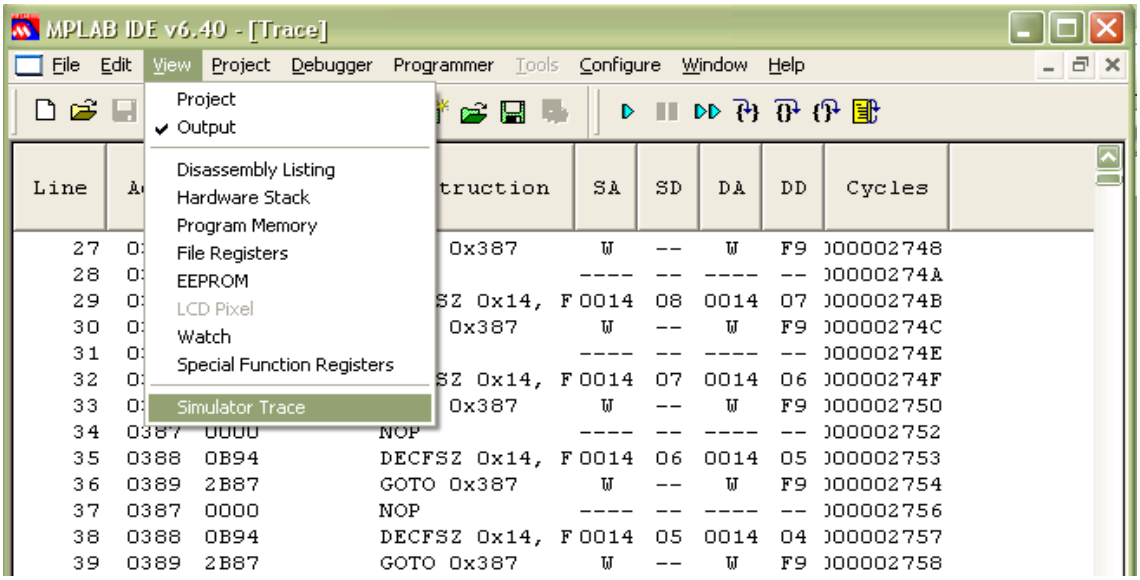


Figura 1.40 Simulación de traza

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

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

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

1.7.11 SIMULACIÓN DE ENTRADAS

Una de las operaciones más habituales de cualquier simulación consiste en variar los valores de las líneas de entrada. A esto se denomina “estimular” la entrada. Para cambiar los estimulos de una entrada de un puerto hay que seleccionar el menú *Debugger* > *Stimulus.* En la ventana que aparece, selecciónа la pestaña *Pin Stimulus* (figura 1.41).

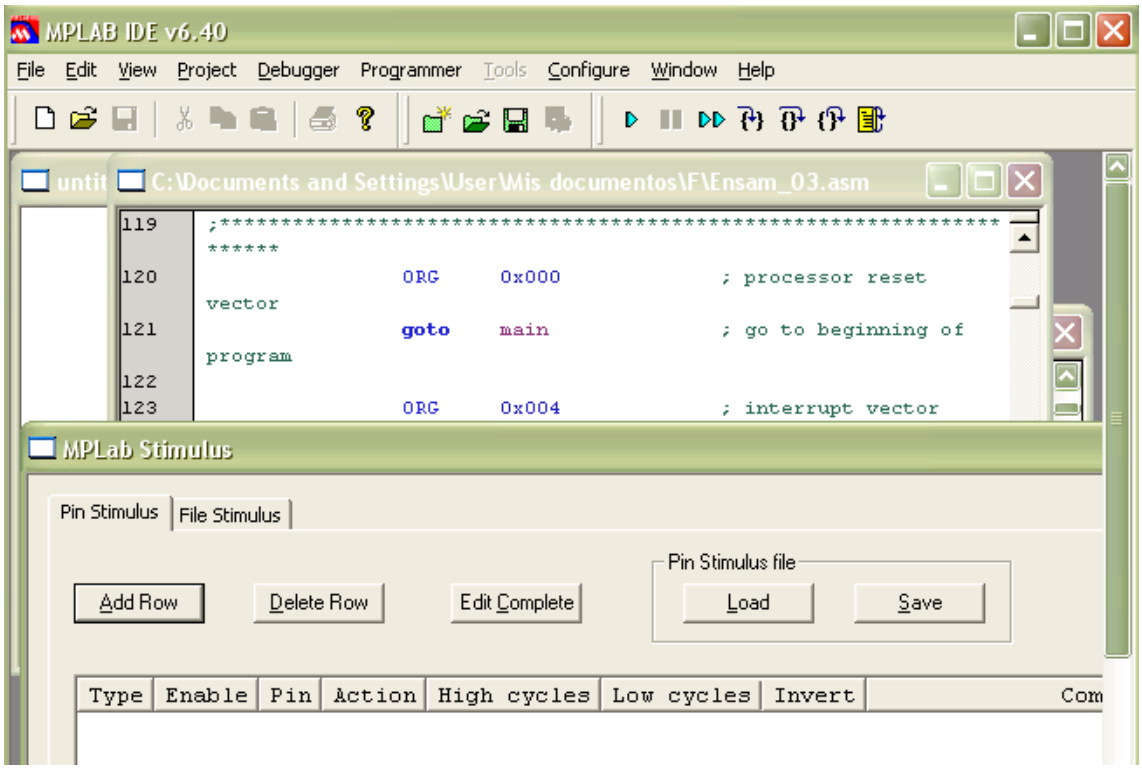


Figura 1.41 Menú para entrar en la ventana de estímulos

Si pulsa sobre *Add Row* , se irán añadiendo diferentes filas. Cada una de estas filas corresponde a un estímulo sobre una línea de entrada. La forma de editarlo es pulsar sobre la casilla correspondiente y seleccionar la patilla a la

que se quiere vincular y l tipo de cambio que se desea realizar con ese pin para cada pulsación.

- High, poner la entrada a “1”.
- Low, poner la entrada a “0”.
- Toggle, cambiar de valor cada vez que se pulse. Ésta es la más habitual.
- Pulse, cambia el estado del pin y retorna de nuevo a su valor actual.

La figura 1.42 muestra como se ha configurado para las cinco líneas del puerto A, como entrada y en modo *Toggle.*

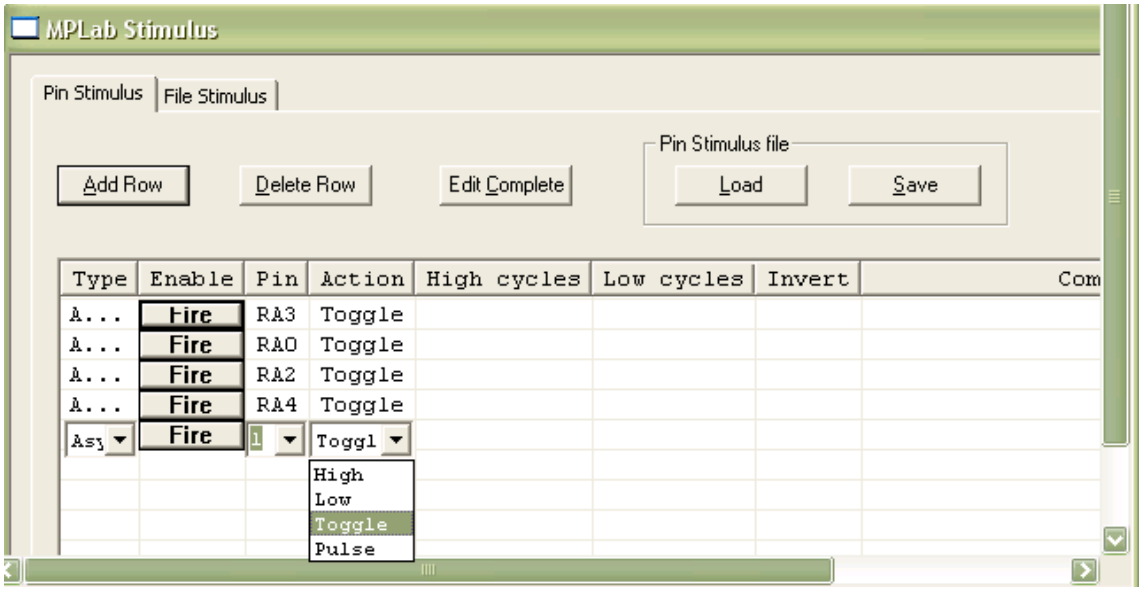


Figura 1.42 Configurar los estímulos para el Puerto A como entrada y modo Toggle

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

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

1.7.12 GRABACIÓN CON EL ARCHIVO HEXADECIMAL

Una vez simulado el programa y comprobado que funciona correctamente, es tiempo de grabar físicamente dicho programa en el PIC16F84. Con el ensamblado del archivo fuente se ha generado un archivo ejecutable con los códigos máquinas a grabar en el PIC16F84. Este archivo tiene extención *.hex y se graba en el PIC16F84A con ayuda de un grabador y el sotware asociado. Para ello deben seguir los siguientes pasos:

- Conectar el grabador al ordenador. Insertar el PIC16F84 en el zócalo correspondiente, teniendo en cuenta la orientación correcta del chip guiándose por la muesca de la cápsula.
- Abrir el programa y comprobar que está correctamente configurado.
- Abrir el archivo co extención *.hex que contiene los datos a programar en el PIC16F84A. Para ello, seleccionar el menú *Archivo* > *Abrir archivo* y, una vez dentro de la carpeta apropiada, elegir el fichero a grabar.
- Comprobar que los datos se han cargado en el área de *Código de programa.*
- Proceder a la grabación física del chip.
- Una vez grabado el PIC16F84 se debe extraer del grabador y comprobar su correcto funcionamiento dentro del circuito.

1.8 SOFTWARE DE GRABACIÓN IC-PROG

El **IC-Prog** es uno de los softwares más populares para la grabación de microcontroladores PIC. Permite la programación de muchos dispositivos y está probado con numerosos programadores. Es de libre distribución y en la página Web www.ic-prog.com se puede descargar y recoger toda la información de uso.

Una vez descargado, la instalación de este software es muy sencilla, basta con descomprimir el fichero *icprog.zip* y seguir el procedimiento usual en

Windows. Este archivo consta del fichero *icprog.exe*, que contiene todo el código necesario para su funcionamiento, con versiones para cualquier sistema operativo Windows. En caso de utilizar este software con Windows XP, 2000 o NT, es necesario descargar el archivo *icprog.sys* de la misma Web y situarlo en la misma carpeta, junto con el *icprog.exe.*

1.8.1 PROCESO DE GRABACIÓN

Antes de nada hay que conectar el programador a uno de los puertos serie COM disponibles en el ordenador. A continuación se inserta el microcontrolador PIC en el zócalo del programador respetando la correcta orientación de la cápsula.

Una vez que el programa esté correctamente instalado, los pasos a seguir para trabajar con el *IC-Prog 1.05C* son los siguientes:

1. La primera vez que se ejecuta entrará en una pantalla de presentación de la que sale aceptando todas las opciones por defecto. Seguidamente aparece una pantalla en inglés, similar a la figura 1.43, donde se presenta toda la información necesaria para programar el dispositivo. Esta pantalla posee, al menos:

    - Un área de código *(Program Code)*, donde se almacena la información a grabar. La columna de la izquierda contiene la dirección física de la memoria del dispositivo, (*Address)*. En el centro del campo se presenta el valor hexadecimal y la columna de la derecha contiene la misma información en código ASCII.
    - Un área de configuración (*Configuration),* donde se indica el valor de algunos parámetros necesarios para la correcta grabación

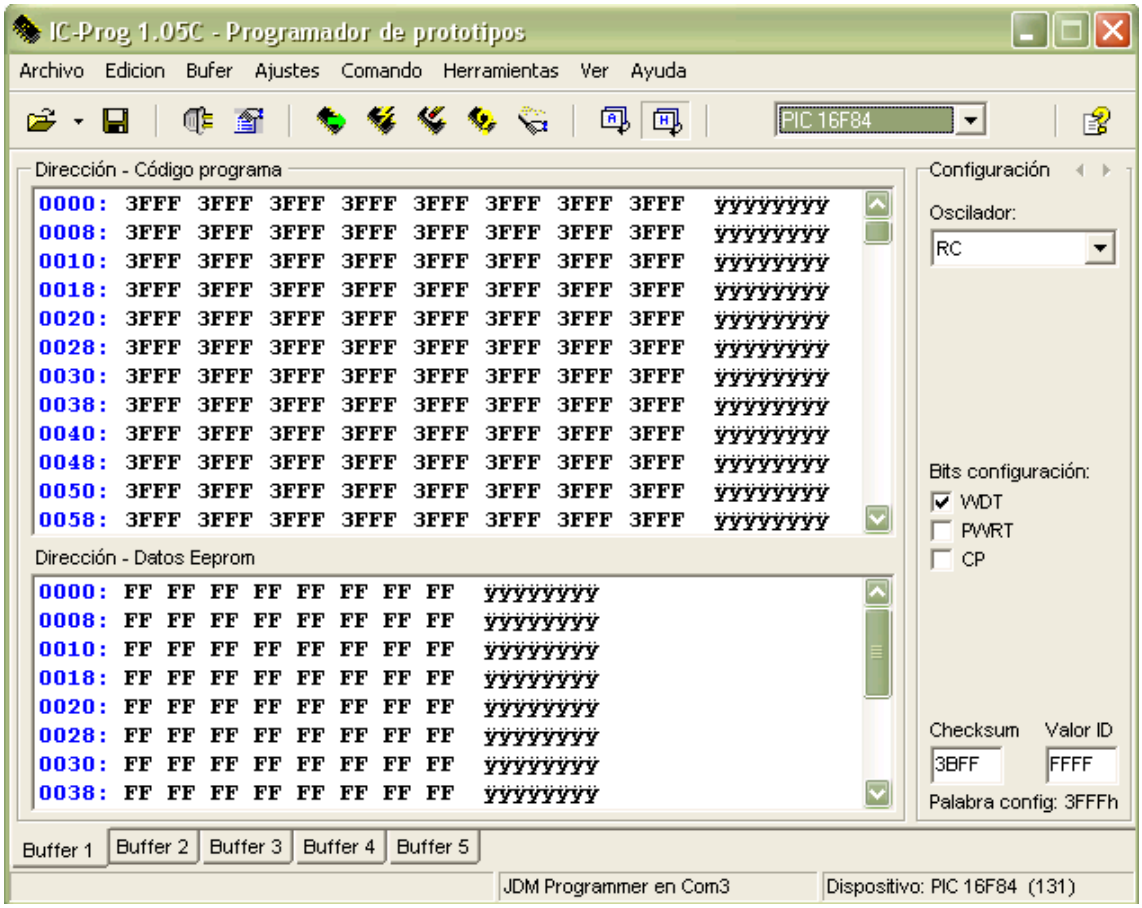


Figura 1.43 Pantalla típica del IC-Prog

2. Para cambiar el idioma de debe seleccionar en el menú *Setting* > *Option* > *Language* y elegir idioma (figura 1.44).

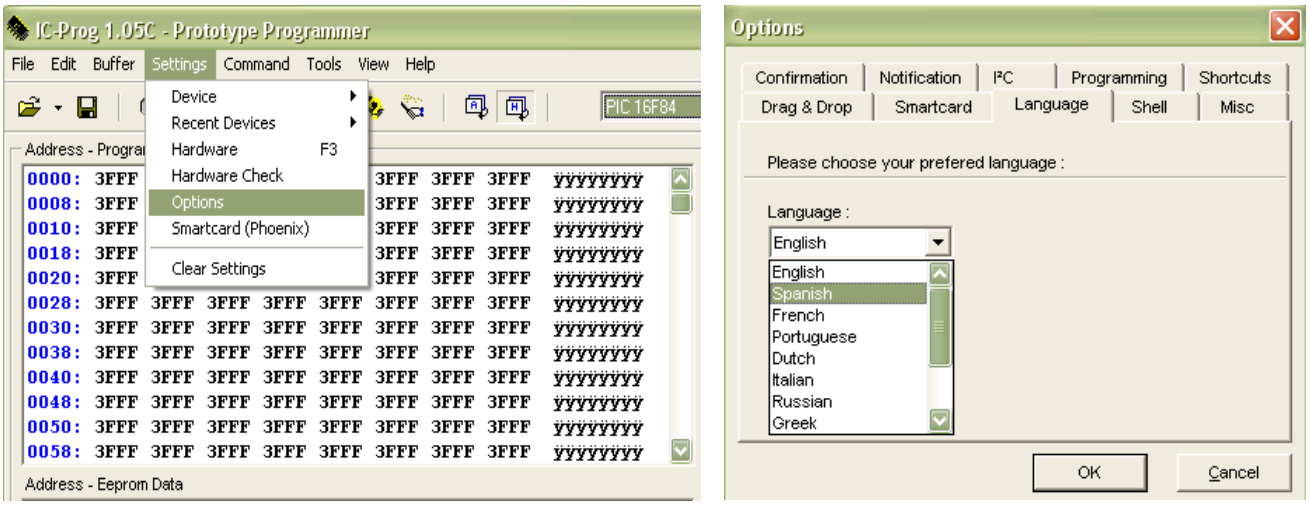


Figura 1.44 Elección del idioma

3. Configurar el hardware necesario para programar los microcontroladores PIC, es decir adaptá el IC-Prog al programador utilizado, en el caso que se ocupe un programador compatible con JDM. Para ello hay que acceder al menú *Ajustes > Tipo hardware,* con lo que aparecerá la pantalla de la figura 1.45, en la que se debe elegir el tipo de programador como JDM y seleccionar el puerto serie adecuado (COM1 o COM2), según lo tenga conectado en el ordenador.

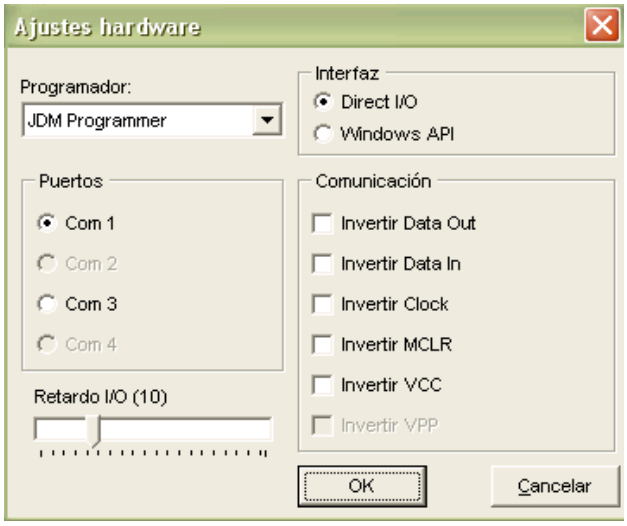


Figura 1.45 Selección del programador

4. A continuación se debe seleccionar el dispositivo a grabar, en este caso el microcontrolador PIC16F84, en el menú *Ajuste* > *Dispositivo* > *Microchip PIC* > *Más* > *PIC16F84,* tal como se describe en la figura 1.46.

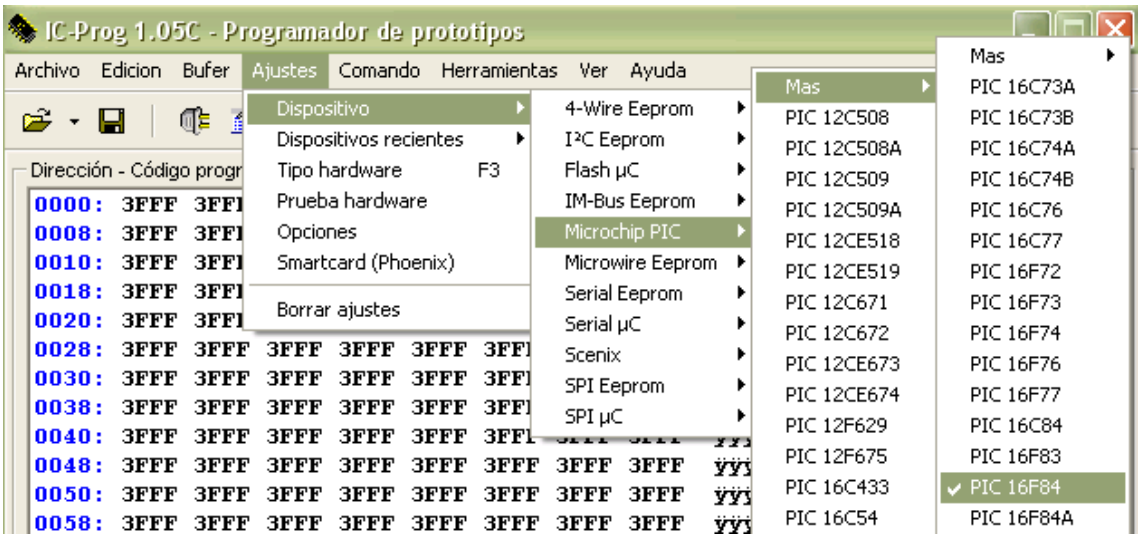


Figura 1.46 Selección del microcontrolador.

El nombre del dispositivo seleccionado aparecerá en una ventana de la barra de herramientas (figura 1.47). Pulsando en la flecha de la ventana se puede elegir cualquiera de los dispositivos soportados por el software *IC-Prog.*

Figura 1.47 El nombre del dispositivo seleccionado aparece en la ventana

5. Elegir el oscilador que va a utilizar el microcontrolador en cuestión (LP, RC, XT, HS). Para ello en la ventana *Oscilador* se elige el tipo XT (oscilador a cristal de cuarzo) figura 1.48.

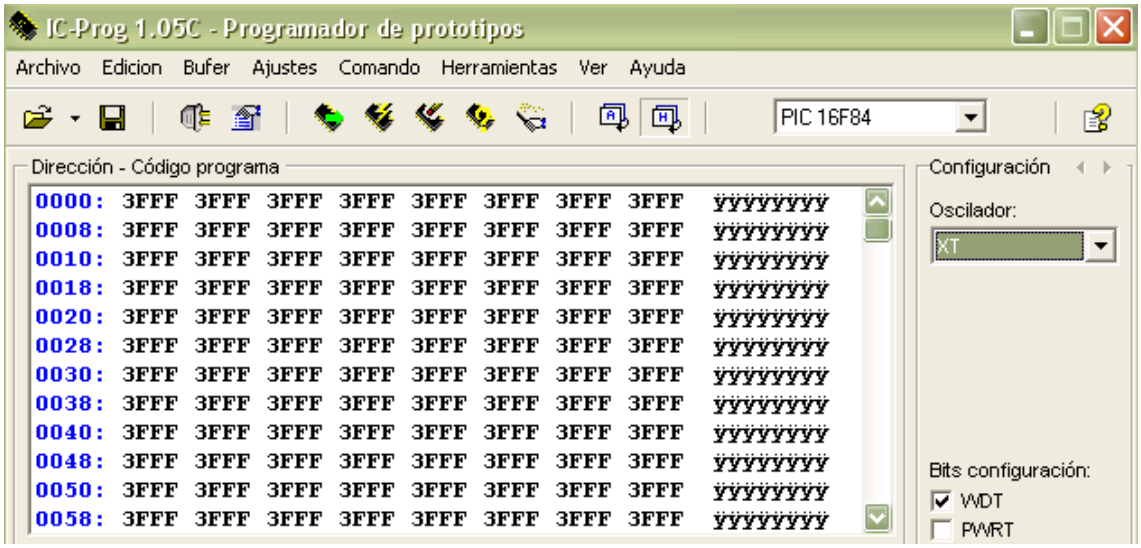


Figura 1.48 Selección del tipo de oscilador y de los bits de configuración.

6. A continuación es necesario activar los **Bits de configuración** que permiten seleccionar varias configuraciones del dispositivo (figura 1.48). En la pantalla de IC-Prog se muestra tres:

- **WDT** (Watchdog Timer).

Habilitación de Watchdog, En aplicación sencilla se deshabilita.

- **PWRT** (Power-up Timer).

Temporizador al encendido. En aplicación sencilla se activa.

- **CP** (Code Protect).

Protección de código de programa. Cuando se programa la protección del código, no es posible leer el contenido de la memoria, de tal manera que el código del programa no se puede copiar, ni alterar, aunque sí se puede volver a borrar completamente todo el microcontrolador. En aplicaciones sencillas se suele deshabilitar.

7. El IC-Prog ya ésta en condiciones de proceder a la grabación de datos en el dispositivo insertado en el programador. Para ello, en la pantalla de edición se escribe los datos del programa de control a

grabar. O se puede cargar estos datos más eficazmente a partir de un archivo, sin necesidad de teclear.

Figura 1.49 Datos a grabar en el microcontrolador

8. Para proceder a la grabación del chip basta con activar el menú *Comand > Programa todo* (figura 1.50) o bien pulsar la tecla de función F5. El chip comenzará a ser programado con los datos cargados en el buffer activo.

Figura 1.50 Comenzar a programar el PIC16F84.

9. El proceso de grabación se irá mostrando, tal como puede apreciarse en la figura 1.51. El tiempo empleado en la grabación del PIC16F84 dependerá de la rapidez del ordenador con el que se esté trabajando.

Figura 1.51 Pantalla que aparecen durante el proceso de programación y verificación

En caso de que la verificación haya sido correcta, se informará de tal hecho mediante una ventana que indicará que la verificación es correcta y el proceso de grabación habrá finalizado.

Si un microcontrolador está protegido contra la lectura de código, es decir tiene habilitada la opción CP (Code Protect) del área de configuración (figura 1.52).

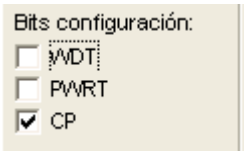


Figura 1.52 La protección del código puede ocasionar una pantalla de aparente error

Los datos grabados no pueden ser leídos en la fase de verificación y, por tanto, ésta no puede realizarse visualizando un error de verificación, sin embargo la grabación puede haber sido realizada correctamente.

10. Una vez grabado el PIC16F84 se debe extraer del programa y comprobar su correcto funcionamiento dentro del circuito.

CAPÍTULO II

DISEÑO DEL MARCADOR DE TIEMPO

2.1 DISEÑO ESTRUCTURAL

A través de un diagrama de bloques se presenta de una manera práctica el diseño del marcador de tiempo.

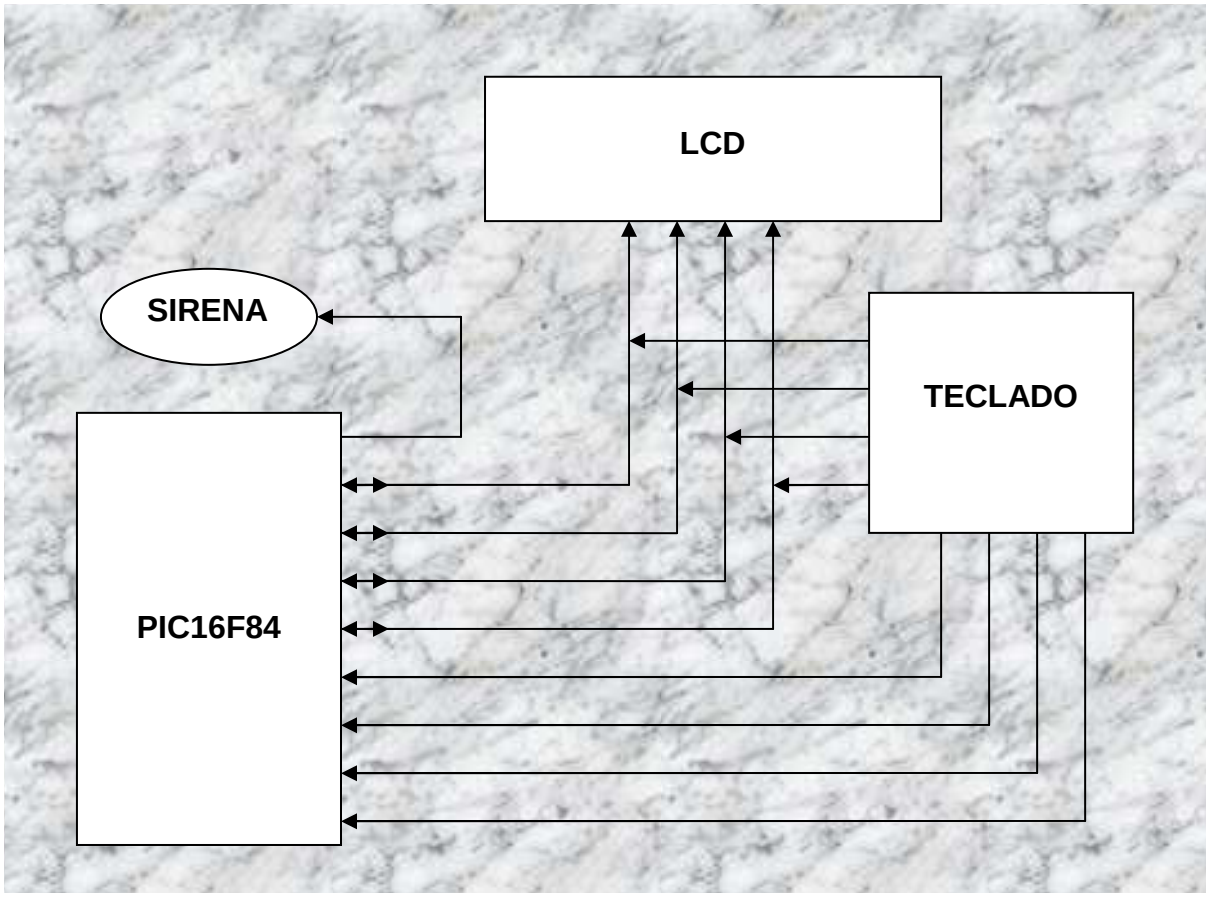


Figura 2.1 diagrama de bloques de un marcador de tiempo programable

2.2 DETALLE DE LAS PARTES QUE SE SOLUCIONARÁ MEDIANTE HARDWARE.

2.2.1 TECLADO MATRICIAL

Los sistemas con microcontroladores tienen como finalidad el proceso de datos esto se lo obtiene a través de una forma manual o automática por medio de sensores que midan parámetros físicos, en este caso va a ser de una forma manual por que los datos van a ser suministrados por el usuario a través de un teclado matricial.

Un teclado matricial está constituido por una matriz de pulsadores dispuestos en filas y columnas cuyo objetivo es reducir el número de líneas necesarias para su conexión

A continuación se presenta la constitución interna de un teclado matricial hexadecimal que se utilizó en el proyecto.

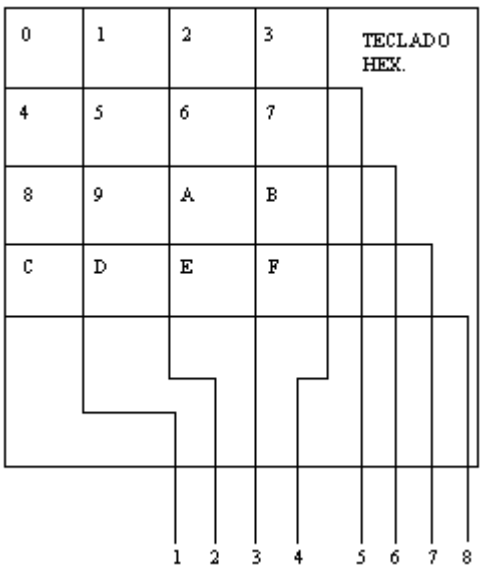


Figura 2.2 Constitución interna de un teclado matricial hexadecimal

2.2.1.1 ALGORITMO PARA LA PROGRAMACIÓN DEL TECLADO

Las técnicas de programación usadas para explorar el teclado está esquematizada en el diagrama de flujo de la subrutina Teclado_LeeOrdenTecla que lee el orden de la tecla pulsada. Se entiende como orden el número que le corresponde a cada tecla leyéndola de izquierda a derecha y de arriba hacia abajo, independientemente de la serigrafía sobre ella.

La exploración del teclado debe deducir el orden de la tecla pulsada sin importar lo serigrafiado sobre ella. Cada tecla tiene asignado el número de orden indicado y que se va contabilizando en la variable Tecl_TeclaOrden. El programa utiliza una tabla de conversión para convertir el orden de la tecla a su valor real para cada tipo de tecla.

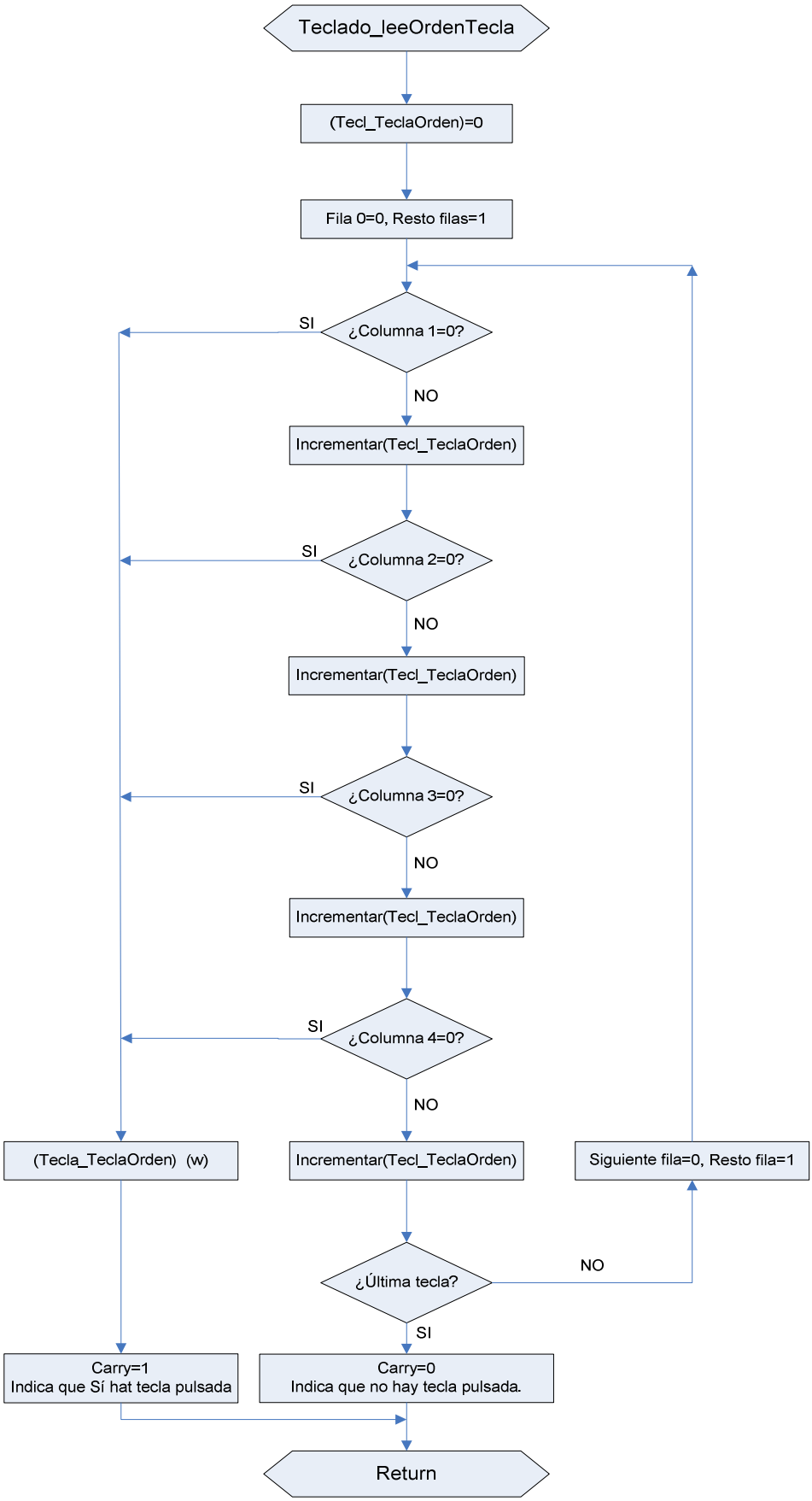


Figura 2.3 Para la exploración de un teclado hexadecimal

2.2.1.2 LIBRERÍA DE SUBRUTINAS

La librería TECLADOS.INC contiene las subrutinas de control del teclado hexadecimal.

Para la confección del programa se utilizarán las siguientes subrutinas:

- “Teclado_Inicializada”

Configura las líneas del puerto B según la conexión del teclado realizado y comprueba que no hay pulsado tecla alguna al principio.

- “Teclado_Espara DejaPulsar”

Permanece en esta subrutina mientras siga pulsando la tecla. Se utiliza para que no repita la misma lectura varias veces.

- “Teclado_LeeOrdenTecla”

Lee el teclado obtenido en el registro W el orden de la tecla pulsada. Además, posiciona el flan Carry para indicar si se ha presionado alguna tecla.

- “Teclado_LeeHex”

Lee un teclado hexadecimal de la tecla pulsada. Además posiciona el flan Carry para indicar si se ha presionado alguna tecla.

```
;************************** Librería "TECLADO.INC"*********************************
;
; Librería de subrutinas para la gestión de un teclado organizado en una matriz
; de 4 x 4 y conectado al Puerto B
;
;              RB4  RB5  RB6  RB7
;               ^    ^    ^    ^
;              |------|------|------|------|
;     RB0 -->|  0  |  1   |  2   |   3  |
;              |------|------|------|------|
;     RB1 -->|  4   |  5   |  6   |   7   |
;              |------|------|------|------|
;     RB2 -->|  8   |   9 |  10 |   11 |
;              |------|------|------|------|
;     RB3 -->| 12  |  13 | 14  |  15  |
;              |------|------|------|------|
;
; Los números que se han dibujado dentro de cada cuadrado son el orden de
; las teclas que no tienen por qué coincidir con lo serigrafiado sobre ellas. El
; paso del número de orden de la tecla al valor que hay serigrafiado sobre la
; misma se hace con una tabla de conversión.
;
; ZONA DE DATOS*****************************************************************
;
        CBLOCK
        Tecl_TeclaOrden             ; Orden de la tecla a chequear.
        ENDC

Tecl_UltimaTecla   EQU  d'15'       ; Valor de orden de la última tecla
utilizada.

;
Subrutina"Teclado_LeeHex"*****************************************************
```

```
;
; Cada tecla tiene asignado un número de orden que es contabilizado en la
; variable
; Tecl_TeclaOrden. Para convertir a su valor según el tipo de teclado en
; concreto se utiliza una tabla de conversión.
; A continuación se expone la relación entre el número de orden de la tecla y
; los valores correspondientes para el teclado hexadecimal más utilizado.
;
;          ORDEN DE TECLA:          TECLADO HEX. UTILIZADO:
;               0   1   2   3              0  1  2  3
;               4   5   6   7              4  5  6  7
;               8   9  10  11              8  9  A  B
;              12  13  14  15               C  D  E  F
;
; Así, en este ejemplo, la tecla "8" ocupa el orden 8, la tecla "F" ocupa el orden
15 y la tecla "9" el orden 9.

; Si cambia el teclado también hay que cambiar de tabla de conversión.
;
; Entrada:    En (W) el orden de la tecla pulsada.
; Salida:     En (W) el valor hexadecimal para este teclado concreto.
;
Teclado_LeeHex
        call    Teclado_LeeOrdenTecla       ; Lee el Orden de la tecla pulsada.
        btfss   STATUS,C                    ; ¿Pulsa alguna tecla?, ¿C=1?
        goto    Tecl_FinLeeHex              ; No, por tanto sale.
        call    Tecl_ConvierteOrdenEnHex    ; Lo convierte en su valor real
.                                               mediante tabla.
        bsf     STATUS,C                    ; Vuelve a posicionar el Carry,
.                                               porque la
Tecl_FinLeeHex                              ; instrucción "addwf PCL,F" lo pone .
a "0".
return
;
```

```
Tecl_ConvierteOrdenEnHex                    ; Según el teclado utilizado resulta:
        addwf PCL,F
        DT     0h,1h,2h,3h                  ; Primera fila del teclado.
        DT     4h,5h,6h,7h          ; Segunda fila del teclado
        DT     8h,9h,0Ah,0Bh                ; Tercera fila del teclado.
        DT     0Ch,0Dh,0Eh,0Fh              ; Cuarta fila del teclado.
Teclado_FinTablaHex
;
; Esta tabla se sitúa al principio de la librería con el propósito de que no supere
; la posición 0FFh de memoria ROM de programa. De todas formas, en caso
; que así fuera  visualizaría el siguiente mensaje de error en el proceso de
; ensamblado:
;
    IF (Teclado_FinTablaHex > 0xFF)
       ERROR     "Atención: La tabla ha superado el tamaño de la página de los"
       MESSG     "primeros 256 bytes de memoria ROM. NO funcionará correc."
    ENDIF

; Subrutina "Teclado_Inicializa" -------------------------------------------------
;
; Esta subrutina configura las líneas del Puerto B según la conexión del teclado
; realizada y comprueba que no hay pulsada tecla alguna al principio.

Teclado_Inicializa
        bsf         STATUS,RP0    ; Configura las líneas del puerto:
        movlw               b'11110000'      ; <RB7:RB4> entradas, <RB3:RB0>
salidas
        movwf               PORTB
        bcf         OPTION_REG,NOT_RBPU ;Habilita R de Pull-Up del Puerto B.
        bcf         STATUS,RP0    ; Acceso al banco 0.
;       call        Teclado_EsperaDejePulsar
;       return
;
; Subrutina "Teclado_EsperaDejePulsar" -------------------------------------------
```

```
;
; Permanece en esta subrutina mientras siga pulsada la tecla.
;
Teclado_Comprobacion   EQU   b'11110000'

Teclado_EsperaDejePulsar:
        movlw         Teclado_Comprobacion   ; Pone a cero las cuatro líneas
.        movwf                PORTB                    ; Puerto B.
Teclado_SigueEsperando
        call     Retardo_20ms                  ; Espera a que se estabilicen los
;                                               ;niveles de tensión.
        movf   PORTB,W                        ; Lee el Puerto B.
        sublw Teclado_Comprobacion            ; Si es lo mismo que escribió es .
;que ya no pulsa
        btfss   STATUS,Z                       ;tecla alguna.
        goto   Teclado_SigueEsperando
        return
;
; Subrutina "Teclado_LeeOrdenTecla" ----------------------------------------------------
;
; Lee el teclado, obteniendo el orden de la tecla pulsada.
;
; Salida:       En (W) el número de orden de la tecla pulsada. Además Carry se
;                pone a "1" si se pulsa una tecla ó a "0" si no se pulsa tecla
;                alguna.
;
Teclado_LeeOrdenTecla:
        clrf     Tecl_TeclaOrden            ; Todavía no ha empezado a chequear
.                                           ; el teclado.
        movlw         b'11111110'                ; Va a chequear primera fila.
Tecl_ChequeaFila                        ; (Ver esquema de conexión).
        movwf         PORTB                     ; Activa la fila correspondiente.
Tecl_Columna1
```

```
        btfss   PORTB,4                 ; Chequea la 1ª columna buscando un  .
; cero.
        goto    Tecl_GuardaValor        ; Sí, es cero y por tanto guarda su valor .
; y sale.
        incf    Tecl_TeclaOrden,F               ; Va a chequear la siguiente tecla.
Tecl_Columna2                           ; Repite proceso para las siguientes
        btfss   PORTB,5                 ; columnas.
        goto    Tecl_GuardaValor
        incf    Tecl_TeclaOrden,F
Tecl_Columna3
        btfss   PORTB,6
        goto    Tecl_GuardaValor
        incf    Tecl_TeclaOrden,F
Tecl_Columna4
        btfss   PORTB,7
        goto    Tecl_GuardaValor
        incf    Tecl_TeclaOrden,F
;
; Comprueba si ha chequeado la última tecla, en cuyo caso sale. Para ello
; testea si el contenido del registro Tecl_TeclaOrden es igual al número de
; teclas del teclado.
;
Tecl_TerminaColumnas
        movlw       Tecl_UltimaTecla
        subwf Tecl_TeclaOrden,W  ; (W) = (Tecl_TeclaOrden)-Tecl_UltimaTecla.
        btfsc   STATUS,C            ; ¿C=0?, ¿(W) negativo? ¿TeclaOrden)<15?
        goto    Tecl_NoPulsada      ; No, se ha llegado al final del chequeo.
        bsf     STATUS,C            ; Sí. Va a chequear la siguiente fila.
        rlf     PORTB,W             ; Apunta a la siguiente fila.
        goto    Tecl_ChequeaFila

Tecl_NoPulsada
        bcf     STATUS,C                ; Posiciona C=0, indicando no ha pulsado
        goto    Tecl_FinTecladoLee      ; tecla alguna y sale.
```

```
Tecl_GuardaValor
        movf   Tecl_TeclaOrden,W     ;El orden de la tecla pulsada en (W) y sale.
        bsf    STATUS,C              ;Como hay tecla tecla pulsada, pone C=1.
Tecl_FinTecladoLee
        return
```

2.2.2 VISUALIZADOR LCD

Las pantallas de cristal líquido o display LCD para mensajes tiene la capacidad de mostrar cualquier carácter alfanumérico, permitiendo representar la información que genera cualquier equipo electrónico de una forma fácil y económica.

La pantalla consta de una matriz de caracteres distribuidos en una, dos, tres, o cuatro líneas de 16 hasta 40 caracteres cada línea.

El proceso de visualización es gobernada por un microcontrolador incorporado a la pantalla.

Su fácil manejo lo hace ideal para dispositivos que necesitan una capacidad de visualización pequeña o media, entre sus principales características tenemos:

- Consumo muy reducido, del orden de 7,5 mW.
- Pantalla de caracteres ASCII, además de las características japoneses Kanji, caracteres griegos y símbolos matemáticos.
- Desplazamiento de los caracteres hacia la izquierda o a la derecha.
- Memoria de 40 caracteres por línea de pantalla, visualizándose 16 caracteres por línea.
- Movimiento del cursor y cambio de su aspecto.
- Permite que el usuario pueda programar ocho caracteres.

Puede ser gobernado de dos formas principales: Conexión con bus de 4 bits, Conexión con bus de 8 bits.

2.2.2.1 PATILLAJE

Un LCD se conecta fácilmente a un microcontrolador como se muestra en la figura 2.4.

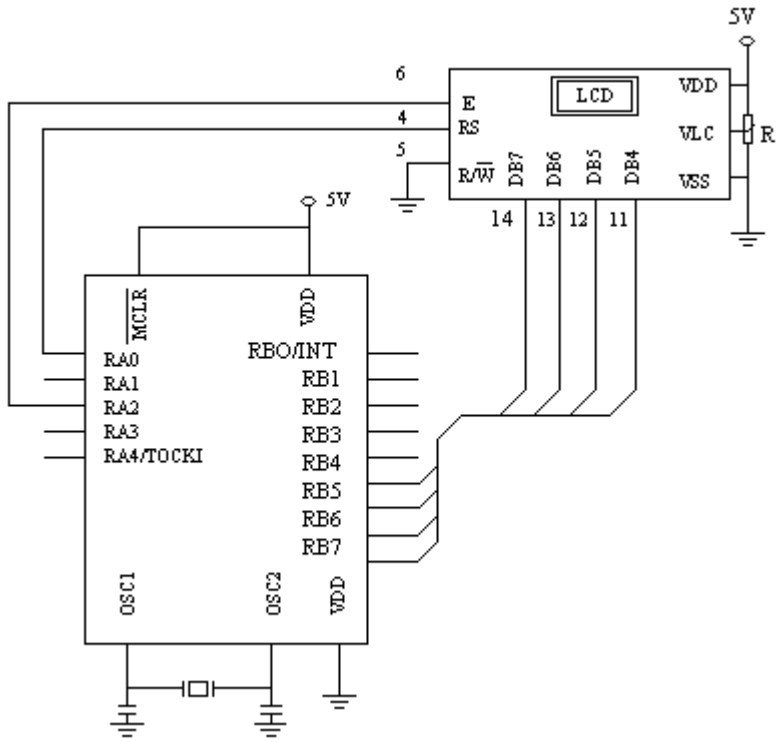


Figura 2.4 Conexión del módulo LCD al PIC16F84A mediante bus de 4 líneas

En la figura 2.4 se ilustra la forma de conectar un LCD al puerto B del microcontrolador mediante 4 líneas y sin lectura del Busy Flag lo que implicaría utilizar el mínimo posible de pines del microcontrolador para el control del display LCD.

Las líneas del bus datos son triestados y pasan a estado de alta impedancia cuando el LCD no está habilitado.

La alimentación es de +5V, la regulación de contraste se realiza mediante el voltaje obtenido al dividir los 5V con una resistencia ajustable de 10K y aplicársela al pin V_{LC} (VO) como se describe en la figura 2.4.

Tabla 2.1Función de los pines en un LM016L

SEÑAL	DEFINICIÓN	PINES	FUNCIÓN
DB0….DB7	*Data Bus*	7….14	Bus de Datos
E	*Enable*	6	E=0, LCD no habilitado E=1, LCD habilitado
R/W	*Read/Write*	5	R/W=0, escribe en LCD R/W=1, lee del LCD
RS	*Register Select*	4	R/S=0, Modo Comando R/S=1, Modo Carácter
V_{LC} (V0)	*Liquid cristal driving Voltaje*	3	Tensión para ajustar el Contraste
V_{DD}	*Power Suplí Voltaje*	2	Tensión de alimentación +5 Voltios
V_{SS}	*Ground*	1	Masa

2.2.2.2 MEMORIA DDRAM

El LM016L posee una zona de memoria RAM llamada **DDRAM** donde se almacenan los caracteres que se pueden representar.

Tiene una capacidad de 80 bytes, 40 por cada línea, de los cuales sólo 32 se pueden visualizar a la vez 16 bytes por línea como se observa en la figura 2.5.

En pantalla se visualizan 32 caracteres: 16 de cada fila															
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Figura 2.5 DDRAM

Las dos direcciones más importantes de la DDRAM son:

- Dirección 00h, que es el comienzo de la primera línea.
- Dirección 40h, que el comienzo de la segunda línea.

Cada vez que se escribe un dato en la DDRAM automáticamente se apunta a la siguiente posición, donde se realizará la escritura del próximo carácter.

2.2.2.3 CARACTERES DEFINIDOS EN LA CGROM

El módulo LM016L posee una zona de memoria interna no volátil llamada CGROM donde se almacena una tabla con los 192 caracteres que pueden ser visualizados.

Para visualizar un carácter debe recibir el código correspondiente, si desea visualizar el carácter “A” el LCD debe recibir por el bus de datos el código b”01000001” donde cada carácter tiene su representación de 8 bits.

2.2.2.4 MODOS DE FUNCIONAMIENTO

Los modos de funcionamiento principales del LM016L son:

2.2.2.4.1 Modo comando

Cuando por el bus de datos el LCD recibe instrucciones como:

“Borrar Display”
“Mover Display”
“Desplazar a izquierda”, etc.

Para trabajar en modo comando, el pin RS debe estar a “0” , el pin R/W también debe ser “0” para indicar que está realizando una operación de escritura, una operación en este modo tarda un máximo de 1.64 ms.

2.2.2.4.2 Modo carácter o dato

Cuando por el bus de datos el visualizador LCD recibe un carácter de escritura en la DDRAM. Es decir, cuando se envía al LCD el carácter ASCII a visualizar.

Para trabajar en este modo, el pin RS debe estar a “1” , el pin R/W debe ser “0” para indicar que esta haciendo una operación de escritura, una operación en este modo tarda un máximo de 40us.

2.2.2.4.3 Modo lectura del “busy flag” o lcd ocupado

En el bit 7 del bus de datos el LCD informa al microcontrolador de que está ocupado.

Para ello se lee el bus de datos con RS=0 y R/W=1, si el bit 7 del bus de datos es “1” indica que la pantalla LCD está ocupada realizando operaciones internas y no puede aceptar nuevas instrucciones ni datos. Hay que esperar a que el Busy Flag valga “0” para enviarle la siguiente instrucción o carácter.

Este modo es muy importante ya que evita posibles problemas de tiempo, de manera que no se realiza ninguna operación con el LCD hasta comprobar que no esta ocupado.

El pin R/W permite leer el registro de estado en el modo Busy Flag que sólo sirve para comprobar si el controlador ha terminado de realizar la instrucción que se ha enviado y así poder enviar más.

Para realizar un control sencillo se puede hacer pausas después de cada instrucción o envío de datos para no tener que leer el registro de estado, con ello se evita el modo de lectura de Busy Flag la principal ventaja que se

encontraría con esto se ahorrara un pin del microcontrolador porque la línea de R/W no es necesario y se puede conectar directamente a masa.

El retardo sustituye al Busy Flag que se lo debe utilizar antes de realizar cualquier nueva operación con el LCD, este retardo debe ser mayor de 1,64 ms si trabaja en modo comando y mayor de 40 us si trabaja en modo dato.

2.2.2.5 COMANDOS DE CONTROL

Los comandos que admite el módulo LM016L se puede visualizar en la siguiente tabla 2.2.

Tabla 2.2 Comando del visualizador LCD LM016L

COMANDO	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Clear Display	0	0	0	0	0	0	0	0	0	1
Return Home	0	0	0	0	0	0	0	0	1	*
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S
Display Control	0	0	0	0	0	0	1	D	C	B
Cursor and display shift	0	0	0	0	0	1	S/C	R/L	*	*
Function Set	0	0	0	0	1	DL	N	F	*	*
Set CGRAM Address	0	0	0	1	CCRAM Address					
Set DDRAM Address	0	0	1	DDRAM Addres						
Read Busy Flag	0	1	BF	DDRAM Addres						
Write RAM	1	0	Write Data							
Read RAM	1	1	Read Data							

Los comandos se envían a través del bus de datos, para que el LCD lo reconozca hay que poner la señal RS a nivel bajo.

Para mayor comprensión se a detallar los comandos y símbolos de esta tabla:

2.2.2.5.1 Clear display (00000001)

Borra pantalla y devuelve el cursor a la posición inicial (dirección 0 de la DDRAM.

2.2.2.5.2 Return home (0000001x)

Devuelve el cursor la posición original de la DDRAM (dirección 00h) quedando intacto su contenido.

2.2.2.5.3 Entry mode set (000001 i/d s)

Modo entrada, establece las características de escritura de los datos *Shift e Increment/Decrement:*

S=O: La información visualizada en pantalla no se desplaza al escribir un caracter nuevo.

S=1: La información visualizada se desplaza al escribir un nuevo caracter. La pantalla se desplaza en el sentido indicado por el bit I/D cuando el cursor llega al filo de la pantalla.

I/D=0: Decremento de la posición del cursor. Se decrementa el puntero de la DDRAM.

2.2.2.5.4 Display control (00001dcb).

Control de la pantalla.

B=0: Blink OOF, no hay efecto de parpadeo del cursor.

B=1: Blink ON, efecto de parpadeo del cursor.

C=0: Cursor OFF, el cursor no se visualiza.

C=1: Cursor ON, el cursor es visualizado.

D=0: Display OFF, el display se apaga.

D=1: Display ON, el display se enciende.

2.2.2.5.5 Cursor and display shift(0001s/c r/l xx)

Control de los desplazamientos del cursor y de la pantalla:

R/L=0 Left. A la izquierda.

R/L=1 Right. A la derecha.

S/C=0 El efecto de desplazamiento se aplica sólo sobre el cursor sin alterar el contenido de la DDRAM.

S/C=1 El efecto de desplazamiento se aplica sobre todo el display .

2.2.2.5.6 Function set(001 dl n f xx)

Características de control hardware:

F=0 Font. Caracteres de 5 x 7 puntos.

F=1 Font. Caracteres de 5 x 10 puntos.

N=0 Number Line. Pantalla de una 1 línea.

N=1 Number Line. Pantalla de 2 líneas.

DL=0 Data Length. Comunicación con 4 bits.

Indica al display LCD que solamente se va a utilizar las líneas DB7, DB6, DB5, y DB4 para enviarle los datos y que se hará enviando primero el nibble alto, y a continuación el nibble bajo del dato.

DL=1 Data Length. Comunicación con 8 bits.

2.2.2.5.7 Set cgram address

Se va a escribir sobre la dirección CGRAM señalada.

2.2.2.5.8 Set ddram address (1ddddddd)

Esta instrucción se utiliza para modificar el puntero a la DDRAM

2.2.2.5.9 Read busy flag

Lee el BF indicando si hay una operación interna en curso y lee, además, el contenido de la dirección DDRAM apuntada.

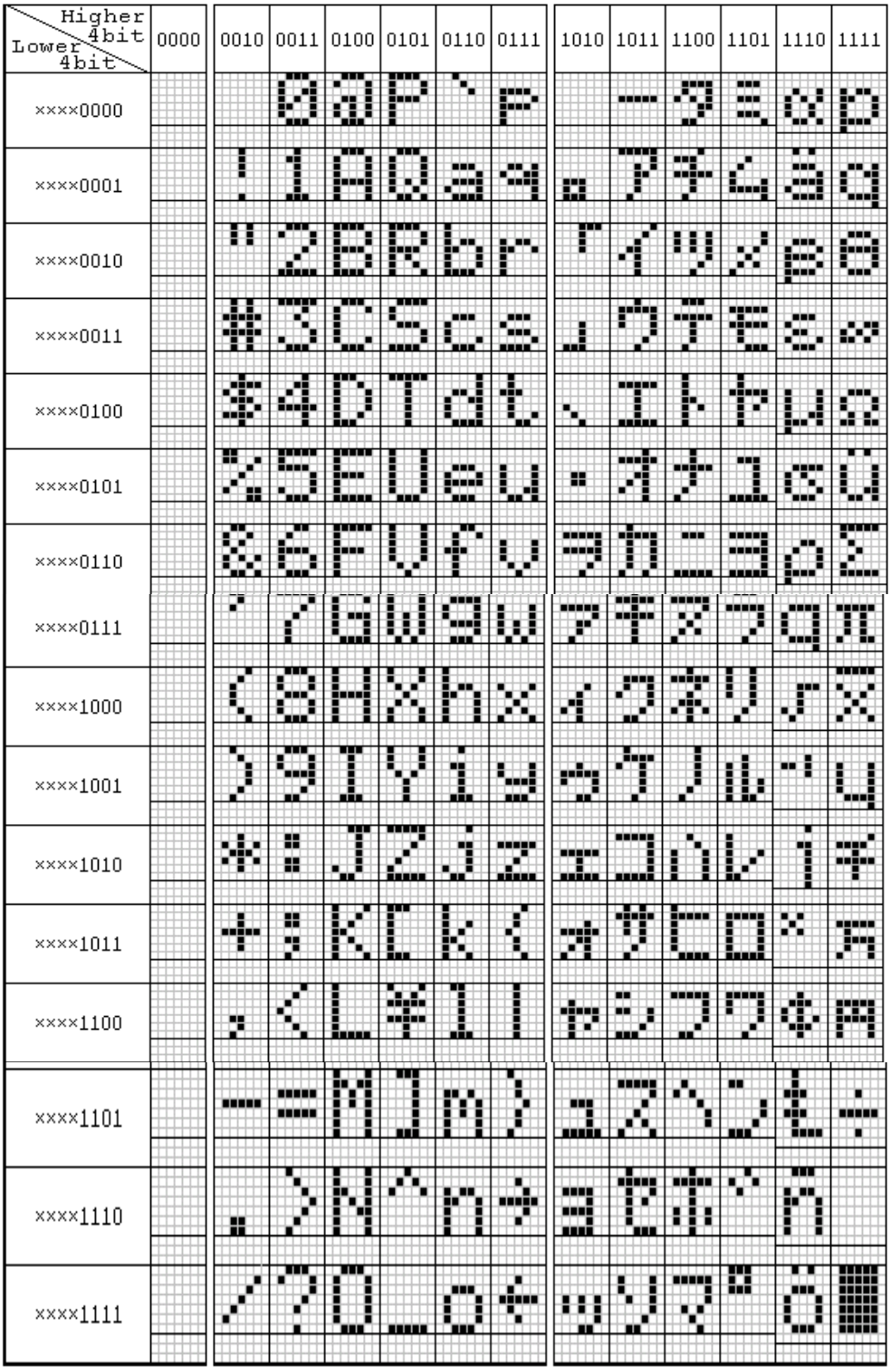


Figura 2.6 Caracteres definidos dentro de la tabla CGROM

3.1 SOFTWARE QUE HACE FUNCIONAR AL SISTEMA

A continuación se indica todos los pasos que se siguió para la elaboración del software el mismo que hace funcionar de una manera precisa y esperada de acuerdo a las expectativas.

```
;*****************************************************************
; Programa para un marcador de tiempo programable en tiempo real con
; puesta en hora.
; Visualiza los datos a través de un LCD.
; La actualización del reloj se lo consigue mediante un teclado.
; El marcador de tiempo programable contiene diez tiempos libres en el cual el
; usuario puede hacer usos de el de acuerdo a su necesidad.

;*****************************************************************


        list      p=16F84A          ; list directive to define processor
        #include <p16F84A.inc>       ; processor specific variable definitions

        __CONFIG   _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

; '__CONFIG' directive is used to embed configuration data within .asm file.
; The lables following the directive are located in the respective .inc file.
; See respective data sheet for additional information on configuration word.




; VARIABLE DEFINITIONS*****************************************
w_temp       EQU     0x0C       ; variable used for context saving
status_temp   EQU     0x0D       ; variable used for context saving

CBLOCK     0x0E
        Horas, Minutos, Segundos, Aux_Tiempo
        Cont, Cont_OnOff, Reg_dly1, Reg_dly2, Reg_dly3
        LCD_AUX, LCD_ADDR, BANDERAS, MODO, MODO_AUX
        Cont_char, Aux_Macro, TECLA, TECLA_ANT, Dato_AuxH, Dato_AuxL
        Num_Alarma, Num_Alarma_Aux
        Horas_Al, Minutos_Al, Cont_Alarma
ENDC


#define LCD_RS          PORTA, 0
#define LCD_E           PORTA, 1
#define LCD_PORT        PORTB

#define        _Txt          0
#define        _Txt2         1
```

```
#define       _TxtOnOff   2

LCD_TXT   MACRO      Direccion, Texto
      LOCAL       Txt_Macro, lazo_lcd_txt, fin_lcd_txt, Txt

      ;bcf    INTCON, GIE
      movlw       Direccion
      call    LCD_REG
      clrf    Cont_char

lazo_lcd_txt
      movlw       HIGH Txt
      movwf       PCLATH

      movf  Cont_char, W
      call    Txt_Macro
      movwf       Aux_Macro
      xorlw  '\n'

      btfsc  STATUS, Z
      goto   fin_lcd_txt

      movf  Aux_Macro, W
      call    LCD_DATO
      incf    Cont_char, f
      goto   lazo_lcd_txt

Txt_Macro
      addlw LOW Txt
      btfsc  STATUS, C
      incf    PCLATH
      movwf       PCL
Txt
      DT     Texto

fin_lcd_txt
;     bsf     INTCON, GIE
      ENDM


;*****************************************************************
              ORG     0x000           ; processor reset vector
              goto    main            ; go to beginning of program

              ORG     0x004           ; interrupt vector location
              movwf   w_temp          ; save off current W register contents
              movf  STATUS,w          ; move status register into W register
              movwf       status_temp           ; save off contents of STATUS
register
```

```
                btfss   INTCON, T0IE
                goto    $+3
                btfsc   INTCON, T0IF
                goto    interrup_t0

                btfss   INTCON, RBIE
                goto    $+3
                btfsc   INTCON, RBIF
                goto    interrup_rb

; isr code can go here or be located as a call subroutine elsewhere
fin_interrup

                movf    status_temp,w     ; retrieve copy of STATUS register
                movwf         STATUS              ; restore pre-isr STATUS register
contents
                swapf   w_temp,f
                swapf   w_temp,w        ; restore pre-isr W register contents
                retfie                  ; return from interrupt



main
        BANKSEL   TRISA
        clrf    TRISA
        movlw         b'11110000'
        movwf         TRISB
        movlw         b'00000110'
        movwf         OPTION_REG
        BANKSEL   INTCON

        clrf    PORTA

        bsf     INTCON, T0IE
        bsf     INTCON, GIE

        movlw         -.78
        movwf         TMR0
        movlw         .100
        movwf         Cont

        clrf    Horas
        clrf    Minutos
        clrf    Segundos

        clrf    MODO

        bsf     BANDERAS, _Txt
        bsf     BANDERAS, _Txt2
        call    LCD_INI
```

```
        clrf    TECLA
        clrf    TECLA_ANT

Loop
        clrf    PORTB
        bsf     INTCON, RBIE

Loop2
        btfss   BANDERAS, _Txt
        goto    Loop2
        bcf     INTCON, RBIE
        bcf     BANDERAS, _Txt
        clrf    PCLATH
        movf    MODO, W
        addwf   PCL, f
        goto    MODO0
        goto    MODO1
        goto    MODO2
        goto    MODO3
        goto    MODO4
        goto    MODO5
        goto    MODO6
        goto    MODO7
        goto    MODO8
        goto    MODO9
        goto    MODO10
        goto    MODO11

;-----------------------------------------------------
MODO0
        btfss   BANDERAS, _Txt2
        goto    L1_MODO0
        bcf     BANDERAS, _Txt2
        movlw           0x01
        call    LCD_REG
        LCD_TXT     0x86,"HORA\n"
L1_MODO0
        call    Mostrar_Hora

        movf    Segundos, f
        btfss   STATUS, Z
        goto    Loop

        clrf    Num_Alarma_Aux

Loop_Verif_Alarma
        movf    Num_Alarma_Aux, W
        incf    Num_Alarma_Aux, f
        call    EEPROM_READ
        xorwf   Horas, W
```

```
        btfss   STATUS, Z
        goto    L1_Verif_Alarma1

        movf    Num_Alarma_Aux, W
        incf    Num_Alarma_Aux, f
        call    EEPROM_READ
        xorwf   Minutos, W
        btfss   STATUS, Z
        goto    L1_Verif_Alarma2
        bsf     PORTA, 2
        movlw   .10                     ;.05 cambiar para 5 segundos
        movwf   Cont_Alarma
        goto    Loop

L1_Verif_Alarma1
        incf    Num_Alarma_Aux, f
L1_Verif_Alarma2
        movlw   .10                     ;.05 cambiar para 5 segundos
        xorwf   Num_Alarma_Aux, W
        btfss   STATUS, Z
        goto    Loop_Verif_Alarma
        goto    Loop

;-----------------------------------------------------
MODO1
MODO2
MODO3
        btfss   BANDERAS, _Txt2
        goto    L1_MODO1
        bcf     BANDERAS, _Txt2
        call    TXT_IGUALAR
        movlw   ' '
        movwf   Dato_AuxH
        movwf   Dato_AuxL
        movlw   .30
        movwf   Cont_OnOff

L1_MODO1
        call    Mostrar_Hora
        goto    Loop

;-----------------------------------------------------
MODO4   movlw   0xC4
        goto    MODO6+2

MODO5   movlw   0xC7
        goto    MODO6+2

MODO6   movlw   0xCA
        goto    MODO6+2
```

```
        call    LCD_REG
        btfsc   BANDERAS, _TxtOnOff
        goto    $+6
        movf    Dato_AuxH, W
        call    LCD_DATO
        movf    Dato_AuxL, W
        call    LCD_DATO
        goto    Loop
        movlw       ' '
        call    LCD_DATO
        movlw       ' '
        call    LCD_DATO
        goto    Loop

MODO7
        movlw       0x01
        call    LCD_REG
        call    TXT_ALARMA

        decf    Num_Alarma, W
        addwf Num_Alarma, W
        addlw -1
        call    EEPROM_READ
        movwf       Horas_Al
        decf    Num_Alarma, W
        addwf Num_Alarma, W
        call    EEPROM_READ
        movwf       Minutos_Al

        call    Mostrar_Alarma
        goto    Loop

MODO8
MODO9
        btfss   BANDERAS, _Txt2
        goto    L1_MODO8
        bcf     BANDERAS, _Txt2
        movlw       0x01
        call    LCD_REG
        call    TXT_IGUALAR_A

        movlw       ' '
        movwf       Dato_AuxH
        movwf       Dato_AuxL

        movlw       .30
        movwf       Cont_OnOff

L1_MODO8
```

```
        call    Mostrar_Alarma
        goto    Loop

MODO10
        movlw       0xC6
        goto    MODO6+2

MODO11
        movlw       0xC9
        goto    MODO6+2

; remaining code goes here


TXT_IGUALAR
        LCD_TXT    0x82,"AJUSTAR HORA\n"
        return

TXT_IGUALAR_A
        LCD_TXT    0x80,"AJUST. ALARMA \n"
        goto    TXT2_ALARMA
        return

TXT_ALARMA
        LCD_TXT    0x84,"ALARMA \n"
TXT2_ALARMA

        movf   Num_Alarma, W
        sublw 0x0A
        btfss   STATUS, Z
        goto    $+6
        movlw       '1'
        call    LCD_DATO
        movlw       '0'
        call    LCD_DATO
        goto    $+6
        movlw       '0'
        call    LCD_DATO
        movf   Num_Alarma, W
        addlw 0x30
        call    LCD_DATO

        return


Mostrar_Alarma
        movlw       0xC6
        call    LCD_REG

        movf   MODO, W
```

```
        xorlw  8
        btfss  STATUS, Z
        goto   L1_Mostrar_HorasA

        btfss  BANDERAS, _TxtOnOff
        goto   L1_Mostrar_HorasA
        movlw        ' '
        call   LCD_DATO
        movlw        ' '
        call   LCD_DATO
        goto   L2_Mostrar_HorasA
L1_Mostrar_HorasA
        swapf Horas_Al, W
        andlw 0x0F
        addlw 0x30
        call   LCD_DATO

        movf   Horas_Al, W
        andlw 0x0F
        addlw 0x30
        call   LCD_DATO
L2_Mostrar_HorasA

        movlw        ':'
        call   LCD_DATO

        movf   MODO, W
        xorlw  9
        btfss  STATUS, Z
        goto   L1_Mostrar_MinutosA

        btfss  BANDERAS, _TxtOnOff
        goto   L1_Mostrar_MinutosA
        movlw        ' '
        call   LCD_DATO
        movlw        ' '
        call   LCD_DATO
        goto   L2_Mostrar_MinutosA
L1_Mostrar_MinutosA
        swapf Minutos_Al, W
        andlw 0x0F
        addlw 0x30
        call   LCD_DATO

        movf   Minutos_Al, W
        andlw 0x0F
        addlw 0x30
        call   LCD_DATO
L2_Mostrar_MinutosA
        return
```

```
Mostrar_Hora
        movlw       0xC4
        call    LCD_REG

        movf    MODO, W
        xorlw   1
        btfss   STATUS, Z
        goto    L1_Mostrar_Horas

        btfss   BANDERAS, _TxtOnOff
        goto    L1_Mostrar_Horas
        movlw       ' '
        call    LCD_DATO
        movlw       ' '
        call    LCD_DATO
        goto    L2_Mostrar_Horas
L1_Mostrar_Horas

        swapf Horas, W
        andlw 0x0F
        addlw 0x30
        call    LCD_DATO

        movf    Horas, W
        andlw 0x0F
        addlw 0x30
        call    LCD_DATO
L2_Mostrar_Horas

        movlw       ':'
        call    LCD_DATO

        movf    MODO, W
        xorlw   2
        btfss   STATUS, Z
        goto    L1_Mostrar_Minutos

        btfss   BANDERAS, _TxtOnOff
        goto    L1_Mostrar_Minutos
        movlw       ' '
        call    LCD_DATO
        movlw       ' '
        call    LCD_DATO
        goto    L2_Mostrar_Minutos
L1_Mostrar_Minutos

        swapf Minutos, W
```

```
        andlw 0x0F
        addlw 0x30
        call    LCD_DATO

        movf  Minutos, W
        andlw 0x0F
        addlw 0x30
        call    LCD_DATO
L2_Mostrar_Minutos

        movlw       ':'
        call    LCD_DATO

        movf  MODO, W
        xorlw  3
        btfss  STATUS, Z
        goto   L1_Mostrar_Segundos

        btfss  BANDERAS, _TxtOnOff
        goto   L1_Mostrar_Segundos
        movlw       ' '
        call    LCD_DATO
        movlw       ' '
        call    LCD_DATO
        return
L1_Mostrar_Segundos

        swapf Segundos, W
        andlw 0x0F
        addlw 0x30
        call    LCD_DATO

        movf  Segundos, W
        andlw 0x0F
        addlw 0x30
        call    LCD_DATO
        return


interrup_t0
        movlw       -.78
        movwf       TMR0

        decfsz      Cont, f
        goto   fin_Cont
        movlw       .100
        movwf       Cont
        call    Inc_tiempo

        movf  MODO, f
```

```
        btfsc   STATUS, Z
        bsf     BANDERAS, _Txt


        decfsz          Cont_Alarma, f
        goto    fin_Cont
        bcf     PORTA, 2


fin_Cont
        movf    MODO, f
        btfsc   STATUS, Z
        goto    fin_t0

        movlw           7
        xorwf   MODO, W
        btfsc   STATUS, Z
        goto    fin_t0

        decfsz          Cont_OnOff, f
        goto    fin_t0
        movlw           .30
        movwf           Cont_OnOff
        bsf     BANDERAS, _Txt
        movlw            1<<_TxtOnOff
        xorwf   BANDERAS, f

fin_t0
        bcf     INTCON, T0IF
        goto    fin_interrup


interrup_rb
        movf    TECLA, W
        movwf           TECLA_ANT
        call    decod_tecla
        movwf           TECLA

        movf    TECLA, W
        xorwf   TECLA_ANT, W
        btfsc   STATUS, Z
        goto    fin_rb

        bsf     BANDERAS, _Txt
        bsf     BANDERAS, _Txt2
        bsf     BANDERAS, _TxtOnOff

        movf    TECLA, W
        xorlw   'A'
        btfsc   STATUS, Z
```

```
        goto   TeclaA

        movf   TECLA, W
        xorlw  'B'
        btfsc  STATUS, Z
        goto   TeclaB

        movf   TECLA, W
        xorlw  'F'
        btfsc  STATUS, Z
        goto   TeclaF

        movf   TECLA, W
        xorlw  'C'
        btfsc  STATUS, Z
        goto   TeclaC

        movf   TECLA, W
        xorlw  'D'
        btfsc  STATUS, Z
        goto   TeclaD

        movf   TECLA, W
        xorlw  'E'
        btfsc  STATUS, Z
        goto   TeclaE


        movlw          '0'
        subwf TECLA, W
        btfss  STATUS, C
        goto   $+5
        movf   TECLA, W
        sublw '9'
        btfsc  STATUS, C
        goto   TeclaNum


fin_rb
        bcf    INTCON, RBIF
        goto   fin_interrup


TeclaA
;       movlw          ' '
;       movwf          Dato_AuxH
;       movwf          Dato_AuxL

        movf   MODO, f              ; MODO=0?
        btfss  STATUS, Z
```

```
goto   $+4
movlw       3
movwf       MODO
goto   fin_rb

movf   MODO, W          ; MODO=1?
xorlw  1
btfss  STATUS, Z
goto   $+4
movlw       0
movwf       MODO
goto   fin_rb

movf   MODO, W          ; MODO=2?
xorlw  2
btfss  STATUS, Z
goto   $+4
movlw       1
movwf       MODO
goto   fin_rb

movf   MODO, W          ; MODO=3?
xorlw  3
btfss  STATUS, Z
goto   $+4
movlw       2
movwf       MODO
goto   fin_rb

movf   MODO, W          ; MODO=4?
xorlw  4
btfss  STATUS, Z
goto   $+4
movlw       0
movwf       MODO
goto   fin_rb

movf   MODO, W          ; MODO=5?
xorlw  5
btfss  STATUS, Z
goto   $+4
movlw       1
movwf       MODO
goto   fin_rb

movf   MODO, W          ; MODO=6?
xorlw  6
btfss  STATUS, Z
goto   fin_rb
movlw       2
```

```
        movwf   MODO
        goto    fin_rb


TeclaB
;       movlw   ' '
;       movwf   Dato_AuxH
;       movwf   Dato_AuxL

        movf    MODO, f         ; MODO=0?
        btfss   STATUS, Z
        goto    $+4
        movlw   1
        movwf   MODO
        goto    fin_rb

        movf    MODO, W         ; MODO=1?
        xorlw   1
        btfss   STATUS, Z
        goto    $+4
        movlw   2
        movwf   MODO
        goto    fin_rb

        movf    MODO, W         ; MODO=2?
        xorlw   2
        btfss   STATUS, Z
        goto    $+4
        movlw   3
        movwf   MODO
        goto    fin_rb

        movf    MODO, W         ; MODO=3?
        xorlw   3
        btfss   STATUS, Z
        goto    $+4
        movlw   0
        movwf   MODO
        goto    fin_rb

        movf    MODO, W         ; MODO=4?
        xorlw   4
        btfss   STATUS, Z
        goto    $+4
        movlw   2
        movwf   MODO
        goto    fin_rb

        movf    MODO, W         ; MODO=5?
        xorlw   5
```

```
btfss  STATUS, Z
goto   $+4
movlw       3
movwf       MODO
goto   fin_rb

movf   MODO, W          ; MODO=6?
xorlw  6
btfss  STATUS, Z
goto   $+4
movlw       4
movwf       MODO
goto   fin_rb

movf   MODO, W          ; MODO=7?
xorlw  7
btfss  STATUS, Z
goto   $+4
movlw       8
movwf       MODO
goto   fin_rb

movf   MODO, W          ; MODO=8?
xorlw  8
btfss  STATUS, Z
goto   $+4
movlw       9
movwf       MODO
goto   fin_rb

movf   MODO, W          ; MODO=8?
xorlw  9
btfss  STATUS, Z
goto   $+4
movlw       7
movwf       MODO
goto   fin_rb

movf   MODO, W          ; MODO=10?
xorlw  .10
btfss  STATUS, Z
goto   $+4
movlw       9
movwf       MODO
goto   fin_rb

movf   MODO, W          ; MODO=11?
xorlw  .11
btfss  STATUS, Z
goto   $+4
```

```
        movlw   7
        movwf   MODO
        goto    fin_rb

TeclaC
        movf    MODO, W         ; MODO=7?
        xorlw   7
        btfss   STATUS, Z
        goto    fin_rb


        decf    Num_Alarma, f

        movf    Num_Alarma, f
        btfss   STATUS, Z
        goto    fin_rb
        movlw   0x0A
        movwf   Num_Alarma

        goto    fin_rb

TeclaD
        movf    MODO, W         ; MODO=7?
        xorlw   7
        btfss   STATUS, Z
        goto    fin_rb

        incf    Num_Alarma, f
        movlw   0x0B
        xorwf   Num_Alarma, W
        btfss   STATUS, Z
        goto    fin_rb
        movlw   0x01
        movwf   Num_Alarma
        goto    fin_rb

TeclaE
        movf    MODO, W         ; MODO=0?
        btfss   STATUS, Z
        goto    $+6
        movlw   7
        movwf   MODO
        movlw   0x01
        movwf   Num_Alarma
        goto    fin_rb
        clrf    MODO
        goto    fin_rb

TeclaF
        bsf     BANDERAS, _Txt
```

```
        bsf     BANDERAS, _Txt2
        bsf     BANDERAS, _TxtOnOff
        movf    MODO, W
        xorlw   1
        btfss   STATUS, Z
        goto    $+3
        movlw           2
        movwf           MODO

        movf    MODO, W
        xorlw   2
        btfss   STATUS, Z
        goto    $+3
        movlw           3
        movwf           MODO

        movf    MODO, W
        xorlw   3
        btfss   STATUS, Z
        goto    $+3
        movlw           0
        movwf           MODO

;-------------------------------------------------------
        movf    MODO, W
        xorlw   4
        btfss   STATUS, Z
        goto    L4_TeclaF
        movf    Dato_AuxH, W
        addlw -0x30
        movwf           Aux_Tiempo

        movlw           ' '
        xorwf   Dato_AuxH, W
        btfsc   STATUS, Z
        clrf    Aux_Tiempo
        swapf Aux_Tiempo, f

        movf    Dato_AuxL, W
        addlw -0x30
        addwf Aux_Tiempo, f

        movf    Aux_Tiempo, W
        sublw 0x23
        btfss   STATUS, C
        goto    $+3
        movf    Aux_Tiempo, W
        movwf           Horas
        movlw           2
        movwf           MODO
```

```
L4_TeclaF
;-----------------------------------------------------
        movf   MODO, W
        xorlw  5
        btfss  STATUS, Z
        goto   L5_TeclaF
        movf   Dato_AuxH, W
        addlw -0x30
        movwf          Aux_Tiempo

        movlw          ' '
        xorwf  Dato_AuxH, W
        btfsc  STATUS, Z
        clrf      Aux_Tiempo
        swapf Aux_Tiempo, f

        movf   Dato_AuxL, W
        addlw -0x30
        addwf Aux_Tiempo, f

        movf   Aux_Tiempo, W
        sublw 0x59
        btfss  STATUS, C
        goto   $+3
        movf   Aux_Tiempo, W
        movwf          Minutos
        movlw          3
        movwf          MODO
L5_TeclaF
;-----------------------------------------------------
        movf   MODO, W              ; MODO=6?
        xorlw  6
        btfss  STATUS, Z
        goto   L6_TeclaF
        movf   Dato_AuxH, W
        addlw -0x30
        movwf          Aux_Tiempo

        movlw          ' '
        xorwf  Dato_AuxH, W
        btfsc  STATUS, Z
        clrf      Aux_Tiempo
        swapf Aux_Tiempo, f

        movf   Dato_AuxL, W
        addlw -0x30
        addwf Aux_Tiempo, f

        movf   Aux_Tiempo, W
        sublw 0x59
```

```
        btfss   STATUS, C
        goto    $+3
        movf    Aux_Tiempo, W
        movwf       Segundos
        movlw       0
        movwf       MODO
L6_TeclaF
;-------------------------------------------------------
        movf    MODO, W             ; MODO=10?
        xorlw   .10
        btfss   STATUS, Z
        goto    L10_TeclaF
        movf    Dato_AuxH, W
        addlw -0x30
        movwf       Aux_Tiempo

        movlw       ' '
        xorwf   Dato_AuxH, W
        btfsc   STATUS, Z
        clrf    Aux_Tiempo
        swapf Aux_Tiempo, f

        movf    Dato_AuxL, W
        addlw -0x30
        addwf Aux_Tiempo, f

        movf    Aux_Tiempo, W
        sublw 0x23
        btfss   STATUS, C
        goto    $+4
        movf    Aux_Tiempo, W
        movwf       Horas_Al
        call    Guardar_Horas_Al
        movlw       9
        movwf       MODO
L10_TeclaF
;-------------------------------------------------------
        movf    MODO, W             ; MODO=11?
        xorlw   .11
        btfss   STATUS, Z
        goto    L11_TeclaF
        movf    Dato_AuxH, W
        addlw -0x30
        movwf       Aux_Tiempo

        movlw       ' '
        xorwf   Dato_AuxH, W
        btfsc   STATUS, Z
        clrf    Aux_Tiempo
        swapf Aux_Tiempo, f
```

```
        movf   Dato_AuxL, W
        addlw -0x30
        addwf Aux_Tiempo, f

        movf   Aux_Tiempo, W
        sublw 0x59
        btfss   STATUS, C
        goto   $+4
        movf   Aux_Tiempo, W
        movwf       Minutos_Al
        call    Guardar_Minutos_Al
        movlw       7
        movwf       MODO
L11_TeclaF
;-------------------------------------------------------

        goto   fin_rb

TeclaNum
        movf   MODO, W          ; MODO=1?
        xorlw  1
        btfss   STATUS, Z
        goto   $+3
        movlw       4
        movwf       MODO

        movf   MODO, W          ; MODO=2?
        xorlw  2
        btfss   STATUS, Z
        goto   $+3
        movlw       5
        movwf       MODO

        movf   MODO, W          ; MODO=3?
        xorlw  3
        btfss   STATUS, Z
        goto   $+3
        movlw       6
        movwf       MODO

        movf   MODO, W          ; MODO=8?
        xorlw  8
        btfss   STATUS, Z
        goto   $+3
        movlw       .10
        movwf       MODO

        movf   MODO, W          ; MODO=9?
        xorlw  9
```

```
        btfss   STATUS, Z
        goto    $+3
        movlw       .11
        movwf       MODO


        movlw       ' '
        xorwf   Dato_AuxH, W
        btfsc   STATUS, Z
        goto    $+4
        movlw       ' '
        movwf       Dato_AuxH
        movwf       Dato_AuxL

        movf    Dato_AuxL, W
        movwf       Dato_AuxH
        movf    TECLA, W
        movwf       Dato_AuxL
        goto    fin_rb

decod_tecla
        call    DLY1ms
        movlw       b'00001110'
        movwf       PORTB
        btfss   PORTB, 4
        retlw   '0'
        btfss   PORTB, 5
        retlw   '1'
        btfss   PORTB, 6
        retlw   '2'
        btfss   PORTB, 7
        retlw   '3'

        call    DLY1ms
        movlw       b'00001101'
        movwf       PORTB
        btfss   PORTB, 4
        retlw   '4'
        btfss   PORTB, 5
        retlw   '5'
        btfss   PORTB, 6
        retlw   '6'
        btfss   PORTB, 7
        retlw   '7'

        call    DLY1ms
        movlw       b'00001011'
        movwf       PORTB
        btfss   PORTB, 4
        retlw   '8'
```

```
        btfss   PORTB, 5
        retlw   '9'
        btfss   PORTB, 6
        retlw   'A'
        btfss   PORTB, 7
        retlw   'B'

        call    DLY1ms
        movlw       b'00000111'
        movwf       PORTB
        btfss   PORTB, 4
        retlw   'C'
        btfss   PORTB, 5
        retlw   'D'
        btfss   PORTB, 6
        retlw   'E'
        btfss   PORTB, 7
        retlw   'F'

        clrf    PORTB

        retlw   0


Inc_tiempo
        incf    Segundos, f
        movf    Segundos, W
        sublw   0x09
        btfsc   STATUS, DC
        goto    $+3
        movlw       0x06
        addwf   Segundos, f
        movf    Segundos, W
        sublw   0x59
        btfsc   STATUS, C
        return
        clrf    Segundos
        incf    Minutos, f
        movf    Minutos, W
        sublw   0x09
        btfsc   STATUS, DC
        goto    $+3
        movlw       0x06
        addwf   Minutos, f
        movf    Minutos, W
        sublw   0x59
        btfsc   STATUS, C
        return
        clrf    Minutos
        incf    Horas, f
```

```
        movf   Horas, W
        sublw 0x09
        btfsc   STATUS, DC
        goto   $+3
        movlw       0x06
        addwf Horas, f
        movf   Horas, W
        sublw 0x23
        btfsc   STATUS, C
        return
        clrf    Horas
        return


DLY1ms
        movlw       .249
        movwf       Reg_dly1
        nop
        decfsz      Reg_dly1, f
        goto   $-2
        return

DLY15ms
        movlw       .15
        movwf       Reg_dly2
        call    DLY1ms
        decfsz      Reg_dly2, f
        goto   $-2
        return

DLY4ms
        movlw       4
        goto   DLY15ms+1

DLY100us
        movlw       .34
        movwf       Reg_dly3
        decfsz      Reg_dly3, f
        goto   $-1
        return



;…….……
; Subrutina para enviar datos al LCD
;…….……
LCD_DATO
        bsf    LCD_RS
        movwf       LCD_AUX
        swapf LCD_AUX, W
```

```
        movwf       LCD_PORT
        bsf     LCD_E
        nop
        bcf     LCD_E
        movf    LCD_AUX, W
        movwf       LCD_PORT
        bsf     LCD_E
        nop
        bcf     LCD_E
        call    DLY4ms
        return


;……..……
; Subrutina para enviar instrucciones al LCD
;……..……
LCD_REG
        bcf     LCD_RS
        goto    LCD_DATO+1

;……..……
; Rutina de Inicialización del Módulo LCD
;……..……
LCD_INI
        call    DLY15ms

        bcf     LCD_RS

        movlw       b'0011'
        movwf       LCD_PORT

        bsf     LCD_E
        nop
        bcf     LCD_E
        call    DLY4ms

        movlw       b'0011'
        movwf       LCD_PORT
        bsf     LCD_E
        nop
        bcf     LCD_E
        call    DLY100us

        movlw       b'0011'
        movwf       LCD_PORT
        bsf     LCD_E
        nop
        bcf     LCD_E
        call    DLY100us
```

```
        movlw       b'0010'
        movwf       LCD_PORT
        bsf     LCD_E
        nop
        bcf     LCD_E
        call    DLY100us

        movlw       b'00101000' ; Function Set
        call    LCD_REG
        movlw       b'00001100' ; Display ON/OFF
        call    LCD_REG
        movlw       b'00000001' ; Clear Display
        call    LCD_REG
        movlw       b'00000110' ; Entry Mode Set
        call    LCD_REG
        return


EEPROM_READ
        bcf     STATUS, RP0     ; Banco 0
        movwf       EEADR               ;
        bsf     STATUS, RP0     ; Banco 1
        bsf     EECON1, RD      ; Start read operation
        bcf     STATUS, RP0     ; Banco 0
        movf    EEDATA, W       ; W = EEDATA
        return

EEPROM_WRITE
        bsf     STATUS, RP0     ; Banco 1
        btfsc   EECON1, WR      ; Esperar fin
        goto    $-1             ; de escritura
        bcf     STATUS, RP0     ; Banco 0
        movwf       EEADR
        bsf     STATUS, RP0     ;Bank 1
        bsf     EECON1, WREN    ;Enable writes

        movlw       0x55        ;Write 55h to
        movwf       EECON2              ;EECON2
        movlw       0xAA        ;Write AAh to
        movwf       EECON2              ;EECON2

        bsf     EECON1, WR
        bcf     EECON1, WREN
        bcf     STATUS, RP0     ; Banco 0
        return

Guardar_Horas_Al
        movf    Horas_Al, W
        movwf       EEDATA
        decf    Num_Alarma, W
```

```
        addwf Num_Alarma, W
        addlw -1
        call    EEPROM_WRITE
        return

Guardar_Minutos_Al
        movf  Minutos_Al, W
        movwf       EEDATA
        decf  Num_Alarma, W
        addwf Num_Alarma, W
        call    EEPROM_WRITE
        return



        ORG  0x2100
        DE    09,10,10,30,11,00,11,30,12,00
        DE    13,00,13,30,14,00,14,30,15,00

        END                 ; directive 'end of program'
```

3.2 DISEÑO TÉCNICO

3.2.1 CÁLCULO DE LAS POTENCIAS REQUERIDAS PARA EL CORRECTO FUNCIONAMIENTO ELÉCTRICO DEL SISTEMA.

Para el cálculo de potencia se basa en las características eléctricas máximas admisibles para el microcontrolador PIC16F84 con una frecuencia de 4 MHz, donde se toma en consideración la corriente máxima de salida en alto por el conjunto del puerto A mas la corriente máxima de salida en bajo por el conjunto del puerto B y las corrientes de de Vss y Vdd:

100mA+150mA+150mA+100mA=500 mA

Con este cálculo podemos deducir que el fusible ideal en la entrada del circuito es de 500mA.

Para la protección del circuito del relé se implanta un fusible de 5 A por lo que en nuestra sirena ya esta especificado por el fabricante.

3.2.2 DISEÑO DE LA FUENTE DE PODER.

Para la fuente de poder del circuito se utilizó un transformador de 110 VAC que modifica la señal a 9 VAC, la señal ingresa a través de un puente de diodos el cual entrega una señal rectificada de 12VDC la cual ingresa a un grupo de condensadores para eliminar el rizado de la señal, se utilizó un diodo zener de 5.1V para modificar la señal de 12 VDC a 5.1VDC como se puede observar en la figura 2.7.

La ventaja que presenta la fuente de poder es que a través de esta fuente se puede alimentar el circuito del PIC16F84 con los 5.1VDC y al circuito del relee con los 12VDC.

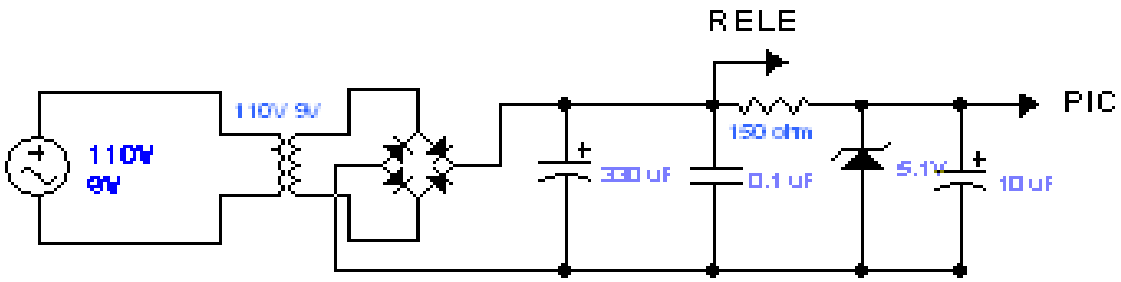


Figura 2.7 Fuente de alimentación para el marcador de tiempo

3.2.3 CONSTRUCCIÓN DEL PROTOTIPO.

Para la construcción del prototipo se basa en la figura (2.11) que muestra de una manera detalla todos los elementos utilizados en el proyecto denominado marcador de tiempo programable, el mismo que sirve para armar en las placas diseñas anteriormente y encontrar daños, realizar modificaciones en caso de que se necesitará.

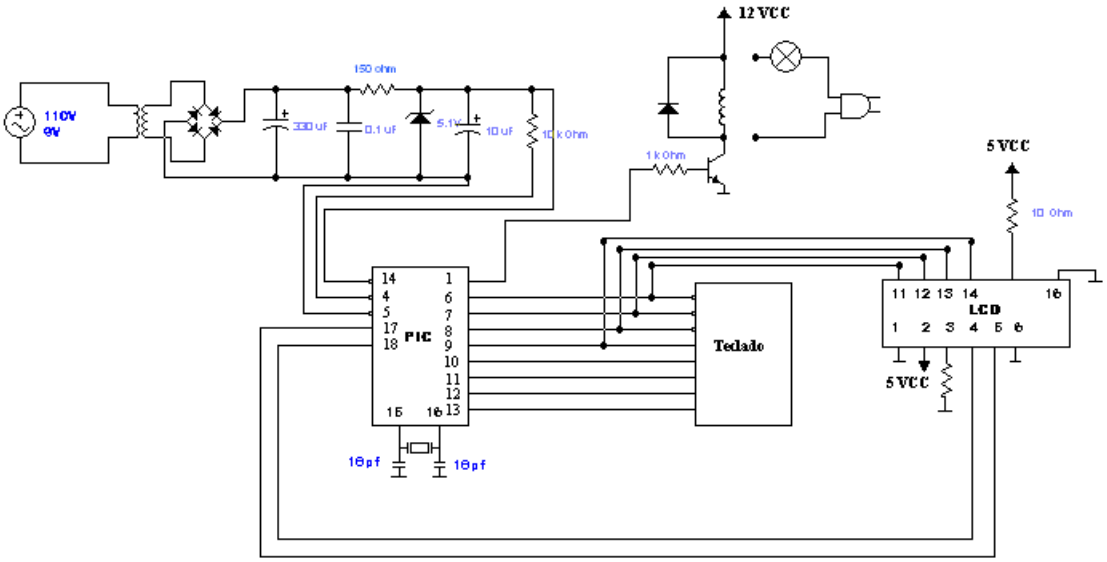


Figura 2.11 Diagrama del marcador de tiempo

Una vez ya diseñado el circuito y tenerlo en placas el siguiente paso es soldar todos los elementos que se va a utilizar para este paso es muy conveniente polarizar adecuadamente los elementos a utilizar en caso de que sea conveniente y antes de ponerlos verificar que estén en buen estado para luego no tener inconvenientes en el resultado final, debe verificar que no estén haciendo corto entre los elementos y que la suelda sea limpiada correctamente para que no exista desgaste de la suelda en el futuro.

Antes de soldar el PIC no olvidar grabarlo como se lo indicó anteriormente y verificar que todos sus pines estén en buen estado.

3.2.4 DISEÑO DE LOS CIRCUITOS IMPRESOS.

Para la elaboración de los circuitos impresos se utilizó el software Circuit Marke, una vez ya completado el trabajo con dicho software se ingresa al estudio de fotograbado en el cual se obtiene la placa lista para soldar todos los elementos que están en el circuito

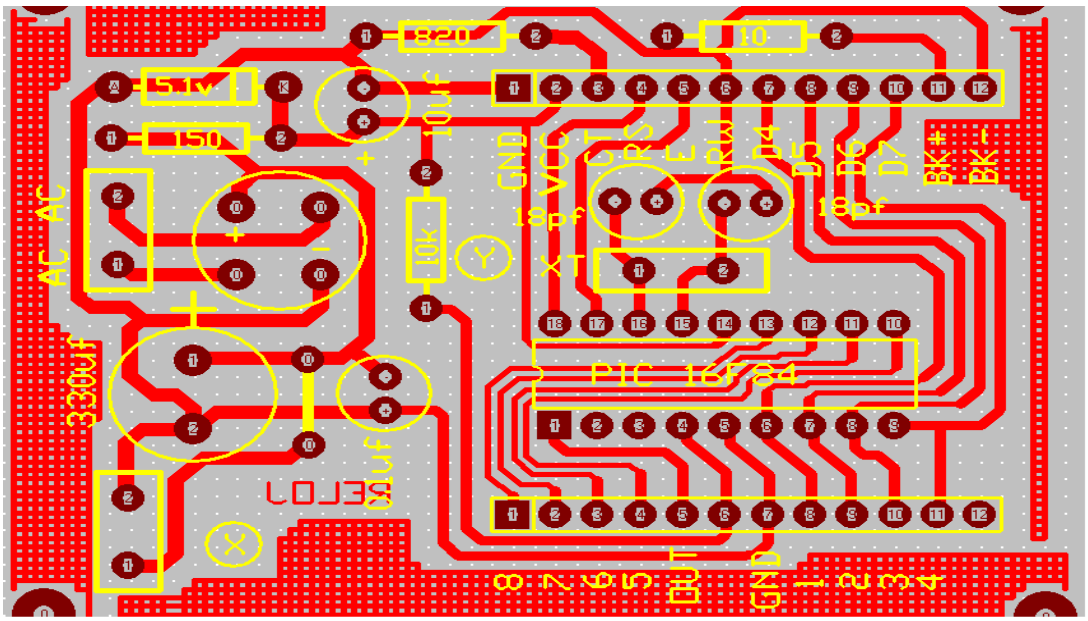


Figura 2.8 Circuito impreso para el funcionamiento del PIC16F84

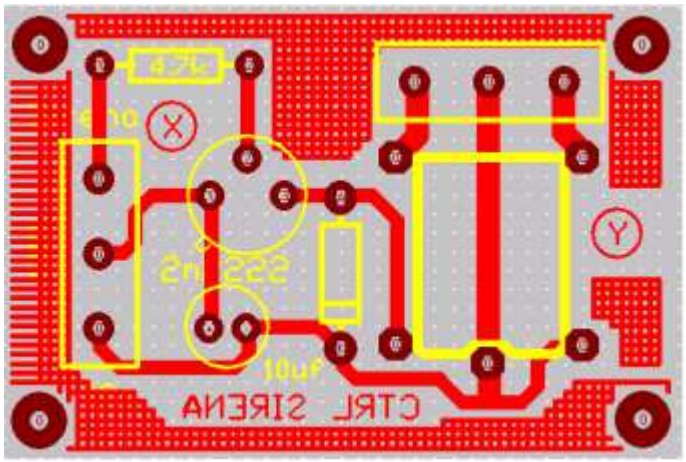


Figura 2.9 Circuito impreso para el control de la sirena

3.2.5 DISEÑO DE UNA CAJA QUE CONTENDRÁ LOS CIRCUITOS

La caja que contiene los circuitos está diseñada para poderlo abrir con facilidad en caso de que exista una avería en el circuito, en la parte exterior está compuesta por un LCD para poder visualisar los tiempos de alarma y la hora programada, además consta de un teclado el cual se utiliza para ingresar tiempos de alarma requerido y hora fijada.

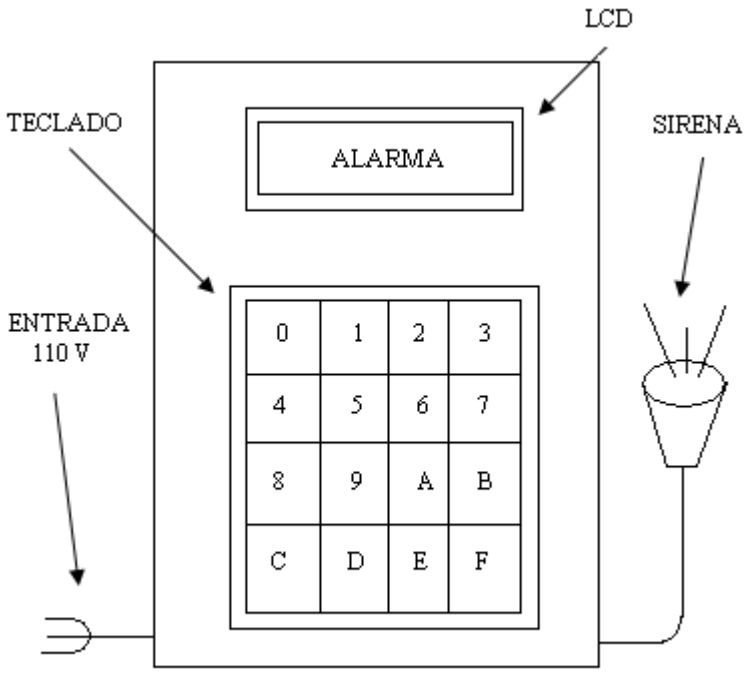


Figura 2.10 diseño de la caja que contiene el marcador de tiempo programable

En el interior de la caja contiene dos placas una que contiene el LCD y comanda todo el programa, y la segunda placa que contiene el relé la misma que pone en funcionamiento la sirena.

4 PRUEBAS DE FUNCIONAMIENTO Y MANUAL DE OPERACIÓN

4.1 PRUEBA DE CADA UNO DE LOS BLOQUES FUNCIONALES DEL SISTEMA.

En las pruebas funcionales del sistema se empieza probando la fuente de poder que tenga los 5.1 voltios que es la base fundamental para la alimentación del PIC16F84 una vez ya probado esto a través de un multímetro digital se instala el circuito del marcador de tiempo programable.

Para probar el correcto funcionamiento del software apoyándose en el programa Proteus el cual permite simular el programa y verificar si cumple con las expectativas propuestas.

Ya probado el programa en Proteus se lo graba en el PIC16F84 y con un cronómetro se verifica el tiempo estipulado en el programa que es de diez segundos para la duración de cada tiempo de programación, a través de esto se asegura que los tiempos son correctos.

A través de estas pruebas se asegura que el marcador de tiempo programable funcionará con éxito.

4.2 PRUEBAS ESTÁTICAS DEL SISTEMA EN CONJUNTO

Para las pruebas estáticas es necesario un medidor de continuidad con el cual se puede verificar continuidad en cada una de las pistas que se utiliza en el sistema, al momento de presentarse un corto o separación entre pistas es necesario corregir para el correcto funcionamiento del sistema.

Concluido con éxito las pruebas estáticas se procede a soldar los materiales requeridos, en ciertos materiales verificando la polaridad si lo tienen.

4.3 PRUEBAS DINÁMICAS DEL SISTEMA EN CONJUNTO

Se verifica el voltaje requerido a la salida de la fuente de poder, el voltaje en la alimentación del PIC y del LCD, en las siguientes pruebas se puede dar cuenta

que la placa está soldada correctamente y cumple con lo requerido para el funcionamiento del sistema.

4.3 MANUAL DE OPERACIÓN PARA EL MARCADOR DE TIEMPO PROGRAMABLE.

A continuación se explicará de una manera detallada el correcto funcionamiento para manipular el marcador de tiempo programable.

Cuando se enciende el marcador de tiempo programable se vé la imperiosa necesidad de ajustar una hora para ello se sigue los siguientes pasos.

1. Presione la tecla **B** y ajuste la hora.
2. Para ajustar minutos presione la tecla **F** y ajuste los minutos.
3. Para ajustar segundos presione la tecla **F** y ajuste segundos.
4. Para que muestre la hora ya fijada presiono la tecla **F**.

Para ingresar a los tiempos de alarma:

Presione la tecla **E**.

Para observar los tiempos de alarma.

1. Presione la tecla **C** y da una visualización descendente.
2. Presione la tecla **D** y da una visualización ascendente.

Para ajustar la alarma a los tiempos que sean requeridos, seleccione la alarma con las teclas ya indicadas **C** y **D** una vez ya seleccionado.

1. Presione la tecla **B** y le permite ajustar la hora.
2. Presione la tecla **F** y le permite ajustar minutos.
3. Para que muestre la alarma ya fijada presione la tecla **F.**

Para seguir ajustando los tiempos de memoria seleccione con las teclas **D** y **C**.

Para salir del modo de alarma.

1. Presione la tecla **E.**

Una vez ya fuera del modo de alarma entra en el modo reloj.

CAPÍTULO III

CONCLUSIONES

- La culminación de este proyecto dentro del Colegio Jan Amós Comenius ayudará a realizar el cambio de horas de una manera automática sin tener que depender del recurso humano para realizar dicha función.

- Con la ejecución del proyecto se obtiene un sistema que permite informar a todos los docentes de su inicio y fin de hora clase.

- Con la utilización del PIC16F84 se pudo satisfacer completamente con todas las expectativas requeridas para este proyecto por que nos permite intervalos de tiempos exactos y no existe retrasos en cambios de hora.

- Se pudo conocer de manera real cómo trabajan los microcontroladores y a través de esto seguirnos involucrando más en el mundo de la electrónica y a su vez poder palpar de una manera real como la tecnología sigue avanzando rápidamente y debemos actualizados para poder ser útiles a la sociedad.

- Con la implementación del proyecto ayudará que no exista molestias entre docentes, el marcador de tiempo programable nos proporciona cambios de hora para que cada docente se pueda orientar su tiempo de hora clase.

- El presente trabajo ha servido como instrumento para poner en practica los conocimientos adquiridos durante nuestra formación en la Carrera de Aviónica y da la pauta para aseverar que la formación teórica recibida más la aplicación de la misma en la vida diaria, a través de practicas, permitirán hacer de los alumnos de nuestra escuela y del instituto en general, profesionales de alta calidad.

RECOMENDACIONES

- Con la adecuada manipulación del marcador de tiempo programable alargara la vida útil del proyecto ya mencionado.

- El marcador de tiempo programable debe estar en un lugar fijo donde se pueda manipular fácilmente y protegido de las inclemencias del tiempo.

- La sirena se la debe ubicar en un lugar despejado y de esta manera no afectar la salud auditiva de los que están involucrados en la escuela.

- Tomar todas las medidas de seguridad que estén al alcance para poder salvaguardar nuestra integridad personal, no solo en la elaboración de este proyecto sino también en futuros trabajos a realizar.

- En la información que se maneja en el manual de operación tiene que ser lo más claro posible con el fin de que cualquier persona que no tenga conocimientos de electrónica pueda manipular sin ningún problema el marcador de tiempo programable.

GLOSARIO

AUTOMATIZAR

Convertir ciertos movimientos corporales en movimientos automáticos o indeliberados.

CARGA

Cantidad de electricidad acumulada en un cuerpo

CICLO

Conjunto de una serie de fenómenos u operaciones que se repiten ordenadamente.

CIRCUITO

Conjunto de conductores que recorre una corriente eléctrica, y en el cual hay generalmente intercalados aparatos productores o consumidores de esta corriente.

DESBORDANTE

Que sale de sus límites o de la medida

FICHERO

Conjunto organizado de informaciones almacenadas en un soporte común.

INTENSIDAD ELECTRICA

Magnitud física que expresa la cantidad de electricidad que atraviesa un conductor en la unidad de tiempo. Su unidad en el Sistema Internacional es el amperio.

INTERVALO

Espacio o distancia que hay de un tiempo a otro o de un lugar a otro.

MAPEAR

Localizar y representar gráficamente la distribución relativa de las partes de un todo.

MODULAR

Modificar los factores que intervienen en un proceso para obtener distintos resultados.

NIBBLES

Medios bytes

PERIODO

Espacio de tiempo que incluye toda la duración de algo.

PIC

Control Interfase Periférico

TEMPORIZADOR

Sistema de control de tiempo que se utiliza para abrir o cerrar un circuito en uno o más momentos determinados, y que conectado a un dispositivo lo pone en acción.

TENSIÓN

Voltaje con que se realiza una transmisión de energía eléctrica.

VOLÁTIL

Mudable, inconstante.