

1. INTRODUCCION A LOS MICROCONTROLADORES.....	1
1.1 CONTROLADOR Y MICROCONTROLADOR.....	1
1.2 DIFERENCIA ENTRE MICROPROCESADOR Y MICROCONTROLADOR.....	2
1.3 APLICACIONES DE LOS MICROCONTROLADORES.....	4
1.4 EL MERCADO DE LOS MICROCONTROLADORES.....	5
1.5 ¿QUÉ MICROCONTROLADOR EMPLEAR?	6
1.6 RECURSOS COMUNES A TODOS LOS MICROCONTROLADORES.....	9
1.6.1 Arquitectura básica.....	9
1.6.2 El procesador o UCP	10
1.6.3 Memoria.....	11
1.6.4 Puertas de Entrada y Salida	14
1.6.5 Reloj principal.....	14
1.7 RECURSOS ESPECIALES.....	14
1.7.1 Temporizadores o “Timers”.....	15
1.7.2 Perro guardián o “Watchdog”	16
1.7.3 Protección ante fallo de alimentación o “Brownout”.....	16
1.7.4 Estado de reposo ó de bajo consumo	16
1.7.5 Conversor A/D (CAD)	17
1.7.6 Conversor D/A (CDA)	17
1.7.7 Comparador analógico.....	17
1.7.8 Modulador de anchura de impulsos o PWM	17
1.7.9 Puertas de E/S digitales.....	18
1.7.10 Puertas de comunicación.....	18
1.8 HERRAMIENTAS PARA EL DESARROLLO DE APLICACIONES.....	19
2. LA FAMILIA DE LOS PIC COMO ELECCIÓN.	21
2.1 CARACTERÍSTICAS RELEVANTES.....	22
2.1.1 Arquitectura.....	22
2.1.2 Segmentación.....	22
2.1.3 Formato de las instrucciones.....	23
2.1.4 Juego de instrucciones.....	23
2.1.5 Todas las instrucciones son ortogonales.....	23
2.1.6 Arquitectura basada en un “banco de registros”	23
2.1.7 Diversidad de modelos de microcontroladores con prestaciones y recursos diferentes	23
2.1.8 Herramientas de soporte potentes y económicas.....	24
2.2 LAS GAMAS DE PIC	24
2.2.1 La gama enana: PIC12C(F)XXX de 8 patitas	25
2.2.2 Gama baja o básica: PIC16C5X con instrucciones de 12 bits.....	26
2.2.3 Gama media. PIC16CXXX con instrucciones de 14 bits.....	28
2.2.4 Gama alta: PIC17CXXX con instrucciones de 16 bits.....	29
3. LOS REGISTROS DE LA GAMA MEDIA	31
3.1 ORGANIZACIÓN DE LA MEMORIA DE DATOS	31
3.2 REGISTROS ESPECÍFICOS	32
4. REPERTORIO DE INSTRUCCIONES.....	35
4.1 CARACTERÍSTICAS GENERALES	35

4.2 DEFINICIONES Y ABREVIATURAS	35
4.3 REPERTORIO DE INSTRUCCIONES DE LA GAMA MEDIA.....	37
4.4 INSTRUCCIONES DE LA GAMA BAJA.....	41
5. PROGRAMACIÓN DE LOS mCONTROLADORES PIC.....	42
5.1 INTRODUCCIÓN.....	42
5.2 EL ENTORNO DE TRABAJO MPLAB	44
5.2.1 El ensamblador	44
5.2.2 Creando un nuevo proyecto	46
5.2.3 Ensamblando.....	48
5.2.4 Simulación bajo windows	49
5.2.5 Otras opciones del MPLAB.....	50
5.3 EJEMPLOS BÁSICOS DE PROGRAMACIÓN.....	51
5.3.1 El sistema de E/S. interrupciones y LED's	51
5.3.2 Contar y visualizar	55
5.3.3 Teclado matricial	57
5.3.4 Tablas y subrutinas.....	60
5.3.5 Manejo de interrupciones	63
5.3.6 Manejo de una pantalla LCD. Creación de una librería.	67
5.3.7 Uso de una librería: LCD.LIB	77
5.3.8 El Watchdog.....	81
5.3.9 Notas para el profesor sobre la elaboración de estos programas.....	81
6. EL COMPILADOR DE C.....	83
6.1 INTRODUCCIÓN.....	83
6.2 EL PRIMER PROGRAMA EN C.....	83
6.3 ¿ QUÉ PODEMOS USAR DEL C CONVENCIONAL?	86
6.4 LIBRERÍAS Y FUNCIONES	89
6.4.1 La librería GETCHAR.....	89
6.4.2 La librería IO	90
6.4.3 Librería EE_READ.....	90
6.4.4 Librería EE_WRITE	90
6.4.5 TAMBIÉN CONVIENE SABER	91
7. EL PROGRAMADOR.....	92
7.1 INTRODUCCIÓN.....	92
7.2 DE LA PROGRAMACIÓN PARALELA A LA PROGRAMACIÓN SERIE.....	93
7.3 SOFTWARE Y UTILIZACIÓN	96
7.4 INSTRUCCIONES DE USO RESUMIDAS DEL PROGRAMADOR.....	100
8. APLICACIÓN PRÁCTICA: UN CONTADOR CONTROLADO POR INTERRUPCIÓN.....	103
9. BIBLIOGRAFÍA	107
9.1 BIBLIOGRAFÍA ESCRITA.....	107
9.2 BIBLIOGRAFÍA ELECTRÓNICA.	108

1. INTRODUCCION A LOS MICROCONTROLADORES.

Los microcontroladores están conquistando el mundo. Están presentes en nuestro trabajo, en nuestra casa y en nuestra 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 de nuestro hogar. Pero la invasión acaba de comenzar y el nacimiento del siglo **XXI** será testigo de la conquista masiva de estos diminutos computadores, que gobernarán la mayor parte de los aparatos que fabricaremos y usamos los humanos.

1.1 *Controlador y microcontrolador.*

Recibe el nombre de **controlador** el dispositivo que se emplea para el gobierno de uno o varios procesos. Por ejemplo, el controlador que regula el funcionamiento de un horno dispone de un sensor que mide constantemente su temperatura interna y, cuando traspasa los límites prefijados, genera las señales adecuadas que accionan los efectores que intentan llevar el valor de la temperatura dentro del rango estipulado.

Aunque el concepto de controlador ha permanecido invariable a través del tiempo, su implementación física ha variado frecuentemente. Hace tres décadas, los controladores se construían exclusivamente con componentes de lógica discreta, posteriormente se emplearon los microprocesadores, que se rodeaban con chips de memoria y E/S sobre una tarjeta de circuito impreso. En la actualidad, todos los elementos del controlador se han podido incluir en un chip, el cual recibe el nombre de **microcontrolador**. Realmente consiste en un sencillo pero completo computador contenido en el corazón (chip) de un circuito integrado.

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

Un microcontrolador dispone normalmente de los siguientes componentes:

- Procesador o UCP (Unidad Central de Proceso).
- Memoria RAM para Contener los datos.

- Memoria para el programa tipo ROM/PROM/EPROM.
- Líneas de E/S para comunicarse con el exterior.
- Diversos módulos para el control de periféricos (temporizadores, Puertas Serie y Paralelo, CAD: Conversores Analógico/Digital, CDA: Conversores Digital/Analógico, etc.).
- Generador de impulsos de reloj que sincronizan el funcionamiento de todo el sistema.

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

- Aumento de prestaciones: un mayor control sobre un determinado elemento representa una mejora considerable en el mismo.
- Aumento de la fiabilidad: al reemplazar el microcontrolador por un elevado número de elementos disminuye el riesgo de averías y se precisan menos ajustes.
- Reducción del tamaño en el producto acabado: La integración del microcontrolador en un chip disminuye el volumen, la mano de obra y los stocks.
- 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).

1.2 Diferencia entre microprocesador y microcontrolador.

El microprocesador es un circuito integrado que contiene la Unidad Central de Proceso (UCP), también llamada procesador, de un computador. La UCP está formada por la Unidad de Control, que interpreta las instrucciones, y el Camino de Datos, que las ejecuta.

Las patitas de un microprocesador sacan al exterior las líneas de sus buses de direcciones, datos y control, para permitir conectarle con la Memoria y los Módulos de E/S y configurar un computador implementado por varios circuitos integrados. Se dice que un microprocesador es un sistema abierto porque su configuración es variable de acuerdo con la aplicación a la que se destine. (Figura 1.1.)

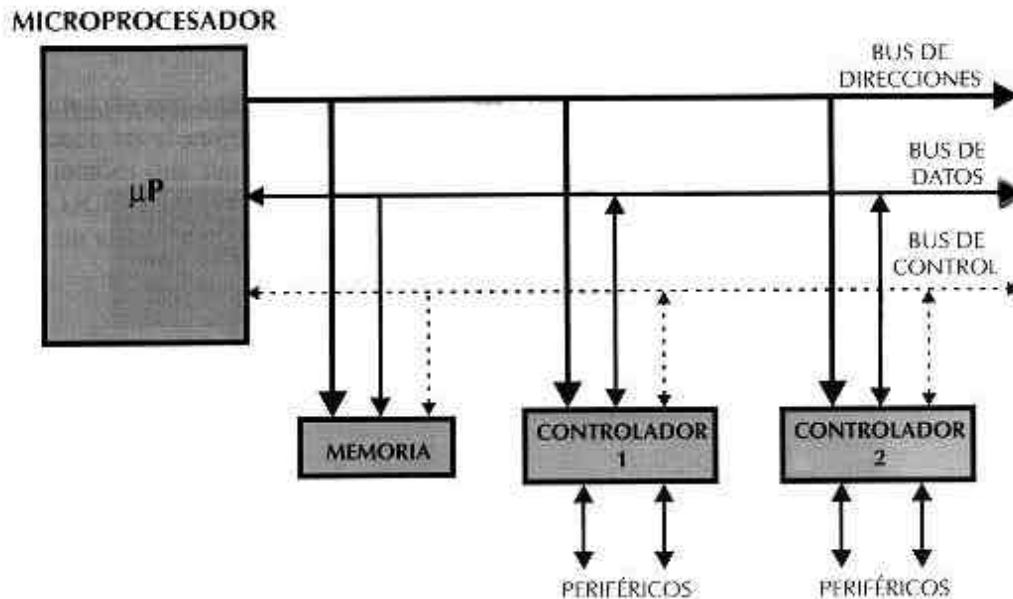


Figura 1.1. Estructura de un sistema abierto basado en un microprocesador. La disponibilidad de los buses en el exterior permite que se configure a la medida de la aplicación.

Si sólo se dispusiese de un modelo de microcontrolador, éste debería tener muy potenciados todos sus recursos para poderse adaptar a las exigencias de las diferentes aplicaciones. Esta potenciación supondría en muchos casos un despilfarro. En la práctica cada fabricante de microcontroladores oferta un elevado número de modelos diferentes, desde los más sencillos hasta los más poderosos. Es posible seleccionar la capacidad de las memorias, el número de líneas de E/S, la cantidad y potencia de los elementos auxiliares, la velocidad de funcionamiento, etc. Por todo ello, un aspecto muy destacado del diseño es la selección del microcontrolador a utilizar.

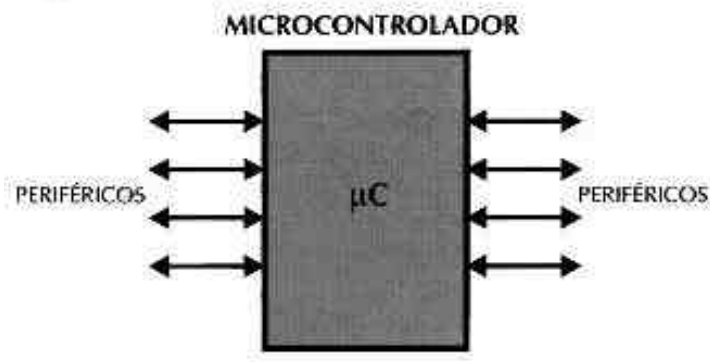


Figura 1.2. El microcontrolador es un sistema cerrado. Todas las partes del computador están contenidas en su interior y sólo salen al exterior las líneas que gobiernan los periféricos.

1.3 Aplicaciones de los microcontroladores.

Cada vez existen más productos que incorporan un microcontrolador con el fin de aumentar sustancialmente sus prestaciones, reducir su tamaño y coste, mejorar su fiabilidad y disminuir el consumo.

Algunos fabricantes de microcontroladores superan el millón de unidades de un modelo determinado producidas en **una semana**. Este dato puede dar una idea de la masiva utilización de estos componentes.

Los microcontroladores están siendo empleados en multitud de sistemas presentes en nuestra vida diaria, como pueden ser juguetes, horno microondas, frigoríficos, televisores, computadoras, impresoras, módems, el sistema de arranque de nuestro coche, etc. Y otras aplicaciones con las que seguramente no estaremos tan familiarizados como instrumentación electrónica, control de sistemas en una nave espacial, etc. Una aplicación típica podría emplear varios microcontroladores para controlar pequeñas partes del sistema. Estos pequeños controladores podrían comunicarse entre ellos y con un procesador central, probablemente más potente, para compartir la información y coordinar sus acciones, como, de hecho, ocurre ya habitualmente en cualquier PC.

1.4 El mercado de los microcontroladores.

Aunque en el mercado de la microinformática la mayor atención la acaparan los desarrollos de los microprocesadores, lo cierto es que se venden cientos de microcontroladores por cada uno de aquéllos.

Existe una gran diversidad de microcontroladores. Quizá la clasificación más importante sea entre microcontroladores de 4, 8, 16 ó 32 bits. Aunque las prestaciones de los microcontroladores de 16 y 32 bits son superiores a los de 4 y 8 bits, la realidad es que los microcontroladores de 8 bits dominan el mercado y los de 4 bits se resisten a desaparecer. La razón de esta tendencia es que los microcontroladores de 4 y 8 bits son apropiados para la gran mayoría de las aplicaciones, lo que hace absurdo emplear micros más potentes y consecuentemente más caros. Uno de los sectores que más tira del mercado del microcontrolador es el mercado automovilístico. De hecho, algunas de las familias de microcontroladores actuales se desarrollaron pensando en este sector, siendo modificadas posteriormente para adaptarse a sistemas más genéricos. El mercado del automóvil es además uno de los más exigentes: los componentes electrónicos deben operar bajo condiciones extremas de vibraciones, choques, ruido, etc. y seguir siendo fiables. El fallo de cualquier componente en un automóvil puede ser el origen de un accidente.

En cuanto a las técnicas de fabricación, cabe decir que prácticamente la totalidad de los microcontroladores actuales se fabrican con tecnología CMOS 4 (Complementary Metal Oxide Semiconductor). Esta tecnología supera a las técnicas anteriores por su bajo consumo y alta inmunidad al ruido.

La distribución de las ventas según su aplicación es la siguiente:

- Una tercera parte se absorbe en las aplicaciones relacionadas con los computadores y sus periféricos.
- La cuarta parte se utiliza en las aplicaciones de consumo (electrodomésticos, juegos, TV, vídeo, etc.)
- El 16% de las ventas mundiales se destinó al área de las comunicaciones.
- Otro 16% fue empleado en aplicaciones industriales.

- El resto de los microcontroladores vendidos en el mundo, aproximadamente un 10% fueron adquiridos por las industrias de automoción.

También los modernos microcontroladores de 32 bits van afianzando sus posiciones en el mercado, siendo las áreas de más interés el procesamiento de imágenes, las comunicaciones, las aplicaciones militares, los procesos industriales y el control de los dispositivos de almacenamiento masivo de datos.

1.5 ¿Qué microcontrolador emplear?

A la hora de escoger el microcontrolador a emplear en un diseño concreto hay que tener en cuenta multitud de factores, como la documentación y herramientas de desarrollo disponibles y su precio, la cantidad de fabricantes que lo producen y por supuesto las características del microcontrolador (tipo de memoria de programa, número de temporizadores, interrupciones, etc.):

Costes. Como es lógico, los fabricantes de microcontroladores compiten duramente para vender sus productos. Y no les va demasiado mal ya que sin hacer demasiado ruido venden 10 veces más microcontroladores que microprocesadores.

Para que nos hagamos una idea, para el fabricante que usa el microcontrolador en su producto una diferencia de precio en el microcontrolador de algunas pesetas es importante (el consumidor deberá pagar además el coste del empaquetado, el de los otros componentes, el diseño del hardware y el desarrollo del software). Si el fabricante desea reducir costes debe tener en cuenta las herramientas de apoyo con que va a contar: emuladores, simuladores, ensambladores, compiladores, etc. Es habitual que muchos de ellos siempre se decanten por microcontroladores pertenecientes a una única familia.

Aplicación. Antes de seleccionar un microcontrolador es imprescindible analizar los requisitos de la aplicación:

- *Procesamiento de datos:* puede ser necesario que el microcontrolador realice cálculos críticos en un tiempo limitado. En ese caso debemos asegurarnos de seleccionar un dispositivo suficientemente rápido para ello. Por otro lado, habrá que tener en cuenta la precisión de los datos a manejar: si no es suficiente con un

microcontrolador de 8 bits, puede ser necesario acudir a microcontroladores de 16 ó 32 bits, o incluso a hardware de coma flotante. Una alternativa más barata y quizá suficiente es usar librerías para manejar los datos de alta precisión. -

- *Entrada Salida:* para determinar las necesidades de Entrada/Salida del sistema es conveniente dibujar un diagrama de bloques del mismo, de tal forma que sea sencillo identificar la cantidad y tipo de señales a controlar. Una vez realizado este análisis puede ser necesario añadir periféricos hardware externos o cambiar a otro microcontrolador más adecuado a ese sistema.
- *Consumo:* algunos productos que incorporan microcontroladores están alimentados con baterías y su funcionamiento puede ser tan vital como activar una alarma antirrobo. Lo más conveniente en un caso como éste puede ser que el microcontrolador esté en estado de bajo consumo pero que despierte ante la activación de una señal (una interrupción) y ejecute el programa adecuado para procesarla.
- *Memoria:* para detectar las necesidades de memoria de nuestra aplicación debemos separarla en memoria volátil (RAM), memoria no volátil (ROM, EPROM, etc.) y memoria no volátil modificable (EEPROM). Este último tipo de memoria puede ser útil para incluir información específica de la aplicación como un número de serie o parámetros de calibración.

El tipo de memoria a emplear vendrá determinado por el volumen de ventas previsto del producto: de menor a mayor volumen será conveniente emplear EPROM, OTP y ROM. En cuanto a la cantidad de memoria necesaria puede ser imprescindible realizar una versión preliminar, aunque sea en pseudo-código, de la aplicación y a partir de ella hacer una estimación de cuánta memoria volátil y no volátil es necesaria y si es conveniente disponer de memoria no volátil modificable.

- *Ancho de palabra:* el criterio de diseño debe ser seleccionar el microcontrolador de menor ancho de palabra que satisfaga los requerimientos de la aplicación. Usar un microcontrolador de 4 bits supondrá una reducción en los costes importante,

mientras que uno de 8 bits puede ser el más adecuado si el ancho de los datos es de un byte. Los microcontroladores de 16 y 32 bits, debido a su elevado coste, deben reservarse para aplicaciones que requieran sus altas prestaciones (Entrada/Salida potente o espacio de direccionamiento muy elevado).

- *Diseño de la placa:* la selección de un microcontrolador concreto condicionará el diseño de la placa de circuitos. Debe tenerse en cuenta que quizá usar un microcontrolador barato encarezca el resto de componentes del diseño.

Los microcontroladores más populares se encuentran, sin duda, entre las mejores elecciones:

8048 (Intel). Es el padre de los microcontroladores actuales, el primero de todos. Su precio, disponibilidad y herramientas de desarrollo hacen que todavía sea muy popular.

8051 (Intel y otros). Es sin duda el microcontrolador más popular. Fácil de programar, pero potente. Está bien documentado y posee cientos de variantes e incontables herramientas de desarrollo.

80186, 80188 y 80386 EX (Intel). Versiones en microcontrolador de los populares microprocesadores 8086 y 8088. Su principal ventaja es que permiten aprovechar las herramientas de desarrollo para PC.

68HC11 (Motorola y Toshiba). Es un microcontrolador de 8 bits potente y popular con gran cantidad de variantes.

683xx (Motorola). Surgido a partir de la popular familia 68k, a la que se incorporan algunos periféricos. Son microcontroladores de altísimas prestaciones.

PIC (MicroChip). Familia de microcontroladores que gana popularidad día a día. Fueron los primeros microcontroladores RISC.

Es preciso resaltar en este punto que existen innumerables familias de microcontroladores, cada una de las cuales posee un gran número de variantes.

1.6 Recursos comunes a todos los microcontroladores.

Al estar todos los microcontroladores integrados en un chip, su estructura fundamental y sus características básicas son muy parecidas. Todos deben disponer de los bloques esenciales Procesador, memoria de datos y de instrucciones, líneas de E/S, oscilador de reloj y módulos controladores de periféricos. Sin embargo, cada fabricante intenta enfatizar los recursos más idóneos para las aplicaciones a las que se destinan preferentemente.

En este apartado se hace un recorrido de todos los recursos que se hallan en todos los microcontroladores describiendo las diversas alternativas y opciones que pueden encontrarse según el modelo seleccionado.

1.6.1 Arquitectura básica

Aunque inicialmente todos los microcontroladores adoptaron la arquitectura clásica de von Neumann, en el momento presente se impone la arquitectura Harvard. La arquitectura de von Neumann se caracteriza por disponer de una sola memoria principal donde se almacenan datos e instrucciones de forma indistinta. A dicha memoria se accede a través de un sistema de buses único (direcciones, datos y control).

La arquitectura Harvard dispone de dos memorias independientes una, que contiene sólo instrucciones y otra, sólo datos. Ambas disponen de sus respectivos sistemas de buses de acceso y es posible realizar operaciones de acceso (lectura o escritura) simultáneamente en ambas memorias. Figura 1.3.



Figura 1.3. La arquitectura Harvard dispone de dos memorias independientes para datos y para instrucciones, permitiendo accesos simultáneos.

Los microcontroladores PIC responden a la arquitectura Harvard.

1.6.2 El procesador o UCP

Es el elemento más importante del microcontrolador y determina sus principales características, tanto a nivel hardware como software.

Se encarga de direccionar la memoria de instrucciones, recibir el código OP de la instrucción en curso, su decodificación y la ejecución de la operación que implica la instrucción, así como la búsqueda de los operandos y el almacenamiento del resultado.

Existen tres orientaciones en cuanto a la arquitectura y funcionalidad de los procesadores actuales.

CISC: Un gran número de procesadores usados en los microcontroladores están basados en la filosofía CISC (Computadores de Juego de Instrucciones Complejo). Disponen de más de 80 instrucciones máquina en su repertorio, algunas de las cuales son muy sofisticadas y potentes, requiriendo muchos ciclos para su ejecución. Una ventaja de los procesadores CISC es que ofrecen al programador instrucciones complejas que actúan como macros.

RISC: Tanto la industria de los computadores comerciales como la de los microcontroladores están decantándose hacia la filosofía RISC (Computadores de Juego de Instrucciones Reducido). En estos procesadores el repertorio de instrucciones máquina es muy reducido y las instrucciones son simples y, generalmente, se ejecutan en un ciclo.

La sencillez y rapidez de las instrucciones permiten optimizar el hardware y el software del procesador.

SISC: En los microcontroladores destinados a aplicaciones muy concretas, el juego de instrucciones, además de ser reducido, es “específico”, o sea, las instrucciones se adaptan a las necesidades de la aplicación prevista. Esta filosofía se ha bautizado con el nombre de SISC (Computadores de Juego de Instrucciones Específico).

1.6.3 Memoria

En los microcontroladores la memoria de instrucciones y datos está integrada en el propio chip. Una parte debe ser no volátil, tipo ROM, y se destina a contener el programa de instrucciones que gobierna la aplicación. Otra parte de memoria será tipo RAM, volátil, y se destina a guardar las variables y los datos.

Hay dos peculiaridades que diferencian a los microcontroladores de los computadores personales:

1. No existen sistemas de almacenamiento masivo como disco duro o disquetes.
2. Como el microcontrolador sólo se destina a una tarea en la memoria ROM, sólo hay que almacenar un único programa de trabajo.

La RAM en estos dispositivos es de poca capacidad pues sólo debe contener las variables y los cambios de información que se produzcan en el transcurso del programa. Por otra parte, como sólo existe un programa activo, no se requiere guardar una copia del mismo en la RAM pues se ejecuta directamente desde la ROM.

Los usuarios de computadores personales están habituados a manejar Megabytes de memoria, pero, los diseñadores con microcontroladores trabajan con capacidades de ROM comprendidas entre 512 bytes y 8 k bytes y de RAM comprendidas entre 20 y 512 bytes.

Según el tipo de memoria ROM que dispongan los microcontroladores, la aplicación y utilización de los mismos es diferente. Se describen las cinco versiones de memoria no volátil que se pueden encontrar en los microcontroladores del mercado.

1º. ROM con máscara

Es una memoria no volátil de sólo lectura cuyo contenido se graba durante la fabricación del chip. El elevado coste del diseño de la máscara sólo hace aconsejable el

empleo de los microcontroladores con este tipo de memoria cuando se precisan cantidades superiores a varios miles de unidades.

2ª. OTP

El microcontrolador contiene una memoria no volátil de sólo lectura “programable una sola vez” por el usuario. OTP (One Time Programmable). Es el usuario quien puede escribir el programa en el chip mediante un sencillo grabador controlado por un programa desde un PC.

La versión OTP es recomendable cuando es muy corto el ciclo de diseño del producto, o bien, en la construcción de prototipos y series muy pequeñas.

Tanto en este tipo de memoria como en la EPROM, se suele usar la encriptación mediante fusibles para proteger el código contenido.

3ª EPROM

Los microcontroladores que disponen de memoria EPROM (Erasable Programmable Read Only Memory) pueden borrarse y grabarse muchas veces. La grabación se realiza, como en el caso de los OTP, con un grabador gobernado desde un PC. Si, posteriormente, se desea borrar el contenido, disponen de una ventana de cristal en su superficie por la que se somete a la EPROM a rayos ultravioleta durante varios minutos. Las cápsulas son de material cerámico y son más caros que los microcontroladores con memoria OTP que están hechos con material plástico.

4ª EEPROM

Se trata de memorias de sólo lectura, programables y borrables eléctricamente EEPROM (Electrical Erasable Programmable Read Only Memory). Tanto la programación como el borrado, se realizan eléctricamente desde el propio grabador y bajo el control programado de un PC. Es muy cómoda y rápida la operación de grabado y la de borrado. No disponen de ventana de cristal en la superficie.

Los microcontroladores dotados de memoria EEPROM una vez instalados en el circuito, pueden grabarse y borrarse cuantas veces se quiera sin ser retirados de dicho circuito. Para ello se usan “grabadores en circuito” que confieren una gran flexibilidad y rapidez a la hora de realizar modificaciones en el programa de trabajo.

El número de veces que puede grabarse y borrarse una memoria EEPROM es finito, por lo que no es recomendable una reprogramación continua. Son muy idóneos para la enseñanza y la Ingeniería de diseño.

Se va extendiendo en los fabricantes la tendencia de incluir una pequeña zona de memoria EEPROM en los circuitos programables para guardar y modificar cómodamente una serie de parámetros que adecuan el dispositivo a las condiciones del entorno.

Este tipo de memoria es relativamente lenta.

5ª FLASH

Se trata de una memoria no volátil, de bajo consumo, que se puede escribir y borrar. Funciona como una ROM y una RAM pero consume menos y es más pequeña.

A diferencia de la ROM, la memoria FLASH es programable en el circuito. Es más rápida y de mayor densidad que la EEPROM.

La alternativa FLASH está recomendada frente a la EEPROM cuando se precisa gran cantidad de memoria de programa no volátil. Es más veloz y tolera más ciclos de escritura/borrado.

Las memorias EEPROM y FLASH son muy útiles al permitir que los microcontroladores que las incorporan puedan ser reprogramados “en circuito”, es decir, sin tener que sacar el circuito integrado de la tarjeta. Así, un dispositivo con este tipo de memoria incorporado al control del motor de un automóvil permite que pueda modificarse el programa durante la rutina de mantenimiento periódico, compensando los desgastes y otros factores tales como la compresión, la instalación de nuevas piezas,

etc. La reprogramación del microcontrolador puede convertirse en una labor rutinaria dentro de la puesta a punto.

1.6.4 Puertas de Entrada y Salida

La principal utilidad de las patitas que posee la cápsula que contiene un microcontrolador es soportar las líneas de E/S que comunican al computador interno con los periféricos exteriores.

Según los controladores de periféricos que posea cada modelo de microcontrolador, las líneas de E/S se destinan a proporcionar el soporte a las señales de entrada, salida y control.

1.6.5 Reloj principal

Todos los microcontroladores disponen de un circuito oscilador que genera una onda cuadrada de alta frecuencia, que configura los impulsos de reloj usados en la sincronización de todas las operaciones del sistema.

Generalmente, el circuito de reloj está incorporado en el microcontrolador y sólo se necesitan unos pocos componentes exteriores para seleccionar y estabilizar la frecuencia de trabajo. Dichos componentes suelen consistir en un cristal de cuarzo junto a elementos pasivos o bien un resonador cerámico o una red R-C.

Aumentar la frecuencia de reloj supone disminuir el tiempo en que se ejecutan las instrucciones pero lleva aparejado un incremento del consumo de energía.

1.7 RECURSOS ESPECIALES

Cada fabricante oferta numerosas versiones de una arquitectura básica de microcontrolador. En algunas amplía las capacidades de las memorias, en otras incorpora nuevos recursos, en otras reduce las prestaciones al mínimo para aplicaciones muy simples, etc. La labor del diseñador es encontrar el modelo mínimo que satisfaga todos los requerimientos de su aplicación. De esta forma, minimizará el coste, el hardware y el software.

Los principales recursos específicos que incorporan los microcontroladores son:

- Temporizadores o “Timers”.
- Perro guardián o “Watchdog”.
- Protección ante fallo de alimentación o “Brownout”.
- Estado de reposo o de bajo consumo.
- Conversor A/D.
- Conversor D/A.
- Comparador analógico.
- Modulador de anchura de impulsos o PWM.
- Puertas de E/S digitales.
- Puertas de comunicación.

1.7.1 Temporizadores o “Timers”

Se emplean para controlar periodos de tiempo (temporizadores) y para llevar la cuenta de acontecimientos que suceden en el exterior (contadores).

Para la medida de tiempos se carga un registro con el valor adecuado y a continuación dicho valor se va incrementando o decrementando al ritmo de los impulsos de reloj o algún múltiplo hasta que se desborde y llegue a 0, momento en el que se produce un aviso.

Cuando se desean contar acontecimientos que se materializan por cambios de nivel o flancos en alguna de las patitas del microcontrolador, el mencionado registro se va incrementando o decrementando al ritmo de dichos impulsos.

1.7.2 Perro guardián o “Watchdog”

Cuando el computador personal se bloquea por un fallo del software u otra causa, se pulsa el botón del reset y se reinicializa el sistema. Pero un microcontrolador funciona sin el control de un supervisor y de forma continuada las 24 horas del día. El Perro guardián consiste en un temporizador que, cuando se desborda y pasa por 0, provoca un reset automáticamente en el sistema.

Se debe diseñar el programa de trabajo que controla la tarea de forma que refresque o inicialice al Perro guardián antes de que provoque el reset. Si falla el programa o se bloquea, no se refrescará al Perro guardián y, al completar su temporización, “ladrará y ladrará” hasta provocar el reset.

1.7.3 Protección ante fallo de alimentación o “Brownout”

Se trata de un circuito que resetea al microcontrolador cuando el voltaje de alimentación (VDD) es inferior a un voltaje mínimo (“brownout”). Mientras el voltaje de alimentación sea inferior al de brownout el dispositivo se mantiene reseteado, comenzando a funcionar normalmente cuando sobrepasa dicho valor.

1.7.4 Estado de reposo ó de bajo consumo

Son abundantes las situaciones reales de trabajo en que el microcontrolador debe esperar, sin hacer nada, a que se produzca algún acontecimiento externo que le ponga de nuevo en funcionamiento. Para ahorrar energía, (factor clave en los aparatos portátiles), los microcontroladores disponen de una instrucción especial (SLEEP en los PIC), que les pasa al estado de reposo o de bajo consumo, en el cual los requerimientos de potencia son mínimos. En dicho estado se detiene el reloj principal y se “congelan” sus circuitos asociados, quedando sumido en un profundo “sueño” el microcontrolador. Al

activarse una interrupción ocasionada por el acontecimiento esperado, el microcontrolador se despierta y reanuda su trabajo.

1.7.5 Conversor A/D (CAD)

Los microcontroladores que incorporan un Conversor A/D (Analógico/Digital) pueden procesar señales analógicas, tan abundantes en las aplicaciones. Suelen disponer de un multiplexor que permite aplicar a la entrada del CAD diversas señales analógicas desde las patitas del circuito integrado.

1.7.6 Conversor D/A (CDA)

Transforma los datos digitales obtenidos del procesamiento del computador en su correspondiente señal analógica que saca al exterior por una de las patitas de la cápsula. Existen muchos efectores que trabajan con señales analógicas.

1.7.7 Comparador analógico

Algunos modelos de microcontroladores disponen internamente de un Amplificador Operacional que actúa como comparador entre una señal fija de referencia y otra variable que se aplica por una de las patitas de la cápsula. La salida del comparador proporciona un nivel lógico 1 ó 0 según una señal sea mayor o menor que la otra.

También hay modelos de microcontroladores con un módulo de tensión de referencia que proporciona diversas tensiones de referencia que se pueden aplicar en los comparadores.

1.7.8 Modulador de anchura de impulsos o PWM

Son circuitos que proporcionan en su salida impulsos de anchura variable, que se ofrecen al exterior a través de las patitas del encapsulado.

1.7.9 Puertas de E/S digitales

Todos los microcontroladores destinan algunas de sus patitas a soportar líneas de E/S digitales. Por lo general, estas líneas se agrupan de ocho en ocho formando Puertas.

Las líneas digitales de las Puertas pueden configurarse como Entrada o como Salida cargando un 1 ó un 0 en el bit correspondiente de un registro destinado a su configuración.

1.7.10 Puertas de comunicación

Con objeto de dotar al microcontrolador de la posibilidad de comunicarse con otros dispositivos externos, otros buses de microprocesadores, buses de sistemas, buses de redes y poder adaptarlos con otros elementos bajo otras normas y protocolos. Algunos modelos disponen de recursos que permiten directamente esta tarea, entre los que destacan:

- UART, adaptador de comunicación serie asíncrona.
- USART, adaptador de comunicación serie síncrona y asíncrona
- Puerta paralela esclava para poder conectarse con los buses de otros microprocesadores.
- USB (Universal Serial Bus), que es un moderno bus serie para los PC.
- Bus I²C, que es un interfaz serie de dos hilos desarrollado por Philips.
- CAN (Controller Area Network), para permitir la adaptación con redes de conexión multiplexado desarrollado conjuntamente por Bosch e Intel para el cableado de dispositivos en automóviles. En EE.UU. se usa el J1850.

1.8 Herramientas para el desarrollo de aplicaciones.

Uno de los factores que más importancia tiene a la hora de seleccionar un microcontrolador entre todos los demás es el soporte tanto software como hardware de que dispone. Un buen conjunto de herramientas de desarrollo puede ser decisivo en la elección, ya que pueden suponer una ayuda inestimable en el desarrollo del proyecto.

Las principales herramientas de ayuda al desarrollo de sistemas basados en microcontroladores son:

- Desarrollo del software:

Ensamblador. La programación en lenguaje ensamblador puede resultar un tanto ardua para el principiante, pero permite desarrollar programas muy eficientes, ya que otorga al programador el dominio absoluto del sistema. Los fabricantes suelen proporcionar el programa ensamblador de forma gratuita y en cualquier caso siempre se puede encontrar una versión gratuita para los microcontroladores más populares.

Compilador. La programación en un lenguaje de alto nivel (como el C) permite disminuir el tiempo de desarrollo de un producto. No obstante, si no se programa con cuidado, el código resultante puede ser mucho más ineficiente que el programado en ensamblador. Las versiones más potentes suelen ser muy caras, aunque para los microcontroladores más populares pueden encontrarse versiones demo limitadas e incluso compiladores gratuitos.

- Depuración: debido a que los microcontroladores van a controlar dispositivos físicos, los desarrolladores necesitan herramientas que les permitan comprobar el buen funcionamiento del microcontrolador cuando es conectado al resto de circuitos.

Simulador. Son capaces de ejecutar en un PC programas realizados para el microcontrolador. Los simuladores permiten tener un control absoluto sobre la ejecución de un programa, siendo ideales para la depuración de los mismos. Su gran inconveniente es que es difícil simular la entrada y salida de datos del microcontrolador. Tampoco cuentan con los posibles ruidos en las entradas, pero, al menos, permiten el

paso físico de la implementación de un modo más seguro y menos costoso, puesto que ahorraremos en grabaciones de chips para la prueba in-situ.

Placas de evaluación. Se trata de pequeños sistemas con un microcontrolador ya montado y que suelen conectarse a un PC desde el que se cargan los programas que se ejecutan en el microcontrolador. Las placas suelen incluir visualizadores LCD, teclados, LEDs, fácil acceso a los pines de E/S, etc. El sistema operativo de la placa recibe el nombre de programa monitor. El programa monitor de algunas placas de evaluación, aparte de permitir cargar programas y datos en la memoria del microcontrolador, puede permitir en cualquier momento realizar ejecución paso a paso, monitorizar el estado del microcontrolador o modificar los valores almacenados los registros o en la memoria.

Emuladores en circuito. Se trata de un instrumento que se coloca entre el PC anfitrión y el zócalo de la tarjeta de circuito impreso donde se alojará el microcontrolador definitivo. El programa es ejecutado desde el PC, pero para la tarjeta de aplicación es como si lo hiciese el mismo microcontrolador que luego irá en el zócalo. Presenta en pantalla toda la información tal y como luego sucederá cuando se coloque la cápsula.

2. LA FAMILIA DE LOS PIC COMO ELECCIÓN.

¿Qué es lo que ocurre con los PIC?, ¿Por qué están en boca de todos?. Hemos buscado en multitud de bibliografía y realmente nadie da una respuesta concreta, pero una aproximación a la realidad puede ser esta:

Los PIC tienen “ángel”, tienen “algo” que fascina a los diseñadores, puede ser la velocidad, el precio, la facilidad de uso, la información, las herramientas de apoyo... . Quizás un poco de todo eso es lo que produce esa imagen de sencillez y utilidad. Es probable que en un futuro próximo otra familia de microcontroladores le arrebatase ese “algo”.

Queremos constatar que para las aplicaciones más habituales (casi un 90%) la elección de una versión adecuada de PIC es la mejor solución; sin embargo, dado su carácter general, otras familias de microcontroladores son más eficaces en aplicaciones específicas, especialmente si en ellas predomina una característica concreta, que puede estar muy desarrollada en otra familia.

Los detalles más importantes que vuelven “locos” a los profesionales de la microelectrónica y microinformática y las razones de la excelente acogida que tienen los PIC son los siguientes:

- ® Sencillez de manejo: Tienen un juego de instrucciones reducido; 35 en la gama media.
- ® Buena información, fácil de conseguir y económica.
- ® Precio: Su coste es comparativamente inferior al de sus competidores.
- ® Poseen una elevada velocidad de funcionamiento. Buen promedio de parámetros: velocidad, consumo, tamaño, alimentación, código compacto, etc.
- ® Herramientas de desarrollo fáciles y baratas. Muchas herramientas software se pueden recoger libremente a través de Internet desde Microchip (<http://www.microchip.com>).

- ® Existe una gran variedad de herramientas hardware que permiten grabar, depurar, borrar y comprobar el comportamiento de los PIC.
- ® Diseño rápido.
- ® La gran variedad de modelos de PIC permite elegir el que mejor responde a los requerimientos de la aplicación.

Una de las razones del éxito de los PIC se basa en su utilización. Cuando se aprende a manejar uno de ellos, conociendo su arquitectura y su repertorio de instrucciones, es muy fácil emplear otro modelo.

2.1 Características relevantes.

Descripción de las características más representativas de los PIC:

2.1.1 Arquitectura.

La arquitectura del procesador sigue el modelo Harvard. En esta arquitectura, la CPU se conecta de forma independiente y con buses distintos con la memoria de instrucciones y con la de datos.

La arquitectura Harvard permite a la CPU acceder simultáneamente a las dos memorias. Además, propicia numerosas ventajas al funcionamiento del sistema como se irán describiendo.

2.1.2 Segmentación.

Se aplica la técnica de segmentación (“pipe-line”) en la ejecución de las instrucciones.

La segmentación permite al procesador realizar al mismo tiempo la ejecución de una instrucción y la búsqueda del código de la siguiente. De esta forma se puede ejecutar cada instrucción en un ciclo (un ciclo de instrucción equivale a cuatro ciclos de reloj).

Las instrucciones de salto ocupan dos ciclos al no conocer la dirección de la siguiente instrucción hasta que no se haya completado la de bifurcación.

2.1.3 Formato de las instrucciones.

El formato de todas las instrucciones es de la misma longitud

Todas las instrucciones de los microcontroladores de la gama baja tienen una longitud de 12 bits. Las de la gama media tienen 14 bits y más las de la gama alta. Esta característica es muy ventajosa en la optimización de la memoria de instrucciones y facilita enormemente la construcción de ensambladores y compiladores.

2.1.4 Juego de instrucciones.

Procesador RISC (Computador de Juego de Instrucciones Reducido).

Los modelos de la gama baja disponen de un repertorio de 33 instrucciones, 35 los de la gama media y casi 60 los de la alta.

2.1.5 Todas las instrucciones son ortogonales

Cualquier instrucción puede manejar cualquier elemento de la arquitectura como fuente o como destino.

2.1.6 Arquitectura basada en un “banco de registros”

Esto significa que todos los objetos del sistema (puertas de E/S, temporizadores, posiciones de memoria, etc.) están implementados físicamente como registros.

2.1.7 Diversidad de modelos de microcontroladores con prestaciones y recursos diferentes

La gran variedad de modelos de microcontroladores PIC permite que el usuario pueda seleccionar el más conveniente para su proyecto.

2.1.8 Herramientas de soporte potentes y económicas

La empresa Microchip y otras que utilizan los PIC ponen a disposición de los usuarios numerosas herramientas para desarrollar hardware y software. Son muy abundantes los programadores, los simuladores software, los emuladores en tiempo real, Ensambladores, Compiladores C, Intérpretes y Compiladores BASIC, etc.

La arquitectura Harvard y la técnica de segmentación son los principales recursos en los que se apoya el elevado rendimiento que caracteriza estos dispositivos programables, mejorando dos características esenciales:

1. Velocidad de ejecución.
2. Eficiencia en la compactación del código.

2.2 Las gamas de PIC

Una de las labores más importantes del ingeniero de diseño es la elección del microcontrolador que mejor satisfaga las necesidades del proyecto con el mínimo presupuesto.

Para resolver aplicaciones sencillas se precisan pocos recursos, en cambio, las aplicaciones grandes requieren numerosos y potentes. Siguiendo esta filosofía Microchip construye diversos modelos de microcontroladores orientados a cubrir, de forma óptima, las necesidades de cada proyecto. Así, hay disponibles microcontroladores sencillos y baratos para atender las aplicaciones simples y otros complejos y más costosos para las de mucha envergadura.

Microchip dispone de cuatro familias de microcontroladores de 8 bits para adaptarse a las necesidades de la mayoría de los clientes potenciales.

En la mayor parte de la bibliografía encontrareis tan solo tres familias de microcontroladores, con lo que habrán despreciado la llamada gama enana, que es en realidad una subfamilia formada por componentes pertenecientes a las otras gamas. En nuestro caso hemos preferido comentarla dado que los PIC enanos son muy apreciados

en las aplicaciones de control de personal, en sistemas de seguridad y en dispositivos de bajo consumo que gestionan receptores y transmisores de señales. Su pequeño tamaño los hace ideales en muchos proyectos donde esta cualidad es fundamental.

2.2.1 La gama enana: PIC12C(F)XXX de 8 patitas

Se trata de un grupo de PIC de reciente aparición que ha acaparado la atención del mercado. Su principal característica es su reducido tamaño, al disponer todos sus componentes de 8 patitas. Se alimentan con un voltaje de corriente continua comprendido entre 2,5 V y 5,5 V, y consumen menos de 2 mA cuando trabajan a 5 V y 4 MHz. El formato de sus instrucciones puede ser de 12 o de 14 bits y su repertorio es de 33 o 35 instrucciones, respectivamente. En la Figura 2.1 se muestra el diagrama de conexionado de uno de estos PIC.

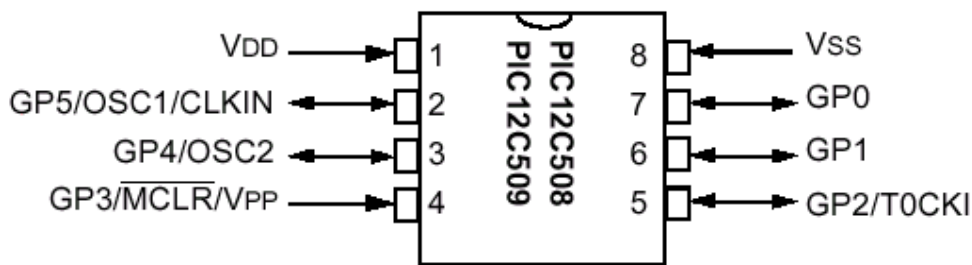


Figura 2.1. Diagrama de conexiones de los PIC12Cxxx de la gama enana.

Aunque los PIC enanos sólo tienen 8 patitas, pueden destinar hasta 6 como líneas de E/S para los periféricos porque disponen de un oscilador interno R-C.

En la Tabla 2.1 se presentan las principales características de los modelos de esta subfamilia, que el fabricante tiene la intención de potenciar en un futuro próximo. Los modelos 12C5xx pertenecen a la gama baja, siendo el tamaño de las instrucciones de 12 bits; mientras que los 12C6xx son de la gama media y sus instrucciones tienen 14 bits. Los modelos 12F6xx poseen memoria Flash para el programa y EEPROM para los datos.

MODELO	MEMORIA PROGRAMA	MEMORIA DATOS	FRECUENCIA MAXIMA	LINEAS E/S	ADC 8BITS	TEMPORIZADORE S	PATITAS
PIC12C508	512x12	25x8	4 MHz	6		TMR0 + WDT	8
PIC12C509	1024x12	41x8	4 MHz	6		TMR0 + WDT	8
PIC12C670	512x14	80x8	4 MHz	6		TMR0 + WDT	8
PIC12C671	1024x14	128x8	4 MHz	6	2	TMR0 + WDT	8
PIC12C672	2048x14	128x8	4 MHz	6	4	TMR0 + WDT	8
PIC12C680	512X12 FLASH	80x8 16x8 EEPROM	4 MHz	6	4	TMR0 + WDT	8
PIC12C681	1024x14 FLASH	80x8 16x8 EEPROM	4 MHz	6		TMR0 + WDT	8

Tabla 2.1. Características de los modelos PIC12C(F)XXX de la gama enana.

2.2.2 Gama baja o básica: PIC16C5X con instrucciones de 12 bits.

Se trata de una serie de PIC de recursos limitados, pero con una de la mejores relaciones coste/prestaciones. Sus versiones están encapsuladas con 18 y 28 patitas y pueden alimentarse a partir de una tensión de 2,5 V, lo que les hace ideales en las aplicaciones que funcionan con pilas teniendo en cuenta su bajo consumo (menos de 2 mA a 5 V y 4 MHz). Tienen un repertorio de 33 instrucciones cuyo formato consta de 12 bits. No admiten ningún tipo de interrupción y la Pila sólo dispone de dos niveles. En la Figura 2.2 se muestra el diagrama de conexionado de uno de estos PIC.

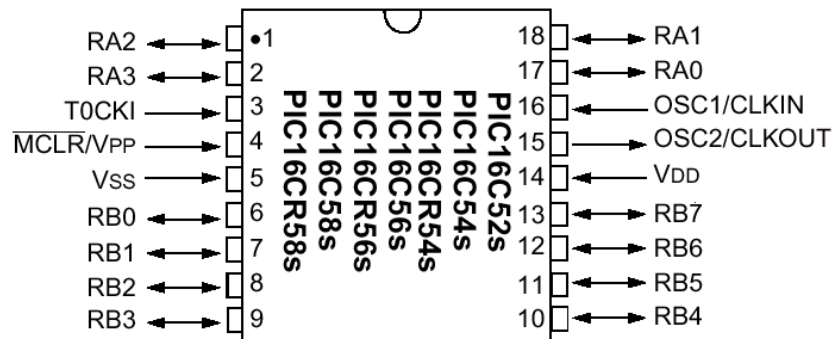


Figura 2.2: Diagrama de patitas de los PIC de la gama baja que responden a la nomenclatura PIC16C54/56.

Al igual que todos los miembros de la familia PIC16/17, los componentes de la gama baja se caracterizan por poseer los siguientes recursos: (en la Tabla 2.2 se presentan las principales características de los modelos de esta familia).

1. Sistema POR (“Power On Reset”)

Todos los PIC tienen la facultad de generar una autoreinicialización o autoreset al conectarles la alimentación.

2. Perro guardián (Watchdog o WDT)

Existe un temporizador que produce un reset automáticamente si no es recargado antes que pase un tiempo prefijado. Así se evita que el sistema quede “colgado” dado en esa situación el programa no recarga dicho temporizador y se genera un reset.

3. Código de protección

Cuando se procede a realizar la grabación del programa, puede protegerse para evitar su lectura. También disponen los PIC de posiciones reservadas para registrar números de serie, códigos de identificación, prueba, etc.

MODELO	MEMORIA PROGRAMA (x12 BITS)		MEMORIA DATOS (bytes)	FRECUENCIA MÁXIMA	LÍNEAS E/S	TEMPORIZADORES	PATITAS
	EPROM	ROM					
PIC16C52	384		25	4 MHz	4	TMR0 + WDT	18
PIC16C54	512		25	20 MHz	12	TMR0 + WDT	18
PIC16C54A	512		25	20 MHz	12	TMR0 + WDT	18
PIC16CR54A		512	25	20 MHz	12	TMR0 + WDT	18
PIC16C55	512		24	20 MHz	20	TMR0 + WDT	28
PIC16C56	1 K		25	20 MHz	12	TMR0 + WDT	18
PIC16C57	2 K		72	20 MHz	20	TMR0 + WDT	28
PIC16CR57B		2 K	72	20 MHz	20	TMR0 + WDT	28
PIC16C58A	2 K		73	20 MHz	12	TMR0 + WDT	18
PIC16CR58A		2 K	73	20 MHz	12	TMR0 + WDT	18

Tabla 2.2. Características de los modelos PIC16C(R)5X de la gama baja

4. Líneas de E/S de alta corriente

Las líneas de E/S de los PIC pueden proporcionar o absorber una corriente de salida comprendida entre 20 y 25 mA, capaz de excitar directamente ciertos periféricos.

5. Modo de reposo (Bajo consumo o “sleep”)

Ejecutando una instrucción (SLEEP), la CPU y el oscilador principal se detienen y se reduce notablemente el consumo.

Para terminar el comentario introductorio sobre los componentes de la gama baja conviene nombrar dos restricciones importantes:

- La pila o “stack” sólo dispone de dos niveles lo que supone no poder encadenar más de dos subrutinas.
- Los microcontroladores de la gama baja no admiten interrupciones.

2.2.3 Gama media. PIC16CXXX con instrucciones de 14 bits

Es la gama más variada y completa de los PIC. Abarca modelos con encapsulado desde 18 patitas hasta 68, cubriendo varias opciones que integran abundantes periféricos. Dentro de esta gama se halla el «fabuloso PIC16X84» y sus variantes. En la Figura 2.3 se muestra el diagrama de conexionado de uno de estos PIC.

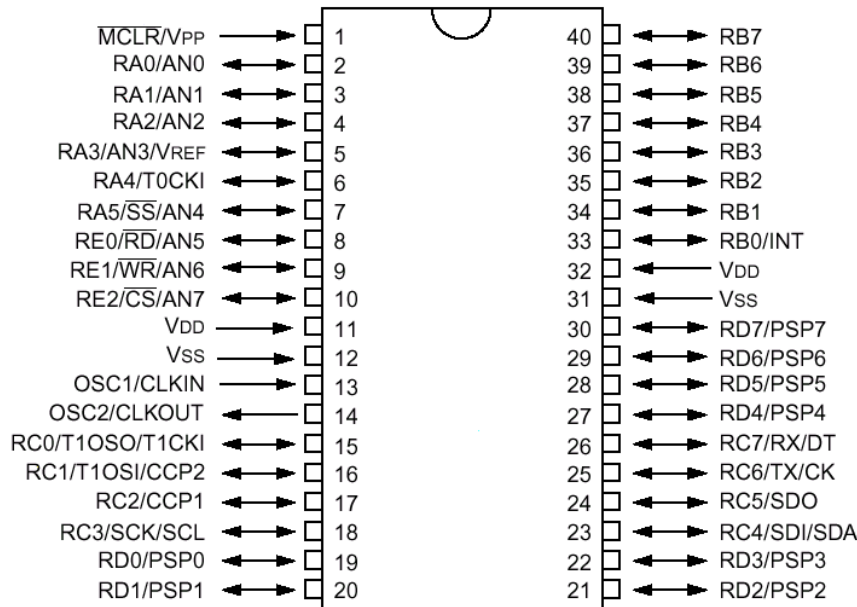


Figura 2.3. Diagrama de patitas del PIC16C74, uno de los modelos más representativos de la gama media.

En esta gama sus componentes añaden nuevas prestaciones a las que poseían los de la gama baja, haciéndoles más adecuados en las aplicaciones complejas. Admiten interrupciones, poseen comparadores de magnitudes analógicas, convertidores A/D, puertos serie y diversos temporizadores.

El repertorio de instrucciones es de 35, de 14 bits cada una y compatible con el de la gama baja. Sus distintos modelos contienen todos los recursos que se precisan en las aplicaciones de los microcontroladores de 8 bits. También dispone de interrupciones y una Pila de 8 niveles que permite el anidamiento de subrutinas. En la Tabla 2.3 se presentan las principales características de los modelos de esta familia.

MODELO	MEMORIA PROGRAMA	MEMORIA DATOS		REGISTROS ESPECÍFICOS	TEMPORIZADORES	INTERRUPCIONES	E/S	RANGO VOLTAGE	PATITAS
		RAM	EEPROM						
PIC16C84	1Kx14 EEPROM	36	64	11	TMR0 + WDT	4	13	2-6	18
PIC16PB4	1Kx14 FLASH	68	64	11	TMR0 + WDT	4	13	2-6	18
PIC16FB3	512x14 FLASH	36	64	11	TMR0 + WDT	4	13	2-6	18
PIC16CR4	1Kx14 ROM	68	64	11	TMR0 + WDT	4	13	2-6	18
PIC16CRB3	512x14 ROM	36	64	11	TMR0 + WDT	4	13	2-6	18

Tabla 2.3. Características relevantes de los modelos PIC16X8X de la gama media.

Encuadrado en la gama media también se halla la versión PIC14C000, que soporta el diseño de controladores inteligentes para cargadores de baterías, pilas pequeñas, fuentes de alimentación ininterrumpibles y cualquier sistema de adquisición y procesamiento de señales que requiera gestión de la energía de alimentación. Los PIC 14C000 admiten cualquier tecnología de las baterías como Li-Ion, NiMH, NiCd, Ph y Zinc.

El temporizador TMR1 que hay en esta gama tiene un circuito oscilador que puede trabajar asincrónamente y que puede incrementarse aunque el microcontrolador se halle en el modo de reposo (“sleep”), posibilitando la implementación de un reloj en tiempo real.

Las líneas de E/S presentan una carga “pull-up” activada por software.

2.2.4 Gama alta: PIC17CXXX con instrucciones de 16 bits.

Se alcanzan las 58 instrucciones de 16 bits en el repertorio y sus modelos disponen de un sistema de gestión de interrupciones vectorizadas muy potente. También incluyen variados controladores de periféricos, puertas de comunicación serie y paralelo

con elementos externos, un multiplicador hardware de gran velocidad y mayores capacidades de memoria, que alcanza los 8 k palabras en la memoria de instrucciones y 454 bytes en la memoria de datos.

Quizás la característica más destacable de los componentes de esta gama es su arquitectura abierta, que consiste en la posibilidad de ampliación del microcontrolador con elementos externos. Para este fin, las patitas sacan al exterior las líneas de los buses de datos, direcciones y control, a las que se conectan memorias o controladores de periféricos. Esta facultad obliga a estos componentes a tener un elevado numero de patitas comprendido entre 40 y 44. Esta filosofía de construcción del sistema es la que se empleaba en los microprocesadores y no suele ser una práctica habitual cuando se emplean microcontroladores. En la tabla 2.4 se muestran las características más relevantes de los modelos de esta gama, que sólo se utilizan en aplicaciones muy especiales con grandes requerimientos.

Tabla 2.4. Características más destacadas de los modelos PIC17CXXX de la gama alta.

Con vistas al siglo XXI, Microchip lanzará la gama “mejorada” PIC18CXXX, probablemente cuando este documento caiga en vuestras manos ya lo este.

MODELO	MEMORIA PROGRAMA	MEMORIA DATOS/RAM	REGISTROS ESPECÍFICOS	TEMPORIZADORES	CAF	PWM	CAD 10-BIT	INTERRUPCIONES	E/S	MULTIPLICADOR HARDWARE	PATITAS
PIC17C42A	20x16	232	40	4 + WDT	2	2		11	33	8x8	40/44
PIC17C43	48x16	454	40	4 + WDT	2	2		11	33	8x8	40/44
PIC17C44	8Kx16	454	48	4 + WDT	2	2		11	33	8x8	40/44
PIC17C52	8Kx16	454	76	4 + WDT	4	3	12	18	50	8x8	64/68
PIC17C55	16Kx16	902	76	4 + WDT	4	3	12	18	50	8x8	64/68

3. LOS REGISTROS DE LA GAMA MEDIA

3.1 Organización de la memoria de datos

La ampliación de recursos en los PIC forzó en los catalogados como de gama media una nueva estructura y la modificación de algunas instrucciones (partiendo, claro está, de la gama baja). Una de las diferencias fundamentales es, precisamente, la ampliación de memoria de registros, a los que se unieron algunos nuevos de sistema, y la accesibilidad a parte de los mismos que antes quedaban ocultos, como OPTION o TRIS, de los que hablaremos a continuación.

De este modo se optó por dos bancos de registros de 128 posiciones cada uno, la mayoría de los cuales son de propósito general. En el siguiente esquema, que muestra esta organización, las direcciones con casillas blancas muestran posiciones de registros específicos, y las grises generales.

Dirección	BANCO 0	BANCO 1	Dirección
00	INDF	INDF	80
01	TMR0	OPTION	81
02	PCL	PCL	82
03	STATUS	STATUS	83
04	FSR	FSR	84
05	PORT A	TRIS A	85
06	PORT B	TRIS B	86
07			87
08			88
09			89
0A	PCLATH	PCLATH	8A
0B	INTCON	INTCON	8B
0C	PIR1	PIE1	8C
0D			8D
0E		PCON	8E
0F			8F
10			90
11			91
12			92
13			93
14			94
15			95
16			96
17			97
18			98
19			99

Dirección	BANCO 0	BANCO 1	Dirección
1A			9A
1B			9B
1C			9C
1D			9D
1E			9E
1F	CMCON	VRCON	9F
.	Registros de Propósito General	Registros de Propósito General	.
.			.
.			.
7F			FF

3.2 Registros específicos

El PC. Direccionamiento del programa: El PC consta de 13 bits, con lo que es posible direccionar hasta 8K palabras, separadas en bancos de 2K. El byte de menos peso de la dirección se guarda en el registro PCL, sito en la posición 0x02 del banco 0, mientras los 5 bits de más peso se guardan en los 5 bits de menos peso del registro PCLATH (dirección 0x08). Puesto que las instrucciones CALL y GOTO sólo cuentan con 11 bits, sus saltos serán relativos a la página en la que estemos. El cambio real de página se hará cambiando los bits PCLATH.4 y PCLATH.3.

El STATUS. Registro de estado.

R/W	R/W	R/W	R	R	R/W	R/W	R/W
IRP	RP1	RP0	/TO	/PD	Z	DC	C

C: Acarreo en el 8º bit.
1 = acarreo en la suma y no en la resta. 0 = acarreo en la resta y no en la suma

DC: Acarreo en el 4º bit de menor peso.
Igual que C.

Z: Zero.
1 = El resultado de alguna operación es 0. 0 = El resultado es distinto de 0

/PD: Power Down.
1 = Recién encendido o tras CLRWDT. 0 = Tras ejecutar una instrucción SLEEP

/TO: Timer Out.
1 = Recién encendido, tras CLRWDT, o SLEEP. 0 = Saltó el WDT

RP1:RP0: Página de memoria de programa
Sólo en los PIC16C56/57

El OPTION. Registro de opciones

R/W	R/W	R/W	R	R	R/W	R/W	R/W
RBU	INTDEG	T0CS	T0SE	PSA	PS2	PS1	PS0

RBPU: Conexión de cargas Pull-Up para la puerta B.
1 = Cargas Pull-Up desconectadas

INTDEG: Tipo de flanco para la interrupción.
1 = RB0/INT sensible a flanco ascendente. 0 = RB0/INT sensible a flanco descendente.

T0CS: Fuente de reloj para el contador (registro TMR0).
1 = Pulsos por pata T0CLK (contador). 0 = Pulsos igual a reloj interno / 4 (temporizador).

T0SE: Tipo de flanco activo del T0CLK.
1 = Incremento TMR0 en flanco descendente. 0 = Incremento en flanco ascendente

PSA: Asignación del divisor de frecuencia.
1 = Divisor asignado al WDT. 0 = Divisor asignado al TMR0.

PSA2:PSA0: Valor del divisor de frecuencia.

PS2	PS1	PS0	División del TMR0
0	0	0	1 / 2
0	0	1	1 / 4
0	1	0	1 / 8
0	1	1	1 / 16
1	0	0	1 / 32
1	0	1	1 / 64
1	1	0	1 / 128
1	1	1	1 / 256

PS2	PS1	PS0	División del WDT
0	0	0	1 / 1
0	0	1	1 / 2
0	1	0	1 / 4
0	1	1	1 / 8
1	0	0	1 / 16
1	0	1	1 / 32
1	1	0	1 / 64
1	1	1	1 / 128

El INTCON. Registro de Interrupciones

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

GIE: Activación global de interrupciones..
1 = Interrupciones activadas. 0 = Interrupciones desactivadas.

PEIE. Activación de la interrupción de periféricos (comparador)
1 = Activada. 0 = Desactivada.

T0IE: Activación de la interrupción del TMR0.
1 = Activada. 0 = Desactivada.

INTE: Activación de la interrupción externa.
1 = Activada. 0 = Desactivada.

RBIE: Activación de la interrupción de la puerta B.
1 = Activada. 0 = Desactivada.

T0IF: Indicador de TMR0 se ha desbordado.
1 = TMR0 desbordado. Borrar por software. 0 = No se ha desbordado.

INTF: Software de estado de la interrupción externa

El PCON. Registro identificador del reset

						R/W /POR	R/W /BO
/POR: Señalizador de Power On Reset (reset por activación del micro). 1 = No hubo Power On Reset. 0 = Ha habido Power On reset. /BO: Señalizador de Brown-Out (Caída de tensión). 1 = No hubo Brown Out. 0 = Hubo Brown-Out							

Los registros PIE1 y PIR1 están relacionados con el comparador, así como CMCON y VRCON, y no serán explicados ya que el PIC16C84, en el que se centra este estudio, carece de él. El registro FSR es utilizado en la gama baja, por lo que tampoco nos ocuparemos de él.

Los registros TMR0, PORTA, PORTB, TRISA Y TRISB, serán, por comodidad, estudiados en el apartado de programación, así como el WDT.

4. REPERTORIO DE INSTRUCCIONES

4.1 *Características generales*

Habiendo escogido los diseñadores de PIC la filosofía RISC, su juego de instrucciones es reducido (33 instrucciones en la gama baja), siendo éstas, además, sencillas y rápidas, puesto que casi todas se ejecutan en un único ciclo de máquina (equivalente a 4 del reloj principal). Sus operandos son de gran flexibilidad, pudiendo actuar cualquier objeto como fuente y como destino.

Posee tres tipos bien diferenciados de direccionamiento, estos son:

1º Inmediato: El valor del dato está incluido en el propio código OP, junto a la instrucción.

2º Directo: La dirección del dato está incluido en el propio código OP, junto a la instrucción.

3º Indirecto: La dirección de la memoria de datos que guarda el operando está contenida en un registro.

Nosotros nos centraremos en la gama media, que tiene 35 instrucciones. La mayoría son idénticas a las de la gama baja, si bien las diferencias serán convenientemente explicadas.

4.2 *Definiciones y abreviaturas*

Ante todo es conveniente que usted tenga clara la estructura interna del micro, puesto que las instrucciones la referencian, y puesto que en cualquier micro la comprensión de la nomenclatura de sus componentes es esencial. De este modo hemos creado la siguiente tabla para ayudarle a comprender las abreviaturas:

Abreviatura	Descripción
PC	Contador de Programa que direcciona la memoria de instrucciones. Tiene un tamaño de 11 bits en la gama baja, de los cuales los 8 de menos peso configuran el registro PCL que ocupa el registro 0x02 del área de datos.
TOS	Cima de la pila, con 2 niveles en la gama baja y 8 en la media
WDT	Perro guardián (Watchdog)
W	Registro W, similar al acumulador
F	Suele ser un campo de 5 bits (ffff) que contiene la dirección del banco de registros, que ocupa el banco 0 del área de datos. Direcciona uno de esos registros.
D	Bit del código OP de la instrucción, que selecciona el destino. Si d=0, el destino es W, y si d=1 el destino es f.
Dest	Destino (registro W o f)
TO	Bit “Time Out” del registro de estado
PD	Bit “Power Down” del registro de estado
b	Suele ser un campo de 3 bits (bbb) que determinan la posición de un bit dentro de un registro de 8 bits
k	Se trata, normalmente, de un campo de 8 bits (kkkkkkkk) que representa un dato inmediato. También puede constar de 9 bits en las instrucciones de salto que cargan al PC
x	Valor indeterminado (puede ser un 0 o un 1). Para mantener la compatibilidad con las herramientas software de Microchip conviene hacer x = 0
label	Nombre de la etiqueta
[]	Opciones
()	Contenido
@	Se asigna a
<>	Campo de bits de un registro
Î	Pertenece al conjunto
Z	Señalizador de cero en W. Pertenece al registro de estado
C	Señalizador de acarreo en el octavo bit del W. Pertenece al registro de estado
DC	Señaliza el acarreo en el 4 bit del W. Pertenece al registro de estado
Itálicas	Términos definidos por el usuario

4.3 Repertorio de instrucciones de la gama media

<p>ADDLW Suma un literal</p> <p>Sintaxis: [label] ADDLW k Operandos: $0 \leq k \leq 255$ Operación: $(W) + (k) \Rightarrow (W)$ Flags afectados: C, DC, Z Código OP: 11 111x kkkk kkkk</p> <p>Descripción: Suma el contenido del registro W y k, guardando el resultado en W.</p> <p>Ejemplo: ADDLW 0xC2</p> <p>Antes: W = 0x17 Después: W = 0xD9</p>	<p>ADDWF W + F</p> <p>Sintaxis: [label] ADDWF f,d Operandos: $d \in [0,1], 0 \leq f \leq 127$ Operación: $(W) + (f) \Rightarrow (dest)$ Flags afectados: C, DC, Z Código OP: 00 0111 dfff ffff</p> <p>Descripción: Suma el contenido del registro W y el registro f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p> <p>Ejemplo: ADDWF REG,0</p> <p>Antes: W = 0x17., REG = 0xC2 Después: W = 0xD9, REG = 0xC2</p>	<p>ANDLW W AND literal</p> <p>Sintaxis: [label] ANDLW k Operandos: $0 \leq k \leq 255$ Operación: $(W) \text{ AND } (k) \Rightarrow (W)$ Flags afectados: Z Código OP: 11 1001 kkkk kkkk</p> <p>Descripción: Realiza la operación lógica AND entre el contenido del registro W y k, guardando el resultado en W.</p> <p>Ejemplo: ANDLW 0xC2</p> <p>Antes: W = 0x17 Después: W = 0xD9</p>
<p>ANDWF W AND F</p> <p>Sintaxis: [label] ANDWF f,d Operandos: $d \in [0,1], 0 \leq f \leq 127$ Operación: $(W) \text{ AND } (f) \Rightarrow (dest)$ Flags afectados: Z Código OP: 00 0101 dfff ffff</p> <p>Descripción: Realiza la operación lógica AND entre los registros W y f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p> <p>Ejemplo: ANDWF REG,0</p> <p>Antes: W = 0x17., REG = 0xC2 Después: W = 0x17, REG = 0x02</p>	<p>BCF Borra un bit</p> <p>Sintaxis: [label] BCF f,b Operandos: $0 \leq f \leq 127, 0 \leq b \leq 7$ Operación: $0 \Rightarrow (f < b) >$ Flags afectados: Ninguno Código OP: 01 00bb bfff ffff</p> <p>Descripción: Borra el bit b del registro f</p> <p>Ejemplo: BCF REG,7</p> <p>Antes: REG = 0xC7 Después: REG = 0x47</p>	<p>BSF Activa un bit</p> <p>Sintaxis: [label] BSF f,b Operandos: $0 \leq f \leq 127, 0 \leq b \leq 7$ Operación: $1 \Rightarrow (f < b) >$ Flags afectados: Ninguno Código OP: 01 01bb bfff ffff</p> <p>Descripción: Activa el bit b del registro f</p> <p>Ejemplo: BSF REG,7</p> <p>Antes: REG = 0x0A Después: REG = 0x8A</p>
<p>BTFSC Test de bit y salto</p> <p>Sintaxis: [label] BTFSC f,d Operandos: $d \in [0,1], 0 \leq f \leq 127$ Operación: Salto si $(f < b) > = 0$ Flags afectados: Ninguno Código OP: 01 10bb bfff ffff</p> <p>Descripción: Si el bit b del registro f es 0, se salta una instrucción y se continúa con la ejecución. En caso de salto, ocupará dos ciclos de reloj.</p> <p>Ejemplo: BTFSC REG,6 GOTO NO_ES_0 SI_ES_0 Instrucción NO_ES_0 Instrucción</p>	<p>BTFSS Test de bit y salto</p> <p>Sintaxis: [label] BTFSS f,d Operandos: $d \in [0,1], 0 \leq f \leq 127$ Operación: Salto si $(f < b) > = 1$ Flags afectados: Ninguno Código OP: 01 11bb bfff ffff</p> <p>Descripción: Si el bit b del registro f es 1, se salta una instrucción y se continúa con la ejecución. En caso de salto, ocupará dos ciclos de reloj.</p> <p>Ejemplo: BTFSS REG,6 GOTO NO_ES_0 SI_ES_0 Instrucción NO_ES_0 Instrucción</p>	<p>CALL Salto a subrutina</p> <p>Sintaxis: [label] CALL k Operandos: $0 \leq k \leq 2047$ Operación: $PC \Rightarrow Pila; k \Rightarrow PC$ Flags afectados: Ninguno Código OP: 10 0kkk kkkk kkkk</p> <p>Descripción: Salto a una subrutina. La parte baja de k se carga en PCL, y la alta en PCLATCH. Ocupa 2 ciclos de reloj.</p> <p>Ejemplo: ORIGEN CALL DESTINO</p> <p>Antes: PC = ORIGEN Después: PC = DESTINO</p>

<p>CLRF Borra un registro</p> <p>Sintaxis: [label] CLRF f Operandos: $0 \leq f \leq 127$ Operación: : $0x00 \Rightarrow (f), 1 \Rightarrow Z$ Flags afectados: Z Código OP: 00 0001 1fff ffff</p> <p>Descripción: El registro f se carga con 0x00. El flag Z se activa.</p> <p>Ejemplo: : CLRF REG</p> <p>Antes: REG = 0x5A Después: REG = 0x00, Z = 1</p>	<p>CLRW Borra el registro W</p> <p>Sintaxis: [label] CLRW Operandos: Ninguno Operación: : $0x00 \Rightarrow W, 1 \Rightarrow Z$ Flags afectados: Z Código OP: 00 0001 0xxx xxxx</p> <p>Descripción: El registro de trabajo W se carga con 0x00. El flag Z se activa.</p> <p>Ejemplo: : CLRW</p> <p>Antes: W = 0x5A Después: W = 0x00, Z = 1</p>	<p>CLRWDW Borra el WDT</p> <p>Sintaxis: [label] CLRWDW Operandos: Ninguno Operación: : $0x00 \Rightarrow WDT, 1 \Rightarrow /TO$ $1 \Rightarrow /PD$ Flags afectados: /TO, /PD Código OP: 00 0000 0110 0100 Descripción: Esta instrucción borra tanto el WDT como su preescaler. Los bits /TO y /PD del registro de estado se ponen a 1.</p> <p>Ejemplo: : CLRWDW Después: Contador WDT = 0, Preescalas WDT = 0, /TO = 1, /PD = 1</p>
<p>COMF Complemento de f</p> <p>Sintaxis: [label] COMF f,d Operandos: $d \in [0,1], 0 \leq f \leq 127$ Operación: : $(/f), 1 \Rightarrow (dest)$ Flags afectados: Z Código OP: 00 1001 dfff ffff</p> <p>Descripción: El registro f es complementado. El flag Z se activa si el resultado es 0. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f..</p> <p>Ejemplo: : COMF REG,0</p> <p>Antes: REG = 0x13 Después: REG = 0x13, W = 0XEC</p>	<p>DECF Decremento de f</p> <p>Sintaxis: [label] DECF f,d Operandos: $d \in [0,1], 0 \leq f \leq 127$ Operación: : $(f) - 1 \Rightarrow (dest)$ Flags afectados: Z Código OP: 00 0011 dfff ffff</p> <p>Descripción: Decrementa en 1 el contenido de f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p> <p>Ejemplo: : DECF CONT,1</p> <p>Antes: CONT = 0x01, Z = 0 Después: CONT = 0x00, Z = 1</p>	<p>DECFSZ Decremento y salto</p> <p>Sintaxis: [label] DECFSZ f,d Operandos: $d \in [0,1], 0 \leq f \leq 127$ Operación: : $(f) - 1 \Rightarrow d$; Salto si R=0 Flags afectados: Ninguno Código OP: 00 1011 dfff ffff</p> <p>Descripción: Decrementa el contenido del registro f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Si la resta es 0 salta la siguiente instrucción, en cuyo caso costaría 2 ciclos.</p> <p>Ejemplo: : DECFSZ REG,0 GOTO NO_ES_0 SI_ES_0 Instrucción NO_ES_0 Salta instrucción anterior</p>
<p>GOTO Salto incondicional</p> <p>Sintaxis: [label] GOTO k Operandos: $0 \leq k \leq 2047$ Operación: : $k \Rightarrow PC \langle 8:0 \rangle$ Flags afectados: Ninguno Código OP: 10 1kkk kkkk kkkk</p> <p>Descripción: Se trata de un salto incondicional. La parte baja de k se carga en PCL, y la alta en PCLATCH. Ocupa 2 ciclos de reloj.</p> <p>Ejemplo: : ORIGEN GOTO DESTINO</p> <p>Antes: PC = ORIGEN Después: PC = DESTINO</p>	<p>INCF Decremento de f</p> <p>Sintaxis: [label] INCF f,d Operandos: $d \in [0,1], 0 \leq f \leq 127$ Operación: : $(f) + 1 \Rightarrow (dest)$ Flags afectados: Z Código OP: 00 1010 dfff ffff</p> <p>Descripción: Incrementa en 1 el contenido de f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p> <p>Ejemplo: : INCF CONT,1</p> <p>Antes: CONT = 0xFF, Z = 0 Después: CONT = 0x00, Z = 1</p>	<p>INCFSZ Incremento y salto</p> <p>Sintaxis: [label] INCFSZ f,d Operandos: $d \in [0,1], 0 \leq f \leq 127$ Operación: : $(f) + 1 \Rightarrow d$; Salto si R=0 Flags afectados: Ninguno Código OP: 00 1111 dfff ffff</p> <p>Descripción: Incrementa el contenido del registro f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Si la resta es 0 salta la siguiente instrucción, en cuyo caso costaría 2 ciclos.</p> <p>Ejemplo: : INCFSZ REG,0 GOTO NO_ES_0 SI_ES_0 Instrucción NO_ES_0 Salta instrucción anterior</p>

<p>IORLW W OR literal</p> <p>Sintaxis: [label] IORLW k Operandos: $0 \leq k \leq 255$ Operación: (W) OR (k) \Rightarrow (W) Flags afectados: Z Código OP: 11 1000 kkkk kkkk</p> <p>Descripción: Se realiza la operación lógica OR entre el contenido del registro W y k, guardando el resultado en W.</p> <p>Ejemplo: IORLW 0x35</p> <p>Antes: W = 0x9A Después: W = 0xBF</p>	<p>IORWF W AND F</p> <p>Sintaxis: [label] IORWF f,d Operandos: $d \in [0,1], 0 \leq f \leq 127$ Operación: (W) OR (f) \Rightarrow (dest) Flags afectados: Z Código OP: 00 0100 dfff ffff</p> <p>Descripción: Realiza la operación lógica OR entre los registros W y f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p> <p>Ejemplo: IORWF REG,0</p> <p>Antes: W = 0x91, REG = 0x13 Después: W = 0x93, REG = 0x13</p>	<p>MOVLW Cargar literal en W</p> <p>Sintaxis: [label] MOVLW f Operandos: $0 \leq f \leq 255$ Operación: (k) \Rightarrow (W) Flags afectados: Ninguno Código OP: 11 00xx kkkk kkkk</p> <p>Descripción: El literal k pasa al registro W.</p> <p>Ejemplo: MOVLW 0x5A</p> <p>Después: REG = 0x4F, W = 0x5A</p>
<p>MOVF Mover a f</p> <p>Sintaxis: [label] MOVF f,d Operandos: $d \in [0,1], 0 \leq f \leq 127$ Operación: (f) \Rightarrow (dest) Flags afectados: Z Código OP: 00 1000 dfff ffff</p> <p>Descripción: El contenido del registro f se mueve al destino d. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Permite verificar el registro, puesto que afecta a Z.</p> <p>Ejemplo: MOVF REG,0</p> <p>Después: W = REG</p>	<p>MOVWF Mover a f</p> <p>Sintaxis: [label] MOVWF f Operandos: $0 \leq f \leq 127$ Operación: W \Rightarrow (f) Flags afectados: Ninguno Código OP: 00 0000 1fff ffff</p> <p>Descripción: El contenido del registro W pasa al registro f.</p> <p>Ejemplo: MOVWF REG,0</p> <p>Antes: REG = 0xFF, W = 0x4F Después: REG = 0x4F, W = 0x4F</p>	<p>NOP No operar</p> <p>Sintaxis: [label] NOP Operandos: Ninguno Operación: No operar Flags afectados: Ninguno Código OP: 00 0000 0xx0 0000</p> <p>Descripción: No realiza operación alguna. En realidad consume un ciclo de instrucción sin hacer nada.</p> <p>Ejemplo: CLRWDT</p> <p>Después: Contador WDT = 0, Preescalares WDT = 0, /TO = 1, /PD = 1</p>
<p>RETFIE Retorno de interrup.</p> <p>Sintaxis: [label] RETFIE Operandos: Ninguno Operación: : 1 \Rightarrow GIE; TOS \Rightarrow PC Flags afectados: Ninguno Código OP: 00 0000 0000 1001</p> <p>Descripción: El PC se carga con el contenido de la cima de la pila (TOS): dirección de retorno. Consume 2 ciclos. Las interrupciones vuelven a ser habilitadas.</p> <p>Ejemplo: RETFIE</p> <p>Después: PC = dirección de retorno GIE = 1</p>	<p>RETLW Retorno, carga W</p> <p>Sintaxis: [label] RETLW k Operandos: $0 \leq k \leq 255$ Operación: : (k) \Rightarrow (W); TOS \Rightarrow PC Flags afectados: Ninguno Código OP: 11 01xx kkkk kkkk</p> <p>Descripción: El registro W se carga con la constante k. El PC se carga con el contenido de la cima de la pila (TOS): dirección de retorno. Consume 2 ciclos.</p> <p>Ejemplo: RETLW 0x37</p> <p>Después: PC = dirección de retorno W = 0x37</p>	<p>RETURN Retorno de rutina</p> <p>Sintaxis: [label] RETURN Operandos: Ninguno Operación: : TOS \Rightarrow PC Flags afectados: Ninguno Código OP: 00 0000 0000 1000</p> <p>Descripción: El PC se carga con el contenido de la cima de la pila (TOS): dirección de retorno. Consume 2 ciclos.</p> <p>Ejemplo: RETURN</p> <p>Después: PC = dirección de retorno</p>

<p>RLF Rota f a la izquierda</p> <p>Sintaxis: [label] RLF f,d Operandos: $d \in [0,1]$, $0 \leq f \leq 127$ Operación: Rotación a la izquierda Flags afectados: C Código OP: 00 1101 dfff ffff</p> <p>Descripción: El contenido de f se rota a la izquierda. El bit de menos peso de f pasa al carry (C), y el carry se coloca en el de mayor peso. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p> <p>Ejemplo: RRF REG,0</p> <p>Antes: REG = 1110 0110, C = 0 Después: REG = 1110 0110, W = 1100 1100, C = 1</p>	<p>RRF Rota f a la derecha</p> <p>Sintaxis: [label] RRF f,d Operandos: $d \in [0,1]$, $0 \leq f \leq 127$ Operación: Rotación a la derecha Flags afectados: C Código OP: 00 1100 dfff ffff</p> <p>Descripción: El contenido de f se rota a la derecha. El bit de menos peso de f pasa al carry (C), y el carry se coloca en el de mayor peso. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p> <p>Ejemplo: RRF REG,0</p> <p>Antes: REG = 1110 0110, C = 1 Después: REG = 1110 0110, W = 01110 0011, C = 0</p>	<p>SLEEP Modo bajo consumo</p> <p>Sintaxis: [label] SLEEP Operandos: Ninguno Operación: $0x00 \Rightarrow \text{WDT}$, $1 \Rightarrow / \text{TO}$ $0 \Rightarrow \text{WDT Preescaler}$, $0 \Rightarrow / \text{PD}$ Flags afectados: /PD, /TO Código OP: 00 0000 0110 0011</p> <p>Descripción: El bit de energía se pone a 0, y a 1 el de descanso. El WDT y su preescaler se borran. El micro para el oscilador, llenando al modo “durmiente”.</p> <p>Ejemplo: SLEEP</p> <p>Preescalas WDT = 0, /TO = 1, /PD = 1</p>
<p>SUBLW Resta Literal - W</p> <p>Sintaxis: [label] SUBLW k Operandos: $0 \leq k \leq 255$ Operación: $(k) - (W) \Rightarrow (W)$ Flags afectados: Z, C, DC Código OP: 11 110x kkkk kkkk Descripción: Mediante el método del complemento a dos el contenido de W es restado al literal. El resultado se almacena en W.</p> <p>Ejemplos: SUBLW 0x02</p> <p>Antes: W=1, C=?, Después: W=1, C=1 Antes: W=2, C=?, Después: W=0, C=1 Antes: W=3, C=?, Después: W=FF, C=0 (El resultado es negativo)</p>	<p>SUBWF Resta f – W</p> <p>Sintaxis: [label] SUBWF f,d Operandos: $d \in [0,1]$, $0 \leq f \leq 127$ Operación: $(f) - (W) \Rightarrow (\text{dest})$ Flags afectados: C, DC, Z Código OP: 00 0010 dfff ffff Descripción: Mediante el método del complemento a dos el contenido de W es restado al de f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p> <p>Ejemplos: SUBWF REG,1</p> <p>Antes: REG = 0x03, W = 0x02, C = ? Después: REG=0x01, W = 0x4F, C=1 Antes: REG = 0x02, W = 0x02, C = ? Después: REG=0x00, W = 0x02, C= 1 Antes: REG= 0x01, W= 0x02, C= ? Después: REG=0xFF, W=0x02, C= 0 (Resultado negativo)</p>	<p>SWAPF Intercambio de f</p> <p>Sintaxis: [label] SWAPF f,d Operandos: $d \in [0,1]$, $0 \leq f \leq 127$ Operación: $(f <3:0>) \Leftrightarrow (f <7:4>)$ Flags afectados: Ninguno Código OP: 00 1110 dfff ffff</p> <p>Descripción: Los 4 bits de más peso y los 4 de menos son intercambiados. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p> <p>Ejemplo: SWAPF REG,0</p> <p>Antes: REG = 0xA5 Después: REG = 0xA5, W = 0x5A</p>
<p>XORLW W OR literal</p> <p>Sintaxis: [label] XORLW k Operandos: $0 \leq k \leq 255$ Operación: $(W) \text{ XOR } (k) \Rightarrow (W)$ Flags afectados: Z Código OP: 11 1010 kkkk kkkk</p> <p>Descripción: Se realiza la operación lógica XOR entre el contenido del registro W y k, guardando el resultado en W.</p> <p>Ejemplo: XORLW 0xAF</p> <p>Antes: W = 0xB5 Después: W = 0x1A</p>	<p>XORWF W AND F</p> <p>Sintaxis: [label] XORWF f,d Operandos: $d \in [0,1]$, $0 \leq f \leq 127$ Operación: $(W) \text{ XOR } (f) \Rightarrow (\text{dest})$ Flags afectados: Z Código OP: 00 0110 dfff ffff</p> <p>Descripción: Realiza la operación lógica XOR entre los registros W y f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.</p> <p>Ejemplo: XORWF REG,0</p> <p>Antes: W = 0xB5, REG = 0xAF Después: W = 0xB5, REG = 0x1A</p>	<p><i>La gama media tiene un total de 35 instrucciones, cada una de las cuales ocupan 14 bits.</i></p>

4.4 Instrucciones de la gama baja

La gama baja carece de 4 de las instrucciones de la gama media. Estas son ADDLW, RETFIE, RETURN y SUBLW. En cambio, y puesto que los registros OPTION y TRIS no son accesibles, se añaden las dos siguientes:

OPTION Carga del reg. option Sintaxis: [label] OPTION Operandos: Ninguno Operación: (W) \Rightarrow OPTION Flags afectados: Ninguno Código OP: 0000 0000 0010 Descripción: El contenido del registro W se carga en el registro OPTION, registro de sólo lectura en el que se configura el funcionamiento del preescaler y el TMR0. Ejemplo: : OPTION Antes: W= 0x06, OPTION = 0x37 Después: W= 0x06, OPTION = 0x06	TRIS Carga del registro TRIS Sintaxis: [label] TRIS f Operandos: $5 \leq f \leq 7$ Operación: (W) \Rightarrow Registro TRIF<f> Flags afectados: Ninguno Código OP: 0000 0000 0fff Descripción: El contenido del registro W se carga en el registro TRISA, TRISB o TRISC, según el valor de f. Estos registros de sólo lectura configuran las patillas de un puerto como de entrada o salida. Ejemplo: : TRIS PORTA Antes: W=0xA5, TRISA=0x56 Después: W=0xA5, TRISA=0xA5
---	---

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

Otras diferencias fundamentales entre ambas gamas son, por ejemplo, que en la baja el vector de reset (la primera línea que se ejecuta tras un reset) es la última de la memoria correspondiente, mientras que en la media es la 0x00. El vector de interrupción (situado en la 0x04) no existe en la gama baja.

En la gama baja se necesita emplear los bits PA2-PA0 como parte alta de la dirección de programa en las instrucciones CALL y GOTO.

Las páginas de la gama baja son más pequeñas, y se deben revisar todas las escrituras de los registros OPTION, ESTADO y FSR en la conversión del código.

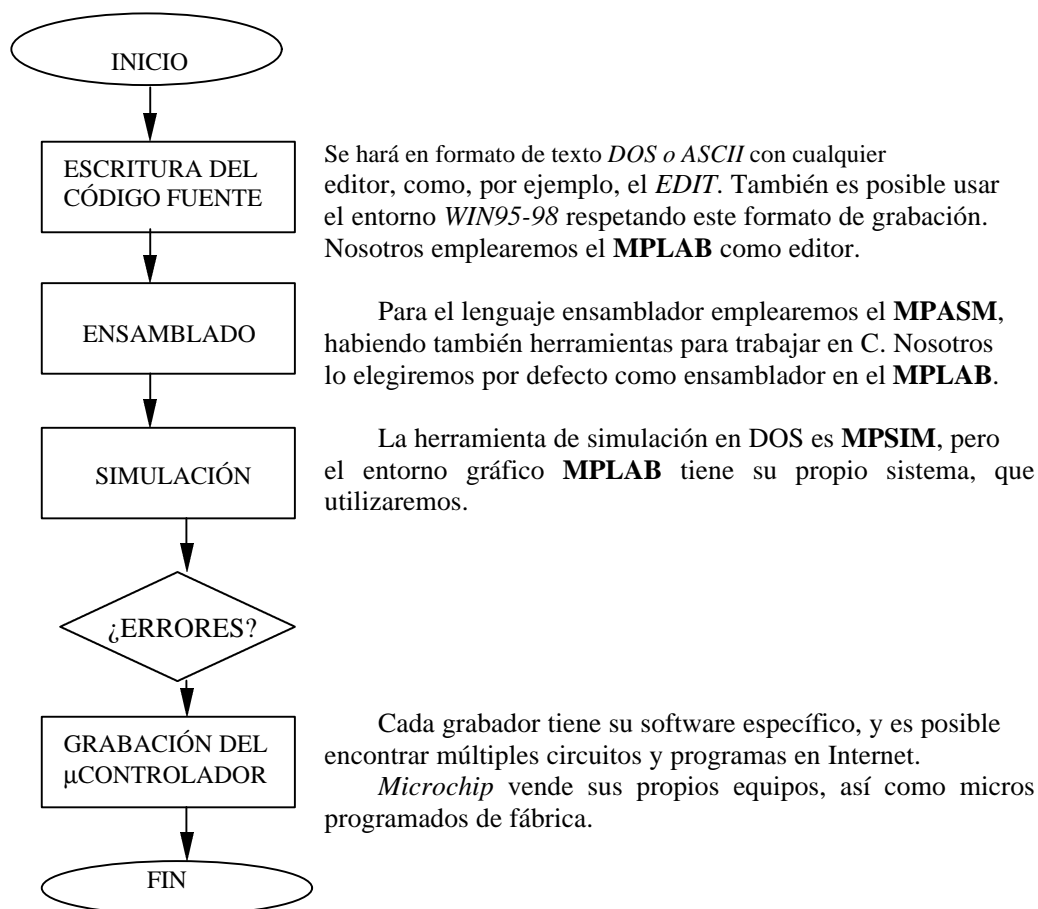
Por tanto, fácilmente comprobamos que el código entre ambas gamas no es 100% compatible.

5. PROGRAMACIÓN DE LOS μ CONTROLADORES PIC

5.1 Introducción

En busca de la sencillez que debe acompañar a este documento, ya que se ha pensado elaborarlo a modo de prácticas, nos centraremos directamente en la programación de los μ controladores que nos ocupan, los PIC, es decir, no explicaremos en sí las estrategias asociadas al diseño, ya que han sido impartidas en otras asignaturas o pueden ser fácilmente halladas en otros textos, sino que, paso a paso, iremos viendo la confección de distintas rutinas, cada vez más complejas, hasta familiarizarnos con todas sus instrucciones y componentes.

No obstante acompañamos nuestro trabajo con un organigrama con las distintas fases de implementación, en las que, de hecho, suponemos, tras una fase de estudio del problema, elegido ya el mejor μ controlador, así como decidido el sistema de conexión de patillas de E/S correcto.

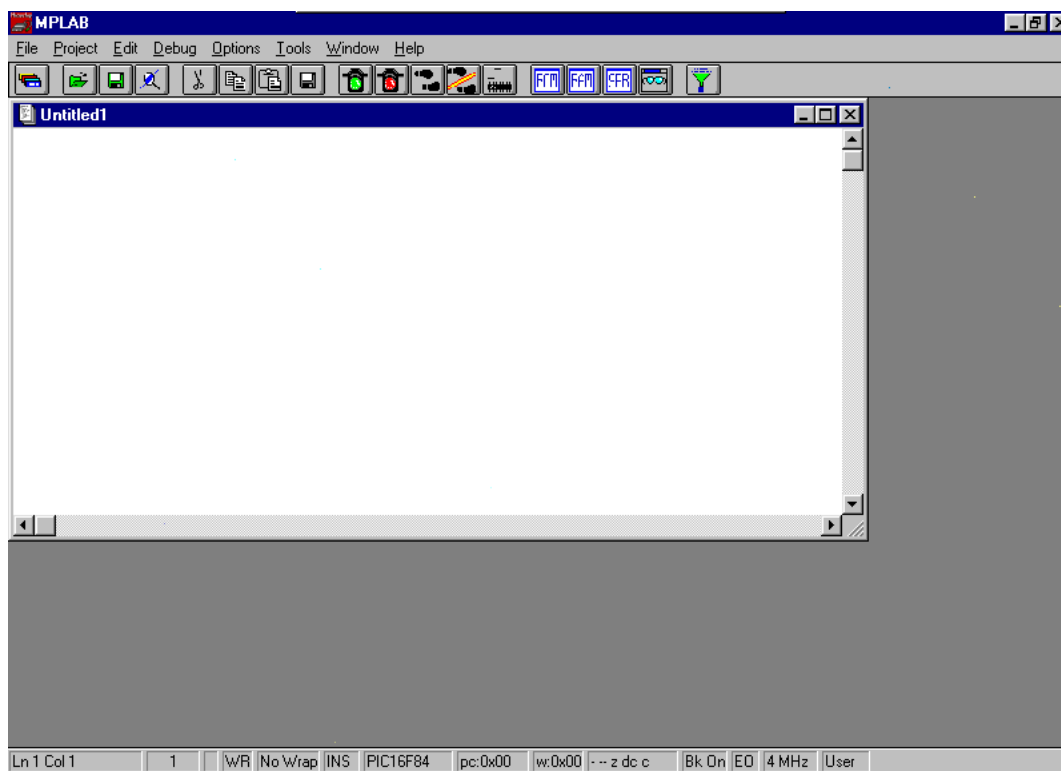


Las herramientas **MPLAB**, **MPASM** y **MPSIM** se pueden encontrar y bajar gratuitamente de internet en la dirección **www.microchip.com**. Nosotros **facilitamos en el CD adjunto** las actualizaciones de *abril del 2000*.

2.- PRIMEROS PASOS CON EL MPLAB

Antes de comenzar a escribir programas es necesario conocer las herramientas disponibles para desarrollarlos. De entre ellas el entorno para nosotros más interesante es el MPLAB, no por ser el más eficiente, ya que de hecho probablemente no lo es, sino por ser el más accesible: se puede bajar gratis a través de internet o pedirse, también gratis, a Sagitrón, su distribuidor en España. Además es gráfico, funcionando perfectamente bajo Windows.

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



Para ayudarnos emplearemos el sencillo programa **suma.asm**, incluido en el CD adjunto, el cual, simplemente, realiza una suma de los registros PCL (0x02) y la entrada del Puerto A (0x05).



El modo de abrir una ventana con **suma.asm** es el habitual, a través del menú **FILE**, seguido de **OPEN**, o con el icono típico, que encabeza este párrafo (en amarillo, no en verde).

5.2 El entorno de trabajo MPLAB

5.2.1 El ensamblador

El ensamblador que utiliza por defecto el MPLAB es el **MPASM**, que conserva de sus tiempos bajo MS-DOS. Debido a que **suma.asm** ha sido escrito con determinadas directivas muy comunes en cualquier implementación para MPASM conviene comenzar por ellas antes de embarcarse auténticamente en la descripción de menús y submenús del MPLAB.

Ya la primera línea, **LIST P = 16C84**, es importante, puesto que con ella definimos el procesador a utilizar durante todos los procesos (ensamblado, **emulación**, **grabación**). Los dos resaltados pueden llegar a ser especialmente críticos, en caso de recurrir a un procesador erróneo, haciéndonos perder jornadas de trabajo o chips por despiste. Es por ello, junto al hecho de facilitar la identificación del programa sólo con un primer vistazo, que preferimos esta forma a la de seleccionarlo por menú, también posible y detallada más abajo en este mismo texto.

La directiva **ORG**, seguida de una posición de memoria, indica al ensamblador dónde debe situar en la misma el siguiente fragmento de código, siendo también recomendable incluirla en todo programa, como mínimo, antes de la primera instrucción. Los casos de direcciones especiales serán también descritos más adelante en el presente texto.

La directiva **END** es imprescindible e indica al ensamblador el final del programa.

El ; es empleado a modo de comando **REM**, es decir, se sobreentiende que lo que le sigue es un comentario.

El ensamblador exige una cierta tabulación mínima de sus distintos elementos. De este modo la definición de variables podrá escribirse en la 1ª columna de cualquier línea, mientras que las directivas e instrucciones deberán ir en la 2ª columna, como mínimo. Las tabulaciones características son las empleadas por nosotros, ya que, aunque no son imprescindibles, clarifican la lectura del programa.

Las cifras se expresan de acuerdo con la presente tabla:

BASE	REPRESENTACIÓN
<i>DECIMAL</i>	d'12'
<i>HEXAGESIMAL</i>	0x0c / h'0c' / 0c / 0ch
<i>BINARIO</i>	b'1010'

El uso de las mayúsculas y minúsculas en este código obedece a una serie de reglas o normas de estilo, comunes entre los programadores en ensamblador, que, aunque no son obligatorias, facilitan la lectura del código fuente. Un resumen de las reglas empleadas es el siguiente:

Directivas del compilador en mayúsculas.

Nombres de variables en minúsculas

Nemónicos (instrucciones) en mayúsculas

Programa bien tabulado.

Suponemos que usted será ya capaz de comprender el programa. Si no es así le recomendamos que examine el sentido de sus instrucciones en el apartado respectivo de este manual. Para que compruebe si es así le retamos a una sencilla propuesta: el programa tiene instrucciones innecesarias, détéctelas y elimínelas. El resultado correcto sería suma2.asm, contenido también en el CD.

5.2.2 Creando un nuevo proyecto

En **MPLAB** es posible abrir un fichero en ensamblador (*.asm) y ensamblarlo para poder obtener el fichero de entrada de un grabador (*.hex), pero también es posible el uso de proyectos que utilicen varios *.asm, permitiendo así reutilizar código con mayor facilidad, al ser este más modular.

Es, pues, muy conveniente saber crear un proyecto, el cuál se abrirá gracias al menú **Project**, mediante su opción **Open Project...**, muy similar a Open File.



También será posible buscar el icono adecuado cambiando la barra de iconos, para lo cuál emplearemos el que antecede estas líneas.

Existen 4 barras, *Edit*, *Debug*, *Proj* y *User*, cuyo nombre aparece en el registro más a la izquierda de la barra de información (en la parte inferior de la ventana). Tanto en *User* como en *Proj* existen iconos capaces también de abrir un proyecto (una carpeta verde).

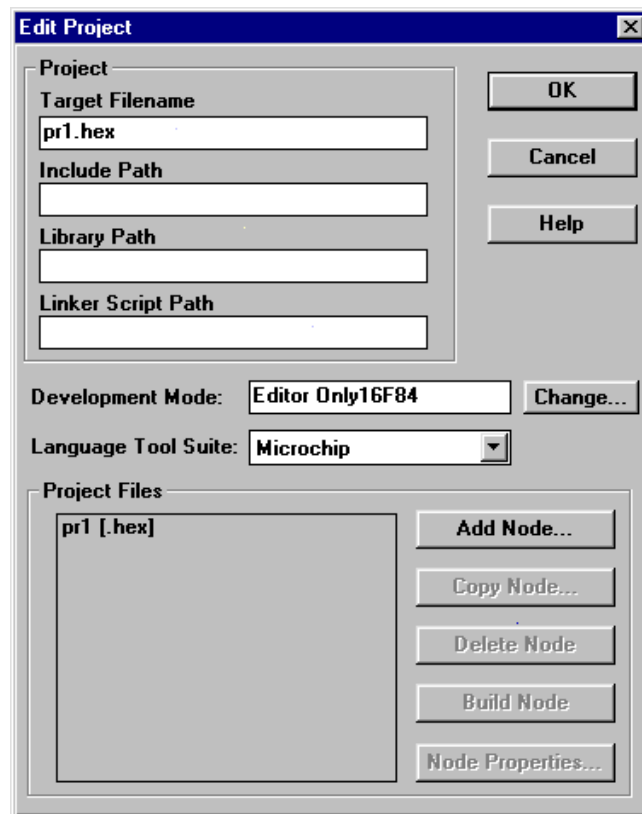


Compruebe usted mismo en esta barra (*Proj*) a qué opción corresponde cada icono situándose sobre él y mirando seguidamente la barra de información (parte inferior de la ventana).

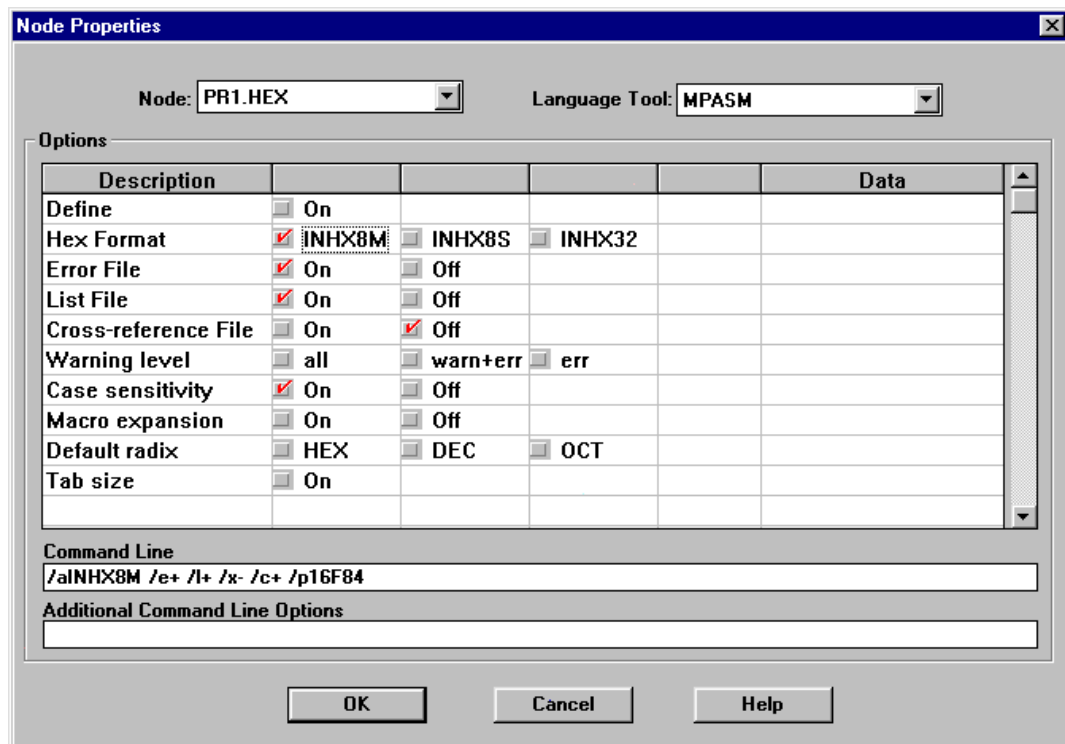


Esta barra (*User*) es una selección de iconos de las otras tres.

Escoja la opción New Project... y dele el nombre pr1.pjt. Llegará entonces a una ventana como la que se muestra en la página siguiente:




Seleccione **pr1 [.hex]** en el frame **Project Files** y se le facilitará la opción **Node Properties**. Seleccione también dicha opción. Obtendrá entonces esta nueva ventana:

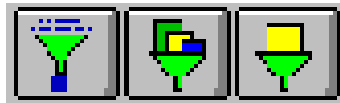


Seleccione el **Hex Format** deseado (de momento **INHX8M**), es decir, el tipo de fichero en el que el ensamblador deberá convertir los *.asm. Todos los formatos suelen ser aceptados por los grabadores, a quienes van destinados. Ya puede darle a **OK**.

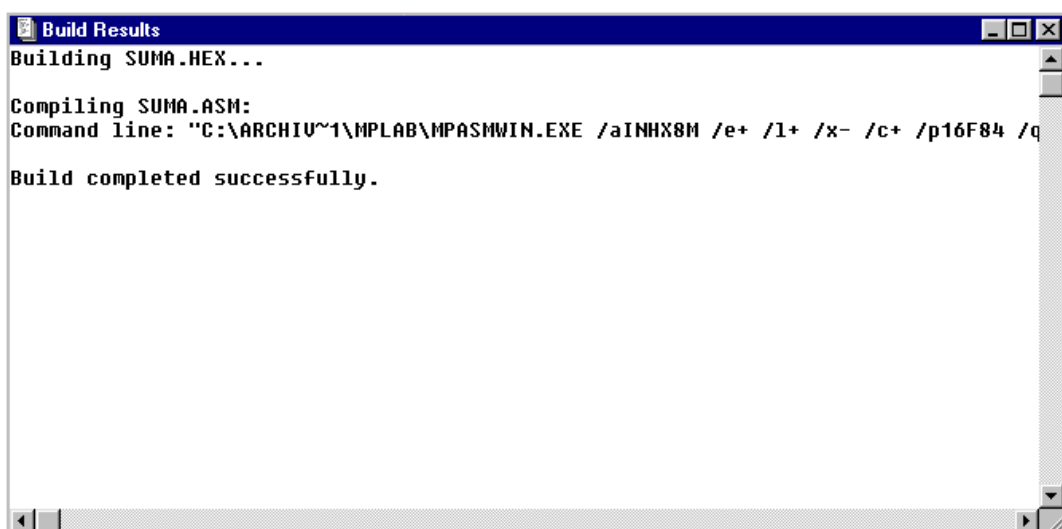
El proyecto sigue estando vacío, y habrá que añadirle nuestro programa (**suma.asm**) para que pueda ensamblarse y probarse. Esto sólo será posible si proyecto y programa están en la misma carpeta. Observará que ahora tiene activa la opción **Add Node** del frame Project Files. Selecciónela y añada **suma.asm**. Ya tenemos un proyecto activo y funcional. En próximos apartados hablaremos del uso de múltiples módulos o librerías.

5.2.3 Ensamblando

 El proceso de ensamblado es muy sencillo. Bastará con, sobre el menú **Project**, pulsar la opción **Build Node**, que ensamblaría sólo la ventana activa, o la opción **Build All**, que ensamblaría todos los nodos del proyecto. Por último la opción **Make Project** ensamblaría todos los nodos de un proyecto y los unificaría en un único *.hex. Los pulsadores dibujados en este párrafo, de la barra *Proj*, contienen todas estas opciones.



Y el resultado es una ventana como esta, con mensajes, errores y warnings.



Corrija los errores que le marque el ensamblador y el programa será sintácticamente correcto. Un número situado tras el nombre del programa le indica la línea exacta a que está asociado el mensaje. Este botón de la barra *Edit* numerará las líneas automáticamente si su listado es largo.



5.2.4 Simulación bajo windows

Una vez corregidos todos los errores el programa ya está listo para ser grabado en el PIC y probado sobre el terreno, pero resulta más práctico (normalmente), y más fiable, si antes se lleva a cabo una simulación por software. **MPLAB** tiene una herramienta de simulación software, el **MPLAB-SIM** (mire la barra y el menú **Debug**).



Notará algo de incomodidad debida a la escasa automatización de algunas tareas: por ejemplo, la simulación comienza mediante la opción **Run** (semáforo verde), y se detiene con **Halt** (semáforo rojo), pero no comenzará a correr si no hace un **Reset** mediante la pulsación de:



Este icono es **Step**, ejecución paso a paso, y avanzará una línea de programa cada vez que lo pulse.



Este icono es **Change Program Counter**, y es utilizado para cambiar el contador de programa (salto a otra línea de memoria de programa en ejecución, para, por ejemplo, probar sólo una rutina específica si el resto ya lo sabemos correcto).



Este icono **Create a New Watch Window** permite editar variables para ver su valor durante la ejecución.



Este comando permite cambiar el valor de direcciones de memoria en tiempo de simulación y en cualquiera de sus áreas (datos, programa, E²PROM, pila)

directamente (por su número) o a través de su nombre (mnemotécnico definido en el ensamblador, como lo es en el ejemplo OPERANDO1). Es muy útil, combinada con las dos anteriores, para probar fácilmente todas las variantes de una rutina o zona de código determinada sin tener que ejecutar para cada una de nuevo todo el código.



Con este comando se pueden definir puntos de parada (**Breaks**) en la ejecución para, mediante **Run**, no necesitar recorrer línea a línea todo el programa si deseamos ejecutar todo un proceso de golpe hasta esa línea.



Define condiciones de parada (**Conditional Breaks**), es decir, valores de variables o pines (E/S) ante las que parar si se producen.

Por favor, experimente con todas estas opciones. Piense que la excesiva sencillez de **suma.asm** hace que *Run* sea poco útil, pero sí lo es *Step*. Observe como la directiva del ensamblador **END** no tiene equivalente en las instrucciones del procesador, con lo que el micro no se detendrá en ese punto. Téngalo en cuenta en el futuro y solucione el problema, por ejemplo, con un **GOTO**.

Si no le funcionan estos comandos, lea el siguiente punto.

5.2.5 Otras opciones del MPLAB

Es interesante también comprobar qué opciones contiene el menú **Options**. Por ejemplo, con su comando **Development mode...**, en el que es posible elegir el tipo de microprocesador sobre el que simular y **activar el modo de simulación (MPLAB-SIM simulator)**, no siempre activo por defecto. Si ha tenido problemas con la simulación en el apartado anterior, seguramente no tendrá activa aún esta opción.

Con **Default Editor Modes** y sus **Current Editor Modes** podrá cambiar las condiciones generales del editor (por ejemplo, cambiar de ensamblador a C, si tiene el C), incluso diferenciando cuál se debe emplear en cada nodo del proyecto.

Con el submenú **Processor Setup** podremos cambiar cosas como la velocidad del reloj (para controlar el tiempo de ejecución) o la activación del **WatchDog**.

A través del menú **Window**, podrá ver la memoria de programa (submenú **Program Memory**) y la de la EEPROM (si lo desea y la va a utilizar; submenú **EEPROM Memory**).

5.3 Ejemplos básicos de programación

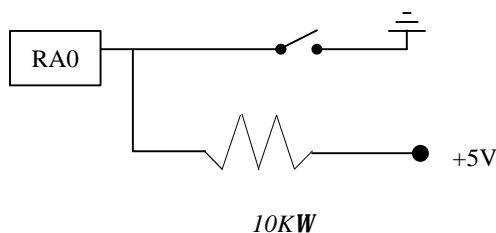
5.3.1 El sistema de E/S. interrupciones y LED's

Debiendo escoger un modelo de PIC para utilizar durante estas prácticas hemos creído conveniente recurrir al más utilizado en la bibliografía como ejemplo, el PIC16C84. Esta elección tampoco es aleatoria y está bien motivada por las siguientes razones: Pertenece a la gama media, con lo que, al sólo contar con 2 instrucciones más que la gama baja, no necesitaremos grandes explicaciones sobre la gama baja. No teniendo tantas funcionalidades como la gama alta, éstas no siempre son necesarias, y sólo suponen un paso más si la gama media es bien comprendida. Su memoria E²PROM lo hace indispensable en muchas aplicaciones (aunque no las más sencillas) y muy recomendable durante la fase de desarrollo, puesto que ésta es de fácil regrabación, y permite no perder el micro en caso de errores en el código. El modelo PIC16F84 tiene memoria FLASH.

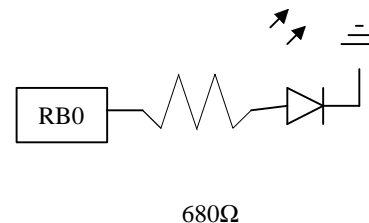
El siguiente paso en esa línea para la comprensión del micro será el estudio de su sistema de E/S. Para ello planteamos el siguiente problema:

Se colocan tres interruptores en las líneas RA0, RA1 y RA2 (puerto A) de un PIC16C84 y cuatro leds en las líneas RB0, RB1, RB2 y RB3 (puerto B), tal y como se refleja en las figuras:

Conexión de interruptores en RA0, RA1 y RA2



Conexión de leds en RB0, RB1, RB2 y RB3



Tomando los interruptores (RA2, RA1, RA0) como un número binario, le sumaremos 2 y sacaremos el resultado por los leds (led apagado=0, led encendido = 1, RB3 bit de mayor peso).

Usaremos el listado del programa **sbinary.asm**, si bien será desarrollado paso a paso.

Lo primero será definir las posiciones de las variables que queremos utilizar. Esto se puede hacer a través de la directiva **include <p16c84.inc>**, que incluye ya la definición (en base a la literatura inglesa) de todos los registros y bits. Busque, como curiosidad, el fichero “16C84.inc” y lea su contenido. Encontrará un archivo *.inc por cada micro de los soportados por el MPLAB.

Nosotros respetaremos la nomenclatura inglesa, pero escribiremos a mano las variables que nos interesan. El motivo es muy simple: en ese p16C84.inc no se ha considerado el problema de los bancos de memoria, que se deben intercambiar a través del registro de estado, de modo que TRISA y TRISB no son funcionales.

De este modo para nosotros PORTA representará al puerto A, PORTB representará al puerto B, TRISA y TRISB serán ellos mismos y W será el registro de estado. Hemos castellanizado el STATUS como ESTADO por comodidad, y hemos llamado BANCO al bit dentro de ESTADO que determina el banco de datos con el que se va a trabajar (recordad que el PIC16C84 tiene dos bancos de datos).

El programa, por lo demás, es bastante evidente. Necesitamos sólo introducir el sentido de los TRIS. Este registro está asociado a los distintos puertos y en él cada bit representa un pin del puerto al que se refiera. Un 0 en uno de sus bits representará que el pin es de salida, y un 1 que es de entrada. De este modo cada pata será independiente, dándonos mayor flexibilidad para la implementación física de los diseños. Es posible incluso, aunque no habitual, cambiar la dirección del pin durante la ejecución del programa. Esta opción puede llegar a darnos la posibilidad de manejar varios dispositivos con un solo puerto.

Ejecute el programa en modo simulación y observe detalladamente como funciona. Recuerde que con la misma dirección de registro referenciamos al puerto y a su tris; todo depende del valor del 5° bit del registro STATUS, que cambia el banco de datos.

```
LIST P=16C84

porta EQU 0x05 ;La dirección 0x05 corresponde al registro porta (puerto A) en el banco0
TRISA EQU 0x05 ; y TRISA en banco 1
portb EQU 0x06 ;La dirección 0x06 corresponde al registro portb (puerto B) en el banco1
TRISB EQU 0x06 ; y TRISB en banco 1
estado EQU 0x03 ; La dirección del registro de estado es la 0x03
banco EQU 5 ; Bit del registro de estado correspondiente al banco de datos

ORG 0
    BSF estado,banco; Activamos banco 1, que contiene los TRISA y TRISB

    MOVLW 0xFF ; Ponemos a 1's el registro W
    MOVWF TRISA ; El registro TRIS del puerto A se pone entero a 1's
                ; cada bit de TRIS representa una patilla idéntica a su
                ; número de orden. Si se pone a 1 representa que es una
                ; entrada.

    MOVLW 0x00 ; Ponemos a 0's el registro W
    MOVWF TRISB ; El registro TRIS del puerto B se pone entero a 0's,
                ; cada 0 representa que su pin correspondiente es de salida.

    BCF estado,banco; Activamos banco 0, que contiene los puertos A y B

inicio MOVF porta,W ; Llevamos a W la entrada de los pulsadores por el puerto A
    ADDLW 0x02 ; Sumamos 2 al W
    MOVWF portb ; Y sacamos el resultado por el puerto B

    GOTO inicio ; Volvemos a empezar

END
```

Con el **goto** final nos aseguramos que la rutina se ejecute constantemente, atendiendo a las posibles nuevas entradas.

Observe mediante el simulador la imposibilidad de cambiar el valor del **PORTA**, intentando alterar el valor de su dirección de memoria, puesto que es una entrada y depende sólo de valores eléctricos. En cambio sobre **PORTB** (salida) sí podremos. Observe también como **trisa** no queda con el valor 0xff, tal y como sería aparentemente lógico, sino con el 0x1f, ya que los pins que quedan a 0 no están implementados (el puerto A sólo tiene 5 entradas).

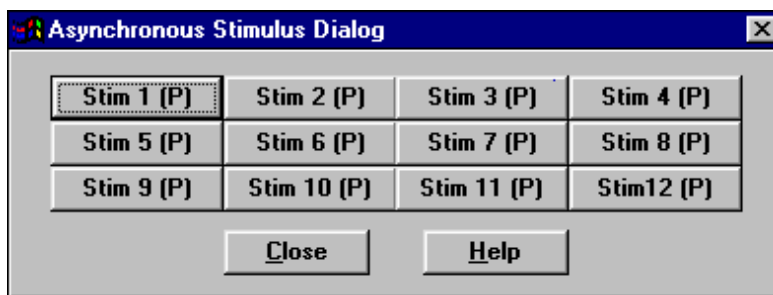
Por último observe la diferencia entre el valor del inspector (el **watch**) en mayúsculas (**TRISA**) y el minúsculas (**trisa**). En mayúsculas **TRISA** referencia siempre a la dirección de registro 0x05, y variará según el banco al que apunte **ESTADO**, puesto que ha sido definida como variable. Se la puede modificar por programa. En minúsculas **trisa** es una variable interna que referencia al banco 1 (dirección 0x85) y no referenciable por programa, De hecho es la única manera de

visualizarla y cambiarla en tiempo de simulación, ya que, bajo Windows, no estamos sometidos a la limitación de los bancos de registros. Puede llegar a ser muy engañoso; no se despiste o variará accidentalmente la salida de un puerto si no cambia convenientemente de banco.

Esta es otra de las incomodidades aún existentes en el manejo del **MPLAB**, todavía en desarrollo, junto a la de que las herramientas de cortar y pegar también sean internas, es decir, ni dejen texto en el portapapeles ni sean capaces de tomarlo¹.

Pruebe a variar los valores de **trisa** y **PORTA**. Comprobará como cuando un puerto es de entrada se toma su valor directamente del patillaje, independientemente del valor del registro del puerto, pero conserva el original (su valor de memoria) si se lo vuelve a considerar de salida. Recuerde, ante posibles respuestas curiosas, que sólo se utilizan en el puerto A los bits implementados (RA4-RA0), y el resto se consideran 0, aunque intente modificarlos

¿ Cómo, pues, cambiar los estímulos de una entrada de un puerto? Emplearemos el menú **DEBUG** con el submenú **SIMULATOR STIMULUS** y la opción **ASINCRONOUS STIMULUS...**, lo cuál dará una ventana como la que sigue:



Cada uno de estos botones simula un estímulo sobre una patilla. La forma de editarlos es pulsar el botón de propiedades del ratón (el derecho), sobre uno de ellos, seleccionar la patilla a la que queremos vincularlo y el tipo de cambio que deseamos realizar con él cada pulsación (poner la entrada a uno o High, a cero o Low, que cambie de valor cada vez que se pulse o Toggle). Tras pulsar el botón habrá de ejecutarse la siguiente instrucción antes de ver los cambios a través del inspector.

¹ Este problema parece estar solucionado en la versión de desarrollo del MPLAB 4.99.07, de reciente aparición.

En el ejemplo RA1(L) pondrá a cero el bit 1 del puerto A, RA2(T) cambiará el valor del bit 2 del puerto A y el resto de botones están sin definir.

También es posible crear secuencias síncronas por programa, en función del tiempo.

Plantéese como ejercicio modificar el programa para sumar las entradas RA3-RA0 y RB3-RB0 (todas pulsadores) en el registro 0x0c y poner las patas RA2, RA1 y RB1 con valor 1.

5.3.2 Contar y visualizar

Además de las tan habituales instrucciones de salto (GOTO) y de rutina (GOSUB) contamos con un tercer tipo de salto condicional muy especial, ya que sólo saltará una línea. Las llamaremos de brinco, como traducción de *skip*. Estas instrucciones son BTFSS registro,bit (que brincaré en caso de que el bit valga 1) y BTFSC registro,bit (que lo hará si es 0).

Otro problema al que nos enfrentaremos por primera vez es el del vector de reset y el vector de interrupción. En la familia de los PIC16CXX² el **vector de reset** (situación del contador de programa en caso de reset) está situado en la dirección de programa 0x00 y el de interrupción en la 0x04. De este modo convendrá comenzar nuestros programas en la dirección 0x05, y rellenar la dirección 0x00 con un simple salto a la misma (GOTO 0x05).

Nos planteamos un nuevo problema: *Crearemos un programa para un PIC16C84 funcionando a 4MHZ encargado de contar hasta 0x5f. Cuando lo alcance se detendrá en un bucle no operativo. El valor del contador se visualizará en 8 diodos LED conectados al puerto B.*

La solución es el programa **cuenta.asm**, cuyo listado hallaremos en la página siguiente.

```

LIST    P=16C84                ; Seleccionamos el micro concreto que deseamos emplear

; Asignación de etiquetas a registros.

f            EQU    0x01        ; Indica que el valor de una operación
                                ; se guardará en el registro, no en el W
portb        EQU    0x06        ; Dirección del registro del puerto B
estado       EQU    0x03        ; Dirección del registro de estado
conta        EQU    0x0C        ; Registro sin asignar
                                ; Lo usamos como variable contadora

        ORG    0                ; El programa comienza en la dirección 0 y
                                ; salta a la dirección 5 para sobrepasar
                                ; el vector de interrupción.
        GOTO inicio

        ORG    5

inicio BSF     estado,5          ; Selecciona el banco 1 para poder acceder al TRISB
        MOVLW 0x00              ; Y se especifica que es de salida
        MOVWF portb             ; Selección del banco 0 para trabajar directamente
        BCF   estado,5          ; con el puerto

        CLRF   conta            ; Ponemos nuestro contador a 0

bucle1 INCF   conta,f            ; conta + 1 --> conta (incrementa el contador)
        MOVF   conta,W          ; conta se carga en W
        MOVWF  portb            ; W se carga en el registro de datos del puerto B
        MOVLW  0x5f             ; W <-- 0x5f (Final de cuenta deseado)
        SUBWF  conta,W          ; conta - W --> W. Si es cero, la cuenta está acabada
        BTFSS  estado,2         ; Explora Z y si vale 1 es que W vale 0
                                ; se produce "brinco" en ese caso por fin de cuenta
        GOTO   bucle1           ; Si Z = 0 se vuelve a bucle1

bucle2 GOTO   bucle2            ; Si Z = 1 se produce un bucle infinito

END

```

Si ejecuta este programa tal cuál se llevará una gran decepción: el estado de la cuenta pasará directamente a 0x5F. ¿ A qué se debe esto? Pensemos que el reloj se mueve a una velocidad de 4 Mhz, lo cuál significa 0'25 μ segundos. Como cada instrucción necesita cuatro ciclos de reloj esto implica un ciclo de instrucción de 1 μ segundo, y un total de 7 instrucciones (con una de salto, que ocupa 2 ciclos) de ejecutará en 8 μ segundos. La cuenta es demasiado rápida. Puede cambiar la frecuencia de reloj a través del menú **OPTIONS**, submenú **clock frequency**³.

Proponemos al lector como ejercicio extra que incorpore un retardo al programa que consiga que sea visualizable. Nosotros planteamos una solución en **cuenta2.asm**. ¿ Podría conseguir contar segundos con un reloj de 1 Mhz? Lamentamos comunicarle que no hemos encontrado ningún método, aparte del manual, para calcular el tiempo de ejecución a través del MPLAB.

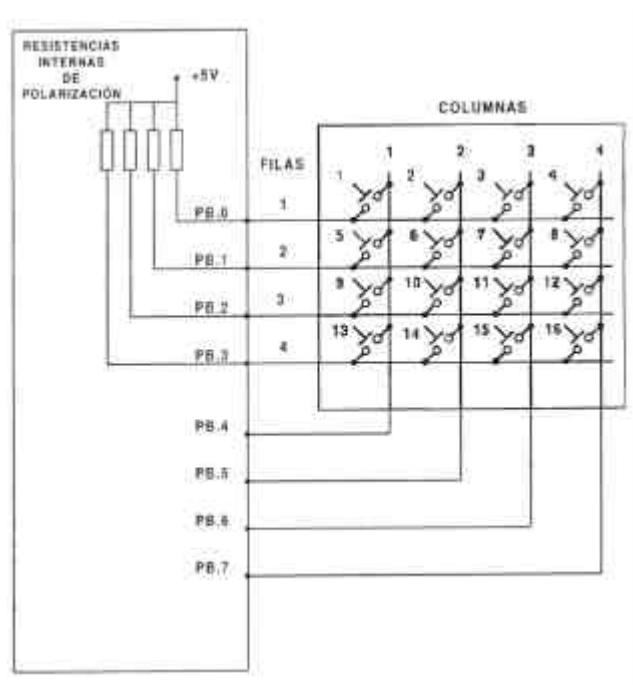
² Los PIC16C5X tienen situado su vector de reset en la última dirección física de memoria

³ En la versión 4.99.07 la activación del WDT se realiza a través del submenú **Development Mode**.

Nos enfrentamos, además, por primera vez al problema del WatchDog. Este es un mecanismo que asegura el reset automático del micro si, pasado un tiempo programable, no se ha ejecutado el comando CLRWDT (CLear WatchDog Timer). Por este motivo no se quedará el micro en modo de bucle infinito salvo que desactive la opción WDT (menú OPTIONS, submenú PROCESSOR SETUP...)

5.3.3 Teclado matricial

Una buena manera de ahorrar líneas de entrada al micro es, en lugar de dedicar una línea exclusiva por cada tecla utilizada, emplear un teclado matricial. El teclado matricial se caracteriza por estar cada una de las teclas conectada a dos líneas (una columna y una fila) que la identifican. De este modo el número de teclas que pueden conectarse es el producto de filas por el de columnas. En el siguiente esquema se muestra la manera correcta de conectar un teclado matricial a un PIC.



La técnica de programación requiere tanto de entradas como de salidas. Las filas están conectadas a las patillas de salida y las columnas a las de entrada.

Se comienza el testeo colocando a '0' la primera fila, y a '1' las restantes. Si la tecla pulsada estuviese en la columna '0', ésta colocaría en su línea un '0' lógico. Bastará con hacer un muestreo de las columnas, buscando el 0, para saber la tecla exacta que fue pulsada en la matriz.

Si no es pulsada ninguna tecla en una fila, las entradas se encuentran en estado flotante, razón por la que son necesarias las resistencias de polarización internas, que mantienen las entradas a nivel alto. Dichas resistencias permiten conectar el teclado matricial sin añadir componentes externos. Su activación se realiza a través del bit **RBPU del registro OPTION**.

El programa ejemplo, llamado **Teclado.asm**, gestiona un teclado hexagesimal (4 filas y 4 columnas) situado en la puerta B. Con el LED situado en RA4 marcaremos si hay o no una tecla pulsada (se encenderá si alguna lo está), mientras que con el resto de LED's conectados al puerto A indicaremos el número binario equivalente a la tecla pulsada.

El WatchDog no está siendo controlado, con lo que le sugerimos que lo desactive para la simulación.

```
LIST P = 16F84           ;Indicamos el modelo de PIC a utilizar

; Definición de registros

opcion    EQU    0X01    ;Dirección del registro OPTION
leds      EQU    0X05    ;Hemos conectado los LED's al puerto A
                        ;La dirección 0x05 corresponde al registro porta
                        ;                               (puerto A) en el banco0
TRISA     EQU    0X05    ;                               y TRISA en banco 1
teclado   EQU    0x06    ;Hemos conectado el teclado al puerto B
                        ;La dirección 0x06 corresponde al registro
                        ;                               portb (puerto B) en el banco1
TRISB     EQU    0X06    ;                               y TRISB en banco 1
estado    EQU    0X03    ; La dirección del registro de estado es la 0x03

; Definición de bits

banco     EQU    5       ; Bit del registro de estado correspondiente
                        ;                               al banco de datos
pulsada    EQU    4       ; Bit del LED que indica si una hay teclas pulsadas
rbpu      EQU    7       ; Bit que activa las resistencias internas del puerto B
Z         EQU    2       ; Bit de registro W con valor cero
C         EQU    0       ; Bit del acarreo. Se pone en el bit 0 al rotar un registro.
```

; Definición de variables

```
cont1    EQU    0X0C ; Contador1 de la pausa
cont2    EQU    0X0D ; Contador2 de la pausa
cont3    EQU    0X0E ; Contador3 de la pausa
tecla    EQU    0X0F ; Número de Tecla
```

```
ORG 0X00 ;Colocamos el vector de reset
```

```
GOTO inicio ; Y saltamos a la dirección 0x05, para evitar
; el vector de interrupción, en 0x04
```

ORG 0X05

```
Inicio    BSF estado,banco ; Cambio de página para cambiar
; los TRIS de cada puerto
MOVW 0x00 ; El puerto A es todo de salida
MOVWF TRISA
MOVW 0xF0 ; Las patillas RB0-RB3 representan las filas del
; teclado, y son salidas
; Las patillas RB4-RB7 representan las columnas,
; y son entradas
MOVWF TRISB
BSF opcion,rbpu ; Conectamos las resistencias de polarización
; interna del puerto B
BCF estado,banco ; Cambio de página de nuevo al banco 0

borrar    CLRF cont1 ; Borra el contador
CLRF cont2 ; Borra el contador
MOVW 0x08
MOVWF cont3 ; Carga el retardo de 0.5 segundos

BCF leds,pulsada ; Borra que la tecla siga pulsada

cuerpo    CLRF tecla ; Tecla actual=0
MOVW 0x0E
MOVWF teclado ; Saca 0 a la fila 1, para testearla

chkcol    BTFSS teclado,4 ; Chequea la columna 0 en busca de un '0'
GOTO numtecla ; si encuentra un 0 muestra el número
; de tecla pulsada
INCF tecla ; si no encuentra el 0, incrementa el número de tecla

BTFSS teclado,5 ; Chequea la columna 1 en busca de un '0'
GOTO numtecla ; si encuentra un 0 muestra el número
; de tecla pulsada
INCF tecla ; si no encuentra el 0, incrementa el número de tecla

BTFSS teclado,6 ; Chequea la columna 2 en busca de un '0'
GOTO numtecla ; si encuentra un 0 muestra el número de tecla

pulsada   INCF tecla ; si no encuentra el 0, incrementa el número de tecla

BTFSS teclado,7 ; Chequea la columna 3 en busca de un '0'
GOTO numtecla ; si encuentra un 0 muestra el número
; de tecla pulsada
INCF tecla ; si no encuentra el 0, incrementa el número de tecla

UltTec?   MOVW 0x10 ; Carga W con el número de teclas +1
SUBWF tecla,w ; y lo compara con el valor actual de tecla
BTFSC estado,Z ; Si hemos llegado a Tecla+1 acabamos
; el ciclo de filas
GOTO cuerpo ; y no se habrá pulsado ninguna tecla
BSF estado,C ; Y ponemos a 1 el bit C para...
RLF teclado ; que la fila 1 pase a ser un 1 en la rotación de pilas,
; y el 0 se desplace

GOTO chkcol

numtecla  MOVF tecla
MOVWF leds ; Pasa el valor de la tecla a los LED's
```

	BSF leds,pulsada	; y activa el indicador de tecla pulsada
pausa	DECFSZ cont1	; Bucle anidado de 3 niveles
	GOTO pausa	; que retarda aprox. 0.5 segundos
	DECFSZ cont2	
	GOTO pausa	
	DECFSZ cont3	
	GOTO pausa	
	GOTO borrar	
END		

5.3.4 Tablas y subrutinas

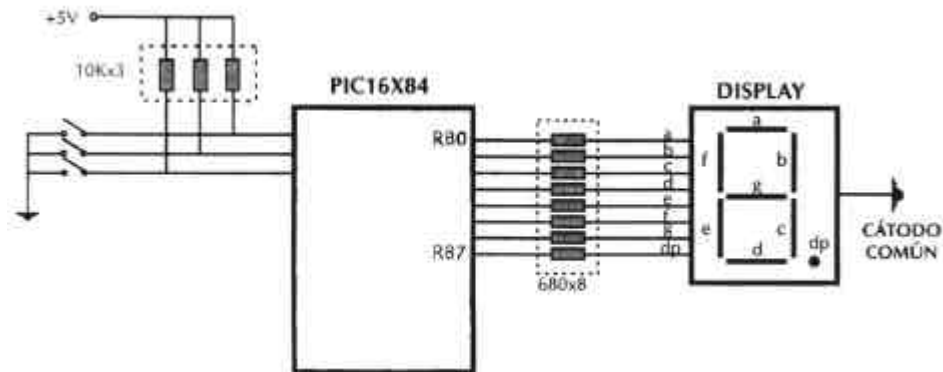
El uso de rutinas, siendo sencillo y no requiriendo un apartado específico, es esencial, puesto que simplifica los programas, haciéndolos, además, más modulares. Llamaremos a una subrutina con la orden **CALL**, seguida de la etiqueta que la encabeza o su dirección en memoria. Para regresar a la siguiente instrucción tras el **CALL** bastará con situar, en la última línea de la subrutina, el comando **RETURN**; también será posible emplear **RETLW k**, el cuál se diferencia del anterior por situar en el registro **W** el valor **k**.

En los procesadores de gama media y alta existe una manera específica de regresar en caso de interrupción: **RETFIE**. Las interrupciones generan un salto a la dirección 0x04 que es tratado como una subrutina para permitir la continuación normal del programa a partir del punto en que se llevó a cabo la interrupción. Este regreso se lleva a cabo mediante **RETFIE**, que, además de restaurar el contador de programa (**PC**) habilita de nuevo las interrupciones (ya que estas son deshabilitadas mientras se está atendiendo una).

La pila de la gama media permite guardar hasta 8 saltos, es decir, admite un máximo de 8 saltos a subrutina anidados. Recuerde este dato para no excederse en las llamadas a las mismas ni en los niveles de un método tipo “Divide y Vencerás”. Recuerde también siempre un posible salto por interrupción, para no pasar de 7 si ésta está habilitada. En la gama baja sólo contará con dos niveles en la pila.

Una tabla, en cambio, es una lista **ROM** de constantes en la memoria de programa, y que pueden ser desde notas musicales a caracteres **ASCII**. Son especialmente útiles para la conversión de unos códigos a otros, como, por ejemplo, para pasar un hexagesimal a un display. Para generarlas se aprovecha la cualidad de **RETLW** para situar un dato en el registro **W**.

Supongamos que deseamos sacar, de forma secuencial, un número del 0 al 9 en un display de 7 segmentos conectado a la puerta B, tal y como figura en el esquema siguiente:



En el siguiente programa, llamado **Cuenta.asm**, se necesitan ambas cosas, tanto las rutinas como las tablas. El display elegido es de cátodo común, es decir, los segmentos se iluminan cuando las salidas de la puerta B están a nivel alto.

En la siguiente tabla podrás comparar el código hexagesimal y su equivalente en 7 segmentos:

HEXAGESIMAL	7-SEGMENTOS
\$00	\$3F
\$01	\$06
\$02	\$5B
\$03	\$4F
\$04	\$66
\$05	\$6D
\$06	\$7D
\$07	\$07
\$08	\$7F
\$09	\$6F

```

LIST P = 16F84          ;Indicamos el modelo de PIC a utilizar

; Definición de registros

portb      EQU    0x06  ;Hemos conectado el display al puerto B
              ;La dirección 0x06 corresponde al registro PORTB
              ;(puerto B) en el banco 1
TRISB      EQU    0X06  ; y TRISB en banco 1
Estado     EQU    0X03  ; La dirección del registro de estado es la 0x03
pc         EQU    0x02  ; Contador de Programa, es decir,
              ;dirección de memoria actual de programa

; Definición de bits

banco      EQU    0X05  ; Bit del registro de estado correspondiente
              ; al banco de datos
Z          EQU    0X02  ; Bit indicador de que el registro W está a cero

; Definición de constantes

w          EQU    0      ; Destino de operación = w
f          EQU    1      ; Destino de operación = registro

; Definición de variables

contador   EQU    0X0C   ; Contador
digito     EQU    0X0D   ; Para almacenar el dígito

; Comienzo del programa.

ORG 0X00   ; Cubrimos el vector de reset

          GOTO inicio    ; Saltamos a la primera dirección tras el vector de interrupción

; ***** Inicialización de variables *****

ORG 0X05

inicio     BSF estado,banco ; Cambiamos a la segunda página de memoria
           CLRf TRISB       ; Programa la puerta B como de todo salidas
           BCF estado,banco ; Volvemos a la página 0.
           CLRf portb       ; Apaga el display, por si había residuos
           CLRf contador    ; Borra el contador (dirección 0x0C)
           CLRw              ; Borramos el registro W

```



```

; ***** Cuerpo Principal *****
Reset      CLRF digito      ; Comienza a contar por el 0

Siguien     MOVF digito,w    ; Coloca el siguiente dígito a evaluar en W
            CALL Tabla      ; Llama a la subrutina para coger el dato
                        ; y hacer la conversión decimal-7 segmentos
            MOVWF portb

Pausa       DECFSZ contador  ; Decrementa contador y repite
            GOTO Pausa      ; hasta que valga 0
            INCF digito,f    ; Incrementa el valor del dígito al siguiente
            MOVF digito,w    ; Pone el valor del dígito en W
            XORLW 0x0A       ; Chekea si el dígito sobrepasa el valor 9
            BTFSC estado,Z   ; Comprobando con un xor si W vale 0 (Z=1)
            GOTO Reset      ; Si Z=1 resetea el dígito y comienza de nuevo la
cuenta      GOTO Siguien     ; En caso contrario, continua la cuenta
                        ; con el siguiente dígito

Tabla       ADDWF pc,f       ; Suma al contador de programa el valor de offset, es decir,
valor,      ; el valor del dígito. Así se genera un PC distinto según su
                        ; asegurando que vaya a la línea correcta de la tabla
            RETLW 0x3F       ; 0 en código 7 segmentos
            RETLW 0x06       ; 1 en código 7 segmentos
            RETLW 0x5B       ; 2 en código 7 segmentos
            RETLW 0x4F       ; 3 en código 7 segmentos
            RETLW 0x66       ; 4 en código 7 segmentos
            RETLW 0x6D       ; 5 en código 7 segmentos
            RETLW 0x7D       ; 6 en código 7 segmentos
            RETLW 0x07       ; 7 en código 7 segmentos
            RETLW 0x7F       ; 8 en código 7 segmentos
            RETLW 0x6F       ; 9 en código 7 segmentos

END

```

5.3.5 Manejo de interrupciones

GENERALIDADES SOBRE LAS INTERRUPCIONES

La **interrupción** es una técnica que coloca al programa temporalmente en suspenso mientras el microcontrolador ejecuta otro conjunto de instrucciones en respuesta a un suceso. Las causas de una interrupción pueden ser externas, como la activación de una patilla con el nivel lógico apropiado, e internas, como las que pueden producirse al desbordarse un temporizador como el TMR0.

Los **sucesos internos capaces de producir una interrupción** más destacables son el *desbordamiento del temporizador TMR0*, *fin de la escritura de la EEPROM*, *finalización de la conversión A/D*. Los **sucesos externos** principales son la *activación del pin 0 de la puerta B (PB0/INT)*, *el cambio de estado en las patitas 4-7 de la puerta B*, y *el desbordamiento del temporizador TMR0*.

Cuando se produce una interrupción el procesador ejecuta una **Rutina de Servicio de Interrupción (RSI)**, y, al terminar, el programa principal continúa donde fue interrumpido. La dirección en la que se debe situar la rutina de interrupción es la 0x04, y es recomendable, para terminarla, usar la instrucción RETFIE, en lugar de RETURN, puesto que, al activarse una interrupción, el mecanismo de las mismas se deshabilita como medida de seguridad. RETFIE sirve para rehabilitarlas.

Como las rutinas pueden modificar el contenido de los registros del procesador, al iniciarlas conviene guardar en la pila el valor de los mismos y restaurarlos antes del RETURN. Antes de regresar la RSI debe determinar la causa de la interrupción, borrar la bandera apropiada antes de salir y, por supuesto, dar servicio a la interrupción.

A primera vista, **salvar y restaurar los registros sin modificar sus contenidos** no es una tarea fácil. El contenido del registro W debe guardarse primero, junto con todos los registros que han pasado por W para el almacenamiento temporal de sus posiciones. El hecho de mover W a otro registro corrompe la bandera Z, modificando el registro de Estado. Microchip recomienda una secuencia de código que permite salvar y restaurar los registros sin modificarlos. La mostramos en la siguiente secuencia de código:

```

; ***** SALVAR *****
MOVWF Almacen1_W      ; Guardamos contenido de W en su sitio
SWAPF estado,w        ; Swap del contenido de estado en W
MOVWF Almacen2_S      ; Guarda el contenido de ESTADO
...
...
...

; ***** FIN RUTINA RSI *****
SWAPF Almacen2_S,w    ; Deja estado como estaba
MOVWF estado         ; Y lo restaura
SWAPF Almacen1_W,f
SWAPF Almacen1_W,w
RETFIE

```

La instrucción SWAPF mueve los datos sin afectar a la bandera Z del registro de ESTADO. Aunque los conjuntos de 4 bits se invierten en el proceso, posteriormente son restaurados en su situación inicial. Si se empleara la instrucción MOVF se corrompería el bit Z.

REGISTROS DE INTERRUPCIÓN Y BANDERAS

Cada causa de interrupción actúa con dos señales. Una de ellas actúa como señalizador o bandera que indica si se ha producido o no la interrupción, mientras que la otra funciona como permiso o prohibición de la interrupción en sí. Los PIC 16C84 y 16C71 disponen de 4 fuentes de interrupción válidas o no por la puesta a 1 de los correspondientes bits del registro **INTCON (0x0B ó 0x8B)**.

GIE	EEIE/ ADIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
-----	---------------	------	------	------	------	------	------

registro INTCON

Bit 0

Bit 7

El bit **GIE** (Global Interrupt Enable) habilita todas las interrupciones. Cada tipo de interrupción tiene, a su vez, otro bit que la habilita o deshabilita. Las interrupciones son capaces de despertar al chip de su estado de reposo. El bit GIE se borra en cuanto se está atendiendo una interrupción, para evitar que se atienda otra. Volverá a valer 1 si se vuelve de la interrupción mediante RETFIE, como ya ha sido explicado varias veces. Preferimos reiterarnos por ser motivo de posibles problemas.

Todas las interrupciones saltan a la dirección 0x04, por lo que será labor del programador identificar la causa de interrupción.

El bit **RBIE** habilita la interrupción RB, es decir, interrupción ante cambios en las patas RB4-RB7. **RBIF** es la bandera que indica que se ha producido esta interrupción.

El bit **INTE** activa la interrupción por la pata INT/RB0. El bit **INTF** es la bandera que indica si se ha producido esta interrupción.

El bit **T0IE** habilita la interrupción por desbordamiento del TMR0. El bit **T0IF** es la bandera que indica si se ha producido la interrupción.

El **bit 6** del registro INTCON es distinto para el 16C71 (ADIE) y para el 16C84 (EEIE). En el 16C71 activa las interrupciones procedentes del convertor A/D, y el 16C84 las procedentes de la E²PROM. Sus respectivas banderas están en el registro ADCON1 ó EECON1.

EJEMPLO DEL MANEJO DE INTERRUPCIONES. EL TMR0.

Vamos a hacer en el programa **parpadeo.asm** que, gracias a la interrupción del TMR0, que un LED parpadee con una frecuencia de 200 ms. Una vez inicializadas las puertas, el predivisor de TMR0 se carga con 78 y se habilita la interrupción (GIE+T0IE)

El número de cuentas es FF-N, siendo N el número con el que se carga TMR0. Como se carga con 78, $N = 256 - 78 = 178$. Si ponemos el divisor de frecuencia del TMR0 a 128 (bit PS0, PS1 y PS2 del registro OPTION a 1 1 0) y con un oscilador de 4 MHz, donde el ciclo de instrucción es de un μ segundo, ocurre que...

$$\text{Tiempo Total} = N * \text{valor predivisor} * \text{ciclo instrucción}$$

$$\text{Tiempo Total} = 178 * 128 * 1 \mu\text{s} = 9984 \mu\text{s} = 9'98 \text{ ms}$$

Mediante el programa principal comprobamos que el valor del contador es 20 (200 ms). Si es así el LED es conmutado, encendiéndolo o apagándolo según su estado anterior.

No tenemos en cuenta el WatchDog, por lo que conviene deshabilitarlo, y nos son indiferentes los valores del W en todo momento, por lo que nos es igual guardar su valor o no al saltar a la rutina de interrupción.

```

LIST P = 16F84           ;Indicamos el modelo de PIC a utilizar

; Definición de registros

portb    EQU    0x06      ; Puerto B
TRISB    EQU    0x06      ; y TRISB en banco 1
estado   EQU    0x03      ; La dirección del registro de estado es la 0x03
intcon    EQU    0x0B      ; Registro controlador de interrupciones
opcion   EQU    0x81      ; Registro OPTION
tmr0      EQU    0x01      ; Registro del Timer0 (TMR0)

; Definición de bits

banco     EQU    0x05      ; Bit del registro de estado correspondiente
                        ; al banco de datos. En ESTADO
Z         EQU    0x02      ; Bit indicador de que el registro W está a cero. ESTADO
t0if      EQU    0x02      ; Bit de INTCON que indica que se produjo
                        ; interrupción por desbordamiento del timer0
t0ie      EQU    0x05      ; Bit de INTCON que habilita o no la interrupción
                        ; por desbordamiento del timer0

; Definición de constantes

w         EQU    0         ; Destino de operación = w
f         EQU    1         ; Destino de operación = registro

; Definición de variables

contador   EQU    0x0C      ; Contador

; Comienzo del programa.

ORG 0x00      ; Cubrimos el vector de reset

                GOTO inicio      ; Saltamos a la primera dirección tras
                        ; el vector de interrupción

ORG 0x04      ; Cubrimos el vector de interrupción

                GOTO RSI      ; Y saltamos a la rutina de servicio de interrupción

; ***** Inicialización de variables *****

ORG 0x05

inicio      BSF estado,banco      ; Seleccionamos el banco 1
                MOVLW 0x06      ; En binario 0000 0110
                MOVWF opcion      ; Ponemos el predivisor a 128

```

```

, *****
,
, *** OPTION.7 = 0 Resistencias de polarización deshabilitadas ***
, *** OPTION.6 = 0 Interrupción externa por flanco bajada (no se usa) **
, *** OPTION.5 = 0 Fuente de reloj interna ***
, *** OPTION.4 = 0 Flanco de señal externa (no lo usamos) ***
, *** OPTION.3 = 0 Divisor asignado al TMR0 ***
, *** OPTION.2 = 1 OPTION.1= 1 OPTION.0= 0 División por 128 ***
, *****
,
, CLRF TRISB ; La puerta B es toda de salida
, BCF estado,banco ; Volvemos a la página 0
, CLRF portb ; Borramos todos los LEDS
, CLRF contador ; Contador = 0
, MOVLW 0xB2 ; Cargamos el timer con 78 decimal (0xB2)
, MOVWF tmr0
, MOVLW 0xA0 ; 1010 0000 en binario
, MOVWF intcon; Habilitamos GIE y TOIE (interrupción del timer0)

, ***** Cuerpo principal *****
,
, ***** Mira si hay 20 cuentas de 10 ms *****
, ***** Y, si las hay, cambia el LED *****
,
, Bucle MOVF contador,w ; Se incrementa cada 10 ms en uno al
, ; producirse la interrupción
, XORLW 0x14 ; Ha llegado a 200 ms si llevamos 20 (0x14) cuentas
, BTFSS estado,Z ; Si es así, salta para cambiar el LED
, GOTO Bucle ; Si no es así, prueba otra vez
, CLRF contador ; El contador vuelve a 0 para iniciar el nuevo ciclo
, BTFSS portb,1 ; Está encendido ? Si sí, apaga
, GOTO Encien
, Apaga BCF portb,1 ; Apaga el LED
, GOTO Bucle
, Encien BSF portb,1 ; Enciende el LED
, GOTO Bucle

, ***** RSI: Rutina de servicio de interrupción *****
, ***** Salta al desbordarse el TMR0, cada 10 ms *****
,
, RSI BTFSS intcon,t0if ; Salta si la interrupción es TMR0
, RETFIE ; Interrupción desconocida, regresar
, ; (es un mecanismo de seguridad).
, INCF contador,f ; El contador es incrementado cada 10 ms
, MOVLW 0xB2 ; Recarga el valor inicial del TMR0
, MOVWF tmr0
, BCF intcon,t0if ; Borra la bandera de interrupción
, BSF INTCON,t0ie ; Habilita de nuevo la interrupción
, RETFIE

,
, END

```

5.3.6 Manejo de una pantalla LCD. Creación de una librería.

INTRODUCCIÓN

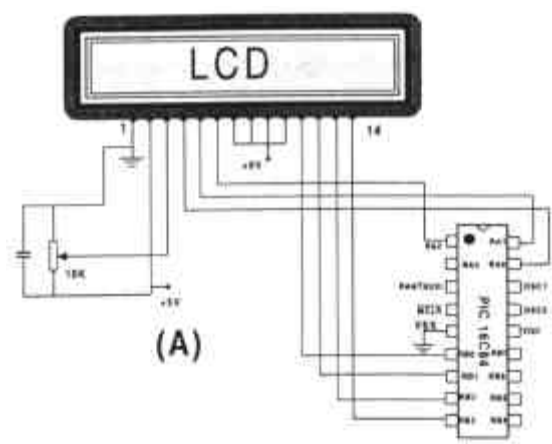
Una LCD estándar es una pantalla de cristal líquido con una matriz de 16, 32, 40 u 80 caracteres de 5x7 pixeles, contando, además, con un microcontrolador (generalmente el Hitachi 44780) que lo gobierna. Normalmente cada línea contiene entre 8 y 80 caracteres, y suelen ser capaces de mostrar caracteres ASCII, japoneses, griegos...; o símbolos matemáticos. Su bus de conexión puede ser de 4 u 8 bits.

El consumo de este tipo de módulos es muy bajo (7'5mW), y, gracias a su sencillo manejo, son ideales para dispositivos que requieren una visualización pequeña o media.

Expondremos el uso de una librería, la LCD.LIB, preparada para una pantalla de cristal líquido con dos líneas de 16 caracteres y una conexión de 8 bits.

ADAPTACIÓN DE UNA PANTALLA LCD

El módulo LCD que vamos a trabajar tiene 14 patillas, cuya descripción se hace en la figura que sigue a este párrafo. Su alimentación es de +5 V, y la regulación del contraste se realiza dividiendo esos +5V mediante un potenciómetro de 10 k. Para el módulo de 8 bits requeriremos 11 líneas (uno de 4 bits necesitaría sólo 7). De ellas hay tres de control, que son EN (habilitación), I/D (Instrucción/Datos) y R/W (Lectura/Escritura). En un modo de 4 bits usaríamos sólo las líneas DB4-DB7 de datos.



La activación de la línea EN (habilitación) es la que permite a la LCD leer el resto de líneas, es decir, si la desactivamos no reaccionará ante los cambios en el resto de líneas. La línea R/W se conectará a masa, para ahorrar una línea, en todos los casos en los que no sea necesario el modo de lectura.

Pin	Nombre del pin	Función del pin
01	Vss	Masa
02	Vdd	+ 5 V
03	Vo ó Vee	Ajuste de contraste
04	I/D ó RS	Selección de modo
05	R/W	Lectura / Escritura
06	E ó EN	Validación (1) / Deshabilitación (0)
07	DB0	Línea de datos (bit de menos peso)
08	DB1	Línea de datos
09	DB2	Línea de datos
10	DB3	Línea de datos

11	DB4	Línea de datos
12	DB5	Línea de datos
13	DB6	Línea de datos
14	DB7	Línea de datos (bit de mas peso)

Habitualmente el puerto A del micro es utilizado para manejar las líneas de control (en la LCD.LIB PORTA.2 se conectará a EN, y habilitará la LCD, PORTA.1 manejará la lectura/escritura, y, finalmente, la PORTA.0 se encargará de la selección de modo), mientras la puerta B es utilizada para datos.

La secuencia de escritura debe seguir los siguientes pasos:

- 1) Línea I/D a 0 o a 1, según se trate de comandos o datos
- 2) Línea R/W a 0 (1 en caso de escritura)
- 3) Línea EN a 1 (se habilita la LCD)
- 4) Escritura de datos en el bus DB.
- 5) Línea EN a 0 (deshabilitación de la LCD)

La misma secuencia en un módulo de 4 bits cambiaría:

- 1) Línea I/D a 0 o a 1, según se trate de comandos o datos
- 2) Línea R/W a 0 (1 en caso de escritura)
- 3) Línea EN a 1 (se habilita la LCD)
- 4) Escritura en los 4 bits de mayor peso del DB de la LCD.
- 5) Línea EN = 0
- 6) Línea EN = 1
- 7) Escribir de nuevo los 4 bits de menor peso
- 8) Línea EN = 0 (deshabilitación de la LCD).

Las dos secuencias de 4 bits se concatenarían dentro del LCD para formar 8 bits.

Al resetear una LCD o encenderla ésta se queda a la espera de instrucciones. Usualmente se suele empezar encendiendo la pantalla, colocando el cursor y configurando la escritura de derecha a izquierda.

La LCD contiene una RAM propia en la que almacena los datos, que se denomina DDRAM. Independientemente del número de caracteres visibles, la DDRAM contará con 80 posiciones. Los caracteres no visibles se visualizarán provocando un desplazamiento.

La utilización de la LCD es lenta. Una escritura o lectura puede tardar entre 40 y 120 μ segundos; otras instrucciones pueden llegar a los 5 ms. Para lograr que el PIC no necesite esperar tiene una instrucción de 1 μ seg que lee la dirección del contador y una bandera interior de ocupado. Cuando la bandera de ocupado (BF) está a 1, la LCD no puede leer ni escribir.

En nuestro ejemplo, del que a continuación mostramos el esquema, las líneas de datos se comparten con el teclado y una barra de diodos. Compartir la puerta B es una de las ventajas del PIC, puesto que le da una gran capacidad de reconfiguración, por su sencillez y rapidez.

HABILITACIÓN DE UNA LCD. RUTINA LCD_HABILITA

La línea EN de habilitación de una LCD necesita estar activada durante, al menos, 500 ns. La rutina LCD_Habilita de la LCD.LIB se asegura de que así sea, siendo LCDE = PORTA.2, es decir, EN:

```
LCD_Habilita      ; Envía un impulso de habilitación de 500 ns a la LCD para
                  ; completar la operación de escribir un registro o un carácter
                  ; La instrucción NOP sólo es necesaria para procesadores de
                  ; una velocidad superior a 8 MHz. Si el procesador es de
                  ; más de 16 MHz, se debe añadir un segundo NOP

                  BSF LCDE      ; Pone a 1 la línea EN (habilita la LCD)
                  NOP          ; Pausa para 250 ns extra
                  ; (según velocidad del micro)
                  BCF LCDE      ; Pone a 0 la línea EN (deshabilita la LCD)
                  RETURN
```

SELECCIÓN DE MODO (COMANDO/DATOS).

La línea I/D selecciona entre el modo comando si vale 0 o el modo datos si es 1. Una llamada tipo CALL LCD_Comando asegurará dicho modo antes de una habilitación de la LCD; lo mismo sucederá con LCD_Carácter.

```
LCD_Comando      ; Carga W con una constante software LCD de la tabla de
                  ; igualdades. LCD_Comando saca el comando a la LCD y
                  ; activa la línea de comando de la LCD, y la propia LCD
                  ; mediante la llamada a LCD_Habilita, completando así
                  ; el comando.

                  BCF LCDModo ; Entra en modo registro
                  MOVWF portb ; y envía W a LCD en puertoB.
                  ; W, por tanto, habrá de tener ya el valor
                  ; del comando antes de que el programa
                  ; invoque a LCD_Comando
                  CALL LCD_Chequea ; Chequea la bandera de LCD ocupada
                  GOTO LCD_Habilita ; Envía el comando

LCD_Carácter      ; Carga W con el código ASCII del carácter que desea
                  ; enviar a la LCD. Activará para ello el modo datos
                  ; y, posteriormente, la LCD mediante una llamada a
```



```

; LCD_Habilita para completar el envío del mismo.
BCF LCDModo ; Envía modo registro
MOVWF portb ; y envía W a LCD en puertoB
; W, por tanto, habrá de tener ya el
; valor del carácter antes de la llamada a
; esta rutina.
CALL LCD_Chequea ; Chekea la bandera de LCD ocupada
BSF LCDModo ; Envía modo datos
GOTO LCD_Habilita ; y envía el carácter

```

Debemos recordar que es la línea R/W la que determina si se lee o escribe, debiendo estar debidamente activada según nuestros deseos antes de cualquier intento de acceso a la LCD.

COMANDO DE LA PANTALLA LCD

Aunque pueden variar, en el caso que nos ocupa y en el estándar los comandos de la LCD son:

Comando	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Borra Pantalla	0	0	0	0	0	0	0	0	0	1
Cursor a Casa	0	0	0	0	0	0	0	0	1	*
Modo Introducción	0	0	0	0	0	0	0	1	I/D	S
Pantalla On/Off	0	0	0	0	0	0	1	D	C	B
Modo Desplazamiento	0	0	0	0	0	1	S/C	R/L	*	*
Función	0	0	0	0	1	DL	líneas	Font	*	*
Dirección CGRAM	0	0	0	1	Dirección CGRAM					
Dirección DDRAM	0	0	1	Dirección DDRAM						
Lectura ocupado y dirección contador	0	1	BF	Dirección DDRAM						
Escribe RAM	1	0	Escribe Dato							
Lee RAM	1	1	Lee Dato							

Borra Pantalla: La borra y sitúa el cursor en su posición inicial (la 0).

Cursor a Casa: El cursor va a la posición inicial (la 0), pero sin borrar nada.

Modo instrucción: Configura la dirección del cursor I/D. Cuando I=1 incrementa la posición del cursor, y D=0 la decrementa. Mientras S=1 significa que hay desplazamiento en la pantalla. La operación se ejecuta durante la I/O de los datos.

Pantalla On/Off: Coloca en movimiento al cursor o genera desplazamiento en la pantalla. D para toda la pantalla, C para cursor On/Off, y B hace parpadear el cursor.

Desplazamiento Cursor/Pantalla: S/C indica el movimiento del cursor o desplazamiento en la pantalla, R/L la dirección a derecha o izquierda. No se varía el contenido de la DDRAM.

Función: DL indica la longitud de datos del interfaz; N el número de líneas de la pantalla y F el tipo de caracteres.

Dirección CGRAM: Coloca el dato enviado o recibido en la CGRAM después de este comando.

Dirección DDRAM: Coloca el dato enviado o recibido en la DDRAM después de la ejecución de este comando.

Bandera de ocupado BF: Lee BF indicando si hay una operación interna en curso y lee, además, el contenido de la dirección contador.

Escribe RAM: Escribe un dato en la RAM (ya sea DDRAM o CGRAM).

Lee RAM: Lee datos de la RAM (ya sea DDRAM o CGRAM).

Nombre Bit	Estado y Funcionamiento	
I/D	0 = Decrementa posición cursor	1 = Incrementa posición cursor
S	0 = Sin desplazamiento	1 = Con desplazamiento
D	0 = Pantalla Off	1 = Pantalla On
C	0 = Cursor Off	1 = Cursor On
B	0 = Parpadeo cursor Off	1 = Parpadeo cursor On
S/C	0 = Mueve el cursor	1 = Desplaza la pantalla
R/L	0 = Desplaza a la izquierda	1 = Desplaza a la derecha
DL	0 = Interfaz de 4 bits	1 = interfaz de 8 bits
Líneas	0 = 1 línea datos visible	1 = 2 líneas datos visibles
Font	0 = 5 x 7 pixeles	1 = 5 x 10 píxel
BF	0 = Puede aceptar instrucción	1 = Operación interna en curso

DEFINICIÓN DE VALORES DE CONSTANTES DE COMANDOS PARA UTILIZAR EN LAS RUTINAS DE LCD.LIB

Nombre constante	Valor	Significado
LCDLinea1	0x80	Coloca cursor en la posición 1 línea 1
LCDLinea2	0x0C	Coloca el cursor en la posición 1 línea 2
LCDCLR	0x01	Borra la pantalla + LCDLinea1
LCDCasa	0x02	Como LCDLinea1
LCDInc	0x06	El cursor incrementa su posición tras cada carácter
LCDDec	0x04	El cursor decrementa su posición tras cada carácter
LCDOn	0x0C	Enciende la pantalla
LCDOff	0x08	Apaga la pantalla
CursOn	0x0E	Enciende pantalla mas cursor
CursOff	0x0C	Apaga pantalla mas cursor
CursBlink	0x0F	Enciende la pantalla con cursor parpadeando
LCDIzda	0x18	Desplaza los caracteres mostrados a la izquierda
LCDDecha	0x1C	Desplaza los caracteres mostrados a la derecha
CursIzda	0x10	Mueve el cursor una posición a la izquierda
CursDecha	0x14	Mueve el cursor una posición a la derecha
LCDFuncion	0x38	Programa una interface 8 bits, pantalla 2 líneas, fuente 5x7 pixeles
LCDCGRAM	0x40	Programa el generador de caracteres del usuario RAM

DIRECCIONADO DE LA RAM

La RAM de una LCD no tiene direccionamiento continuo y lineal, pues el mapa depende de los caracteres y líneas que tenga el módulo.

Tamaño Pantalla	Visible	
Una Línea	Posición Carácter	Dirección DDRAM
1 x 8	00 – 07	0x00 – 0x07
1 x 16	00 – 15	0x00 – 0x0F
1 x 20	00 – 19	0x00 – 0x13
1 x 24	00 – 23	0x00 – 0x17
1 x 32	00 – 31	0x00 – 0x1F
1 x 40	00 – 39	0x00 – 0x27

Tamaño Pantalla	Visible	
Dos Línea	Posición Carácter	Dirección DDRAM
1 x 16	00 – 15	0x00 – 0x0F + 0x40 – 0x4F
1 x 20	00 – 19	0x00 – 0x13 + 0x40 – 0x53
1 x 24	00 – 23	0x00 – 0x17 + 0x40 – 0x57
1 x 32	00 – 31	0x00 – 0x1F + 0x40 – 0x5F
1 x 40	00 – 39	0x00 – 0x27 + 0x40 – 0x67

LISTADO DE LA LIBRERÍA

; LCD.LIB

```

, *****
,
, ***   LCD.LIB proporciona las siguientes funciones:   ***
, ***
, ***   - Configuración de las puertas                     ***
, ***       - Comandos en modo registro                   ***
, ***       - Exploración de LCD Busy - Ocupado           ***
, ***       - LCD Enable (habilitación)                   ***
, *****

```

; Variables

; Ninguna

; Requisitos

```

; Dos niveles libres de pila para llamadas anidadas.
; La rutina LCD_Inic necesita 5 ms para inicializar
; el LCD. Pausas mayores son aceptables. LCD_Inic
; llama a Pausa_5ms que debe ser incluida en la llamada
; del programa. Si el programa que usa esta librería
; tiene un bucle de retardo mayor de 5 ms, puede usarlo
; colocándole la etiqueta Pausa_5ms la etiqueta de su rutina.

```

; Uso

; Para iniciar la LCD después del encendido:

```

; 1.- Llama LCD_Port, que inicializa Puerta A y Puerta B para la LCD
; 2.- Llame LCD_Inic que inicializa el controlador de la LCD

```

; Para escribir un comando o un carácter en la pantalla LCD:

```

; 1.- Llame LCD_Port que inicializa Puerta A y Puerta B para la LCD
; 2.- Mueva un comando LCD o un carácter ASCII a W
; 3.- Llame LCD_Comando ó LCD_Caracter para enviar un comando o carácter
;      respectivamente a la pantalla LCD.

```

```

; ***** Definición de Constantes
;
; ***** Correspondientes a registros del PIC

estado      EQU    0x03      ; La dirección del registro de estado es la 0x03
portA       EQU    0x05      ; Puerto A
portB       EQU    0x06      ; Puerto B
TRISA       EQU    0x05      ; La dirección del registro TRISA en banco 1
TRISB       EQU    0x06      ; y de TRISB en banco 1

; ***** Correspondientes a bits de registros del PIC

banco       EQU    0x05      ; Bit del registro de estado correspondiente
;                               al banco de datos. En ESTADO

; ***** Correspondencias de los pines con líneas hardware

LCD         EQU    0x05      ; La LCD está en el puerto A
LCDModo     EQU      0      ; Selecciona Registro LCD
LCDRW       EQU      1      ; Lectura / Escritura LCD
LCDE        EQU      2      ; Habilita LCD

; ***** Comandos de Software para la LCD

LCDLinea1   EQU    0x80      ; Dirección comienzo línea1
LCDLinea2   EQU    0x0C      ; Dirección comienzo línea2
LCDCLR      EQU    0x01      ; Borra pantalla, cursor a casa
LCDCasa     EQU    0x02      ; Cursor a casa, DDRAM sin cambios
LCDInc      EQU    0x06      ; Modo incrementa cursor
LCDDec      EQU    0x04      ; Modo decrementa cursor
LCDOn       EQU    0x0C      ; Pantalla On
LCDOff      EQU    0x08      ; Pantalla Off
CursOn      EQU    0x0E      ; Pantalla On, cursor On
CursOff     EQU    0x0C      ; Pantalla On, cursor Off
CursBlink   EQU    0x0F      ; Pantalla On, Cursor parpadeante
LCDIzda     EQU    0x10      ; Mueve cursor a la izquierda
LCDDecha    EQU    0x14      ; Mueve cursor a la derecha
LCDFuncion  EQU    0x38      ; Inicializa registro función
LCDCGRAM    EQU    0x40      ; Dirección origen CGRAM

; ***** RUTINAS *****

ORG 0x05    ; Empezamos aquí por seguridad.
;           ; En la rutina principal del programa habrá que definir ORG 0
;           ; El resto del programa irá a continuación de estas rutinas

LCD_Port    ; *****
;           ; *** Inicializa los buffers triestado de la Puerta B como salidas ***
;           ; *** para líneas de datos de la LCD. Coloca las líneas de la ***
;           ; *** puerta A como salidas: I/D, E/S, En. ***
;           ; *****

BSF estado,banco ; Pasamos a la página 1
MOVLW 0xF8       ; En binario 1111 1000, para poner RA2, RA1 y RA0
ANDWF TRISA      ; como salidas del puerto A
CLRF TRISB       ; Y todo el puerto B como salidas de datos
BCF estado,banco ; Volvemos a la página 0, para poder tocar los puertos
BCF LCD,LCDE     ; Y deshabilitamos, por si las moscas, la LCDE
RETURN

LCD_Inic     ; *****
;           ; *** Inicialización de la LCD según el manual de datos de Optrex. ***
;           ; *** Configura funciones de LCD para pantalla DMC16207 ***
;           ; *** Produce reset por software, borra la memoria y ***
;           ; *** activa la pantalla. ***
;           ; *****

```

```

MOVLW LCDFuncion      ; Carga W con orden inicia LCD
CALL LCD_Comando      ; y lo envía a la LCD
CALL Pausa_5ms         ; ... y espera
MOVLW LCDFuncion      ; Repite la operación.
CALL LCD_Comando
CALL Pausa_5ms

; Si desea otra configuración de pantalla habrá de cambiar la siguiente
; tanda de órdenes. Aquí encendemos la LCD, quitamos el cursor
; y borramos
; la pantalla, provocando que con cada carácter el cursor avance hacia la
; derecha.

BCF LCD,LCDModo        ; Entramos en modo registro
MOVWF portB            ; y envía W a LCD en la puerta B
CALL CLD_Chequea       ; Explora la bandera de ocupado
BSF LCD,LCDModo        ; Entra en modo ASCII
GOTO LCD_Habilita      ; Envía carácter ASCII

```

LCD_Comando

```

; *****
; *** Carga W con una constante software LCD de la tabla anterior ***
; *** y saca el comando a la LCD, pulsando la línea de habilitación ***
; *** con LCD_Habilita para completar el comando ***
; *****

BCF LCD,LCDModo        ; Entra en modo registro
MOVWF portB            ; y envía el comando a la puerta B
CALL LCD_Chequea       ; Chequea la bandera de ocupado
GOTO LCD_Habilita      ; Y envía el comando

```

LCD_Character

```

; *****
; *** Carga W con el código del carácter ASCII para enviarlo a la ***
; *** LCD. Después activa la LCD con LCD_Habilita para ***
; *** completar el envío ***
; *****

BCF LCD,LCDModo        ; Entra en modo registro
MOVWF portB            ; y envía W a la LCD en la puerta B
CALL LCD_Chequea       ; explora la bandera de LCD ocupado
BSF LCD,LCDModo        ; Entra en modo ASCII
GOTO LCD_Habilita      ; Y envía en carácter ASCII

```

LCD_Chequea

```

; *****
; *** Explora el estado de la bandera Busy (ocupado) de la LCD ***
; *** Y espera que termine cualquier comando previo antes de ***
; *** volver a la rutina que le llamó ***
; *****

```

```

BSF LCD,LCDModo        ; Coloca la LCD en modo Lectura
BSF estado,banco       ; Coloca la página 1 de registros
MOVLW 0xFF             ; Configura como entrada la puerta B
MOVWF TRISB            ; cambiando el trisB
BCF estado,banco       ; Vuelve a la página 0 para leer el puerto B
BSF LCD,LCDE           ; Habilita la LCDE
NOP                    ; Pausa para a 8 MHz esperar la estabilidad
                        ; de salidas LCD

Bucle BTFSC portB,7     ; Explora el bit de ocupado LCD y
GOTO Bucle             ; espera a que valga 1 (si es 0, está ocupado).
BCF LCD,LCDE           ; Deshabilita la LCD
BCF estado,banco       ; Coloca la página 1 de registros
CLRF TRISB             ; Coloca de nuevo el puerto B
                        ; como de todo salidas
BCF estado,banco       ; Y regresa a la página 0
BCF LCD,LCDModo        ; Pone la LCD en modo escritura
RETURN                 ; Aquí la LCD ya está libre

```

```

LCD_Habilita
, *****
, *** Envía un pulso de habilitación a la LCD de 500 ns para completar ***
, *** la operación de escribir un registro o un carácter. ***
, *** la instrucción NOP sólo es necesaria para procesadores de una ***
, *** velocidad mayor de 8 Mhz. Para procesadores a más de 16 MHz, ***
, *** añadir un segundo NOP ***
, *****
,
    BSF LCD, LCDE      ; Pone a 1 la línea Enable (habilita)
    NOP                ; y espera 1 ciclo (250 ns a 8 MHz)
    BCF LCD,LCDE       ; Pone a 0 la línea Enable (deshabilita)
    RETURN

```

Si usted pasa el corrector a la librería observará que le dará 3 errores. Dos de ellos están en la rutina Pausa_5ms, inexistente. Esta rutina depende grandemente del micro, la implementación del mismo, y la velocidad de su reloj, por lo que, de emplear la librería, debería crearla usted mismo en su programa. El otro indica que falta la directiva END. No la ponga. Las librerías no deben acabar con un END. En el siguiente punto veremos una sencilla aplicación para utilizar esta librería.

5.3.7 Uso de una librería: LCD.LIB

El uso de una librería es bien sencillo si conocemos sus variables y sus rutinas internas. Aprovecharemos la creada en el apartado anterior para crear un pequeño programa que sitúe en la LCD la palabra PIC.

Pocas cosas debemos hacer resaltar para un paso tan sencillo, pero es importante saber que las rutinas del programa **LCDPIC.asm** se situarán a continuación de las de la librería. Como no puede predecir, a priori, en qué dirección de memoria acabará la librería (y comprobarlo a mano es pesado) no se debe comenzar el mismo como habitualmente lo hacemos (con una directiva ORG). Sin embargo sí debemos emplear una para colocar el vector de reset, pero esta irá al final del programa, para no interferir.

La rutina que ejecuta la pausa de 5 ms está basada en la que empleamos en parpadeo.asm.

En cualquier caso hemos querido hacer notar el uso de dos directivas más de ensamblador y que, curiosamente, no hemos encontrado en bibliografías distintas de las suministradas por el fabricante. Estas son #DEFINE y macro.

#DEFINE es empleado para crear sustituciones dentro del texto del programa que lo simplifiquen. Nosotros hemos pensado que un cambio de banco es más evidente y comprensible dentro del programa si se escribe como BANCOx (siendo x el número de banco) que con la instrucción completa (BCF estado,banco). La forma correcta es **#DEFINE NOMBRE TEXTO**, con lo que, cada vez que el compilador encuentre la orden NOMBRE, la sustituirá por el texto. El problema que se nos plantea es que, si bien es más flexible que la directiva EQU, puesto que esta sólo nos permitía asignar un valor, sólo se nos permite con **#DEFINE** una línea de texto, y esta debe ser fija.

Este problema se soluciona mediante **macro**. Esta directiva tiene la siguiente forma:

```
NOMBRE macro ARGUMENTO1, ARGUMENTO2, ETC
        TEXTO
        TEXTO...
endm
```

De este modo NOMBRE será sustituido como comando por la secuencia completa definida tras macro hasta endm, y los sucesivos argumentos serán, a su vez, sustituidos dentro del texto.

En nuestro ejemplo se repetía por tres veces la escritura de un carácter, cada vez distinto, y sólo se requerían dos líneas para cada una, por lo que no merecía la pena crear una rutina para simplificarlo. Fue en cada caso sustituida por una única línea del tipo PON_ASCII 0x40, que sitúa en la LCD el carácter 0x40. Lo hicimos como sigue:

```
PON_ASCII macro ASCII
        MOVLW ASCII
        CALL LCD_Character
Endm
```

Otra directiva que ayuda mucho para determinados programas es **if**. Supongamos, por ejemplo, que nuestro programa debe estar diseñado para funcionar bajo dos micros, uno a 2 MHz y otro a 4 MHz. ¿Cómo solucionaremos entonces el problema de la necesaria pausa de 5 milisegundos? Podemos sacar una copia modificada del programa o hacerlo mediante un if. Tenga en cuenta que, en este caso, lo empleamos para que

usted vea su uso, y, por eso, parece un poco forzado. Su verdadera utilidad se encontrará a la hora de crear librerías más o menos universales. Abra, como curiosidad, cualquiera de las librerías *.inc que se suministran junto al MPLAB y lo comprobará.

Su forma de uso es:

```

IF NOMBRE OPERADOR VALOR
    COMANDOS1
ELSE
    COMANDOS2
ENDIF

```

En donde nombre será una etiqueta definida previamente, el operador será = (igual), >=, <=, >, <, != (distinto). COMANDOS1 se ejecutará si se cumple NOMBRE OPERADOR VALOR, y COMANDOS2 se ejecutará si no se cumple.

Las directivas **ifndef nombre** y **ifndef nombre** funcionan de idéntica manera, pero en caso de que nombre haya sido definido o no, respectivamente.

Listado del programa

```
include <LCD.LIB>
```

```
; Definición de registros
```

```

estado EQU 0X03 ; La dirección del registro de estado es la 0x03
intcon EQU 0x0B ; Registro controlador de interrupciones
opcion EQU 0x81 ; Registro OPTION
tmr0 EQU 0x01 ; Registro del Timer0 (TMR0)

```

```
; Definición de bits
```

```

banco EQU 0X05 ; Bit del registro de estado correspondiente al banco de datos.
                ;En ESTADO
Z EQU 0X02 ; Bit indicador de que el registro W está a cero. En ESTADO
t0if EQU 0x02 ; Bit de INTCON que indica que se produjo interrupción
                ; por desbordamiento del timer0
t0ie EQU 0x05 ; Bit de INTCON que habilita o no la interrupción
                ; por desbordamiento del timer0

```

```
; Definición de constantes
```

```

w EQU 0 ; Destino de operación = w
f EQU 1 ; Destino de operación = registro

```

```
; Definición de variables
```

```
contador EQU 0X0C ; Contador
```

```
; Definiciones para el ensamblador
```

```

#DEFINE BANCO0 BCF estado,banco      ; Sirve para situarse en banco 0
#DEFINE BANCO1 BSF estado,banco      ; Sirve para situarse en banco 1

; Definición de macros

PON_ASCII macro ASCII
    MOVLW ASCII
    CALL LCD_Caracter
endm

, ***** CUERPO DEL PROGRAMA *****
,
inic      ; Preparamos el timer para la pausa, como en parpadeo.asm
    BSF estado,banco      ; Seleccionamos el banco 1
    MOVLW 0x06            ; En binario 0000 0101
    MOVWF opción          ; Ponemos el predivisor a 64
    *****
,
, *** OPTION.7 = 0 Resistencias de polarización deshabilitadas ***
, *** OPTION.6 = 0 Interrupción externa por flanco bajada (no se usa) ***
, *** OPTION.5 = 0 Fuente de reloj interna ***
, *** OPTION.4 = 0 Flanco de señal externa (no lo usamos) ***
, *** OPTION.3 = 0 Divisor asignado al TMR0 ***
, *** OPTION.2 = 1 Opción.1= 0 Opción.1= 0 División por 64 ***
, *****
    BANCO0                ; Y volvemos al banco 0

; **** Comenzamos con la LCD

    CALL LCD_Port          ; Inicializa los puertos, para acoplarlos
                           ; al diseño especificado de la LCD.
    CALL LCD_Inic          ; Inicializa los valores de la LCD y
                           ; la enciende tal cuál la necesitamos:
                           ; Resetea la LCD, borra la memoria y activa la pantalla
    PON_ASCII 0x50         ; Carácter ASCII de la P mayúscula (80 decimal)
    PON_ASCII 0x49         ; Carácter ASCII de la I mayúscula (73 decimal)
    PON_ASCII 0x43         ; Carácter ASCII de la C mayúscula (67 decimal)

Pausa_5ms
    if velocidad == 4      ; Para un micro a 4 MHz
        MOVLW 0xB2        ; Cargamos el timer con 78 decimal (0xB2)
    else
        MOVLW 0xD8        ; Si no deducimos que funciona a 2 MHz
                           ; y cargamos el timer con 39 (la mitad)
    endif
    MOVWF tmr0
    MOVLW 0xA0             ; 1010 0000 en binario
    MOVWF intcon           ; Habilitamos GIE y TOIE (interrupción del timer0)
                           ; Deshabilitamos TOIF (bandera de salto producido)
espera BTFSS intcon,t0if   ; Esperamos a que la bandera se active
    GOTO espera
    RETURN

, ***** RSI: Rutina de servicio de interrupción *****
, ***** Salta al desbordarse el TMR0, cada 5 ms *****
,
RSI      RETURN           ; Queda deshabilitada la interrupción mientras no sea necesaria
                           ; No se borra la bandera GIE ni la TOIF

ORG 0X00                ; Cubrimos el vector de reset

    GOTO inic            ; Saltamos a la primera dirección tras el vector de interrupción

ORG 0x04                ; Cubrimos el vector de interrupción

    GOTO RSI             ; Y saltamos a la rutina de servicio de interrupción

END

```

Pruebe usted con otras variantes, como, por ejemplo, hacer parpadear PIC, o desplazarlo por la pantalla.

5.3.8 El Watchdog

Conocer la existencia y el manejo del **Watchdog** es fundamental, en muchos casos. Esta herramienta es un contador de 8 bits que, al desbordarse, produce el reseteo del micro. La única forma de evitar este reseteo es, por tanto, borrarlo por software cada cierto tiempo con la instrucción **CLRWDT**, que devuelve su valor a 0. Su velocidad normal es la de una cuenta por cada ciclo de instrucción, aunque puede asignársele el preescaler para reducir su frecuencia (vea el registro **option** para aprender el uso de este divisor).

Su utilización es opcional, y se activa (o no) durante el proceso de grabación del micro. Todos los grabadores que conocemos y hemos usado tienen en sus menús o funciones la opción específica.

Sirve para evitar posibles problemas de grabación no controlados o controlables por la razón que sea, como, por ejemplo, bucles infinitos, esperas exageradamente largas de alguna determinada entrada, etc., y es especialmente interesante en ambientes con mucho ruido, ya que éste puede afectar al PC, mandándolo a ejecutar una línea al azar.

5.3.9 Notas para el profesor sobre la elaboración de estos programas

Todos los programas de este apartado están sacados de los libros "µcontroladores PIC: Teoría y Práctica", y "µControladores PIC, la solución en un chip", de J. M^a Angulo. Su documentación, muchas veces insuficiente, bajo nuestro criterio, ha sido ampliada, en todos los casos. Han sido adaptados al entorno MPLAB (pues estaban orientados a programas de MS-DOS) y simulados para comprobar que carecían de errores, así como corregidos, cuando era pertinente. Los comentarios sobre los mismos, lejos de ser sacados de los mencionados libros, son basados en sus respectivas pruebas. Todos los programas tienen modificaciones sobre los originales para demostrar nuestra

comprensión de los mismos (por ejemplo, sacar un resultado por una serie de LEDS, en lugar de guardarlos en un registro). Las excepciones son los dos primeros, **suma.asm** y **suma2.asm**, por razones evidentes (tienen muy pocas líneas y son básicos) y **LCD.LIB**, en la que, si bien si fueron corregidos determinados errores, no tenía sentido modificar el programa, ya que gestiona una LCD. Para demostrar nuestra comprensión de la misma se creó **LCDPIC.asm**, inexistente en nuestra bibliografía, y con la que partimos de 0 para su elaboración.

6. EL COMPILADOR DE C

6.1 Introducción

Prosiguiendo con nuestro estudio sobre las herramientas de programación para la gama media hemos buscado distintas herramientas gratuitas que ofrezcan soporte para el lenguaje C. Microchip se limita a ofrecer tan solo los MPLAB-C17 y MPLAB-C18, limitadas a la gama alta (PIC17CXX y PIC18CXX, respectivamente).

Hemos encontrado varias, pero, entre ellas, quepa tal vez destacar la que aquí estudiaremos, el PIC-C Compiler, al facilitar, además del programa, el código fuente del mismo para posibles posteriores revisiones. Funciona en entorno MS-DOS y es del año 95.

Si bien, como explicamos en su momento, el entorno MPLAB, es del tipo *container*, y carece de lenguajes en su código, dejando la tarea de compilación para otros programas, requiere que estos creen, a su fin, determinados ficheros para atender sus propias necesidades, como la de mostrar errores del programa. Nuestro C es anterior a la creación del MPLAB, y, en consecuencia, no sigue esos criterios, con lo que le resultará imposible incorporarlo.

Su uso es verdaderamente sencillo. Bastará con llamar al programa, que se encuentra en la carpeta Source, seguido de la ruta y nombre del programa a compilar, como, por ejemplo, **pic_cc prog.c**. La salida del programa es un fichero en ensamblador, el cuál, en este caso, se denominaría **prog.s**.

6.2 El primer programa en C

Veamos un sencillo programa de los adjuntados en la carpeta **TESTS**, el **test0.c**, el cuál ha sido transcrito sin modificar, pero añadiéndole comentarios:

```
char aa, bb, cc, dd;    ‘ Creamos cuatro variables de tipo char (caracteres,
tamaño 1 byte, lo usual)

main()                  ‘ Proceso principal
{
    char kk;             ‘ Nuevo char, utilizable solo en main
```

```

aa=3;          ' Damos un par de valores, y calculamos una
multiplicación
bb=5;
cc=aa*bb;

kk=cc+1;
dd=myfunc( kk );    ' Llamada a la función myfunc, con el parámetro kk.
                    ' Su valor de vuelta es asignado a la variable dd

}

myfunc( t )
char t;
{
    char c;

    c=t+4;          ' Calculamos la función deseada
    return( c );    ' Devolvemos el valor c.

}

```

No es nuestro deseo tampoco hacer un estudio profundo del lenguaje c, del que es abundante la bibliografía y con el que pensamos que se hallará familiarizado cualquier lector que se aventure en estas páginas, pero sí hablaremos de sus ventajas, sus desventajas y hasta dónde llega su desarrollo como lenguaje (qué encontraremos en sus librerías).

Por lo pronto, hete aquí la comparación directa con el resultado de la compilación, **test0.s**. Es fácil apreciar lo simple, escueto, legible y fácil de modificar que resulta el código en c.

```

;Small C PIC16C84;
; Coder (1.0 2/10/95)
; Version 0.002

; Front End (PIC Ver 1.0 2/19/95)

include '16c84.h'

; *****code segment cseg*****

org _CSEG

; Begin Function

main_

```

```

sub _stackptr, #1
mov _primary, #3
mov aa_      , _primary
mov _primary, #5
mov bb_      , _primary
mov _primary, aa_
call _push_
mov _primary, bb_
call _pop_
call _mul_
mov cc_      , _primary
mov _primary, #0
add _primary, _stackptr
call _push_
mov _primary, cc_
call _push_
mov _primary, #1
call _pop_
add _primary, _secondary
call _pop_
call _putstk_
mov _primary, #0
add _primary, _stackptr
call _indr_
call _push_
;:(# args passed) mov W, #1
call myfunc_
add _stackptr, #1
mov dd_      , _primary
_1_
add _stackptr, #1
ret

```

; Begin Function

```

myfunc_
sub _stackptr, #1
mov _primary, #0
add _primary, _stackptr
call _push_
mov _primary, #2
add _primary, _stackptr
call _indr_
call _push_
mov _primary, #4
call _pop_
add _primary, _secondary
call _pop_
call _putstk_
mov _primary, #0
add _primary, _stackptr
call _indr_
jmp _2_
_2_
add _stackptr, #1
ret

```

```

;*****need mult/div standard library*****

include 'mllibpic.h'

; *****data segment dseg*****

org _DSEG

aa_    ds    1
bb_    ds    1
cc_    ds    1
dd_    ds    1

;0 error(s) in compilation
; literal pool:0
; global pool:112
; Macro pool:51
end

```

Pocos comentarios quedan al respecto. De hecho es prácticamente indescifrable el resultado. Sin embargo, como siempre, una buena programación directa en ensamblador reducirá código, y será de ejecución más rápida. Muchas veces, incluso, resultará imprescindible para determinados módulos. Sin embargo, también necesitará un mayor tiempo de estudio, desarrollo e implementación que en el caso del c.

6.3 ¿ Qué podemos usar del c convencional?

Pues, por ejemplo, como apreciaremos del ejemplo **test2.c**, los **punteros**:

```
char val;
```

```
main()
{
```

```
    char aa,bb,cc,dd;
```

```
    load( &aa, &bb, &cc, &dd); ' El carácter & referencia a la dirección de la variable
    val=aa+bb+cc+dd;
```

```
}
```

```
load( a, b, c, d )
```

```
char *a, *b, *c, *d;
```

```
que,
```

```
{
```

```
    *a = 1;
```

```
    *b = 2;
```

```
    *c = 3;
```

```
    *d = 4;
```

```
}
```

‘ Y aquí a, b, c y d son direcciones de memoria, con lo

‘ para aludir a sus valores, las precedemos de *.

O, como en el **test1.c**, disfrutaremos de las ventajas de un bucle **while**, o de los operadores lógicos, como *igual*, *distinto*, *>=*, etc.

```
#define TRUE 1
#define FALSE 0

char ad, d1, d2, d3;

main()
{
    char cnt;
    while(TRUE) {
        ad=GetAD();
        cnt=0;
        while (ad>=100) {
            ad=ad-100;
            cnt++;
        }
        d1=cnt;
        cnt=0;
        while (ad>=10) {
            ad=ad-10;
            cnt++;
        }
        d2=cnt;
        cnt=0;
        while (ad>0) {
            ad=ad-1;
            cnt++;
        }
        d3=cnt;
    }
}

GetAD()
{
    char v;
    v=0xff;
    return(v);
}
```

O, como en **test3.c**, de la herramienta **switch**, para escoger entre distintos valores de una variable.

```
char z;

main()
{
    char i;
    for (i=0; i<5; i++) {
        switch(i) {
            case 0: z=f1();
                    break;
            case 1: z=f2();
```

```

        break;
    case 2: z=f3();
        break;
    case 3: z=f4();
        break;
    case 4: z=f5();
        break;
    }
}
}

f1() { return(1); }
f2() { return(2); }
f3() { return(3); }
f4() { return(4); }
f5() { return(5); }

```

O, como en el caso de **test4.c**, de la simple sentencia condicional **if**:

```

char z;

main()
{
    char i;
    i=0;
    while(i<5) {
        if (i==0)
            z=f1();
        if (i==1)
            z=f2();
        if (i==2)
            z=f3();
        if (i==3)
            z=f4();
        if (i==4)
            z=f5();
        i++;
    }
}

f1() { return(1); }
f2() { return(2); }
f3() { return(3); }
f4() { return(4); }
f5() { return(5); }

```

O, como en **test6.c**, del bucle **for**, y de las **cadenas de caracteres** (la sentencia **GetChar** será explicada en el siguiente apartado):

```

/* example of including static strings in compiler */

char a, b, c, d;

main() {

```

```

a="hello";
b="goodbye";
for (d=0; d<7; d++)
    c=GetChar(b,d);    /* index through the string */
for (d=0; d<5; d++)    /* ditto */
    c=GetChar(a,d);
}

#include "getchar.c"

```

O, incluso, podrá escribir directamente en los puertos, como en **test7.c**. En este caso, hemos vuelto a comentarlo para la más sencilla comprensión:

```

/* test i/o port modes */

#include "io.c"

char a, b;

main()
{
    SetP_A(0x03);          ' Se establece como puerto A la dirección 0x03
    SetP_B(0x0f);          ' Se establece como puerto B la dirección 0x0F
    a=RdPortA();           ' El valor del puerto A se guarda en la variable a
    b=RdPortB();           ' El valor del puerto B se guarda en la variable b
    WrPortA(0x1);          ' Escribimos en el puerto A el valor 0x01
    WrPortB(0x55);         ' Escribimos en el puerto B el valor 0x55
}

```

En realidad es posible asignar como puerto A o B cualquier dirección física de memoria de datos, aunque esta no sea propiamente un puerto. Esto puede ser útil para manejar directamente registros como STATUS o INTCON.

Pero dejamos para el último apartado lo no descrito en los ejemplos, es decir, aquellas funciones incluidas en las librerías.

6.4 Librerías y funciones

6.4.1 La librería GETCHAR

Esta librería sólo contiene la función GetChar (cadena, índice), que, como pudimos ver en el ejemplo **test6.c**, coge de una cadena de caracteres (definida, por ejemplo, como a = "cadena") el carácter situado en la posición índice. Por ejemplo, b = GetChar (a,2) nos daría b = "d", ya que se empieza a contar desde 0.

6.4.2 La librería IO

Esta librería se encarga de la gestión de puertos, y contiene varias funciones:

SetP_A(valor) asigna como dirección del puerto A valor.

SetP_B(valor) asigna como dirección del puerto B valor.

RdPortA() y **RdPortB()** devuelven el valor leído en ambos puertos (siempre habrá que definir sus direcciones antes, o tendremos resultados imprevistos).

WrPortA(valor) y **WrPortB(valor)**, respectivamente, enviarán valor a las salidas de los puertos A y B. Como en el caso de las anteriores, deben estar previamente definidas sus direcciones para no sufrir imprevistos.

6.4.3 Librería EE_READ

Contiene la función **ee_read (addr)**, que leerá un valor de la memoria EEPROM interna del microprocesador, sito en la dirección addr.

6.4.4 Librería EE_WRITE

Contiene la función **ee_write (addr)**, que escribirá un valor de la memoria EEPROM interna del microprocesador, sito en la dirección addr.

Estas dos últimas funciones no han sido detalladas en el apartado de ensamblador por no aparecer en la bibliografía asociada. Consultando el Databook en el apartado correspondiente al PIC16C84 se nos mostrará la manera apropiada de hacerlo, a través de los registros EECON1 (dirección 0x08 en la página 0), EECON2 (dirección 0x09 en la página 0), EEDATA (dirección 0x08 en la página 1, 0x88 como dirección absoluta) y EEADR (dirección 0x09 en la página 1, 0x89 como dirección absoluta).

6.4.5 También conviene saber

Este C asume la directiva **#asm**, a continuación de la cuál seguirá un trozo de código en ensamblador. Para regresar a la programación en C deberemos incluir, tras la secuencia de código, **#endasm**. Puede ver ejemplos editando cualquiera de las librerías, sitas en el directorio **Pic_lib**. Podrá, a su vez, hacer referencia a los argumentos de la rutina en la que esté el código mediante #0, #1 ... #n, siendo el número la posición del argumento en la llamada a la rutina, comenzando por 0.

Las definiciones de constantes en ensamblador están en **Pic_rt\16c84**. **_portA** y **_portB**, por ejemplo, referencian a esos puertos, así como **eedata**, **eeaddr**, **eecon1** y **eecon2**. También contiene macros como **_push_**, **_pop_**, **_swap_**, **_swaps_**, o **_indr_**, que son usados internamente por el compilador. Puesto que manejan los cuatro registros reservados para el C (estos son **_primary**, **_secondary**, **_temp** y **_stackptr**) no conviene que los use si no los comprende muy bien, ya que podría alterar el buen funcionamiento de su programa. Las direcciones de esto cuatro registros son 0x2f, 0x2e, 0x2c y 0x2b. Procure no usarlos en su código en ensamblador.

Es posible generar una rutina específica de interrupción. El paso a seguir sería editar el fichero **Pic_rt\16c84** y alterar la línea:

```
interrupt jmp $      ; set to your interrupt routine
```

Cambiándola por:

```
interrupt call RSI_  
            jmp $
```

A partir de ese momento bastará con que cree una rutina llamada **RSI()** para que sea interpretada como de servicio de interrupción y ejecutada

7. EL PROGRAMADOR.

7.1 Introducción.

Hemos encontrado una cantidad de programadores importante dispersos por la web y la bibliografía, con diversas complejidades y prestaciones diferentes. Por ello, hemos elegido el programador más sencillo posible pero que mantuviera un nivel de prestaciones alto; con un puñado de componentes poco costosos nuestro programador es capaz de manejar la mayoría de los microcontroladores PIC actuales con una gran facilidad, excluyendo algunas excepciones muy raras.

Aunque no se le puede dar el calificativo de programador de producción, nuestro montaje cumple suficientemente las especificaciones de MICROCHIP para entrar dentro de la categoría que este fabricante llama programadores de desarrollo. En el momento de escribir estas líneas, se pueden programar todos los circuitos PIC de la lista que se proporciona en la Tabla 7.1 pero, teniendo en cuenta su esquema y la evolución constante de su software de control, esta lista es susceptible de evolucionar a medida que se comercializan nuevos circuitos, es más probablemente en el momento en que se lea este texto esta lista ya habrá aumentado. Como muchos de sus homólogos, nuestro montaje se conecta al puerto paralelo del PC, pero no es necesario disponer del último Pentium III a 450 MHz, porque un viejo AT 286 bajo DOS puede ser suficiente, sin tener que renunciar por ello a una excelente facilidad de uso.

0: PIC16F83	18: PIC12CE674	36: PIC16C64	54: PIC16C710
1: PIC16CR83	19: PIC14000	37: PIC16C64A	55: PIC16C71
2: PIC16C84	20: PIC16C54	38: PIC16C64B	56: PIC16C711
3: PIC16F84	21: PIC16C55	39: PIC16CR64	57: PIC16C72
4: PIC16F84A	22: PIC16C56	40: PIC16C65	58: PIC16C73
5: PIC16CR84	23: PIC16C57	41: PIC16C65A	59: PIC16C73A
6: PIC16F873	24: PIC16C57C	42: PIC16C65B	60: PIC16C73B
7: PIC16F874	25: PIC16C58	43: PIC16C66	61: PIC16C74
8: PIC16F876	26: PIC16C554	44: PIC16C67	62: PIC16C74A
9: PIC16F877	27: PIC16C556	45: PIC16C620	63: PIC16C74B
10: PIC12C508	28: PIC16C558	46: PIC16C620A	64: PIC16C76
11: PIC12C508A	29: PIC16C61	47: PIC16C621	65: PIC16C77
12: PIC12C509	30: PIC16C62	48: PIC16C621A	66: PIC16C923
13: PIC12C509A	31: PIC16C62A	49: PIC16C622	67: PIC16C924
14: PIC16C505	32: PIC16C62B	50: PIC16C622B	68: PIC16C642
15: PIC12C671	33: PIC16CR62	51: PIC16CE623	69: PIC16C662
16: PIC12CE673	34: PIC16C63	52: PIC16CE624	70: PIC16C715
17: PIC12C672	35: PIC16C63A	53: PIC16CE625	

Tabla 7.1: Lista de PICs programables.

Nuestro programador necesita, además, una alimentación que puede ser continua o alterna, comprendida entre 12 y 30V y que no es preciso que esté estabilizada. Un enchufe de la red o cualquier alimentación de laboratorio puede ser apropiada, sobre todo teniendo en cuenta que el consumo de corriente es inferior a 100 mA. Como los programadores profesionales, nuestro montaje es capaz, naturalmente, de leer, verificar, programar y comparar los PIC sin ninguna restricción, lo mismo que puede leer y programar sus fusibles de configuración. Por supuesto, también puede borrar los circuitos provistos de memoria de tipo EEPROM y permitir el acceso a la memoria de datos de los circuitos dotados de éstas cuando se realizan, asimismo, con tecnología EEPROM. Ello nos lleva a decir que es verdaderamente completo, por lo que puede satisfacer tanto a un desarrollador ocasional como a un usuario intensivo de circuitos PIC.

7.2 De la programación paralela a la programación serie

Mientras que las memorias mas conocidas, las UVPROM, desde la “vieja” 2716 a las más recientes 27512 o 271024, se programan en paralelo (es decir, aplicando simultáneamente a la memoria la dirección a programar y el dato a colocar en esa dirección), numerosas memorias recientes contenidas en los microcontroladores se programan en serie. En el caso de la programación serie sólo se precisan tres líneas de señal, frente a las más de diez necesarias en la programación en paralelo (hasta 26 incluso para una 271024, que es una memoria de 1 megaoctetos y que necesita, por tanto, 8 líneas de datos y 17 líneas de direcciones). La ganancia de espacio es evidente y la simplificación del diseño de circuito impreso que resulta de ello es también enorme.

Los microcontroladores PIC de MICROCHIP están todos provistos de memoria de acceso serie y, como ciertos encapsulados no tienen más que 8 patillas, éstas se reparten según los modos de funcionamiento. La Tabla 7.2 muestra así, en el caso del 12C508 (que es un encapsulado de 8 patillas elegido a título de ejemplo), cómo se realiza la programación respecto a las conexiones del circuito. En la práctica, tres patillas del encapsulado cambian momentáneamente de función durante la fase de programación para dar acceso a la memoria de programa interna; este cambio se desencadena simplemente aplicando la tensión “alta” de programación en la patilla VPP.

Nº DE PATILLA	FUNCIÓN EN MODO NORMAL	FUNCIÓN EN MODO PROGRAMACIÓN
1 (VDD)	Alimentación positiva	Alimentación positiva
2	GP5 o OSC1 o CLKIN	GP5 o OSC1 o CLKIN
3	GP4 o OSC2	GP4 o OSC2
4	GP3 o MCLR	VPP
5	GP2 o TOCKI	GP2 o TOCKI
6	GP1	Reloj de programación
7	GP0	Datos de programación
8 (VSS)	Masa	Masa

Tabla 7.2: Funciones de las patillas del PIC12C508

Aunque las memorias de PIC se programan en serie, nuestro programador se conecta al puerto paralelo del PC. En efecto, por una parte este puerto se puede controlar muy fácilmente por software y, por otra parte, suministra niveles TTL directamente utilizables. Además, debemos disponer de algunas líneas de control para conmutar las diversas alimentaciones del microcontrolador en el curso de la programación, lo cual es mucho más fácil de realizar en un puerto paralelo que en un puerto serie. El esquema completo de nuestro programador se presenta en la Figura 7.1 y vamos a comprobar que se puede analizar fácilmente. Las señales de un puerto paralelo son señales TTL, y por esto, son bastante “maltratadas” en su viaje por los cables de unión un poco largos o de mala calidad. Por esta razón, se restauran un poco por medio de los inversores contenidos en el circuito IC1. Además, como este circuito dispone de salidas a colector abierto, permite controlar fácilmente los tres transistores T1, T2 y T3 que van a continuación. T1 y T2 permiten aplicar la tensión alta de programación VPP a las patillas adecuadas del zócalo universal del programador; patillas que difieren según el tipo de PIC programado. No se puede esperar, en efecto, disponer de la misma asignación de pines en un encapsulado DIL de 8 patillas, que en un DIL de 40. En cuanto al transistor T3, gobierna la tensión normal de alimentación VDD, aplicada igualmente al zócalo universal. Éste permite no alimentar el circuito a programar más que cuando es verdaderamente necesario acceder a él, evitando de esta forma cualquier problema durante su inserción o extracción del zócalo de programación.

Para indicar la aplicación o no de estas tensiones, se utilizan dos LEDs rojos, D1 y D2, gobernados por las dos tensiones VPP. En cuanto al diodo D3, se enciende simplemente cuando el programador está bajo tensión, con el fin de señalar el buen funcionamiento de la alimentación.

diferentes alimentaciones, de la línea de datos y de la línea de reloj, se realiza según la asignación de pines de los diferentes circuitos.

La alimentación del programador es muy simple pero muy tolerante. Dos tensiones estabilizadas son, en efecto, necesarias: 5V para la alimentación normal o VDD y 13V para la tensión alta de programación o VPP. IC2, que es un 78L05, se encarga de la producción de 5V, mientras que IC3, que es un 78L08, produce los 13V. Este no está, en efecto, referenciado a masa como se hace normalmente, sino a la salida de IC2, produciendo así $5 + 8 = 13V$.

Con el fin de adaptarse a cualquier fuente externa, nuestros reguladores están precedidos por un filtrado generoso y por un puente rectificador. Por esto, se puede aplicar en la entrada J1 cualquier tensión alterna comprendida entre 12 y 20V o cualquier tensión continua comprendida entre 16 y 30V, sabiendo que el consumo de corriente necesario es de aproximadamente 100 mA.

7.3 Software y utilización

Numerosos programas pueden utilizarse con nuestro programador y están disponibles en Internet. Nosotros hemos elegido el programa P16Pro de Bojan Dobaj, que encontrareis en el cdrom adjunto. Este software se adapta perfectamente a nuestro montaje. El programa se llama P16PR363.zip, de donde, una vez descomprimido aparece un cierto número de ficheros, entre ellos el ejecutable P16PRO.EXE

Antes de ejecutar su P16Pro recién instalado, comience por conectar su programador al puerto paralelo del PC. Conecte el programador a una alimentación, y verifique la presencia de 5V en la salida de IC2, y de 13V en la salida de IC3. El LED verde debe estar encendido y los LEDs rojos pueden estar encendidos o apagados según lo que hayan hecho previamente con su puerto paralelo.

Ejecute entonces el programa P16PRO. Tratándose de un programa DOS se ejecuta bajo DOS, en modo DOS si trabajáis con Windows 9X, o bien en una ventana DOS en este último caso.

Cualquiera que sea su modo de ejecución se accede a la pantalla principal reproducida en la Figura 7.2. Esta última muestra, por defecto, tres o cuatro ventanas (dependiendo del tipo de circuito PIC elegido). La ventana principal muestra el contenido de la memoria de programa, la ventana secundaria el estado de los “fusibles” de configuración del circuito y, según el caso, una tercera ventana muestra el contenido de la memoria de datos para los PIC provistos de una EEPROM de datos. Una última ventana, finalmente, resume las principales funciones accesibles de forma inmediata por medio de las teclas F1 a F10.

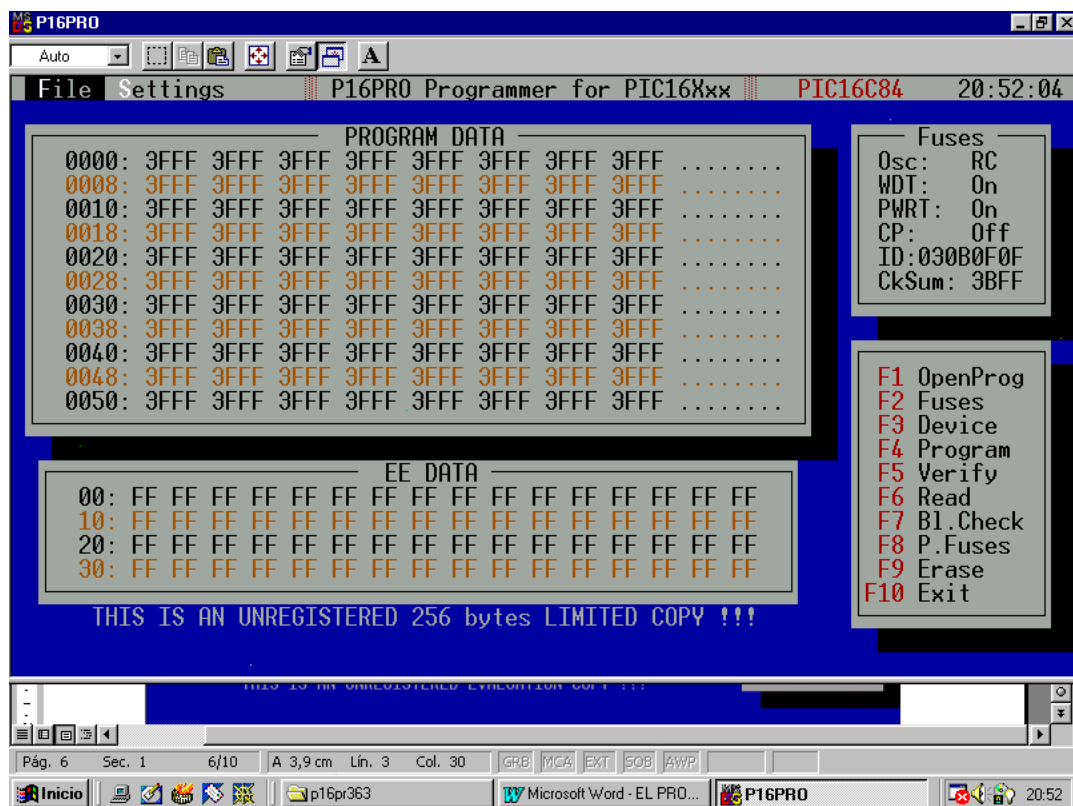


Figura 7.2: Pantalla de inicio del programador.

Dos menús desplegables accesibles en la parte superior izquierda de la pantalla, pulsando la tecla Alt, dan acceso a todo lo concerniente a los ficheros (menú FILE) ya a la configuración hardware (menú Settings). Para abrir uno de ellos, basta con pulsar la tecla ENTER cuando el menú elegido se encuentre en vídeo inverso.

La primera operación a realizar consiste en configurar los parámetros en función de su programador. Para esto, vaya al menú Settings y sitúese en Other por medio de las teclas de desplazamiento del cursor (FlechaArriba/FlechaAbajo). Entonces puede usted seleccionar el puerto paralelo al cual se encuentra conectado su montaje. Con este fin, desplace el cursor sobre el nombre del puerto apropiado y hágalo pasar de inactivo a activo por medio de la barra espaciadora. Pulse OK para validar y después vuelva al menú Settings, pero esta vez vaya a Hardware y asegúrese de que aparece una ventana idéntica a la de la Figura 7.3. En caso contrario, modifique su ventana en consecuencia (según el mismo principio que para la elección del puerto), para obtener la misma que nuestra Figura 7.3, a falta de la cual, su programador no podrá funcionar.

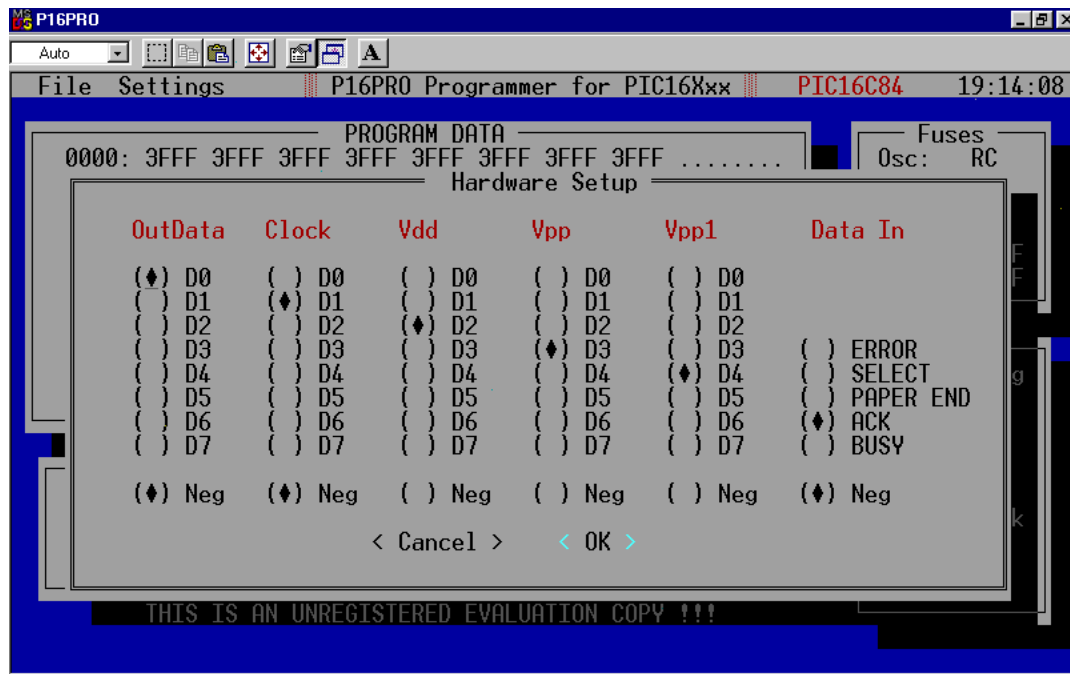


Figura 7.3: Configuración del programador.

A continuación, puede usted colocar en el zócalo un circuito PIC seleccionado de entre la lista de circuitos soportados por el programador y respetando las indicaciones de la Figura 7.4, fijaros en la muesca del zócalo para colocarlo en el sentido correcto. Seleccione la referencia de su circuito por medio de la tecla F3 y luego pruebe las diversas funciones del programador. Si usted utiliza un circuito borrable, por ejemplo, un 16C84 o un 16F84, puede incluso intentar programarlo “para ver”, ya que siempre le será posible borrarlo después. Posteriormente, se presenta un resumen de las

instrucciones de uso de este programador, cuyas principales funciones son, sin embargo, suficientemente explícitas, para que usted no encuentre ninguna dificultad en utilizarlas. Si tal fuese el caso, sepa que se encuentra una nota explicativa en inglés dentro del fichero SMANUAL.ENG creado durante la descompresión del fichero inicial.

Un fallo de funcionamiento es muy improbable, vista la simplicidad del montaje; las únicas dificultades que eventualmente podría encontrar son errores de programación o de lectura al utilizar cables de unión de más de 3m entre el PC y el programador. Si, no obstante, se produce dicho fenómeno, que se manifiesta por cambios más o menos aleatorios en los datos leídos o programados, acorte el cable de unión o utilice un cable de mejor calidad.

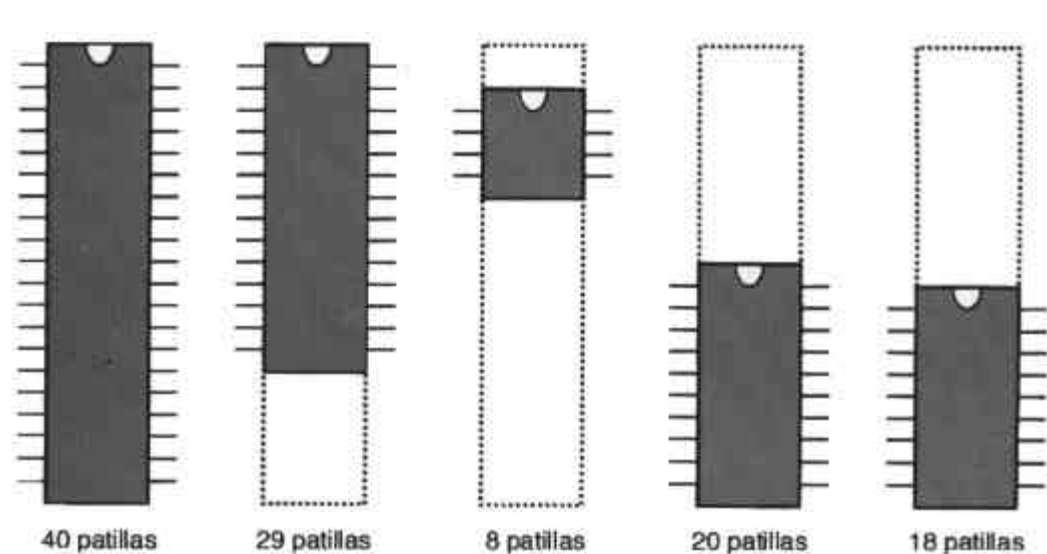


Figura 7.4: Modo de colocar los circuitos integrados en el zócalo universal en función de su tamaño.

7.4 Instrucciones de uso resumidas del programador

Las teclas F1 a F10 dan acceso directamente a las funciones siguientes:

F1: selecciona el fichero a utilizar en el directorio actual.

F2: da acceso a la ventana de definición de “fusibles” de configuración del circuito a programar, pero no efectúa su programación.

F3: permite elegir el tipo de circuito PIC a programar.

F4: programa el circuito PIC seleccionado con F3, cuyo tipo aparece en la parte superior derecha de la pantalla junto con el fichero previamente seleccionado con F1 o por medio del menú File.

F5: compara el contenido del PIC colocado en el zócalo de programación con el fichero previamente seleccionado con F1 o con el menú File.

F6: lee el contenido del PIC colocado en el zócalo de programación y lo muestra en la ventana o ventanas correspondientes.

F7: comprueba que el PIC no ha sido programado todavía.

F8: programa solamente los fusibles de configuración del PIC conforme al estado definido previamente por medio de F2; estado que se muestra permanentemente en la ventana Fuses.

F9: borra el PIC si éste es borrrable (versiones provistas de memoria EEPROM), Si el circuito seleccionado por medio de la tecla F3 no es borrrable eléctricamente, este comando se encuentra difuminado e inaccesible.

F10: permite salir del programa.

Los menús File y Settings son accesibles pulsando la tecla Alt del teclado y desplazándose después por medio de las teclas de desplazamiento del cursor (FlechaDerecha/FlechaIzquierda). Para abrirlos hay que pulsar la tecla Enter. Las opciones de menú que aparecen entonces son accesibles, o bien desplazándose por medio de las teclas FlechaAbajo/FlechaArriba, o bien directamente pulsando la letra que se encuentra resaltada (o en vídeo inverso según el modo de presentación). Por ejemplo, la opción Hardware del menú Settings, es accesible pulsando directamente H cuando se encuentra abierto dicho menú. El cierre de una ventana abierta por una de estas opciones de menú puede realizarse situando el cursor sobre Cancel para anular las

modificaciones eventualmente hechas por error, o situándose sobre OK para validar las selecciones hechas en la ventana, o bien pulsando Escape.

Cuando una ventana contiene casillas para marcar, como por ejemplo en el caso de la función que permite seleccionar el puerto paralelo, basta con situarse sobre la casilla de su elección por medio de las teclas de desplazamiento del cursor, para poder poner o quitar la marca usando la barra espaciadora.

Habiendo precisado esto, el menú File da acceso a las funciones siguientes:

Ⓡ Open Program: juega el mismo papel que la tecla F1 y permite cargar en la memoria de programa el contenido del fichero seleccionado. El software reconoce los formatos estándar del ensamblador de MICROCHIP, a saber, **INH8M o INH16**.

Ⓡ Save Program: graba el contenido de la memoria de programa en el fichero de su elección.

Ⓡ Open Data y Save Data: juegan el mismo papel que los dos comandos que acabamos de ver, pero para los circuitos que contienen memoria EEPROM de datos. En caso contrario, estos comandos están difuminados y son inaccesibles.

Ⓡ Edit Program: permite editar el contenido de la memoria de programa actual. La dirección y el contenido deseados deben ser introducidos en hexadecimal. Una vez que se ha introducido una dirección, el hecho de pulsar Enter hace pasar automáticamente a la dirección siguiente. Para salir de este modo, basta con pulsar la tecla Escape.

Ⓡ Fill Program: permite rellenar una zona de memoria con el dato de su elección. Todas las entradas se hacen en hexadecimal.

Ⓡ Edit Data y Fill Data: juegan el mismo papel que los dos comandos que acabamos de ver pero para los circuitos que contienen memoria EEPROM de datos. En caso contrario, estos comandos están difuminados y son inaccesibles.

Ⓡ Clear Buffer: repone al estado en blanco (00 o FF según el caso) toda la memoria de programa, así como la memoria EEPROM de datos para los circuitos que la contienen.

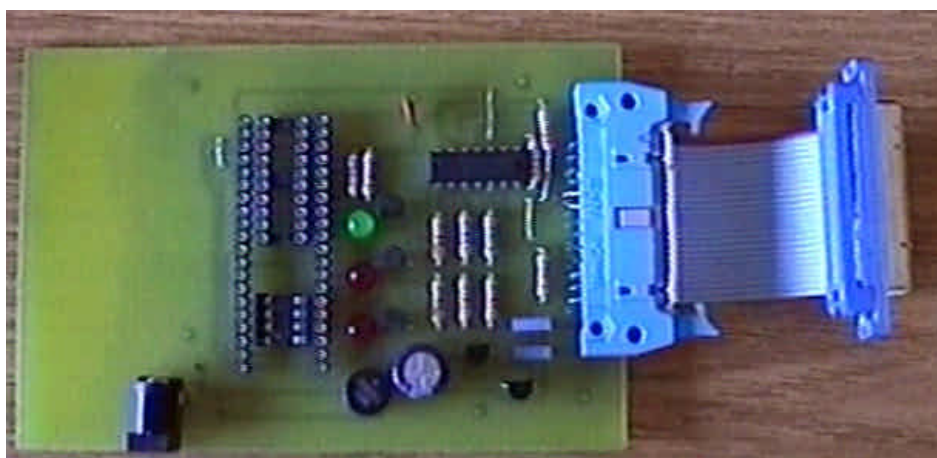
Ⓡ About: muestra el copyright del programa y su número de versión.

Ⓡ Exit: permite salir del programa, como la tecla F10.

En cuanto al menú Settings, da acceso a las funciones siguientes:

- Ⓜ Device: permite seleccionar el tipo de microcontrolador, como la tecla F3.
- Ⓜ Fuses: permite definir el estado de los fusibles, como la tecla F2.
- Ⓜ ID: permite definir el estado de la palabra de identificación contenida en los circuitos PIC. Esta palabra puede ser utilizada como suma de prueba, como número de serie o bien puede ignorarse (a su elección).
- Ⓜ Hardware: da acceso a la ventana de definición de las diferentes líneas de conexión del programador al PC. Esta ventana debe ser rellenada conforme a la que le presentamos en la Figura 7.3 y no debe modificarse o el programador no funcionará.
- Ⓜ Other: permite seleccionar el puerto paralelo utilizado, así como diversas opciones como salvar la configuración al salir del programa, las ventanas de visualización en pantalla, etc.
- Ⓜ Save: permite salvar toda la configuración actual en el fichero P16PRO.INI al salir del programa, incluido el tipo de micro seleccionado, con el fin de poder recuperarla en la siguiente ejecución del programa.

Por último, debajo de estas líneas tenéis como ha quedado nuestro programador:



8. APLICACIÓN PRÁCTICA: UN CONTADOR CONTROLADO POR INTERRUPCIÓN

La teoría desarrollada en el presente documento necesita ser apoyada con una demostración práctica por muchos motivos: el primero, personal, es el de demostrar nuestra capacidad ante nuestro tutor para llevar a cabo un diseño real, así como el buen funcionamiento del grabador desarrollado. Sin embargo pensamos que servirá a quien lo lea para comprobar in situ como se “mueve” un PIC y así como para ver un esquema hardware básico sobre el que comenzar a medrar otros posibles diseños.

Para ello hemos decidido hacer una modificación sobre el programa **cuenta.asm**, del punto 6 del apartado de programación, llamado **tablas y subrutinas**, que realizaba una cuenta cíclica de 0 a 9 sobre un 7 segmentos de cátodo común.

En este caso hemos añadido un control por interrupción simulado mediante un pulsador que, al activarse, detendrá la cuenta, y la volverá a cero cuando de suelte el botón.

El problema aparecido es el hecho de que el siete segmentos implementado en esa ocasión controlaba mediante el bit 7 del puerto b el punto decimal, absolutamente innecesario para esta experiencia, mientras que ocupaba el bit 0 del mismo puerto para el *segmento a* del 7 segmentos, que es el único pin disponible para controlar directamente una interrupción externa. Se ha resuelto eliminando el punto decimal y desplazando un bit cada uno de los otros segmento, con lo que observará la tabla de los mismos cambiada.

No cabe destacar más sobre el programa, ya que le suponemos con los conocimientos necesarios, después de leído este manual, como para entender su código, que adjuntamos a continuación. Lo hemos denominado **display.asm**.

```

; *****
; Programa Display.asm
; Contamos hasta 0x5f.
; El valor del contador se visualizará en 8 diodos LED conectados al puerto B
; a partir de la patilla 1, sin gestión de punto decimal
; Preparado para PIC16F84
; Velocidad del reloj: 4 MHz
; Ciclo de instrucción: 1 MHz = 1 microsegundo
; Interrupciones: A través de PB.0, para detener y recomenzar la cuenta.
; Perro guardián: Desactivado
; Tipo de Reloj: XT
; Protección del código: Desactivado
; *****

LIST P = 16F84          ;Indicamos el modelo de PIC a utilizar

; Definición de registros

portb EQU 0x06 ;Hemos conectado el teclado al puerto B
          ;La dirección 0x06 corresponde al registro PORTB (puerto B)
          ; en el banco1
TRISB EQU 0x06 ; y TRISB en banco 1
estado EQU 0x03 ; La dirección del registro de estado es la 0x03
pc EQU 0x02 ; Contador de Programa, dirección de memoria actual de programa
intcon EQU 0x0B ; Registro gestor de interrupciones
opcion EQU 0x01 ; Registro OPTION. Recordar que está en el banco 1.

; Definición de bits

banco EQU 0x05 ; Bit del registro de estado correspondiente al banco de datos
Z EQU 0x02 ; Bit indicador de que el registro W está a cero
int EQU 0x00 ; Bit de interrupción externa, es el 0 en el puerto B.
intdeg EQU 0x06 ; Bit 6 de OPTION, que indica si la interrupción PB0 es por nivel
alto.
intf EQU 0x01 ; Bit 1 de INTCON, flag de interrupción por PB0.
inte EQU 0x04 ; Bit 4 de INTCON, habilitador de interrupción por PB0.
GIE EQU 0x07 ; Bit 7 de INTCON, habilitador de interrupciones.

; Definición de constantes

w EQU 0 ; Destino de operación = w
f EQU 1 ; Destino de operación = registro

; Definición de variables

contador EQU 0x0C ; Contador
digito EQU 0x0D ; Para almacenar el dígito

; Comienzo del programa.

ORG 0x00 ; Cubrimos el vector de reset

GOTO inicio ; Saltamos a la primera dirección tras el vector de interrupción

ORG 0x04 ; Vector de interrupción

GOTO RSI

```

```

, ***** Inicialización de variables *****
ORG 0X05

inicio BSF estado,banco ; Cambiamos a la segunda página de memoria
        CLRF TRISB ; Programa la puerta B como de todo salidas
        BSF TRISB,int ; Salvo la pata de interrupción PB0, que es de entrada
        BSF opcion,intdeg ; Interrupción PB0 cuando esté a nivel alto.
        BCF estado,banco ; Volvemos a la página 0.
        BCF intcon,intf ; Borramos el flag de interrupción por PB0.
        BSF intcon,GIE ; Habilitamos las interrupciones.
        BSF intcon,inte ; Habilitamos la interrupción por PB0.
        CLRF portb ; Apaga el display, por si había residuos
        CLRF contador ; Borra el contador (dirección 0x0C)
        CLRW ; Borramos el registro W

, ***** Cuerpo Principal *****

Reset CLRF digito ; Comienza a contar por el 0

Siguien MOVF digito,w ; Coloca el siguiente dígito a evaluar en W
        CALL Tabla ; Llama a la subrutina para coger el dato
                ; y hacer la conversión decimal-7 segmentos
        MOVWF portb

Pausa DECFSZ contador ; Decrementa contador y repite
        GOTO Pausa ; hasta que valga 0
        INCF digito,f ; Incrementa el valor del dígito al siguiente
        MOVF digito,w ; Pone el valor del dígito en W
        XORLW 0x0A ; Chekea si el dígito sobrepasa el valor 9
        BTFSC estado,Z ; Comprobando con un xor si W vale 0 (Z=1)
        GOTO Reset ; Si Z=1 resetea el dígito y comienza de nuevo la cuenta
        GOTO Siguien ; En caso contrario, continua la cuenta

, ***** La tabla queda definida aquí *****

Tabla ADDWF pc,f ; Suma al contador de programa el valor de offset, es decir,
                ; el valor del dígito. Así se genera un PC distinto
                ; según su valor,
                ; asegurando que vaya a la línea correcta de la tabla
        RETLW 0x7F ; 0 en código 7 segmentos (desplazado a la izquierda)
        RETLW 0x0C ; 1 en código 7 segmentos (desplazado a la izquierda)
        RETLW 0xB6 ; 2 en código 7 segmentos (desplazado a la izquierda)
        RETLW 0x9F ; 3 en código 7 segmentos (desplazado a la izquierda)
        RETLW 0xCC ; 4 en código 7 segmentos (desplazado a la izquierda)
        RETLW 0xDA ; 5 en código 7 segmentos (desplazado a la izquierda)
        RETLW 0xFA ; 6 en código 7 segmentos (desplazado a la izquierda)
        RETLW 0x0F ; 7 en código 7 segmentos (desplazado a la izquierda)
        RETLW 0xFF ; 8 en código 7 segmentos (desplazado a la izquierda)
        RETLW 0xDF ; 9 en código 7 segmentos (desplazado a la izquierda)

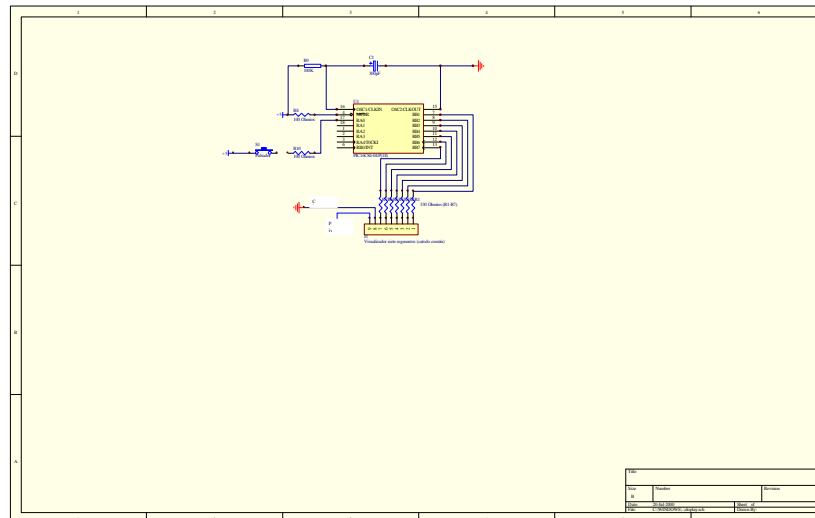
RSI BTFSS intcon,intf ; Si no es interrumpido por PB0, volver al programa
    RETFIE

pulsado BTFSC portb,0 ; Retenemos hasta que se suelte el pulsador
        GOTO pulsado
        MOVLW 0xFF ; Puesto que se habrá de incrementar
        MOVWF digito ; Ponemos el marcador a FF
        BCF intcon,intf ; Borramos la bandera de interrupción
        BSF intcon,inte ; Y rehabilitamos la interrupción por PB0
        RETFIE

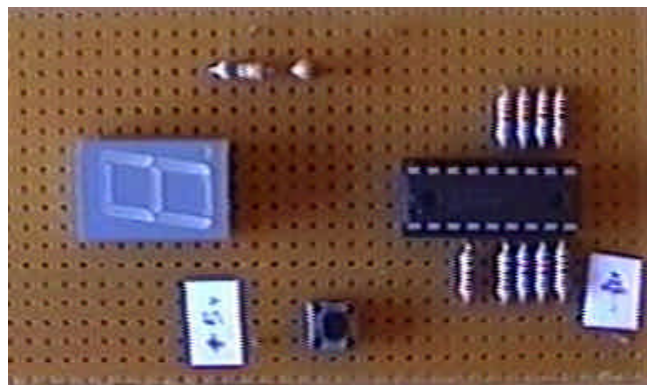
END

```

El esquema electrónico del visualizador lo tenéis a continuación, pero hemos tenido problemas para su correcta visualización desde Word por lo que podéis ampliar la imagen siguiente o podéis acudir al CD adjunto donde se encuentra el esquema y una copia de evaluación de Protel 98, un programa para la creación de circuitos:



Finalmente este es nuestro circuito una vez montado:



9. BIBLIOGRAFÍA

9.1 Bibliografía escrita.

Microcontroladores PIC. La Solución en un Chip.

J. M^a. Angulo Usategui, E. Martín Cuenca, I. Angulo Martínez
Ed. Paraninfo. (1997)

Microcontroladores PIC. Diseño práctico de aplicaciones.

J. M^a. Angulo Usategui, I. Angulo Martínez
Mc Graw Hill [1999]

Microcontroladores.

Vicente Torres.
Servicio Publicaciones UPV.

Programming and Customizing the Pic Microcontroller

Myke Predko
Mc Graw Hill [1999]

Electrónica. Microcontroladores y Microprocesadores.

Fascículos coleccionables. Editorial Multipress SA.

PIC16/17 Microcontroller Data Book.

Microchip Technology Inc. (1995 - 96)

Technical Training Workbook de Microchip

Microchip Technology Inc. (1999)

Embedded Control Handbook

Microchip Technology Inc. (1995 - 96)

July 1999 Technical Library CD-ROM

Microchip Technology Inc. (1999)

Microchip Technical CD-ROM First Edition 2000

Microchip Technology Inc. (2000)

MPSIM Simulator Quick Reference Guide

Microchip Technology Inc. (1996)

MPASM Assembler Quick Reference Guide

Microchip Technology Inc. (1996)

MPSIM Simulator User's guide

Microchip Technology Inc. (1995)

MPASM Assembler User's Guide

Microchip Technology Inc. (1995)

9.2 Bibliografía electrónica.

Microchip.	http://www.microchip.com
Parallax.	http://www.parallaxinc.com
Página Web de CX2FW: Información y Links.	http://www.angelfire.com/tx/cx2fw/cx2fw.html
Página de Javier Alzate: Microcontroladores PIC16CXX.	http://www.geocities.com/CapeCanaveral/Lab/9827/microcon.htm
Private Users System: Programador de Pics.	http://www.lokekieras.es/personales/mgsanz/programa.htm
El Rincón del Pic.	http://members.es.tripod.de/~InfoE/infop.htm
Microsystems Engineering: Los autores de los libros de Pics en castellano.	http://www.arrakis.es/~msyseng
Links sobre Pics de David Tait.	http://www.man.ac.uk/~mbhstdj/piclinks.html
Archivos sobre Pics de David Tait.	http://www.labyrinth.net.au/~donmck/dtait/index.html
De todo un poco (Electrónica): Algunos circuitos.	http://www.arrakis.es/~ldr2000/manny/circuitos
Rei Project: Mod Chip: Algunos proyectos.	http://chip.aeug.org
NewFound Electronics: Programador de Pics.	http://www.new-elect.com
Dontronics.	http://www.dontronics.com
The Picmicro Ring.	http://members.tripod.com/~mdileo/pmring.html
Microcontroladores: Información, Herramientas y Programador.	http://www.geocities.com/TheTropics/2174/micro.html
Microcontrollers: Enlaces.	http://www.us-epanorama.net/microprocessor.html
NOPPP, the "No-Parts" PIC Programmer.	http://www.CovingtonInnovations.com/noppp/nopp-p-sp.html
Parallel Port PIC16C5X/XX Programmer.	http://www.labyrinth.net.au/~donmck/dtait/upp.html
Microchip Net resources.	http://www.geocities.com/SiliconValley/Way/5807
PIC16/17 Microcontroller & Basic Stamp: Con algunos proyectos.	http://www.doc.ic.ac.uk/~ih/doc/pic
La página del autor de Programming and Customizing the Pic Microcontroller: Con algunos circuitos.	http://www.myke.com/PICMicro
Pic Programming. Getting Started: 4 pasos para empezar con los Pic.	http://www.pp.clinet.fi/~newmedia/pic/index.html
Pic Programmer 2.	http://www.jdm.homepage.dk/newpic.htm
GNUPic "Free Microcontroller Software Tools	http://huizen.dds.nl/~gnupic/index.html
Propic2: Programador de Pics.	http://www.propic2.com
PicProg Home Page: Programador de Pics.	http://virtuaweb.com/picprog
Mundo Electrónico: Enlaces.	http://www.geocities.com/CapeCanaveral/Campus/9468/mundo.htm
The Electronic Projects Page: Algunos proyectos.	http://www.blichfeldt.dk
Bengt Lindgrens HomePage: Programador y archivos.	http://home5.swipnet.se/~w-53783
The ultimate source for Pic and SX Tools	http://www.adv-transdata.com
P16PRO & PICALL PIC programmers: Otro programador	http://www.geocities.com/SiliconValley/Peaks/9620
Proyecto de Gaspar Vidal que utiliza los Pic como soporte hardware.	http://www.geocities.com/CapeCanaveral/Campus/8775/proyecto/pfc.htm
Diseño de sistemas con microcontroladores:	http://www.info-ab.uclm.es/~amartine
Enlaces.	
Los Microcontroladores: Información sobre algunos modelos.	http://www.gherson.homepage.com
Free PIC 16x84 programmer with margining support.	http://www.ise.pw.edu.pl/~wzab/picprog/picprog.html
FlashPIC Developer for PIC16F84 and PIC16F87x Series PIC Microcontrollers.	http://www.cybermedix.co.nz/flashpic
Sagitron: Distribuidor de Microchip en España.	http://www.sagitron.es
Indicadores y controles basados en micros pic: Otro	http://chasque.chasque.apc.org/franky/pics.htm

proyecto.

EDU-PIC: PIC Microcontrollers in education.

Programmer for PIC-processors.

<http://pages.hotbot.com/edu/edu-pic>

<http://www.qsl.net/lz2rr/pic.html>