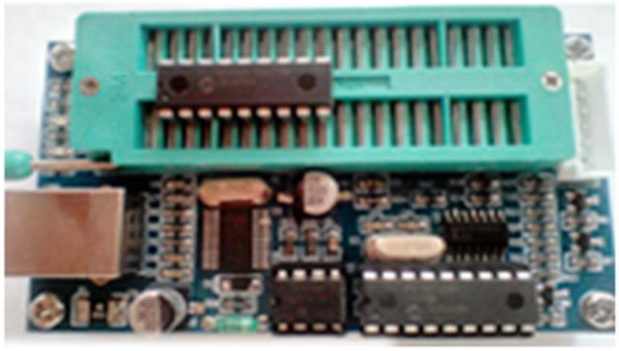
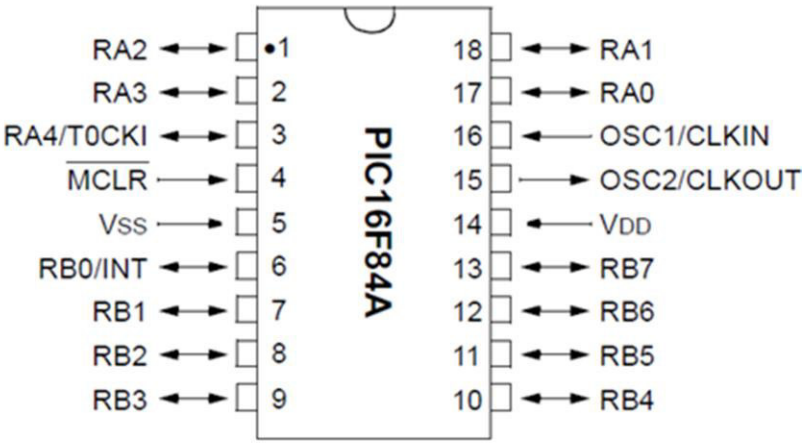
Microcontrolador PIC 16F84A

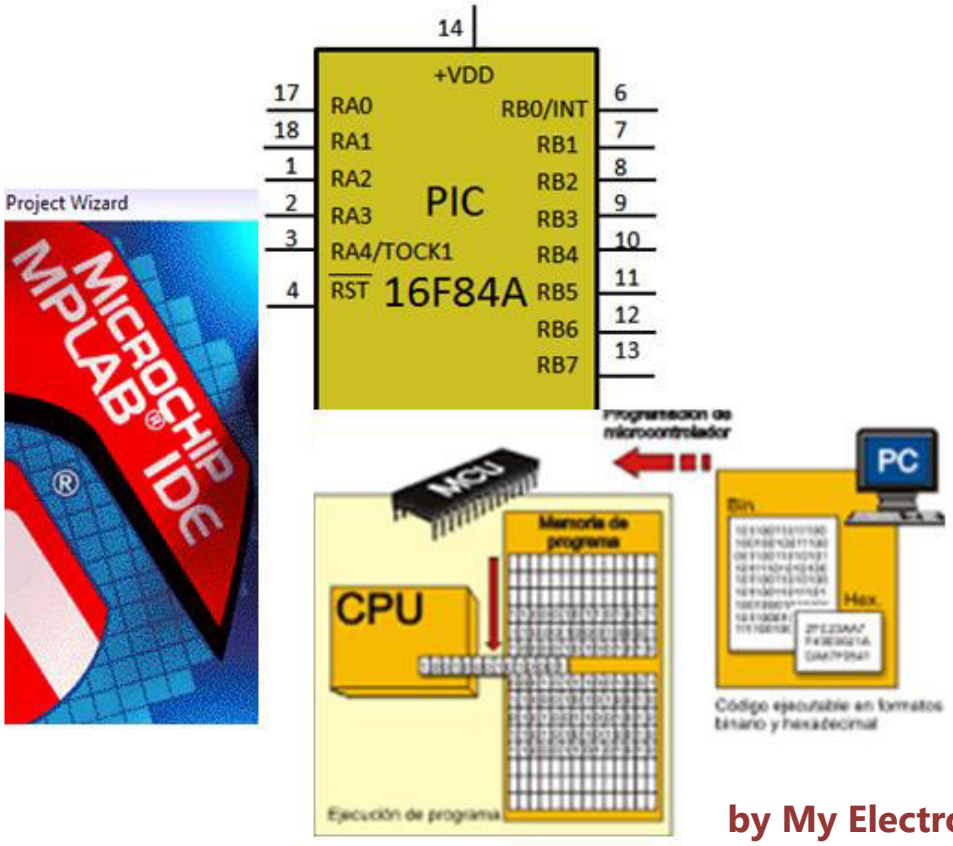
Características y Programación





Microcontrolador PIC 16F84A

Características y Programación



by My Electronic

INDICE DE CONTENIDO

1. MICROCONTROLADOR

Un microcontrolador es un circuito integrado digital que puede ser usado para muy diversos propósitos debido a que es *programable*. Está compuesto por una unidad central de proceso (CPU), memorias (ROM y RAM) y líneas de entrada y salida (periféricos).

Como podrás darte cuenta, un microcontrolador tiene los mismos bloques de funcionamiento básicos de un ordenador lo que nos permite tratarlo como un pequeño dispositivo microordenador.

Un microcontrolador puede usarse para muchas aplicaciones algunas de las cuales son: manejo de sensores, controladores, juegos, calculadoras, agendas, avisos lumínicos, secuenciador de luces, cerrojos electrónicos, control de motores, relojes, alarmas, robots, lavadoras, teclados, teléfonos móviles, ratones etc. El límite es la imaginación.

El funcionamiento de un microcontrolador se basa en un hardware que ya viene integrado en un solo chip y para usarlo se debe especificar su funcionamiento por software a través de programas que indiquen las instrucciones que el microcontrolador debe realizar. En una memoria se guardan los programas y un elemento llamado CPU se encarga de procesar paso por paso las instrucciones del programa. Los lenguajes de programación que se usan para este fin son **ensamblador** y **C**, pero antes de grabar un programa en el microcontrolador hay que compilarlo a **hexadecimal** que es el formato con el que funciona el microcontrolador.

Cuando hablamos del microcontrolador estamos hablando de un "**chip**" un circuito integrado programable que controla periféricos, y que posee internamente un sistema que contiene entre otras cosas una unidad central de proceso, unas memorias de datos y programas, unos puertos de entrada y salida, es decir estamos hablando de un pequeño ordenador diseñado para realizar unas funciones específicas.

Para diseñar programas es necesario conocer los bloques funcionales básicos del microcontrolador, estos bloques son:

- **CPU** (Unidad central de proceso)
- **Memoria ROM** (Memoria de solo lectura)
- **Memoria RAM** (Memoria de acceso aleatorio)
- **Líneas de entrada y salida** (Periféricos)

La forma en la que interactúan estos bloques dependerá de su arquitectura.

La **CPU** posee, de manera independiente, una memoria de acceso rápido para almacenar datos denominada **registros**, si estos registros son de 8 bits se dice que el microcontrolador es de 8 bits.

Para grabar un programa en un microcontrolador se necesita básicamente tres cosas:

1. Un ordenador
2. Software de programación (incluyendo un compilador).
3. Un programador de PIC

Y obviamente también se necesita un microcontrolador, por ejemplo, el PIC 16F84A.

El ordenador, a través del software de grabación, se encarga de enviar el programa que se desea grabar al microcontrolador por medio del circuito programador.

Hay multitud de microcontroladores con más memoria, entradas y salidas, frecuencia de trabajo, coste, subsistemas integrados y un largo etc. dependiendo de cada tipo de microcontrolador. El presente documento está basado en el popular microcontrolador **PIC 16F84A** del fabricante Microchip Technology Inc ya que es un sistema sencillo, barato y potente para muchas aplicaciones electrónicas.

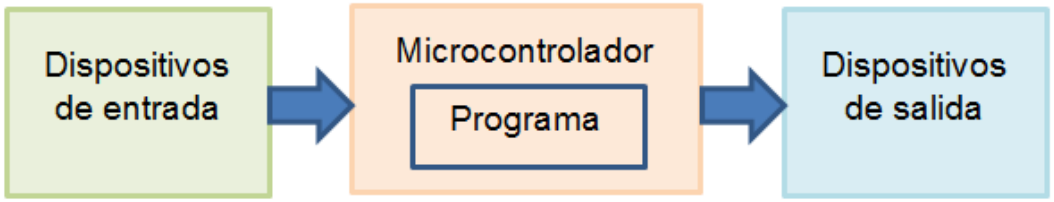
Algunas empresas destacadas en la fabricación de microcontroladores

Microchip Technology, denominada comúnmente **Microchip** es una de las empresas líderes en la fabricación de microcontroladores. Para esta empresa, los microcontroladores se conocen con el apodo «**PIC**».

Debido a sus bajos costos, desempeño eficiente, gran documentación y fácil adquisición, los microcontroladores de Microchip, conocidos simplemente como PIC, será el utilizado a lo largo de este documento de la serie PIC 16F84A.

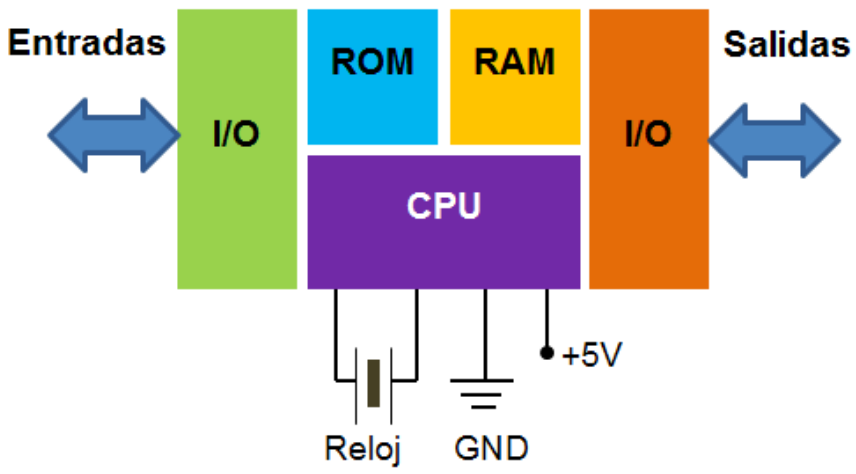
Atmel, otra empresa líder en este campo es famosa por crear los microcontroladores sobre los que se basan los **arduinos**. Existen otras alternativas, ofrecidas por empresas como Texas Instruments, **Freescale**, entre otras. Sus productos pueden ser encontrados en tiendas de electrónica, aunque no siempre se consiguen fácilmente por lo que podemos acudir a las ventas por internet para obtenerlos.

El diagrama de un sistema microcontrolador sería algo así:



Los dispositivos de entrada pueden ser un teclado, un interruptor, un sensor, etc. Los dispositivos de salida pueden ser LED's, pequeños parlantes, zumbadores, interruptores de potencia (tiristores, optoacopladores), u otros dispositivos como relés, luces, un secador de pelo, etc.

En la siguiente figura se muestra en bloques el microcontrolador. Se puede observar que se adapta tal como es un ordenador, con su alimentación, el circuito de reloj, la **CPU**, las memorias **RAM** y **ROM**, y por supuesto, sus puertos de comunicación de entradas y salidas **I/O**, listos para conectarse al mundo exterior.



Estructura interna de un Microcontrolador

Estas son las funciones especiales de las cuales disponen algunos microcontroladores:

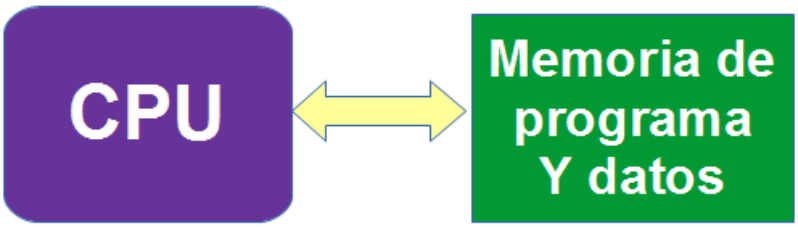
- **Conversores análogo a digital** (A/D). En el caso de que se requiera medir señales analógicas, por ejemplo temperatura, voltaje, luminosidad, etc.
- **Temporizadores programables** (Timer's). Si se requiere medir períodos de tiempo entre eventos, generar temporizaciones o salidas con frecuencia específica, etc.
- **Interfaz serial RS-232**. Cuando se necesita establecer comunicación con otro microcontrolador o con un ordenador.
- **Memoria de programa**. Del tipo Flash, que durante su funcionamiento es de solo lectura. Sólo se ejecutará el código contenido en esta memoria, pudiendo almacenar en ella una cantidad limitada de datos como parte de la instrucción **RETLW**.
- **Memoria de datos**. Es del tipo **EEPROM**. Para desarrollar una aplicación donde los datos no se alteren a pesar de quitar la alimentación, que es un tipo de memoria **ROM** que se puede programar o borrar eléctricamente sin necesidad de circuitos especiales. Está organizada en dos páginas o bancos de registro para cambiar de página se utiliza un bit de registro **STATUS** (RP0). Cada banco se divide a su vez en dos áreas: registro de funciones especiales (RFS) y registro de propósito general (RGP).
- Salidas **PWM** (modulación por ancho de pulso). Para quienes requieren el control de motores DC o cargas resistivas, existen microcontroladores que pueden ofrecer varias de ellas.
- Técnica llamada de **Interrupciones**. Cuando una señal externa activa una línea de interrupción, el microcontrolador deja de lado la tarea que está ejecutando, atiende dicha interrupción, y luego continúa con lo que estaba haciendo.
- **Direccionamiento**. Los modos de direccionamiento tratan sobre la forma de mover los datos de unas posiciones de memoria a otras. Los modos de direccionamiento son **Inmediato**, **Directo** y **Bit a Bit**.

Las altas prestaciones de los microcontroladores derivan de las características de su arquitectura. Están basados en una arquitectura tipo **Harvard** que posee buses y espacios de memoria por separado para el **programa** y los **datos**, lo que lo hace que sean más rápidos que los microcontroladores basados en la arquitectura tradicional de Von Neuman.

2. ARQUITECTURA DE MICROCONTROLADORES

La arquitectura de un microcontrolador permite definir la estructura de su funcionamiento, las dos arquitecturas principales usadas en la fabricación de microcontroladores son: arquitectura de **Von Neumann** y arquitectura **Harvard.** Además, estas arquitecturas pueden tener procesadores de tipo **CISC** o de tipo **RISC**.

- Arquitectura de Von Neumann

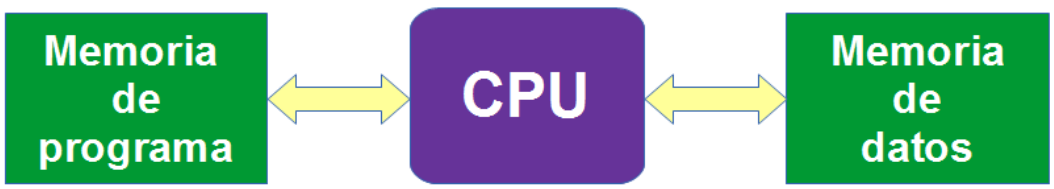


Arquitectura de Von Neumann

En esta arquitectura, los *datos* y las **instrucciones** circulan por el mismo bus ya que estos son guardados en la misma memoria, su principal ventaja es el ahorro de líneas de entrada-salida pero esto supone una disminución en la velocidad con la que se realizan los procesos.

Este tipo de arquitectura es hoy en día muy común en los computadores personales, y fue muy común en la construcción de microcontroladores hasta que se descubrieron las grandes ventajas de la arquitectura Harvard.

- Arquitectura Harvard



Arquitectura Harvard

A diferencia de la anterior, en la arquitectura Harvard existe una memoria específica para datos y una memoria específica para las instrucciones, de esta forma se usan dos buses bien diferenciados. Con esto se logra trabajar con las dos memorias simultáneamente y en consecuencia se obtiene mucha más velocidad en la ejecución de los programas.

Actualmente, la tendencia de los microcontroladores es usar este tipo de arquitectura.

- Procesador de tipo CISC (Complex Instruction Set Computer)

Un procesador que permita manejar un amplio juego de instrucciones es llamada de tipo **CISC** que en español significa «Ordenador con Juego de Instrucciones Complejo», programar en este tipo de arquitectura requiere en algunos casos del dominio de hasta centenares de instrucciones.

- Procesador de tipo RISC (Reduced Instruction Set Computer)

Cuando un procesador está diseñado para manejar pocas instrucciones pero sin afectar las prestaciones del ordenador es llamada de tipo **RISC** que en español significa «Ordenador con Juego de Instrucciones Reducido», esto permite programar con mucha más facilidad y, por si fuera poco, los circuitos de tipo **RISC** disponen de una estructura que busca como mínimo la instrucción próxima a ejecutar mientras realiza la instrucción actual. Esta estructura permite lograr no solo mayor velocidad de proceso sino también procesar cada instrucción con la misma velocidad.

NOTA: Microchip introdujo la arquitectura **Harvard** con procesador tipo **RISC** en sus microcontroladores cuando el mercado era dominado por microcontroladores con arquitectura de Von Neumann, desde entonces las ventajas que ofreció esta nueva tecnología permitieron a Microchip sobresalir como uno de los más grandes fabricantes de microcontroladores en el mundo.

3. CARACTERÍSTICAS DEL MICROCONTROLADOR PIC 16F84A

El significado de **PIC** lo define el fabricante como "**P**rogramable **I**ntegrado **C**ircuito" que quiere decir que el integrado se puede programar y reprogramar adaptándose a nuestra necesidades mediante unas instrucciones que hayamos introducido en el programa. Es capaz de modificar su comportamiento tanta veces que se haga necesario y en función del cambio de una serie de instrucciones.

Posee una **ALU**, Unidad Aritmética Lógica, de 8 bits capaces de realizar operaciones de desplazamientos, lógicas, sumas y restas. Posee un registro de trabajo (W) no direccionable que usa en operaciones con la ALU. Dependiendo de la instrucción ejecutada, la ALU puede afectar a los bits de Acarreo, Acarreo Digital (**DC**) y Cero (**Z**) del registro de estado (**STATUS**).

El **PIC16F84A** tiene 68 bytes de **RAM** y 64 bytes de **EEPROM** de datos, tiene 12 registros de función específica **FSR**. Con 8 niveles de Pila hardware y tres modos de direccionamiento: **directo**, **indirecto** y **relativo**. Dispone de cuatro fuentes de interrupción y 13 pines de E/S, puertoA de 5 bits y el puertoB de 8 bits, con control individual bidireccional y dispone de un Timer0/ Contador con reloj independiente y la gran ventaja dispone de Perro Guardián (**WDT**).

El **PIC16F84A** tiene un contador de programa **PC** de 13 bits capaz de dirigir 8 kilobytes x 14 espacio de memoria de programa. Para el F84A, el primer 1 kilobyte x 14 (0000h-03FFh) físicamente es implementado (Figura 2-1). El tener acceso a una posición anterior de la dirección físicamente puesta en práctica causará un recirculado. Por ejemplo, para posiciones 20h, 420h, 820h, C20h, 1020h, 1420h, 1820h, y 1C20h, la instrucción será la misma. El vector RESET (REPONER) está en la dirección 0000h y el vector INTERRUPT está en la 0004h, con cuatro fuentes de interrupción, véase el documento interrupciones descrito en estas páginas.

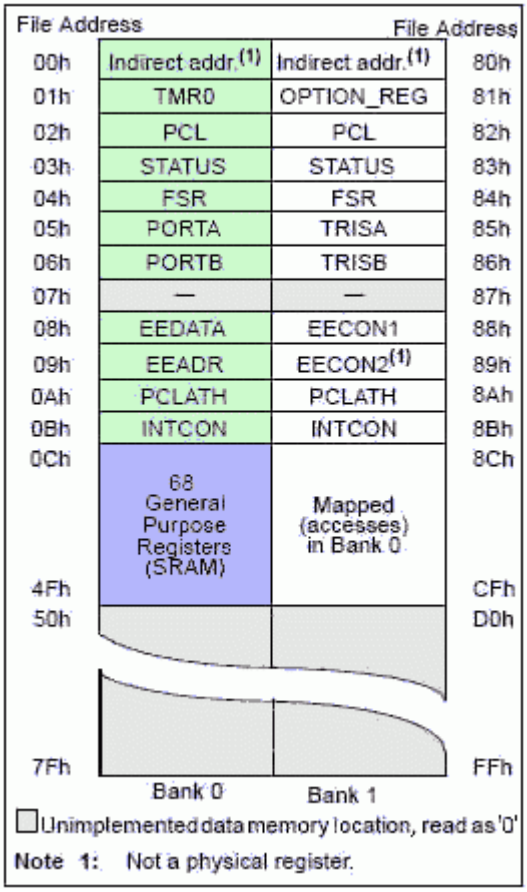
Hay dos bloques de memoria en el **PIC16F84A**, estos son la **memoria de programa** y la **memoria de datos**. Cada bloque tiene su propio bus, de modo que el acceso a cada bloque pueda ocurrir durante el mismo ciclo de oscilador. La memoria de datos adicional mayor puede ser direccionada en la RAM de objetivo general y los Registros de Función Especiales (**SFRs**). La operación de los **SFRs** que controla el «corazón» (reloj) son usados para controlar los módulos periféricos.

El área de memoria de datos también contiene la memoria de datos **EEPROM**. Ésta memoria no es mapeada directamente en la memoria de datos, pero es mapeada indirectamente. Es decir, un puntero de dirección indirecta, especifica la dirección de la memoria de datos EEPROM para lectura/escritura. Los 64 octetos de datos de la memoria EEPROM tienen la gama de dirección 00h-3Fh. Más detalles de la memoria EEPROM se pueden encontrar en la Sección 3.0 de las hojas de características.

La memoria de datos es dividida en dos áreas. La primera, es el área de Registros de Función Especial (**SFR**), mientras la segunda es el área de Registros de Propósito General (**GPR**). Los SFRs controlan el modo de operación del dispositivo. Partes de memoria de datos son mapeadas, esto es tanto para el área SFR como para el área GPR. El área GPR es mapeada para permitir una capacidad mayor de 116 bytes de RAM de propósito general.

Las áreas mapeadas del SFR son para los registros que controlan las funciones periféricas. Los bancos requieren el empleo de bits de control para la selección de banco. Estos bits de control son localizados en el Registro **STATUS**. En la figura 2-2 se muestra la organización de mapa de memoria de datos.

FIGURA:2-2 MAPA REGISTROS PIC16F84A



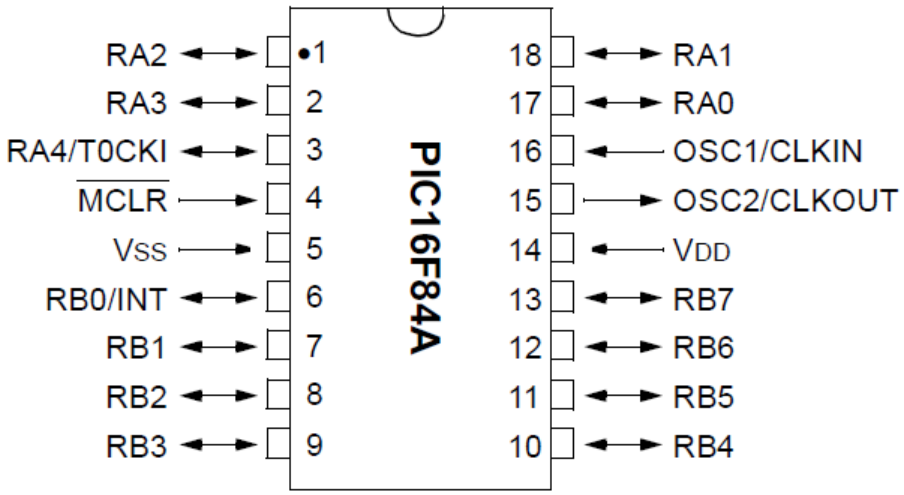
Las Instrucciones MOVWF y MOVF pueden mover valores del registro W a cualquier posición en la RAM al archivo de registro («F«) y viceversa. Se puede tener acceso directo a toda la memoria de datos utilizando la dirección absoluta de cada archivo de registro o indirectamente mediante los Registros de Función Especial (SFR). La dirección indirecta usa el valor actual del bit RP0 para el acceso en las áreas mapeadas de memoria de datos.

Resumen y características principales del PIC16F84A:

- Set de 35 instrucciones
- Memoria de programa de 1 KB (equivale a 1024 instrucciones)
- Máxima velocidad de operación: 20 MHz
- 68 Bytes de RAM
- 64 Bytes de EEPROM
- 4 fuentes de interrupción
- 2 puertos de salida
- 13 Líneas de I/O configurables individualmente
- 25 mA de corriente por pin
- 1,75 KB de tamaño de la memoria de programa
- Chip de 18 pines
- Alimentación de 2 a 5,5 voltios
- 1 temporizador de 8 bits

Se programa en ensamblador mediante 35 instrucciones, con códigos de instrucción de 14 bits de ancho. La edición y compilación del programa se hace mediante la aplicación MPLAB o MPASM de Microchip. Editamos en ensamblador y lo compilamos en hexadecimal y finalmente ese fichero hexadecimal se graba en un programador PIC.

- Descripción de los pines



- **VDD**: Alimentación positiva (5 voltios)
- **VSS:** Tierra (0 voltios)
- **MCLR**: pin de resetear a nivel bajo
- **OSC1, OSC2**: Conexión de oscilador
- **RA0, RA1, RA2, RA3, RA4**: líneas I/O del puerto A
- **RB0, RB1, RB2, RB3, RB4, RB5, RB6, RB7**: líneas I/O del puerto B

Pines 5 y 14 (VSS y VDD): Son respectivamente los pines de alimentación del microcontrolador, la masa o tierra es **VSS** y la conexión +5 V positiva **VDD**. La tensión de alimentación de un PIC está comprendida entre 2V y 6V aunque se recomienda no sobrepasar los 5.5V.

Pin 4 (MCLR / Vpp): Es un pin de múltiples aplicaciones, es la entrada de **Reset** (master clear) si está a nivel bajo y también es la habilitación de la tensión de programación cuando se está programando el dispositivo. Cuando su tensión es la de **VDD** el PIC funciona normalmente.

Pines 15 y 16 (OSC1/CLKIN y OSC2/CLKOUT): Corresponden a los pines de la entrada externa de reloj y salida de oscilador a cristal respectivamente.

Pines 1, 2, 3, 17 y 18 (RA0-RA4/TOCKI): Es el **PORT A**. Corresponden a 5 líneas bidireccionales de E/S (definidas por programación). Es capaz de entregar niveles TTL cuando la alimentación aplicada en VDD es de 5V ± 5%. El pin RA4/TOCKI como entrada puede programarse en funcionamiento normal o como entrada del contador/temporizador

TMR0. Cuando este pin se programa como entrada digital, funciona como un disparador de Schmitt (**Schmitt trigger**), puede reconocer señales un poco distorsionadas y llevarlas a niveles lógicos (cero y cinco voltios). Cuando se usa como salida digital se comporta como colector abierto; por lo tanto se debe poner una resistencia de pull-Up (resistencia externa conectada a un nivel de cinco voltios). Como salida, la lógica es inversa: un "0" escrito al pin del puerto entrega a la salida un "1" lógico. Este pin como salida no puede manejar cargas como fuente, sólo en el modo sumidero.

Pines 6, 7, 8, 9, 10, 11, 12, 13 (RB0-RB7): Es el **PORT B**. Corresponden a ocho líneas bidireccionales de E/S (definidas por programación). Pueden manejar niveles TTL cuando la tensión de alimentación aplicada en VDD es de 5V ± 5%. RB0 puede programarse además como entrada de interrupciones externas INT. Los pines RB4 a RB7 pueden programarse para responder a interrupciones por cambio de estado. Las patas RB6 y RB7 se corresponden con las líneas de entrada de reloj y entrada de datos respectivamente, cuando está en modo programación del integrado.

- Los Fuses del PIC

Estas 4 "variables" del PIC16F84A, sirven para configurar ciertos aspectos del microcontrolador. Cada **FUSE** activa o desactiva una opción de funcionamiento.

1. **Oscilator** (Oscilador): Es el modo de oscilación que va a usar el PIC. Cada vez que el PIC recibe un pulso eléctrico del oscilador da un paso para ejecutar una instrucción (4 impulsos para completar una), por lo que podemos decir que es una señal que le recuerda al PIC que tiene que seguir avanzando.

   Según esto, el PIC puede usar 4 tipos de oscilador:

   - **XT**: Es un acrónimo que viene de XTAL conocido como cristal de cuarzo . Este modo de funcionamiento implica que tendremos que disponer de un cristal de cuarzo externo al PIC y dos condensadores. El valor del cristal generalmente será de 4Mhz o 10Mhz, y los condensadores serán cerámicos de entre 27 y 33 nF. La exactitud de este dispositivo es muy alta, por lo que lo hace muy recomendable para casi todas las aplicaciones.
   - **RC**: Este es el sistema más sencillo y económico. Se basa en un montaje con una resistencia y un condensador. La velocidad a la que oscile el PIC

dependerá de los valores del condensador y de la resistencia. En la hoja de características del PIC están los valores.

- **HS**: “High Speed” Para cuando necesitemos aplicaciones de "alta velocidad", entre 8 y 10Mhz. Se basa también en un cristal de cuarzo, como el **XT**
- **LP**: "Low Power" la velocidad máxima a la que podemos poner el PIC con este oscilador es de 200Khz. Al igual que el XT y el HS, necesitaremos de un cristal de cuarzo y unos condensadores.

2. **WDT** (**Watchdog Timer**): El famoso "perro" del PIC. (perro guardián). Esta es una capacidad del PIC de autoresetearse. Es muy útil, por ejemplo si un PIC, por un descuido de programación, se queda en un bucle infinito, esta "utilidad" lo sacará de él. Su funcionamiento es sumamente sencillo. Simplemente es un registro que debemos borrar cada cierto tiempo. Si transcurrido un cierto tiempo el registro no ha sido borrado el PIC se resetea. La instrucción para borrar el registro es **CLRWDT**. Con poner un par de ellos a lo largo de nuestro código es suficiente para tener una garantía de que el PIC no se quede en bucle infinito.

3. **PWRT** (**Power Up Timer Reset**): Si activamos este FUSE, lo que conseguimos es que se genere un retardo en la inicialización del PIC. Esto se usa para que la tensión se estabilice, por lo que se recomienda su uso.

4. **Code Protect**: Protección del código. Lo único que hace es impedir que algún curioso se apropie de tu creación no tiene efecto alguno en el correcto funcionamiento del PIC, ni que no se pueda sobrescribir su contenido. Lo único que hace es impedir su lectura.

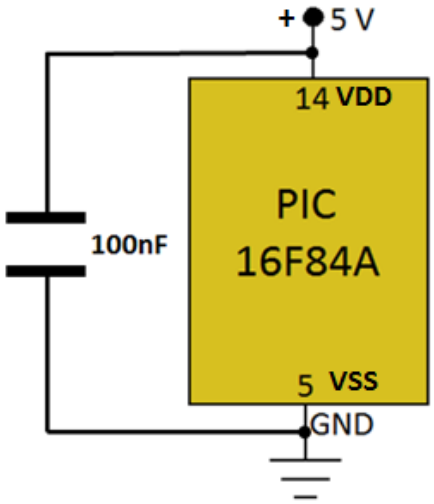
Estas variables se suelen poner en el inicio de la programación en ensamblador para indicarle al microcontrolador el tipo de **Oscilator** utilizado, si se habilita o no el **Watchdog,** si se habilita o no el **PWRT,** o si utilizamos o no la **protección del código.**

El programador PIC K-150 posee la opción de configurar los fuses sin tenerlo que declarar en la programación.

- La alimentación

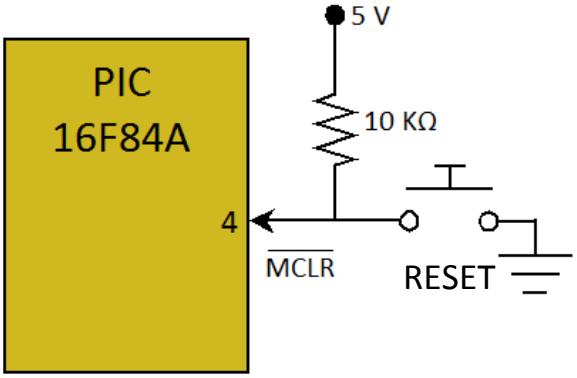
El PIC 16F84A se **alimenta a 5 voltios** entre los pines Vdd (+) pin 14 y Vss (-) pin 5. El consumo del circuito depende de las cargas en los puertos y de la frecuencia de trabajo.

El voltaje de alimentación debe estar comprendido entre 2.0 y 5.5 voltios (preferiblemente usar una alimentación regulada de 5 voltios). Para mejorar su desempeño se puede adicionar un condensador cerámico de 100 nF y 63V conectado en paralelo y lo más cerca posible de los pines 5 y 14 VSS y VDD de la entrada de tensión del microcontrolador.



- Circuito de Reset

El circuito de reset ($\overline{MCLR}$) se encuentra en el pin 4 del PIC 16F84A, este pin se activa a nivel bajo"0" y se le conectarán dos componentes: una resistencia de 10 kΩ a VDD (+5 voltios) y un pulsador a GND (masa). Este circuito inicializa o resetea el microcontrolador cuando se activa a nivel bajo. Para que funcione y corra el programa del microcontrolador debe estar el pin 4 a nivel alto.



Cuando ocurre un **Reset**, el contador de programa (PC) apunta a la dirección 0000h, y el micro se inicia nuevamente. Por esta razón, en la primera dirección del programa se debe escribir todo lo relacionado con la iniciación del mismo.

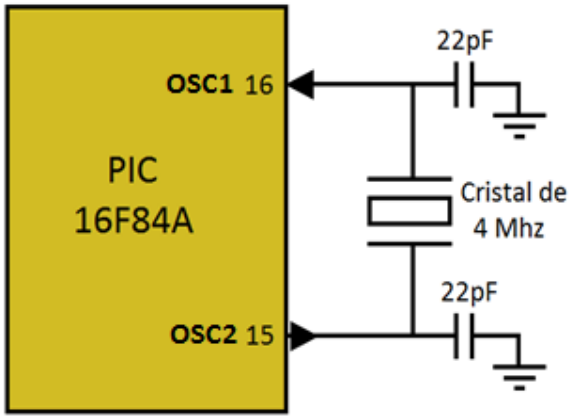
- Circuito de Reloj

El circuito de reloj oscilador marcará la frecuencia de trabajo del microcontrolador en los pines 15 y 16 (OSC2 y OSC1). Estos osciladores pueden ser del tipo:

- **RC** Formado por una resistencia y un condensador
- **HS** se utiliza un cristal de cuarzo o resonador cerámico (Hasta 20 Mz)
- **XT** Cristal o resonador hasta 4 Mhz
- **LP** Bajo consumo (hasta 200Khz)

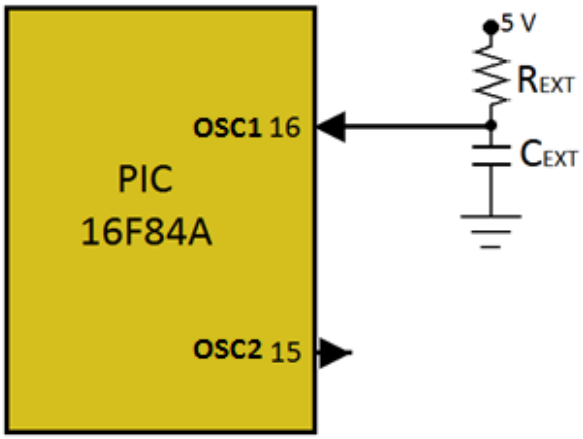
Mode	Freq	OSC1/C1	OSC2/C2
LP	32 kHz 200 kHz	68 - 100 pF 15 - 33 pF	68 - 100 pF 15 - 33 pF
XT	100 kHz 2 MHz 4 MHz	100 - 150 pF 15 - 33 pF 15 - 33 pF	100 - 150 pF 15 - 33 pF 15 - 33 pF
HS	4 MHz 20 MHz	15 - 33 pF 15 - 33 pF	15 - 33 pF 15 - 33 pF

Un microcontrolador ejecuta su programa paso por paso, la unidad que mide el paso más básico es denominada ciclo de máquina y la velocidad con que se ejecuta está directamente relacionada con la frecuencia del oscilador. En este caso utilizamos un cristal de cuarzo de 4 MHz entre los pines 15 y 16, OSC1 y OSC2, también conectaremos dos condensadores cerámicos de 22 pF entre OSC1 y masa y entre OSC2 y masa.



En el momento de programar el microcontrolador se debe especificar qué tipo de oscilador se usa, por ejemplo, **XT** cuando utilizamos un cristal de cuarzo de 4MHZ. Esto se hace a través de unos fusibles llamados "fusibles de configuración" o fuses, que se configura en el programador PIC.

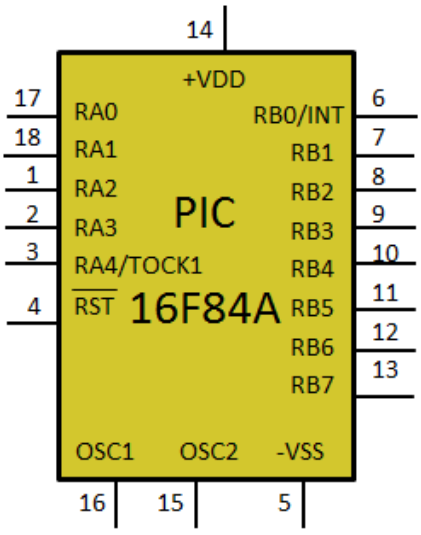
Si la aplicación que vamos a crear no es exigente con la precisión ni la estabilidad de la frecuencia, podemos optar por usar una red **RC**, Resistencia-Condensador, como circuito oscilador, para ello usaremos el siguiente esquema:



Los valores de **REXT** recomendados por **Microchip** son desde 5kΩ hasta 100 kΩ y para CEXT se recomienda mayor de 20 pF. Como se puede apreciar, el pin 15 del PIC queda libre, este pin entrega en su salida oscilaciones con frecuencia igual a 1/4 de la frecuencia del oscilador RC.

- Puertos de entradas y salidas

Los puertos de **entradas** y **salidas** del microcontrolador al exterior, enviamos e introducimos señales digitales TTL (5V) de forma que podemos comunicar el microcontrolador con el exterior.

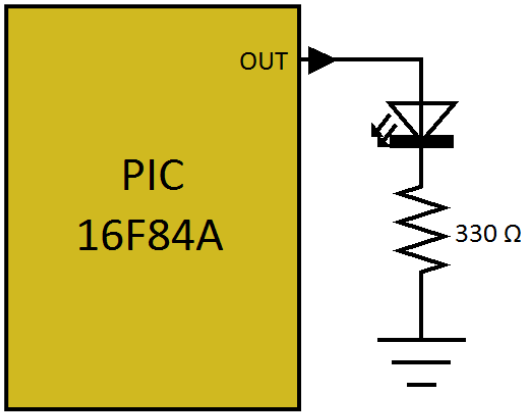


En el microcontrolador **PIC16F84A** posee 2 puertos de entrada y salida E/S. Sus nombres son **RA** y **RB**. El **puerto RA** consta de 5 líneas de entrada y salida: RA0, RA1, RA2, RA3 y RA4, un caso particular es RA4/TOCK1 que puede actuar como pin de entrada o como entrada de impulsos para un contador denominado **TMRO**.

El **puerto RB** consta de 8 líneas de entrada y salida que son RB0, RB1, RB2, RB3, RB4, RB5, RB6 y RB7. Cada línea del RA o del RB se puede configurar como entrada o salida mediante 2 registros llamados **TRISA** y **TRISB**.

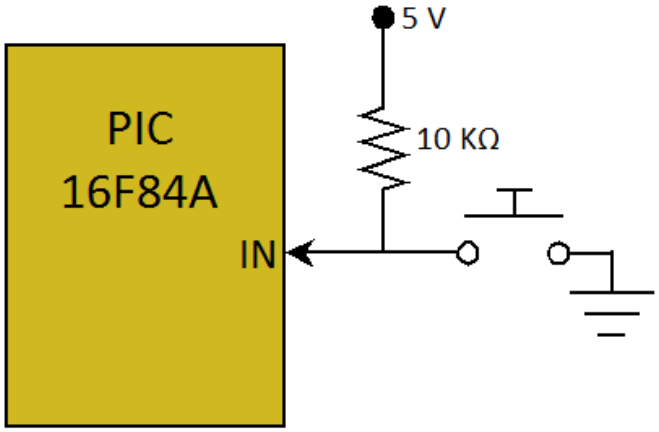
- ### Las líneas de salida

Si lo que queremos es solo testear el comportamiento del microcontrolador, basta con poner en cada salida un LED en serie con una resistencia.



- ### Las líneas de entrada

Si la entrada se da a través de un interruptor será obligatorio conectar una resistencia de 10 kΩ a VDD sin importar si la línea se usará o no, una solución sencilla para evitar la conexión de tantos resistores es configurar como salidas las líneas que no vayamos a usar en nuestro proyecto.

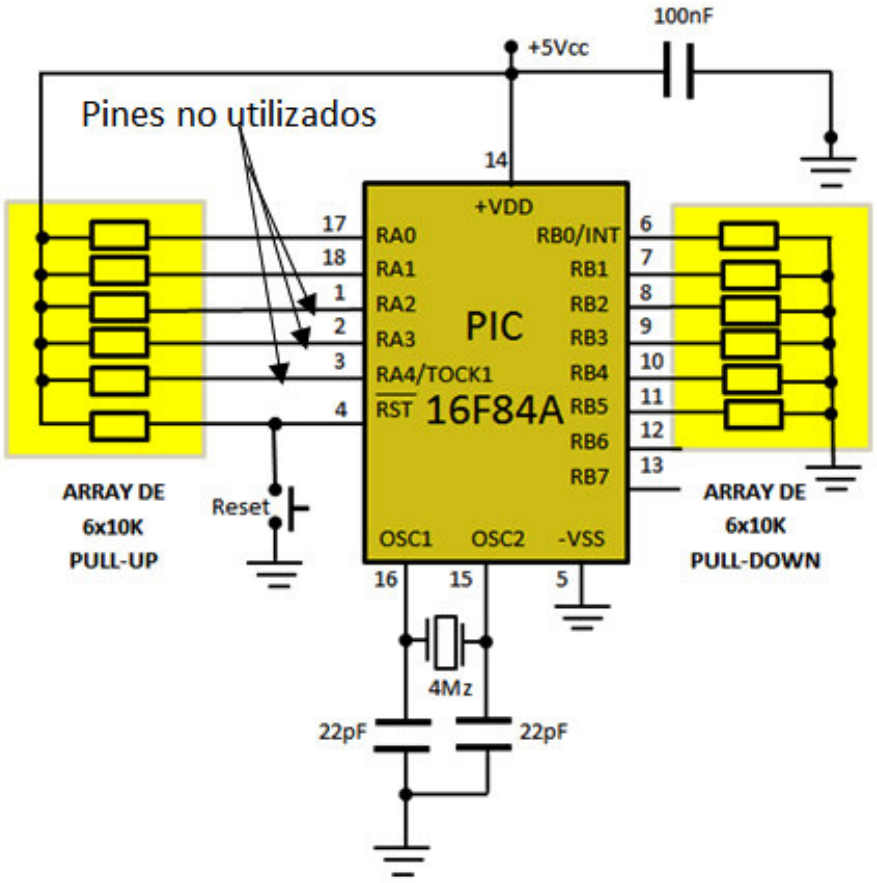


Resistencias de PULL-UP y PULL-DOWN

La electrónica digital maneja dos estados lógicos para su funcionamiento: el «1» y el «0». Supongamos que un circuito digital tiene una entrada y conectamos un interruptor entre dicha entrada y **VSS** (masa), analizando veremos que el cero ya está definido por el interruptor cuando este está cerrado, pero si el interruptor estuviera abierto no hay un estado definido porque la entrada no tiene contacto con ningún cero ni con ningún uno lógico (sería un pin sin conexión), esto se llama **estado flotante** y puede tener consecuencias como el mal funcionamiento del circuito. Esto se soluciona poniendo una resistencia entre el pin de entrada y **VDD** (+5v) llamada resistencia de **pull-up**, su objetivo es asegurar un «1» lógico cuando el interruptor esté abierto. Si lo que se desea es asegurar un «0» lógico, la resistencia se conecta a **VSS** (masa) y recibe el nombre de resistencia de **pull-down**.

Existen unos componentes que se denominan **ARRAYS** que contienen un grupo de resistencias del mismo valor que se conectan, por una parte, todas las patas de un extremo de la resistencia se unen y se conectan, bien al positivo de la fuente +VDD (**PULL-UP**) o al negativo –VSS (**PULL-DOWN**), el otro extremo de cada una de las resistencias van conectadas al puerto en cuestión del microcontrolador para polarizar el puerto y no dejarlo en estado flotante.

Cuando estos dispositivos sean de tecnología CMOS, todos los pines deben estar conectados a alguna parte, nunca dejarlos al aire porque se puede dañar el integrado. Los pines que no se estén usando se deben conectar a la fuente de alimentación de +5V, por medio de una resistencia de 10K, como se muestra en la siguiente figura.



Conexión del Pull-Up y Pull-Down

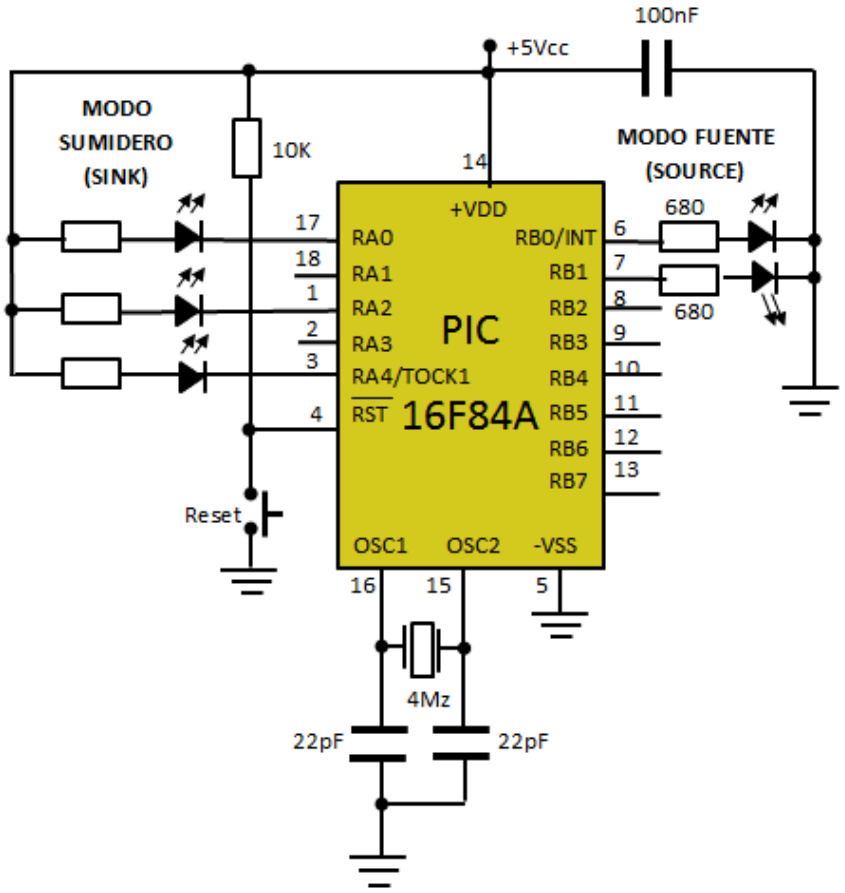
Nota: Se aconseja conectar cualquier línea de entrada a VDD mediante una resistencia de 10 kΩ, omitir este paso podría causar daños en el PIC.

- ### Capacidad de corriente en los puertos

La máxima capacidad de corriente de cada uno de los pines de los puertos en modo **sumidero** (sink) es de 25 mA y en modo **fuente** (source) es de 20 mA. La máxima capacidad de corriente total de los puertos es:

	PUERTO A	PUERTO B
Modo Sumidero	80 mA	150 mA
Modo Fuente	50 mA	100 mA

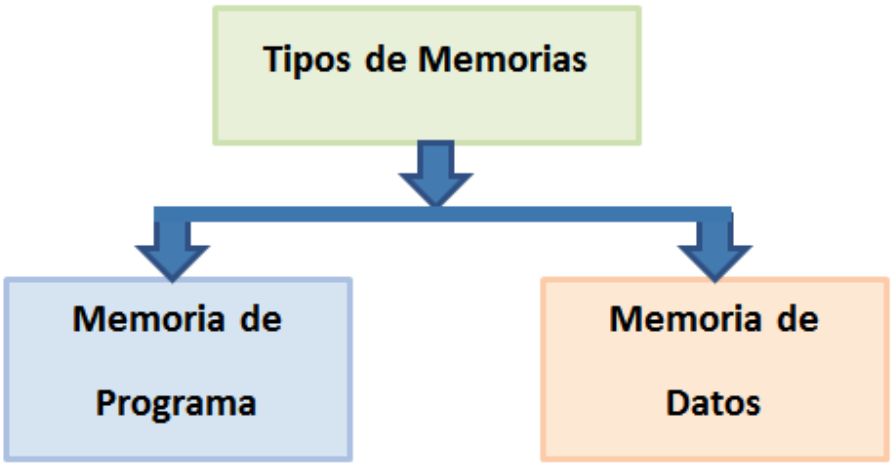
En el modo **Sumidero** el puerto proporciona una salida negativa para activar la carga y en el modo **Fuente** el puerto proporciona una salida positiva para activar la carga. Así se vería la conexión para ambos modos de funcionamiento.



Conexión del modo sumidero y modo fuente

- Memorias

Todo dispositivo programable necesita de una memoria para poder almacenar el programa, poder manejar variables y almacenar datos.

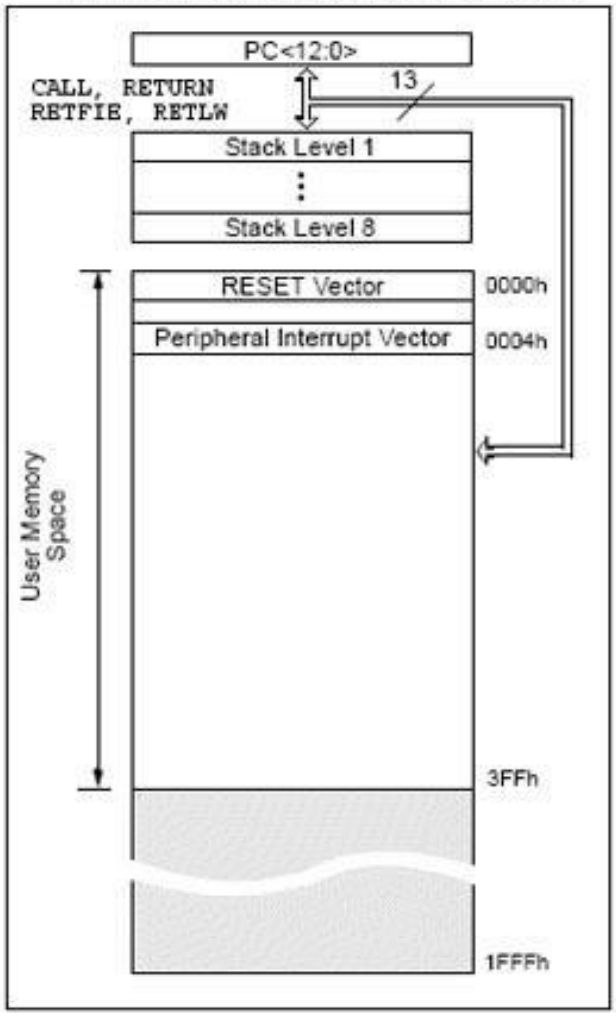


- Memoria de programa

Esta sería la memoria de instrucciones, aquí es donde almacenaremos nuestro programa o código que el micro debe ejecutar. No hay posibilidad de utilizar memorias externas de ampliación. Son 5 los tipos de memoria. pero sólo describiré dos:

- **Memorias EEPROM.** (Electrical Erasable Programmable Read Only Memory) Es una memoria de sólo lectura Programable y borrable eléctricamente. Ésta tarea se hace a través de un circuito grabador y bajo el control de un PC. El número de veces que puede grabarse y borrarse una memoria **EEPROM** es finito aproximadamente 1000 veces. Este tipo de memoria es relativamente lenta.
- **Memorias FLASH.** Disponible en el PIC16F84. Posee las mismas características que la EEPROM, pero ésta tiene menor consumo de energía y mayor capacidad de almacenamiento, por ello está sustituyendo a la memoria **EEPROM**.

FIGURA 2-1: MAPA MEMORIA PROGRAMA Y STACK - PIC16F84A



En el **PIC16F84A** la memoria de programa o de instrucciones es una memoria tipo **FLASH** de 1024 instrucciones "1K". En esta memoria se graban las instrucciones del programa. Una por una, como en renglones de un cuaderno. Cada renglón es una instrucción, otro renglón otra instrucción. La cantidad de renglones disponibles está limitada por la capacidad de la memoria. En el PIC16F84 contamos con 1024 renglones para escribir en cada uno de ellos las instrucciones del programa.

A cada instrucción le asignamos un número consecutivo: del 0 al 1023, a este número le llamamos dirección. Así podríamos decir, por ejemplo, que la instrucción número 61 realiza un bucle.

Dirección (No. De ins-truccion)	Memoria del programa Instrucciones ↓
1	
2	
3	
4	
1023	

Memoria del programa del PIC16F84
Memoria Flash de 1024 instrucciones

En esta memoria almacenaremos el programa que ejecutará el microcontrolador que al ser una memoria **FLASH** se puede borrar, grabar o reprogramar las veces que sea necesario. El programa que editamos, compilamos y que finalmente grabamos en el PIC, mediante un programador se almacena en esta memoria.

Existe un registro especial llamado **contador de programa** "**PC**" cuya finalidad es avanzar por las instrucciones del programa de forma secuencial excepto cuando se encuentran instrucciones de salto.

Memoria de datos

La memoria de datos sirve para almacenar variables, leer puertos de entrada o escribir en los puertos de salida, por ejemplo, el valor de un contador que va cambiando según el número de veces que se activa un interruptor o el tiempo que dura un proceso. Podemos también acceder al temporizador o al registro **EEPROM**.

La memoria de datos es dividida en dos bancos que contienen los Registros de Propósito General y los Registros de Función Especial. El banco 0 se selecciona por el aclarado (0) del bit de RP0 (STATUS<5>). Al activar el bit RP0 se selecciona el Banco 1. Cada Banco se extiende hasta 7Fh (128 bytes). Las doce primeras posiciones de cada Banco son reservadas para los Registros de Función Especial. El resto, son Registros de Propósito General, implementados en la práctica como la RAM estática.

La memoria de datos está formada por dos zonas diferentes:

- **MEMORIA RAM** de 68 registros
- **MEMORIA EEPROM** de 68 registros cuya característica principal es que no se perderán los datos cuando se desconecta la alimentación.

RAM estática ó SRAM: donde residen los Registros Específicos (SFR) con 24 posiciones de tamaño byte, aunque dos de ellas no son operativas y los Registros de Propósito General (GPR) con 68 posiciones. La RAM del PIC16F84A se encuentra dividida en dos bancos (banco 0 y banco 1) de 128 bytes cada uno (7Fh). Ver figura 2-2.

La principal característica de esta memoria llamada **RAM** es que es volátil. Es decir cuando el PIC se desconecta, esta memoria pierde sus valores.

EEPROM: de 64 bytes donde, opcionalmente, se pueden almacenar datos que no se pierden al desconectar la alimentación.

El **PIC16F84A** tiene 80 “renglones” de memoria de datos que están numerados del 0 al 79. Cada uno de estos registros (renglones) tiene 8 bits. En cada bit podemos escribir/leer un “0” o un “1”.

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

Dir.	Memoria de datos
0	7 6 5 4 3 2 1 0
1	
2	
3	
4	
5	Puerto A
6	Puerto B
7	
8	
9	
10	
11	
12	
13	
14	
77	
78	
79	

Memoria de datos del PIC16F84

- **Registro de propósito general GPR**

Cada Registro de Propósito General (**GPR**) es de 8 bits de ancho y es direccionado directa o indirectamente por los SFR. Las direcciones de **GPR** en el Banco 1 son mapeadas a direcciones en el Banco 0. Como un ejemplo, direccionando la posición 0Ch u 8Ch tendremos acceso al mismo GPR.

- **Registro de función especial SFR**

Los Registros de Función Especial **SFR** (Figura 2-2 y Tabla 2-1) son usados por la **CPU** y funciones periféricas para controlar la operación del dispositivo. Estos registros son la **RAM estática**. Los registros de función especial pueden ser clasificados en dos juegos, central y periférico.

FIGURA:2-2 MAPA REGISTROS PIC16F84A

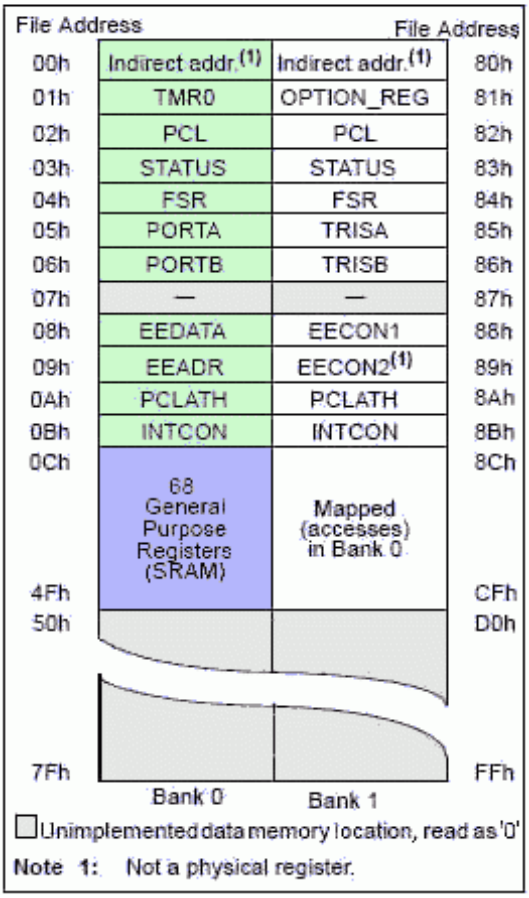


TABLA 2-1: SUMARIO DE LOS REGISTRO DE FUNCIÓN ESPECIAL

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on RESET	Details on page
Bank 0											
00h	INDF	Uses contents of FSR to address Data Memory (not a physical register)								---- ----	11
01h	TMR0	8-bit Real-Time Clock/Counter								xxxx xxxx	20
02h	PCL	Low Order 8 bits of the Program Counter (PC)								0000 0000	11
03h	STATUS(2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	8
04h	FSR	Indirect Data Memory Address Pointer 0								xxxx xxxx	11
05h	PORTA(4)	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	16
06h	PORTB(5)	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	18
07h	—	Unimplemented location, read as '0'								—	—
08h	EEDATA	EEPROM Data Register								xxxx xxxx	13,14
09h	EEADR	EEPROM Address Register								xxxx xxxx	13,14
0Ah	PCLATH	—	—	—	Write Buffer for upper 5 bits of the PC(1)					---0 0000	11
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	10
Bank 1											
80h	INDF	Uses Contents of FSR to address Data Memory (not a physical register)								---- ----	11
81h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	9
82h	PCL	Low order 8 bits of Program Counter (PC)								0000 0000	11
83h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	8
84h	FSR	Indirect data memory address pointer 0								xxxx xxxx	11
85h	TRISA	—	—	—	PORTA Data Direction Register					---1 1111	16
86h	TRISB	PORTB Data Direction Register								1111 1111	16
87h	—	Unimplemented location, read as '0'								—	—
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	13
89h	EECON2	EEPROM Control Register 2 (not a physical register)								---- ----	14
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC(1)					---0 0000	11
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	10

4. INTRODUCCIÓN A LA PROGRAMACIÓN

En los ordenadores, un programa es una secuencia de pasos que el ordenador (o microcontrolador) debe realizar para resolver algún problema o necesidad, la metodología usada para resolver este tipo de problemas recibe el nombre de **algoritmo**. A menudo, un problema podrá resolverse de diferentes maneras empleando diferentes algoritmos, por eso cuando dos o más personas intentan resolverlo por su propia cuenta, suelen obtener programas diferentes pero con la misma función.

- **Etapas de la programación**

Hay unas etapas bien definidas en lo que se refiere a realizar e implementar un programa, que se muestra en la siguiente figura.

Etapas de la programación

1. Planteamiento del problema o necesidad: el objetivo de escribir un programa es dar solución a algo usando un ordenador o microcontrolador, por ejemplo, si deseamos hacer que un LED encienda y apague consecutivamente ya tenemos nuestro problema planteado.

2. Diseño de la solución: las ideas que se nos ocurran para dar solución a lo que queremos realizar se llaman algoritmos, podemos expresar estos algoritmos con diagramas de flujo o con secuencias de instrucciones llamadas *pseudocódigo*. Siguiendo con el ejemplo anterior, para lograr nuestro objetivo con el LED un algoritmo general sería el siguiente:

- Encender el LED
- Mantenerlo así por un espacio de tiempo
- Apagar el LED
- Mantenerlo apagado otro espacio de tiempo
- Repetir de nuevo los pasos

Este es un algoritmo que describe los pasos que se deben seguir para tener un LED «parpadeante», descripciones como estas son llamadas pseudocódigo, pues es una primera aproximación a lo que será el código definitivo del programa. Hasta ahora, el problema no está solucionado, este algoritmo se debe escribir en un lenguaje de programación.

3. Escritura del código fuente (lenguaje de programación): esta es una etapa muy importante, de aquí depende gran parte del funcionamiento del dispositivo que se programará. A través de un computador, y usando un lenguaje de programación se escribirán formalmente las sentencias que el microcontrolador deberá realizar.

Existen diversos tipos de lenguajes de programación, pero básicamente tenemos los de **bajo nivel** y los de **alto nivel**. Un lenguaje de bajo nivel describe las instrucciones casi de la misma forma como las realizará la máquina; un lenguaje de alto nivel es más abstracto y cada instrucción escrita puede traducirse en muchas instrucciones para la máquina sin que podamos percibirlo al programar.

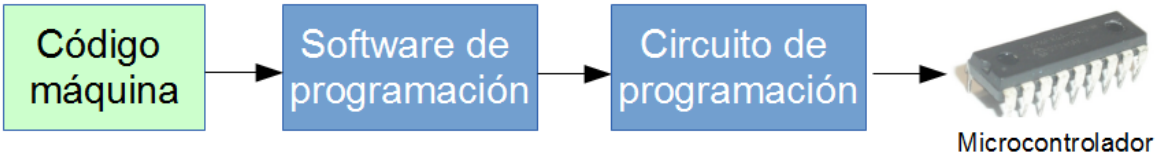
Un ejemplo aplicable a microcontroladores de lenguaje de bajo nivel es **ensamblador** cuyo código fuente se guarda con extensión *.asm; un ejemplo de lenguaje de alto nivel es **C**, en este caso su código fuente se guarda con extensión *.c.

Nota: Todo programa debe tener un inicio y un final en su código fuente.

4.Compilación (objeto): en esta etapa, un software se encarga de convertir las instrucciones dadas en el lenguaje de programación al lenguaje propio del dispositivo que se va a programar, para el caso de un microcontrolador, el lenguaje al que se traduce es el *hexadecimal*, el archivo obtenido también recibe el nombre de objeto y se guarda con extensión *.hex.

Proceso de compilación

5. Implementación: si nos refiriéramos a un computador sería el proceso de instalación del programa; para el caso de un microcontrolador lo llamaríamos «quemado» y consiste en transferir el programa desde un PC hacia el micro, para hacer esto posible se necesita de un circuito intermedio llamado *circuito programador*.



Proceso de «quemado» de un microcontrolador

6. Verificación: consiste en comprobar si el dispositivo ha cumplido con los requisitos, si no es así habrá que regresar hacia la etapa de diseño para realizar las correcciones o ajustes.

Nota: Puede incluirse una etapa adicional que es la simulación, aunque omitirla no impide la programación puede resultarnos muy útil en muchos casos.

- **Estructura de un código fuente**

Cada programa debe tener un inicio y un fin bien determinados. Las variables y las constantes derivan de los datos y condiciones del problema a resolver, las operaciones se encargan de hacer los procesos y comunicar los resultados.

Cuando se hace un programa, se especifica cómo se hará uso de las memorias disponibles en el microcontrolador, hay una memoria encargada de guardar las instrucciones llamada **memoria de programa**; otra memoria, llamada **memoria de datos**, que se encargará de guardar cada resultado que surja a medida que el programa corre. Como las memorias tienen un límite de capacidad, siempre hay que estar atento de no excederlo.

Luego por lo tanto el código fuente es el lenguaje **ensamblador** utilizado para programar nuestro microcontrolador y nos permitirá estudiar mejor al micro y predecir mejor su comportamiento; cuando necesitemos hacer programas muy complejos pasaremos al lenguaje *C*.

Para comenzar a escribir un programa en lenguaje ensamblador, lo primero que debemos saber es como deben de ir colocados las operaciones, instrucciones y comentarios.

En cada renglón se deben diferenciar **cuatro columnas separadas** poniendo un espacio en blanco entre ellas, en la primera columna se escriben las **etiquetas** que asignamos a determinada línea del código, la segunda columna es la **instrucción**, la tercera columna es el **registro**, bit o dato al que se le aplica la instrucción y en la cuarta los **comentarios**. La estructura de cada renglón escrito quedaría así:

```
Etiqueta    Instrucción    Registro/Bit/Dato    ; comentario
```

No siempre se ponen etiquetas a cada renglón, en ese caso habrá que dejar la columna vacía pero se debe respetar el espacio en blanco antes de escribir la instrucción así:

```
            Instrucción    Registro/Bit/Dato    ; comentario
```

Cuando escribimos un programa es muy importante agregar comentarios que nos guíen cada vez que queremos estudiar o modificar el código, en ellos también podemos agregar información adicional, por ejemplo, la conexión de los pines del microcontrolador. Los comentarios se escriben precedidos de un punto y coma (;) así:

```
Etiqueta    Instrucción    Registro/Bit/Dato    ; comentario
```

Ejemplo:

INICIO

```
;-------------------------------------------------
; Programa para control  de depósito
; -------------------------------------------------
        LIST P=16f84A
```

DECLARACION DE VARIABLES

```
        status EQU  h'03'
        porta  EQU   h'05'
        portb  EQU   h'06'
        ORG   0
```

OPERACIONES

```
; ---configuramos porta y portb -----
        BSF             status,5            ;accedemos banco 1
        MOVLW           b'00000011'         ;entradas
        MOVWF           porta               ;al puerto A
        MOVLW           b'00000000'         ;salidas
        MOVWF           portb               ;al puerto B
        BCF             status,5            ; accedemos al banco 0
;------ programa principal ------
inicio  MOVF            porta,0             ;leemos portA
        ANDLW           b'00000011'         ;solo interesa las 2 primera entradas
        MOVWF           portb               ;pasamos los datos a la salida
        GOTO            inicio              ;bucle
```

FIN

```
        END                                 :fin del programa
```

5. PROGRAMACIÓN EN ENSAMBLADOR

En el capítulo anterior hemos visto una introducción a la programación, especialmente en las etapas de la programación y la estructura del código fuente.

Para programar un microcontrolador necesitamos conocer las instrucciones para generar el código fuente para posteriormente compilarlo en hexadecimal y poderlo grabar en el microcontrolador.

En la siguiente tabla se muestra el juego de 35 instrucciones que el fabricante **Microchip** utiliza para la programación del microcontrolador en ensamblador. Cada instrucción en ensamblador le corresponde un código máquina hexadecimal de 14 bit. Éstos operandos al compilarlos se traduce en código máquina hexadecimal que es el que entiende el microcontrolador.

TABLE 7-2: JUEGO DE INSTRUCCIONES DEL PIC16CXXX

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb	LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS							
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff		C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff		Z	1,2
CLRF	f	Clear f	1	00 0001 lfff ffff		Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx		Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff		Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff		Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1 (2)	00 1011 dfff ffff			1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff		Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1 (2)	00 1111 dfff ffff			1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff		Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff		Z	1,2
MOVWF	f	Move W to f	1	00 0000 lfff ffff			
NOP	-	No Operation	1	00 0000 0xx0 0000			
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff		C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff		C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff		C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff			1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff		Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS							
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff			1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff			1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff			3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff			3
LITERAL AND CONTROL OPERATIONS							
ADDLW	k	Add literal and W	1	11 111x kkkk kkkk		C,DC,Z	
ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk		Z	
CALL	k	Call subroutine	2	10 0kkk kkkk kkkk			
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100		$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk			
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk		Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk			
RETFIE	-	Return from interrupt	2	00 0000 0000 1001			
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk			
RETURN	-	Return from Subroutine	2	00 0000 0000 1000			
SLEEP	-	Go into standby mode	1	00 0000 0110 0011		$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk		C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk		Z	

Tabla de juego de instrucciones de los PIC 16C/FFXX

Un listado en código fuente es un fichero **ASCII** con extensión **ASM** que está formado por 4 columnas para identificar las diferentes funciones:

- Etiquetas: Dan nombre a determinadas partes del programa (hasta 32 caracteres)
- Instrucciones: Son las instrucciones que se pasan al microcontrolador o una DIRECTIVA al ensamblador.
- Datos: Datos u operandos para las instrucciones.
  - Registros (f)
  - Bits
  - Etiquetas
  - Número constante literal (L)
- Comentarios siempre después de poner el punto y coma ";" son descripciones para hacer más legible el listado.
- En la página 28 se muestra la tabla 2-1, que es un sumario de las funciones especiales de los registros de 8 bits.

Dentro del microcontrolador hay unos registros especiales que determinan algunas de las características notables del microcontrolador:

- Temporizador/Contador **TMR0**
- Perro guardián watch Dog (**WD**)
- Interrupciones.
- Reset (Reinicio del sistema **MCLR**)

- **Descripción del juego de instrucciones**

Seguidamente se hace una descripción detallada de las instrucciones que son generales para todos los PIC, además de las particularidades de cada instrucción, por orden alfabético, que presentan para cada micro tanto para el PIC16F84A como para el PIC12C508A.

NOTAS:

1- La palabra Literal significa «NÚMERO» como el número 9 o 16h. El número 16h es un número hexadecimal y en valores decimales esto representa el número: «veintidós».

2- Destino de la instrucción según el designador **OP**: Si el bit de código OP de la instrucción es 0, el destino es W y si es 1, el destino es el registro f, o sea, se selecciona el destino donde se guarda el resultado de una operación.

3- Cuando una instrucción termina con W o F, el destino del resultado será el registro W o el propio archivo f, se define con el designador '0' o '1' de la propia instrucción.

0 = W

1 = F

Por ejemplo:

ADDWF 1F,0 el resultado es almacenado en el registro de trabajo **W**.

ADDWF 1F,1 el resultado es almacenado en el mismo registro **(F**).

Aunque son muchas las instrucciones que se van a describir, es importante tenerla en cuenta para hacer una consulta y conocer su significado y, algún que otro ejemplo. A lo largo de este documento muchas de estas instrucciones que se utilizan en los ejemplos de programación se vuelven a describir.

ADDLW Esto significa: Agregar (sumar) el Literal al registro W (acumulador o registro de trabajo) resultado en W.

ADDLW 00 al FF Un número fijo (llamado literal) es sumado al registro W (registro de trabajo). El literal (número) puede estar comprendido entre el 00 y FF. En el registro STATUS se ven afectadas tres banderas (o flags) por la orden ADDLW (Z, DC y C):

- **C** Se pone a 1 si se produce un acarreo desde el bit de mayor peso (desbordamiento).
- **DC** Se pone a 1 si se genera un acarreo del bit 3 al bit 4.
- **Z** Se pone a 1 si el resultado de la operación es cero.

Esta instrucción no está disponible para el '508A. Si quiere usar esta instrucción en el '508A o un programa en él la requiere/contiene, emplee las 3 instrucciones siguientes:

Por ejemplo:

```
ADDLW 80          ; mueve 80h a W
```

ADDWF Esto significa: Suma aritmética de W y un archivo (f).

ADDWF 00 a 1F,0 El resultado es almacenado en el registro de trabajo W, debido al valor 0 en la instrucción.

ADDWF 00 a 1F,1 El resultado es almacenado en el mismo archivo, debido al valor 1 del designado en la instrucción.

ADDWF (sumar) el contenido del registro W con el contenido de un archivo. El resultado puede ser guardado en el registro W (designado = 0) o emplazado en el archivo llamado (designado = 1). Con la orden ADDWF, en el registro STATUS se ven afectados los bits: C (Carry), Z (Cero) y el DC (Dígito Carry).

Si el resultado de una instrucción ADD rebasa FF, la bandera C (Carry) es puesta a 1, si tiene cualquier otro valor es 0.

Si el resultado de una instrucción ADD es cero 0000 0000, la bandera Z (Cero) se pone a 1 y 0 si tiene cualquier otro valor.

La suma se realiza en aritmética binaria pura y sin signo. Si hay un (desborde) acarreo del bit 7, es decir que el resultado es mayor de 255, el bit C (bandera Carry) resulta 1, en caso contrario resulta 0. El PIC supervisa si hay acarreo del bit 3, es decir que, la suma de los dos (nibbles) mitades menos significativas (bits 0 a 3) resulta mayor de 15, el bit DC (digit carry) se pone a 1, en caso contrario se pone a 0.

Por ejemplo: Si agregamos 21h a 3Ch, el resultado es 5Dh, esto no afecta la bandera Carry, por lo que la bandera DC (dígito carry) será puesta a 1, pero si a 2Dh le agregamos 3Eh, el resultado es 6Bh, lo que desborda el contador (6B>FF) por lo que la bandera C (Carry) será puesta a 1.

ANDLW Esto significa: producto lógico AND del Literal con el registro W.

ANDLW 00 a FF El objetivo de la operación es, descubrir cuantos bits de L y W, en binarios están a 1. Si ambos bits son cero, el resultado es cero y la instrucción usada en este caso es XOR. Esta instrucción hace un AND lógico entre un número fijo (literal) y el contenido del registro W, el resultado es colocado en el registro W. Con la orden ANDLW, en el registro STATUS se ven afectados los bits: C (Carry), Z (Cero) y el DC (Dígito Carry). El literal puede ir de 00 a FF.

La operación AND puede decirse que se usa para enmascarar (separar o quitar) los bits que nos interesen

ANDWF Esto significa: Operación AND producto lógico de **W** con el archivo f [AND'ed entre W y f].

ANDWF 00 a 1F,0 El resultado se almacena en el registro **W**, por el valor 0 en la instrucción.

ANDWF 00 a 1F,1 El resultado se almacena en el archivo f, por el valor en la instrucción.

Al **registro W** se le aplica el producto **AND** con un archivo **f**. Como dos archivos juntos no se pueden operar (AND), un archivo debe ponerse en W y se hace AND con el segundo archivo. Véase arriba para hacer operaciones de enmascarar con la operación AND. Con la instrucción ANDWF, sólo se afecta la bandera Z (cero). Si la respuesta es cero, la bandera Z en el registro STATUS se pone a 1, si la respuesta es distinta de cero, la bandera se pone a 0.

BCF Esto significa: Bit Clear File (pon a «0» o aclara el bit indicado (detrás de la coma) en el archivo f). Ver también BSF.

BCF 00 a 1F,bit Hay sobre 300 instrucciones (incluidas en los micros, trabajando internamente) para esta orden. Hay 79 archivos en el PIC16F84A, los 13 primeros archivos se llaman Registros de Función Especial (SFR's), el resto (66) se llaman Archivos de Propósito General (GPR's) del 0Ch a 4Fh. No afecta los bits de STATUS.

BCF se usa para limpiar [**clear**] un bit (pone a 0 el bit identificado en el archivo f). Por ej.: BCF 06h,5 significa que el bit 5 del archivo 06 debe ser puesto a «0» (aclarado), el resto de bits no se influyen. Ver figura de la derecha. El archivo 6 contiene líneas de E/S comúnmente se llaman I/O del puerto.

BSF Esto significa: Bit Set File (poner a 1 el bit indicado, en el archivo f).

BSF 00 a 1F,bit Hay casi 300 instrucciones para esta orden. Hay 79 archivos en el PIC16F84A, de las que los primeros 13 son los SFR's y los siguientes 66 son los conocidos GPR's. Estos GPR's ocupan del 0Ch al 4Fh y cada uno tiene 8 bits. BSF significa poner a 1 lógico el bit especifico en el archivo f. No afecta los bits de **STATUS**.

Por ejemplo: BSF 06h,5 indica que el bit 5 del archivo 6 será puesto a 1, este archivo 6 contiene 8 líneas E/S, comúnmente se llaman líneas I/O del puerto. El resultado es la inversa a la instrucción anterior, el 0 se sustituye por un 1.

BTFSC Esto significa: Bit Test, Skip if Clear (Bit de Test, Salta si es «0»).

BTFSC 00 a 1F,bit Hay casi 300 instrucciones para esta orden, para cubrir los 79 archivos, cada uno con 8 bits. BTFSC significa, comprobar el bit identificado en el registro llamado y si es 0 saltar una instrucción (no ejecuta la instrucción que sigue). No afecta los bits de **STATUS**.

```
Etiqueta:   BTFSC 06h,4      ; comprueba si el bit 4 es 0
GOTO        Etiqueta2        ; si no es 0, salta hasta Etiqueta2
            CALL Dlay        ; si es 0, llama a subrutina Dlay
```

En el ejemplo anterior **BTFSC 06h,4** es la forma de comprobar si el bit 4 en el archivo 6 es «0», si es cero, salta la próxima instrucción (pasar sin ejecutar) y continuar con la posterior.

BTFSS Esto significa: Bit Test, Skip if Set (Bit de Test, Salta si es «1»).

BTFSS 00 a 1F, bit También hay casi 300 instrucciones para esta orden, para cubrir los 79 (4Fh) archivos, cada uno con 8 bits. BTFSS significa, comprobar el bit identificado en el registro llamado y salta la próxima instrucción si es 1. No afecta los bits de STATUS.

En **BTFSS 3,2** comprobamos el bit 2 del registro 3 y si dicho bit es 1, salta la próxima instrucción, si no, continua con la siguiente.

```
Etiqueta: BTFSS 03h,2 ; comprueba si el bit 2 es 1
GOTO Etiqueta2 ; si no, va a Etiqueta2
CALL Dlay ; si es 1, llama a subrutina Dlay para seguir. ; si es 1 viene a esta instrucción y sigue.
```

CALL Esto significa: Llamada incondicional a subrutina.

En un programa, esto se escribe como: **CALL Salida** o **CALL Tono1**, etc. donde Salida o Tono1 son etiquetas. Un **RETURN** debería encontrarse al final de la subrutina CALL para que el micro vuelva a la siguiente instrucción después de la instrucción CALL que la llamó, de lo contrario se producirá un desborde de pila, con el consiguiente bloqueo del programa. No afecta los bits de STATUS.

CALL Etiqueta Los programas deberían ser escritos de forma que las pocas primeras instrucciones pongan al micro en el Inicio de Programa Principal. El «Programa Principal»

se localizará físicamente al final del listado y éste puede estar en el final de la memoria o a mitad del camino, si el programa es aproximadamente 250 bytes de largo.

Después de las primeras instrucciones que llevan al micro a: GOTO Inicio, se colocan todas las subrutinas necesarias para el programa. Con una orden CALL se llamará a las subrutinas y al final de cada subrutina debe tener una instrucción RETURN. Una llamada remota puede hacer de subrutina pero esta segunda subrutina no puede llamar otra subrutina.

Cada vez que se ejecuta una instrucción CALL, un valor de dirección es colocado (empujado) en la Pila (Stack), entonces el micro sabe dónde volver después de ejecutada la subrutina, la Pila sólo puede tener 8 niveles de alto, entonces es necesario llevar cuidado para no quedarse sin Pila (Stack).

```
Loop2: BTFSS 05,2       ;¿esta apretado el pulsador?
       GOTO Loop2       ;No. Salta a Loop2
       MOVLW 01         ;SÍ.
       XORWF 06,1       ; encender el LED
       CALL Delay       ;Llamada a rutina de Retardo
       GOTO Loop1       ; para ver LED encendido.
Delay: DECFSZ 1Bh,1     ; Subrutina anidada de Retardo
       GOTO Delay       ; retardo exterior
       DECFSZ 1Ch,1     ; retardo interior
       GOTO Delay

       RETURN
```

CLRF Esto significa: Clear f [Limpia f] (poner a 0 los 8 bits del archivo f)

CLRF 00 a 1F El contenido de un archivo se pone a 0 (Clear) y el bit Z del registro STATUS es puesto a 1, esto es, los ocho bits se ponen a «0». Por esto hay que tener en cuenta el bit Z (cero, flag Z). Los 12 primeros archivos son SRF's y los siguientes 68, del 07h al 4Fh son los llamados GPR's.

CLRW Esto significa: Clear W (limpiar el registro de trabajo – llamado acumulador en otros micros)

CLRW El **registro W** (de trabajo) es aclarado, todos los bits son puestos a 0. Cuando el W es puesto a 0, la bandera cero (flag Z) es puesta a 1, en otras palabras la bandera Z del registro STATUS es puesta a 1.

CLRWDT Esto significa: Limpiado (puesta a 0) del Temporizador Perro Guardián (El bit WDT = 0).

CLRWDT La instrucción repone (resetea) el Temporizador Perro Guardián, esto también repone el preescaler del WDT y consume 2 ciclos máquina.

COMF Esto significa: Complemento del archivo f.

COMF 00 a 1F,0 El resultado estará en W por el valor 0 detrás de la coma.

COMF 00 a 1F,1 El resultado estará en f. El contenido f es complementado (los 0's se cambian a 1's y los 1's a 0's).

DECF Esto significa: Decremento del archivo f .

DECF 00 a 1F,0 El resultado estará en W. El contenido del archivo f es decrementado y puesto «W«.

DECF 00 a 1F,1 Aquí, el resultado estará en f. El contenido del archivo «f» es decrementado, significa que es deducido (tomado) 1 del archivo. Si el archivo es 0000 0000 esto da la vuelta a 1111 1111 (255) afectando a la bandera Z. Cuando el archivo es 0000 0001 y se ejecuta una instrucción DECF, el archivo pasa a 0000 0000 y la bandera Z (cero) del STATUS se pone a 1, en otro caso es 0.

DECFSZ Esto significa: DECrement f, Skip if Zero (Decrementa el archivo f y salta si es 0).

DECFSZ 00 a 1F,0 El resultado estará en W.

DECFSZ 00 a 1F,1 El resultado estará en f. La instrucción DECFSZ tiene muchos empleos.

Un empleo importante está en una subrutina de retardo. En esta rutina la instrucción **DECFSZ** crea un lazo en el que el micro es enviado a una instrucción por encima-del-programa y se ejecutará un juego de instrucciones una y otra vez, esta es una táctica de pérdida de tiempo para crear un retardo. No afecta al registro **STATUS**.

Cada vez que el micro llega a **DECFSZ**, el contenido del archivo es decrementado y si el archivo no es cero, se ejecutará la siguiente instrucción en el programa. La siguiente instrucción es normalmente **GOTO** y ésta envía de nuevo al micro por encima-del-programa. Por ejemplo:

```
ret:    DECFSZ 0Ch,0        ; Decrementa 0C y si es 0, salta una línea
        GOTO ret            ; no es 0, ejecuta esta línea
…                           ; si es 0, viene hasta aquí.
…                           ;sigue programa
```

GOTO Esto significa: Bifurcación Incondicional.

GOTO k **GOTO** es la bifurcación incondicional en la que el micro es enviado a la dirección especificada. Esta instrucción carga simplemente la constante k en el registro **PC** (contador de programa). Esta es la típica instrucción de salto incondicional a cualquier posición de la memoria de programa. La constante literal k es la dirección de destino del salto, es decir la nueva dirección de memoria de programa a partir de la cual comenzarán a leerse las instrucciones después de ejecutar la instrucción **GOTO**. No afecta al registro **STATUS**.

También se usa **$-n** o **$+n** donde **n** es el número de líneas que ha de, volver atrás o avanzar en el programa. Por ejemplo:

```
ret DECFSZ 0Ch,0 ; decrementa 0Ch, si es 0 salta 1 instrucción
GOTO $-1         ; No, vuelve 1 línea atrás. No requiere etiqueta.
…                ; Si, sigue programa
```

INCF Esto significa: Incrementar el archivo **f**.

INCF 00 a 1F,0 El resultado del incremento estará en **W**.

INCF 00 a 1F,1 El resultado estará en f. El contenido del archivo 'f' es incrementado, esto simplemente significa que se agrega 1 al archivo, si el archivo es 1111 1111 (255) esto da la vuelta a 0000 0000. Cuando el archivo es FFh y se ejecuta la instrucción **INCF**, el archivo pasa a 0000 0000 y la bandera Z (cero) es puesta a 1 en otro caso es 0.

INCFSZ Esto significa: INCrement f and Skip if 0 (Incrementar el archivo f y salta si es 0).

INCFSZ 00 a 1F,0 El resultado estará en **W**.

INCFSZ 00 a 1F,1 El resultado estará en f. Normalmente la función de decremento DECFSZ se usa para crear un retardo, pero también se puede usar un INCFSZ. No afecta al registro STATUS.

Esto trabaja así: Si el contenido de un archivo es incrementado y el resultado no es 0, entonces la siguiente instrucción es ejecutada con un GOTO una dirección anterior y ejecutará otro INCFSZ. Eventualmente el archivo será 1111 1111 y el próximo incremento lo devolverá a 0000 0000, el resultado será cero y la instrucción GOTO será ignorada, ejecutándose la siguiente instrucción.

IORLW Esto significa: Inclusive OR Literal con W.

IORLW 00 a FF El contenido del archivo W es sumado (lógico) [OR'ed] con un número. El resultado es colocado en registro de trabajo W, el número literal puede ser desde 00 a FF. Afecta al bit Z del registro STATUS.

Esto es simplemente una operación OR (suma lógica) y el objeto de su realización es cambiar dos o más bits a «1», si un bit es ORed con 0, la respuesta no se altera, si el bit es ORed con 1, el resultado es 1.

Ejemplo: Si el registro W se carga con 1111 0000 (es una máscara de 4 bits altos F0h) y un número como 0100 1110 (4Eh) es ORed con W, el resultado es 1111 1110 (FEh).

IORWF Esto significa: Inclusive OR con el archivo f

IORWF 00 a 1F,0 El resultado estará en W.

IORWF 00 a 1F,1 El resultado estará en f. El contenido del registro W es ORed con el archivo f, esto simplemente es una operación «OR» y el objeto de su realización es cambiar dos o más bits a «1». Si un bit es ORed con 0, la respuesta no se altera, si el bit es ORed con un 1 el resultado es 1. Afecta al bit Z del registro STATUS.

Ejemplo: Si el registro W es cargado con 1111 0000 (F0h es una máscara de 4 bits altos) y un archivo con un número como 0100 1110 (4Rh) es ORed con W, el resultado es 1111 1110 (FEh).

MOVF Esto significa: Mueve el contenido del archivo 00 a 1F dentro y fuera del archivo o al W.

MOVF 00 a 1F,0 El contenido del archivo es movido al W. El resultado estará en W.

MOVF 00 a 1F,1 El resultado estará en f. Para esta instrucción MOVF 00 a 1F,1 el contenido es movido fuera del archivo y devuelto a él otra vez, pero no entra en W. Esto es una prueba útil ya que la bandera Z (cero) del STATUS se ve afectada. Si el contenido es cero, la bandera Z es puesta a 1, si el contenido es 1, la bandera Z es 0.

MOVFW Esta forma de instrucción, no es válida (no se recomienda su uso), a pesar de que el propio MPLAB la admita, significa mover el contenido del archivo F, al registro de trabajo W. Cando se encuentre esta forma de expresión, debe modificarse por:

MOVF f,W donde f es un registro entre 00 y FF. También puede usarse MOVF f,0 que viene a ser lo mismo.

MOVLW Esto significa: Mueve Literal a **W**.

MOVLW 00 a FF Un número f (Literal) es cargado en el registro W. El Literal puede ser 00 a FF. No afecta al registro **STATUS**.

MOVWF Esto significa: Copia **W** al archivo llamado f.

MOVWF 00 a 1F Esta instrucción copia datos del registro W al archivo f. No afecta al registro **STATUS**.

NOP Esto significa: Ninguna operación. Es decir, el micro no realiza ninguna operación, sólo consume el tiempo de 1 instrucción.

OPTION Esto significa: Carga registro **OPTION**. El contenido del registro W es cargado en el registro OPTION.

RETFIE Esto significa: Cuando hay una interrupción, **RETURN** con valor de lo alto de la Pila y lo deja en el PC.

RETFIE Carga el **PC** "Contador de Programa" con el valor que se encuentra en la parte superior de la pila, asegurando así la vuelta de la interrupción. Pone a 1 el bit **GIE**, con el fin de autorizar o habilitar de nuevo que se tengan en cuenta las interrupciones. TOS → PC, 1 → GIE. No afecta al registro **STATUS**.

RETLW Esto significa: **RETURN** con Literal en **W.**

RETLW 00 a FF El registro **W** es cargado con el valor del literal, normalmente devuelve un dato procedente de una tabla. El Contador de Programa (PC) es cargado de la cima

de la pila (Stack), que corresponde a la dirección de **RETURN** de programa. No afecta al registro **STATUS**.

RETURN Esto significa: Retorno de Subrutina. Esta instrucción está al final de una rutina o subrutina. No afecta al registro STATUS.

RLF Esto significa: Rotar el archivo f a Izquierda con aCarreo (Carry).

RLF 00 a 1F,0 El resultado se almacena en **W**.

RLF 00 a 1F,1 Resultado se almacena en f. El contenido de un archivo, es rotado un bit a la izquierda por la bandera Carry (acarreo), esto requiere de 9 desplazamientos para recuperar el valor original del archivo. Afecta al bit **C** del **STATUS**.

El uso de: **RLF**…..Reg, Destino……..; rota los bits de un registro un lugar la izquierda.

Si Reg = b'00010110′

Después de la instrucción: Reg = b'0010110C' donde C es el valor del bit STATUS,C en el momento de ejecutarse la instrucción **RLF**.

Veamos con más detalle cómo trabaja la función **RLF**:

Un grupo de 8 bits es registro, o sea: Registro = B7 B6 B5 B4 B3 B2 B1 B0

Al aplicar la instrucción RLF…..Reg,F ocurre que: (STATUS,C <== B7) <== B6 B5 B4 B3 B2 B1 B0 (C <== STATUS,C)

Esto significa que, todos los bits de Reg son rotados (desplazados) una posición hacia la izquierda. El espacio generado a la derecha de Reg es decir, el bit0 (B0) del registro, es ocupado por el valor que tenía en ese momento el bit C del registro STATUS. A su vez, el Bit7 (B7) de Reg sale del Registro y se rellena con el bit C del registro STATUS.

Repasemos otra vez. Reg = b'00001100′ (Ch = .12) y STATUS,C = 0

Aplicamos; RLF Reg,F Entonces: Reg = b'00011000′ (18h = .24) y STATUS,C = 0

Podemos comprobar que antes de aplicar la **RLF**, Reg valía 12 en sistema decimal. Después de la instrucción RLF Reg vale 24. Hemos multiplicado a Reg por dos, utilizando la instrucción RLF. Ahora, consideremos el siguiente caso:

Reg = b'00001100′ (Ch) y STATUS,C = 1

Aplicamos; RLF Reg,F Entonces: Reg = b'00011001' (19h = .25) y STATUS,C = 0

En este caso, antes de la aplicación de RLF Reg valía 12 en decimal y después de aplicar la instrucción Reg vale 25 en decimal, por qué ocurre este error si hemos aplicado la misma instrucción al mismo registro Reg. Debemos considerar el motivo.

El motivo radica en que el bit C del registro STATUS, antes de ejecutar la instrucción RLF, valía 1, en el segundo caso y ocupó el bit0 del Reg al ejecutar la instrucción RLF. Por tanto, en este segundo caso, al hacer RLF Reg,F equivale a hacer Reg * 2 + 1.

Precauciones a tener en cuenta para evitar incurrir en este error. Para asegurarnos en una multiplicación por dos, siempre limpiaremos el bit C del STATUS antes de aplicar la instrucción RLF, asegurándonos así que el bit STATUS,C no «corrompa» la operación.

- Configuración de los puertos

Para configurar los puertos del PIC es necesario conocer la tabla de registros de la memoria de datos, la cual como dijimos y vimos, está dividida en el **BANK 0** y **BANK 1**.

Los registros importantes en la configuración de los puertos son:

STATUS dirección 0x3
PORTA dirección 0x5
PORTB dirección 0x6
TRISA dirección 0x5
TRISB dirección 0x6

Por defecto el PIC tendrá todos los puertos de E/S, es decir los puertos **RA** y **RB**, colocados como entrada de datos, y si queremos cambiarlos habrá que configurarlos.
Al configurar los puertos deberás tener en cuenta que:

- Si asignas un **CERO** (0) a un pin, éste quedará como salida y...
- Si le asignas un **UNO** (1), quedará como entrada

Esta asignación se hace en:

- **TRISA** para los pines del PUERTO A (5 bits)
- **TRISB** para los pines del PUERTO B (8 bits)

Por ejemplo, si **TRISA** es igual a 11110 todos sus pines serán entradas salvo RA0 que esta como salida. Si **TRISB** es igual a 00000001 todos sus pines serán salidas salvo RB0 que esta como entrada.

Cuando el PIC arranca se encuentra en el **BANCO 0**, como **TRISA** y **TRISB** están en el **BANCO 1** no queda otra, deberemos cambiar de banco. Esto se logra a través del Registro **STATUS**. **STATUS** es un Registro de 8 bits u 8 casillas, en el cual el Nº 5 (RP0) define la posición del banco en donde nos encontramos.

Si pones un **CERO** (0) a **RP0** estaremos en el **BANCO 0**

Si le pones un **UNO** (1) estaremos en el **BANCO 1**

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

Ya están configurados los puertos, pero lo aclararemos con un ejemplo completo. Vamos a escribir un código que configure todos los pines del puerto A como entrada y todos los del puerto B como salida.

```
;--------------Encabezado------------
            list    p=16f84          ; usaremos el PIC 16f84
            radix   hex              ; y la numeración hexadecimal
;------------mapa de memoria---------
estado      equ     0x03             ; aquí le asignamos nombres a los
trisa       equ     0x05             ; registros indicando la posición
trisb       equ     0x06             ; en la que se encuentran
;-------Configuración de puertos-------
reset       org     0x00              ; origen del programa, aquí comienza
                                      ; siempre que ocurra un reset
            goto    inicio            ; salto a "inicio"
            org     0x05              ; origen del código de programa
inicio      bsf     estado,5          ; pongo rp0 a 1 y paso al banco1
            movlw   b'11111'          ; cargo W con 11111
            movwf   trisa             ; y paso el valor a trisa
            movlw   b'00000000'       ; cargo W con 00000000
            movwf   trisb             ; y paso el valor a trisb
            bcf     estado,5          ; pongo rp0 a 0 y regreso al banco0
;-----------------------------------------------

            end                       ; fin del programa
```

A continuación se explica el inicio, declaración de variables, las operaciones y fin del programa:

```
;--------------Encabezado------------
      list    p=16f84      ; usaremos el PIC 16f84

      radix   hex          ; y la numeración hexadecimal
```

Aquí le indica al ensamblador para que microcontrolador estas codificando (PIC16F84) y cuál será el sistema de numeración que utilizarás (hexadecimal).

```
;------------mapa de memoria---------
estado    equ       0x03          ; Aquí le asignamos nombres a los
trisa     equ       0x05          ; registros indicando la posición
trisb     equ       0x06          ; en la que se encuentran
```

Recuerda lo de la memoria de datos, el registro **STATUS**, que está en la posición 0x03 de la memoria de datos tiene la etiqueta "estado" **equ** es algo así como igual, es decir, se le está asignando el nombre estado al registro que está en la posición 0x03 de la memoria de datos. Y lo mismo se hizo con **trisa** y **trisb**.

```
;-------Configuración de puertos-------
reset   org     0x00                ; origen del programa, aquí comenzaré
                                    ; siempre que ocurra un reset
        goto    inicio              ; salto a "inicio"
        org     0x05                ; origen del código de programa
inicio  bsf     estado,5            ; pongo rp0 a 1 y paso al banco1
        movlw   b'11111'            ; cargo W con 11111
        movwf   trisa               ; y paso el valor a trisa
        movlw   b'00000000'         ; cargo W con 00000000
        movwf   trisb               ; y paso el valor a trisb
        bcf     estado,5            ; pongo rp0 a 0 y regreso al banco0
```

La directiva **org** indica el sitio de la memoria en donde se escribe una parte del programa. En este caso el contador de programa apuntará a la dirección 0x00 (reset) entonces ejecutará la instrucción que sigue a continuación, (saltar a la etiqueta inicio) y nuestro código de programa comienza en la dirección de memoria 0x05 (aquí salto por encima de la interrupción 0x04).

BSF (SET FILE REGISTER), es la instrucción que pone un uno en el bit del registro especificado, en este caso pone a uno el bit 5 del registro STATUS (el rp0), para pasar al banco 1.

movlw mueve el siguiente literal al Registro **W**.

W es el **Registro de Trabajo** y lo usamos para almacenar momentáneamente los datos que queremos mover. Una vez hecho esto pasamos el dato a **trisa**, o a **trisb**, según el caso.

movwf significa que mueva el contenido del registro **W** al registro f, en este caso f sería trisa o trisb.

BCF (BIT CLEAR FILE REGISTER), ésta instrucción limpia el bit del registro especificado, o lo pone a cero, en este caso pone a cero el bit 5 del registro STATUS para regresar al banco 0.

```
;-----------------------------------------
      end                ; fin del programa
;-----------------------------------------
```

A continuación se describe básicamente, algunos ejemplos de las formas que se realizan en la programación.

Por ejemplo para referirnos al registro 57 sería preferible llamarle "**Contador de Tiempo**". Así no tendríamos que recordar su dirección sino solamente su nombre. Esto se logra con una declaración que se coloca al principio del programa:

```
ContadorDeTiempo   equ   d'57'
```

Recuerda que en la programación no se admiten espacios en el interior del nombre y todos los números decimales se escriben entre apóstrofos y antecedido por la letra d. Por ejemplo 15 se escribe d'15'... 45 se escribe d'45'...

El nombre del registro lo escoge el programador. De este modo no tendrá que recordar el número de registro sino su nombre que esperemos este asociado con la función que tiene el registro en el programa.

Para referirnos a los bits de los registros seguiremos estos pasos:

```
ContadorDeTiempo,0      ; Se refiere al Bit0 del registro ContadorDeTiempo
ContadorDeTiempo,6      ; Se refiere al BIT6 del Registro ContadorDeTiempo
```

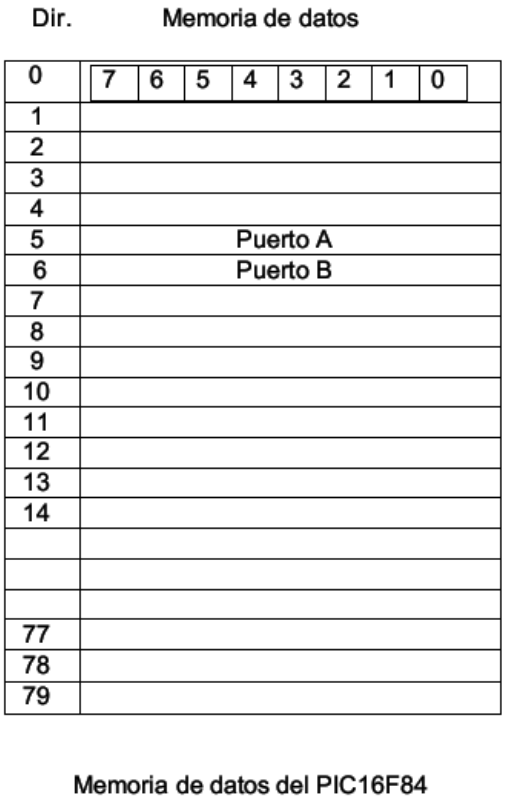
Otro ejemplo. Si declaramos al inicio del programa:

```
CuentaPiezas    equ    d'13'
CuentaPiezas,4          ; Se refiere al BIT4 del registro CuentaPiezas de la memoria RAM
CuentaPiezas,2          ; Se refiere al BIT2 del registro número 13, llamado CuentaPiezas
```

- ## Registros especiales

La memoria de datos (RAM) está dividida en dos grupos: a las primeras 12 direcciones se les llama "registros especiales" y al resto se les llama "registros de uso general".

Los **registros especiales** tienen usos muy particulares, pero dos de ellos es interesante que lo conozcamos y comentarlos seguidamente: el número 5 y el número 6, que tienen que ver con el **Puerto A** y el **Puerto B**.



Memoria de datos del PIC16F84

En la dirección **5** de la memoria de datos está el **Puerto A** de 5 bits:

No	No	No	RA4	RA3	RA2	RA1	RA0

En la dirección **6** de la memoria de datos está el **Puerto B** de 8 bits:

RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

Por lo tanto, en la dirección 5 está el Puerto A y en la dirección 6 está el puerto B, y se identifica en la programación como d'5' y d'6'.

Al igual que otras direcciones de la memoria RAM, mediante la declaración "**equ**" podremos asignar un nombre al Puerto A o al Puerto B, el nombre que le daremos a esta dirección en nuestro programa. Por ejemplo:

```
portA       equ     d'5'    ;significa que el Puerto A se llamará portA en nuestro programa
Rel_Salida  equ     d'6'    ;significa  que el Puerto B se llamará Rel_Salida
```

Hagamos las siguientes declaraciones al principio del programa:

```
PuertoA         equ   d'5'      ; declaramos el puerto A
PuertoB         equ   d'6'      ; declaramos el puerto B
                PuertoA,1       ;se refiere a RA1 es decir el pin 18
                PuertoB,7       ;se refiere a RB7 es decir el pin 13
```

- Operaciones orientadas a BITS

Para referirnos a estos registros, también llamados **direcciones de memoria**, lo podemos hacer por su dirección, número consecutivo, pero es más fácil que se les ponga un nombre. Estas tres instrucciones son: **clrf** (pone el registro a 0), **bsf** (pone a 1 bit) y **bcf** (pone a 0 el bit). Las tres instrucciones siguientes se aplican a los registros de la memoria de datos RAM, registros especiales y también de uso general. Veamos el siguiente ejemplo.

```
PortA       equ         d'5'        ; declaramos el puerto A
PortB       equ         d'6'        ; declaramos el puerto B
Contador    equ         d'13'       ; declaramos Contador
            clrf        PortB       ;pone en cero los 8 bits de Puerto B  RB7, RB6...RB0
            clrf        Contador    ;pone en cero los 8 bits de Contador
            bsf         PortA,1     ;pone en "1" RA1
            bcf         PortB,7     ;pone en "0" RB7
```

OPERACIONES ORIENTADAS A BITS			
Nemotécnicos		**Operación**	**Estados afectados**
BCF	f,b	Limpiar bit b de f	
BSF	f,b	Activar bit b de f	
BTFSC	f,b	Probar bit b de f, saltar si cero	
BTFSS	f,b	Probar bit b de f, saltar si uno	

- Acumulador

El acumulador es un Registro de 8 bits. También es llamado **registro de trabajo** (Work) se identifica con la letra **W**. El acumulador se usa esencialmente para:

- Mover el contenido de un registro a otro
- Para inicializar un registro con un valor determinado
- Para realizar alguna operación lógica o aritmética.

Supongamos que queremos cargar el Puerto B con unos en sus ocho bits, tenemos que pasar forzosamente por el acumulador: "carga el acumulador con unos y mueve el contenido del acumulador al Puerto A".

OPERACIONES ORIENTADAS A REGISTROS			
Nemotécnicos		**Operación**	**Estados afectados**
ADDWF	f,d	Sumar W y f	C,DC,Z
ANDWF	f,d	AND entre W y f	Z
CLRF	f	Limpiar f	Z
CLRW		Limpiar W	Z
COMF	f,d	Complementar f	Z
DECF	f,d	Decrementar f	Z
DECFSZ	f,d	Decrementar f, saltar si cero	
INCF	f,d	Incrementar f	Z
INCFSZ	f,d	Incrementar f, saltar si cero	
IORWF	f,d	OR entre W y f	Z
MOVF	f,d	Mover f	Z
MOVWF	f	Mover W a F	
NOP		No Operación	
RLF	f,d	Rotar a la izquierda a través del carry	C
RRF	f,d	Rotar a la derecha a través del carry	C
SUBWF	f,d	Restar W de f	C,DC,Z
SWAPF	f,d	Intercambiar nibbles de f	
XORWF	f,d	OR exclusiva entre W y f	Z

En esta tabla aparecen **W**, **f** y **d**. Recuerda que...

W : es el registro de trabajo y almacena datos de forma momentánea

f : es la dirección de un registro, si es llamada apunta al contenido de ese registro

d : es el destino donde se guarda el resultado de una operación, si es 1 se guarda en el registro f, y si es 0 en W.

○ Instrucciones para el acumulador

Las dos instrucciones siguientes se aplican al acumulador y a la memoria de datos: **movlw**, **movwf**

```
PuertoA     equ     d'5'              ; declaramos el puerto A
PuertoB     equ     d'6'              ; declaramos el puerto B
            clrf    equ  PuertoB      ; pone a 0 los 8 bits del puerto B
            movlw   b'11111111'       ; carga en el acumulador de bits b'11111111'
            movwf   PuertoB           ; mueve el contenido b'11111111' al puerto B
```

b'11111111' es la manera de representar el patrón de bits de un registro, con los ocho bits entre apostrofes precedidos por la letra b.

movlw quiere decir: "carga en el acumulador el patrón de bits siguiente"

```
movlw   b'11111111'   ; carga en el acumulador el patrón de bits b'11111111'
```

movwf quiere decir: "transfiere el contenido del acumulador al registro ..."

```
movwf   PuertoB   ; transfiere el contenido del acumulador que era b'11111111 puerto B
```

Al final de estas dos instrucciones el Puerto B tendría "uno" en todos sus bits y si fueran salidas pues encenderían todos los pines de este puerto.

Otro ejemplo:

```
PuertoA     equ     d'5'              ; declaramos el puerto A
PuertoB     equ     d'6'              ; declaramos el puerto B
Contador    equ     d'13'             ; declaramos contador
            clrf    PuertoB     ; pone a 0 el puerto B
            movlw   b'10101010' ; carga el acumulador con el patrón b'10101010'
            movwf   PuertoB     ; mueve el contenido del patrón al puertoB
            movlw   b'00000001' ; carga el acumulador con el patrón b'00000001'
            movwf   Contador    ; mueve el acumulador patrón a Contador
```

Al final de este programa tendríamos en PuertoB b'10101010' y en el Contador b'00000001'

Un programa de una instrucción:

```
        ; Este es un programa de una instrucción
        ; ------------------------------------------------
PuertoA     equ    d'5'
PuertoB     equ    d'6'
            Org    d'0'              ;define el origen
            movlw  b'00001111'       ;carga acumulador con b'00001111
            End                      ;fin del programa
```

Como ya se ha comentado anteriormente, el punto y coma";" se utiliza para hacer comentarios a las instrucciones que vamos editando. El compilador no las traduce, simplemente no la toma en cuenta. El punto y coma le indica al compilador que lo que sigue en ese renglón no debe ser considerado.

El ";" es muy útil para hacer un encabezado o usándolo después de una instrucción para hacer un comentario sobre las misma. Un programa bien comentado será más fácil de entenderse.

Las declaraciones **org** define la dirección de la memoria donde iniciamos a colocar las siguientes instrucciones:

```
        org         d'0'          ;define el origen
```

Quiere decir que las instrucciones que siguen serán grabadas en la memoria a partir de la dirección d'0'.

Aunque parece lógico que comencemos a escribir el programa a partir de la dirección d'0' para la programación de los PIC se ha de indicar en detalle todas las instrucciones.

End se usa para terminar el programa y **org** para iniciar el programa.

Un programa para encender algunas salidas:

```
; Este es un programa para encender algunas salidas
;-------------------------------------------------------------------------
        PuertoA         equ  d'5'
        PuertoB         equ  d'6'
        Org             d'0'            ; inicializa el programa
Inicio
        Movlw           b'00001111'
        Tris            PuertoB
PrendeSalidas
        movlw           b'11111110'   ; carga acumulador con b'11111111'
        movwf           PuertoB
        goto            Inicio
        End                           ; fin del programa
```

En este ejemplo, las palabras: Inicio y PrendeSalidas son **Etiquetas**, las usamos para mantener el programa bien documentado y como referencia para otras instrucciones que se coloquen más adelante. Las Etiquetas son como nombres del renglón. Siempre se escriben a partir de la columna 1.

En el renglón que sigue a la etiqueta Inicio, se instruye al PIC cuales pines del Puerto B deseamos como Entradas y cuales como Salidas.

Esto se logra mediante dos instrucciones:

```
        movlw   b'11111111'

        Tris    PuertoB
```

Tris PuertoB quiere decir: "define las entradas / salidas del PuertoB según el patrón del acumulador".

Un "0" asigna Salida, Un "1" asigna Entrada.

Entonces:

RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
0	0	0	0	0	0	0	1

RB7, RB6, RB5, RB4, RB3, RB2, RB1se definen como Salidas.

RB0 se definen como Entradas.

Después de la etiquete PrendeSalidas encontramos:

```
movlw   b'11111110'    ;carga acumulador con b'11111110'

movwf   PuertoB        ; mueve el valor del acumulador en el puertoB
```

Esto hace encender RB7, RB6, RB5, RB4, RB3, RB2, RB1, pero no RB0

Antes de la instrucción **End** encontramos la instrucción

```
goto   Inicio
```

La instrucción **goto** hace que el micro regrese a la etiqueta Inicio y repita las instrucciones en un bucle infinito.

- Operaciones orientadas a literales y de control

Estas 13 instrucciones nos permiten realizar determinadas acciones de control en la programación, como llamar a subrutina, saltar a una dirección, retornar a una subrutina, etc. Ver la siguiente tabla:

OPERACIONES ORIENTADAS A LITERALES Y DE CONTROL			
Nemotécnicos		**Operación**	**Estados afectados**
ADDLW	k	Sumar literal k a W	C,DC,Z
ANDLW	k	AND entre k y W	Z
CALL	k	Llamar subrutina	
CLRWDT		Limpiar WDT	-TO,-TD
GOTO	K	Salta a dirección k	
IORLW	K	OR entre k y W	Z
MOVLW	K	Cargar W con literal k	
RETFIE		Retornar de interrupción	
RETLW	K	Retornar y cargar W con k	
RETURN		Retornar de subrutina	
SLEEP		Ir al modo de bajo consumo	-TO,-TD
SUBLW	K	Restarle k a W	C,DC,Z
XORLW	K	OR exclusiva entre k y W	Z

- ## Configuración y manejo de salidas

Ya tenemos una noción básica de lo que es programar en lenguaje ensamblador, ahora iremos aprendiendo poco a poco más instrucciones para manejar al PIC 16F84A.

Empezaremos poniendo un encabezado a modo de comentario que nos dé información del programa que escribiremos, esto no solo le da orden a nuestro trabajo sino también nos sirve de soporte para cualquier consulta posterior.

```
prog 1: Bloc de notas
Archivo  Edición  Formato  Ver  Ayuda
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Autor: Sherlin E. Hernández C.
;
; Fecha: 22/Marzo/2014
;
; Título: Configuración de las salidas del PIC 16F84A
;
; Descripción: Este código tiene como finalidad configurar todas las líneas
;              del PIC 16F84A como salidas y mostrar un dato en el puerto B.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Información preliminar del inicio del código fuente

Tengamos en cuenta que, en cada renglón, todo lo que va después de un punto y coma será ignorado por el ensamblador, esto lo aprovecharemos para hacer todos los comentarios que sean necesarios para tener un código fuente ordenado y comprensible.

Ahora escribiremos la biblioteca, también conocida como librería:

```
;-----------------------------------------------------------------------
; BIBLIOTECA
;-----------------------------------------------------------------------

        LIST     P=PIC16F84A      ; PIC a usar
        #INCLUDE <P16F84A.INC>    ; Lista de etiquetas de Microchip
```

Indicándole al programa el microcontrolador a usar

Luego viene la palabra de configuración de los **FUSES**, indicando que se usará el oscilador tipo **XT** y se dejará habilitado el retardo de encendido **PWRTE** y deshabilitados **WDT** y **CP**:

```
;-----------------------------------------------------------------------
; CONFIGURACIÓN DE FUSIBLES
;-----------------------------------------------------------------------

        __CONFIG _XT_OSC & _WDT_OFF & _PWRTE_ON & _CP_OFF
```

Palabra de configuración

Ahora viene el mapa de memoria, donde asignaremos nombres a los registros que vamos a usar (algunos prefieren llamar a este proceso declaración de variables):

```
;-------------------------------------------------------------------------
; MAPA DE MEMORIA
;-------------------------------------------------------------------------

STATUS  EQU     0X03    ; Registro de estado
PORTA   EQU     0X05    ; Registro de manejo del puerto A
PORTB   EQU     0X06    ; Registro de manejo del puerto B
```

Mapa de memoria

En seguida viene el inicio del programa y la configuración de los puertos, en este caso todas las líneas serán configuradas como salidas:

```
;-------------------------------------------------------------------------
; CONFIGURACIÓN DE INICIO
;-------------------------------------------------------------------------

        ORG     0X00            ; Punto de inicio del programa,
        BSF     STATUS,5        ; Nos dirigimos hacia el banco 1 de la memoria.
        MOVLW   b'00000'        ; Cargo 5 dígitos binarios en cero,
        MOVWF   0X05            ; los paso a TRISA para configurar PORTA como salida.
        MOVLW   b'00000000'     ; Cargo 8 dígitos binarios en cero.
        MOVWF   0X06            ; los paso a TRISB para configurar PORTB como salida.
        BCF     STATUS,5        ; Regresamos al banco 0 de la memoria.
```

Inicio y configuración de los puertos

Ahora sí viene lo interesante que es la elaboración del programa propiamente dicho, por ahora solo haremos que las líneas del puerto B se pongan en ‘1’ y en ‘0’ intercaladamente de la siguiente manera:

```
;-------------------------------------------------------------------------
; CUERPO DEL PROGRAMA
;-------------------------------------------------------------------------

        MOVLW   b'10101010'     ; Cargo en W el valor que voy a mostrar
        MOVWF   PORTB           ; y lo pongo en el puerto B
```

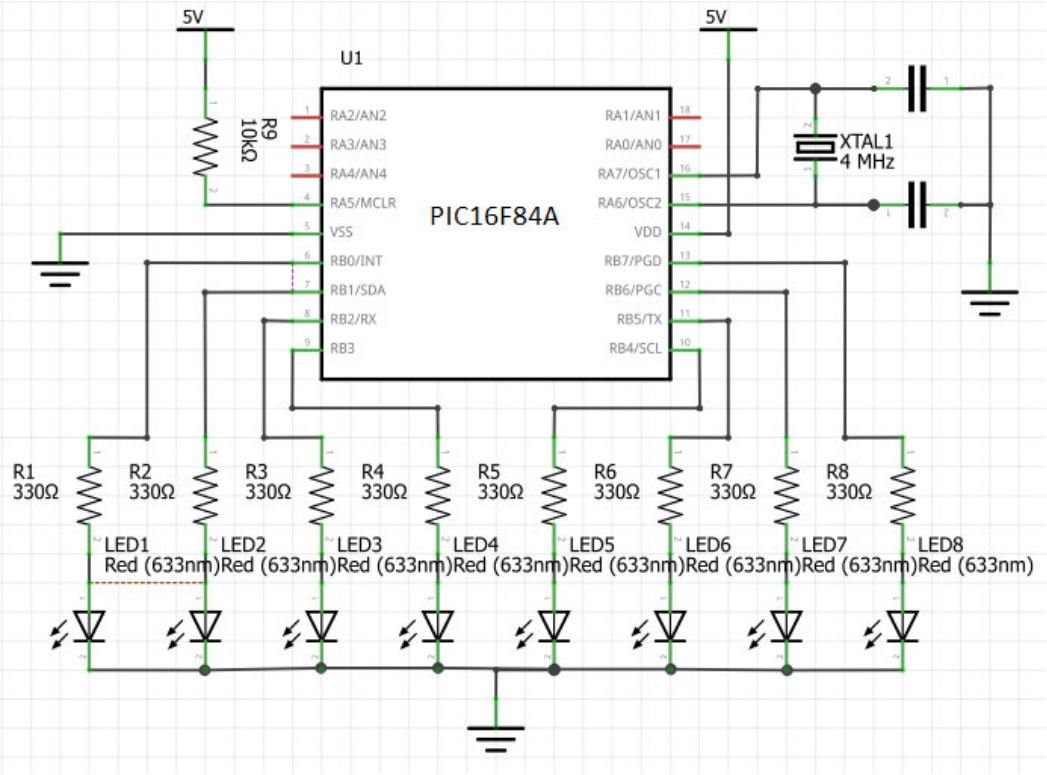
Transfiriendo un dato al puerto B

Con esto ya tenemos las salidas del puerto B intercaladas entre ‘ceros’ y ‘unos’. Ahora indicaremos que ha finalizado el programa:

```
;-------------------------------------------------------------------------
; FINALIZACIÓN
;-------------------------------------------------------------------------

        END
```

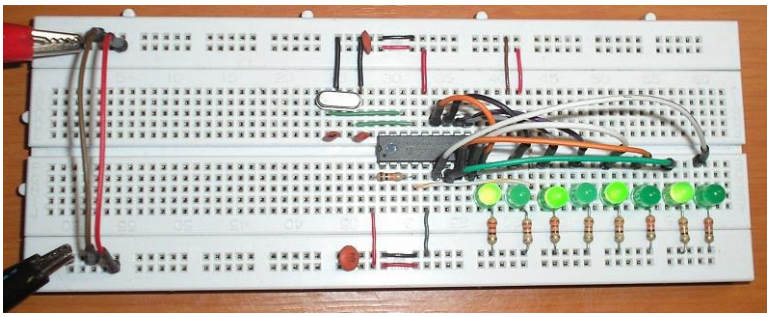
Finalización del código en ensamblador

Cuando el programa ya esté elaborado se guarda con formato .asm y se compila en .hex y solo queda comprobar su funcionamiento. El esquemático del circuito que se debe usar se muestra en la siguiente figura:



Esquemático del circuito con PIC16F84A

Cuando pongamos a funcionar el microcontrolador deberíamos ver algo como esto:



Montaje en protoboard del circuito a probar

Donde vemos claramente que el programa realizó su función: donde pusimos ‘1’ se encendió el LED y donde pusimos ‘0’ se quedó apagado. Nótese la correspondencia de bits entre el valor puesto en PORTB (b’10101010′) y el valor visto en los LED’s: RB7 se controla con el bit más a la izquierda, mientras que RB0 se controla con el bit más a la derecha.

- Rutinas de retardo

Bien, ya pudimos sacar un dato por el puerto B del PIC 16F84A, ahora aprenderemos nuevas instrucciones que nos permitan «jugar» con las salidas del microcontrolador.

```
        GOTO    etiqueta
```

La instrucción **GOTO** es básicamente lo que hace es ir a la línea de código donde se encuentre la etiqueta especificada, algunos programas lo usa para realizar un bucle infinito. Para empezar veamos el siguiente ejemplo de programación utilizando la instrucción **GOTO** para ir al inicio del programa y repitiendo la secuencia indefinidamente.

```
inicio  MOVLW   b'11111111'   ; W se carga con «unos»
        MOVWF   PORTB         ;para transferirlos al puerto B.
        MOVLW   b'00000000'   ; W se carga con «ceros»
        MOVWF   PORTB         ; para transferirlos al puerto B.
        GOTO    inicio        ; Va a inicio para repetir la secuencia
```

Este código simplemente enciende los LED's y luego los apaga, la instrucción GOTO hace que esto se repita una y otra vez creando un bucle infinito. Si implementamos el código no seremos capaces de ver lo que ocurre, solo veremos los diodos LED encendidos todo el tiempo y quizás con la mitad del brillo normal; no es que el programa no realice lo esperado sino que las instrucciones se realizan tan rápido que nuestra vista no podrá percibir las secuencias.

Recordando el concepto de ciclo de máquina tenemos que para un cristal de cuarzo de 4MHz cada instrucción básica demora en ejecutarse «1 microsegundo», lo cual es un tiempo demasiado corto en el que prácticamente no nos damos cuenta de nada.

La solución a este problema es aumentar el tiempo que hay desde el encendido de los LED's hasta el apagado y viceversa con el fin de que nuestra vista pueda apreciar la *secuencia*, para lograr esto habrá que poner al PIC a «quemar» tiempo, este proceso recibe el nombre de **rutina de retardo**.

○ Cómo hacer una rutina de retardo

Antes de generar una rutina de retardo es conveniente aprender una instrucción muy importante:

```
CALL        etiqueta
```

Esta instrucción sirve para hacer llamadas a distintas líneas de código. Es muy parecida a la instrucción **GOTO**, la diferencia es que **CALL** es una instrucción obligada a regresar al sitio desde donde se hizo la llamada, esta propiedad la podemos usar para realizar funciones y de ese modo podemos separar el código de un programa en diferentes partes cada una realizando una función específica. Por lo tanto, usaremos **CALL** para llamar a una función a la que se le asignó la etiqueta «**retardo**», con lo cual el código que llevamos quedaría de la siguiente manera:

```
inicio    MOVLW   b'11111111'    ; W se carga con «unos»
          MOVWF   PORTB          ; para transferirlos al puerto B.
          CALL    retardo        ; Llama a un retardo de tiempo
          MOVLW   b'00000000'    ; W se carga con «ceros»
          MOVWF   PORTB          ; para transferirlos al puerto B.
          CALL    retardo        ; Llama a un retardo de tiempo
          GOTO    inicio         ; Va a inicio para repetir la secuencia
```

Ya se ha llamado al retardo, ahora habrá que crearlo. Para generar retardos lo que se hace es ordenarle al PIC que haga algunas instrucciones adicionales con el objetivo de ir consumiendo tiempo. La instrucción más usada para este fin es:

```
DECFSZ      registro,1                                          .
```

La función de esta instrucción es restarle una unidad al registro especificado y si el resultado da cero se saltará la siguiente línea del código, el '1' que va después de la coma es simplemente un indicador que le dice al programa que el resultado de la resta lo guarde en el mismo registro (si hubiera un '0' el resultado se guardaría únicamente en el registro **W**). Para usar esta instrucción se hace necesario usar un registro de propósito general, el PIC 16F84A dispone de 68 ubicados desde la dirección **0Ch** (0X0C) hasta la **4Fh** (0X4F). A continuación se le dará un valor inicial a un registro que se ha llamado «reg» y luego se le aplicará la función **DECFSZ**:

```
        MOVLW    0X03      ; W se carga con el número 3
        MOVWF    reg       ; pone el 3 en el registro
        DECFSZ   reg,1     ; Le resta una unidad al registro y el resultado es 3-1=2
```

No «quemó» mucho tiempo, por eso vamos a agregar un GOTO para crear un bucle, en este caso finito ya que se requiere que en cierto momento el programa se salga del bucle:

```
        MOVLW    0X03      ; W se carga con el número 3
        MOVWF    reg       ;  pone el 3 en el registro
etiqueta DECFSZ  reg,1     ; Le resta una unidad al registro

        GOTO     etiqueta  ; Regresa y vuelve a restar
```

Lo que hace este código es asignarle el número 3 al registro y luego lo decremento, como el resultado da 2 la función GOTO envía de nuevo al programa a decremento al registro y ahora el resultado da 1; se realiza de nuevo la función GOTO, pero cuando se decrementa, el valor del registro pasa de 1 a 0 y esta vez ya no se realizará la instrucción GOTO, con lo cual, el programa se sale del bucle que habíamos creado. Básicamente, esta es una manera de gastar más tiempo realizando instrucciones, ahora le aumentaremos un poco su complejidad para obtener rutinas de retardo con mayor duración de tiempo.

- Rutinas de retardo anidadas

Al añadir una rutina de retardo dentro de otra conseguimos aumentar significativamente el tiempo gastado en instrucciones. A continuación se presenta el código para tres rutinas de retardo anidadas, para el cual usaremos tres registros que he convenido llamar reg1, reg2 y reg3:

```
retardo     MOVLW    0X20      ; W se carga con el número 20h (Comienza la llamada)

            MOVWF    reg3      ; y se pasa a reg3
externo     MOVLW    0X30      ; W se carga con el número 30h
            MOVWF    reg2      ; y se pasa a reg2
mitad       MOVLW    0X50      ; W se carga con el número 50h
            MOVWF    reg1      ; y se pasa a reg1
interno     DECFSZ   reg1,1    ; Le resta una unidad a reg1
```

```
GOTO    interno     ; sigue decrementando hasta que reg1 llegue a 0
DECFSZ  reg2,1      ; Le resta una unidad a reg2 cuando reg1 llegue a 0
GOTO    mitad       ; vuelve a cargar reg1 y se repite la rutina interna
DECFSZ  reg3,1      ; Le resto una unidad a reg3 cuando reg2 llegue a 0
GOTO    externo     ; vuelve a cargar reg2 y reg1, repite la rutina en la mitad

RETURN              ; Termina la llamada y regresa
```

El código se ejecuta cuando la función CALL dirija al programa hacia la etiqueta «retardo», estando allí lo primero que se hace es cargar los tres registros, una vez hecho esto comenzará a decrementarse el registro ubicado en el bucle interno (reg1), cuando reg1 llegue a '0' se decrementa reg2 y se repite el bucle interno. Cuando reg2 llegue a '0' se repetirá el bucle de la mitad y cuando reg3 llegue a '0' se finaliza el bucle externo. Notemos que entre más adentro esté un bucle más veces se repetirá. La instrucción RETURN se encarga de finalizar la llamada, con lo cual el programa regresa hacia la línea siguiente desde donde fue hecha dicha llamada.

- ### Calculando el tiempo del retardo

Ya habiendo entendido el código lo más importante ahora es diseñarlo para que el retardo dure el tiempo que queramos implementar. Lo que tenemos que hacer es sumar la cantidad de ciclos de máquina que gasta nuestra rutina de retardo y multiplicar el resultado por el tiempo de duración de un ciclo de máquina, que para un cristal de 4MHz es de 1 microsegundo. A continuación se muestran los ciclos de máquina de las instrucciones más utilizadas cuando se generan retardos:

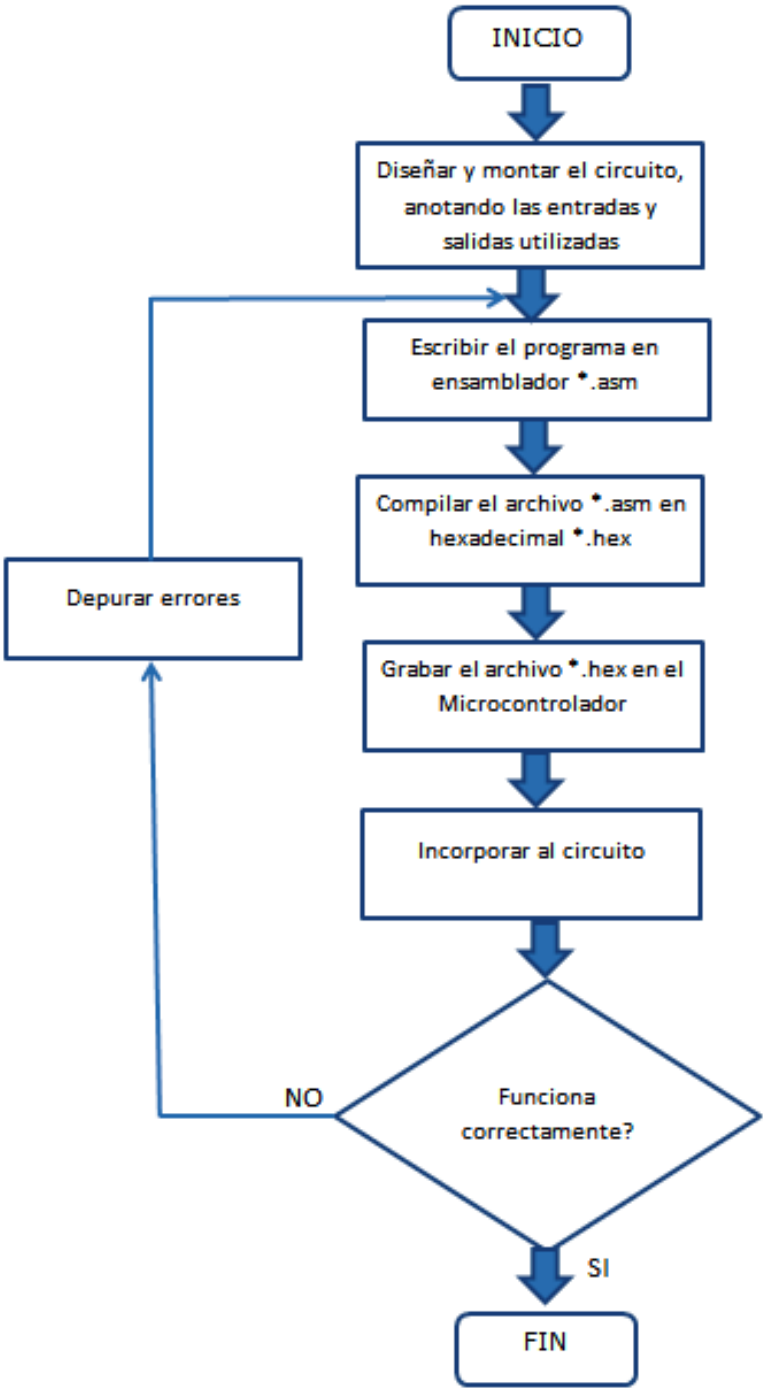
CALL ————-> 2 ciclos de máquina
MOVLW ——> 1 ciclo de máquina
MOVWF ——> 1 ciclo de máquina
DECFSZ ——> 1 ciclo de máquina (si la operación da '0' se gastan 2 ciclos de máquina)
GOTO ————> 2 ciclos de máquina
RETURN ——> 2 ciclos de máquina
NOP ————> 1 ciclo de máquina

Acabamos de incluir una nueva instrucción en nuestro repertorio: **NOP**. Esta instrucción significa no hacer nada («quemar tiempo»), es muy usada para ajustar los tiempos de las rutinas de retardo.

6. DISEÑO DE PROYECTOS

Los microcontroladores PIC pueden ayudarnos a realizar soluciones sencillas, rápidas y baratas. Partiremos de un proyecto sencillo teniendo en cuenta todas las variables que afectan al sistema, desarrollaremos la idea y la implementaremos con las herramientas adecuadas.

Diagrama de flujo del desarrollo de proyectos con microcontrolador PIC.

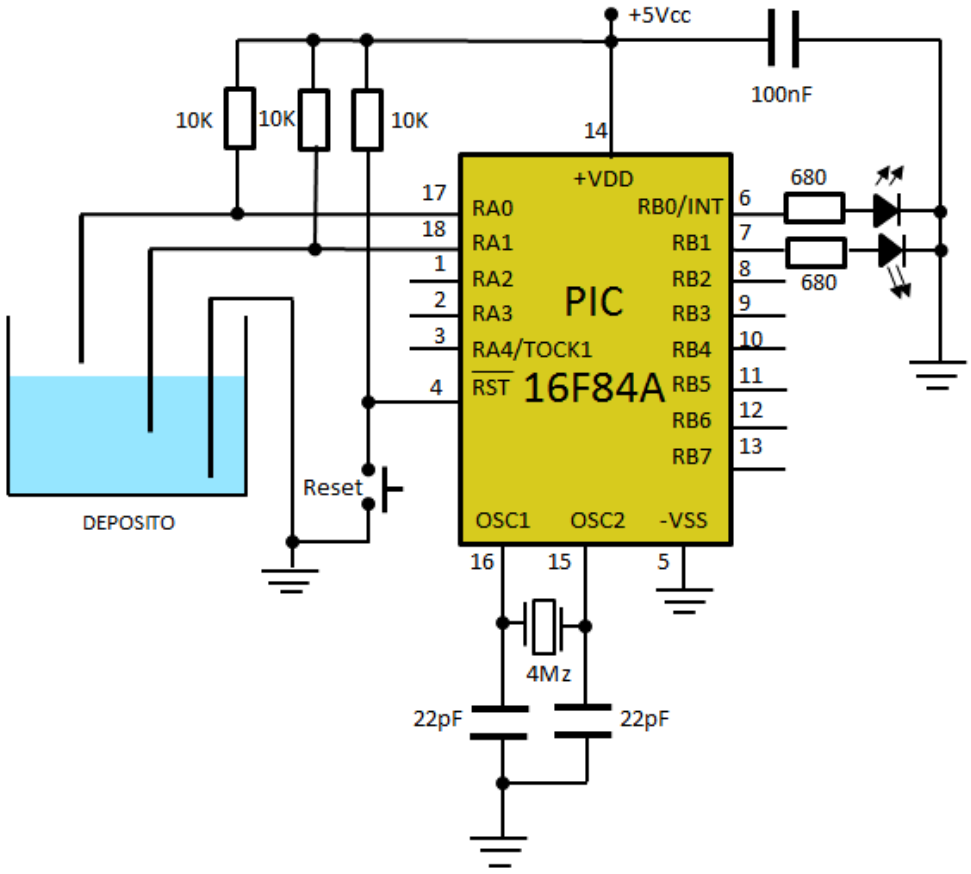


El desarrollo para la programación de un de **PIC** se realiza mediante cinco pasos fundamentales:

1. Diseñar y montar el circuito electrónico utilizando el PIC 16F84A con los puertos de entradas y salidas necesarios y tomando nota de los nombres de los pines utilizados, por ejemplo, RB3 de entrada, RB7 de salida, etc. Es importante tener en cuenta la forma en la que se deben conectar los pines del microcontrolador aun cuando estos no se usen y antes de su programación.
2. Escribimos el programa ensamblador en un editor de texto y lo salvamos con la extensión *.**asm**.
3. Lo compilamos pasando del programa fuente, ensamblador a hexadecimal, que es el programa de código máquina que entiende el microcontrolador PIC.
4. Tras finalizar el paso anterior y utilizando una tarjeta electrónica para programar el PIC, como la K-150, se transfiere el programa compilado en hexadecimal del PC al microcontrolador, ver el capítulo 7 “Programador PIC ICSP K-150”.
5. Una vez finalizado la grabación del PIC, lo extraemos del zócalo ZIF del programador PIC y lo insertamos en el circuito electrónico que habíamos diseñado y montado en el paso 1. Comprobamos que el circuito funciona como habíamos deseado, de lo contrario tendremos que modificar el programa y volver al paso 2.

- ## Diseño del circuito y montaje

En el proyecto diseñamos la solución sobre un circuito electrónico utilizando nuestro microcontrolador PIC 16F84A y seleccionando los puertos de entradas y salidas que se van a utilizar. En este caso se va a diseñar y montar un sencillo medidor de nivel de agua para el control de un depósito. Según el siguiente esquema se ha utilizado dos puertos el puerto A y puerto B. En el puerto A se han utilizado los pines de entrada, RA0 y RA1, cada una de estas entradas detecta un nivel de líquido y se utilizarán dos pines de salida del puerto B, RB0 y RB1, éstos nos señalizan mediante un led rojo la falta de líquido.

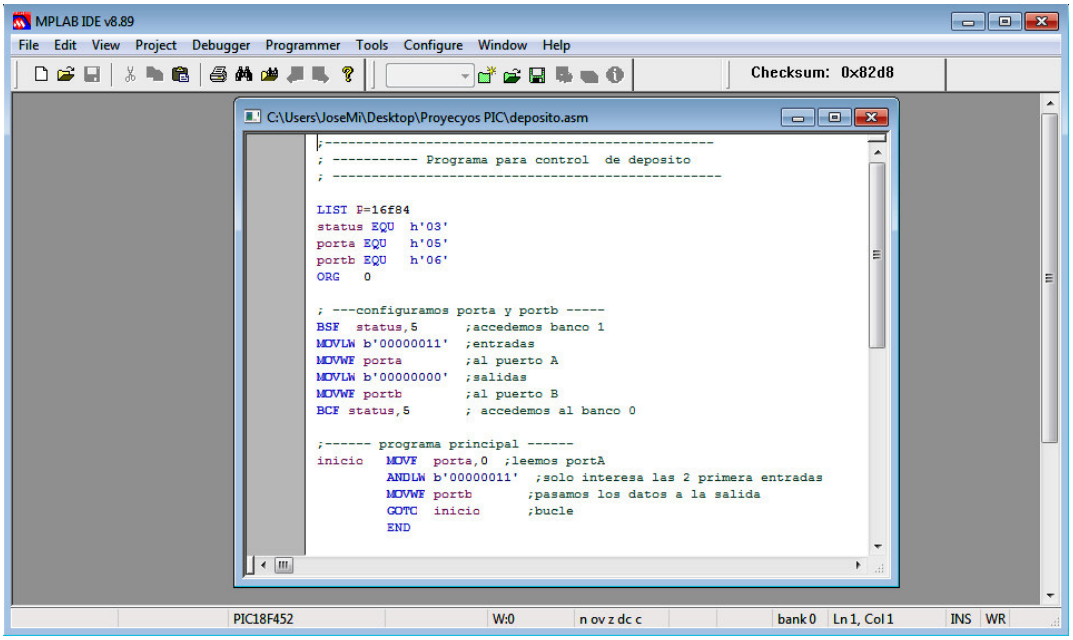


Esquema eléctrico del medidor de nivel de líquido.

En **RA0** y **RA1** tenemos 2 sondas de detección de nivel de agua y en **RB0** y **RB1** tenemos las salidas mediante la señalización de un diodo led. Cuando la sonda no detecta agua se enciende el diodo led.

- Editando el programa en ensamblador

Entrando en el corazón de la creación de proyectos con microcontroladores. Lo primero que necesitaremos es crear una carpeta de proyectos en nuestro ordenador y seguidamente un software para escribir los programas, en este caso basta con un editor de textos básico como el famoso bloc de notas de Windows, o también mediante la aplicación **MPLAB** de **Microchip** que se puede editar y compilar.



Escritura de la programación en ensamblador mediante la aplicación MPLAB

Para escribir un programa en lenguaje ensamblador, en cada renglón se deben diferenciar **cuatro columnas separadas** poniendo un espacio en blanco entre ellas, en la primera columna se escriben las **etiquetas** que asignamos a determinada línea del código, la segunda columna es la **instrucción**, la tercera columna es el **registro**, bit o dato al que se le aplica la instrucción y en la cuarta los **comentarios**. La estructura de cada renglón escrito quedaría así:

```
Etiqueta    Instrucción    Registro/Bit/Dato    ; comentario
```

No siempre se ponen etiquetas a cada renglón, en ese caso habrá que dejar la columna vacía pero se debe respetar el espacio en blanco antes de escribir la instrucción así:

```
            Instrucción    Registro/Bit/Dato    ; comentario
```

Cuando escribimos un programa es muy importante agregar comentarios que nos guíen cada vez que queremos estudiar o modificar el código, en ellos también podemos agregar información adicional, por ejemplo, la conexión de los pines del microcontrolador. Los comentarios se escriben precedidos de un punto y coma (;) así:

```
Etiqueta    Instrucción    Registro/Bit/Dato    ; comentario
```

Nota: A lo largo de un programa en ensamblador se hace uso frecuente del registro **W**, el cual se encarga de almacenar momentáneamente los datos que van a procesarse.

La estructura de un programa en ensamblador es la siguiente:

1. Biblioteca
2. Configuración de fusibles
3. Mapa de memoria
4. Configuración de inicio
5. Cuerpo del programa
6. Fin

1. Biblioteca: Esta es la parte del código donde se indica el microcontrolador que se va a programar, para el PIC 16F84A se escribe lo siguiente:

```
LIST P=PIC16F84A
#INCLUDE <P16F84A.INC>
```

2. Configuración de fusibles: Indica qué fusibles se usarán, por ahora vamos a usar la siguiente palabra de configuración:

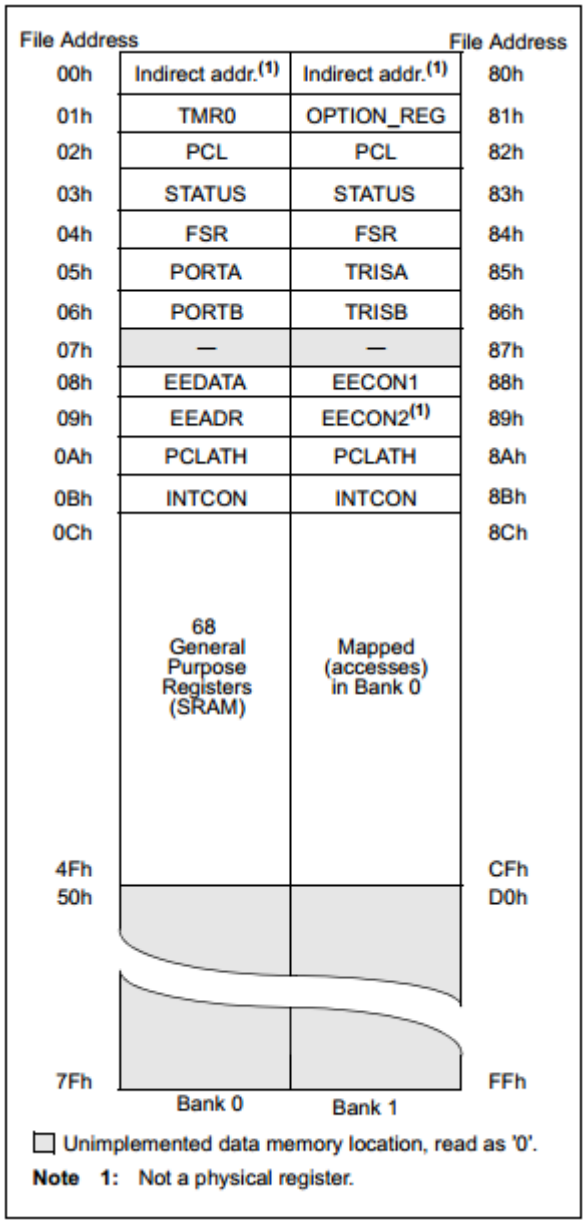
```
__CONFIG    _XT_OSC & _WDT_OFF & _PWRTE_ON & _CP_OFF
```

La cual indica que se usará un oscilador XT (4MHz) y se activará el temporizador de encendido; no se usará perro guardián ni protección de código.

3. Mapa de memoria: Recordemos que producto de su arquitectura Harvard, el microcontrolador posee dos memorias (de datos y de programa). El mapa de memoria es la parte del código que le asigna nombres a los registros de la memoria de datos que serán usados, no es obligatorio hacer esto pero sí es aconsejable pues facilita la elaboración de programas. Para asignar un nombre a un registro se debe conocer su dirección, por el momento nos interesan las direcciones de los registros de configuración y manejo de los puertos del PIC (0X05 y 0X06); también nos interesa la dirección de un registro muy importante llamado STATUS (0X03):

```
PORTA   EQU   0X05; Registro de configuración y manejo del puerto A
PORTB   EQU   0X06; Registro de configuración y manejo del puerto B
STATUS  EQU   0X03; Registro de estado
```

La instrucción «EQU» sirve para asignarle un nombre al registro especificado, el cual tiene una dirección en formato hexadecimal. Para saber con qué registros cuenta la memoria de datos véase la siguiente figura.



Nótese que la memoria de datos del PIC 16F84A está dividida en dos partes llamadas banco 0 y banco 1. Para elegir el banco que se va a acceder, el registro STATUS tiene un bit encargado de esta función.

Ejemplo:

Si estoy en el banco 0 y modifico el registro de dirección 0X05, estaría modificando el registro PORTA; pero si estoy en el banco 1 y modifico también el registro de dirección 0X05 ahora estaría modificando el registro **TRISA**.

4. Configuración de inicio: En esta parte del código se indica el inicio del programa y se configuran las salidas y las entradas. El programa iniciará a partir de donde pongamos la siguiente línea:

```
ORG    0X00    ; Indica que aquí inicia el programa
```

Ya se puede configurar los puertos, si observamos la distribución de la memoria de datos veremos que los registros de configuración TRISA y TRISB se encuentran en el banco 1, por eso se hace necesario direccionarnos hacia ese banco usando el registro STATUS de la siguiente manera:

```
BSF    STATUS,5  ; Esta sentencia nos dirige hacia el banco 1 de la memoria
```

La función BSF significa poner un «1» en el bit indicado, en este caso, el bit 5 del registro STATUS. Cuando este bit, llamado RP0, se encuentra en «1» se activa el banco 1 y cuando está en «0» se activa el banco 0.

Estando en el banco 1 procedemos a acceder a los registros **TRISA** y **TRISB**; para configurar al puerto A como salida escribimos:

```
MOVLW   b'00000'  ; Con esto cargamos en W el número 0 en formato binario
MOVWF   0X05      ; Y con esto lo transferimos hacia el registro TRISA
```

Cuando ponemos TRISA en cero estamos configurando al puerto A como salida.

Para configurar el puerto B el procedimiento es similar:

```
MOVLW   b'00000000'  ; Con esto cargamos en W el número 0 en formato binario
MOVWF   0X06         ; Y con esto lo transferimos hacia el registro TRISB
```

Nótese que para configurar TRISA utilizamos en formato binario 5 ceros porque el puerto A tiene 5 líneas mientras que para configurar TRISB utilizamos 8 ceros porque el puerto B tiene 8 líneas.

Ya habiendo configurado los puertos deberíamos regresar al banco 0 que es desde donde se manejan, esto se hace con la siguiente instrucción:

```
BCF     STATUS,5  ; Esta sentencia nos dirige hacia el banco 0 de la memoria
```

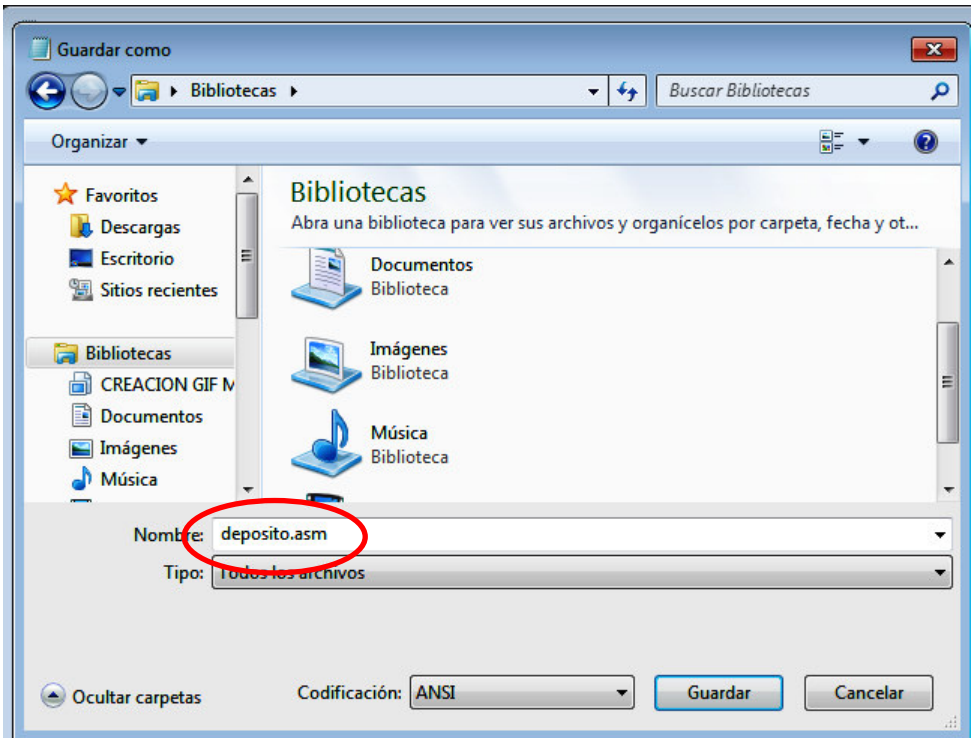
5. Cuerpo del programa: Aquí escribiremos las instrucciones que hemos diseñado para hacer funcionar al microcontrolador de la forma deseada. Con lo visto hasta ahora ya aprendimos dos instrucciones de manejo muy importantes: BSF (función para poner un «1» en determinado bit) y BCF (función para poner un «0» en determinado bit).

6. Fin: Para que el código en lenguaje ensamblador esté completo es necesario escribir una línea que indica la finalización de dicho código:

```
END      ; Con esta sencilla palabra terminamos el programa.
```

Para guardar el código debemos hacerlo en formato *.asm, en Windows lo hacemos de la siguiente manera: damos clic en Archivo>Guardar como y le damos un nombre al archivo seguido de la terminación «.asm», con esto, el texto queda automáticamente guardado en formato ensamblador con extensión *.asm.

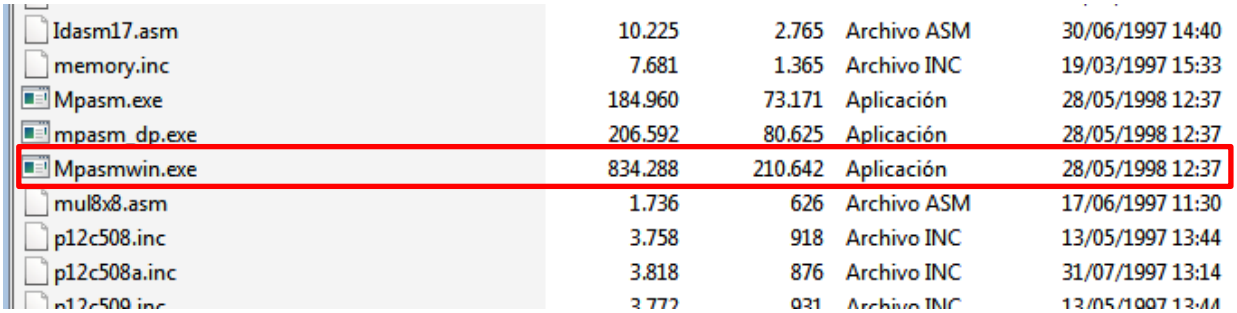
```
;-------------------------------------------------
; Programa para control  de depósito
; -------------------------------------------------

        LIST P=16f84
        status EQU  h'03'
        porta EQU   h'05'
        portb EQU   h'06'
        ORG   0

; ---configuramos porta y portb -----

        BSF             status,5            ;accedemos banco 1
        MOVLW           b'00000011'         ;entradas
        MOVWF           porta               ;al puerto A
        MOVLW           b'00000000'         ;salidas
        MOVWF           portb               ;al puerto B
        BCF             status,5            ; accedemos al banco 0

;------ programa principal ------

inicio  MOVF            porta,0             ;leemos portA
        ANDLW           b'00000011'         ;solo interesa las 2 primera entradas
        MOVWF           portb               ;pasamos los datos a la salida
        GOTO            inicio              ;bucle
        END                                 :fin del programa
```



Forma de guardar un código en lenguaje ensamblador

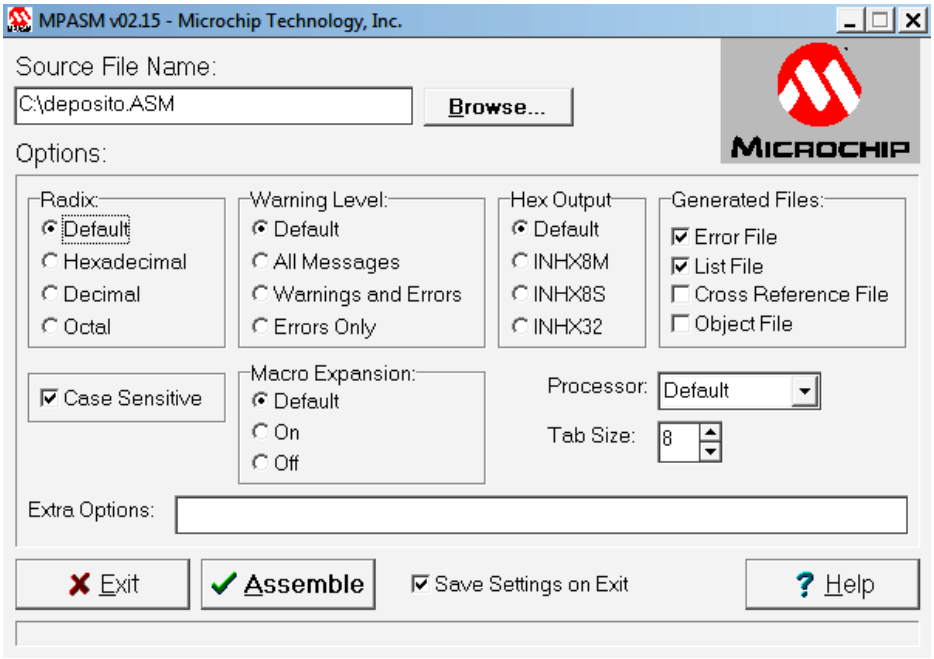
- Compilación en hexadecimal

Una vez que tenemos el archivo en ensamblador, "deposito.asm", tenemos que convertirlo a hexadecimal que es el código máquina que entiende el microcontrolador. Para ello descargamos de la web de **Microchip** el fichero comprimido **asm2150.zip** de la versión v02.15. Lo descomprimimos y ejecutamos el archivo "Mpasmwin.exe".

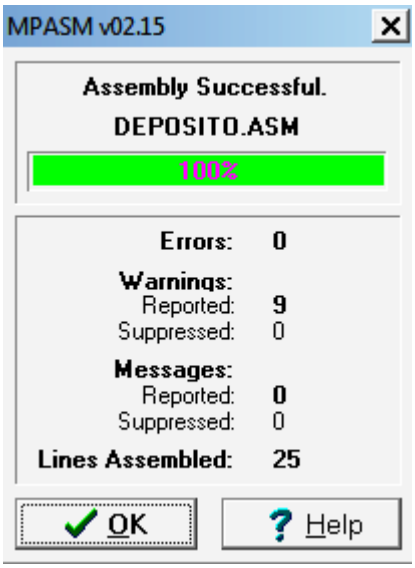


Ejecutamos el archivo Mpasmwin

Abrimos la aplicación para Windows **Mpasmwin.exe** y nos aparece la siguiente ventana:



Introducimos el nombre del fichero ensamblador “deposito.asm” y pulsamos el botón de **Assemble** que nos generará el fichero en hexadecimal **deposito.hex** en la carpeta de proyectos PIC, siempre que no existan errores de ensamblado, de lo contrario no lo crea. Si todo está correcto y sin errores, nos aparece la siguiente ventana:



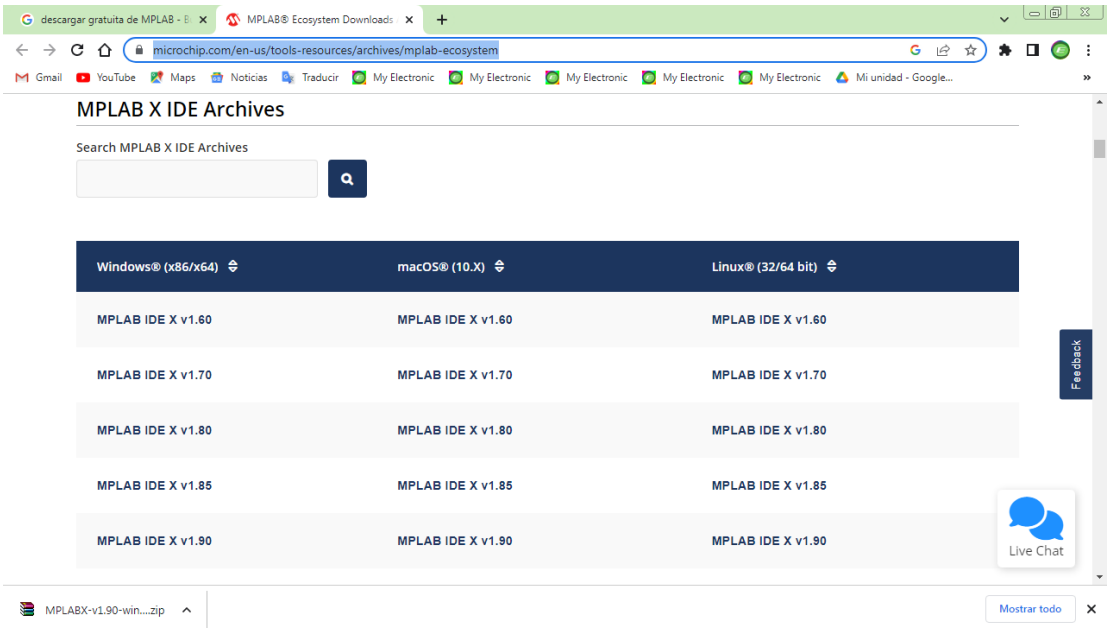
Con la aplicación **MPLAB IDE v8.89** se puede editar y compilar el programa, creando un directorio de proyecto.

MPLAB IDE (Integrated Development Environment) es un software profesional implementado por la empresa Microchip, compatible con XP, Vista y Windows 7 y con versiones recientes disponibles para MAC y Linux. MPLAB IDE es utilizado como un poderoso auxiliar para el desarrollo de sistemas basados en los microcontroladores PIC.

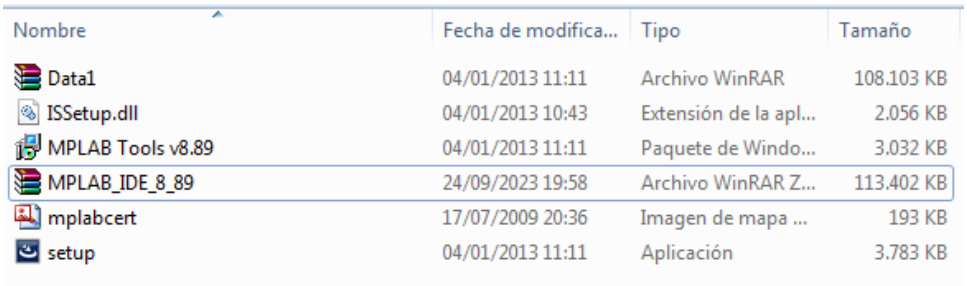
Su distribución es libre y gratuita y se puede realizar directamente la descarga del sitio de Microchip: https://www.microchip.com/

Para descargar la aplicación MPLAB de Microchip accedemos a la siguiente dirección: https://www.microchip.com/en-us/tools-resources/archives/mplab-ecosystem

Nos aparece la siguiente ventana con los archivos de diferentes versiones y sistemas operativos. Seleccionamos la versión que nos interesa, por ejemplo, para Windows MPLAP IDE v8.89 y lo descargamos.

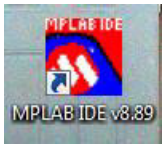


Nos descargará un archivo comprimido WinRAR.ZIP de unos 113MB.

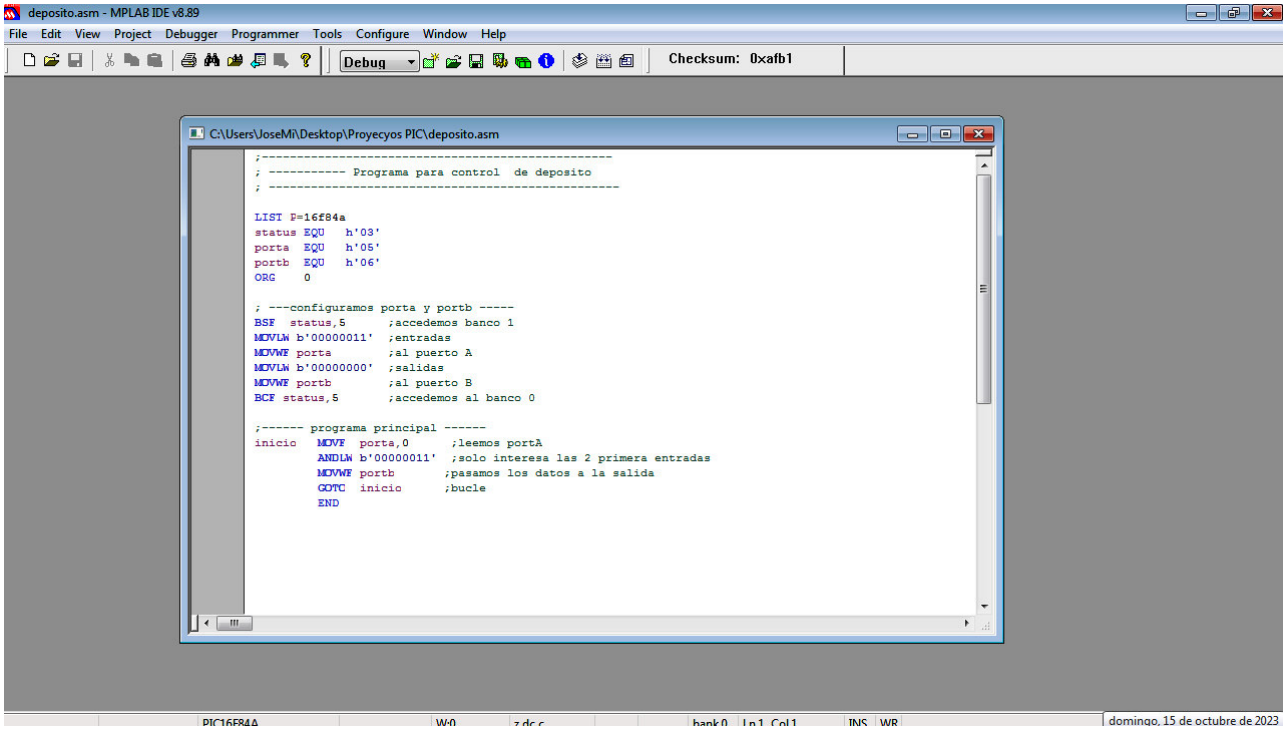


Descomprimimos el fichero MPLAB_IDE_8_89 y nos genera un directorio con el mismo nombre y con los archivos para su instalación.

Una vez instalada la aplicación nos crea un icono en el escritorio y pulsamos sobre el:

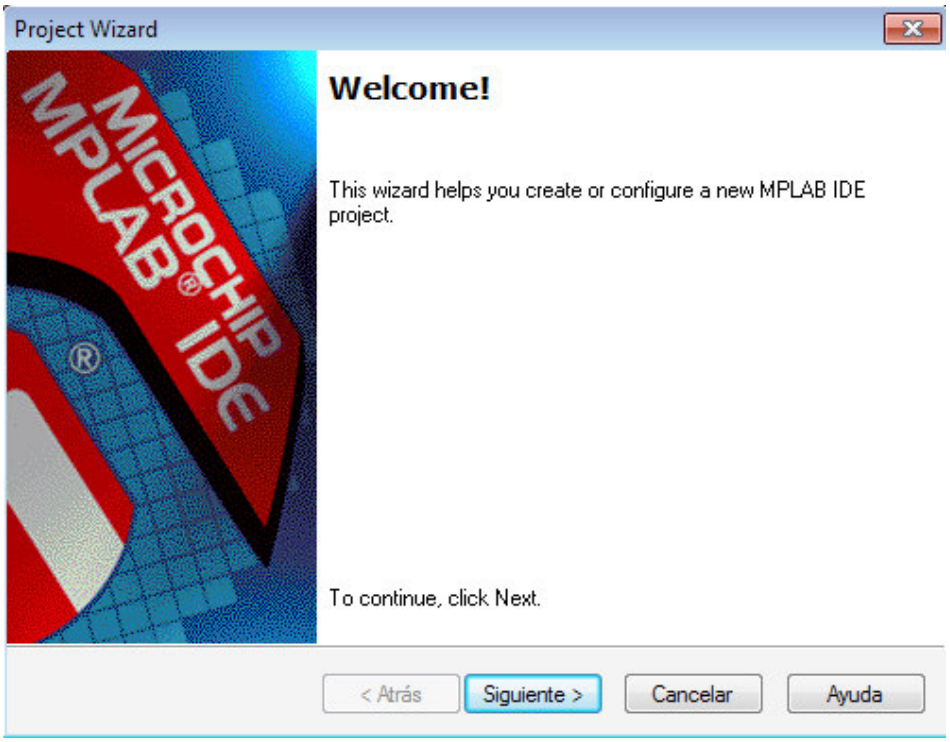


Para poder escribir nuestro programa en ensamblador accedemos a la pestaña “File” y seguidamente a “New” y a continuación ya podemos escribir las instrucciones del programa y lo grabamos en File/Save como DEPOSITO.ASM

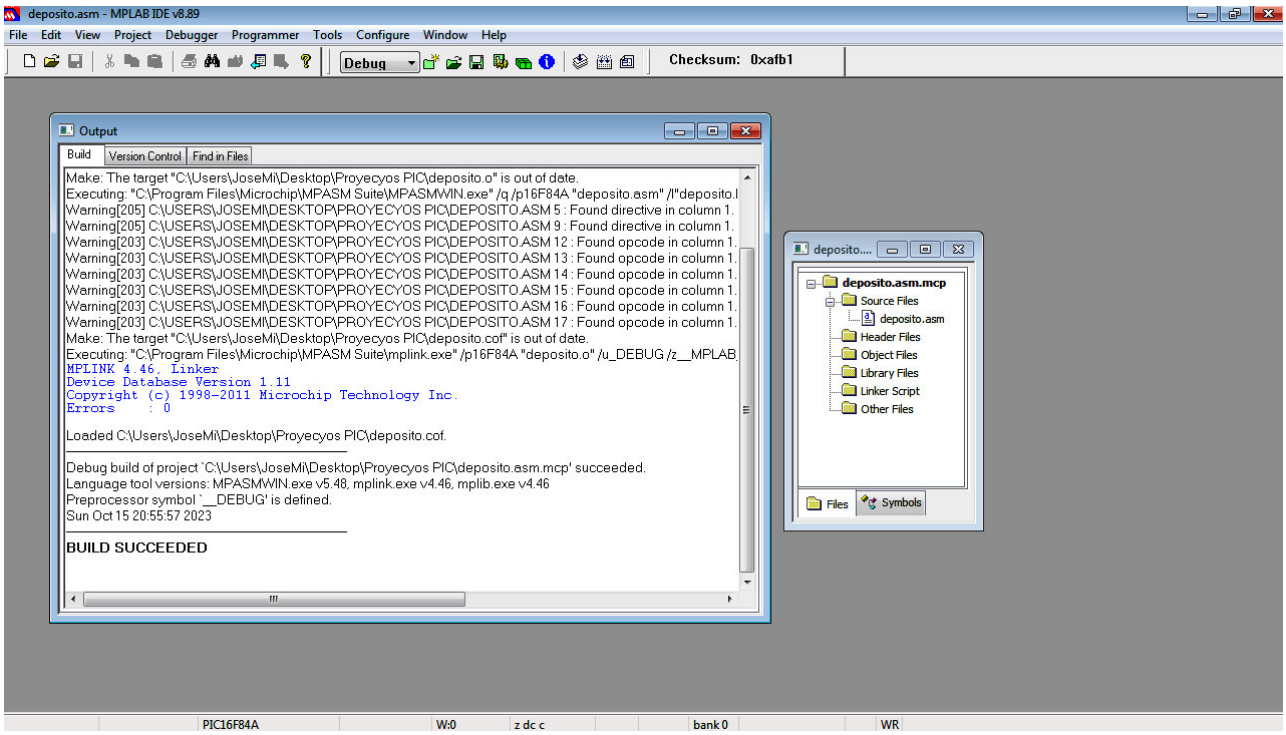


Detalle de edición en ensamblador mediante la aplicación MPLAB IDE v8.89

seguidamente nos vamos a crear proyecto en Project/Project Wizard…donde nos pedirá crear la carpeta de proyectos y los archivos *.asm



Al generar el proyecto Wizard crea el entorno de proyecto “deposito.asm.mcp” podemos compilarlo pulsando F10 Make o Control+F10 “Build All” en la pestaña de “Project” y aparece una ventana de la secuencia de operaciones de ensamblado donde nos muestra si existen errores y el número de fila del error, de lo contrario muestra al final **BUILD SUCCEEDED** “CONSTRUCCIÓN EXITOSA”. Ya podemos ver el archivo DEPOSITO.HEX en la carpeta de proyecto para grabarlo en nuestro microcontrolador.

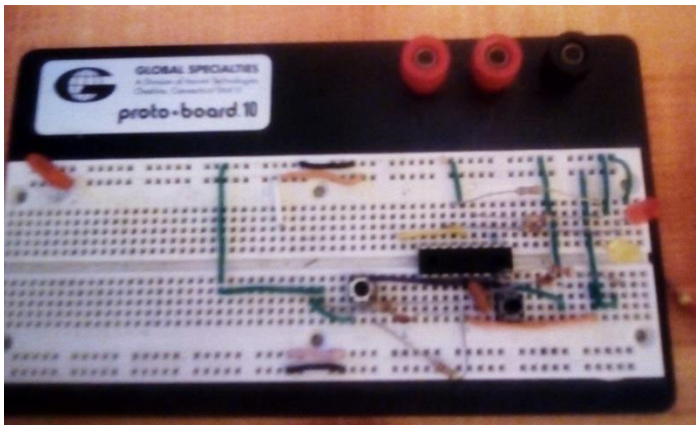


- Grabación del archivo hexadecimal en el microcontrolador PIC

Si hemos generado el archivo deposito.hex sin ningún problema pasaremos a insertar el microcontrolador PIC16F84A en el zócalo zif del programador y grabamos el programa en el microcontrolador.

- Comprobación de lo programado en el microcontrolador

Si todo ha salido sin haber errores pasaremos a quitar el microcontrolador PIC del zócalo del programador e insertarlo en el circuito de la proto-board.10, ya montado, y debemos comprobar que funciona correctamente, es decir, las entradas y salidas realizan correctamente lo deseado.



7. PROGRAMADOR PIC ICSP K-150

El **K-150** es un programador de microcontroladores PIC por puerto USB. Dispone de un zócalo ZIF de 40 pines para poder realizar la programación sin dañar los pines del chip. También permite la programación en circuito ICSP con un cable provisto que consigue que la aplicación final se pueda actualizar fácilmente sin retirar el dispositivo del producto final a través del ICSP.

Éste programador K-150 soporta los siguientes microcontroladores de las series: PIC10XX, PIC12CXX, PIC12FXX, PIC16CXX, PIC16FXX y PIC18FXX.

Estos tipos de microcontroladores PIC su memoria de programa es del tipo **Flash/EEPROM** y permite la ejecución de instrucciones de 200 nanosegundos lo que nos permitirá grabarlo hasta unas 10000 veces.

- Especificaciones del programador

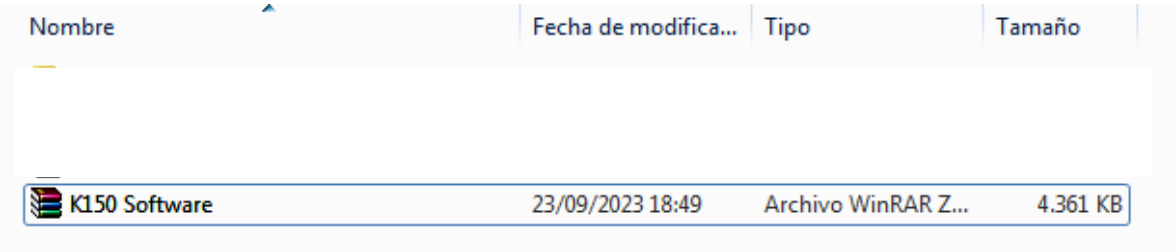
Soporta los PICs más populares de Microchip de 8 a 40 pines.
La alimentación de voltaje es mediante el puerto USB del ordenador.
Zócalo ZIF de 40 pines.
Cabezal ICSP.
Software de programación de fácil uso.

- **Descarga e Instalación del software y Drivers**

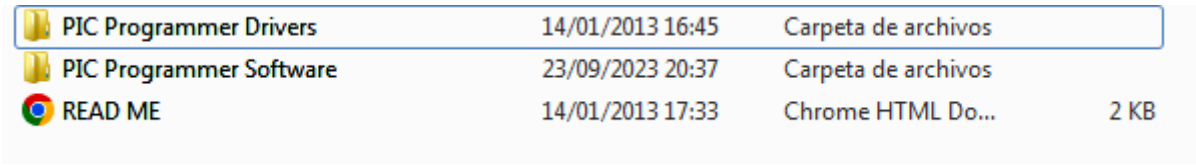
Lo primero que debemos hacer es descargar el software y drivers para el programador K-150 que se encuentran en el siguiente enlace:

SOFTWARE+DRIVERS en https://buyhere22.com/components/k150/

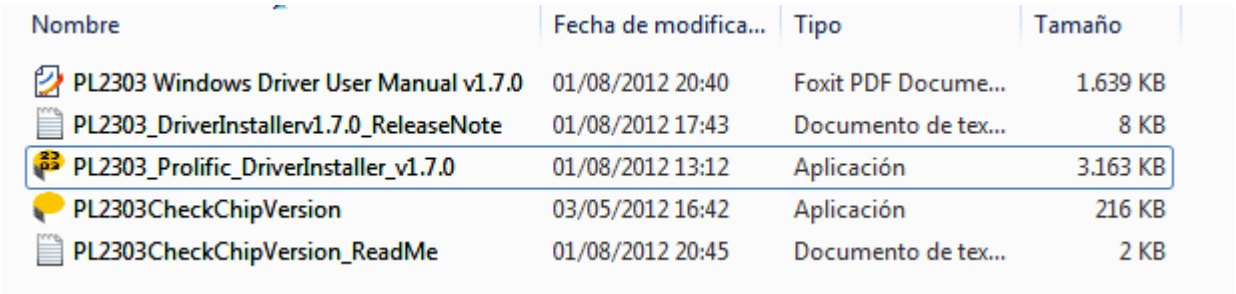
Una vez descargado, extraemos las dos carpetas contenidas en un archivo comprimido del tipo WinRAR ZIP con el nombre “K150 Software”



a continuación abrimos la carpeta llamada "**PIC Programmer Drivers**".



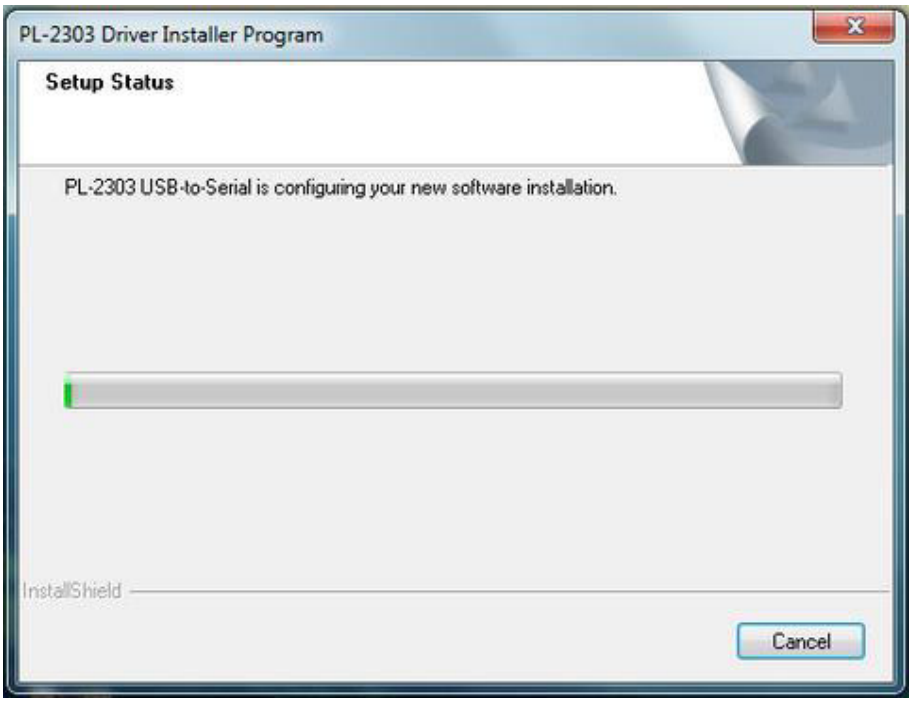
y ejecutamos la aplicación "**PL2303_Prolific_DriverInstaller_v1.7.0**".



A continuación nos desplegará la siguiente pantalla:

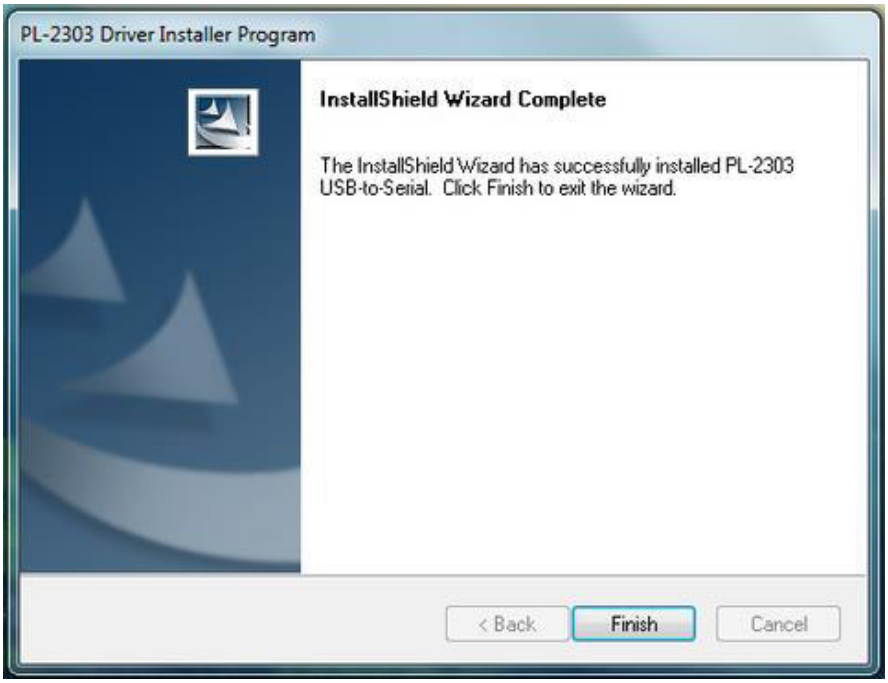
Instalación para PL-2303 USB-to-Serial

Seleccionaremos "**Next**" y se desplegará la siguiente pantalla en donde nos muestra el avance de la instalación:



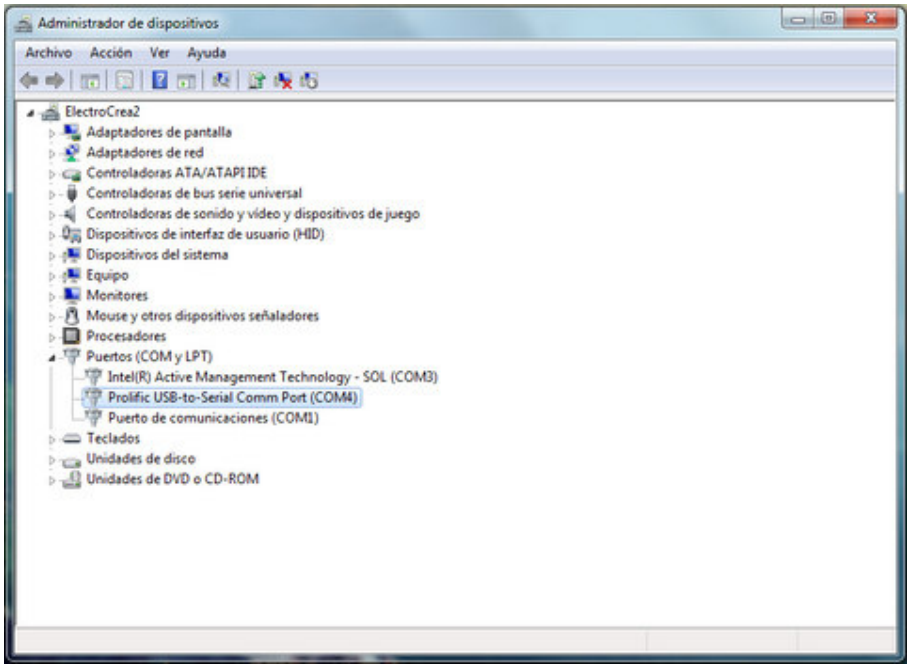
Instalando el programa del Driver en el ordenador

Una vez terminado el proceso de instalación, presionamos el botón "**Finish**":



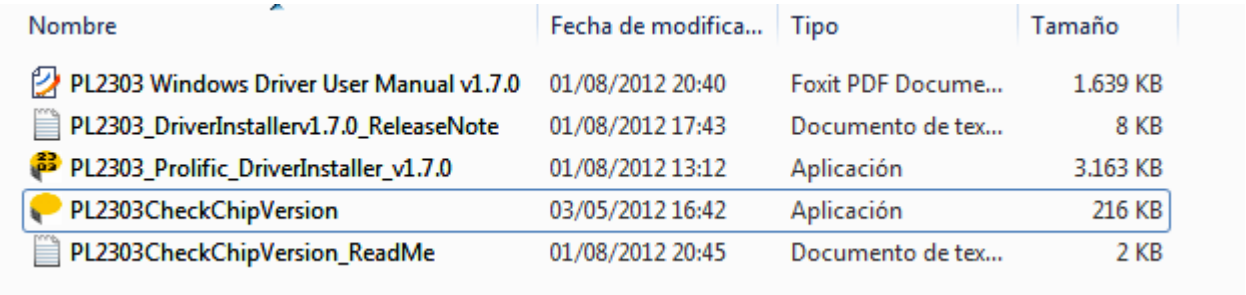
La instalación se ha completado con éxito

El siguiente paso es conectar el cable USB al programador K-150 y nuestro ordenador lo reconocerá, como en este caso: "**Prolific USB-to-Serial Comm Port (COM4)**":

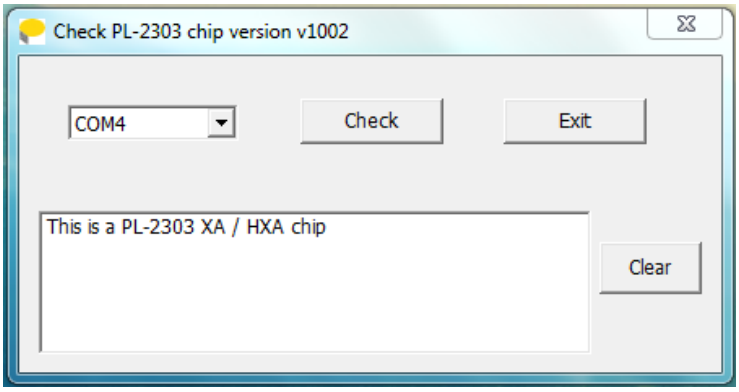


En el Administrador de dispositivo debe aparecer el driver instalado en **Puertos COM y LPT**

La compatibilidad depende de la versión del chip del programador, para saber que versión es la que tienes, debes de abrir el programa "**PL2303CheckChipVersion**", que se encuentra en la misma carpeta "**PIC Programmer Drivers**".



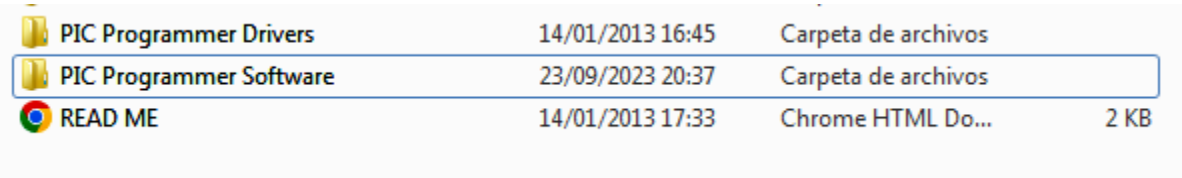
Abrimos la aplicación **PL2303CheckChipVersion** y nos aparece la siguiente ventana y seleccionamos el puerto y presionamos "**Check**":



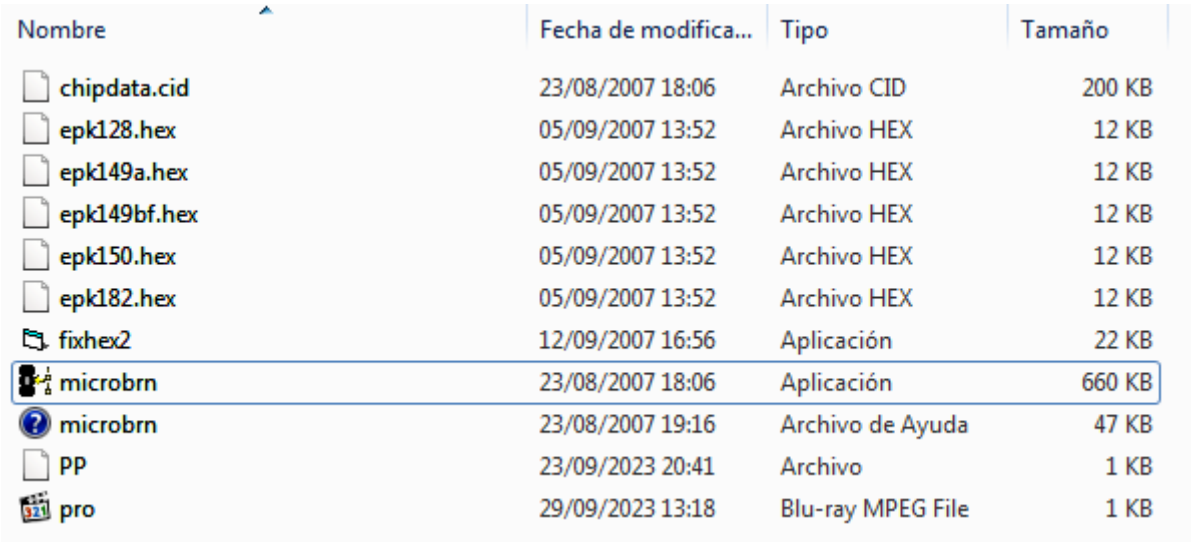
Nos aparece la versión del Chip es "**PL-2303 XA/HXA**", y como se puede observar es compatible en un 100% con Windows 7, pero no con Windows 8.1:

Chip Version	Windows 2000 / XP (32 & 64 bit)	Vista / Windows 7 (32 & 64 bit)	Windows 8 (32 & 64 bit)
PL-2303H	Yes	Yes	Not Supported*
PL-2303HX (Rev A) or HXA	Yes	Yes	Not Supported*
PL-2303X or XA	Yes	Yes	Not Supported*
PL-2303HX (Rev D) or HXD	Yes	Yes	Yes
PL2303TA	Yes	Yes	Yes
PL2303TB	Yes	Yes	Yes
PL2303EA	Yes	Yes	Yes
PL2303RA	Yes	Yes	Yes
PL2303SA	Yes	Yes	Yes

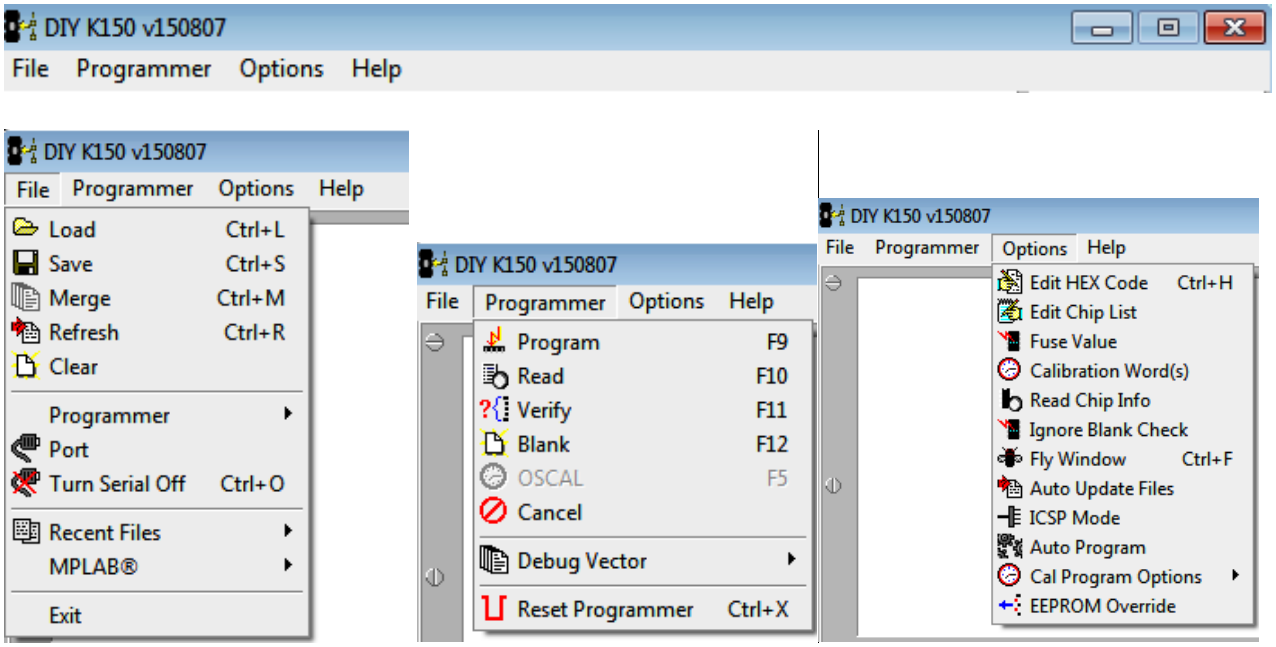
Una vez verificada la compatibilidad del Chip e instalados los drivers, estamos listos para programar nuestro PIC; para ello abriremos la carpeta "**PIC Programmer Software**"



y ejecutamos la aplicación "**microbrn**"

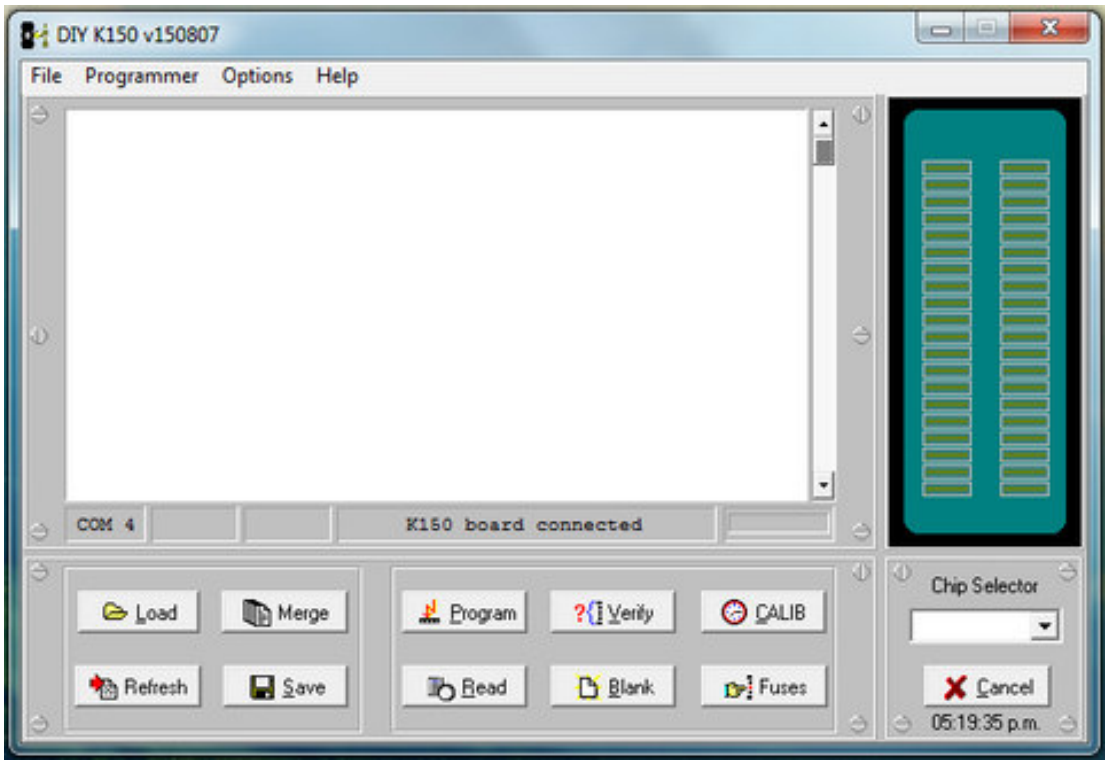


Al ejecutarse el software **microbrn** nos aparece un programador muy intuitivo y fácil de utilizar. En la parte superior de la ventana del programador nos aparece con una barra de herramientas en la cual podemos disponer de las siguientes pestañas: File-Programmer-Options-Help, y éstos a su vez con sus correspondientes opciones:



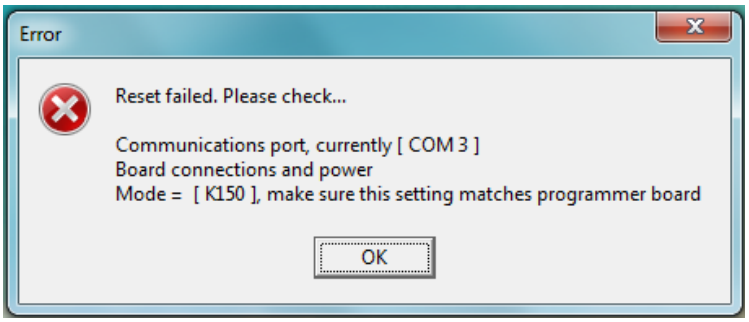
Al final de este capítulo se muestra un **Glosario de operandos** que describe cada uno de las operaciones que aparecen en el menú y en el submenú.

En la ventana principal disponemos de botones de acceso directo a las funciones más usadas, también podemos observar un diagrama que nos explica cómo debemos colocar nuestro microcontrolador en el zócalo ZIF de 40 pines.



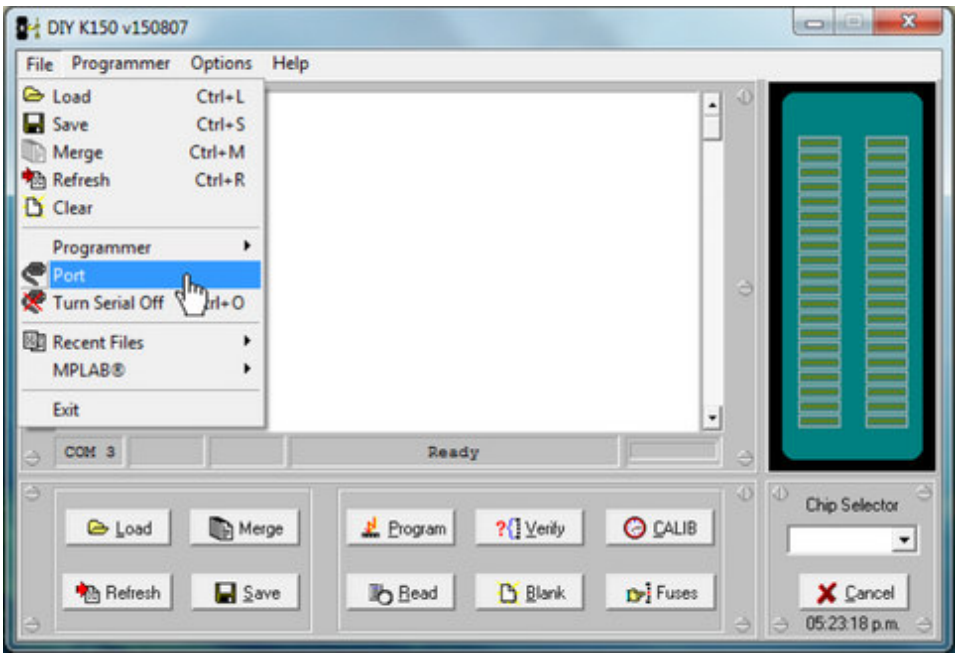
Ventana principal del K-150

Nos puede ocurrir que nos salga un error en la pantalla, normalmente porque no está configurado el puerto serie del programador dentro de la aplicación.



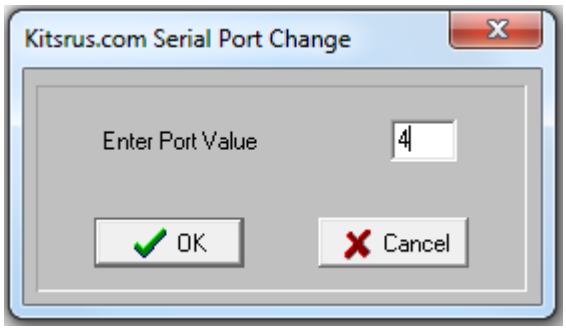
Error del puerto serie introducido

En este caso debemos presionar el botón "**OK**" y a continuación dirigirnos hacia la opción "**File**" y después "**Port**" para insertar el puerto serie que nos proporciona el ordenador:



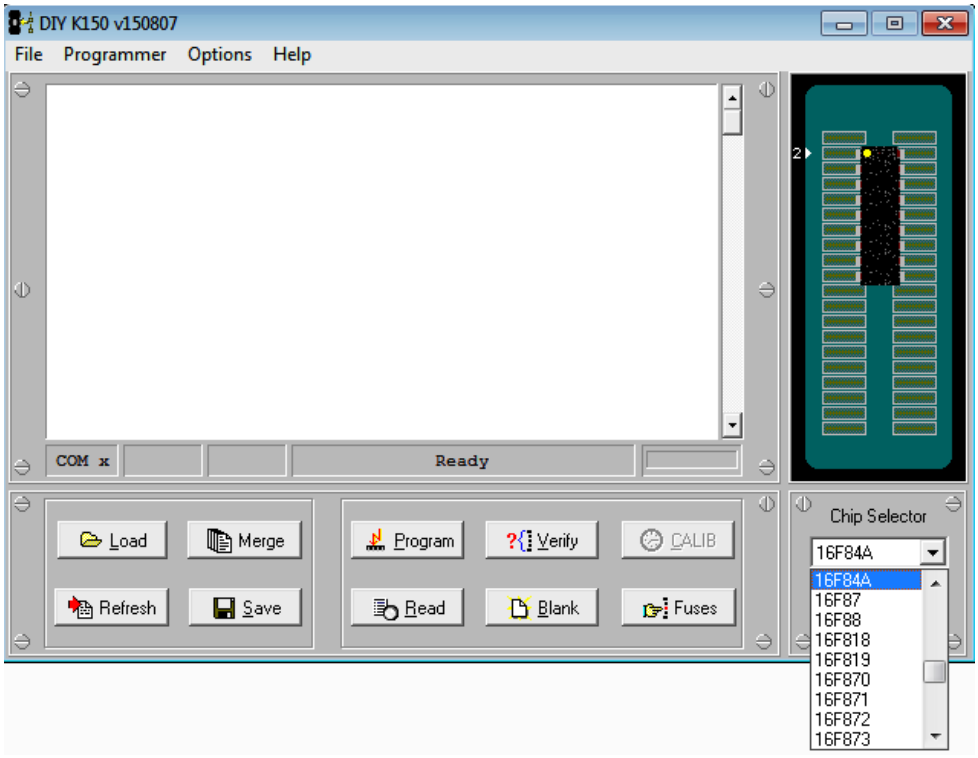
Abrimos la opción File y pulsamos en Port para añadir el valor del puerto serie

A continuación se desplegará la siguiente ventana, e introduciremos el número de puerto COM en que se encuentra conectado nuestro programador en la lista de dispositivos, en este caso introducimos el puerto serie **COM 4**, así que introduciremos "**4**" y presionamos "**OK**":

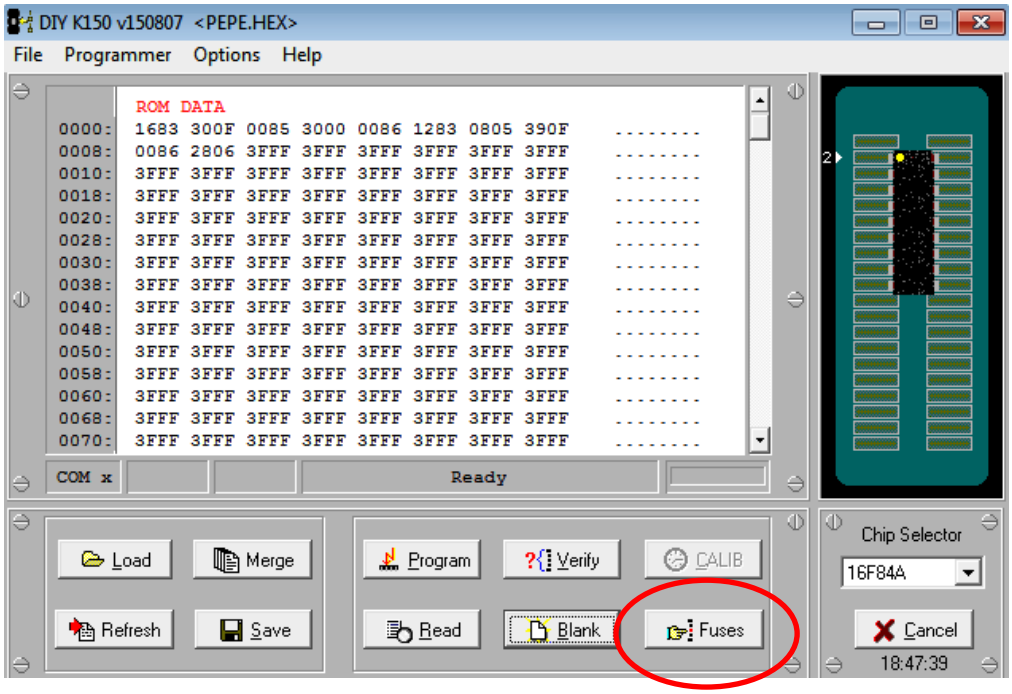


Introduciendo el valor del puerto serie que el sistema ha detectado

Después seleccionaremos el PIC que vamos a grabar a través de **Chip Selector**, en este caso el 16F84A, y lo posicionamos sobre el Zócalo ZIF de 40 pines, tal y como nos lo muestra el programa de la siguiente figura, a partir del pin 2, y estaremos listos para seleccionar y cargar el programa con el que programaremos nuestro PIC:

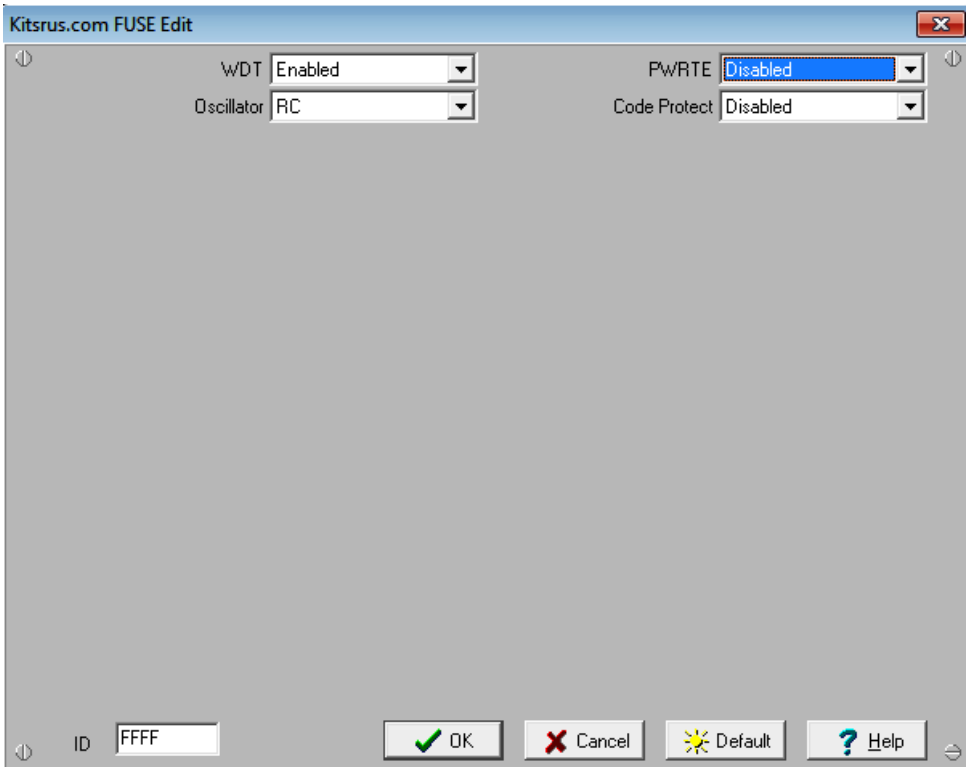


Seleccionamos el PIC que vamos a programar en **Chip Selector**



Si pulsamos el botón de "**Fuses**" nos permite editar la configuración de los "fuses" del microcontrolador y nos abre una ventana con las opciones de poder habilitar o deshabilitar el **WDT**, **PWRTE**, **Code Protect** y el **Oscilator** configurarlo entre RC/HS/XT/LP.

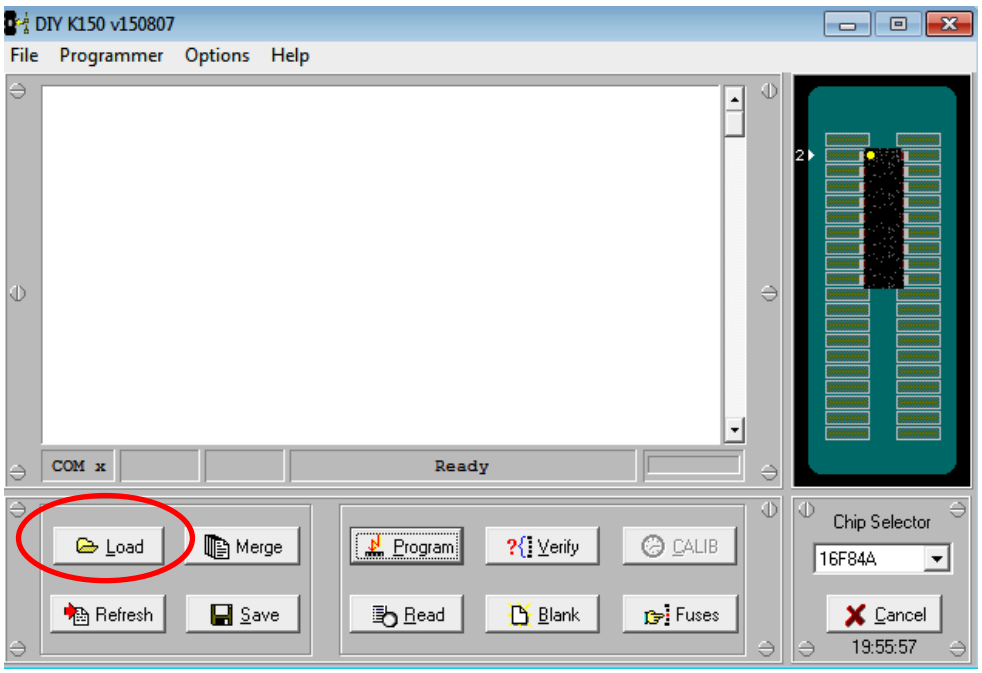
Si elegimos el RC montamos en el circuito electrónico una resistencia de 100K de VDD al pin 16 y un condensador a masa de 33pF, o si seleccionamos XT un cristal de cuarzo de 4MHz entre el pin 15 y 16 conectando 2 condensadores de 22pF a masa.



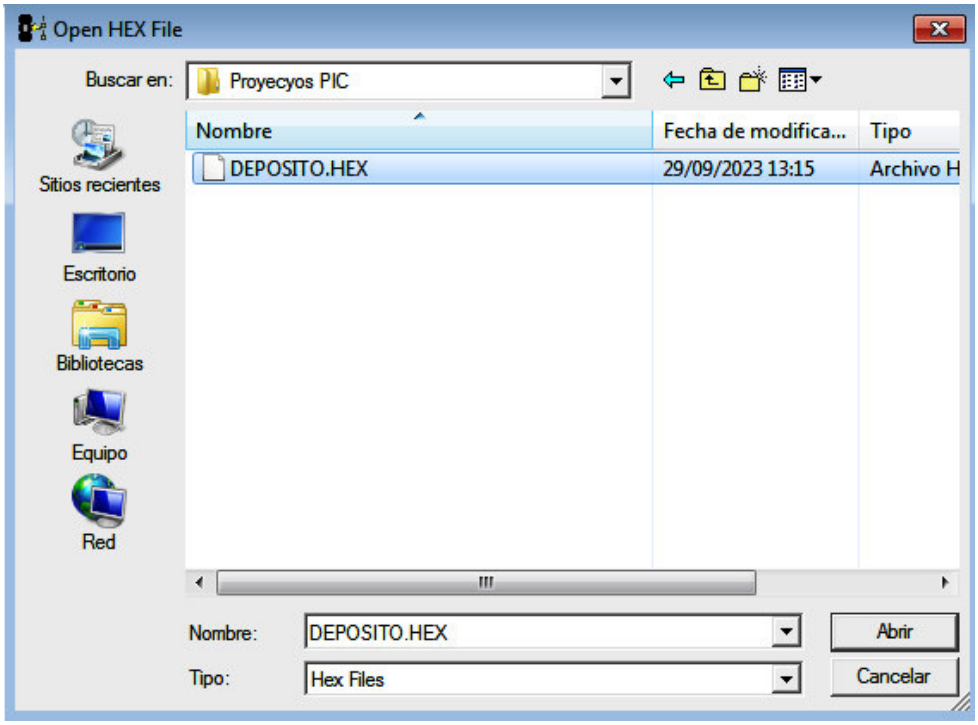
Configuración de los fuses

- **Pasos para la grabación del archivo hexadecimal en el programador**

1. Insertar el microcontrolador PIC16F84A en el zócalo ZIF (Fuerza de inserción cero) del programador K-150. El pin 1 del zócalo ZIF se encuentra al lado de la palanca de bloqueo y se inserta a partir del pin2.
2. Conectar el programador al puerto USB del ordenador. Windows 7 detecta automáticamente el puerto serie.
3. Abrimos el programa del programador K-150 que se encuentra en el directorio **PIC Programmer Software** y ejecutamos el archivo **microbrn**.
4. Si nos dá un problema de acceso al puerto serie tendremos que introducirlo en **File** y **port** e introducir el puerto serie.
5. Una vez que la aplicación reconozca el puerto serie seleccionamos el **Chip selector** en este caso el PIC **16F84A**.
6. Seguidamente pulsamos el botón "**Load**" y buscamos el archivo en hexadecimal que se encuentra en la carpeta de proyectos "DEPOSITO.HEX"



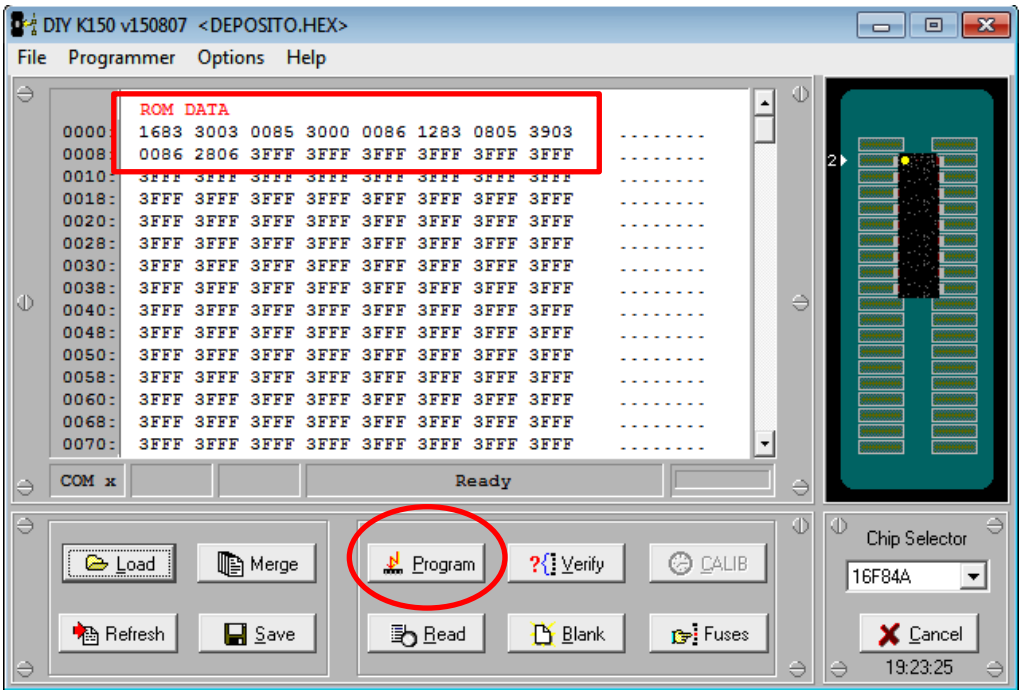
Cargamos el fichero hexadecimal pulsando el botón de Load



Seleccionamos el fichero hexadecimal y pulsamos abrir

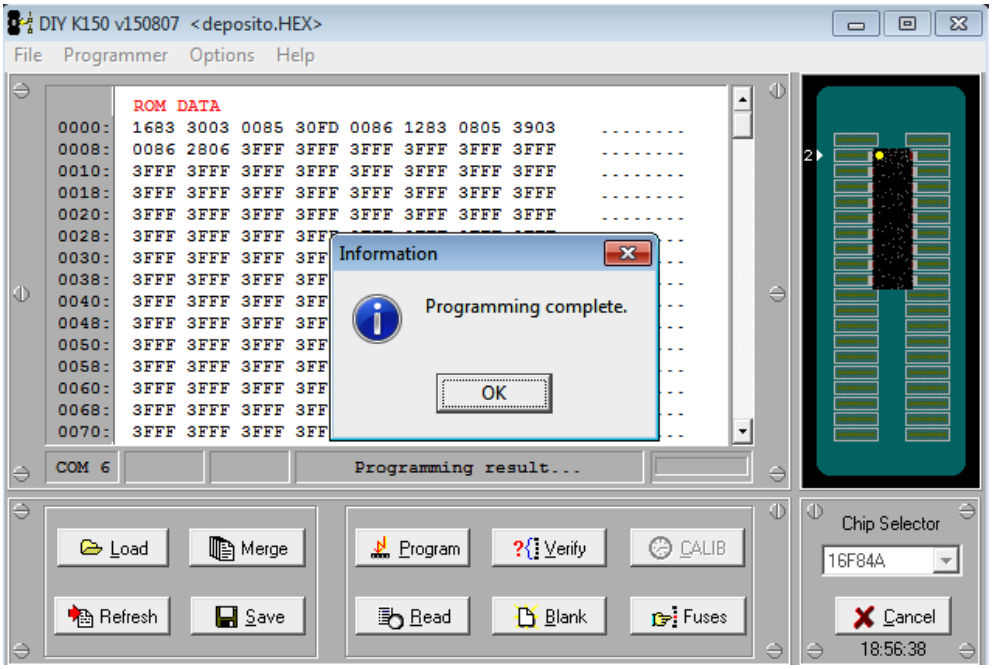
7. Una vez localizado el fichero hexadecimal pulsamos **Abrir** y nos aparece en la ventana del programador los datos del fichero en hexadecimal y observar que los datos en el inicio de la ROM DATA, tenga diferentes valores, por ejemplo: 1683, 3003, 0086, 1283, 0805, etc. Si aparece todo 3FFF, es que no se ha cargado

correctamente o se ha pulsado el botón **Verify**, que comprueba y verifica la memoria borrando los datos y si pulsamos **Program** no nos va a grabar nada.



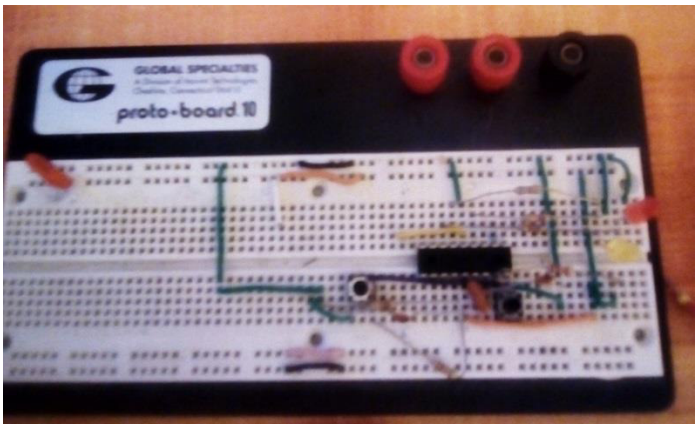
Pulsamos el botón **Program** para grabar el programa en el microcontrolador

8. Si todo está correcto pulsamos el botón de **Program** que graba el archivo hexadecimal en el microcontrolador, al transmitir los datos parpadea un diodo led rojo en la tarjeta K-150. Si no se ha producido ningún error el sistema responde con un **Programming complete**, pulsamos **OK** y ya tenemos grabado el programa en nuestro microcontrolador.



Respuesta del sistema cuando se ha completado la programación del microcontrolador

9. Quitamos el microcontrolador del zócalo zif, levantando la palanca, y lo insertamos en el circuito electrónico que se encuentra en la protoboard y comprobamos su funcionamiento.



- Glosario de operando

FILE	Menú con opciones para el manejo de archivos y configuración de la placa (tipo de programador y puerto COM)
PROGRAMMER	Comandos para la programación de los microcontroladores
OPTIONS	Opciones y herramientas varias que dispone el programador. Desde aquí se puede elegir por ejemplo, la programación ICSP
HELP	Menú de ayuda
LOAD	Carga el archivo HEX
MERGE	Une archivos HEX
REFRESH	Vuelve a cargar el archivo HEX
SAVE	Guarda lo que vemos en pantalla en un archivo HEX
PROGRAM	Graba el microcontrolador
VERIFY	Verifica el microcontrolador
CALIB	Lee el valor OSCAL de algunos microcontroladores
READ	Lee el contenido del microcontrolador
BLANK	Borra el contenido del microcontrolador
FUSES	Permite editar la configuración de los "fuses" del microcontrolador
CHIP SELECTOR	Lista desplegable para elegir el microcontrolador
CANCEL	Cancela la operación en curso

8. ANEXO. DATASHEET PIC16F84A

PIC16F84A

18-pin *Enhanced* Flash/EEPROM 8-Bit Microcontroller

Devices Included in this Data Sheet:

- PIC16F84A
- Extended voltage range device available (PIC16**LF**84A)

High Performance RISC CPU Features:

- Only 35 single word instructions to learn
- All instructions single cycle except for program branches which are two-cycle
- Operating speed: DC - 20 MHz clock input
  DC - 200 ns instruction cycle
- 1024 words of program memory
- 68 bytes of data RAM
- 64 bytes of data EEPROM
- 14-bit wide instruction words
- 8-bit wide data bytes
- 15 special function hardware registers
- Eight-level deep hardware stack
- Direct, indirect and relative addressing modes
- Four interrupt sources:
  - External RB0/INT pin
  - TMR0 timer overflow
  - PORTB<7:4> interrupt on change
  - Data EEPROM write complete

Peripheral Features:

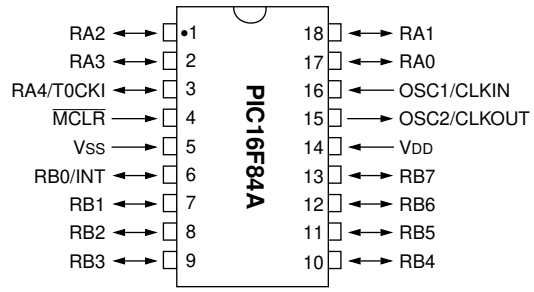
- 13 I/O pins with individual direction control
- High current sink/source for direct LED drive
  - 25 mA sink max. per pin
  - 25 mA source max. per pin
- TMR0: 8-bit timer/counter with 8-bit programmable prescaler

Special Microcontroller Features:

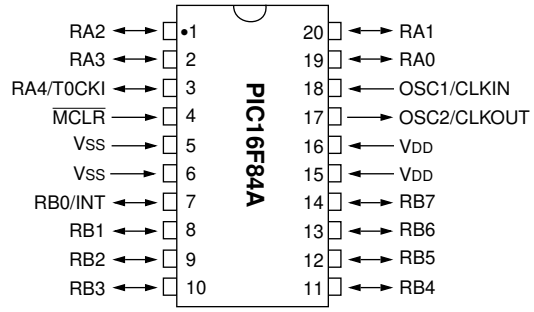
- 1000 erase/write cycles *Enhanced* Flash program memory
- 1,000,000 typical erase/write cycles EEPROM data memory
- EEPROM Data Retention > 40 years
- In-Circuit Serial Programming (ICSP™) - via two pins
- Power-on Reset (POR), Power-up Timer (PWRT), Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Code-protection
- Power saving SLEEP mode
- Selectable oscillator options

Pin Diagrams

PDIP, SOIC



SSOP



CMOS *Enhanced* Flash/EERPOM Technology:

- Low-power, high-speed technology
- Fully static design
- Wide operating voltage range:
  - Commercial: 2.0V to 5.5V
  - Industrial: 2.0V to 5.5V
- Low power consumption:
  - < 2 mA typical @ 5V, 4 MHz
  - 15 µA typical @ 2V, 32 kHz
  - < 0.5 µA typical standby current @ 2V

Table of Contents

To Our Valued Customers

Most Current Data Sheet

To obtain the most up-to-date version of this data sheet, please check our Worldwide Web site at:

http://www.microchip.com

You can determine the version of a data sheet by examining its literature number found on the bottom outside corner of any page. The last character of the literature number is the version number. e.g., DS30000A is version A of document DS30000.

Errata

An errata sheet may exist for current devices, describing minor operational differences (from the data sheet) and recommended workarounds. As device/documentation issues become known to us, we will publish an errata sheet. The errata will specify the revision of silicon and revision of document to which it applies.

To determine if an errata sheet exists for a particular device, please check with one of the following:

- Microchip's Worldwide Web site; http://www.microchip.com
- Your local Microchip sales office (see last page)
- The Microchip Corporate Literature Center; U.S. FAX: (602) 786-7277

When contacting a sales office or the literature center, please specify which device, revision of silicon and data sheet (include literature number) you are using.

Corrections to this Data Sheet

We constantly strive to improve the quality of all our products and documentation. We have spent a great deal of time to ensure that this document is correct. However, we realize that we may have missed a few things. If you find any information that is missing or appears in error, please:

- Fill out and mail in the reader response form in the back of this data sheet.
- E-mail us at webmaster@microchip.com.

We appreciate your assistance in making this a better document.

1.0 DEVICE OVERVIEW

This document contains device-specific information for the operation of the PIC16F84A device. Additional information may be found in the PICmicro™ Mid-Range Reference Manual, (DS33023), which may be downloaded from the Microchip website. The Reference Manual should be considered a complementary document to this data sheet, and is highly recommended reading for a better understanding of the device architecture and operation of the peripheral modules.

The PIC16F84A belongs to the mid-range family of the PICmicro™ microcontroller devices. A block diagram of the device is shown in Figure 1-1.

The program memory contains 1K words, which translates to 1024 instructions, since each 14-bit program memory word is the same width as each device instruction. The data memory (RAM) contains 68 bytes. Data EEPROM is 64 bytes.

There are also 13 I/O pins that are user-configured on a pin-to-pin basis. Some pins are multiplexed with other device functions. These functions include:

- External interrupt
- Change on PORTB interrupt
- Timer0 clock input

Table 1-1 details the pinout of the device with descriptions and details for each pin.

FIGURE 1-1: PIC16F84A BLOCK DIAGRAM

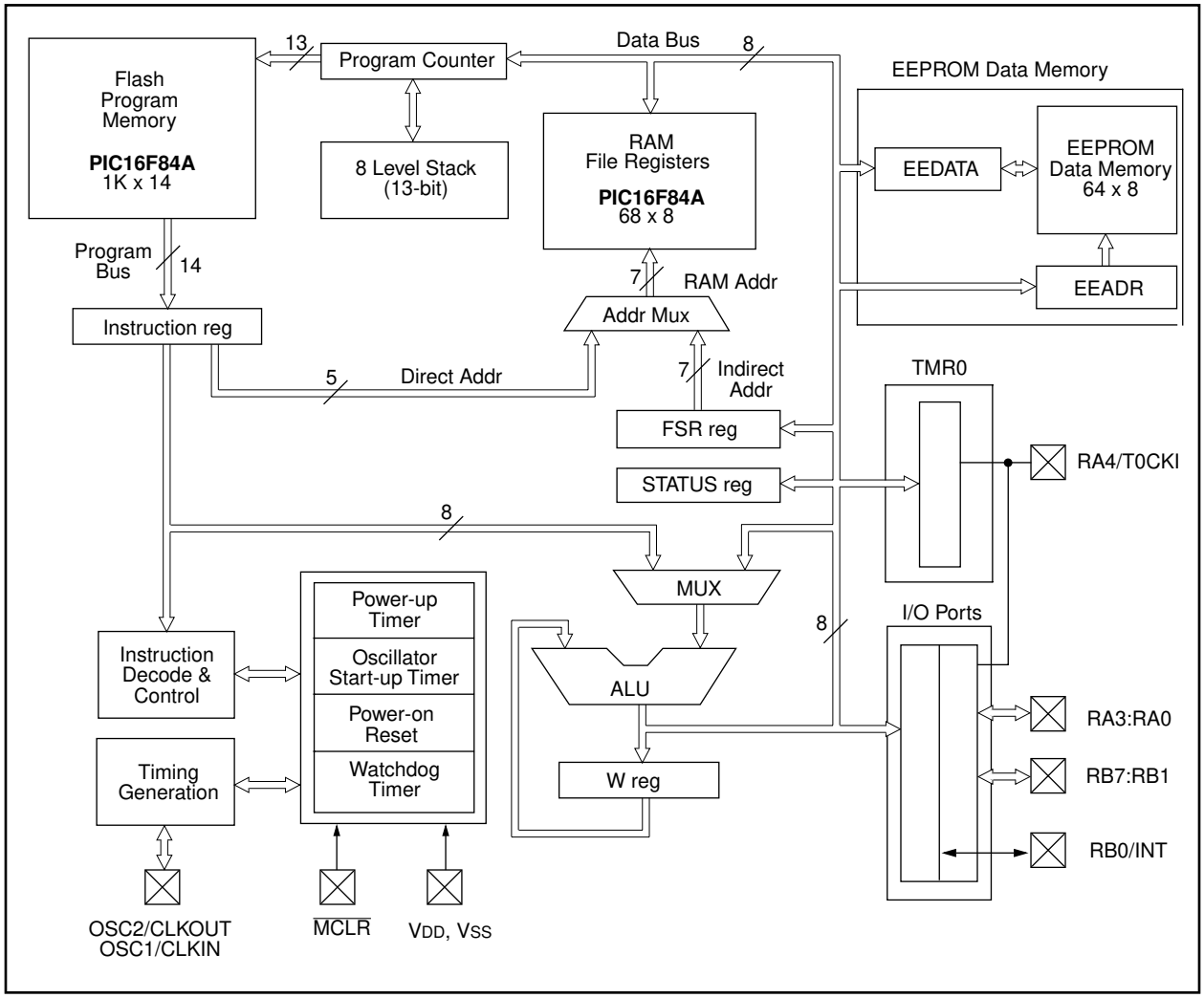


TABLE 1-1 PIC16F84A PINOUT DESCRIPTION

Pin Name	DIP No.	SOIC No.	SSOP No.	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	16	16	18	I	ST/CMOS (3)	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	15	15	19	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
$\overline{MCLR}$	4	4	4	I/P	ST	Master clear (reset) input/programming voltage input. This pin is an active low reset to the device.
						PORTA is a bi-directional I/O port.
RA0	17	17	19	I/O	TTL	
RA1	18	18	20	I/O	TTL	
RA2	1	1	1	I/O	TTL	
RA3	2	2	2	I/O	TTL	
RA4/T0CKI	3	3	3	I/O	ST	Can also be selected to be the clock input to the TMR0 timer/counter. Output is open drain type.
						PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.
RB0/INT	6	6	7	I/O	TTL/ST (1)	RB0/INT can also be selected as an external interrupt pin.
RB1	7	7	8	I/O	TTL	
RB2	8	8	9	I/O	TTL	
RB3	9	9	10	I/O	TTL	
RB4	10	10	11	I/O	TTL	Interrupt on change pin.
RB5	11	11	12	I/O	TTL	Interrupt on change pin.
RB6	12	12	13	I/O	TTL/ST (2)	Interrupt on change pin. Serial programming clock.
RB7	13	13	14	I/O	TTL/ST (2)	Interrupt on change pin. Serial programming data.
VSS	5	5	5,6	P	—	Ground reference for logic and I/O pins.
VDD	14	14	15,16	P	—	Positive supply for logic and I/O pins.

Legend: I= input O = output I/O = Input/Output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in serial programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

2.0 MEMORY ORGANIZATION

There are two memory blocks in the PIC16F84A. These are the program memory and the data memory. Each block has its own bus, so that access to each block can occur during the same oscillator cycle.

The data memory can further be broken down into the general purpose RAM and the Special Function Registers (SFRs). The operation of the SFRs that control the “core” are described here. The SFRs used to control the peripheral modules are described in the section discussing each individual peripheral module.

The data memory area also contains the data EEPROM memory. This memory is not directly mapped into the data memory, but is indirectly mapped. That is, an indirect address pointer specifies the address of the data EEPROM memory to read/write. The 64 bytes of data EEPROM memory have the address range 0h-3Fh. More details on the EEPROM memory can be found in Section 5.0.

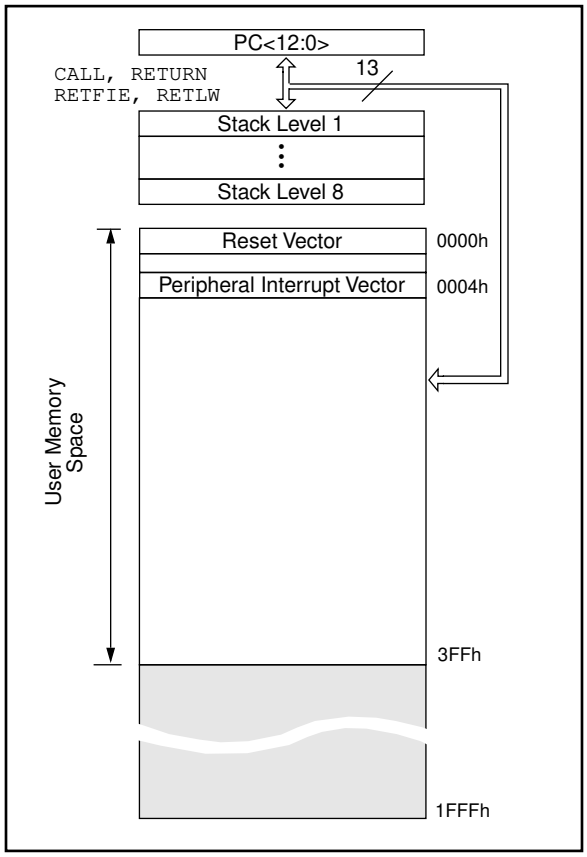
Additional information on device memory may be found in the PICmicro™ Mid-Range Reference Manual, (DS33023).

2.1 Program Memory Organization

The PIC16FXX has a 13-bit program counter capable of addressing an 8K x 14 program memory space. For the PIC16F84A, the first 1K x 14 (0000h-03FFh) are physically implemented (Figure 2-1). Accessing a location above the physically implemented address will cause a wraparound. For example, for locations 20h, 420h, 820h, C20h, 1020h, 1420h, 1820h, and 1C20h will be the same instruction.

The reset vector is at 0000h and the interrupt vector is at 0004h.

FIGURE 2-1: PROGRAM MEMORY MAP AND STACK - PIC16F84A



2.2 Data Memory Organization

The data memory is partitioned into two areas. The first is the Special Function Registers (SFR) area, while the second is the General Purpose Registers (GPR) area. The SFRs control the operation of the device.

Portions of data memory are banked. This is for both the SFR area and the GPR area. The GPR area is banked to allow greater than 116 bytes of general purpose RAM. The banked areas of the SFR are for the registers that control the peripheral functions. Banking requires the use of control bits for bank selection. These control bits are located in the STATUS Register. Figure 2-1 shows the data memory map organization.

Instructions MOVWF and MOVF can move values from the W register to any location in the register file ("F"), and vice-versa.

The entire data memory can be accessed either directly using the absolute address of each register file or indirectly through the File Select Register (FSR) (Section 2.4). Indirect addressing uses the present value of the RP0 bit for access into the banked areas of data memory.

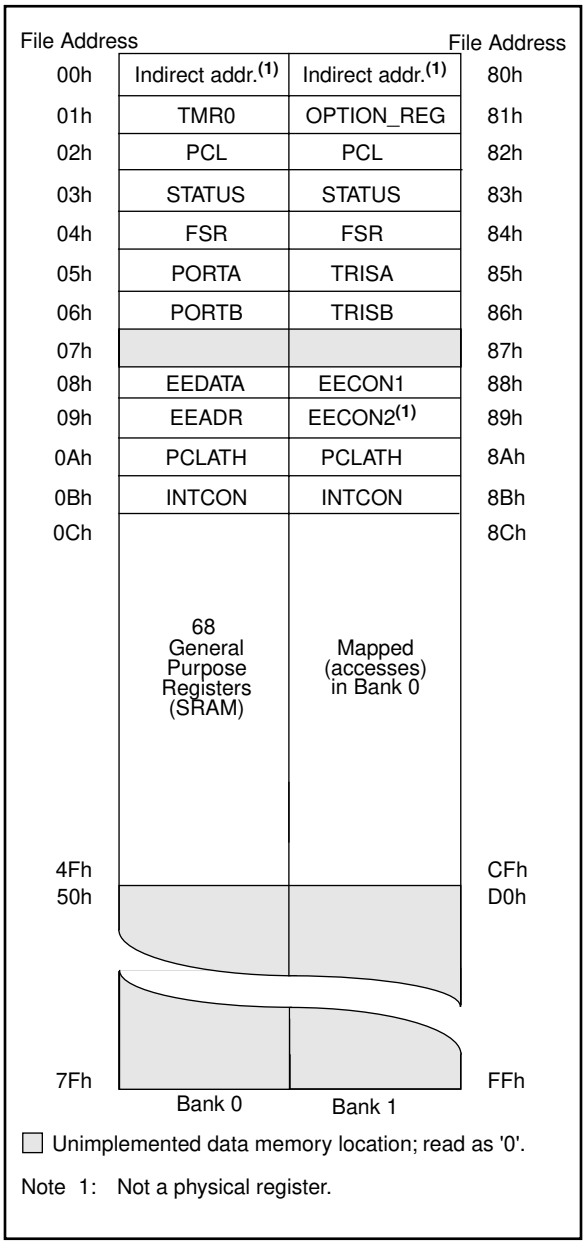
Data memory is partitioned into two banks which contain the general purpose registers and the special function registers. Bank 0 is selected by clearing the RP0 bit (STATUS<5>). Setting the RP0 bit selects Bank 1. Each Bank extends up to 7Fh (128 bytes). The first twelve locations of each Bank are reserved for the Special Function Registers. The remainder are General Purpose Registers implemented as static RAM.

2.2.1 GENERAL PURPOSE REGISTER FILE

Each General Purpose Register (GPR) is 8 bits wide and is accessed either directly or indirectly through the FSR (Section 2.4).

The GPR addresses in bank 1 are mapped to addresses in bank 0. As an example, addressing location 0Ch or 8Ch will access the same GPR.

FIGURE 2-1: REGISTER FILE MAP - PIC16F84A



2.2.2 SPECIAL FUNCTION REGISTERS

The Special Function Registers (Figure 2-1 and Table 2-1) are used by the CPU and Peripheral functions to control the device operation. These registers are static RAM.

The special function registers can be classified into two sets, core and peripheral. Those associated with the core functions are described in this section. Those related to the operation of the peripheral features are described in the section for that specific feature.

TABLE 2-1 REGISTER FILE SUMMARY

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other resets (Note3)
Bank 0											
00h	INDF	Uses contents of FSR to address data memory (not a physical register)								---- ----	---- ----
01h	TMR0	8-bit real-time clock/counter								xxxx xxxx	uuuu uuuu
02h	PCL	Low order 8 bits of the Program Counter (PC)								0000 0000	0000 0000
03h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	000q quuu
04h	FSR	Indirect data memory address pointer 0								xxxx xxxx	uuuu uuuu
05h	PORTA (4)	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	---u uuuu
06h	PORTB (5)	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	uuuu uuuu
07h		Unimplemented location, read as '0'								---- ----	---- ----
08h	EEDATA	EEPROM data register								xxxx xxxx	uuuu uuuu
09h	EEADR	EEPROM address register								xxxx xxxx	uuuu uuuu
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC (1)					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
Bank 1											
80h	INDF	Uses contents of FSR to address data memory (not a physical register)								---- ----	---- ----
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
82h	PCL	Low order 8 bits of Program Counter (PC)								0000 0000	0000 0000
83h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	000q quuu
84h	FSR	Indirect data memory address pointer 0								xxxx xxxx	uuuu uuuu
85h	TRISA	—	—	—	PORTA data direction register					---1 1111	---1 1111
86h	TRISB	PORTB data direction register								1111 1111	1111 1111
87h		Unimplemented location, read as '0'								---- ----	---- ----
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	---0 q000
89h	EECON2	EEPROM control register 2 (not a physical register)								---- ----	---- ----
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC (1)					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u

Legend: x = unknown, u = unchanged. - = unimplemented read as '0', q = value depends on condition.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a slave register for PC<12:8>. The contents of PCLATH can be transferred to the upper byte of the program counter, but the contents of PC<12:8> is never transferred to PCLATH.

2: The $\overline{TO}$ and $\overline{PD}$ status bits in the STATUS register are not affected by a $\overline{MCLR}$ reset.

3: Other (non power-up) resets include: external reset through $\overline{MCLR}$ and the Watchdog Timer Reset.

4: On any device reset, these pins are configured as inputs.

5: This is the value that will be in the port output latch.

2.2.2.1 STATUS REGISTER

The STATUS register contains the arithmetic status of the ALU, the RESET status and the bank select bit for data memory.

As with any register, the STATUS register can be the destination for any instruction. If the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. These bits are set or cleared according to device logic. Furthermore, the $\overline{TO}$ and $\overline{PD}$ bits are not writable. Therefore, the result of an instruction with the STATUS register as destination may be different than intended.

For example, `CLRF STATUS` will clear the upper-three bits and set the Z bit. This leaves the STATUS register as `000u u1uu` (where `u` = unchanged).

Only the `BCF`, `BSF`, `SWAPF` and `MOVWF` instructions should be used to alter the STATUS register (Table 7-2) because these instructions do not affect any status bit.

Note 1:	The IRP and RP1 bits (STATUS<7:6>) are not used by the PIC16F84A and should be programmed as cleared. Use of these bits as general purpose R/W bits is NOT recommended, since this may affect upward compatibility with future products.
Note 2:	The C and DC bits operate as a $\overline{borrow}$ and digit borrow out bit, respectively, in subtraction. See the `SUBLW` and `SUBWF` instructions for examples.
Note 3:	When the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. The specified bit(s) will be updated according to device logic

FIGURE 2-1: STATUS REGISTER (ADDRESS 03h, 83h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

bit 7: **IRP**: Register Bank Select bit (used for indirect addressing)
The IRP bit is not used by the PIC16F84A. IRP should be maintained clear.

bit 6-5: **RP1:RP0**: Register Bank Select bits (used for direct addressing)
00 = Bank 0 (00h - 7Fh)
01 = Bank 1 (80h - FFh)
Each bank is 128 bytes. Only bit RP0 is used by the PIC16F84A. RP1 should be maintained clear.

bit 4: **$\overline{TO}$**: Time-out bit
1 = After power-up, `CLRWDT` instruction, or `SLEEP` instruction
0 = A WDT time-out occurred

bit 3: **$\overline{PD}$**: Power-down bit
1 = After power-up or by the `CLRWDT` instruction
0 = By execution of the `SLEEP` instruction

bit 2: **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1: **DC**: Digit carry/$\overline{borrow}$ bit (for `ADDWF` and `ADDLW` instructions) (For $\overline{borrow}$ the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0: **C**: Carry/$\overline{borrow}$ bit (for `ADDWF` and `ADDLW` instructions)
1 = A carry-out from the most significant bit of the result occurred
0 = No carry-out from the most significant bit of the result occurred
Note: For $\overline{borrow}$ the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (`RRF`, `RLF`) instructions, this bit is loaded with either the high or low order bit of the source register.

2.2.2.2 OPTION_REG REGISTER

The OPTION_REG register is a readable and writable register which contains various control bits to configure the TMR0/WDT prescaler, the external INT interrupt, TMR0, and the weak pull-ups on PORTB.

Note:	When the prescaler is assigned to the WDT (PSA = '1'), TMR0 has a 1:1 prescaler assignment.

FIGURE 2-1: OPTION_REG REGISTER (ADDRESS 81h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit7							bit0

R	= Readable bit
W	= Writable bit
U	= Unimplemented bit, read as '0'
- n	= Value at POR reset

bit 7: **$\overline{\text{RBPU}}$**: PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled (by individual port latch values)

bit 6: **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5: **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4: **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3: **PSA**: Prescaler Assignment bit
1 = Prescaler assigned to the WDT
0 = Prescaler assigned to TMR0

bit 2-0: **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

2.2.2.3 INTCON REGISTER

The INTCON register is a readable and writable register which contains the various enable bits for all interrupt sources.

Note:	Interrupt flag bits get set when an interrupt condition occurs regardless of the state of its corresponding enable bit or the global enable bit, GIE (INTCON<7>).

FIGURE 2-1: INTCON REGISTER (ADDRESS 0Bh, 8Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

bit 7: **GIE:** Global Interrupt Enable bit
1 = Enables all un-masked interrupts
0 = Disables all interrupts

Note: For the operation of the interrupt structure, please refer to Section •.

bit 6: **EEIE**: EE Write Complete Interrupt Enable bit
1 = Enables the EE write complete interrupt
0 = Disables the EE write complete interrupt

bit 5: **T0IE**: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4: **INTE**: RB0/INT Interrupt Enable bit
1 = Enables the RB0/INT interrupt
0 = Disables the RB0/INT interrupt

bit 3: **RBIE**: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2: **T0IF**: TMR0 Overflow Interrupt Flag bit
1 = TMR0 has overflowed (must be cleared in software)
0 = TMR0 did not overflow

bit 1: **INTF**: RB0/INT Interrupt Flag bit
1 = The RB0/INT interrupt occurred
0 = The RB0/INT interrupt did not occur

bit 0: **RBIF**: RB Port Change Interrupt Flag bit
1 = When at least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

2.3 PCL and PCLATH

The program counter (PC) specifies the address of the instruction to fetch for execution. The PC is 13 bits wide. The low byte is called the PCL register. This register is readable and writable. The high byte is called the PCH register. This register contains the PC<12:8> bits and is not directly readable or writable. All updates to the PCH register go through the PCLATH register.

2.3.1 STACK

The stack allows a combination of up to 8 program calls and interrupts to occur. The stack contains the return address from this branch in program execution.

Midrange devices have an 8 level deep x 13-bit wide hardware stack. The stack space is not part of either program or data space and the stack pointer is not readable or writable. The PC is PUSHed onto the stack when a `CALL` instruction is executed or an interrupt causes a branch. The stack is POPed in the event of a `RETURN`, `RETLW` or a `RETFIE` instruction execution. PCLATH is not modified when the stack is PUSHed or POPed.

After the stack has been PUSHed eight times, the ninth push overwrites the value that was stored from the first push. The tenth push overwrites the second push (and so on).

2.4 Indirect Addressing; INDF and FSR Registers

The INDF register is not a physical register. Addressing INDF actually addresses the register whose address is contained in the FSR register (FSR is a *pointer*). This is indirect addressing.

EXAMPLE 2-1: INDIRECT ADDRESSING

- Register file 05 contains the value 10h
- Register file 06 contains the value 0Ah
- Load the value 05 into the FSR register
- A read of the INDF register will return the value of 10h
- Increment the value of the FSR register by one (FSR = 06)
- A read of the INDF register now will return the value of 0Ah.

Reading INDF itself indirectly (FSR = 0) will produce 00h. Writing to the INDF register indirectly results in a no-operation (although STATUS bits may be affected).

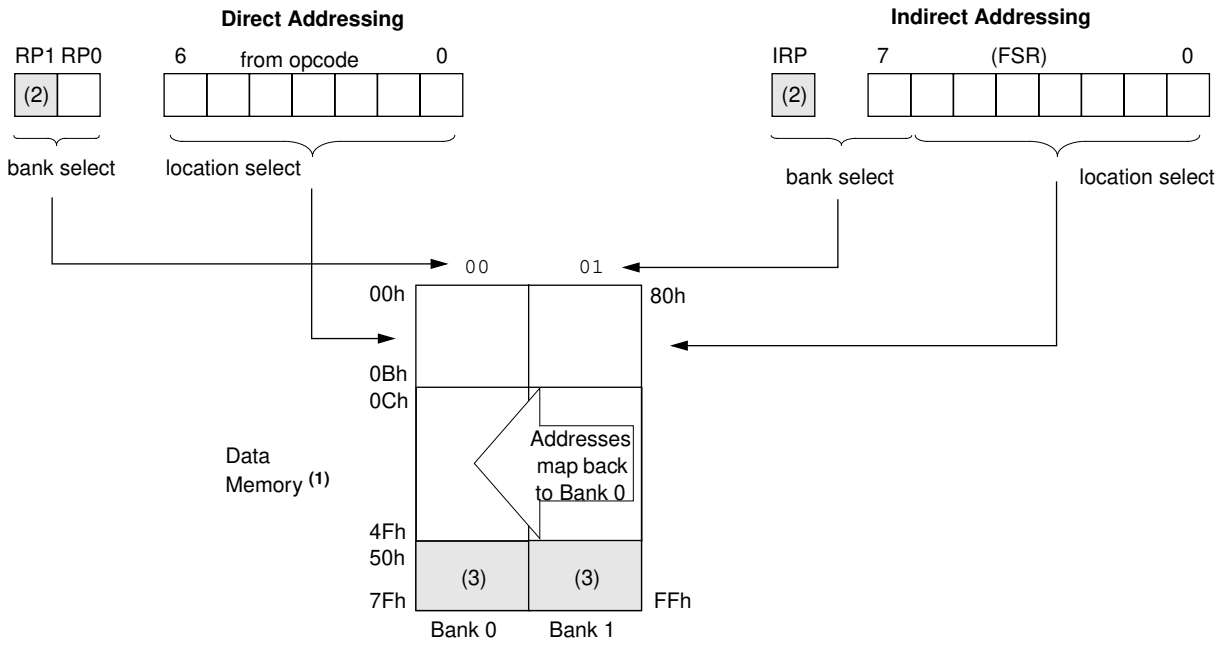
A simple program to clear RAM locations 20h-2Fh using indirect addressing is shown in Example 2-2.

EXAMPLE 2-2: HOW TO CLEAR RAM USING INDIRECT ADDRESSING

```
          movlw  0x20  ;initialize pointer
          movwf  FSR   ;  to RAM
NEXT      clrf   INDF  ;clear INDF register
          incf   FSR   ;inc pointer
          btfss  FSR,4 ;all done?
          goto   NEXT  ;NO, clear next
CONTINUE
          :            ;YES, continue
```

An effective 9-bit address is obtained by concatenating the 8-bit FSR register and the IRP bit (STATUS<7>), as shown in Figure 2-1. However, IRP is not used in the PIC16F84A.

FIGURE 2-1: DIRECT/INDIRECT ADDRESSING



Note 1: For memory map detail see Figure 2-1.
2: Maintain as clear for upward compatiblity with future products.
3: Not implemented.

3.0 I/O PORTS

Some pins for these I/O ports are multiplexed with an alternate function for the peripheral features on the device. In general, when a peripheral is enabled, that pin may not be used as a general purpose I/O pin.

Additional information on I/O ports may be found in the PICmicro™ Mid-Range Reference Manual, (DS33023).

3.1 PORTA and TRISA Registers

PORTA is a 5-bit wide bi-directional port. The corresponding data direction register is TRISA. Setting a TRISA bit (=1) will make the corresponding PORTA pin an input, i.e., put the corresponding output driver in a hi-impedance mode. Clearing a TRISA bit (=0) will make the corresponding PORTA pin an output, i.e., put the contents of the output latch on the selected pin.

Note:	On a Power-on Reset, these pins are configured as inputs and read as '0'.

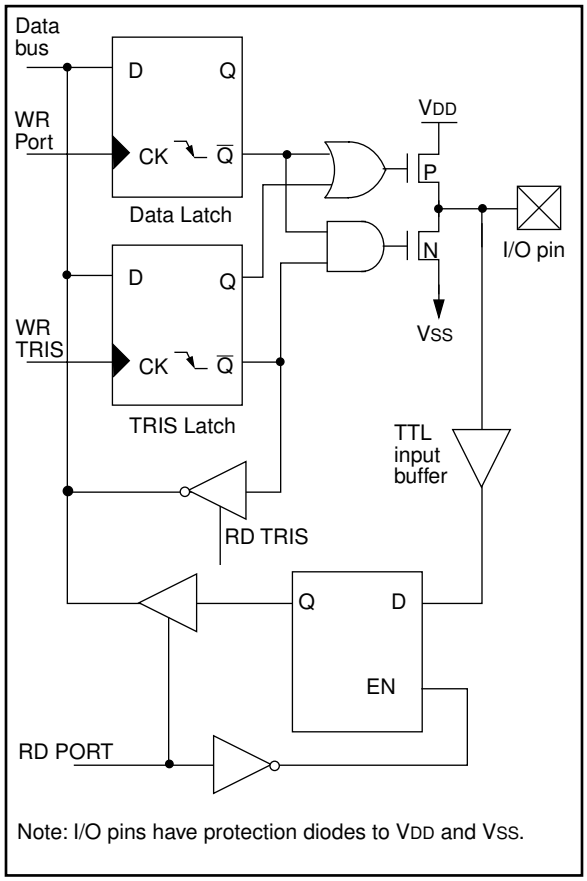
Reading the PORTA register reads the status of the pins whereas writing to it will write to the port latch. All write operations are read-modify-write operations. Therefore a write to a port implies that the port pins are read, this value is modified, and then written to the port data latch.

Pin RA4 is multiplexed with the Timer0 module clock input to become the RA4/T0CKI pin. The RA4/T0CKI pin is a Schmitt Trigger input and an open drain output. All other RA port pins have TTL input levels and full CMOS output drivers.

EXAMPLE 3-1: INITIALIZING PORTA

```
BCF     STATUS, RP0  ;
CLRF    PORTA        ; Initialize PORTA by
                     ; clearing output
                     ; data latches
BSF     STATUS, RP0  ; Select Bank 1
MOVLW   0x0F         ; Value used to
                     ; initialize data
                     ; direction
MOVWF   TRISA        ; Set RA<3:0> as inputs
                     ; RA4 as output
                     ; TRISA<7:5> are always
                     ; read as '0'.
```

FIGURE 3-1: BLOCK DIAGRAM OF PINS RA3:RA0



Note: I/O pins have protection diodes to VDD and VSS.

FIGURE 3-2: BLOCK DIAGRAM OF PIN RA4

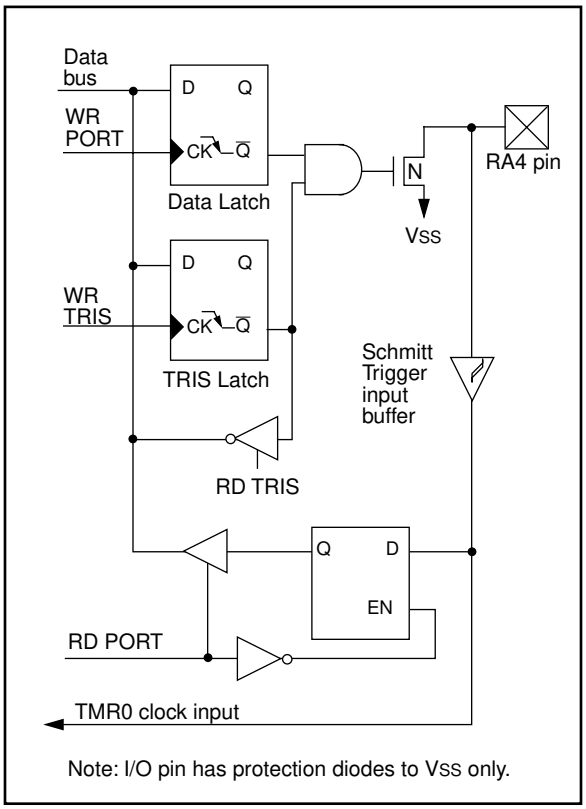


TABLE 3-1 PORTA FUNCTIONS

Name	Bit0	Buffer Type	Function
RA0	bit0	TTL	Input/output
RA1	bit1	TTL	Input/output
RA2	bit2	TTL	Input/output
RA3	bit3	TTL	Input/output
RA4/T0CKI	bit4	ST	Input/output or external clock input for TMR0. Output is open drain type.

Legend: TTL = TTL input, ST = Schmitt Trigger input

TABLE 3-2 SUMMARY OF REGISTERS ASSOCIATED WITH PORTA

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other resets
05h	PORTA	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	---u uuuu
85h	TRISA	—	—	—	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	---1 1111	---1 1111

Legend: x = unknown, u = unchanged, - = unimplemented read as '0'. Shaded cells are unimplemented, read as '0'

3.2 PORTB and TRISB Registers

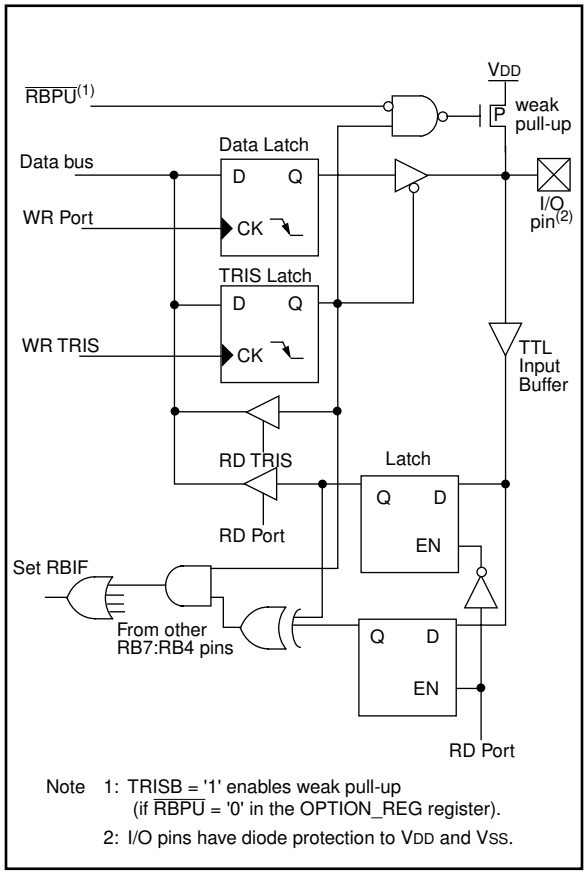
PORTB is an 8-bit wide bi-directional port. The corresponding data direction register is TRISB. Setting a TRISB bit (=1) will make the corresponding PORTB pin an input, i.e., put the corresponding output driver in a hi-impedance mode. Clearing a TRISB bit (=0) will make the corresponding PORTB pin an output, i.e., put the contents of the output latch on the selected pin.

EXAMPLE 3-1: INITIALIZING PORTB

```
BCF    STATUS, RP0  ;
CLRF   PORTB        ; Initialize PORTB by
                    ; clearing output
                    ; data latches
BSF    STATUS, RP0  ; Select Bank 1
MOVLW  0xCF         ; Value used to
                    ; initialize data
                    ; direction
MOVWF  TRISB        ; Set RB<3:0> as inputs
                    ; RB<5:4> as outputs
                    ; RB<7:6> as inputs
```

Each of the PORTB pins has a weak internal pull-up. A single control bit can turn on all the pull-ups. This is performed by clearing bit $\overline{\text{RBPU}}$ (OPTION<7>). The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a Power-on Reset.

FIGURE 3-3: BLOCK DIAGRAM OF PINS RB7:RB4



Note 1: TRISB = '1' enables weak pull-up (if $\overline{\text{RBPU}}$ = '0' in the OPTION_REG register).
2: I/O pins have diode protection to VDD and VSS.

Four of PORTB's pins, RB7:RB4, have an interrupt on change feature. Only pins configured as inputs can cause this interrupt to occur (i.e. any RB7:RB4 pin configured as an output is excluded from the interrupt on change comparison). The input pins (of RB7:RB4) are compared with the old value latched on the last read of PORTB. The "mismatch" outputs of RB7:RB4 are OR'ed together to generate the RB Port Change Interrupt with flag bit RBIF (INTCON<0>).

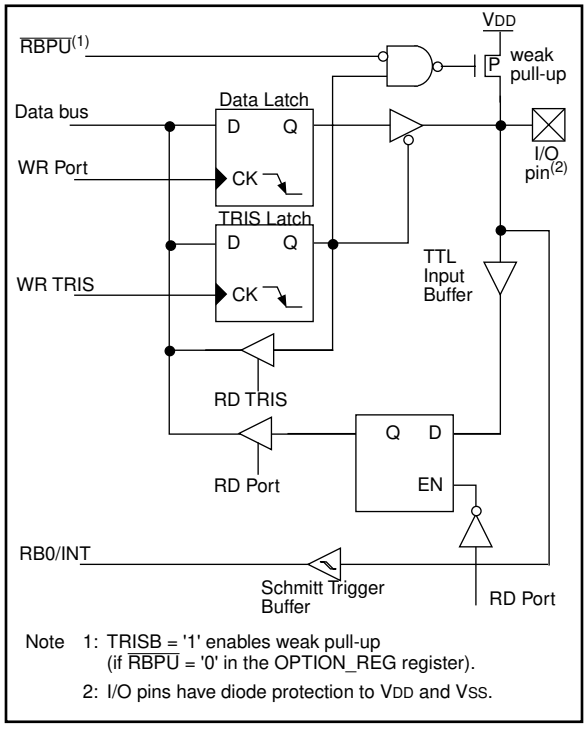
This interrupt can wake the device from SLEEP. The user, in the interrupt service routine, can clear the interrupt in the following manner:

a) Any read or write of PORTB. This will end the mismatch condition.
b) Clear flag bit RBIF.

A mismatch condition will continue to set flag bit RBIF. Reading PORTB will end the mismatch condition, and allow flag bit RBIF to be cleared.

The interrupt on change feature is recommended for wake-up on key depression operation and operations where PORTB is only used for the interrupt on change feature. Polling of PORTB is not recommended while using the interrupt on change feature.

FIGURE 3-4: BLOCK DIAGRAM OF PINS RB3:RB0



Note 1: TRISB = '1' enables weak pull-up (if $\overline{\text{RBPU}}$ = '0' in the OPTION_REG register).
2: I/O pins have diode protection to VDD and VSS.

TABLE 3-3 PORTB FUNCTIONS

Name	Bit	Buffer Type	I/O Consistency Function
RB0/INT	bit0	TTL/ST(1)	Input/output pin or external interrupt input. Internal software programmable weak pull-up.
RB1	bit1	TTL	Input/output pin. Internal software programmable weak pull-up.
RB2	bit2	TTL	Input/output pin. Internal software programmable weak pull-up.
RB3	bit3	TTL	Input/output pin. Internal software programmable weak pull-up.
RB4	bit4	TTL	Input/output pin (with interrupt on change). Internal software programmable weak pull-up.
RB5	bit5	TTL	Input/output pin (with interrupt on change). Internal software programmable weak pull-up.
RB6	bit6	TTL/ST(2)	Input/output pin (with interrupt on change). Internal software programmable weak pull-up. Serial programming clock.
RB7	bit7	TTL/ST(2)	Input/output pin (with interrupt on change). Internal software programmable weak pull-up. Serial programming data.

Legend: TTL = TTL input, ST = Schmitt Trigger.

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.

2: This buffer is a Schmitt Trigger input when used in serial programming mode.

TABLE 3-4 SUMMARY OF REGISTERS ASSOCIATED WITH PORTB

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other resets
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	uuuu uuuu
86h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	1111 1111
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legend: x = unknown, u = unchanged. Shaded cells are not used by PORTB.

4.0 TIMER0 MODULE

The Timer0 module timer/counter has the following features:

- 8-bit timer/counter
- Readable and writable
- Internal or external clock select
- Edge select for external clock
- 8-bit software programmable prescaler
- Interrupt on overflow from FFh to 00h

Figure 4-1 is a simplified block diagram of the Timer0 module.

Additional information on timer modules is available in the PICmicro™ Mid-Range Reference Manual, (DS33023).

4.1 Timer0 Operation

Timer0 can operate as a timer or as a counter.

Timer mode is selected by clearing bit T0CS (OPTION_REG<5>). In timer mode, the Timer0 module will increment every instruction cycle (without prescaler). If the TMR0 register is written, the increment is inhibited for the following two instruction cycles. The user can work around this by writing an adjusted value to the TMR0 register.

Counter mode is selected by setting bit T0CS (OPTION_REG<5>). In counter mode, Timer0 will increment either on every rising or falling edge of pin RA4/T0CKI. The incrementing edge is determined by the Timer0 Source Edge Select bit T0SE (OPTION_REG<4>). Clearing bit T0SE selects the rising edge. Restrictions on the external clock input are discussed below.

When an external clock input is used for Timer0, it must meet certain requirements. The requirements ensure the external clock can be synchronized with the internal phase clock (TOSC). Also, there is a delay in the actual incrementing of Timer0 after synchronization.

Additional information on external clock requirements is available in the PICmicro™ Mid-Range Reference Manual, (DS33023).

4.2 Prescaler

An 8-bit counter is available as a prescaler for the Timer0 module, or as a postscaler for the Watchdog Timer, respectively (Figure 4-2). For simplicity, this counter is being referred to as “prescaler” throughout this data sheet. Note that there is only one prescaler available which is mutually exclusively shared between the Timer0 module and the Watchdog Timer. Thus, a prescaler assignment for the Timer0 module means that there is no prescaler for the Watchdog Timer, and vice-versa.

The prescaler is not readable or writable.

The PSA and PS2:PS0 bits (OPTION_REG<3:0>) determine the prescaler assignment and prescale ratio.

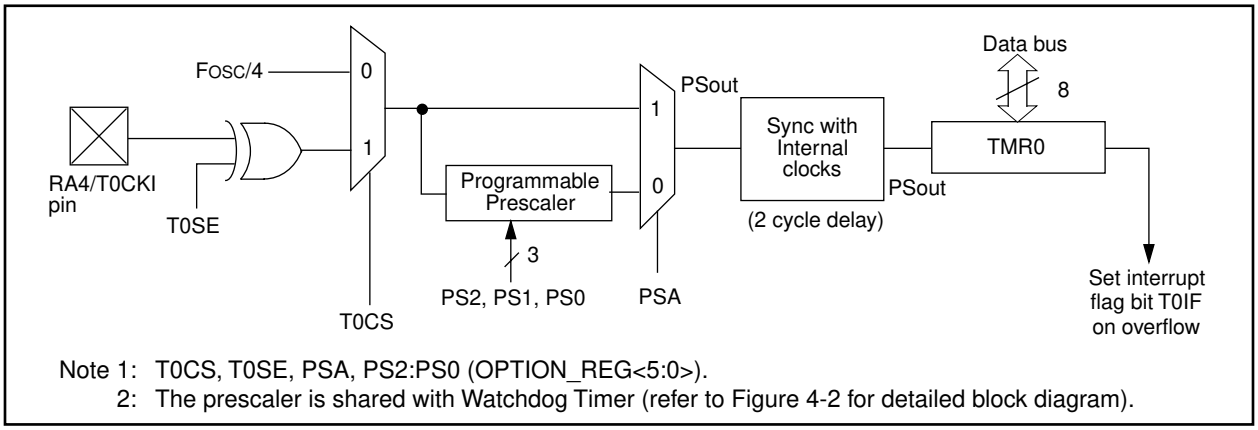
Clearing bit PSA will assign the prescaler to the Timer0 module. When the prescaler is assigned to the Timer0 module, prescale values of 1:2, 1:4, ..., 1:256 are selectable.

Setting bit PSA will assign the prescaler to the Watchdog Timer (WDT). When the prescaler is assigned to the WDT, prescale values of 1:1, 1:2, ..., 1:128 are selectable.

When assigned to the Timer0 module, all instructions writing to the TMR0 register (e.g. `CLRF 1`, `MOVWF 1`, `BSF 1,x`....etc.) will clear the prescaler. When assigned to WDT, a `CLRWDT` instruction will clear the prescaler along with the WDT.

Note:	Writing to TMR0 when the prescaler is assigned to Timer0 will clear the prescaler count, but will not change the prescaler assignment.

FIGURE 4-1: TIMER0 BLOCK DIAGRAM



Note 1: T0CS, T0SE, PSA, PS2:PS0 (OPTION_REG<5:0>).
2: The prescaler is shared with Watchdog Timer (refer to Figure 4-2 for detailed block diagram).

4.2.1 SWITCHING PRESCALER ASSIGNMENT

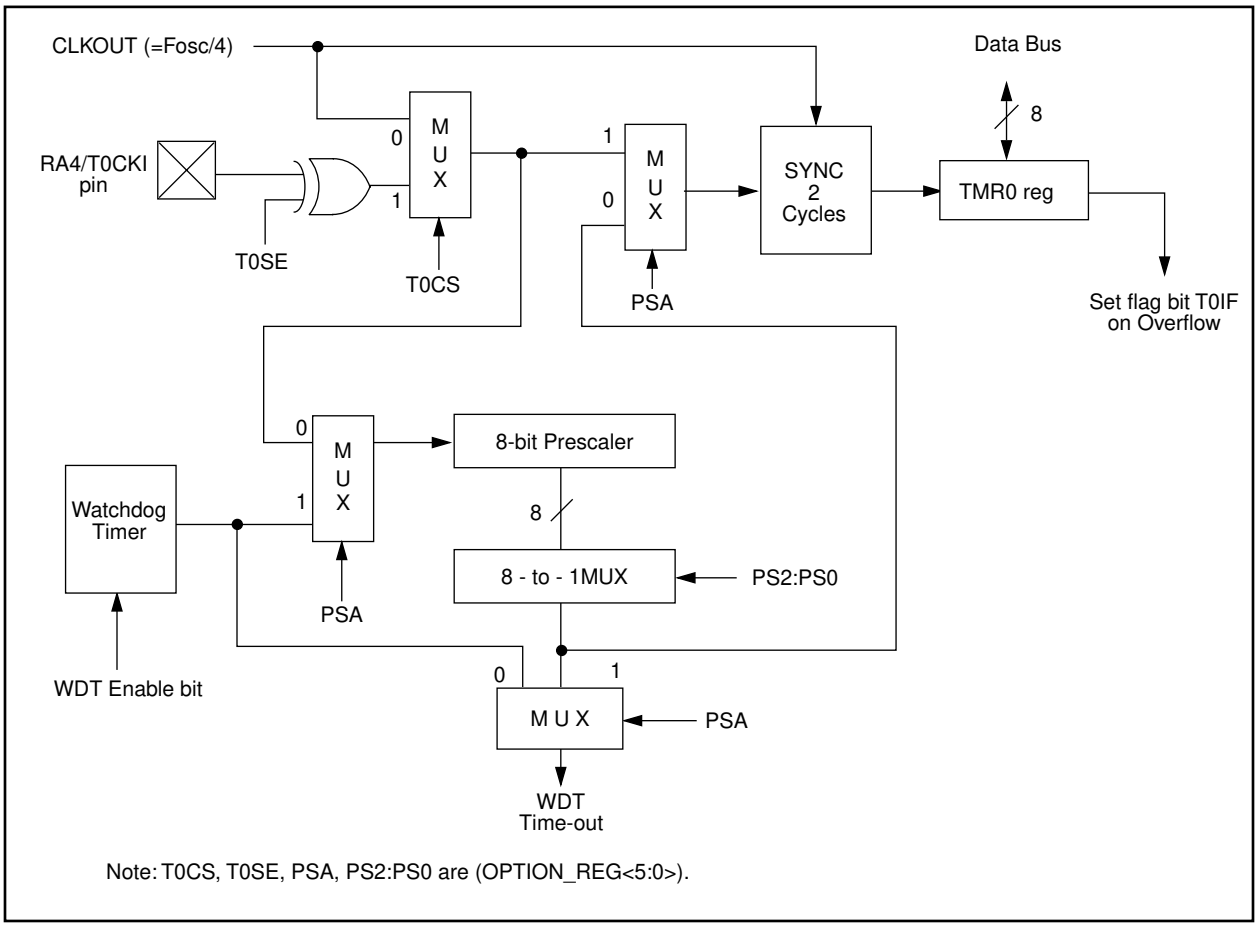
The prescaler assignment is fully under software control, i.e., it can be changed “on the fly” during program execution.

Note:	To avoid an unintended device RESET, a specific instruction sequence (shown in the PICmicro™ Mid-Range Reference Manual, DS3023) must be executed when changing the prescaler assignment from Timer0 to the WDT. This sequence must be followed even if the WDT is disabled.

4.3 Timer0 Interrupt

The TMR0 interrupt is generated when the TMR0 register overflows from FFh to 00h. This overflow sets bit T0IF (INTCON<2>). The interrupt can be masked by clearing bit T0IE (INTCON<5>). Bit T0IF must be cleared in software by the Timer0 module interrupt service routine before re-enabling this interrupt. The TMR0 interrupt cannot awaken the processor from SLEEP since the timer is shut off during SLEEP.

FIGURE 4-2: BLOCK DIAGRAM OF THE TIMER0/WDT PRESCALER



Note: T0CS, T0SE, PSA, PS2:PS0 are (OPTION_REG<5:0>).

TABLE 4-1 REGISTERS ASSOCIATED WITH TIMER0

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other resets
01h	TMR0	Timer0 module's register								xxxx xxxx	uuuu uuuu
0Bh,8Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
81h	OPTION_REG	$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	--11 1111

Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'. Shaded cells are not used by Timer0.

5.0 DATA EEPROM MEMORY

The EEPROM data memory is readable and writable during normal operation (full VDD range). This memory is not directly mapped in the register file space. Instead it is indirectly addressed through the Special Function Registers. There are four SFRs used to read and write this memory. These registers are:

- EECON1
- EECON2 (Not a physically implemented register)
- EEDATA
- EEADR

EEDATA holds the 8-bit data for read/write, and EEADR holds the address of the EEPROM location being accessed. PIC16F84A devices have 64 bytes of data EEPROM with an address range from 0h to 3Fh.

The EEPROM data memory allows byte read and write. A byte write automatically erases the location and writes the new data (erase before write). The EEPROM data memory is rated for high erase/write cycles. The write time is controlled by an on-chip timer. The write-time will vary with voltage and temperature as well as from chip to chip. Please refer to AC specifications for exact limits.

When the device is code protected, the CPU may continue to read and write the data EEPROM memory. The device programmer can no longer access this memory.

Additional information on the Data EEPROM is available in the PICmicro™ Mid-Range Reference Manual, (DS33023).

FIGURE 5-1: EECON1 REGISTER (ADDRESS 88h)

U	U	U	R/W-0	R/W-x	R/W-0	R/S-0	R/S-x
—	—	—	EEIF	WRERR	WREN	WR	RD
bit7							bit0

R = Readable bit
W = Writable bit
S = Settable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

bit 7:5 **Unimplemented:** Read as '0'

bit 4 **EEIF**: EEPROM Write Operation Interrupt Flag bit
1 = The write operation completed (must be cleared in software)
0 = The write operation is not complete or has not been started

bit 3 **WRERR**: EEPROM Error Flag bit
1 = A write operation is prematurely terminated (any $\overline{MCLR}$ reset or any WDT reset during normal operation)
0 = The write operation completed

bit 2 **WREN**: EEPROM Write Enable bit
1 = Allows write cycles
0 = Inhibits write to the data EEPROM

bit 1 **WR**: Write Control bit
1 = initiates a write cycle. (The bit is cleared by hardware once write is complete. The WR bit can only be set (not cleared) in software.
0 = Write cycle to the data EEPROM is complete

bit 0 **RD**: Read Control bit
1 = Initiates an EEPROM read (read takes one cycle. RD is cleared in hardware. The RD bit can only be set (not cleared) in software).
0 = Does not initiate an EEPROM read

5.1 Reading the EEPROM Data Memory

To read a data memory location, the user must write the address to the EEADR register and then set control bit RD (EECON1<0>). The data is available, in the very next cycle, in the EEDATA register; therefore it can be read in the next instruction. EEDATA will hold this value until another read or until it is written to by the user (during a write operation).

EXAMPLE 5-1: DATA EEPROM READ

```
BCF     STATUS, RP0  ; Bank 0
MOVLW   CONFIG_ADDR  ;
MOVWF   EEADR        ; Address to read
BSF     STATUS, RP0  ; Bank 1
BSF     EECON1, RD   ; EE Read
BCF     STATUS, RP0  ; Bank 0
MOVF    EEDATA, W    ; W = EEDATA
```

5.2 Writing to the EEPROM Data Memory

To write an EEPROM data location, the user must first write the address to the EEADR register and the data to the EEDATA register. Then the user must follow a specific sequence to initiate the write for each byte.

EXAMPLE 5-1: DATA EEPROM WRITE

```
        BSF     STATUS, RP0  ; Bank 1
        BCF     INTCON, GIE  ; Disable INTs.
        BSF     EECON1, WREN ; Enable Write
        MOVLW   55h          ;
Required
Sequence
        MOVWF   EECON2       ; Write 55h
        MOVLW   AAh          ;
        MOVWF   EECON2       ; Write AAh
        BSF     EECON1,WR    ; Set WR bit
                             ;   begin write
        BSF     INTCON, GIE  ; Enable INTs.
```

The write will not initiate if the above sequence is not exactly followed (write 55h to EECON2, write AAh to EECON2, then set WR bit) for each byte. We strongly recommend that interrupts be disabled during this code segment.

Additionally, the WREN bit in EECON1 must be set to enable write. This mechanism prevents accidental writes to data EEPROM due to errant (unexpected) code execution (i.e., lost programs). The user should keep the WREN bit clear at all times, except when updating EEPROM. The WREN bit is not cleared by hardware

After a write sequence has been initiated, clearing the WREN bit will not affect this write cycle. The WR bit will be inhibited from being set unless the WREN bit is set.

At the completion of the write cycle, the WR bit is cleared in hardware and the EE Write Complete Interrupt Flag bit (EEIF) is set. The user can either enable this interrupt or poll this bit. EEIF must be cleared by software.

5.3 Write Verify

Depending on the application, good programming practice may dictate that the value written to the Data EEPROM should be verified (Example 5-1) to the desired value to be written. This should be used in applications where an EEPROM bit will be stressed near the specification limit. The Total Endurance disk will help determine your comfort level.

Generally the EEPROM write failure will be a bit which was written as a '0', but reads back as a '1' (due to leakage off the bit).

EXAMPLE 5-1: WRITE VERIFY

```
    BCF   STATUS, RP0 ; Bank 0
    :                 ; Any code can go here
    :                 ;
    MOVF  EEDATA, W   ; Must be in Bank 0
    BSF   STATUS, RP0 ; Bank 1
READ
    BSF   EECON1, RD  ; YES, Read the
                      ;   value written
    BCF   STATUS, RP0 ; Bank 0
;
; Is the value written (in W reg) and
;   read (in EEDATA) the same?
;
    SUBWF EEDATA, W   ;
    BTFSS STATUS, Z   ; Is difference 0?
    GOTO  WRITE_ERR   ; NO, Write error
    :                 ; YES, Good write
    :                 ; Continue program
```

TABLE 5-1 REGISTERS/BITS ASSOCIATED WITH DATA EEPROM

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other resets
08h	EEDATA	EEPROM data register								xxxx xxxx	uuuu uuuu
09h	EEADR	EEPROM address register								xxxx xxxx	uuuu uuuu
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	---0 q000
89h	EECON2	EEPROM control register 2								---- ----	---- ----

Legend: x = unknown, u = unchanged, - = unimplemented read as '0', q = value depends upon condition. Shaded cells are not used by data EEPROM.

6.0 SPECIAL FEATURES OF THE CPU

What sets a microcontroller apart from other processors are special circuits to deal with the needs of real time applications. The PIC16F84A has a host of such features intended to maximize system reliability, minimize cost through elimination of external components, provide power saving operating modes and offer code protection. These features are:

- OSC Selection
- Reset
  - Power-on Reset (POR)
  - Power-up Timer (PWRT)
  - Oscillator Start-up Timer (OST)
- Interrupts
- Watchdog Timer (WDT)
- SLEEP
- Code protection
- ID locations
- In-circuit serial programming

The PIC16F84A has a Watchdog Timer which can be shut off only through configuration bits. It runs off its own RC oscillator for added reliability. There are two timers that offer necessary delays on power-up. One is the Oscillator Start-up Timer (OST), intended to keep the chip in reset until the crystal oscillator is stable. The other is the Power-up Timer (PWRT), which provides a fixed delay of 72 ms (nominal) on power-up only. This design keeps the device in reset while the power supply stabilizes. With these two timers on-chip, most applications need no external reset circuitry.

SLEEP mode offers a very low current power-down mode. The user can wake-up from SLEEP through external reset, Watchdog Timer time-out or through an interrupt. Several oscillator options are provided to allow the part to fit the application. The RC oscillator option saves system cost while the LP crystal option saves power. A set of configuration bits are used to select the various options.

Additional information on special features is available in the PICmicro™ Mid-Range Reference Manual, (DS33023).

6.1 Configuration Bits

The configuration bits can be programmed (read as '0') or left unprogrammed (read as '1') to select various device configurations. These bits are mapped in program memory location 2007h.

Address 2007h is beyond the user program memory space and it belongs to the special test/configuration memory space (2000h - 3FFFh). This space can only be accessed during programming.

FIGURE 6-1: CONFIGURATION WORD - PIC16F84A

R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u
CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	$\overline{\text{PWRTE}}$	WDTE	FOSC1	FOSC0
bit13													bit0

R = Readable bit
P = Programmable bit
- n = Value at POR reset
u = unchanged

bit 13:4 **CP**: Code Protection bit
1 = Code protection off
0 = All memory is code protected

bit 3 **$\overline{\text{PWRTE}}$**: Power-up Timer Enable bit
1 = Power-up timer is disabled
0 = Power-up timer is enabled

bit 2 **WDTE**: Watchdog Timer Enable bit
1 = WDT enabled
0 = WDT disabled

bit 1:0 **FOSC1:FOSC0**: Oscillator Selection bits
11 = RC oscillator
10 = HS oscillator
01 = XT oscillator
00 = LP oscillator

6.2 Oscillator Configurations

6.2.1 OSCILLATOR TYPES

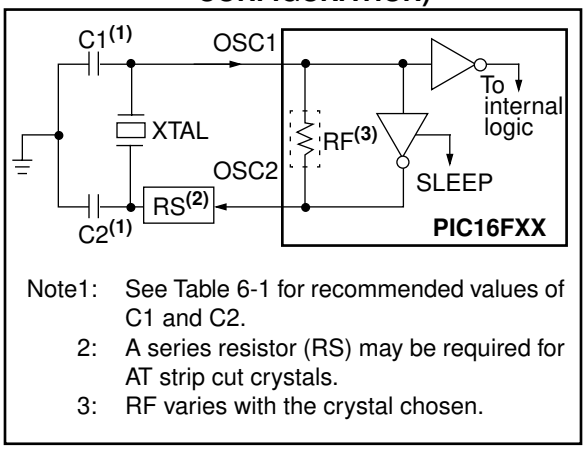
The PIC16F84A can be operated in four different oscillator modes. The user can program two configuration bits (FOSC1 and FOSC0) to select one of these four modes:

- LP Low Power Crystal
- XT Crystal/Resonator
- HS High Speed Crystal/Resonator
- RC Resistor/Capacitor

6.2.2 CRYSTAL OSCILLATOR / CERAMIC RESONATORS

In XT, LP or HS modes a crystal or ceramic resonator is connected to the OSC1/CLKIN and OSC2/CLKOUT pins to establish oscillation (Figure 6-2).

FIGURE 6-2: CRYSTAL/CERAMIC RESONATOR OPERATION (HS, XT OR LP OSC CONFIGURATION)



Note1: See Table 6-1 for recommended values of C1 and C2.
2: A series resistor (RS) may be required for AT strip cut crystals.
3: RF varies with the crystal chosen.

The PIC16F84A oscillator design requires the use of a parallel cut crystal. Use of a series cut crystal may give a frequency out of the crystal manufacturers specifications. When in XT, LP or HS modes, the device can have an external clock source to drive the OSC1/CLKIN pin (Figure 6-3).

FIGURE 6-3: EXTERNAL CLOCK INPUT OPERATION (HS, XT OR LP OSC CONFIGURATION)

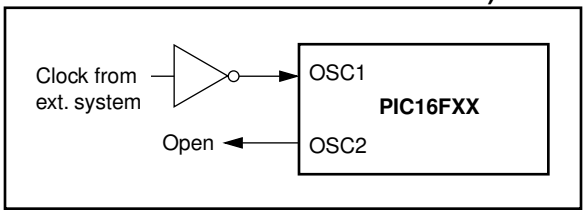


TABLE 6-1 CAPACITOR SELECTION FOR CERAMIC RESONATORS

Ranges Tested:			
Mode	**Freq**	**OSC1/C1**	**OSC2/C2**
XT	455 kHz	47 - 100 pF	47 - 100 pF
	2.0 MHz	15 - 33 pF	15 - 33 pF
	4.0 MHz	15 - 33 pF	15 - 33 pF
HS	8.0 MHz	15 - 33 pF	15 - 33 pF
	10.0 MHz	15 - 33 pF	15 - 33 pF

Note: Recommended values of C1 and C2 are identical to the ranges tested table.
Higher capacitance increases the stability of the oscillator but also increases the start-up time. These values are for design guidance only. Since each resonator has its own characteristics, the user should consult the resonator manufacturer for the appropriate values of external components.

Resonators Tested:		
455 kHz	Panasonic EFO-A455K04B	± 0.3%
2.0 MHz	Murata Erie CSA2.00MG	± 0.5%
4.0 MHz	Murata Erie CSA4.00MG	± 0.5%
8.0 MHz	Murata Erie CSA8.00MT	± 0.5%
10.0 MHz	Murata Erie CSA10.00MTZ	± 0.5%

None of the resonators had built-in capacitors.

TABLE 6-2 CAPACITOR SELECTION FOR CRYSTAL OSCILLATOR

Mode	Freq	OSC1/C1	OSC2/C2
LP	32 kHz	68 - 100 pF	68 - 100 pF
	200 kHz	15 - 33 pF	15 - 33 pF
XT	100 kHz	100 - 150 pF	100 - 150 pF
	2 MHz	15 - 33 pF	15 - 33 pF
	4 MHz	15 - 33 pF	15 - 33 pF
HS	4 MHz	15 - 33 pF	15 - 33 pF
	10 MHz	15 - 33 pF	15 - 33 pF

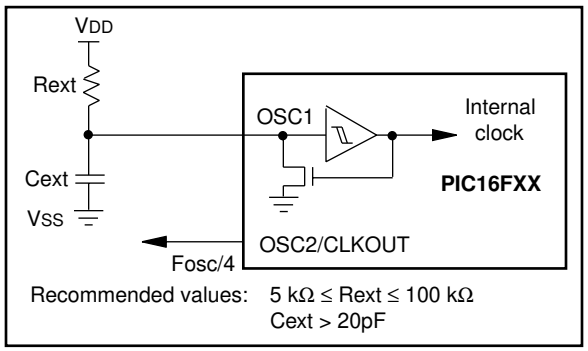
Note: Higher capacitance increases the stability of oscillator but also increases the start-up time. These values are for design guidance only. Rs may be required in HS mode as well as XT mode to avoid overdriving crystals with low drive level specification. Since each crystal has its own characteristics, the user should consult the crystal manufacturer for appropriate values of external components.
For V_{DD} > 4.5V, C1 = C2 ≈ 30 pF is recommended.

Crystals Tested:		
32.768 kHz	Epson C-001R32.768K-A	± 20 PPM
100 kHz	Epson C-2 100.00 KC-P	± 20 PPM
200 kHz	STD XTL 200.000 KHz	± 20 PPM
1.0 MHz	ECS ECS-10-13-2	± 50 PPM
2.0 MHz	ECS ECS-20-S-2	± 50 PPM
4.0 MHz	ECS ECS-40-S-4	± 50 PPM
10.0 MHz	ECS ECS-100-S-4	± 50 PPM

6.2.3 RC OSCILLATOR

For timing insensitive applications the RC device option offers additional cost savings. The RC oscillator frequency is a function of the supply voltage, the resistor (Rext) values, capacitor (Cext) values, and the operating temperature. In addition to this, the oscillator frequency will vary from unit to unit due to normal process parameter variation. Furthermore, the difference in lead frame capacitance between package types also affects the oscillation frequency, especially for low Cext values. The user needs to take into account variation due to tolerance of the external R and C components. Figure 6-4 shows how an R/C combination is connected to the PIC16F84A.

FIGURE 6-4: RC OSCILLATOR MODE



6.3 Reset

The PIC16F84A differentiates between various kinds of reset:

- Power-on Reset (POR)
- $\overline{MCLR}$ reset during normal operation
- $\overline{MCLR}$ reset during SLEEP
- WDT Reset (during normal operation)
- WDT Wake-up (during SLEEP)

Figure 6-5 shows a simplified block diagram of the on-chip reset circuit. The $\overline{MCLR}$ reset path has a noise filter to ignore small pulses. The electrical specifications state the pulse width requirements for the $\overline{MCLR}$ pin.

Some registers are not affected in any reset condition; their status is unknown on a POR reset and unchanged in any other reset. Most other registers are reset to a "reset state" on POR, $\overline{MCLR}$ or WDT reset during normal operation and on $\overline{MCLR}$ reset during SLEEP. They are not affected by a WDT reset during SLEEP, since this reset is viewed as the resumption of normal operation.

Table 6-3 gives a description of reset conditions for the program counter (PC) and the STATUS register. Table 6-4 gives a full description of reset states for all registers.

The $\overline{TO}$ and $\overline{PD}$ bits are set or cleared differently in different reset situations (Section 6.7). These bits are used in software to determine the nature of the reset.

FIGURE 6-5: SIMPLIFIED BLOCK DIAGRAM OF ON-CHIP RESET CIRCUIT

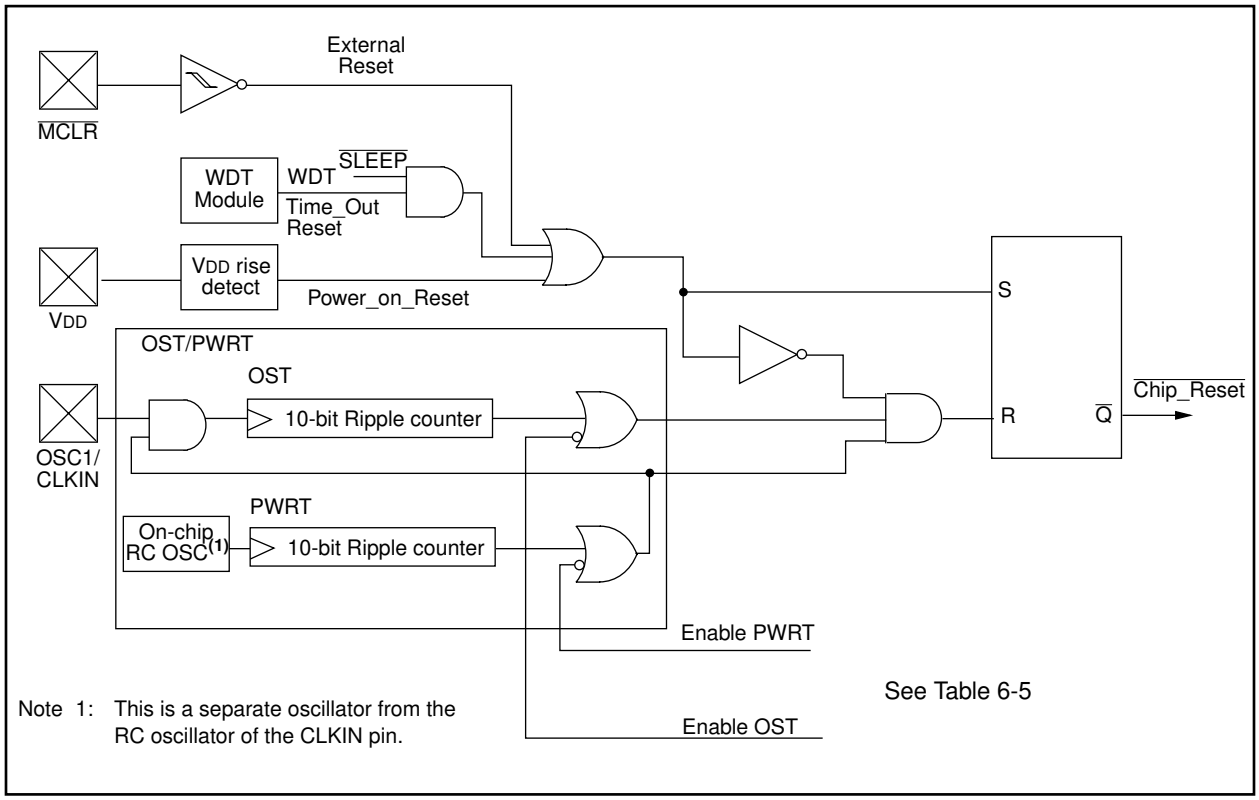


TABLE 6-3 RESET CONDITION FOR PROGRAM COUNTER AND THE STATUS REGISTER

Condition	Program Counter	STATUS Register
Power-on Reset	000h	0001 1xxx
$\overline{MCLR}$ Reset during normal operation	000h	000u uuuu
$\overline{MCLR}$ Reset during SLEEP	000h	0001 0uuu
WDT Reset (during normal operation)	000h	0000 1uuu
WDT Wake-up	PC + 1	uuu0 0uuu
Interrupt wake-up from SLEEP	PC + 1 (1)	uuu1 0uuu

Legend: u = unchanged, x = unknown.
Note 1: When the wake-up is due to an interrupt and the GIE bit is set, the PC is loaded with the interrupt vector (0004h).

TABLE 6-4 RESET CONDITIONS FOR ALL REGISTERS

Register	Address	Power-on Reset	$\overline{MCLR}$ Reset during: – normal operation – SLEEP WDT Reset during normal operation	Wake-up from SLEEP: – through interrupt – through WDT Time-out
W	—	xxxx xxxx	uuuu uuuu	uuuu uuuu
INDF	00h	---- ----	---- ----	---- ----
TMR0	01h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCL	02h	0000h	0000h	PC + 1 (2)
STATUS	03h	0001 1xxx	000q quuu (3)	uuuq quuu (3)
FSR	04h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTA (4)	05h	---x xxxx	---u uuuu	---u uuuu
PORTB (5)	06h	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEDATA	08h	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEADR	09h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCLATH	0Ah	---0 0000	---0 0000	---u uuuu
INTCON	0Bh	0000 000x	0000 000u	uuuu uuuu (1)
INDF	80h	---- ----	---- ----	---- ----
OPTION_REG	81h	1111 1111	1111 1111	uuuu uuuu
PCL	82h	0000h	0000h	PC + 1
STATUS	83h	0001 1xxx	000q quuu (3)	uuuq quuu (3)
FSR	84h	xxxx xxxx	uuuu uuuu	uuuu uuuu
TRISA	85h	---1 1111	---1 1111	---u uuuu
TRISB	86h	1111 1111	1111 1111	uuuu uuuu
EECON1	88h	---0 x000	---0 q000	---0 uuuu
EECON2	89h	---- ----	---- ----	---- ----
PCLATH	8Ah	---0 0000	---0 0000	---u uuuu
INTCON	8Bh	0000 000x	0000 000u	uuuu uuuu (1)

Legend: u = unchanged, x = unknown, - = unimplemented bit read as '0', q = value depends on condition.
Note 1: One or more bits in INTCON will be affected (to cause wake-up).
2: When the wake-up is due to an interrupt and the GIE bit is set, the PC is loaded with the interrupt vector (0004h).
3: Table 6-3 lists the reset value for each specific condition.
4: On any device reset, these pins are configured as inputs.
5: This is the value that will be in the port output latch.

6.4 Power-on Reset (POR)

A Power-on Reset pulse is generated on-chip when VDD rise is detected (in the range of 1.2V - 1.7V). To take advantage of the POR, just tie the $\overline{MCLR}$ pin directly (or through a resistor) to VDD. This will eliminate external RC components usually needed to create Power-on Reset. A minimum rise time for VDD must be met for this to operate properly. See Electrical Specifications for details.

When the device starts normal operation (exits the reset condition), device operating parameters (voltage, frequency, temperature, ...) must be meet to ensure operation. If these conditions are not met, the device must be held in reset until the operating conditions are met.

For additional information, refer to Application Note AN607, "*Power-up Trouble Shooting*."

The POR circuit does not produce an internal reset when VDD declines.

6.5 Power-up Timer (PWRT)

The Power-up Timer (PWRT) provides a fixed 72 ms nominal time-out (TPWRT) from POR (Figure 6-7, Figure 6-8, Figure 6-9 and Figure 6-10). The Power-up Timer operates on an internal RC oscillator. The chip is kept in reset as long as the PWRT is active. The PWRT delay allows the VDD to rise to an acceptable level (Possible exception shown in Figure 6-10).

A configuration bit, $\overline{PWRTE}$, can enable/disable the PWRT. See Figure 6-1 for the operation of the $\overline{PWRTE}$ bit for a particular device.

The power-up time delay TPWRT will vary from chip to chip due to VDD, temperature, and process variation. See DC parameters for details.

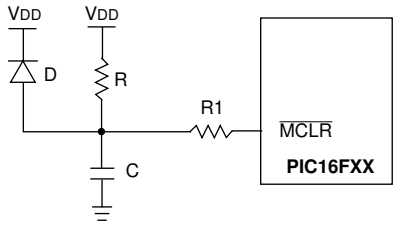
6.6 Oscillator Start-up Timer (OST)

The Oscillator Start-up Timer (OST) provides a 1024 oscillator cycle delay (from OSC1 input) after the PWRT delay ends (Figure 6-7, Figure 6-8, Figure 6-9 and Figure 6-10). This ensures the crystal oscillator or resonator has started and stabilized.

The OST time-out (TOST) is invoked only for XT, LP and HS modes and only on Power-on Reset or wake-up from SLEEP.

When VDD rises very slowly, it is possible that the TPWRT time-out and TOST time-out will expire before VDD has reached its final value. In this case (Figure 6-10), an external power-on reset circuit may be necessary (Figure 6-6).

FIGURE 6-6: EXTERNAL POWER-ON RESET CIRCUIT (FOR SLOW VDD POWER-UP)



Note 1: External Power-on Reset circuit is required only if VDD power-up rate is too slow. The diode D helps discharge the capacitor quickly when VDD powers down.

2: R < 40 kΩ is recommended to make sure that voltage drop across R does not exceed 0.2V (max leakage current spec on $\overline{MCLR}$ pin is 5 μA). A larger voltage drop will degrade VIH level on the $\overline{MCLR}$ pin.

3: R1 = 100Ω to 1 kΩ will limit any current flowing into $\overline{MCLR}$ from external capacitor C in the event of an $\overline{MCLR}$ pin breakdown due to ESD or EOS.

FIGURE 6-7: TIME-OUT SEQUENCE ON POWER-UP ($\overline{MCLR}$ NOT TIED TO V_{DD}): CASE 1

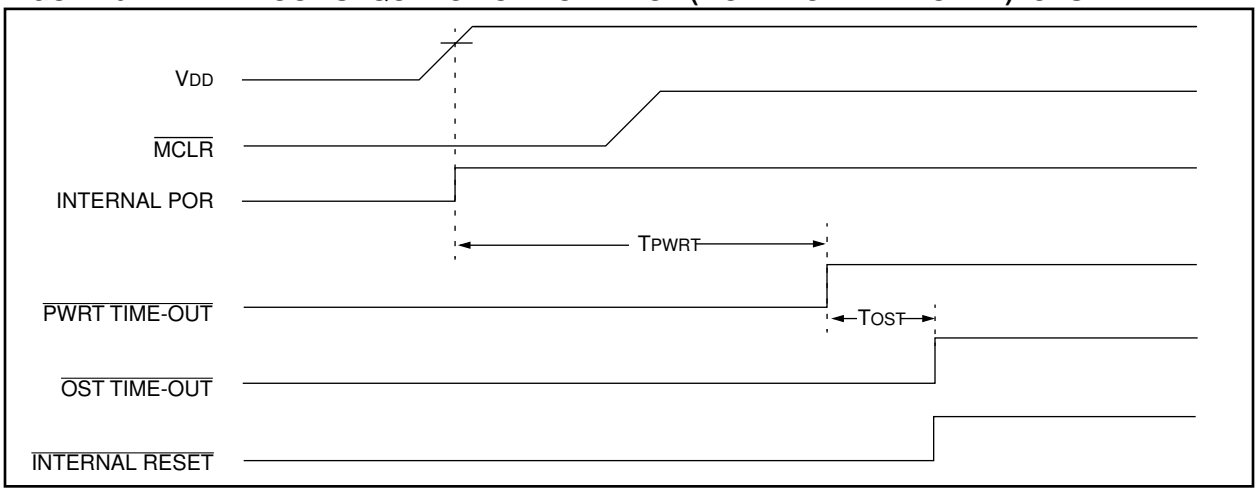


FIGURE 6-8: TIME-OUT SEQUENCE ON POWER-UP ($\overline{MCLR}$ NOT TIED TO V_{DD}): CASE 2

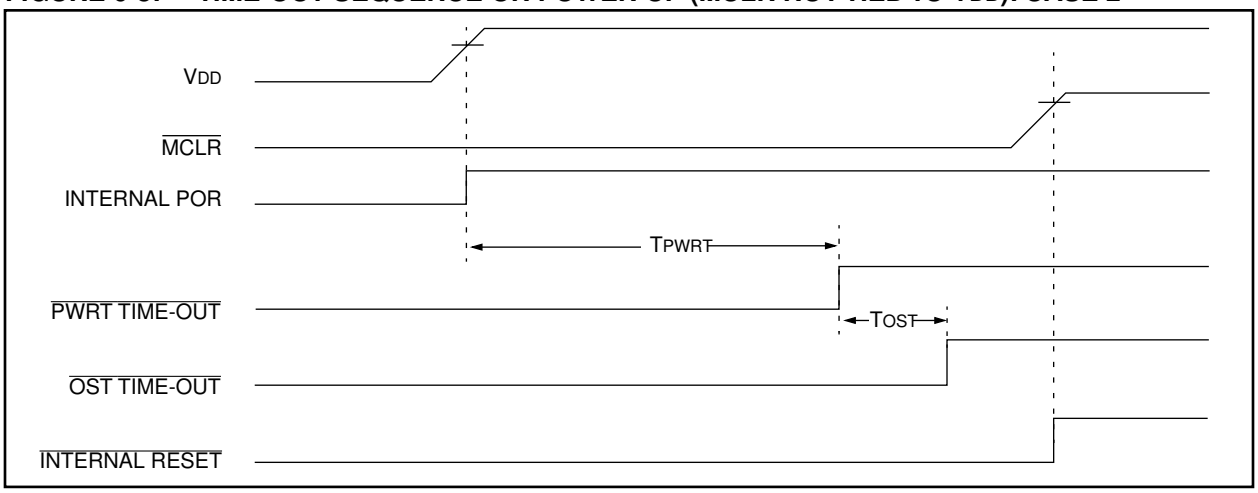


FIGURE 6-9: TIME-OUT SEQUENCE ON POWER-UP ($\overline{MCLR}$ TIED TO V_{DD}): FAST V_{DD} RISE TIME

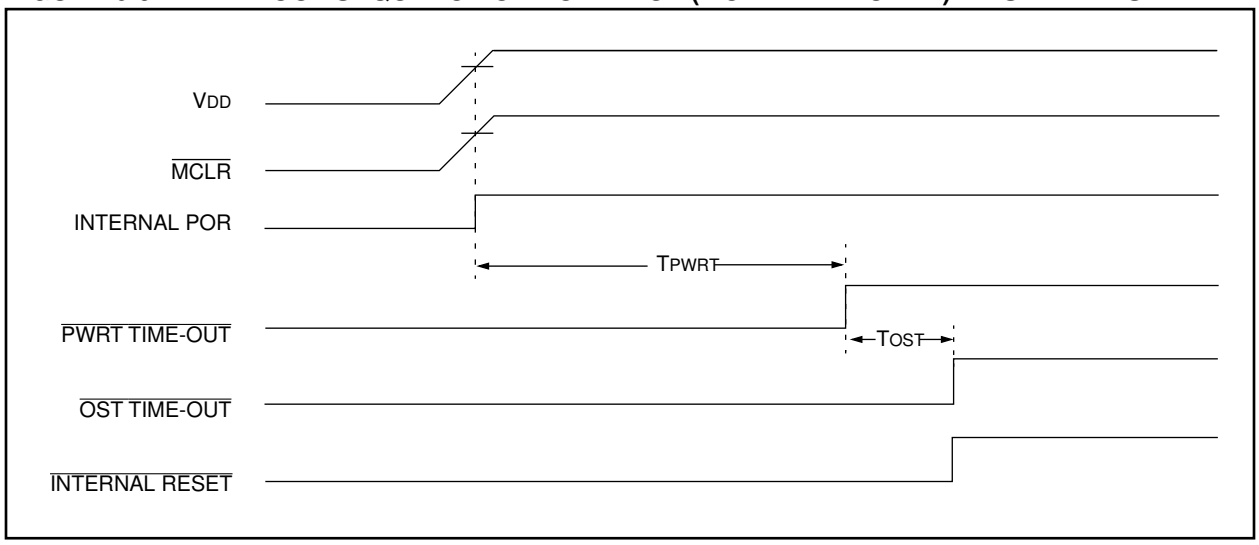
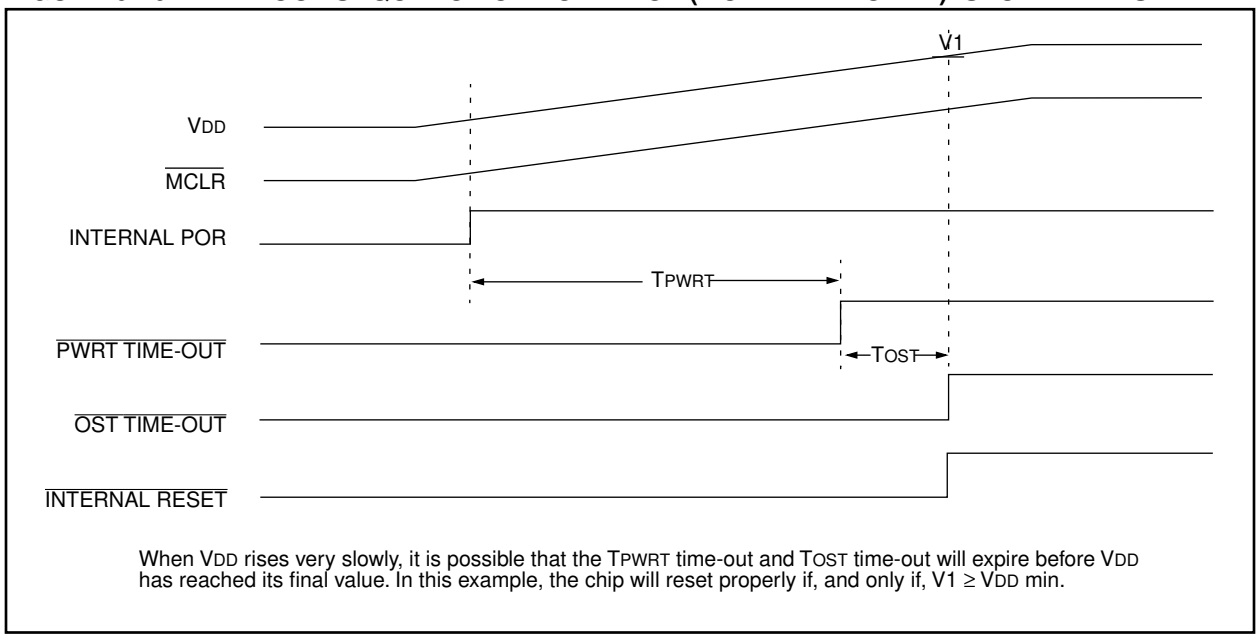


FIGURE 6-10: TIME-OUT SEQUENCE ON POWER-UP ($\overline{MCLR}$ TIED TO V_{DD}): SLOW V_{DD} RISE TIME



When V_{DD} rises very slowly, it is possible that the T_{PWRT} time-out and T_{OST} time-out will expire before V_{DD} has reached its final value. In this example, the chip will reset properly if, and only if, V1 ≥ V_{DD} min.

6.7 Time-out Sequence and Power-down Status Bits ($\overline{TO}$/$\overline{PD}$)

On power-up (Figure 6-7, Figure 6-8, Figure 6-9 and Figure 6-10) the time-out sequence is as follows: First PWRT time-out is invoked after a POR has expired. Then the OST is activated. The total time-out will vary based on oscillator configuration and PWRTE configuration bit status. For example, in RC mode with the PWRT disabled, there will be no time-out at all.

TABLE 6-5 TIME-OUT IN VARIOUS SITUATIONS

Oscillator Configuration	Power-up		Wake-up from SLEEP
	PWRT Enabled	PWRT Disabled	
XT, HS, LP	72 ms + 1024TOSC	1024TOSC	1024TOSC
RC	72 ms	—	—

Since the time-outs occur from the POR reset pulse, if $\overline{MCLR}$ is kept low long enough, the time-outs will expire. Then bringing $\overline{MCLR}$ high, execution will begin immediately (Figure 6-7). This is useful for testing purposes or to synchronize more than one PIC16F84A device when operating in parallel.

Table 6-6 shows the significance of the $\overline{TO}$ and $\overline{PD}$ bits. Table 6-3 lists the reset conditions for some special registers, while Table 6-4 lists the reset conditions for all the registers.

TABLE 6-6 STATUS BITS AND THEIR SIGNIFICANCE

$\overline{TO}$	$\overline{PD}$	Condition
1	1	Power-on Reset
0	x	Illegal, $\overline{TO}$ is set on $\overline{POR}$
x	0	Illegal, $\overline{PD}$ is set on $\overline{POR}$
0	1	WDT Reset (during normal operation)
0	0	WDT Wake-up
1	1	$\overline{MCLR}$ Reset during normal operation
1	0	$\overline{MCLR}$ Reset during SLEEP or interrupt wake-up from SLEEP

6.8 Interrupts

The PIC16F84A has 4 sources of interrupt:

- External interrupt RB0/INT pin
- TMR0 overflow interrupt
- PORTB change interrupts (pins RB7:RB4)
- Data EEPROM write complete interrupt

The interrupt control register (INTCON) records individual interrupt requests in flag bits. It also contains the individual and global interrupt enable bits.

The global interrupt enable bit, GIE (INTCON<7>) enables (if set) all un-masked interrupts or disables (if cleared) all interrupts. Individual interrupts can be disabled through their corresponding enable bits in INTCON register. Bit GIE is cleared on reset.

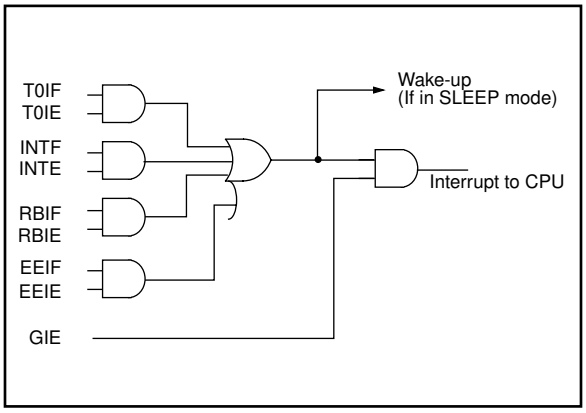
The "return from interrupt" instruction, RETFIE, exits interrupt routine as well as sets the GIE bit, which re-enable interrupts.

The RB0/INT pin interrupt, the RB port change interrupt and the TMR0 overflow interrupt flags are contained in the INTCON register.

When an interrupt is responded to; the GIE bit is cleared to disable any further interrupt, the return address is pushed onto the stack and the PC is loaded with 0004h. For external interrupt events, such as the RB0/INT pin or PORTB change interrupt, the interrupt latency will be three to four instruction cycles. The exact latency depends when the interrupt event occurs. The latency is the same for both one and two cycle instructions. Once in the interrupt service routine the source(s) of the interrupt can be determined by polling the interrupt flag bits. The interrupt flag bit(s) must be cleared in software before re-enabling interrupts to avoid infinite interrupt requests.

Note 1: Individual interrupt flag bits are set regardless of the status of their corresponding mask bit or the GIE bit.

FIGURE 6-11: INTERRUPT LOGIC



6.8.1 INT INTERRUPT

External interrupt on RB0/INT pin is edge triggered: either rising if INTEDG bit (OPTION_REG<6>) is set, or falling, if INTEDG bit is clear. When a valid edge appears on the RB0/INT pin, the INTF bit (INTCON<1>) is set. This interrupt can be disabled by clearing control bit INTE (INTCON<4>). Flag bit INTF must be cleared in software via the interrupt service routine before re-enabling this interrupt. The INT interrupt can wake the processor from SLEEP (Section 6.11) only if the INTE bit was set prior to going into SLEEP. The status of the GIE bit decides whether the processor branches to the interrupt vector following wake-up.

6.8.2 TMR0 INTERRUPT

An overflow (FFh → 00h) in TMR0 will set flag bit T0IF (INTCON<2>). The interrupt can be enabled/disabled by setting/clearing enable bit T0IE (INTCON<5>) (Section 4.0).

6.8.3 PORB INTERRUPT

An input change on PORTB<7:4> sets flag bit RBIF (INTCON<0>). The interrupt can be enabled/disabled by setting/clearing enable bit RBIE (INTCON<3>) (Section 3.2).

Note 1: For a change on the I/O pin to be recognized, the pulse width must be at least TCY wide.

6.8.4 DATA EEPROM INTERRUPT

At the completion of a data EEPROM write cycle, flag bit EEIF (EECON1<4>) will be set. The interrupt can be enabled/disabled by setting/clearing enable bit EEIE (INTCON<6>) (Section 5.0).

6.9 Context Saving During Interrupts

During an interrupt, only the return PC value is saved on the stack. Typically, users wish to save key register values during an interrupt (e.g., W register and STATUS register). This is implemented in software.

Example 6-1 stores and restores the STATUS and W register's values. The User defined registers, W_TEMP and STATUS_TEMP are the temporary storage locations for the W and STATUS registers values.

Example 6-1 does the following:

a) Stores the W register.
b) Stores the STATUS register in STATUS_TEMP.
c) Executes the Interrupt Service Routine code.
d) Restores the STATUS (and bank select bit) register.
e) Restores the W register.

EXAMPLE 6-1: SAVING STATUS AND W REGISTERS IN RAM

```
PUSH    MOVWF   W_TEMP          ; Copy W to TEMP register,
        SWAPF   STATUS, W       ; Swap status to be saved into W
        MOVWF   STATUS_TEMP     ; Save status to STATUS_TEMP register
ISR     :                       :
        :                       ; Interrupt Service Routine
        :                       ;  should configure Bank as required
        :                       ;
POP     SWAPF   STATUS_TEMP, W  ; Swap nibbles in STATUS_TEMP register
                                ; and place result into W
        MOVWF   STATUS          ; Move W into STATUS register
                                ;   (sets bank to original state)
        SWAPF   W_TEMP, F       ; Swap nibbles in W_TEMP and place result in W_TEMP
        SWAPF   W_TEMP, W       ; Swap nibbles in W_TEMP and place result into W
```

6.10 Watchdog Timer (WDT)

The Watchdog Timer is a free running on-chip RC oscillator which does not require any external components. This RC oscillator is separate from the RC oscillator of the OSC1/CLKIN pin. That means that the WDT will run even if the clock on the OSC1/CLKIN and OSC2/CLKOUT pins of the device has been stopped, for example, by execution of a SLEEP instruction. During normal operation a WDT time-out generates a device RESET. If the device is in SLEEP mode, a WDT Wake-up causes the device to wake-up and continue with normal operation. The WDT can be permanently disabled by programming configuration bit WDTE as a '0' (Section 6.1).

6.10.1 WDT PERIOD

The WDT has a nominal time-out period of 18 ms, (with no prescaler). The time-out periods vary with temperature, VDD and process variations from part to part (see DC specs). If longer time-out periods are desired, a prescaler with a division ratio of up to 1:128 can be assigned to the WDT under software control by writing to the OPTION_REG register. Thus, time-out periods up to 2.3 seconds can be realized.

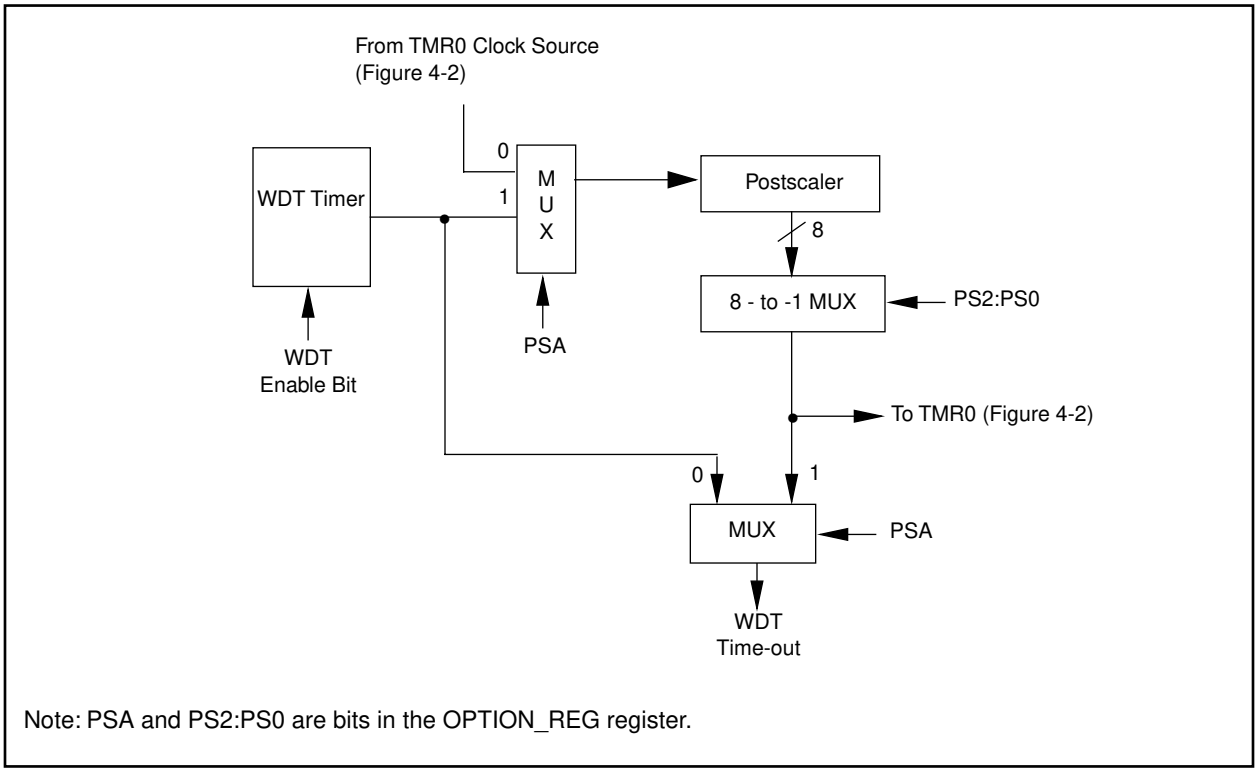
The CLRWDT and SLEEP instructions clear the WDT and the postscaler (if assigned to the WDT) and prevent it from timing out and generating a device RESET condition.

The $\overline{TO}$ bit in the STATUS register will be cleared upon a WDT time-out.

6.10.2 WDT PROGRAMMING CONSIDERATIONS

It should also be taken into account that under worst case conditions (VDD = Min., Temperature = Max., max. WDT prescaler) it may take several seconds before a WDT time-out occurs.

FIGURE 6-12: WATCHDOG TIMER BLOCK DIAGRAM



Note: PSA and PS2:PS0 are bits in the OPTION_REG register.

TABLE 6-7 SUMMARY OF REGISTERS ASSOCIATED WITH THE WATCHDOG TIMER

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other resets
2007h	Config. bits	(2)	(2)	(2)	(2)	PWRTE(1)	WDTE	FOSC1	FOSC0	(2)	
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legend: x = unknown. Shaded cells are not used by the WDT.
Note 1: See Figure 6-1 for operation of the PWRTE bit.
2: See Figure 6-1 and Section 6.12 for operation of the Code and Data protection bits.

6.11 Power-down Mode (SLEEP)

A device may be powered down (SLEEP) and later powered up (Wake-up from SLEEP).

6.11.1 SLEEP

The Power-down mode is entered by executing the SLEEP instruction.

If enabled, the Watchdog Timer is cleared (but keeps running), the $\overline{PD}$ bit (STATUS<3>) is cleared, the $\overline{TO}$ bit (STATUS<4>) is set, and the oscillator driver is turned off. The I/O ports maintain the status they had before the SLEEP instruction was executed (driving high, low, or hi-impedance).

For the lowest current consumption in SLEEP mode, place all I/O pins at either at VDD or VSS, with no external circuitry drawing current from the I/O pins, and disable external clocks. I/O pins that are hi-impedance inputs should be pulled high or low externally to avoid switching currents caused by floating inputs. The T0CKI input should also be at VDD or VSS. The contribution from on-chip pull-ups on PORTB should be considered.

The $\overline{MCLR}$ pin must be at a logic high level (VIHMC).

It should be noted that a RESET generated by a WDT time-out does not drive the $\overline{MCLR}$ pin low.

6.11.2 WAKE-UP FROM SLEEP

The device can wake-up from SLEEP through one of the following events:

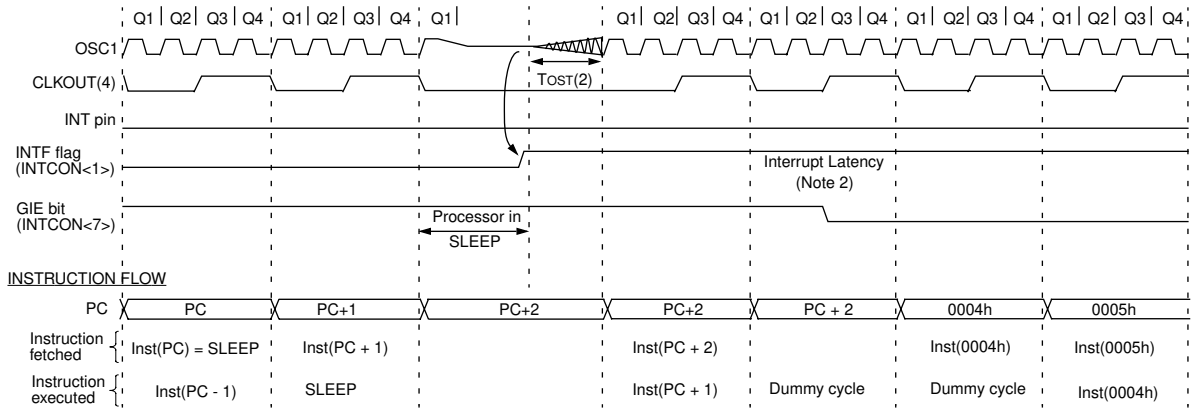
1. External reset input on $\overline{MCLR}$ pin.
2. WDT Wake-up (if WDT was enabled).
3. Interrupt from RB0/INT pin, RB port change, or data EEPROM write complete.

Peripherals cannot generate interrupts during SLEEP, since no on-chip Q clocks are present.

The first event ($\overline{MCLR}$ reset) will cause a device reset. The two latter events are considered a continuation of program execution. The $\overline{TO}$ and $\overline{PD}$ bits can be used to determine the cause of a device reset. The $\overline{PD}$ bit, which is set on power-up, is cleared when SLEEP is invoked. The $\overline{TO}$ bit is cleared if a WDT time-out occurred (and caused wake-up).

While the SLEEP instruction is being executed, the next instruction (PC + 1) is pre-fetched. For the device to wake-up through an interrupt event, the corresponding interrupt enable bit must be set (enabled). Wake-up occurs regardless of the state of the GIE bit. If the GIE bit is clear (disabled), the device continues execution at the instruction after the SLEEP instruction. If the GIE bit is set (enabled), the device executes the instruction after the SLEEP instruction and then branches to the interrupt address (0004h). In cases where the execution of the instruction following SLEEP is not desirable, the user should have a NOP after the SLEEP instruction.

FIGURE 6-13: WAKE-UP FROM SLEEP THROUGH INTERRUPT



Note 1: XT, HS or LP oscillator mode assumed.
2: TOST = 1024TOSC (drawing not to scale) This delay will not be there for RC osc mode.
3: GIE = '1' assumed. In this case after wake- up, the processor jumps to the interrupt routine. If GIE = '0', execution will continue in-line.
4: CLKOUT is not available in these osc modes, but shown here for timing reference.

6.11.3 WAKE-UP USING INTERRUPTS

When global interrupts are disabled (GIE cleared) and any interrupt source has both its interrupt enable bit and interrupt flag bit set, one of the following will occur:

- If the interrupt occurs **before** the execution of a SLEEP instruction, the SLEEP instruction will complete as a NOP. Therefore, the WDT and WDT postscaler will not be cleared, the $\overline{TO}$ bit will not be set and $\overline{PD}$ bits will not be cleared.
- If the interrupt occurs **during or after** the execution of a SLEEP instruction, the device will immediately wake up from sleep. The SLEEP instruction will be completely executed before the wake-up. Therefore, the WDT and WDT postscaler will be cleared, the $\overline{TO}$ bit will be set and the $\overline{PD}$ bit will be cleared.

Even if the flag bits were checked before executing a SLEEP instruction, it may be possible for flag bits to become set before the SLEEP instruction completes. To determine whether a SLEEP instruction executed, test the $\overline{PD}$ bit. If the $\overline{PD}$ bit is set, the SLEEP instruction was executed as a NOP.

To ensure that the WDT is cleared, a CLRWDT instruction should be executed before a SLEEP instruction.

6.12 Program Verification/Code Protection

If the code protection bit(s) have not been programmed, the on-chip program memory can be read out for verification purposes.

Note:	Microchip does not recommend code protecting windowed devices.

6.13 ID Locations

Four memory locations (2000h - 2004h) are designated as ID locations to store checksum or other code identification numbers. These locations are not accessible during normal execution but are readable and writable only during program/verify. Only the four least significant bits of ID location are usable.

6.14 In-Circuit Serial Programming

PIC16F84A microcontrollers can be serially programmed while in the end application circuit. This is simply done with two lines for clock and data, and three other lines for power, ground, and the programming voltage. Customers can manufacture boards with unprogrammed devices, and then program the microcontroller just before shipping the product, allowing the most recent firmware or custom firmware to be programmed.

For complete details of serial programming, please refer to the In-Circuit Serial Programming (ICSP™) Guide, (DS30277).

7.0 INSTRUCTION SET SUMMARY

Each PIC16CXXX instruction is a 14-bit word divided into an OPCODE which specifies the instruction type and one or more operands which further specify the operation of the instruction. The PIC16CXX instruction set summary in Table 7-2 lists **byte-oriented**, **bit-oriented**, and **literal and control** operations. Table 7-1 shows the opcode field descriptions.

For **byte-oriented** instructions, 'f' represents a file register designator and 'd' represents a destination designator. The file register designator specifies which file register is to be used by the instruction.

The destination designator specifies where the result of the operation is to be placed. If 'd' is zero, the result is placed in the W register. If 'd' is one, the result is placed in the file register specified in the instruction.

For **bit-oriented** instructions, 'b' represents a bit field designator which selects the number of the bit affected by the operation, while 'f' represents the number of the file in which the bit is located.

For **literal and control** operations, 'k' represents an eight or eleven bit constant or literal value.

TABLE 7-1 OPCODE FIELD DESCRIPTIONS

Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 0 or 1) The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
d	Destination select; d = 0: store result in W, d = 1: store result in file register f. Default is d = 1
PC	Program Counter
$\overline{TO}$	Time-out bit
$\overline{PD}$	Power-down bit

The instruction set is highly orthogonal and is grouped into three basic categories:

- **Byte-oriented** operations
- **Bit-oriented** operations
- **Literal and control** operations

All instructions are executed within one single instruction cycle, unless a conditional test is true or the program counter is changed as a result of an instruction. In this case, the execution takes two instruction cycles with the second cycle executed as a NOP. One instruction cycle consists of four oscillator periods. Thus, for an oscillator frequency of 4 MHz, the normal instruction execution time is 1 µs. If a conditional test is true or the program counter is changed as a result of an instruction, the instruction execution time is 2 µs.

Table 7-2 lists the instructions recognized by the MPASM assembler.

Figure 7-1 shows the general formats that the instructions can have.

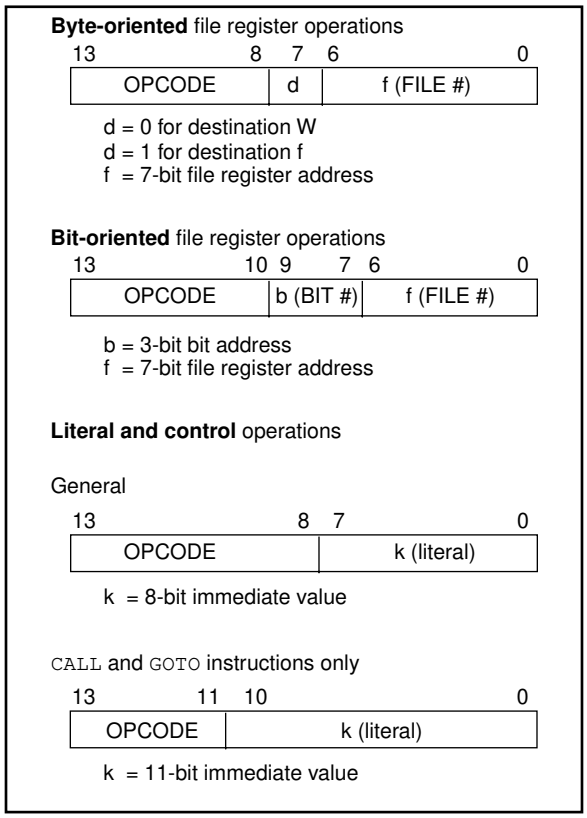
Note: To maintain upward compatibility with future PIC16CXXX products, do not use the OPTION and TRIS instructions.

All examples use the following format to represent a hexadecimal number:

0xhh

where h signifies a hexadecimal digit.

FIGURE 7-1: GENERAL FORMAT FOR INSTRUCTIONS



A description of each instruction is available in the PICmicro™ Mid-Range Reference Manual, (DS33023).

TABLE 7-2 PIC16CXXX INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	**f, d**	Add W and f	1	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF	**f, d**	AND W with f	1	00 0101 dfff ffff	Z	1,2
CLRF	**f**	Clear f	1	00 0001 lfff ffff	Z	2
CLRW	**-**	Clear W	1	00 0001 0xxx xxxx	Z	
COMF	**f, d**	Complement f	1	00 1001 dfff ffff	Z	1,2
DECF	**f, d**	Decrement f	1	00 0011 dfff ffff	Z	1,2
DECFSZ	**f, d**	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff		1,2,3
INCF	**f, d**	Increment f	1	00 1010 dfff ffff	Z	1,2
INCFSZ	**f, d**	Increment f, Skip if 0	1(2)	00 1111 dfff ffff		1,2,3
IORWF	**f, d**	Inclusive OR W with f	1	00 0100 dfff ffff	Z	1,2
MOVF	**f, d**	Move f	1	00 1000 dfff ffff	Z	1,2
MOVWF	**f**	Move W to f	1	00 0000 lfff ffff		
NOP	**-**	No Operation	1	00 0000 0xx0 0000		
RLF	**f, d**	Rotate Left f through Carry	1	00 1101 dfff ffff	C	1,2
RRF	**f, d**	Rotate Right f through Carry	1	00 1100 dfff ffff	C	1,2
SUBWF	**f, d**	Subtract W from f	1	00 0010 dfff ffff	C,DC,Z	1,2
SWAPF	**f, d**	Swap nibbles in f	1	00 1110 dfff ffff		1,2
XORWF	**f, d**	Exclusive OR W with f	1	00 0110 dfff ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	**f, b**	Bit Clear f	1	01 00bb bfff ffff		1,2
BSF	**f, b**	Bit Set f	1	01 01bb bfff ffff		1,2
BTFSC	**f, b**	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff		3
BTFSS	**f, b**	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff		3
LITERAL AND CONTROL OPERATIONS						
ADDLW	**k**	Add literal and W	1	11 111x kkkk kkkk	C,DC,Z	
ANDLW	**k**	AND literal with W	1	11 1001 kkkk kkkk	Z	
CALL	**k**	Call subroutine	2	10 0kkk kkkk kkkk		
CLRWDT	**-**	Clear Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$,$\overline{PD}$	
GOTO	**k**	Go to address	2	10 1kkk kkkk kkkk		
IORLW	**k**	Inclusive OR literal with W	1	11 1000 kkkk kkkk	Z	
MOVLW	**k**	Move literal to W	1	11 00xx kkkk kkkk		
RETFIE	**-**	Return from interrupt	2	00 0000 0000 1001		
RETLW	**k**	Return with literal in W	2	11 01xx kkkk kkkk		
RETURN	**-**	Return from Subroutine	2	00 0000 0000 1000		
SLEEP	**-**	Go into standby mode	1	00 0000 0110 0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	**k**	Subtract W from literal	1	11 110x kkkk kkkk	C,DC,Z	
XORLW	**k**	Exclusive OR literal with W	1	11 1010 kkkk kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

8.0 DEVELOPMENT SUPPORT

8.1 Development Tools

The PICmicro™ microcontrollers are supported with a full range of hardware and software development tools:

- MPLAB™-ICE Real-Time In-Circuit Emulator
- ICEPIC™ Low-Cost PIC16C5X and PIC16CXXX In-Circuit Emulator
- PRO MATE® II Universal Programmer
- PICSTART® Plus Entry-Level Prototype Programmer
- SIMICE
- PICDEM-1 Low-Cost Demonstration Board
- PICDEM-2 Low-Cost Demonstration Board
- PICDEM-3 Low-Cost Demonstration Board
- MPASM Assembler
- MPLAB™ SIM Software Simulator
- MPLAB-C17 (C Compiler)
- Fuzzy Logic Development System (*fuzzy*TECH®–MP)
- KEELOQ® Evaluation Kits and Programmer

8.2 MPLAB-ICE: High Performance Universal In-Circuit Emulator with MPLAB IDE

The MPLAB-ICE Universal In-Circuit Emulator is intended to provide the product development engineer with a complete microcontroller design tool set for PICmicro microcontrollers (MCUs). MPLAB-ICE is supplied with the MPLAB Integrated Development Environment (IDE), which allows editing, “make” and download, and source debugging from a single environment.

Interchangeable processor modules allow the system to be easily reconfigured for emulation of different processors. The universal architecture of the MPLAB-ICE allows expansion to support all new Microchip microcontrollers.

The MPLAB-ICE Emulator System has been designed as a real-time emulation system with advanced features that are generally found on more expensive development tools. The PC compatible 386 (and higher) machine platform and Microsoft Windows® 3.x or Windows 95 environment were chosen to best make these features available to you, the end user.

MPLAB-ICE is available in two versions. MPLAB-ICE 1000 is a basic, low-cost emulator system with simple trace capabilities. It shares processor modules with the MPLAB-ICE 2000. This is a full-featured emulator system with enhanced trace, trigger, and data monitoring features. Both systems will operate across the entire operating speed reange of the PICmicro MCU.

8.3 ICEPIC: Low-Cost PICmicro™ In-Circuit Emulator

ICEPIC is a low-cost in-circuit emulator solution for the Microchip PIC12CXXX, PIC16C5X and PIC16CXXX families of 8-bit OTP microcontrollers.

ICEPIC is designed to operate on PC-compatible machines ranging from 386 through Pentium™ based machines under Windows 3.x, Windows 95, or Windows NT environment. ICEPIC features real time, non-intrusive emulation.

8.4 PRO MATE II: Universal Programmer

The PRO MATE II Universal Programmer is a full-featured programmer capable of operating in stand-alone mode as well as PC-hosted mode. PRO MATE II is CE compliant.

The PRO MATE II has programmable VDD and VPP supplies which allows it to verify programmed memory at VDD min and VDD max for maximum reliability. It has an LCD display for displaying error messages, keys to enter commands and a modular detachable socket assembly to support various package types. In stand-alone mode the PRO MATE II can read, verify or program PIC12CXXX, PIC14C000, PIC16C5X, PIC16CXXX and PIC17CXX devices. It can also set configuration and code-protect bits in this mode.

8.5 PICSTART Plus Entry Level Development System

The PICSTART programmer is an easy-to-use, low-cost prototype programmer. It connects to the PC via one of the COM (RS-232) ports. MPLAB Integrated Development Environment software makes using the programmer simple and efficient. PICSTART Plus is not recommended for production programming.

PICSTART Plus supports all PIC12CXXX, PIC14C000, PIC16C5X, PIC16CXXX and PIC17CXX devices with up to 40 pins. Larger pin count devices such as the PIC16C923, PIC16C924 and PIC17C756 may be supported with an adapter socket. PICSTART Plus is CE compliant.

8.6 SIMICE Entry-Level Hardware Simulator

SIMICE is an entry-level hardware development system designed to operate in a PC-based environment with Microchip's simulator MPLAB™-SIM. Both SIMICE and MPLAB-SIM run under Microchip Technology's MPLAB Integrated Development Environment (IDE) software. Specifically, SIMICE provides hardware simulation for Microchip's PIC12C5XX, PIC12CE5XX, and PIC16C5X families of PICmicro™ 8-bit microcontrollers. SIMICE works in conjunction with MPLAB-SIM to provide non-real-time I/O port emulation. SIMICE enables a developer to run simulator code for driving the target system. In addition, the target system can provide input to the simulator code. This capability allows for simple and interactive debugging without having to manually generate MPLAB-SIM stimulus files. SIMICE is a valuable debugging tool for entry-level system development.

8.7 PICDEM-1 Low-Cost PICmicro Demonstration Board

The PICDEM-1 is a simple board which demonstrates the capabilities of several of Microchip's microcontrollers. The microcontrollers supported are: PIC16C5X (PIC16C54 to PIC16C58A), PIC16C61, PIC16C62X, PIC16C71, PIC16C8X, PIC17C42, PIC17C43 and PIC17C44. All necessary hardware and software is included to run basic demo programs. The users can program the sample microcontrollers provided with the PICDEM-1 board, on a PRO MATE II or PICSTART-Plus programmer, and easily test firmware. The user can also connect the PICDEM-1 board to the MPLAB-ICE emulator and download the firmware to the emulator for testing. Additional prototype area is available for the user to build some additional hardware and connect it to the microcontroller socket(s). Some of the features include an RS-232 interface, a potentiometer for simulated analog input, push-button switches and eight LEDs connected to PORTB.

8.8 PICDEM-2 Low-Cost PIC16CXX Demonstration Board

The PICDEM-2 is a simple demonstration board that supports the PIC16C62, PIC16C64, PIC16C65, PIC16C73 and PIC16C74 microcontrollers. All the necessary hardware and software is included to run the basic demonstration programs. The user can program the sample microcontrollers provided with the PICDEM-2 board, on a PRO MATE II programmer or PICSTART-Plus, and easily test firmware. The MPLAB-ICE emulator may also be used with the PICDEM-2 board to test firmware. Additional prototype area has been provided to the user for adding additional hardware and connecting it to the microcontroller socket(s). Some of the features include a RS-232 interface, push-button switches, a potentiometer for simulated analog input, a Serial EEPROM to demonstrate usage of the I^2C bus and separate headers for connection to an LCD module and a keypad.

8.9 PICDEM-3 Low-Cost PIC16CXXX Demonstration Board

The PICDEM-3 is a simple demonstration board that supports the PIC16C923 and PIC16C924 in the PLCC package. It will also support future 44-pin PLCC microcontrollers with a LCD Module. All the necessary hardware and software is included to run the basic demonstration programs. The user can program the sample microcontrollers provided with the PICDEM-3 board, on a PRO MATE II programmer or PICSTART Plus with an adapter socket, and easily test firmware. The MPLAB-ICE emulator may also be used with the PICDEM-3 board to test firmware. Additional prototype area has been provided to the user for adding hardware and connecting it to the microcontroller socket(s). Some of the features include an RS-232 interface, push-button switches, a potentiometer for simulated analog input, a thermistor and separate headers for connection to an external LCD module and a keypad. Also provided on the PICDEM-3 board is an LCD panel, with 4 commons and 12 segments, that is capable of displaying time, temperature and day of the week. The PICDEM-3 provides an additional RS-232 interface and Windows 3.1 software for showing the demultiplexed LCD signals on a PC. A simple serial interface allows the user to construct a hardware demultiplexer for the LCD signals.

8.10 MPLAB Integrated Development Environment Software

The MPLAB IDE Software brings an ease of software development previously unseen in the 8-bit microcontroller market. MPLAB is a windows based application which contains:

- A full featured editor
- Three operating modes
  - editor
  - emulator
  - simulator
- A project manager
- Customizable tool bar and key mapping
- A status bar with project information
- Extensive on-line help

MPLAB allows you to:

- Edit your source files (either assembly or 'C')
- One touch assemble (or compile) and download to PICmicro tools (automatically updates all project information)
- Debug using:
  - source files
  - absolute listing file

The ability to use MPLAB with Microchip's simulator allows a consistent platform and the ability to easily switch from the low cost simulator to the full featured emulator with minimal retraining due to development tools.

8.11 Assembler (MPASM)

The MPASM Universal Macro Assembler is a PC-hosted symbolic assembler. It supports all microcontroller series including the PIC12C5XX, PIC14000, PIC16C5X, PIC16CXXX, and PIC17CXX families.

MPASM offers full featured Macro capabilities, conditional assembly, and several source and listing formats. It generates various object code formats to support Microchip's development tools as well as third party programmers.

MPASM allows full symbolic debugging from MPLAB-ICE, Microchip's Universal Emulator System.

MPASM has the following features to assist in developing software for specific use applications.

- Provides translation of Assembler source code to object code for all Microchip microcontrollers.
- Macro assembly capability.
- Produces all the files (Object, Listing, Symbol, and special) required for symbolic debug with Microchip's emulator systems.
- Supports Hex (default), Decimal and Octal source and listing formats.

MPASM provides a rich directive language to support programming of the PICmicro. Directives are helpful in making the development of your assemble source code shorter and more maintainable.

8.12 Software Simulator (MPLAB-SIM)

The MPLAB-SIM Software Simulator allows code development in a PC host environment. It allows the user to simulate the PICmicro series microcontrollers on an instruction level. On any given instruction, the user may examine or modify any of the data areas or provide external stimulus to any of the pins. The input/output radix can be set by the user and the execution can be performed in; single step, execute until break, or in a trace mode.

MPLAB-SIM fully supports symbolic debugging using MPLAB-C17 and MPASM. The Software Simulator offers the low cost flexibility to develop and debug code outside of the laboratory environment making it an excellent multi-project software development tool.

8.13 MPLAB-C17 Compiler

The MPLAB-C17 Code Development System is a complete ANSI 'C' compiler and integrated development environment for Microchip's PIC17CXXX family of microcontrollers. The compiler provides powerful integration capabilities and ease of use not found with other compilers.

For easier source level debugging, the compiler provides symbol information that is compatible with the MPLAB IDE memory display.

8.14 Fuzzy Logic Development System (*fuzzy*TECH-MP)

*fuzzy*TECH-MP fuzzy logic development tool is available in two versions - a low cost introductory version, MP Explorer, for designers to gain a comprehensive working knowledge of fuzzy logic system design; and a full-featured version, *fuzzy*TECH-MP, Edition for implementing more complex systems.

Both versions include Microchip's *fuzzy*LAB™ demonstration board for hands-on experience with fuzzy logic systems implementation.

8.15 SEEVAL® Evaluation and Programming System

The SEEVAL SEEPROM Designer's Kit supports all Microchip 2-wire and 3-wire Serial EEPROMs. The kit includes everything necessary to read, write, erase or program special features of any Microchip SEEPROM product including Smart Serials™ and secure serials. The Total Endurance™ Disk is included to aid in trade-off analysis and reliability calculations. The total kit can significantly reduce time-to-market and result in an optimized system.

8.16 KEELOQ® Evaluation and Programming Tools

KEELOQ evaluation and programming tools support Microchips HCS Secure Data Products. The HCS evaluation kit includes an LCD display to show changing codes, a decoder to decode transmissions, and a programming interface to program test transmitters.

TABLE 8-1: DEVELOPMENT TOOLS FROM MICROCHIP

		PIC12C5XX	PIC14000	PIC16C5X	PIC16CXXX	PIC16C6X	PIC16C7XX	PIC16C8X	PIC16C9XX	PIC17C4X	PIC17C7XX	24CXX 25CXX 93CXX	HCS200 HCS300 HCS301
Emulator Products	MPLAB™-ICE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
	ICEPIC™ Low-Cost In-Circuit Emulator			✓	✓	✓	✓	✓	✓				
Software Tools	MPLAB™ Integrated Development Environment	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
	MPLAB™ C17* Compiler									✓	✓		
	*fuzzy*TECH®-MP Explorer/Edition Fuzzy Logic Dev. Tool	✓	✓	✓	✓	✓	✓	✓	✓	✓			
	Total Endurance™ Software Model											✓	
Programmers	PICSTART®Plus Low-Cost Universal Dev. Kit	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
	PRO MATE® II Universal Programmer	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	KEELOQ® Programmer												✓
Demo Boards	SEEVAL® Designers Kit											✓	
	SIMICE	✓		✓									
	PICDEM-14A		✓										
	PICDEM-1			✓	✓			✓		✓			
	PICDEM-2					✓	✓						
	PICDEM-3								✓				
	KEELOQ® Evaluation Kit												✓
	KEELOQ Transponder Kit												✓

NOTES:

9.0 ELECTRICAL CHARACTERISTICS FOR PIC16F84A

Absolute Maximum Ratings †

Ambient temperature under bias ..... -55°C to +125°C

Storage temperature ..... -65°C to +150°C

Voltage on any pin with respect to V_{SS} (except V_{DD}, $\overline{MCLR}$, and RA4) ..... -0.3V to (V_{DD} + 0.3V)

Voltage on V_{DD} with respect to V_{SS} ..... -0.3 to +7.5V

Voltage on $\overline{MCLR}$ with respect to V_{SS} (1) ..... -0.3 to +14V

Voltage on RA4 with respect to V_{SS} ..... -0.3 to +8.5V

Total power dissipation (2) ..... 800 mW

Maximum current out of V_{SS} pin ..... 150 mA

Maximum current into V_{DD} pin ..... 100 mA

Input clamp current, I_{IK} (V_I < 0 or V_I > V_{DD}) ..... ± 20 mA

Output clamp current, I_{OK} (V_O < 0 or V_O > V_{DD}) ..... ± 20 mA

Maximum output current sunk by any I/O pin ..... 25 mA

Maximum output current sourced by any I/O pin ..... 20 mA

Maximum current sunk by PORTA ..... 80 mA

Maximum current sourced by PORTA ..... 50 mA

Maximum current sunk by PORTB ..... 150 mA

Maximum current sourced by PORTB ..... 100 mA

Note 1: Voltage spikes below V_{SS} at the $\overline{MCLR}$ pin, inducing currents greater than 80 mA, may cause latch-up. Thus, a series resistor of 50-100Ω should be used when applying a "low" level to the $\overline{MCLR}$ pin rather than pulling this pin directly to V_{SS}.

Note 2: Power dissipation is calculated as follows: $Pdis = V_{DD} \times \{I_{DD} - \sum I_{OH}\} + \sum \{(V_{DD}-V_{OH}) \times I_{OH}\} + \sum(V_{OL} \times I_{OL})$.

† NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

TABLE 9-1 CROSS REFERENCE OF DEVICE SPECS FOR OSCILLATOR CONFIGURATIONS AND FREQUENCIES OF OPERATION (COMMERCIAL DEVICES)

OSC	PIC16F84A-04	PIC16F84A-20	PIC16LF84A-04
RC	VDD: 4.0V to 5.5V IDD: 4.5 mA max. at 5.5V IPD: 14 µA max. at 4V, WDT dis Freq: 4.0 MHz max. at 4V	VDD: 4.5V to 5.5V IDD: 1.8 mA typ. at 5.5V IPD: 1.0 µA typ. at 5.5V, WDT dis Freq: 4..0 MHz max. at 4V	VDD: 2.0V to 5.5V IDD: 4.5 mA max. at 5.5V IPD: 7.0 µA max. at 2V WDT dis Freq: 2.0 MHz max. at 2V
XT	VDD: 4.0V to 5.5V IDD: 4.5 mA max. at 5.5V IPD: 14 µA max. at 4V, WDT dis Freq: 4.0 MHz max. at 4V	VDD: 4.5V to 5.5V IDD: 1.8 mA typ. at 5.5V IPD: 1.0 µA typ. at 5.5V, WDT dis Freq: 4.0 MHz max. at 4.5V	VDD: 2.0V to 5.5V IDD: 4.5 mA max. at 5.5V IPD: 7.0 µA max. at 2V WDT dis Freq: 2.0 MHz max. at 2V
HS	VDD: 4.5V to 5.5V IDD: 4.5 mA typ. at 5.5V IPD: 1.0 µA typ. at 4.5V, WDT dis Freq: 4.0 MHz max. at 4.5V	VDD: 4.5V to 5.5V IDD: 10 mA max. at 5.5V typ. IPD: 1.0 µA typ. at 4.5V, WDT dis Freq: 20 MHz max. at 4.5V	Do not use in HS mode
LP	VDD: 4.0V to 5.5V IDD: 48 µA typ. at 32 kHz, 2.0V IPD: 0.6 µA typ. at 3.0V, WDT dis Freq: 200 kHz max. at 4V	Do not use in LP mode	VDD: 2.0V to 5.5V IDD: 45 µA max. at 32 kHz, 2.0V IPD: 7 µA max. at 2.0V WDT dis Freq: 200 kHz max. at 2V
The shaded sections indicate oscillator selections which are tested for functionality, but not for MIN/MAX specifications. It is recommended that the user select the device type that ensures the specifications required.			

9.1 DC CHARACTERISTICS: PIC16F84A-04 (Commercial, Industrial) PIC16F84A-20 (Commercial, Industrial)

DC Characteristics Power Supply Pins			Standard Operating Conditions (unless otherwise stated) Operating temperature 0°C ≤ TA ≤ +70°C (commercial) -40°C ≤ TA ≤ +85°C (industrial)				
Parameter No.	**Sym**	**Characteristic**	**Min**	**Typ†**	**Max**	**Units**	**Conditions**
D001	VDD	**Supply Voltage**	4.0	—	5.5	V	XT, RC and LP osc configuration
D001A			4.5	—	5.5	V	HS osc configuration
D002*	VDR	**RAM Data Retention Voltage** (Note 1)	1.5*	—	—	V	Device in SLEEP mode
D003	VPOR	**VDD Start Voltage** to ensure internal Power-on Reset signal	—	VSS	—	V	See section on Power-on Reset for details
D004*	SVDD	**VDD Rise Rate** to ensure internal Power-on Reset signal	0.05*	—	—	V/ms	PWRT enabled (PWRTE bit clear)
D004A*			TBD	—	—		PWRT disabled (PWRTE bit set) See section on Power-on Reset for details
	IDD	**Supply Current**					RC and XT osc configuration (Note 4)
D010		(Note 2)	—	1.8	4.5	mA	FOSC = 4.0 MHz, VDD = 5.5V
D010A			—	3	10	mA	FOSC = 4.0 MHz, VDD = 5.5V (During Flash programming)
							HS osc configuration (PIC16F84A-20)
D013			—	10	20	mA	FOSC = 20 MHz, VDD = 5.5V
D020	IPD	**Power-down Current**	—	7.0	28	µA	VDD = 4.0V, WDT enabled, industrial
D021		(Note 3)	—	1.0	14	µA	VDD = 4.0V, WDT disabled, commercial
D021A			—	1.0	16	µA	VDD = 4.0V, WDT disabled, industrial
		Module Differential Current (Note 5)					
D022*	ΔIWDT	Watchdog Timer	—	6.0	20*	µA	WDTE bit set, VDD = 4.0V, commercial
			—	—	25*	µA	WDTE bit set, VDD = 4.0V, extended

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5.0V, 25˚C unless otherwise stated. These parameters are for design guidance only and are not tested.

Note 1: This is the limit to which VDD can be lowered without losing RAM data.

2: The supply current is mainly a function of the operating voltage and frequency. Other factors such as I/O pin loading and switching rate, oscillator type, internal code execution pattern, and temperature also have an impact on the current consumption.
The test conditions for all IDD measurements in active operation mode are:
OSC1=external square wave, from rail to rail; all I/O pins tristated, pulled to VDD, T0CKI = VDD, MCLR = VDD; WDT enabled/disabled as specified.

3: The power down current in SLEEP mode does not depend on the oscillator type. Power-down current is measured with the part in SLEEP mode, with all I/O pins in hi-impedance state and tied to VDD and VSS.

4: For RC osc configuration, current through Rext is not included. The current through the resistor can be estimated by the formula IR = VDD/2Rext (mA) with Rext in kOhm.

5: The Δ current is the additional current consumed when this peripheral is enabled. This current should be added to the base IDD measurement.

9.2 DC CHARACTERISTICS: PIC16LF84A-04 (Commercial, Industrial)

DC Characteristics Power Supply Pins			Standard Operating Conditions (unless otherwise stated) Operating temperature 0°C ≤ TA ≤ +70°C (commercial) -40°C ≤ TA ≤ +85°C (industrial)				
Parameter No.	**Sym**	**Characteristic**	**Min**	**Typ†**	**Max**	**Units**	**Conditions**
D001	VDD	**Supply Voltage**	2.0	—	5.5	V	XT, RC, and LP osc configuration
D002*	VDR	**RAM Data Retention Voltage** (Note 1)	1.5*	—	—	V	Device in SLEEP mode
D003	VPOR	**VDD Start Voltage** to ensure internal Power-on Reset signal	—	VSS	—	V	See section on Power-on Reset for details
D004* D004A*	SVDD	**VDD Rise Rate** to ensure internal Power-on Reset signal	0.05* TBD	— —	— —	V/ms	PWRT enabled (PWRTE bit clear) PWRT disabled (PWRTE bit set) See section on Power-on Reset for details
	IDD	**Supply Current**					RC and XT osc configuration (Note 4)
D010		(Note 2)	—	1	4	mA	FOSC = 2.0 MHz, VDD = 5.5V
D010A			—	3	10	mA	FOSC = 2.0 MHz, VDD = 5.5V (During Flash programming)
							LP osc configuration
D014			—	15	45	µA	FOSC = 32 kHz, VDD = 2.0V, WDT disabled
D020	IPD	**Power-down Current**	—	3.0	16	µA	VDD = 2.0V, WDT enabled, industrial
D021		(Note 3)	—	0.4	7.0	µA	VDD = 2.0V, WDT disabled, commercial
D021A			—	0.4	9.0	µA	VDD = 2.0V, WDT disabled, industrial
		Module Differential Current (Note 5)					
D022*	ΔIWDT	Watchdog Timer	—	6.0	20*	µA	WDTE bit set, VDD = 4.0V, commercial
			—	—	25*	µA	WDTE bit set, VDD = 4.0V, industrial

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5.0V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

Note 1: This is the limit to which VDD can be lowered in SLEEP mode without losing RAM data.

2: The supply current is mainly a function of the operating voltage and frequency. Other factors such as I/O pin loading and switching rate, oscillator type, internal code execution pattern, and temperature also have an impact on the current consumption.
The test conditions for all IDD measurements in active operation mode are:
OSC1=external square wave, from rail to rail; all I/O pins tristated, pulled to VDD, T0CKI = VDD, MCLR = VDD; WDT enabled/disabled as specified.

3: The power down current in SLEEP mode does not depend on the oscillator type. Power-down current is measured with the part in SLEEP mode, with all I/O pins in hi-impedance state and tied to VDD and VSS.

4: For RC osc configuration, current through Rext is not included. The current through the resistor can be estimated by the formula IR = VDD/2Rext (mA) with Rext in kOhm.

5: The Δ current is the additional current consumed when this peripheral is enabled. This current should be added to the base IDD measurement.

9.3 DC CHARACTERISTICS: PIC16F84A-04 (Commercial, Industrial) PIC16F84A-20 (Commercial, Industrial) PIC16LF84A-04 (Commercial, Industrial)

DC Characteristics All Pins Except Power Supply Pins			Standard Operating Conditions (unless otherwise stated) Operating temperature 0°C ≤ TA ≤ +70°C (commercial) -40°C ≤ TA ≤ +85°C (industrial) Operating voltage VDD range as described in DC spec Section 9.1 and Section 9.2.				
Parameter No.	**Sym**	**Characteristic**	**Min**	**Typ†**	**Max**	**Units**	**Conditions**
		Input Low Voltage					
	VIL	I/O ports					
D030		with TTL buffer	VSS	—	0.8	V	4.5V ≤ VDD ≤ 5.5V (Note 4)
D030A			VSS	—	0.16VDD	V	entire range (Note 4)
D031		with Schmitt Trigger buffer	VSS	—	0.2VDD	V	entire range
D032		$\overline{MCLR}$, RA4/T0CKI	Vss	—	0.2VDD	V	
D033		OSC1 (XT, HS and LP modes)	Vss	—	0.3VDD	V	(Note 1)
D034		OSC1 (RC mode)	Vss	—	0.1VDD	V	
		Input High Voltage					
	VIH	I/O ports		—			
D040		with TTL buffer	2.0	—	VDD	V	4.5V ≤ VDD ≤ 5.5V (Note 4)
D040A			0.25VDD +0.8	—	VDD	V	entire range (Note 4)
D041		with Schmitt Trigger buffer	0.8 VDD	—	VDD		entire range
D042		$\overline{MCLR}$, RA4/T0CKI	0.8 VDD	—	VDD	V	
D043		OSC1 (XT, HS and LP modes)	0.7 VDD	—	VDD	V	(Note 1)
D043A		OSC1 (RC mode)	0.9 VDD		VDD	V	
D050	VHYS	Hysteresis of Schmitt Trigger inputs	—	0.1	—	V	
D070	IPURB	**PORTB weak pull-up current**	50*	250*	400*	µA	VDD = 5.0V, VPIN = VSS
		Input Leakage Current (Note 2,3)					
D060	IIL	I/O ports	—	—	±1	µA	Vss ≤ VPIN ≤ VDD, Pin at hi-impedance
D061		$\overline{MCLR}$, RA4/T0CKI	—	—	±5	µA	Vss ≤ VPIN ≤ VDD
D063		OSC1	—	—	±5	µA	Vss ≤ VPIN ≤ VDD, XT, HS and LP osc configuration

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5.0V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

Note 1: In RC oscillator configuration, the OSC1 pin is a Schmitt Trigger input. Do not drive the PIC16F84A with an external clock while the device is in RC mode, or chip damage may result.

2: The leakage current on the $\overline{MCLR}$ pin is strongly dependent on the applied voltage level. The specified levels represent normal operating conditions. Higher leakage current may be measured at different input voltages.

3: Negative current is defined as coming out of the pin.

4: The user may choose the better of the two specs.

9.4 DC CHARACTERISTICS: PIC16F84A-04 (Commercial, Industrial) PIC16F84A-20 (Commercial, Industrial) PIC16LF84A-04 (Commercial, Industrial)

DC Characteristics All Pins Except Power Supply Pins			Standard Operating Conditions (unless otherwise stated) Operating temperature 0°C ≤ TA ≤ +70°C (commercial) -40°C ≤ TA ≤ +85°C (industrial) Operating voltage VDD range as described in DC spec Section 9.1 and Section 9.2.				
Parameter No.	**Sym**	**Characteristic**	**Min**	**Typ†**	**Max**	**Units**	**Conditions**
		Output Low Voltage					
D080	VOL	I/O ports	—	—	0.6	V	IOL = 8.5 mA, VDD = 4.5V
D083		OSC2/CLKOUT	—	—	0.6	V	IOL = 1.6 mA, VDD = 4.5V, (RC Mode Only)
		Output High Voltage					
D090	VOH	I/O ports (Note 3)	VDD-0.7	—	—	V	IOH = -3.0 mA, VDD = 4.5V
D092		OSC2/CLKOUT (Note 3)	VDD-0.7	—	—	V	IOH = -1.3 mA, VDD = 4.5V (RC Mode Only)
		Open Drain High Voltage					
D150	VOD	RA4 pin	—	—	8.5	V	
		Capacitive Loading Specs on Output Pins					
D100	COSC2	OSC2 pin	—	—	15	pF	In XT, HS and LP modes when external clock is used to drive OSC1.
D101	CIO	All I/O pins and OSC2 (RC mode)	—	—	50	pF	
		Data EEPROM Memory					
D120	ED	Endurance	1M*	10M	—	E/W	25°C at 5V
D121	VDRW	VDD for read/write	VMIN	—	5.5	V	VMIN = Minimum operating voltage
D122	TDEW	Erase/Write cycle time	—	4	8*	ms	
		Program Flash Memory					
D130	EP	Endurance	100*	1000	—	E/W	
D131	VPR	VDD for read	VMIN	—	5.5	V	VMIN = Minimum operating voltage
D132	VPEW	VDD for erase/write	4.5	—	5.5	V	
D133	TPEW	Erase/Write cycle time	—	4	8	ms	

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5.0V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

Note 1: In RC oscillator configuration, the OSC1 pin is a Schmitt Trigger input. Do not drive the PIC16F84A with an external clock while the device is in RC mode, or chip damage may result.

2: The leakage current on the $\overline{MCLR}$ pin is strongly dependent on the applied voltage level. The specified levels represent normal operating conditions. Higher leakage current may be measured at different input voltages.

3: Negative current is defined as coming out of the pin.

4: The user may choose the better of the two specs.

9.5 AC (Timing) Characteristics

9.5.1 TIMING PARAMETER SYMBOLOGY

The timing parameter symbols have been created following one of the following formats:

1. TppS2ppS
2. TppS

T			
F	Frequency	T	Time

Lowercase symbols (pp) and their meanings:

pp			
2	to	os,osc	OSC1
ck	CLKOUT	ost	oscillator start-up timer
cy	cycle time	pwrt	power-up timer
io	I/O port	rbt	RBx pins
inp	INT pin	t0	T0CKI
mc	$\overline{\text{MCLR}}$	wdt	watchdog timer

Uppercase symbols and their meanings:

S			
F	Fall	P	Period
H	High	R	Rise
I	Invalid (Hi-impedance)	V	Valid
L	Low	Z	High Impedance

9.5.2 TIMING CONDITIONS

The temperature and voltages specified in Table 9-2 apply to all timing specifications unless otherwise noted. All timings are measure between high and low measurement points as indicated in Figure 9-1. Figure 9-2 specifies the load conditions for the timing specifications.

TABLE 9-2 TEMPERATURE AND VOLTAGE SPECIFICATIONS - AC

AC CHARACTERISTICS	Standard Operating Conditions (unless otherwise stated) Operating temperature 0°C ≤ TA ≤ +70°C for commercial -40°C ≤ TA ≤ +85°C for industrial Operating voltage VDD range as described in DC spec Section 9.1 and Section 9.2

FIGURE 9-1: PARAMETER MEASUREMENT INFORMATION

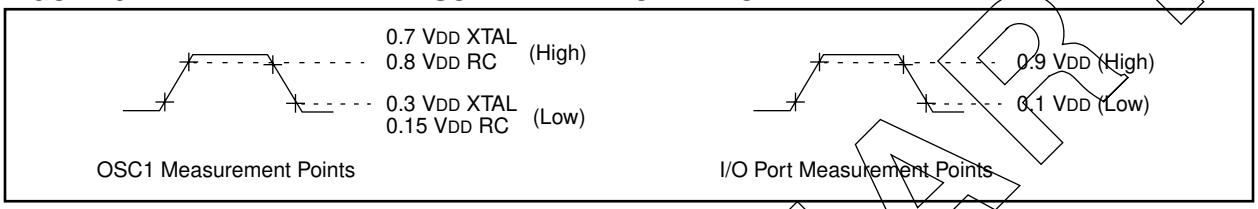
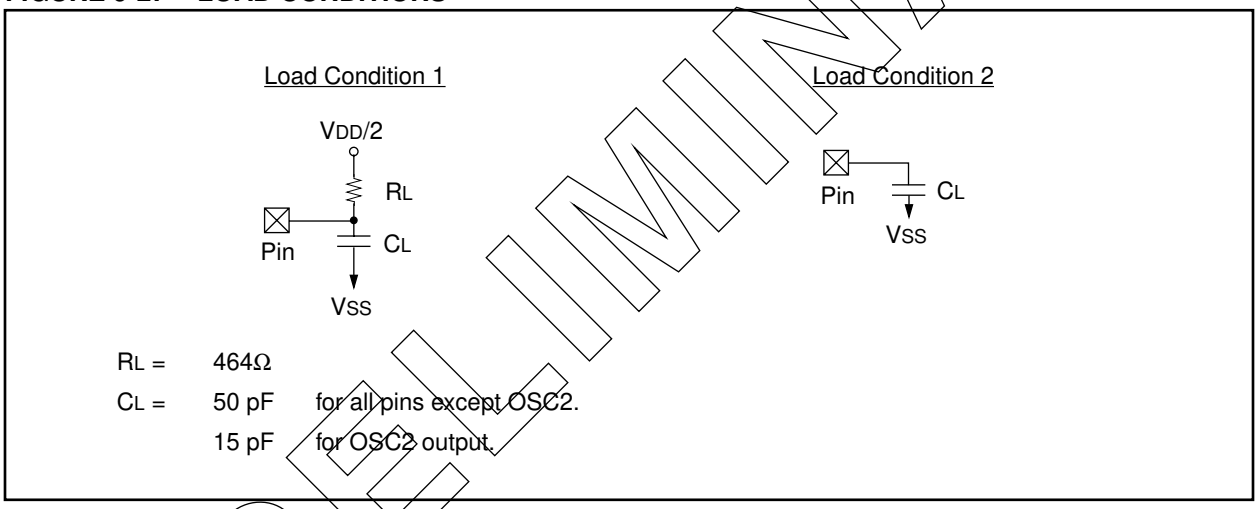


FIGURE 9-2: LOAD CONDITIONS



9.5.3 TIMING DIAGRAMS AND SPECIFICATIONS

FIGURE 9-3: EXTERNAL CLOCK TIMING

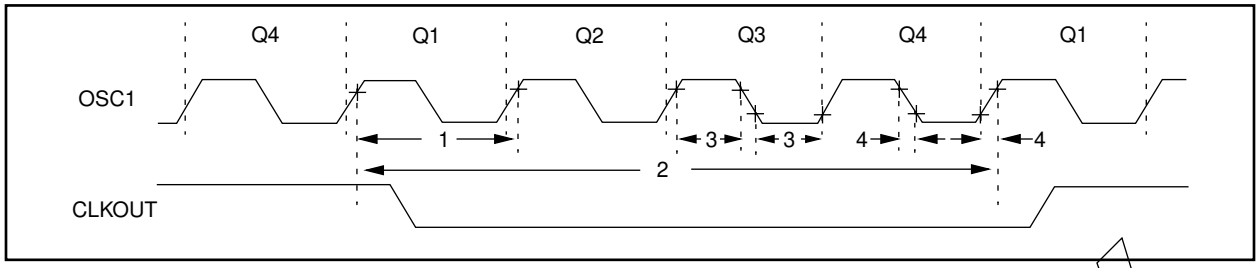


TABLE 9-3 EXTERNAL CLOCK TIMING REQUIREMENTS

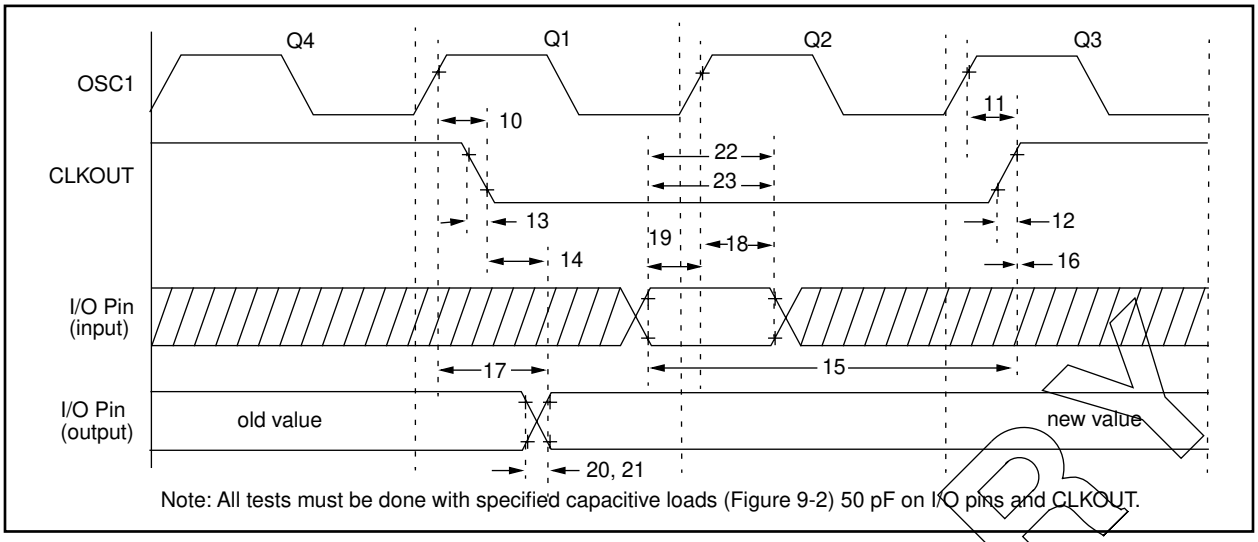
Parameter No.	Sym	Characteristic	Min	Typ†	Max	Units	Conditions	
	FOSC	**External CLKIN Frequency(1)**	DC	—	2	MHz	XT, RC osc	(-04, LF)
			DC	—	4	MHz	XT, RC osc	(-04)
			DC	—	20	MHz	HS osc	(-20)
			DC	—	200	kHz	LP osc	(-04, LF)
		Oscillator Frequency(1)	DC	—	2	MHz	RC osc	(-04, LF)
			DC	—	4	MHz	RC osc	(-04)
			0.1	—	2	MHz	XT osc	(-04, LF)
			0.1	—	4	MHz	XT osc	(-04)
			1.0	—	20	MHz	HS osc	(-20)
			DC	—	200	kHz	LP osc	(-04, LF)
1	Tosc	**External CLKIN Period(1)**	500	—	—	ns	XT, RC osc	(-04, LF)
			250	—	—	ns	XT, RC osc	(-04)
			100	—	—	ns	HS osc	(-20)
			5.0	—	—	µs	LP osc	(-04, LF)
		Oscillator Period(1)	500	—	—	ns	RC osc	(-04, LF)
			250	—	—	ns	RC osc	(-04)
			500	—	10,000	ns	XT osc	(-04, LF)
			250	—	10,000	ns	XT osc	(-04)
			100	—	1,000	ns	HS osc	(-20)
			5.0	—	—	µs	LP osc	(-04, LF)
2	TCY	**Instruction Cycle Time(1)**	0.4	4/Fosc	DC	µs		
3	TosL, TosH	**Clock in (OSC1) High or Low Time**	60 *	—	—	ns	XT osc	(-04, LF)
			50 *	—	—	ns	XT osc	(-04)
			2.0 *	—	—	µs	LP osc	(-04, LF)
			35 *	—	—	ns	HS osc	(-20)
4	TosR, TosF	**Clock in (OSC1) Rise or Fall Time**	25 *	—	—	ns	XT osc	(-04)
			50 *	—	—	ns	LP osc	(-04, LF)
			15 *	—	—	ns	HS osc	(-20)

* These parameters are characterized but no tested.

† Data in "Typ" column is at 5.0V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

Note 1: Instruction cycle period (TCY) equals four times the input oscillator time-base period. All specified values are based on characterization data for that particular oscillator type under standard operating conditions with the device executing code. Exceeding these specified limits may result in an unstable oscillator operation and/or higher than expected current consumption. All devices are tested to operate at "min." values with an external clock applied to the OSC1 pin. When an external clock input is used, the "Max." cycle time limit is "DC" (no clock) for all devices.

FIGURE 9-4: CLKOUT AND I/O TIMING



Note: All tests must be done with specified capacitive loads (Figure 9-2) 50 pF on I/O pins and CLKOUT.

TABLE 9-4 CLKOUT AND I/O TIMING REQUIREMENTS

Parameter No.	Sym	Characteristic		Min	Typ†	Max	Units	Conditions
10	TosH2ckL	OSC1↑ to CLKOUT↓	Standard	—	15	30 *	ns	Note 1
10A			Extended (LF)	—	15	120 *	ns	Note 1
11	TosH2ckH	OSC1↑ to CLKOUT↑	Standard	—	15	30 *	ns	Note 1
11A			Extended (LF)	—	15	120 *	ns	Note 1
12	TckR	CLKOUT rise time	Standard	—	15	30 *	ns	Note 1
12A			Extended (LF)	—	15	100 *	ns	Note 1
13	TckF	CLKOUT fall time	Standard	—	15	30 *	ns	Note 1
13A			Extended (LF)	—	15	100 *	ns	Note 1
14	TckL2ioV	CLKOUT ↓ to Port out valid		—	—	0.5TCY +20 *	ns	Note 1
15	TioV2ckH	Port in valid before CLKOUT ↑	Standard	0.30TCY + 30 *	—	—	ns	Note 1
			Extended (LF)	0.30TCY + 80 *	—	—	ns	Note 1
16	TckH2ioI	Port in hold after CLKOUT ↑		0 *	—	—	ns	Note 1
17	TosH2ioV	OSC1↑ (Q1 cycle) to Port out valid	Standard	—	—	125 *	ns	
			Extended (LF)	—	—	250 *	ns	
18	TosH2ioI	OSC1↑ (Q2 cycle) to Port input invalid (I/O in hold time)	Standard	10 *	—	—	ns	
			Extended (LF)	10 *	—	—	ns	
19	TioV2osH	Port input valid to OSC1↑ (I/O in setup time)	Standard	-75 *	—	—	ns	
			Extended (LF)	-175 *	—	—	ns	
20	TioR	Port output rise time	Standard	—	10	35 *	ns	
20A			Extended (LF)	—	10	70 *	ns	
21	TioF	Port output fall time	Standard	—	10	35 *	ns	
21A			Extended (LF)	—	10	70 *	ns	
22	Tinp	INT pin high	Standard	20 *	—	—	ns	
22A		or low time	Extended (LF)	55 *	—	—	ns	
23	Trbp	RB7:RB4 change INT	Standard	TOSC §	—	—	ns	
23A		high or low time	Extended (LF)	TOSC §	—	—	ns	

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5.0V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

§ By design

Note 1: Measurements are taken in RC Mode where CLKOUT output is 4 x TOSC.

FIGURE 9-5: RESET, WATCHDOG TIMER, OSCILLATOR START-UP TIMER AND POWER-UP TIMER TIMING

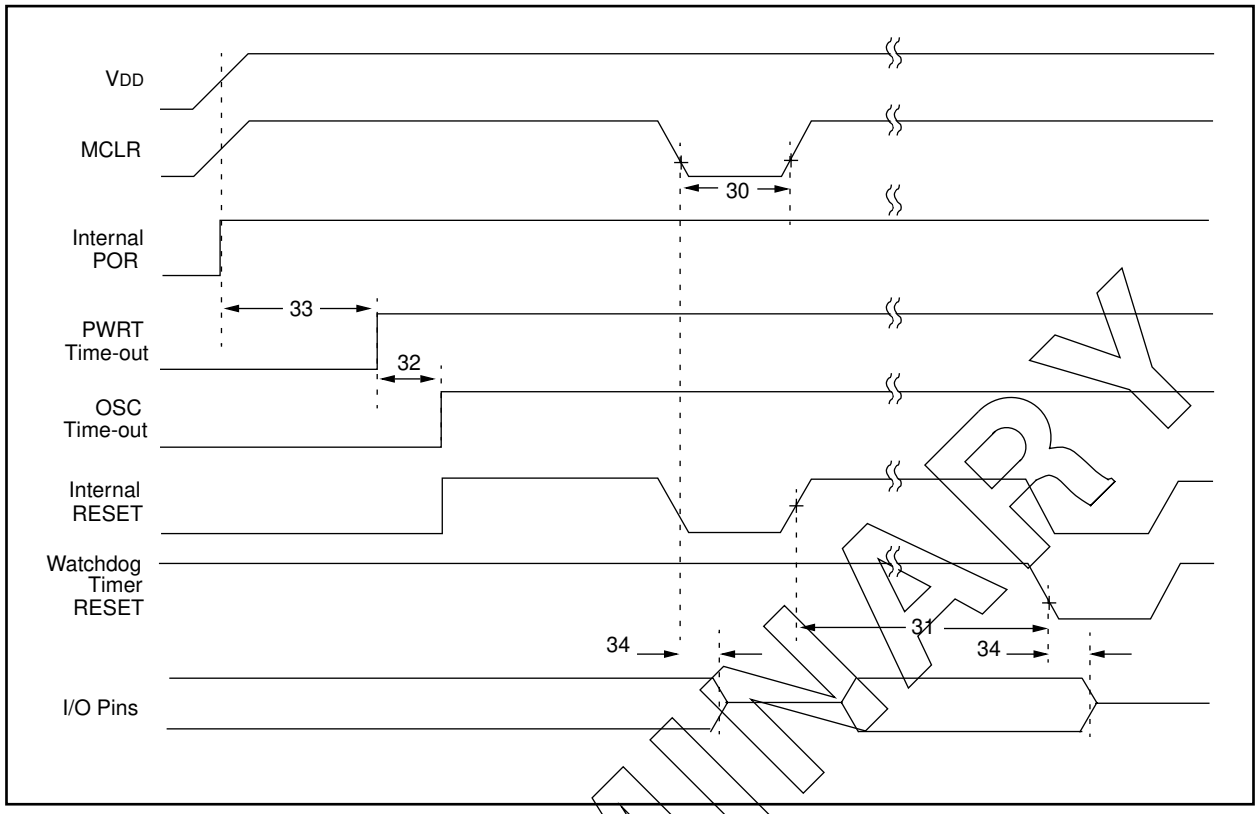


TABLE 9-5 RESET, WATCHDOG TIMER, OSCILLATOR START-UP TIMER AND POWER-UP TIMER REQUIREMENTS

Parameter No.	Sym	Characteristic	Min	Typ†	Max	Units	Conditions
30	TmcL	$\overline{MCLR}$ Pulse Width (low)	2 *	—	—	µs	VDD = 5.0V, extended
31	Twdt	Watchdog Timer Time-out Period (No Prescaler)	7 *	18	33 *	ms	VDD = 5.0V, extended
32	Tost	Oscillation Start-up Timer Period		1024TOSC		ms	TOSC = OSC1 period
33	Tpwrt	Power-up Timer Period	28 *	72	132 *	ms	VDD = 5.0V, extended
34	TIOZ	I/O Hi-impedance from $\overline{MCLR}$ Low or reset	—	—	100 *	ns	

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

FIGURE 9-6: TIMER0 CLOCK TIMINGS

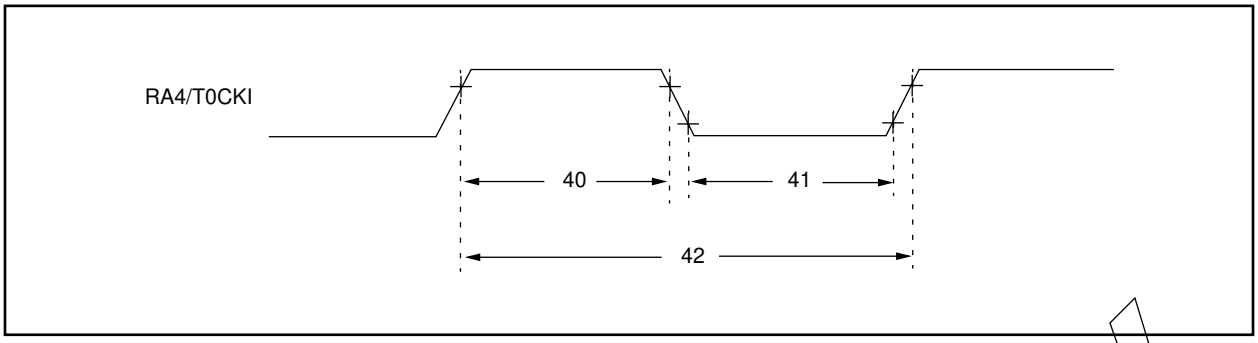


TABLE 9-6 TIMER0 CLOCK REQUIREMENTS

Parameter No.	Sym	Characteristic		Min	Typ†	Max	Units	Conditions
40	Tt0H	T0CKI High Pulse Width	No Prescaler	0.5TCY + 20 *	—	—	ns	
			With Prescaler	50 *	—	—	ns	2.0V ≤ VDD ≤ 3.0V
				30 *	—	—	ns	3.0V ≤ VDD ≤ 6.0V
41	Tt0L	T0CKI Low Pulse Width	No Prescaler	0.5TCY + 20 *	—	—	ns	
			With Prescaler	50 *	—	—	ns	2.0V ≤ VDD ≤ 3.0V
				20 *	—	—	ns	3.0V ≤ VDD ≤ 6.0V
42	Tt0P	T0CKI Period		$\frac{T_{CY} + 40}{N}$ *	—	—	ns	N = prescale value (2, 4, ..., 256)

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5.0V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

10.0 DC & AC CHARACTERISTICS GRAPHS/TABLES

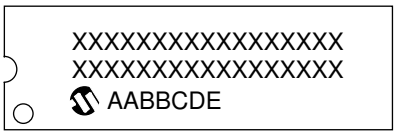
No data available at this time.

NOTES:

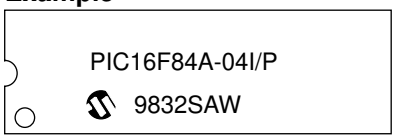
11.0 PACKAGING INFORMATION

11.1 Package Marking Information

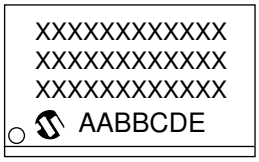
18L PDIP



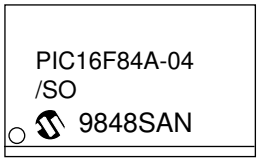
Example



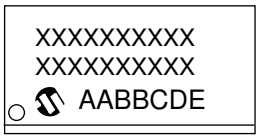
18L SOIC



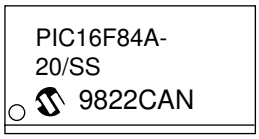
Example



20L SSOP



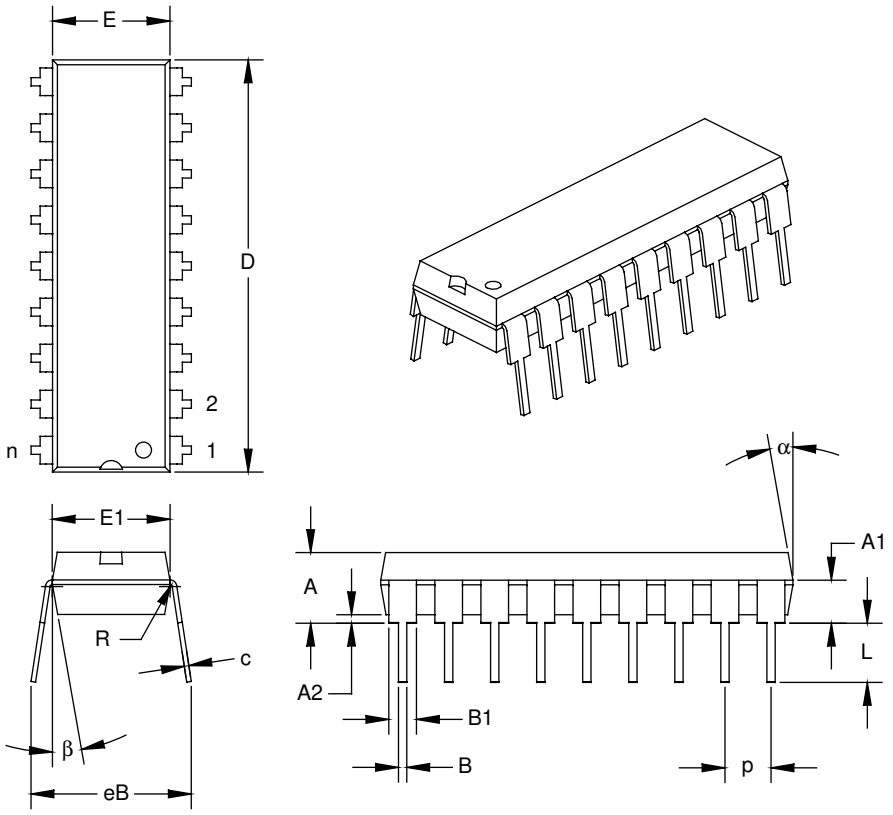
Example



Legend:	MM...M	Microchip part number information
	XX...X	Customer specific information*
	AA	Year code (last 2 digits of calendar year)
	BB	Week code (week of January 1 is week ‘01’)
	C	Facility code of the plant at which wafer is manufactured O = Outside Vendor C = 5” Line S = 6” Line H = 8” Line
	D	Mask revision number
	E	Assembly code of the plant or country of origin in which part was assembled
Note:	In the event the full Microchip part number cannot be marked on one line, it will be carried over to the next line thus limiting the number of available characters for customer specific information.	

* Standard OTP marking consists of Microchip part number, year code, week code, facility code, mask rev#, and assembly code. For OTP marking beyond this, certain price adders apply. Please check with your Microchip Sales Office. For QTP devices, any special marking adders are included in QTP price.

11.2 K04-007 18-Lead Plastic Dual In-line (P) – 300 mil



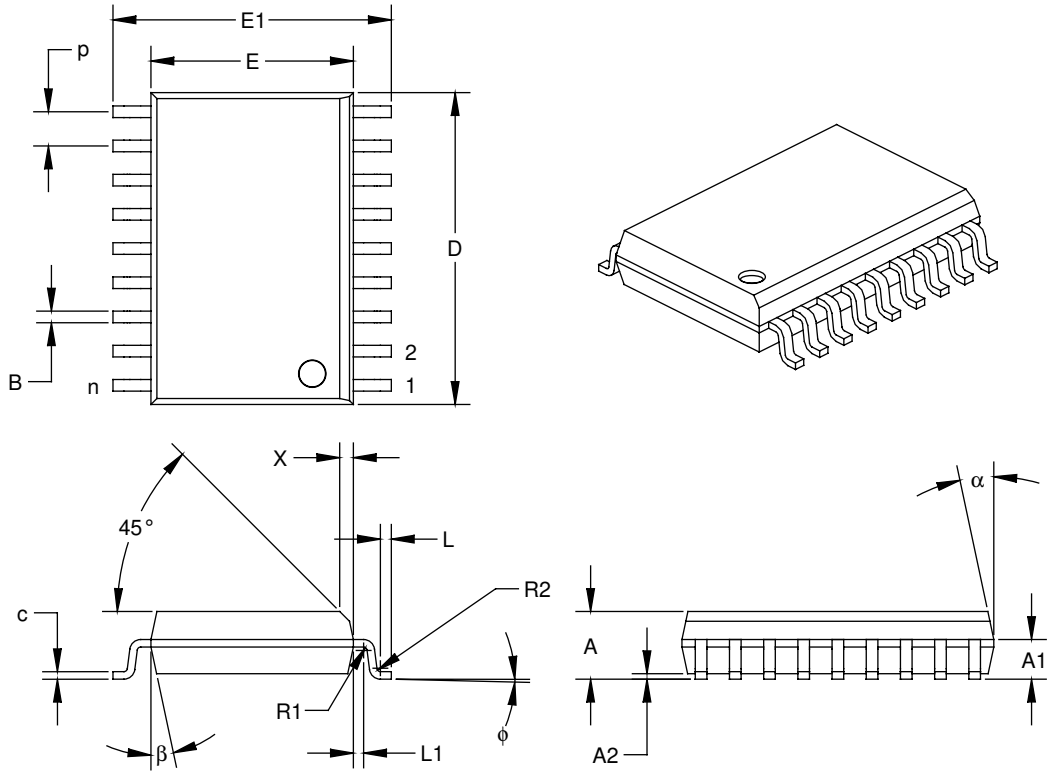
Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
PCB Row Spacing			0.300			7.62	
Number of Pins	n		18			18	
Pitch	p		0.100			2.54	
Lower Lead Width	B	0.013	0.018	0.023	0.33	0.46	0.58
Upper Lead Width	B1†	0.055	0.060	0.065	1.40	1.52	1.65
Shoulder Radius	R	0.000	0.005	0.010	0.00	0.13	0.25
Lead Thickness	c	0.005	0.010	0.015	0.13	0.25	0.38
Top to Seating Plane	A	0.110	0.155	0.155	2.79	3.94	3.94
Top of Lead to Seating Plane	A1	0.075	0.095	0.115	1.91	2.41	2.92
Base to Seating Plane	A2	0.000	0.020	0.020	0.00	0.51	0.51
Tip to Seating Plane	L	0.125	0.130	0.135	3.18	3.30	3.43
Package Length	D‡	0.890	0.895	0.900	22.61	22.73	22.86
Molded Package Width	E‡	0.245	0.255	0.265	6.22	6.48	6.73
Radius to Radius Width	E1	0.230	0.250	0.270	5.84	6.35	6.86
Overall Row Spacing	eB	0.310	0.349	0.387	7.87	8.85	9.83
Mold Draft Angle Top	α	5	10	15	5	10	15
Mold Draft Angle Bottom	β	5	10	15	5	10	15

* Controlling Parameter.

† Dimension “B1” does not include dam-bar protrusions. Dam-bar protrusions shall not exceed 0.003” (0.076 mm) per side or 0.006” (0.152 mm) more than dimension “B1.”

‡ Dimensions “D” and “E” do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.010” (0.254 mm) per side or 0.020” (0.508 mm) more than dimensions “D” or “E.”

11.3 K04-051 18-Lead Plastic Small Outline (SO) – Wide, 300 mil



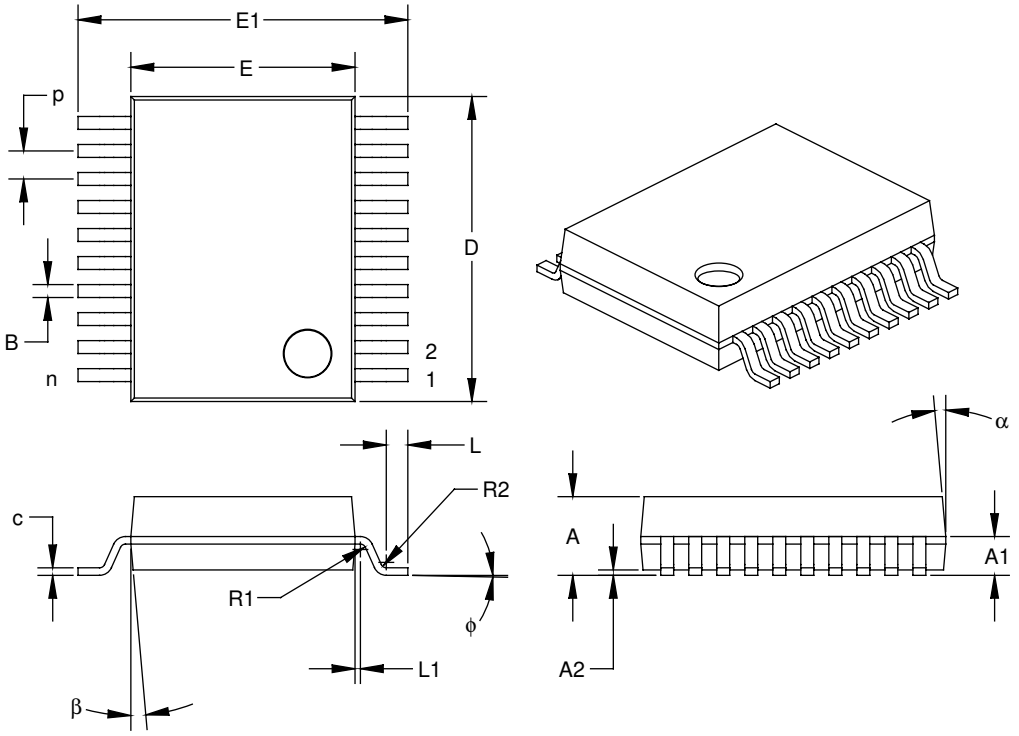
Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Pitch	p		0.050			1.27	
Number of Pins	n		18			18	
Overall Pack. Height	A	0.093	0.099	0.104	2.36	2.50	2.64
Shoulder Height	A1	0.048	0.058	0.068	1.22	1.47	1.73
Standoff	A2	0.004	0.008	0.011	0.10	0.19	0.28
Molded Package Length	D‡	0.450	0.456	0.462	11.43	11.58	11.73
Molded Package Width	E‡	0.292	0.296	0.299	7.42	7.51	7.59
Outside Dimension	E1	0.394	0.407	0.419	10.01	10.33	10.64
Chamfer Distance	X	0.010	0.020	0.029	0.25	0.50	0.74
Shoulder Radius	R1	0.005	0.005	0.010	0.13	0.13	0.25
Gull Wing Radius	R2	0.005	0.005	0.010	0.13	0.13	0.25
Foot Length	L	0.011	0.016	0.021	0.28	0.41	0.53
Foot Angle	ϕ	0	4	8	0	4	8
Radius Centerline	L1	0.010	0.015	0.020	0.25	0.38	0.51
Lead Thickness	c	0.009	0.011	0.012	0.23	0.27	0.30
Lower Lead Width	B†	0.014	0.017	0.019	0.36	0.42	0.48
Mold Draft Angle Top	α	0	12	15	0	12	15
Mold Draft Angle Bottom	β	0	12	15	0	12	15

* Controlling Parameter.

† Dimension “B” does not include dam-bar protrusions. Dam-bar protrusions shall not exceed 0.003” (0.076 mm) per side or 0.006” (0.152 mm) more than dimension “B.”

‡ Dimensions “D” and “E” do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.010” (0.254 mm) per side or 0.020” (0.508 mm) more than dimensions “D” or “E.”

11.4 K04-072 20-Lead Plastic Shrink Small Outine (SS) – 5.30 mm



Units		INCHES			MILLIMETERS*		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Pitch	p		0.026			0.65	
Number of Pins	n		20			20	
Overall Pack. Height	A	0.068	0.073	0.078	1.73	1.86	1.99
Shoulder Height	A1	0.026	0.036	0.046	0.66	0.91	1.17
Standoff	A2	0.002	0.005	0.008	0.05	0.13	0.21
Molded Package Length	D‡	0.278	0.283	0.289	7.07	7.20	7.33
Molded Package Width	E‡	0.205	0.208	0.212	5.20	5.29	5.38
Outside Dimension	E1	0.301	0.306	0.311	7.65	7.78	7.90
Shoulder Radius	R1	0.005	0.005	0.010	0.13	0.13	0.25
Gull Wing Radius	R2	0.005	0.005	0.010	0.13	0.13	0.25
Foot Length	L	0.015	0.020	0.025	0.38	0.51	0.64
Foot Angle	ϕ	0	4	8	0	4	8
Radius Centerline	L1	0.000	0.005	0.010	0.00	0.13	0.25
Lead Thickness	c	0.005	0.007	0.009	0.13	0.18	0.22
Lower Lead Width	B†	0.010	0.012	0.015	0.25	0.32	0.38
Mold Draft Angle Top	α	0	5	10	0	5	10
Mold Draft Angle Bottom	β	0	5	10	0	5	10

* Controlling Parameter.

† Dimension “B” does not include dam-bar protrusions. Dam-bar protrusions shall not exceed 0.003” (0.076 mm) per side or 0.006” (0.152 mm) more than dimension “B.”

‡ Dimensions “D” and “E” do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.010” (0.254 mm) per side or 0.020” (0.508 mm) more than dimensions “D” or “E.”

APPENDIX A: REVISION HISTORY

Version	Date	Revision Description
A	9/14/98	This is a new data sheet. However, the devices described in this data sheet are the upgrades to the devices found in the *PIC16F8X Data Sheet*, DS30430C.

APPENDIX B: CONVERSION CONSIDERATIONS

Considerations for converting from one PIC16X8X device to another are listed in Table B-1.

TABLE B-1: CONVERSION CONSIDERATIONS - PIC16C84, PIC16F83/F84, PIC16CR83/CR84, PIC16F84A

Difference	PIC16C84	PIC16F83/F84	PIC16CR83/ CR84	PIC16F84A
Program Memory size	1k x 14	512 x 14 / 1k x 14	512 x 14 / 1k x 14	1k x 14
Data Memory size	36 x 8	36 x 8 / 68 x 8	36 x 8 / 68 x 8	68 x 8
Voltage Range	2.0V - 6.0V (-40°C to +85°C)	2.0V - 6.0V (-40°C to +85°C)	2.0V - 6.0V (-40°C to +85°C)	2.0V - 5.5V (-40°C to +125°C)
Maximum Operating Frequency	10MHz	10MHz	10MHz	20MHz
Supply Current (I_{DD}). See parameter # D014 in the electrical spec's for more detail.	I_{DD} (typ) = 60µA I_{DD} (max) = 400µA (LP osc, F_{OSC} = 32kHz, V_{DD} = 2.0V, WDT disabled)	I_{DD} (typ) = 15µA I_{DD} (max) = 45µA (LP osc, F_{OSC} = 32kHz, V_{DD} = 2.0V, WDT disabled)	I_{DD} (typ) = 15µA I_{DD} (max) = 45µA (LP osc, F_{OSC} = 32kHz, V_{DD} = 2.0V, WDT disabled)	I_{DD} (typ) = 15µA I_{DD} (max) = 45µA (LP osc, F_{OSC} = 32kHz, V_{DD} = 2.0V, WDT disabled)
Power-down Current (I_{PD}). See parameters # D020, D021, and D021A in the electrical spec's for more detail.	I_{PD} (typ) = 26µA I_{PD} (max) = 100µA (V_{DD} = 2.0V, WDT disabled, industrial)	I_{PD} (typ) = 0.4µA I_{PD} (max) = 9µA (V_{DD} = 2.0V, WDT disabled, industrial)	I_{PD} (typ) = 0.4µA I_{PD} (max) = 6µA (V_{DD} = 2.0V, WDT disabled, industrial)	I_{PD} (typ) = 0.4µA I_{PD} (max) = 9µA (V_{DD} = 2.0V, WDT disabled, industrial)
Input Low Voltage (V_{IL}). See parameters # D032 and D034 in the electrical spec's for more detail.	V_{IL} (max) = 0.2V_{DD} (Osc1, RC mode)	V_{IL} (max) = 0.1V_{DD} (Osc1, RC mode)	V_{IL} (max) = 0.1V_{DD} (Osc1, RC mode)	V_{IL} (max) = 0.1V_{DD} (Osc1, RC mode)
Input High Voltage (V_{IH}). See parameter # D040 in the electrical spec's for more detail.	V_{IH} (min) = 0.36V_{DD} (I/O Ports with TTL, 4.5V ≤ V_{DD} ≤ 5.5V)	V_{IH} (min) = 2.4V (I/O Ports with TTL, 4.5V ≤ V_{DD} ≤ 5.5V)	V_{IH} (min) = 2.4V (I/O Ports with TTL, 4.5V ≤ V_{DD} ≤ 5.5V)	V_{IH} (min) = 2.4V (I/O Ports with TTL, 4.5V ≤ V_{DD} ≤ 5.5V)
Data EEPROM Memory Erase/Write cycle time (T_{DEW}). See parameter # D122 in the electrical spec's for more detail.	T_{DEW} (typ) = 10ms T_{DEW} (max) = 20ms	T_{DEW} (typ) = 10ms T_{DEW} (max) = 20ms	T_{DEW} (typ) = 10ms T_{DEW} (max) = 20ms	T_{DEW} (typ) = 4ms T_{DEW} (max) = 10ms

TABLE B-1: CONVERSION CONSIDERATIONS - PIC16C84, PIC16F83/F84, PIC16CR83/CR84, PIC16F84A

Difference	PIC16C84	PIC16F83/F84	PIC16CR83/ CR84	PIC16F84A
Port Output Rise/Fall time (TioR, TioF). See parameters #20, 20A, 21, and 21A in the electrical spec's for more detail.	TioR, TioF (max) = 25ns (C84) TioR, TioF (max) = 60ns (LC84)	TioR, TioF (max) = 35ns (C84) TioR, TioF (max) = 70ns (LC84)	TioR, TioF (max) = 35ns (C84) TioR, TioF (max) = 70ns (LC84)	TioR, TioF (max) = 35ns (C84) TioR, TioF (max) = 70ns (LC84)
$\overline{MCLR}$ on-chip filter. See parameter #30 in the electrical spec's for more detail.	No	Yes	Yes	Yes
PORTA and crystal oscillator values less than 500kHz	For crystal oscillator configurations operating below 500kHz, the device may generate a spurious internal Q-clock when PORTA<0> switches state.	N/A	N/A	N/A
RB0/INT pin	TTL	TTL/ST* (* Schmitt Trigger)	TTL/ST* (* Schmitt Trigger)	TTL/ST* (* Schmitt Trigger)
EEADR<7:6> and I_{DD}	It is recommended that the EEADR<7:6> bits be cleared. When either of these bits is set, the maximum I_{DD} for the device is higher than when both are cleared.	N/A	N/A	N/A
The polarity of the PWRTE bit	PWRTE	$\overline{PWRTE}$	$\overline{PWRTE}$	$\overline{PWRTE}$
Recommended value of R_{EXT} for RC oscillator circuits	R_{EXT} = 3kΩ - 100kΩ	R_{EXT} = 5kΩ - 100kΩ	R_{EXT} = 5kΩ - 100kΩ	R_{EXT} = 3kΩ - 100kΩ
GIE bit unintentional enable	If an interrupt occurs while the Global Interrupt Enable (GIE) bit is being cleared, the GIE bit may unintentionally be re-enabled by the user's Interrupt Service Routine (the `RETFIE` instruction).	N/A	N/A	N/A
Packages	PDIP, SOIC	PDIP, SOIC	PDIP, SOIC	PDIP, SOIC, SSOP

NOTES:

APPENDIX C: MIGRATION FROM BASELINE TO MIDRANGE DEVICES

This section discusses how to migrate from a baseline device (i.e., PIC16C5X) to a midrange device (i.e., PIC16CXXX).

The following is the list of feature improvements over the PIC16C5X microcontroller family:

1. Instruction word length is increased to 14 bits. This allows larger page sizes both in program memory (2K now as opposed to 512 before) and the register file (128 bytes now versus 32 bytes before).
2. A PC latch register (PCLATH) is added to handle program memory paging. PA2, PA1 and PA0 bits are removed from the status register and placed in the option register.
3. Data memory paging is redefined slightly. The STATUS register is modified.
4. Four new instructions have been added: RETURN, RETFIE, ADDLW, and SUBLW. Two instructions, TRIS and OPTION, are being phased out although they are kept for compatibility with PIC16C5X.
5. OPTION and TRIS registers are made addressable.
6. Interrupt capability is added. Interrupt vector is at 0004h.
7. Stack size is increased to 8 deep.
8. Reset vector is changed to 0000h.
9. Reset of all registers is revisited. Five different reset (and wake-up) types are recognized. Registers are reset differently.
10. Wake up from SLEEP through interrupt is added.
11. Two separate timers, the Oscillator Start-up Timer (OST) and Power-up Timer (PWRT), are included for more reliable power-up. These timers are invoked selectively to avoid unnecessary delays on power-up and wake-up.
12. PORTB has weak pull-ups and interrupt on change features.
13. T0CKI pin is also a port pin (RA4/T0CKI).
14. FSR is a full 8-bit register.
15. "In system programming" is made possible. The user can program PIC16CXX devices using only five pins: VDD, VSS, VPP, RB6 (clock) and RB7 (data in/out).

To convert code written for PIC16C5X to PIC16F84A, the user should take the following steps:

1. Remove any program memory page select operations (PA2, PA1, PA0 bits) for CALL, GOTO.
2. Revisit any computed jump operations (write to PC or add to PC, etc.) to make sure page bits are set properly under the new scheme.
3. Eliminate any data memory page switching. Redefine data variables for reallocation.
4. Verify all writes to STATUS, OPTION, and FSR registers since these have changed.
5. Change reset vector to 0000h.

INDEX

ON-LINE SUPPORT

Microchip provides on-line support on the Microchip World Wide Web (WWW) site.

The web site is used by Microchip as a means to make files and information easily available to customers. To view the site, the user must have access to the Internet and a web browser, such as Netscape or Microsoft Explorer. Files are also available for FTP download from our FTP site.

Connecting to the Microchip Internet Web Site

The Microchip web site is available by using your favorite Internet browser to attach to:

www.microchip.com

The file transfer site is available by using an FTP service to connect to:

ftp://ftp.futureone.com/pub/microchip

The web site and file transfer site provide a variety of services. Users may download files for the latest Development Tools, Data Sheets, Application Notes, User's Guides, Articles and Sample Programs. A variety of Microchip specific business information is also available, including listings of Microchip sales offices, distributors and factory representatives. Other data available for consideration is:

- Latest Microchip Press Releases
- Technical Support Section with Frequently Asked Questions
- Design Tips
- Device Errata
- Job Postings
- Microchip Consultant Program Member Listing
- Links to other useful web sites related to Microchip Products
- Conferences for products, Development Systems, technical information and more
- Listing of seminars and events

Systems Information and Upgrade Hot Line

The Systems Information and Upgrade Line provides system users a listing of the latest versions of all of Microchip's development systems software products. Plus, this line provides information on how customers can receive any currently available upgrade kits.The Hot Line Numbers are:

1-800-755-2345 for U.S. and most of Canada, and

1-602-786-7302 for the rest of the world.

980106

READER RESPONSE

It is our intention to provide you with the best documentation possible to ensure successful use of your Microchip product. If you wish to provide your comments on organization, clarity, subject matter, and ways in which our documentation can better serve you, please FAX your comments to the Technical Publications Manager at (602) 786-7578.

Please list the following information, and use this outline to provide us with your comments about this Data Sheet.

To: Technical Publications Manager Total Pages Sent

RE: Reader Response

From: Name _____

Company _____

Address _____

City / State / ZIP / Country _____

Telephone: (_____) _____ - _____ FAX: (_____) _____ - _____

Application (optional):

Would you like a reply? ___Y ___N

Device: **PIC16F84A** Literature Number: **DS35007A**

Questions:

1. What are the best features of this document?

2. How does this document meet your hardware and software development needs?

3. Do you find the organization of this data sheet easy to follow? If not, why?

4. What additions to the data sheet do you think would enhance the structure and subject?

5. What deletions from the data sheet could be made without affecting the overall usefulness?

6. Is there any incorrect or misleading information (what and where)?

7. How would you improve this document?

8. How would you improve our software, systems, and silicon products?

PIC16F84A PRODUCT IDENTIFICATION SYSTEM

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.

PART NO.	-XX	X	/XX	XXX
Device	Frequency Range	Temperature Range	Package	Pattern

Device	PIC16F84A[1], PIC16F84AT[2] PIC16LF84A[1], PIC16LF84AT[2]	
Frequency Range	04 20	= 4 MHz = 20 MHz
Temperature Range	blank I	= 0°C to +70°C (Commercial) = -40°C to +85°C (Industrial)
Package	P SO SS	= PDIP = SOIC (Gull Wing, 300 mil body) = SSOP
Pattern	3-digit Pattern Code for QTP, ROM (blank otherwise)	

Examples:

a) PIC16F84A -04/P 301 = Commercial temp., PDIP package, 4 MHz, normal VDD limits, QTP pattern #301.

b) PIC16LF84A - 04I/SO = Industrial temp., SOIC package, 200 kHz, Extended VDD limits.

c) PIC16F84A - 20I/P = Industrial temp., PDIP package, 20MHz, normal VDD limits.

Note 1: F = Standard VDD range
LF = Extended VDD range
2: T = in tape and reel - SOIC, SSOP packages only.

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-786-7200 Fax: 480-786-7277
Technical Support: 480-786-7627
Web Address: http://www.microchip.com

Atlanta
Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston
Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508-480-9990 Fax: 508-480-8575

Chicago
Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
Microchip Technology Inc.
4570 Westgrove Drive, Suite 160
Addison, TX 75248
Tel: 972-818-7423 Fax: 972-818-2924

Dayton
Microchip Technology Inc.
Two Prestige Place, Suite 150
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Detroit
Microchip Technology Inc.
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Los Angeles
Microchip Technology Inc.
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York
Microchip Technology Inc.
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

AMERICAS (continued)

Toronto
Microchip Technology Inc.
5925 Airport Road, Suite 200
Mississauga, Ontario L4V 1W1, Canada
Tel: 905-405-6279 Fax: 905-405-6253

ASIA/PACIFIC

Hong Kong
Microchip Asia Pacific
Unit 2101, Tower 2
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2-401-1200 Fax: 852-2-401-3431

Beijing
Microchip Technology, Beijing
Unit 915, 6 Chaoyangmen Bei Dajie
Dong Erhuan Road, Dongcheng District
New China Hong Kong Manhattan Building
Beijing 100027 PRC
Tel: 86-10-85282100 Fax: 86-10-85282104

India
Microchip Technology Inc.
India Liaison Office
No. 6, Legacy, Convent Road
Bangalore 560 025, India
Tel: 91-80-229-0061 Fax: 91-80-229-0062

Japan
Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa 222-0033 Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Shanghai
Microchip Technology
RM 406 Shanghai Golden Bridge Bldg.
2077 Yan'an Road West, Hong Qiao District
Shanghai, PRC 200335
Tel: 86-21-6275-5700 Fax: 86 21-6275-5060

ASIA/PACIFIC (continued)

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan, R.O.C
Microchip Technology Taiwan
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5858 Fax: 44-118 921-5835

Denmark
Microchip Technology Denmark ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Arizona Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 München, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

11/15/99

DNV Certification, Inc. USA — ANSI-RAB QMS ACCREDITED — DNV — DNV MSC The Netherlands Accredited by the RvA
ISO 9001 / QS-9000 REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.