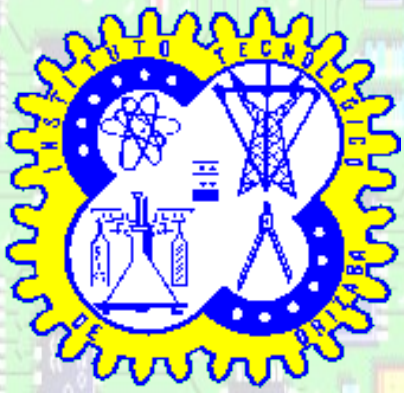
Instituto Tecnológico de Orizaba



Manual Teórico Práctico del microcontrolador PIC16F84A

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

INTRODUCCIÓN

El microcontrolador es uno de los logros más sobresalientes del siglo XX. Hoy en día, existen casi 15,000 millones de PIC's de alguna clase en uso. Para la mitad del siglo próximo, es posible que el microcontrolador típico tenga mayor poder de cómputo que las supercomputadoras más veloces de hoy.

Actualmente los podemos encontrar en cualquier sitio: microondas, frigoríficos, coches, aviones, mandos a distancia, radios, televisores, etc.

El objetivo principal de esta guía consiste en que el lector comprenda el funcionamiento de los microcontroladores y sea capaz de emplearlos en situaciones que requieren una solución práctica.

Dentro de este manual es posible encontrar información acerca del origen de esta línea de microcontroladores PIC, así como las diferentes familias que integran la amplia gama de dispositivos PIC Micro.

Un aspecto fundamental es la explicación clara y explícita de la operación del microcontrolador PIC16F84A, dispositivo al cual está enfocado este manual y del cual se tratan temas relacionados con sus características, su arquitectura, la organización de memoria y los diversos periféricos que posee dicho PIC.

En lo que se refiere a la programación de este dispositivo tan popular, se ha desarrollado un pequeño tutorial acerca del MPLAB, el cual es el programa compilador y simulador más popular para este propósito. De la misma forma, para efectos de facilitar la grabación de un programa en el PIC, se anexó una guía de los grabadores de microcontroladores NOPPP y PonyProg2000. El primero de ellos consiste en un programa muy elemental, mientras que el segundo es un software que soporta la programación de varios dispositivos.

Finalmente, se considera que la práctica es un factor fundamental para la implementación de sistemas mediante microcontroladores, es por esto que se ha incluido dentro de esta obra, una sección de prácticas que ayudarán al usuario a familiarizarse con el manejo del PIC16F84A.

CAPÍTULO 1. INTRODUCCIÓN A LOS MICROCONTROLADORES

1.1 ¿Qué es un microcontrolador?

El microcontrolador se emplea en aplicaciones concretas y no es universal como el microprocesador.

Un microcontrolador es un circuito integrado que contiene una Unidad Central de Proceso (CPU) y una serie de recursos internos en un solo encapsulado. El CPU permite que el microcontrolador pueda ejecutar instrucciones almacenadas en una memoria. Los recursos internos son memoria RAM, memoria ROM, memoria EEPROM, puerto serie, puertos de entrada/salida, temporizadores, comparadores, etc. Se puede decir que es una evolución del microprocesador, al añadirle a este último las funciones que antes era necesario situar externamente con otros circuitos. El ejemplo típico esta en los puertos de entrada/salida y en la memoria RAM, en los sistemas con microprocesadores es necesario desarrollar una lógica de control y unos circuitos para implementar las funciones anteriores, con un microcontrolador no hace falta porque lo lleva todo incorporado, además en el caso de tener que ampliar el sistema ya ofrece recursos que facilitan esto. En resumen, un microcontrolador es un circuito integrado independiente, que no necesita memoria ni puertos externos pues los lleva en su interior, que facilita la tarea de diseño y reduce el espacio, traduciéndose todo a una aplicación final más económica y fiable.

1.2 Diferencia entre microcontrolador y microprocesador

En el diseño de un sistema con un microprocesador, además del procesador y dependiendo del circuito, se requiere de algunos circuitos integrados adicionales, como por ejemplo: memorias RAM para almacenar los datos temporalmente y memorias ROM para almacenar el programa que se encargará del proceso del equipo, un circuito integrado para los puertos de entrada y salida y finalmente un decodificador de direcciones.

Un microcontrolador es un solo circuito integrado que contiene todos los elementos electrónicos que se utilizaban para hacer funcionar un sistema basado con un

microprocesador; es decir contiene en un solo integrado la Unidad de Proceso, la memoria RAM, memoria ROM, puertos de entrada, salida y otros periféricos.

1.3 Historia de los microcontroladores

En 1965 GI formó una división de microelectrónica, destinada a generar las primeras arquitecturas viables de memoria EPROM y EEPROM. De forma complementaria GI Microelectronics División fue también responsable de desarrollar una amplia variedad de funciones digitales y analógicas en las familias AY3-xxxx y AY5-xxxx.

GI también creó un microprocesador de 16 bit, denominado CP1600, a principios de los 70's. Este fue un microprocesador razonable, pero no particularmente bueno manejando puertos de e/s. Alrededor del año 1975 GI diseño un controlador de interfase periférico (PIC) para algunas aplicaciones muy específicas. Fue diseñado para ser muy rápido, además de ser un controlador de e/s para una máquina de 16 bits pero sin necesitar una gran cantidad de funcionalidades, por lo que su lista de instrucciones fue pequeña.

No es de extrañar que la estructura diseñada en 1975 es, sustancialmente, la arquitectura del actual PIC16C5x. Además, la versión de 1975 fue fabricada con tecnología NMOS y sólo estaba disponible en versiones de ROM de máscara, pero seguía siendo un buen pequeño microcontrolador. El mercado, no obstante, no pensó así y el PIC quedó reducido a ser empleado únicamente por grandes fabricantes.

Durante los 80's, GI renovó su apariencia y se reestructuró, centrando su trabajo en sus principales actividades, semiconductores de potencia esencialmente, lo cual siguen haciendo actualmente con bastante éxito. GI Microelectronics Division cambió a GI Microelectronics Inc (una especie de subsidiaria), la cual fue finalmente vendida en 1985 a Venture Capital Investors, incluyendo la fábrica en Chandler, Arizona. La gente de Ventura realizó una profunda revisión de los productos en la compañía, desechando la mayoría de los componentes AY3, AY5 y otra serie de cosas, dejando sólo el negocio de los PIC y de las memorias EEPROM y EPROM. Se tomó la decisión de comenzar una nueva compañía, denominada Arizona Microchip Technology, tomando como elemento diferenciador sus controladores integrados.

Como parte de esta estrategia, la familia NMOS PIC165x fue rediseñada para emplear algo que la misma compañía fabricaba bastante bien, memoria EPROM. De esta forma nació el concepto de basarse en tecnología CMOS, OTP y memoria de programación EPROM, naciendo la familia PIC16C5x.

Actualmente Microchip ha realizado un gran número de mejoras a la arquitectura original, adaptándola a las actuales tecnologías y al bajo costo de los semiconductores.

1.4 Familias de microcontroladores

1.4.1 PIC12CXXX Familia pequeña (Gama enana)

La familia de la gama enana consiste en un grupo de microcontroladores PIC de reciente aparición que ha acaparado la atención del mercado. Su principal característica es su tamaño, ya que todos los integrantes de esta familia poseen 8 pines. Consumen menos de 2mA cuando trabajan a 5V y 4MHz, además de trabajar con alimentación de corriente continua comprendida entre 2.5V y 5.5V. El formato de sus instrucciones puede ser de 12 o 14 bits y su repertorio es de 33 o 35 instrucciones, respectivamente. En la figura 1.4.1.1 se muestra el diagrama de conexiones de los microcontroladores PIC12C508 y PIC12C509, pertenecientes a la familia pequeña. En la tabla 1.4.1.1 se muestran las características de algunos microcontroladores que forman parte de la gama enana.

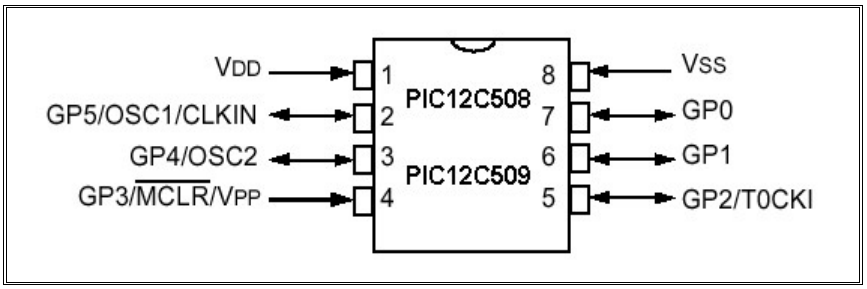


Figura 1.4.1.1 Diagrama de conexiones de los PIC12CXXX de la gama enana.

Tabla 1.4.1.1 Características de los modelos PIC12C(F)XXX de la gama enana

MODELO	MEMORIA PROGRAMA	MEMORIA DATOS	FRECUENCIA MÁXIMA	LÍNEAS E/S	ADC 8 BITS	TEMPORIZADORES	PINES
PIC12C508	512x12	25x8	4MHz	6		TMR0 + WDT	8
PIC12C509	1024x12	41x8	4MHz	6		TMR0 + WDT	8
PIC12C670	512x14	80x8	4MHz	6		TMR0 + WDT	8
PIC12C671	1024x14	128x8	4MHz	6	2	TMR0 + WDT	8
PIC12C672	2048x14	128x8	4MHz	6	4	TMR0 + WDT	8
PIC12C680	512x12 FLASH	80x8 16x8 EPROM	4MHz	6	4	TMR0 + WDT	8
PIC12C681	1024x14 FLASH	80x8 16x8 EPROM	4MHz	6		TMR0 + WDT	8

1.4.2 PIC16C5X Familia base (Gama baja o básica)

La familia 16C5X se consolidó como la base en el desarrollo de nuevas tecnologías ofreciendo la solución costeable más efectiva. Estos microcontroladores cuentan con un conjunto de instrucciones de 12 bits de ancho, 33 instrucciones, 2 niveles de acumulador, sin interrupciones. En algunos casos la memoria es del tipo ROM, definida en fábrica y se ofrecen en empaquetados de 18, 20 y 22 pines. En opciones de empaquetado SOIC o SSOP. Con bajo voltaje de operación hasta de 2 volt, hacen de esta familia un elemento ideal para ser operado en aplicaciones con baterías. En la figura 1.4.2.1 se muestra el PIC16C54, microcontrolador perteneciente a la familia base. En la tabla 1.4.2.1 se muestran las características de algunos microcontroladores que forman parte de la gama baja.

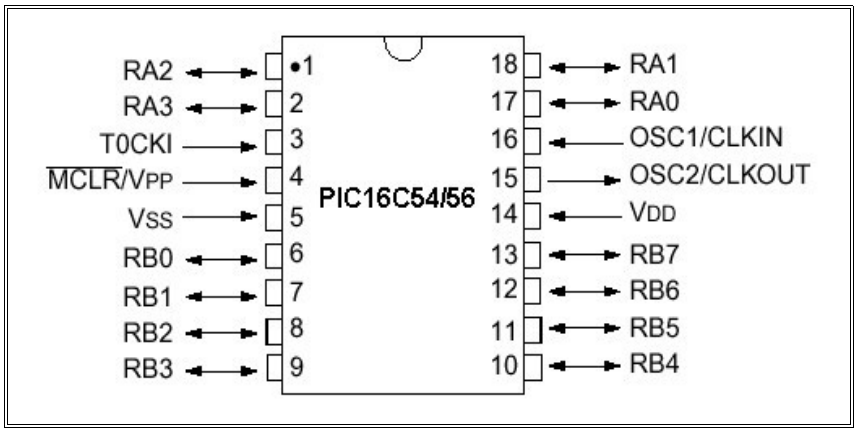


Figura 1.4.2.1. Diagrama de conexiones de los PIC de la gama baja que responden a la nomenclatura PIC16C54/56

Tabla 1.4.2.1. Principales características de los modelos de la gama baja

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

1.4.3 PIC16CXXX Familia de rango medio (Gama media)

La familia de rango medio ofrece un amplio rango de opciones, desde empaquetados de 18 hasta 44 pines, así como un alto nivel de integración de periféricos. Esta familia cuenta con un conjunto de instrucciones de 14 bits de ancho, 35 instrucciones, 8 niveles de acumulador. El PIC16C84 posee memoria EEPROM. y la capacidad de manejar interrupciones. En la figura 1.4.3.1 se muestra el PIC16C745, microcontrolador perteneciente a la familia base. En la tabla 1.4.3.1 se muestran las características de algunos microcontroladores que forman parte de la gama media.

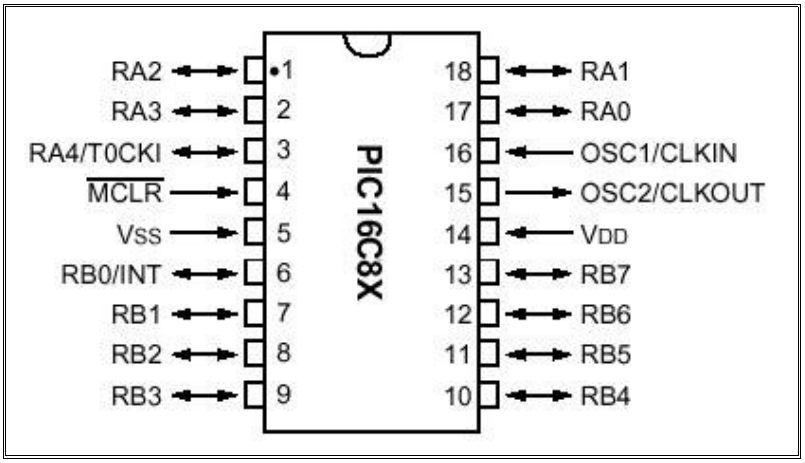


Figura 1.4.3.1. Diagrama de conexiones de los PIC16C8X, modelos representativos de la gama media.

Tabla 1.4.3.1. Características relevantes de los modelos PIC16X8X de la gama media.

MODELO	MEMORIA PROGRAMA	MEMORIA DATOS (Bytes)		REGISTROS ESPECÍFICOS	TEMPORIZADORES	INTERRUPCIONES	E/S	PINES
		RAM	EEPROM					
PIC16C84	1Kx14 EEPROM	36	64	11	TMR0 + WDT	4	13	18
PIC16F84	1Kx14 FLASH	68	64	11	TMR0 + WDT	4	13	18
PIC16F83	512x14 FLASH	36	64	11	TMR0 + WDT	4	13	18
PIC16CR84	1Kx14 ROM	68	64	11	TMR0 + WDT	4	13	18
PIC16CR83	512x14 ROM	36	64	11	TMR0 + WDT	4	13	18

1.4.4 PIC 17CXX Familia de rango alto (Gama alta)

Esta familia de alto rendimiento ofrece la mas alta velocidad de ejecución de todos los microcontroladores de 8 bits de la industria. Utilizando una arquitectura de instrucciones de 16 bits , 55 instrucciones, 16 niveles de acumulador. Mejora el conjunto de instrucciones y las capacidades de interrupción. A menos que se indique, la memoria es del tipo EPROM.

Adicionalmente existen otras familias derivadas, como los PIC16Fxx que emplean memoria del tipo FLASH. Se trata de una memoria no volátil, de bajo consumo, que se puede escribir y borrar en circuito al igual que las EEPROM, pero suelen disponer de mayor capacidad que estas últimas. El borrado sólo es posible con bloques completos y no se pueden realizar sobre posiciones concretas.

En la figura 1.4.4.1 se muestra el diagrama de conexiones de los PIC17C75X, microcontroladores pertenecientes a la familia de rango alto. En la tabla 1.4.4.1 se muestran las características de algunos microcontroladores que forman parte de la gama alta.

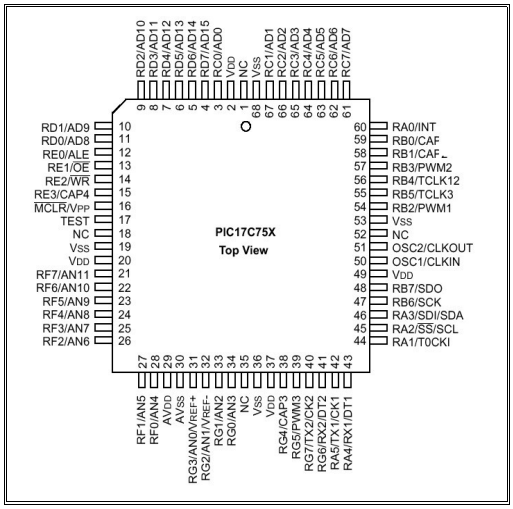


Figura 1.4.4.1 Diagrama de conexiones de los PIC de la gama alta que corresponden a la nomenclatura PIC17C75X

Tabla 1.4.4.1 Características más destacadas de los modelos PIC17CXXX de la gama alta

MODELO	MEMORIA PROGRAMA	MEMORIA DATOS RAM (Bytes)	REGISTROS ESPECÍFICOS	TEMPORIZADORES	CAP	PWM	CAD 10 bit	INTERRUPCIONES	E/S	MULTIPLICADOR HARDWARE	PINES
PIC17C42A	2Kx16	232	48	4 + WDT	2	2		11	33	8 x 8	40/44
PIC17C43	4Kx16	454	48	4 + WDT	2	2		11	33	8 x 8	40/44
PIC17C44	8Kx16	454	48	4 + WDT	2	2		11	33	8 x 8	40/44
PIC17C752	8Kx16	454	76	4 + WDT	4	3	12	18	50	8 x 8	64/68
PIC17C756	16Kx16	902	76	4 + WDT	4	3	12	18	50	8 x 8	64/68
CAP= Capturador; PWM= Modulador por ancho de pulso; CAD= Convertidor Análogo/Digital											

CAPÍTULO 2. CARACTERÍSTICAS DEL PIC16F84A

2.1 Generalidades del PIC16F84A

2.1.1 Descripción general del PIC16F84A

El PIC16F84 al igual que la versión mejorada de mayor velocidad PIC16F84A pertenecen a la familia de microcontroladores de 8 bits PIC16CXX, los cuales son de bajo costo, alto rendimiento y CMOS. Este grupo contiene los siguientes dispositivos:

- PIC16F83
- PIC16F84
- PIC16F84A
- PIC16CR83
- PIC16CR84

Todos los microcontrollers de PICmicro™ emplean una avanzada Arquitectura RISC (Computadores de Juego de Instrucciones Reducido). Los dispositivos PIC16F8X han reforzado características principales, pila profunda de ocho niveles y múltiples fuentes de interrupción internas y externas.

La separación del bus de instrucciones y del bus de datos en la arquitectura Harvard da lugar a un ancho de la palabra de instrucción de 14 bits, con un bus de datos separado de 8 bits de ancho. Los dos conductos de fase de instrucción permite que todas las instrucciones se ejecuten en un solo ciclo, excepto por los saltos del programa, los cuales requieren dos ciclos.

Un total de 35 instrucciones (Juego de instrucciones reducido) están disponibles. Adicionalmente un set del registro largo es usado para lograr un nivel de alto rendimiento.

Los microcontroladores PIC16F8x típicamente logran una compresión de código de 2:1 y un incremento de 4:1 en la mejora de velocidad (a 20 MHz) sobre otros microcontroladores de 8 bits de su misma clase.

En la tabla 2.1.1.1 se muestran las características principales de la serie de microcontroladores PIC16F8x.

Los microcontroladores PIC16F8x poseen 68 bytes de RAM, 64 bytes de memoria de datos EEPROM y 13 pines de entrada / Salida. También está disponible un contador / temporizador.

La familia PIC16CXX tiene características especiales para reducir los componentes externos, reduciendo así el costo, reforzando la confiabilidad del sistema y reduciendo el consumo de energía.

Hay cuatro opciones del oscilador, de los cuales el oscilador RC mediante un solo pin, proporciona una solución de bajo costo, el oscilador LP reduce el consumo de energía, el oscilador XT es un cristal estándar y el oscilador HS se refiere a cristales de alta velocidad.

El modo SLEEP (Bajo de consumo de corriente) ofrece un ahorro de energía.

El usuario puede despertar al chip del modo SLEEP a través de varias interrupciones internas y externas para reestablecer su funcionamiento.

Un confiable temporizador de perro guardián (WDT) está integrado en el chip, así como su propio oscilador RC proporciona protección cuando el programa ha sido ciclado.

Los dispositivos con memoria de programa Flash permite que el mismo dispositivo pueda ser usado como prototipo y como elemento de producción.

La gran capacidad de reprogramar el código del circuito permite su actualización sin que el dispositivo sea removido de su aplicación final.

Tabla 2.1.1.1. Lista de características del PIC16F8x

	PIC16F83	PIC16CR83	PIC16F84	PIC16CR84	PIC16F84A
Máxima frecuencia de operación MHz	10	10	10	10	20
Memoria de programa Flash	512	-	1K	-	IK
Memoria de programa EEPROM	-	-	-	-	-
Memoria de programa ROM	-	512	-	1K	-
Memoria de datos (bytes)	36	36	68	68	68
EEPROM de datos (bytes)	64	64	64	64	64
Módulos Temporizadores	TMRO	TMRO	TMRO	TMRO	TMRO
Fuentes de interrupción	4	4	4	4	4
Pines de I/O	13	13	13	13	13
Rangos de Voltaje (Volts)	4.0-6.0	2.0-6.0	4.0-6.0	2.0-6.0	4.0-6.0
Empaques	Dip 18-pin, SOIC	Dip 18-pin, SOIC	Dip 18-pin, SOIC	Dip 18-pin, SOIC	Dip 18-pin, SOIC

Esto es muy útil en situaciones, donde el dispositivo es difícilmente accesible. También es útil en aplicaciones remotas donde el código necesita ser actualizado.

El PIC16F8x encaja perfectamente en rangos de aplicaciones donde se requiere controlar pequeños motores, en la utilización de censores remotos de baja potencia, cerrojos electrónicos y pequeñas tarjetas con dispositivos de seguridad.

La tecnología Flash/EEPROM optimiza los programas de aplicación (transmisión de códigos, velocidad de motores, receptor de frecuencias, códigos de seguridad, etc.) volviéndolos extremadamente rápidos y convenientes.

Los pequeños empaques hacen perfectos a esta serie de microcontroladores para todas las aplicaciones con limitaciones de espacio.

Bajo costo, bajo consumo de energía, alto rendimiento, fácil de usar y flexibilidad en las entradas/salidas, hacen muy versátil al PIC16F8x en áreas donde no se habían considerado los microcontroladores (funciones de temporizador, comunicación serie, capturadores, comparadores, moduladores por ancho de pulso, y aplicaciones como procesador).

La medida de programación serial en el sistema (por medio de dos pines), ofrece flexibilidad en la optimización del producto después de haber sido ensamblado y probado.

Esta característica puede ser usada para agregar un número de serie a un producto, una calibración o reestablecimiento de datos o programar el dispositivo con la firma de software correspondiente antes de enviarlo.

El diagrama de pines del microcontrolador PIC16F84 se puede apreciar en la figura 2.1.1.1.

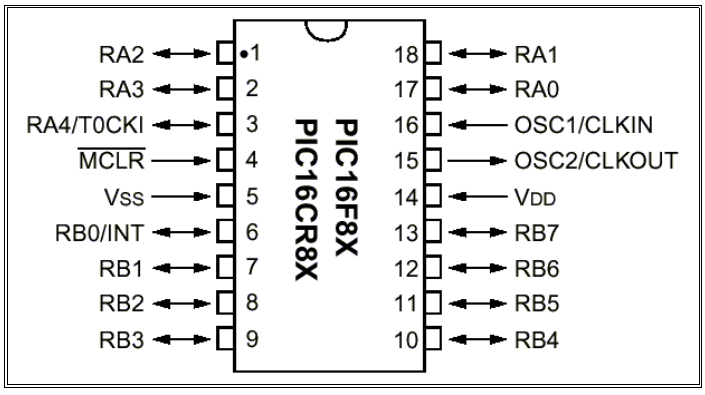


Figura 2.1.1.1. Diagrama de pines del PIC16F84

2.1.2 Características de la estructura RISC del CPU.

Las características de la estructura RISC del CPU son las siguientes:

- Solo 35 sencillas instrucciones para aprender.
- Todas las instrucciones son de un ciclo, excepto los saltos de programa que son de dos ciclos.
- Velocidad: 20MHz (0.2us por instrucción)
- Memoria de programa de 1024 palabras
- Memoria RAM de 68 bytes
- Memoria EEPROM de 64 Bytes
- Palabras de instrucción de 14 bits de ancho
- Bytes de datos de 8 bits de ancho
- 15 registros de función especial en el hardware
- Pila de hardware de 8 niveles de profundidad
- Modos de direccionamiento directo e indirecto
- Cuatro fuentes de interrupción:
  - Pin externo de RB0/INT
  - Temporizador TMR0 de sobreflujo
  - Interrupción en cambio PORTB <RB7:RB4>
  - Escritura completa de la memoria EEPROM

2.1.3 Características de los periféricos

- 13 pines de entrada / salida con control individual de dirección
- Fuente / sumidero de alta corriente para controlador directo de LED.
  - Sumidero de 25mA máximo por pin.
  - Fuente de 25mA máximo por pin.
  - Temporizador/ contador de 8 bits TMR0 con escala programable de 8 bits.

2.2 Arquitectura del PIC16F84A

El alto desempeño de la familia PIC16CXX puede ser atribuido a un número de características comúnmente encontradas en un microprocesador RISC. Los PIC16CXX incluyendo el PIC16F84 usan una arquitectura Harvard. Esta arquitectura tiene la memoria de programa y la de datos en forma separada. Así el dispositivo tiene un bus de memoria de programa y un bus de memoria de datos. La arquitectura Harvard presenta muchas ventajas sobre la tradicional arquitectura Von Neumann, donde el programa y los datos se sacan de la misma memoria (acceso sobre el mismo bus). Separando la memoria de programa y la memoria de datos permite instrucciones de un tamaño de palabra mayor de 8 bits. La memoria de programa tiene 1K posiciones de 14 bits cada una (1K x 14). La mayor parte de sus instrucciones se ejecutan en un ciclo de reloj excepto las instrucciones de salto. Otra aportación frecuente que aumenta el rendimiento del computador es el fomento del paralelismo implícito, que consiste en la segmentación del procesador (pipe-line), descomponiéndolo en etapas para poder procesar una instrucción diferente en cada una de ellas y trabajar con varias a la vez. El PIC16F84 contiene una ALU de 8 bits de trabajo. Un diagrama a bloques simplificado para el PIC16F84 se muestra en la figura 2.2.2.1.

2.2.1 ALU (Unidad Lógica Aritmética)

La ALU es una unidad aritmética de propósito general. Realiza operaciones aritméticas y booleanas entre registros. La ALU es de 8 bits de ancho y capaz de realizar suma, substracción y operaciones lógicas entre registro de propósito especifico, registro de propósito general y el registro de trabajo W. Dependiendo de la instrucción a ejecutar, la ALU puede afectar los valores de acarreo o “carry” (C), el acarreo de dígito o “digit carry” (DC), y Cero (Z) estos bits forman parte del registro ESTADO (STATUS) y son también llamados banderas o señalizadores.

2.2.2 Registro W

El registro de trabajo W es de 8 bits y es usado para las operaciones de la ALU. No es un registro que tenga una dirección especifica. Se accede a él mediante las instrucciones o mnemónicos que involucran la utilización de este registro.

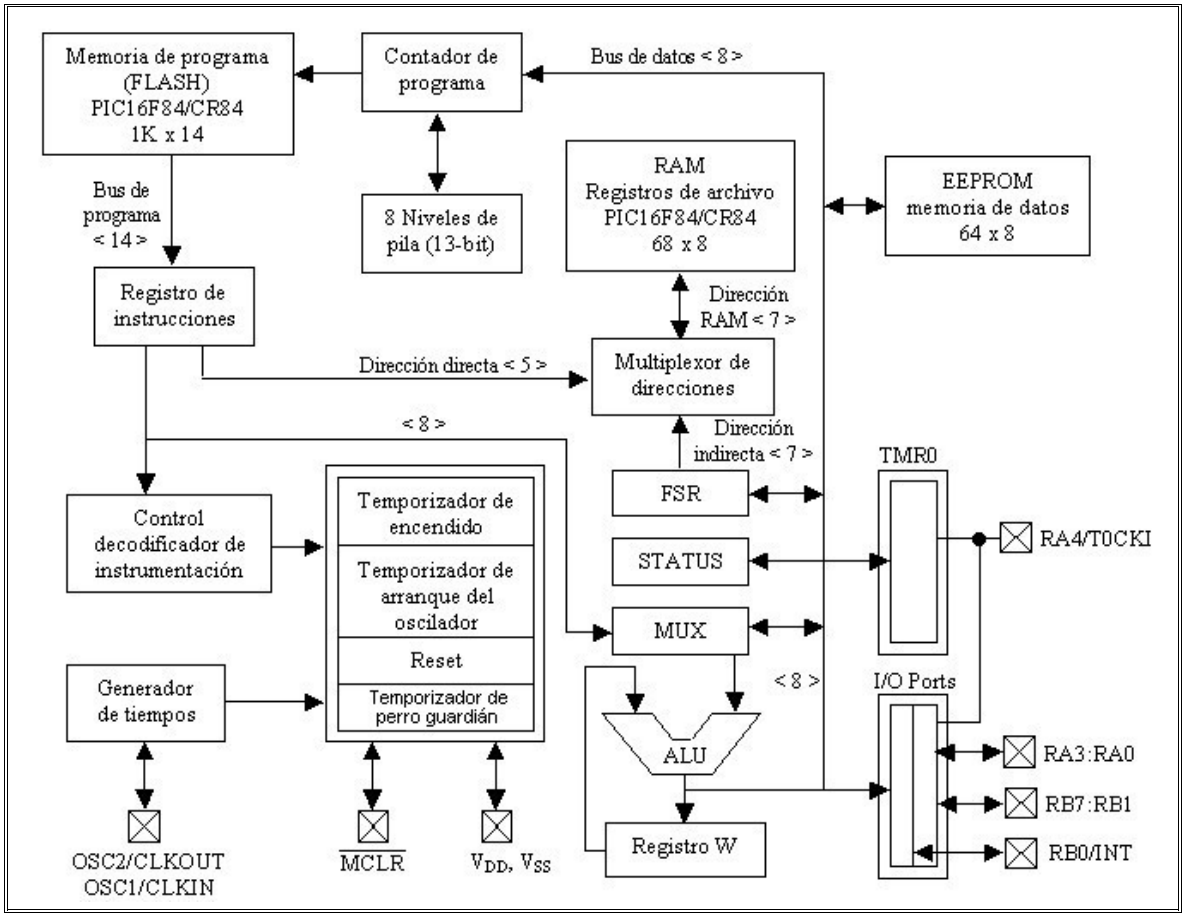


Figura 2.2.2.1. Arquitectura interna del PIC16F84A

2.2.3 Memoria de programa

La memoria de programa del PIC16F84 es de tipo Flash de un tamaño de 1K x 14. Por sus mejores prestaciones, la memoria Flash está sustituyendo a la memoria EEPROM para contener instrucciones. De esta forma, Microchip comercializa dos microcontroladores prácticamente iguales que sólo se diferencian en que la memoria de programa de uno de ellos es tipo EEPROM y la del otro tipo Flash. Se trata del PIC16C84 y el PIC16F84, respectivamente.

2.2.4 Frecuencia de funcionamiento

La frecuencia de trabajo del microcontrolador es un parámetro fundamental a la hora de establecer la velocidad en la ejecución de instrucciones y el consumo de energía.

Cuando un PIC16X8X funciona a 10 MHz, que es su máxima frecuencia, le corresponde un ciclo de instrucción de 400ns, puesto que cada instrucción tarda en ejecutarse cuatro periodos de reloj, o sea, 4 x 100ns = 400ns. Todas las instrucciones del PIC se realizan en un ciclo de instrucción, menos las de salto, que tardan el doble.

Los impulsos de reloj entran por la terminal OSC1/CLKIN y se dividen por 4 internamente, dando lugar a las señales Q1, Q2, Q3 y Q4, mostradas en la Figura 2.2.4.1. Durante un ciclo de instrucción, que comprende las 4 señales mencionadas, se desarrollan las siguientes operaciones.

Q1: Durante este pulso se incrementa el Contador de Programa.

Q4: Durante este pulso se busca el código de la instrucción en la memoria del programa y se carga en el registro de Instrucciones.

Q2-Q3: Durante la activación de estas dos señales se produce la decodificación y la ejecución de la instrucción.

Para conseguir ejecutar cada instrucción en un ciclo de instrucción (excepto las de salto, que tardan dos) se aplica la técnica de la segmentación o «Pipe-line», que consiste en realizar en paralelo las dos fases que comprende cada instrucción.

En realidad cada instrucción se ejecuta en dos ciclos: en el primero se lleva a cabo la fase de búsqueda del código de la instrucción en la memoria del programa y en el segundo se decodifica y se ejecuta (fase de ejecución). La estructura segmentada del procesador permite realizar al mismo tiempo la fase de ejecución de una instrucción y la de búsqueda de la siguiente. Cuando la instrucción ejecutada corresponde a un salto no se conoce cuál será la siguiente hasta que se complete, por eso en esta situación se sustituye la fase de búsqueda de la siguiente instrucción por un ciclo «vacío», originando que las instrucciones de salto tarden en realizarse dos ciclos de instrucción, como se puede apreciar en la figura 2.2.4.2.

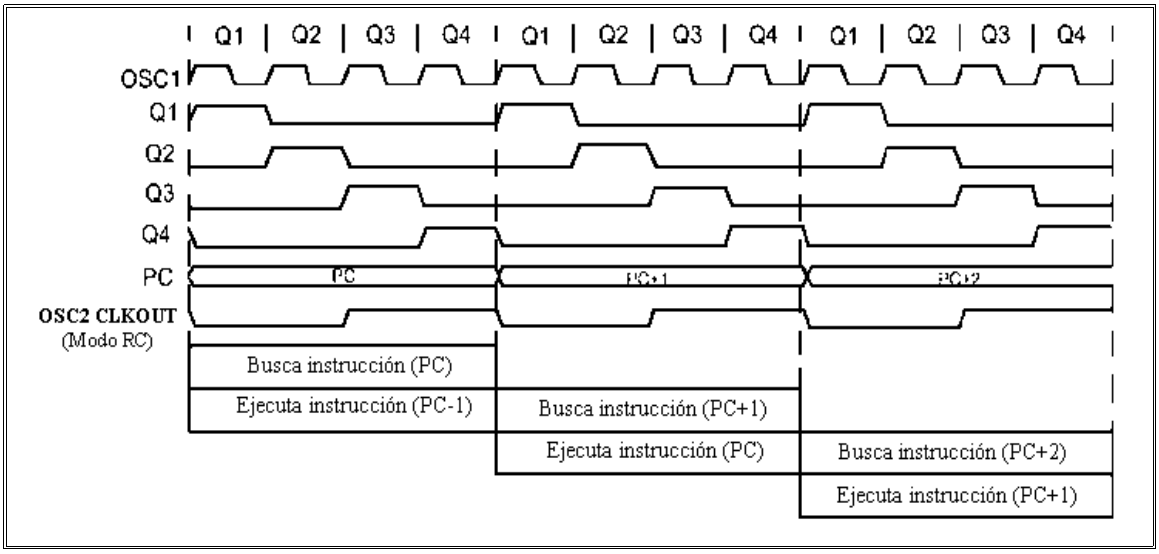


Figura 2.2.4.1 Los impulsos del reloj externo (OSC1) se dividen por 4 formando las señales Q1, Q2, Q3 y Q4, que configuran un ciclo de instrucción.

La técnica de la segmentación unida a la arquitectura Harvard del procesador permite al PIC16F84 supera la velocidad de sus competidores directos. Así, por ejemplo, es 1.54 veces más rápido que el microcontrolador de motorola 68HC05 cuando ambos funcionan a la misma frecuencia de 4 MHz.

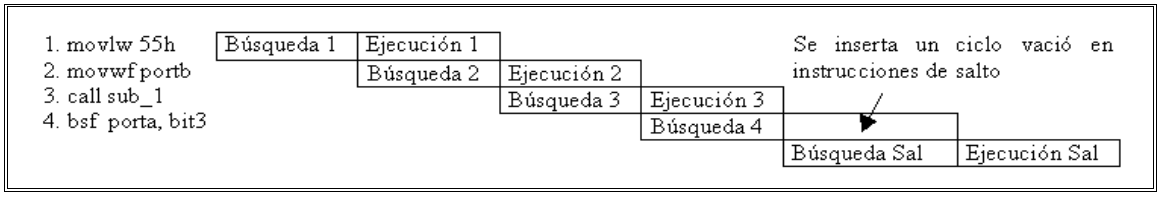


Figura 2.2.4.2. La segmentación permite traslapar en el mismo ciclo la fase de ejecución de una instrucción y la de búsqueda de la siguiente, excepto en las instrucciones de salto

2.3 Organización de memoria

Hay dos clases de memoria en el PIC16F84. Estas son la memoria de programa y la memoria de datos. Cada memoria tiene su propio bus, para que el acceso a cada memoria ocurra durante el mismo ciclo de oscilación.

La memoria de datos puede dividirse en dos tipos de memoria, memoria RAM de propósito general y en registros de función especial (SFR). El área de memoria de datos también contiene los datos de memoria EEPROM. Esta memoria no se traza directamente en la memoria de datos, pero se traza indirectamente. Es decir, un indicador de dirección indirecto especifica la dirección de la memoria de datos EEPROM al leer/escribir. Los 64 bytes de la memoria de datos de la EEPROM tienen un rango de dirección de 0h-3Fh.

2.3.1 Organización de la memoria de programa

La memoria de programa del PIC16F84 es una memoria de 1 Kbyte de longitud y con palabras de 14 bits. Como es del tipo Flash se puede programar y borrar eléctricamente, lo que facilita el desarrollo de los programas y la experimentación. En ella se graba o almacena, el programa o códigos que el microcontrolador debe ejecutar. Dado que el PIC 16F84 tiene un contador de programa de 13 bits, tiene una capacidad de direccionamiento de 8K x 14, pero solamente tiene implementado el primer 1K x 14 (0000h hasta 03FFh), tal como se puede apreciar en la figura 2.3.1.1. Si se direccionan posiciones de memoria superiores a 3FFh se causará un solapamiento con el espacio del primer 1Kbyte (1K x 14).

Para los PIC16F84 y PIC16CR84, solo el primer 1K x 14 (0000h-03FFh) de memoria de programa esta físicamente implementado.

El vector de reset está en la posición 0000h y el vector de interrupción esta en la posición 0004h.

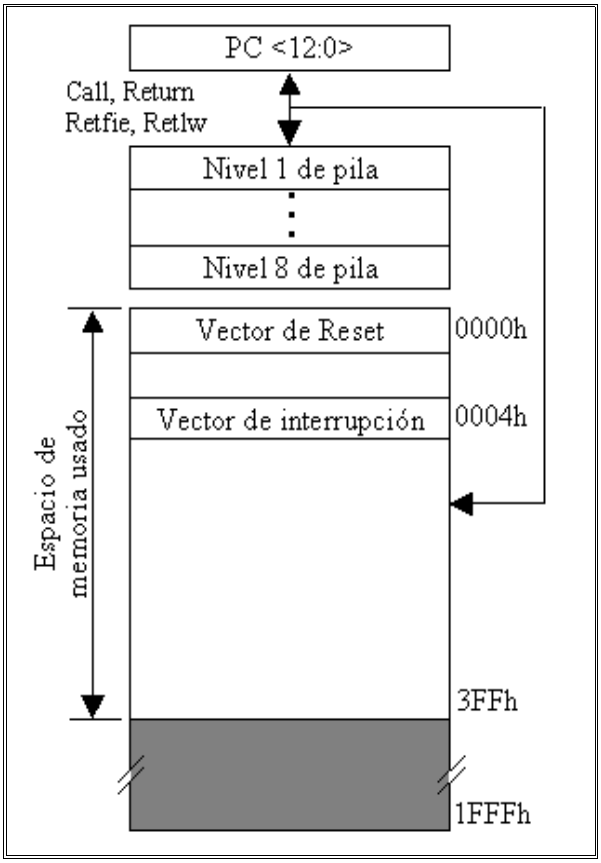


Figura 2.3.1.1. Organización de la memoria de programa.

2.3.1.1 Vector de reset

Cuando ocurre un reset en el microcontrolador, el contador de programa se pone en ceros (0000h). Por esta razón, en la primera dirección del programa se debe escribir todo lo relacionado con la inicialización del mismo.

2.3.1.2 Vector de interrupción

Cuando el microcontrolador recibe una señal de interrupción, el contador de programa apunta a la dirección 0004h de la memoria de programa, por eso allí se debe escribir toda la programación necesaria para atender dicha interrupción.

2.3.2 Organización de la memoria de datos

El PIC16F84 puede direccionar 128 posiciones de memoria RAM, pero solo tiene implementados físicamente las primeros 80 (00h-4Fh en hexadecimal). De estos las primeras 12 son registros que cumplen un propósito especial (SFR) en el control del microcontrolador y las 68 siguientes son registros de uso general (GPR) que se pueden usar para guardar los datos temporales de la tarea que se está ejecutando, justo como se muestra en la figura 2.3.2.1.

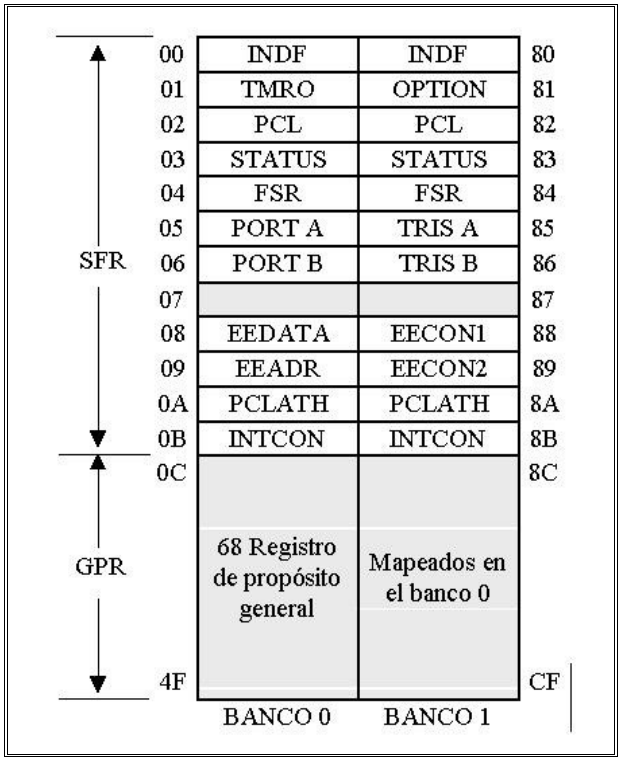


Figura 2.3.2.1. Organización de la memoria RAM del PIC16F84A

SFR: Registros de propósito especifico.

GPR: Registros de propósito general

Los registros están organizados como dos arreglos (bancos) de 128 posiciones de 8 bits cada una (128 x 8); todas las posiciones se pueden acceder directa o indirectamente

(esta última a través del registro selector FSR). Para seleccionar que banco de registros se trabaja en un momento determinado se utiliza el bit RP0 del registro ESTADO (STATUS).

Los registros del SFR (Registros de función especial) se localizan en la memoria de programa de la posición 00h a la posición 0Bh, mientras que, los registros GPR (registros de propósito general) se localizan de la posición 0Ch a la posición 4Fh.

2.3.2.1 Registros de propósito general (GPR)

También son conocidos como archivos de registro de propósito general (GPR). Todos los dispositivos PIC's tienen alguna área de Registros de Propósito General. Cada GPR es de un ancho de 8 bits y es accedido directamente o indirectamente a través del FSR

Los registro de propósito general (GPR) se utilizan para poder guardar alguna información o datos de manera temporal, pero como se trata de una memoria de tipo RAM estática, estos datos se perderán cuando se guarde algún nuevo dato o cuando se corte la alimentación o al momento de un reset.

2.3.2.2 Registro de propósito especial (SFR)

Los Registros de la Función Especiales son usados por el CPU y periféricos para controlar el funcionamiento del dispositivo. Estos registros son del tipo RAM estática.

Los registros de función especial (SFR) son registros que ya tienen una función determinada para el microcontrolador, por este motivo el usuario debe tener cuidado al utilizarlos, mediante la utilización de estos registros el usuario determinará el funcionamiento del microcontrolador y los periféricos del mismo.

En la tabla 2.3.2.2.1 se muestra la organización de los registros SFR del PIC16F84A. Se indica la dirección , el banco, el nombre de registro y cada uno de los bits.

Tabla 2.3.2.2.1. Organización detallada de los registros del PIC16F84A.

Dir	Nombre	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Banco 0									
00h	INDF	Usa el contenido de FSR para el direccionamiento indirecto							
01h	TMR0	Temporizador/contador en tiempo real de 8 bits							
02h	PCL	Contador de programa de 8 bits (PC)							
03h	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
04h	FSR	Direccionamiento indirecto de memoria con INDF							
05h	PORTA	—	—	—	RA4/T0CK	RA3	RA2	RA1	RA0
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT
07h		No implementado. Se lee como “0”							
08h	EEDATA	Registro de datos de EEPROM							
09h	EEDR	Registro de direcciones de EEPROM							
0Ah	PCLATH	—	—	—	5 bits superiores del PC				
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
Banco 1									
80h	INDF	Usa el contenido de FSR para el direccionamiento indirecto							
81h	OPTION	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
82h	PCL	Direccionamiento indirecto de memoria							
83h	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
84h	FSR	Direccionamiento indirecto de memoria							
85h	TRISA	—	—	—	Dato de la dirección del registro PORTA				
86h	TRISB	Dato de la dirección del registro PORTB							
87h		No implementado. Se lee como “0”							
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD
89h	EECON2	Registro de control EEPROM (no es un registro físico)							
0Ah	PCLATH	—	—	—	Buffer de escritura para de 5 bits superiores del PC				
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

2.3.2.3 Registro ESTADO (STATUS) 03h

El registro STATUS se ubica en la posición 03h de la memoria de datos del PIC16F84. Contiene el estado aritmético de la ALU, la causa del reset y los bits de preselección de los bancos de memoria de datos. Como con cualquier registro, el registro de STATUS puede estar destinado para cualquier instrucción.

Los bits Z, DC y C del registro ESTADO, son señalizadores que indican una operación realizada por la ALU, por ejemplo si el resultado de una operación es cero, entonces el señalizador de Z se pondrá en alto (1).

Los bits 5 y 6 (RP0 y RP1) son los bits de selección de banco para el direccionamiento directo de la memoria de datos; solamente RP0 se usa en los PIC16F84. RP1 se puede utilizar como un bit de propósito general de lectura/escritura.

Los bits TO y PD no se pueden modificar por un proceso de escritura, ya que solo muestran la condición por la cual se ocasiona el último reset.

Por ejemplo: CLRF STATUS limpiará los tres bits superiores del registro ESTADO y pondrá en uno el bit Z. Esto dejara el registro de ESTADO como 000u u1uu (donde u=inalterado).

Sólo las instrucciones BCF, BSF, SWAPF MOVLW, MOVF y MOVWF pueden usarse para alterar el registro ESTADO.

Los bits IRP y RP1 (ESTADO <7:6>) no son usados por los PIC16F8X y deben ser programados como cero. El uso de estos bits como bits de propósito general no se recomienda, debido a que esto puede afectar la compatibilidad con futuros productos.

El registro ESTADO (STATUS) se ubica en la dirección 03H del banco 0 y en la dirección 83h del banco 1 de la memoria de datos en la SFR (registros de propósito especifico).

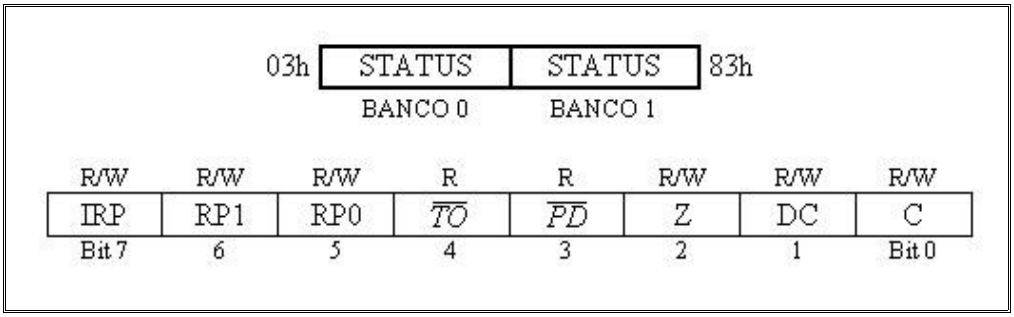


Figura 2.3.2.3.1. Dirección y mapeo de bits del registro ESTADO (STATUS)

A continuación se describen los bits del registro ESTADO que se muestran en la figura 2.3.2.3.1.

IRP: bit de selección de banco (usado para direccionamiento indirecto).
0 = Banco 0, 1 (00h - FFh)
1 = Banco 2, 3 (100h - 1FFh)
Este bit no se utiliza efectivamente en el PIC16F84, debe programarse como 0.

RP1: RP0: Selector de banco (usado para direccionamiento directo).
00 = Banco 0 (00h - 7Fh)
01 = Banco 1 (80h - FFh)
10 = Banco 2 (100h - 17Fh)
11 = Banco 3 (180h - 1FFh)
Solamente RP0 se utiliza en el PIC16F84, RP1 se debe programar como cero.

$\overline{TO}$ *: Time Out* (Bit de finalización del temporizador)
1 = después de energizarse, de instrucción CLRWDT, o instrucción SLEEP
0 = Cuando el circuito de vigilancia Watchdog finaliza la temporización

$\overline{PD}$ *: Power-down* (Bit de bajo consumo)
1 = Se pone automáticamente en 1 después de la conexión de la alimentación al microcontrolador o al ejecutar la instrucción clrwdt.
0 = Se pone automáticamente en 0 mediante la ejecución de la instrucción *sleep*.

Z: Zero (Bit de cero)
1 = El resultado de una operación lógica o aritmética es cero.
0 = El resultado de una operación lógica o aritmética no es cero.

DC: Digit Carry/Borrow (Bit de acarreo de dígito)
1 = En operaciones aritméticas se activa cuando hay acarreo entre el bit 3 y 4
0 = No hay acarreo entre el bit 3 y 4

C: Carry (bit de acarreo)
1 = Indica que se ha producido un acarreo en el bit de más peso del resultado al ejecutar las instrucciones de ADDWF y ADDLW.
0= No se ha producido acarreo.

Registro OPTION (dirección 81h)

El registro OPTION es un registro legible y escribible que contiene varios bits para configurar: los temporizadores TMR0/WDT, las interrupciones INT y TMR0 y la activación de las resistencias de pull-ups del puerto B, además de asignar el valor de preescala con la que va actuar el Temporizador asignado .

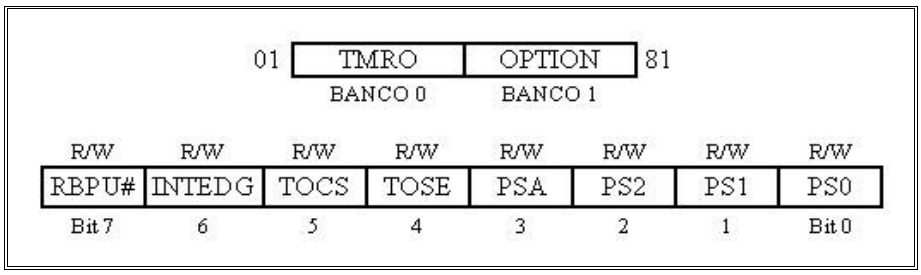


Figura 2.3.2.3.2. Dirección y mapeo de bits del registro OPTION y TMR0

A continuación se describen los bits del registro OPTION/TMR0 que se muestran en la figura 2.3.2.3.2.

RBPU: Resistencia Pull-Up Puerto B

1 = Desactivadas

0 = Activadas

INTEDG: Flanco activo interrupción externa

1 = Flanco ascendente

0 = Flanco descendente

TOCS: Tipo de reloj para el TMR0

1 = Pulsos introducidos a través de T0CKl (contador)

0 = Pulsos de reloj interno Fosc/4 (temporizador)

TOSE: Tipo de flanco en TOCKl

1 = Incremento de TMR0 cada flanco descendente

0 = Incremento de TMR0 cada flanco ascendente

PSA: Asignación del Divisor de frecuencia

1 = El Divisor de frecuencia se le asigna al WDT

0 = El Divisor de frecuencia se le asigna al TMR0

PS2-PS0: Prescaler Value (Valor con el que actúa el Divisor de frecuencia que se muestra en la tabla 2.3.2.3.1)

Tabla 2.3.2.3.1 Valor de preescala asignado al divisor de frecuencia.

PS2	*PS1*	*PS0*	*Divisor del TMR0*	*Divisor del WDT*
0	0	0	1 : 2	1 : 1
0	0	1	1 : 4	1 : 2
0	1	0	1 : 8	1 : 4
0	1	1	1 : 16	1 : 8
1	0	0	1 : 32	1 : 16
1	0	1	1 : 64	1 : 32
1	1	0	1 : 128	1 : 64
1	1	1	1 : 256	1 : 128

2.3.2.4 Registro INTCON

El registro INTCON se localiza en la dirección 0Bh y en la dirección 8Bh de la memoria de datos de la SFR, es un registro de lectura / escritura que contiene los bits de habilitación de las fuentes de interrupción. Las banderas de interrupción se ponen en alto cuando una interrupción ocurre, sin tener en cuenta el estado de su correspondiente bit de habilitación o el permiso global de interrupciones, GIE (INTCON<7>).

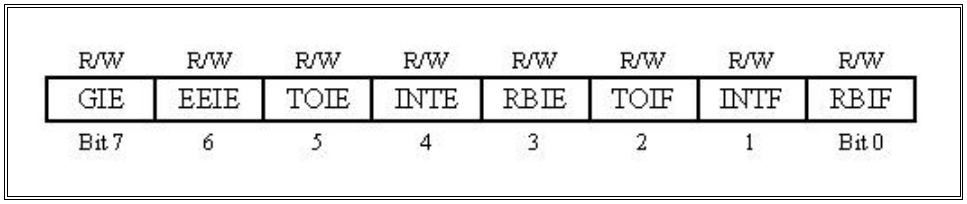


Figura 2.3.2.4.1. Mapeo de bits del registro INTCON.

A continuación se describen los bits del registro INTCON que se muestran en la figura 2.3.2.4.1.

GIE: Habilitador global de interrupciones (Global Interrupt Enable)

1 = Habilita todas las interrupciones

0 = Deshabilita todas las interrupciones

EEIE: Habilitación de interrupción por escritura de la EEPROM (EEPROM Write Interrupt Enable)

1 = Habilita la interrupción EEIE

0 = Deshabilita la interrupción EEIE

TOIE: Habilitación de la interrupción del temporizador TMRO (TMR0 Interrupt Enable)

1 = Habilita la interrupción TOIE

0 = Deshabilita la interrupción TOIE

INTE: Habilitación de la interrupción INT (INT Interrupt Enable)

1 = Habilita la interrupción INTE

0 = Deshabilita la interrupción INTE

RBIE: Habilitación de la interrupción RBIF (RBIF Interrupt Enable)

1 = Habilita la interrupción RBIE

0 = Deshabilita la interrupción RBIE

TOIF: Bandera de interrupción por sobreflujo del TMR0 (TMR0 Overflow Interrupt Flag)

1 = Ha ocurrido un sobreflujo en el TMR0 (solo es limpiado por software)

0 = No hay sobreflujo en el TMR0

INTF: Bandera de interrupción INT (RB0/INT Interrupt Flag)

1 = Ha ocurrido la interrupción externa RB0/INT

0 = No hay interrupción externa por RB0/INT

RBIF: Bandera de interrupción por cambio en el puerto B (RBPort Change Interrupt Flag)

1 = Indica el cambio de estado de los bits RB7:RB4 del puerto B (se limpia por software)

0 = No hay cambios de estado en los bits RB7:RB4 del puerto B.

2.3.2.5 Contador de programa (PCL y PCLATH)

El contador de programa es un registro que nos indica la dirección de la instrucción próxima a ejecutar, este registro se puede leer y escribir, dicho registro se localiza en la posición 02h del banco 0 y en la 82 del banco 1 de la memoria de datos (RAM).

Al igual que todos los registros específicos que controlan la actividad del procesador, el Contador de Programa está implementado sobre un par de posiciones de la memoria RAM. Cuando se escribe el Contador de Programa como resultado de una operación de la ALU, los 8 bits de menos peso del PC residen en el registro PCL, que se encuentra también en la posición 2 de los dos bancos de memoria de datos.

Los bits de más peso, PC<12:8>, residen en los 5 bits de menos peso del registro PCLATH, que ocupa la posición 0Ah de los dos bancos de la memoria RAM. En las instrucciones GOTO y CALL de la gama media, los 11 bits de menos peso del PC provienen del código de la instrucción y los otros dos de los bits PCLATH <4:3>, como se muestra en la figura 2.3.2.5.1.

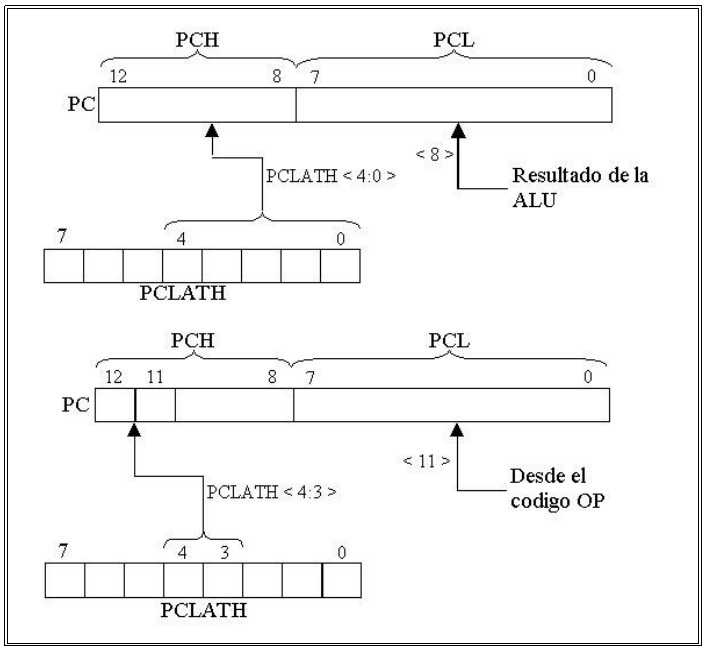


Figura 2.3.2.5.1 Carga del contador de programa

En la parte superior de la figura 2.3.2.5.1 se muestra cómo se carga el PC cuando una instrucción deposita en él el resultado que se obtiene de la ALU: en la parte inferior, se indica la carga del PC en las instrucciones GOTO y CALL.

Con los 11 bits que se cargan en el PC desde el código de las instrucciones GOTO y CALL se puede direccionar una pagina de 2 K de la memoria. Los bits restantes PC <12:11> tienen la misión de apuntar una de las 4 páginas del mapa de memoria y en los modelos de PIC que alcanzan ese tamaño, proceden de PCLATH <4:3>.

La pila es una zona aislada de las memorias de instrucciones y datos. Tiene una estructura LIFO, en la que el último valor guardado es el primero que sale. Tiene 8 niveles de profundidad cada uno con 13 bits. Funciona como un «buffer» circular, de manera que el valor que se obtiene al realizar el noveno «desempilado» (pop) es igual al que se obtuvo en el primero.

La instrucción CALL y las interrupciones originan la carga del contenido del PC en el nivel superior o «cima» de la pila. El contenido del nivel superior se saca de la Pila al ejecutarse las instrucciones RETURN, RETLW y RETFIE. El contenido del registro PCLATH no es afectado por la entrada o salida de información de la pila.

A diferencia de los PIC de primera generación, el 16F84 ante una condición de reset inicia el contador de programa con todos sus bits en “cero”. Durante la ejecución normal del programa, y dado que todas las instrucciones ocupan sólo una posición de memoria, el contador se incrementa en uno con cada instrucción, a menos que se trate de alguna instrucción de salto.

2.3.3 Direccionamiento de la memoria de datos RAM

La memoria de datos RAM de los PIC de la gama media está organizada en cuatro bancos de 128 posiciones con 8 Bits de longitud cada una. En el caso concreto del PIC16F84 sólo están implementadas las 80 primeras posiciones (00h - 4Fh) de los bancos 0 y 1, como se puede apreciar en la figura 2.3.3.1.

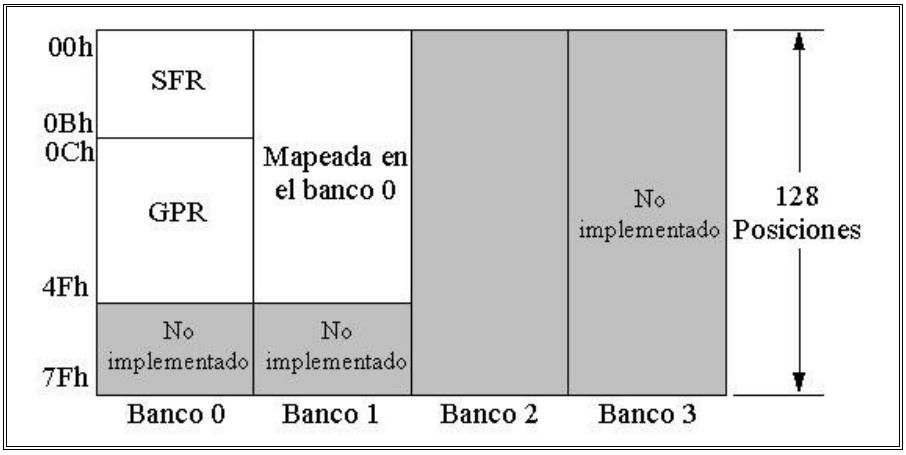


Figura 2.3.3.1. Mapeo general de bancos de los dispositivos PIC.

Para direccionar una posición de la memoria RAM en un PIC hay que especificar, en primer lugar, el banco en el que se encuentra y, después, la dirección relativa en el mismo. Para seleccionar el banco existen dos bits en el registro STATUS del área de la RAM, que se llaman RP0 y RP1, ocupando las posiciones 5 y 6 de dicho registro. La forma de direccionar la RAM, es decir, el modo de direccionamiento, puede tomar dos alternativas: direccionamiento directo y direccionamiento indirecto.

2.3.3.1 Direccionamiento directo

Es la manera de apuntar a la posición de la memoria RAM donde se encuentra el dato que se busca, se empieza escribiendo en los bits RP0 y RP1 del registro STATUS el código correspondiente al banco. La dirección relativa dentro del banco está incluida en el código de la propia instrucción, pues de los 14 bits de longitud, los 7 de menor peso se reservan para esta labor.

Para cargar el código del banco en los bits RP0 y RP1, existen, un par de instrucciones muy eficaces, capaces de poner a 1 ó 0 un bit de cualquier registro.

En el PIC16F84 el direccionamiento directo es aún más sencillo, pues la memoria RAM sólo está implementada sobre los dos primeros bancos, el banco 0 y el banco 1. Esto supone que con un sólo bit se elija el banco, concretamente el bit usado es el RP0, siendo indiferente el estado que tenga el bit RP1. Si RP0 = 0, se accede al banco 0 y si RP0 = 1 se accede al banco 1, como se muestra en la figura 2.3.3.1.1.

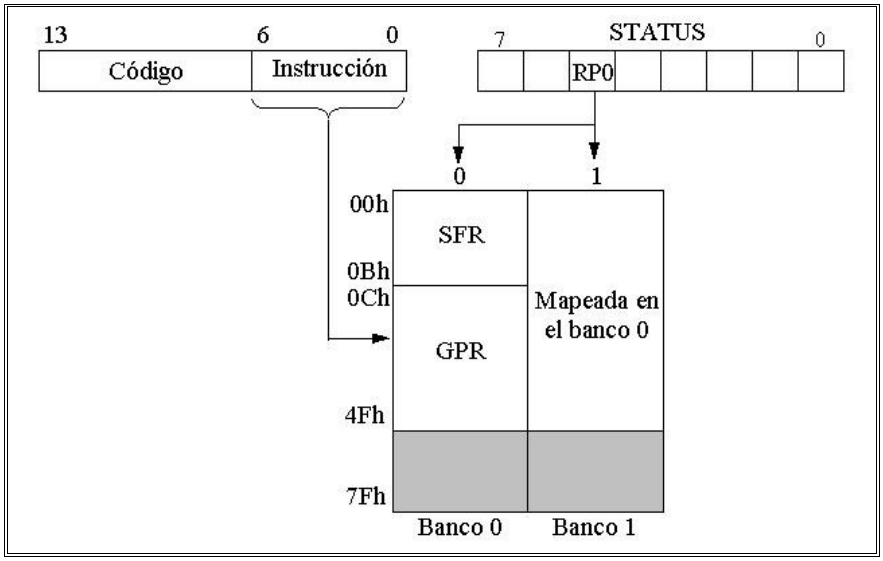


Figura 2.3.3.1.1. Direccionamiento directo en el PIC16F84

2.3.3.2 Direccionamiento indirecto

En los microcontroladores PIC las instrucciones que manejan el direccionamiento indirecto emplean el registro INDF, que ocupa la dirección 00H de los dos bancos del área SFR. En realidad el registro INDF no está implementado físicamente, por lo tanto, cuando se hace referencia a él, en realidad se accede a otro registro específico del área SFR que se llama FSR. En los 7 bits de menos peso del FSR se encuentra el valor de la dirección relativa del registro buscado en el banco seleccionado.

La instrucción con modo de direccionamiento indirecto utiliza como registro operando el INDF, pero el PIC accede al contenido del registro FSR, que ocupa, por duplicado, la dirección 04h de los dos bancos de la RAM.

No sólo es diferente la forma de obtener los 7 bits de la dirección, sino que en el modo indirecto también cambia el modo de especificar el banco. En el direccionamiento directo se usaban los bit RP0 y RP1 para elegir el banco; en el modo indirecto se usa el bit 7, de más peso, del registro FSR (recuerde que los 7 bits de menos peso contenían la dirección relativa). Como en el PIC16F84 sólo están disponibles los bancos 0 y 1, el bit IRP del registro STATUS siempre valdrá 0, justo como se observa en la figura 2.3.3.2.1.

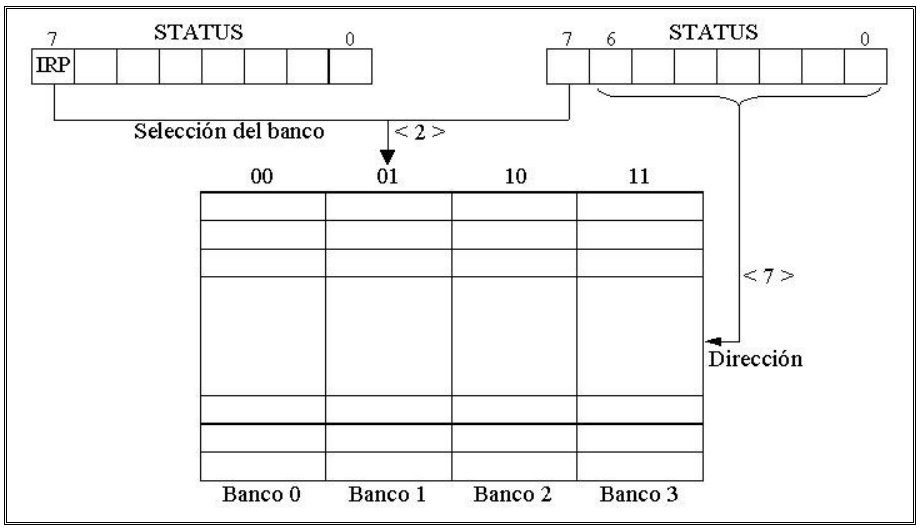


Figura 2.3.3.2.1. Direccionamiento indirecto en los dispositivos PIC.

Si por ejemplo queremos apuntar de forma indirecta la dirección 12h del banco 1 en un PIC16F84, comenzaremos cargando en FSR el valor 12h (0x12), para los 7 bits de menos peso y poniendo a 1 el bit 7, de más peso, de dicho registro, para elegir el banco 1.

```
movlw   0x12        ;Se carga w con 12h
bsf     w,7         ;Se carga a 1 el bit 7 de w
movwf   FSR         ;El valor de w se carga en FSR
```

Otro ejemplo de direccionamiento indirecto es el siguiente

- Se tiene cargado el registro 05h con el valor de 10h
- Se tiene cargado el registro 06h con el valor de 0Ah
- Cargamos el valor 05h en el registro FSR (FSR = 5)
- Una lectura del registro INDF regresará un valor de 10h
- Se incrementa el valor del registro FSR en uno (FSR = 6)
- Ahora una lectura del registro INDF regresara un valor de 0Ah.

El direccionamiento indirecto es muy utilizado en el manejo de tablas, pues nos facilita la lectura y la escritura de la misma, dicho procedimiento consiste en cargar el registro FSR con una valor inicial que nos va a servir de offset para después incrementar o

decrementar el mismo y de esta manera acceder a una posición específica de memoria. dicho procedimiento se puede apreciar en el siguiente ejemplo.

Ejemplo: el siguiente programa limpiará las posiciones 20h – 2Ah de la memoria RAM.

```
          movlw     0x20        ;Inicializa el punto
          movwf     FSR         ;Se carga  FSR con 0x20
Next      clrf      INDF        ;Limpia el registro INDF
          incf      FSR         ;Incrementa el punto
          btfss     FSR,4       ;¿El bit 4 del registro FSR = 1?
          goto      NEXT        ;No, limpia la siguiente posición (salto a NEXT)
Continua                        ;Si, continua el programa
```

2.3.4 La pila

Una pila o stack es un conjunto de registros que guardan información de una forma determinada, ya que tanto la carga como la descarga es especial. En el caso del PIC16F84 la pila está formada por 8 registros de 13 bits cada uno y dotados de una estructura de carga/descarga tipo LIFO, que significa "último en entrar, primero en salir". Cuando se carga información a la pila siempre se introduce por el nivel 1. Si anteriormente había información en el nivel 1, se traslada al nivel 2 y lo de éste al nivel 3 y así sucesivamente. Es decir, cuando se carga información en el nivel 1 lo que había almacenado en cada nivel se desplaza al siguiente, dicho proceso se puede apreciar en la figura 2.3.4.1.

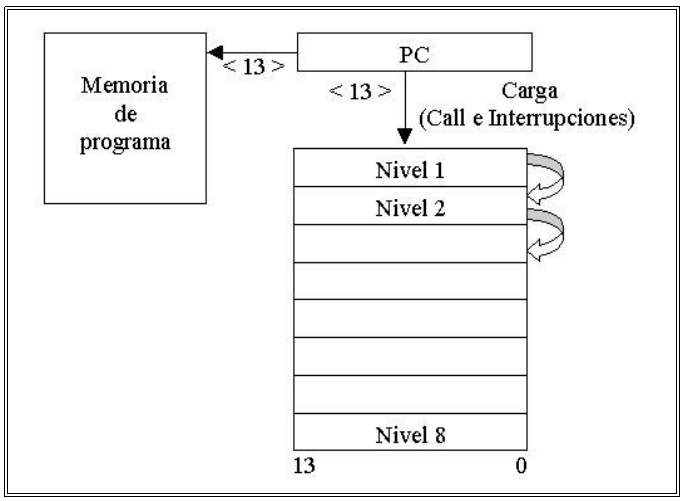


Figura 2.3.4.1. La instrucción CALL y la interrupción, provocan la carga automática del contenido del PC en la pila.

Si la pila esta llena y contiene información en los ocho niveles, la carga de un nuevo dato produce el desplazamiento de la información de cada nivel al siguiente, perdiéndose la que existía en el nivel 8. Para recuperar la información de la pila el desplazamiento de la información se realiza al revés. Sale la que está ocupando el nivel 1 y a éste se le carga con lo del nivel 2. La información se desplaza al nivel anterior y el dato que sale desde el nivel 1 habrá sido el último que se había cargado, dicho proceso se muestra en la figura 2.3.4.2.

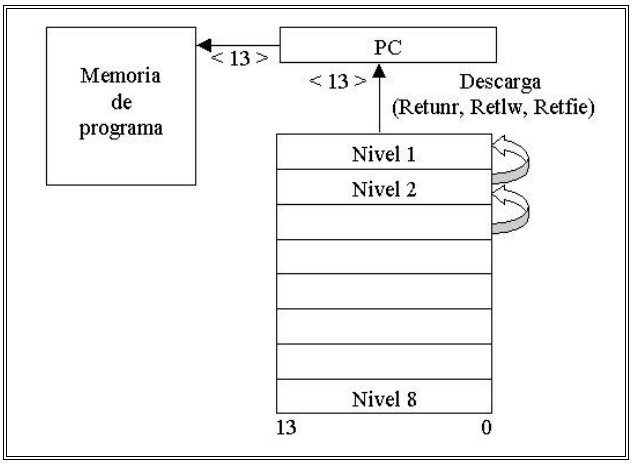


Figura 2.3.4.2. Las instrucciones de RETORNO de subrutina o de interrupción descargan automáticamente el contenido del nivel 1 de la pila sobre el PC.

La pila del PIC funciona automáticamente y se halla relacionada directamente con el contenido del contador de programa (PC). La pila se carga en su nivel 1 con el contenido del PC. Cuando la pila descarga la información del nivel 1 pasa al PC. Cada vez que se ejecuta una instrucción Call o que se produce una interrupción, es necesario salvar el contenido del PC para luego saber retornar al punto donde se salió en el programa principal. La ejecución de una instrucción Call o una interrupción producen el traslado automático del contenido del PC al nivel 1 de la Pila.

La última instrucción de una subrutina o de una rutina de interrupción es la de retorno (Return), que al ejecutarse produce la descarga automática del nivel 1 de la pila

sobre el PC, devolviendo el flujo de control al programa principal en el punto que se abandonó.

El programador debe tener en cuenta que no hay señalizador que indique cuando está la pila llena, siendo su responsabilidad evitar el rebosamiento y la pérdida de información.

2.4 Los puertos de entrada/salida

Los PIC16X8X sólo disponen de dos Puertos de E/S. El Puerto A posee 5 líneas RA0-RA4, y una de ellas soporta dos funciones multiplexadas. Se trata de la RA4/T0CK1, que puede actuar como línea de E/S o como terminal por la que se reciben los impulsos que debe contar TMR0. El Puerto B tiene 8 líneas, RB0-RB7, y también tiene una con funciones multiplexadas, la RB0/INT, que además de línea típica de E/S, también sirve como terminal por la que se reciben los impulsos externos que provocan una interrupción.

Cada línea de E/S puede configurarse independientemente como entrada o como salida, según se ponga a 1 o a 0, respectivamente el bit asociado del registro de configuración de cada Puerto (TRISA y TRISB). Se llaman PUERTO A y PUERTO B a los registros que guardan la información que entra o sale por el Puerto, y ocupan las direcciones 5 y 6 del banco 0 de la memoria de datos. Los registros de configuración TRISA y TRISB ocupan las mismas direcciones pero en el banco 1, como se aprecia en la figura 2.4.1.

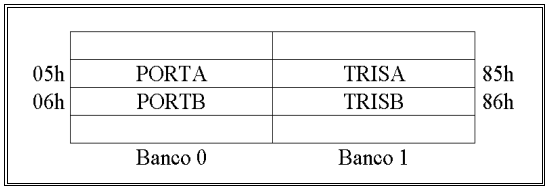


Figura 2.4.1. Mapeo de direcciones de los puertos

La información que entra o sale por los puertos reside en las posiciones 5 y 6 del banco 0, mientras que TRISA y TRISB, que son los registros de configuración, ocupan las direcciones 5 y 6 del banco 1.

2.4.1 Consideraciones de los puertos

Como el PIC16F84 es 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, como se muestra en la figura 2.4.1.1.

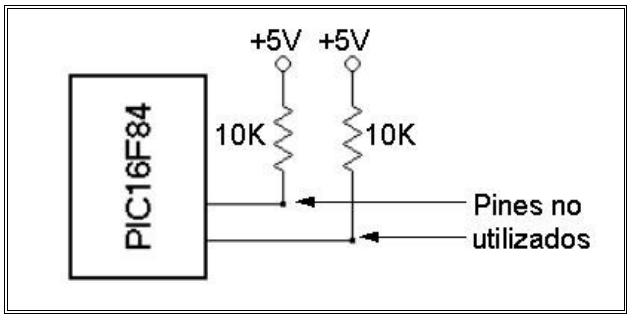


Figura 2.4.1.1. Los puertos no utilizados se deben conectar a la fuente

La máxima capacidad de corriente de cada uno de los pines de los puertos en modo sumidero (sink) es de 25mA y en modo fuente (source) es de 20mA. La máxima capacidad de corriente total de los puertos se muestra en la figura 2.4.1.2.

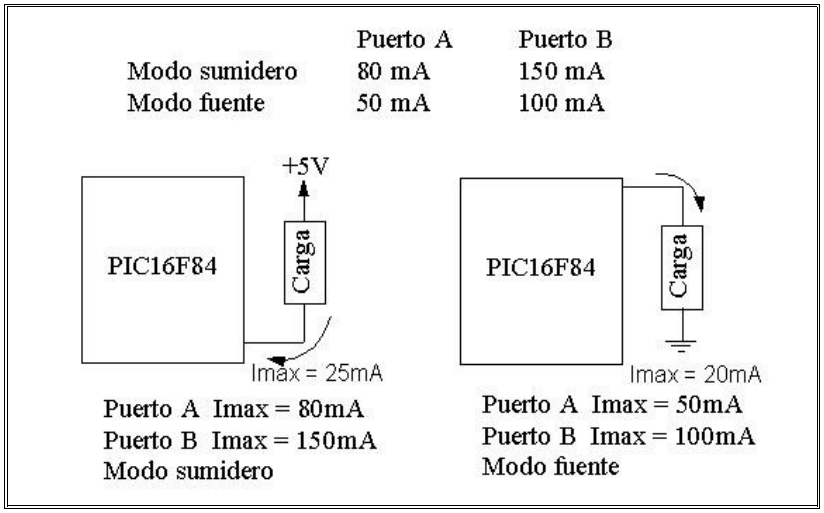


Figura 2.4.1.2. Capacidad máxima de corriente que soportan los puertos.

El consumo de corriente del microcontrolador para su funcionamiento depende del voltaje de operación, la frecuencia y de las cargas que tengan sus pines. Para un reloj de 4 MHz el consumo es de aproximadamente 2mA; aunque este se puede reducir a 40 microamperios cuando se está en el modo sleep (en este modo el microcontrolador se detiene y disminuye el consumo de potencia).

2.4.2 Puerto A (Registros PORTA y TRISA)

Las líneas RA3-RA0 admiten niveles de entrada TTL y de salida CMOS. La línea RA4/TOCK1 dispone de un circuito Disparador Schmith (Schmith Trigger) que proporciona una buena inmunidad al ruido y la salida tiene drenador abierto. RA4 multiplexa su función de E/S con la de entrada de impulsos externos para el TMR0.

En la figura 2.4.2.1 se muestra el registro TRISA, el cual se encarga de configurar las líneas del Puerto A como entradas si están a 1 y como salidas si están a 0. En el circuito de la figura 2.4.2.2 se muestra la adaptación de los pines RA3-RA0 a las señales del procesador.

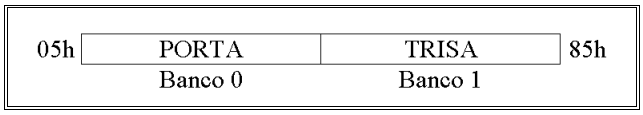


Figura 2.4.2.1. Ubicación de PORTA y TRISA en los bancos del PIC

Cuando se lee una línea de él Puerto A se toma el nivel lógico que tiene en ese momento. Las líneas cuando actúan como salidas están «sujetadas», lo que significa que sus pines sacan el nivel lógico que se haya cargado por última vez en el registro PUERTO A. La escritura de un Puerto implica la operación «lectura/modificación/escritura». Primero se lee el Puerto, luego se modifica el valor y finalmente se escribe en el «latch» de salida.

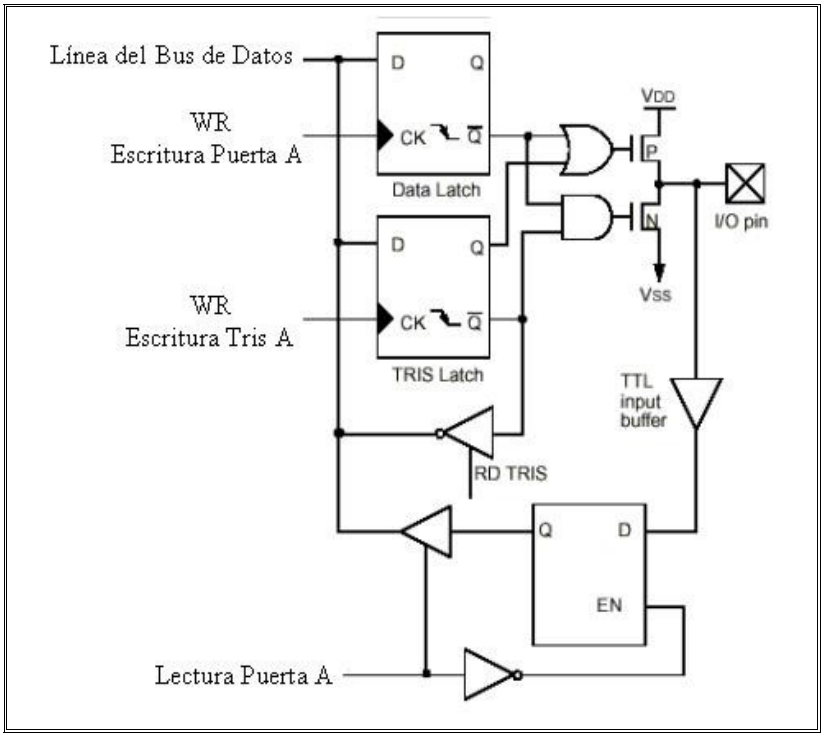


Figura 2.4.2.2. Diagrama de conexiones de los pines RA3-RA0 a las señales del procesador

De la Figura 2.4.2.2 se desprende que cuando se saca un nivel lógico por una línea del Puerto A, primero se deposita en la línea correspondiente del bus interno de datos y se activa la señal WRITE, lo que origina el almacenamiento de dicho nivel en el Flip-Flop de datos. En esta situación, el FF de configuración debería contener un 0 para que actuase como salida. Con estos valores él puerto OR tendría un 0 en su salida y la AND también. Estos valores producen la conducción del transistor PMOS superior y el bloqueo del NMOS. Así, la terminal de E/S queda conectada a la V_{DD} y tiene un nivel alto. Como la línea de salida está sujetada conserva su valor hasta que no es reescrita en el FF tipo D.

Si una línea actúa como entrada, el nivel lógico depositado en ella desde el exterior pasa a la línea correspondiente del bus interno de datos cuando se activa la señal READ y se hace conductor el dispositivo triestado que les une. Al programarse como entrada, los dos transistores MOS de salida quedan bloqueados y la línea en alta impedancia.

Así, que cuando se lee una línea de entrada se obtiene el estado actual que tiene su terminal correspondiente y no el valor que haya almacenado en el FF de datos. La información presente en una línea de entrada se muestrea al iniciarse el ciclo de instrucción y debe mantenerse estable durante su desarrollo.

Al iniciarse el PIC todos los bits de los registros TRIS quedan a 1, con lo que las líneas de los Puertos quedan configuradas como entradas.

Cada línea de salida puede suministrar una corriente máxima de 20 mA y si es entrada puede absorber hasta 25 mA. Al existir una limitación en la disipación máxima de la potencia del chip se restringe la corriente máxima de absorción del PuertoA a 80 mA y la de suministro a 50 mA. El Puerto B puede absorber un máximo de 150 mA y suministrar un total de 100 mA.

Con *movf puerto,w* se lee un puerto y con la instrucción *movwf puerto* se escribe. También existen instrucciones para modificar el valor de un bit particular correspondiente a una línea de un puerto con las instrucciones *bsf puerto,bit* (pone a 1 el bit indicado de él puerto) y *bcf puerto,bit* (pone a cero el bit indicado). Existen instrucciones de salto condicionales que verifican el valor de un bit de un puerto y brincan si vale 1 (*btfss*) o si vale 0 (*btfsc*).

En la tabla 2.4.2.1 se muestra una descripción de cada uno de los pines del puerto A.

Tabla 2.4.2.1. Descripción de las terminales del puerto A

Nombre	*Bit*	*Tipo de Buffer*	*Función*
RA0	Bit 0	TTL	Entrada / salida
RA1	Bit 1	TTL	Entrada / salida
RA2	Bit 2	TTL	Entrada / salida
RA3	Bit 3	TTL	Entrada / salida
RA4/TOCK1	Bit 4	ST	Entrada / salida o entrada de reloj externa para el TMR0. La salida es de tipo drenaje abierto.
TTL = entrada TTL, ST = entrada Schmitt Trigger.			

El siguiente programa muestra un ejemplo de cómo programar el puerto A como salidas, mediante la programación del registro TRISA del banco 1.

```
bsf     ESTADO,RP0      ;Selección del banco 1
movlw   00h             ;Se carga w con cero
movwf   PORTA           ;Se carga el registro TRISA con 00
bcf     ESTADO,RP0      ;Selección del banco 0
```

2.4.3 Puerto B (Registros PORTB y TRISB)

Consta de 8 líneas bidireccionales de E/S, RB7-RB0, cuya información se almacena en el registro PORTB, que ocupa la dirección 6 del banco 0, tal como se muestra en la figura 2.4.3.1. El registro de configuración TRISB ocupa la misma dirección en el banco 1. La línea RB0/INT tiene dos funciones multiplexadas. Además de pines de E/S, actúa como terminal para la repetición de una interrupción externa, cuando se autoriza esta función mediante la adecuada programación del registro INTCON, del que se hablará más adelante.

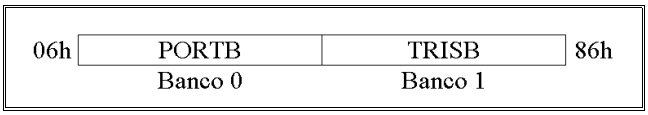


Figura 2.4.3.1. Ubicación de PORTB y TRISB en los bancos del PIC.

A todas las líneas de este puerto se les permite conectar una resistencia Pull-Up de elevado valor con el positivo de la alimentación. Para este fin hay que programar en el registro OPTION el bit RBPU# = 0, afectando la conexión de la resistencia a todas las líneas. Con el Reset todas las líneas quedan configuradas como entradas y se desactivan las resistencias Pull-Up.

Las 4 líneas de más peso, RB7-RB4, pueden programarse para soportar una misión especial. Cuando las 4 líneas actúan como entradas se les puede programar para generar una interrupción si alguna de ellas cambia su estado lógico. Esta posibilidad es muy práctica en el control de teclados.

En la figura 2.4.3.2 se muestra el esquema de conexionado entre los pines RB7-RB4 y las líneas correspondientes del bus de datos interno.

El estado de los pines RB7-RB4 en modo entrada se compara con el valor antiguo que tenían y que se había sujetado durante la última lectura de el Puerto B. El cambio de estado en alguna de esas líneas origina una interrupción y la activación del señalizador RBIF.

La línea RB6 también se utiliza para la grabación serie de la memoria de programa y sirve para soportar la señal de reloj. La línea RB7 constituye la entrada de los datos en serie.

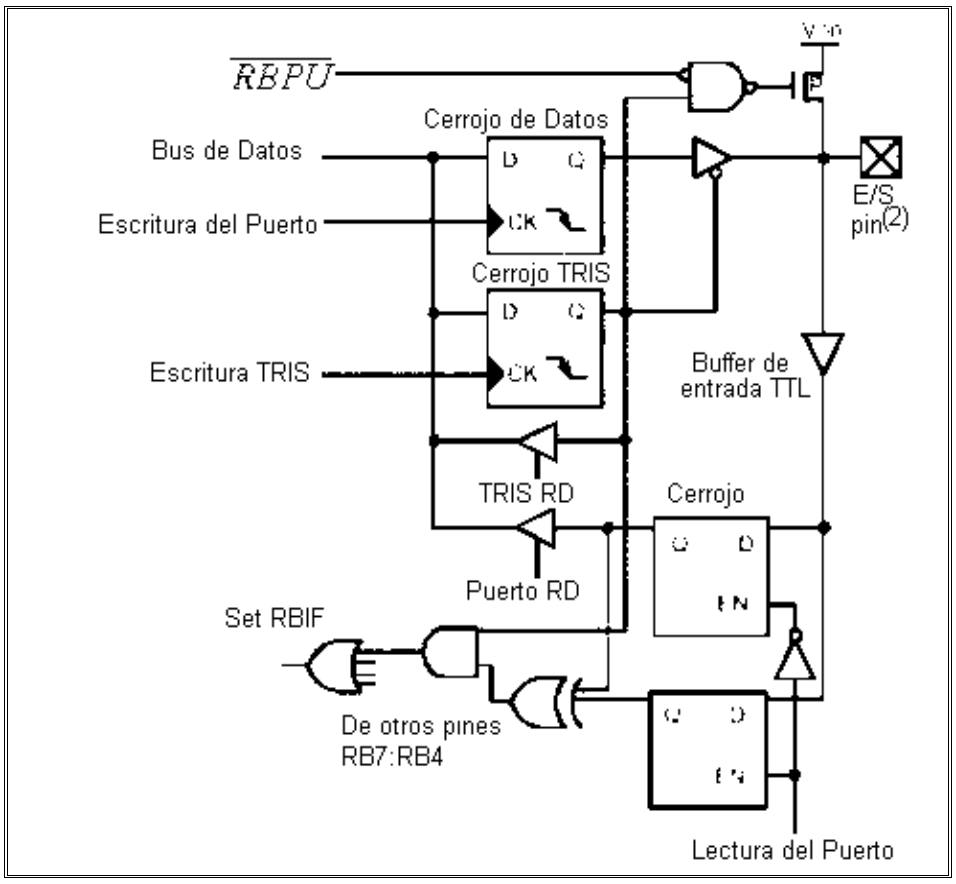


Figura 2.4.3.2. Conexionado de los pines RB7 – RB4 y las líneas correspondientes al bus interno de datos y las señales de control.

Los pines RB3 al RB0 pueden ser programados como líneas de entrada o salida, dependiendo de su configuración en el registro TRISB. El conexionado de dichos pines se muestra en la figura 2.4.3.3.

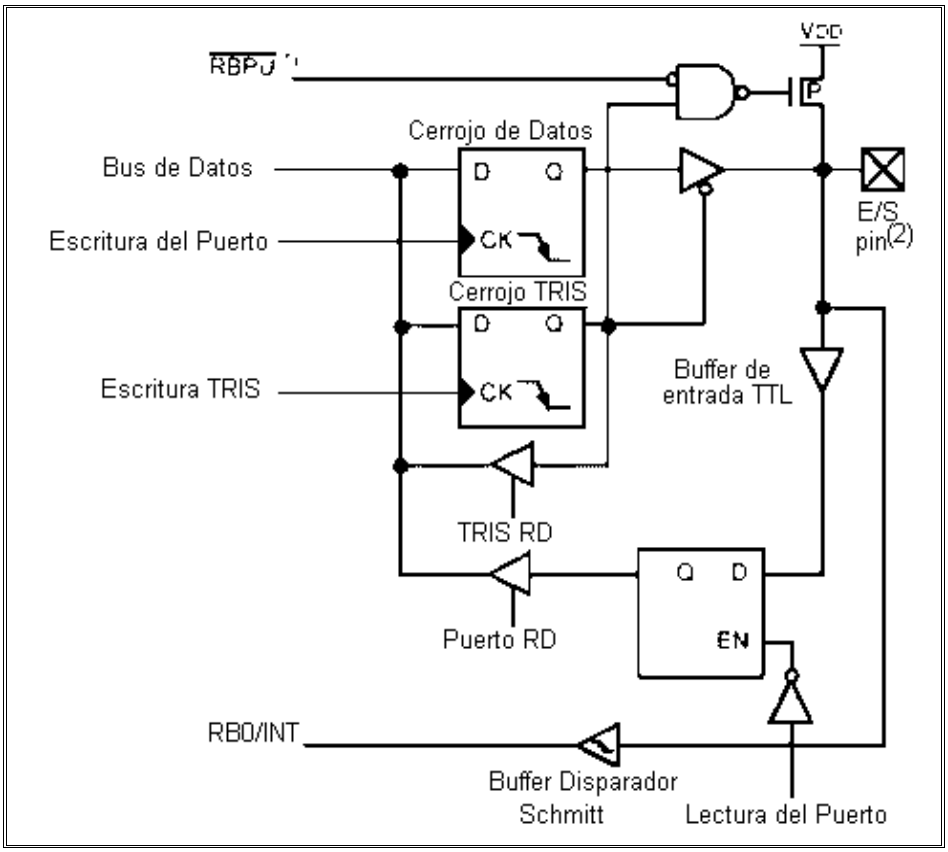


Figura 2.4.3.3. Conexionado de los pines RB3 – RB0 y las líneas correspondientes al bus interno de datos y las señales de control.

El siguiente programa muestra un ejemplo de cómo programar el puerto A como salidas, mediante la programación del registro TRISB del banco 1.

```
bsf     ESTADO,RP0      ;Selección del banco 1
movlw   00h             ;Se carga w con cero
movwf   PORTB           ;Se carga el registro TRISB con 00
bcf     ESTADO,RP0      ;Selección del banco 0
```

En la tabla 2.4.3.1 se muestra una descripción de cada uno de los pines del puerto A.

Tabla 2.4.3. 1. Descripción de las terminales del puerto B.

Nombre	Bit	Tipo de Buffer	Función
RB0/INT	Bit 0	TTL/ST	Entrada / salida o entrada de interrupción externa. Resistencias de Pull-Up por software
RB1	Bit 1	TTL	Entrada / salida (Resistencia de Pull-Up por software)
RB2	Bit 2	TTL	Entrada / salida (Resistencias de Pull-Up por software)
RB3	Bit 3	TTL	Entrada / salida (Resistencias de Pull-Up por software)
RB4	Bit 4	TTL	Entra / salida (con interrupción por cambio) (Resistencias de Pull-Up por software)
RB5	Bit 5	TTL	Entrada / salida (con interrupción por cambio) (Resistencias de Pull-Up por software)
RB6	Bit 6	TTL/ST	Entrada / salida (con interrupción por cambio) (Resistencias de Pull-Up por software)
RB7	Bit 7	TTL/ST	Entrada / salida (con interrupción por cambio) (Resistencias de Pull-Up por software)
TTL = entrada TTL, ST = entrada Schmitt Trigger.			

2.5 Modulo temporizador/contador TMR0

Una de las tareas de los programas de control de dispositivos es determinar intervalos de tiempo al elemento encargado de realizar esta función se le denomina temporizador (timer). También suele ser frecuente contar los impulsos que se producen en el exterior del sistema y el elemento destinado a este fin se denomina contador.

Si las tareas del temporizador o contador se asignarán al programa principal robarían mucho tiempo al procesador. Por este motivo se diseñan recursos específicamente orientados a estas misiones.

Los PIC16X8X poseen un temporizador/contador de 8 bits, llamado TMR0, que actúa de dos maneras diferentes. Como contador de sucesos, que están representados por los impulsos que se aplican a la terminal RA4/T0CKI. Al llegar al valor FFh se desborda el contador y con el siguiente impulso, pasa a 00h, activando un señalizador y/o provocando una interrupción. Como temporizador, cuando se carga en el registro que implementa al recurso un valor inicial se incrementa con cada ciclo de instrucción (Fosc/4) hasta que se desborda, o sea, pasa de FFh a 00h y avisa poniendo a 1 un bit (bandera) y/o provocando una interrupción.

El temporizador se carga con un valor inicial que se va incrementando hasta el desbordamiento. Como contador va incrementando su valor con cada pulso que se le aplica. Al alcanzar el máximo valor binario se «desborda» y pasa a cero, circunstancia que indica una bandera.

Para que el TMR0 funcione como contador de impulsos aplicados a la terminal T0CK1 hay que poner a 1 el bit T0CS, que es el que ocupa la posición 5 del registro OPTION. En esta situación, el registro TMR0 que es el ubicado en la dirección 1 del banco 0 de la memoria de datos, se incrementa con cada flanco activo aplicado en la terminal TOCK1.

El tipo de flanco activo se elige programando el bit T0SE, que es el que ocupa la posición 4 del registro OPTION, si T0SE = 1, el flanco activo es el descendente, y si T0SE= 0, es el ascendente. Cuando se desea que TMR0 funcione como temporizador el bit T0CS = 0. En el diagrama a bloques de la figura 2.5.1 se describe gráficamente el funcionamiento del TMR0.

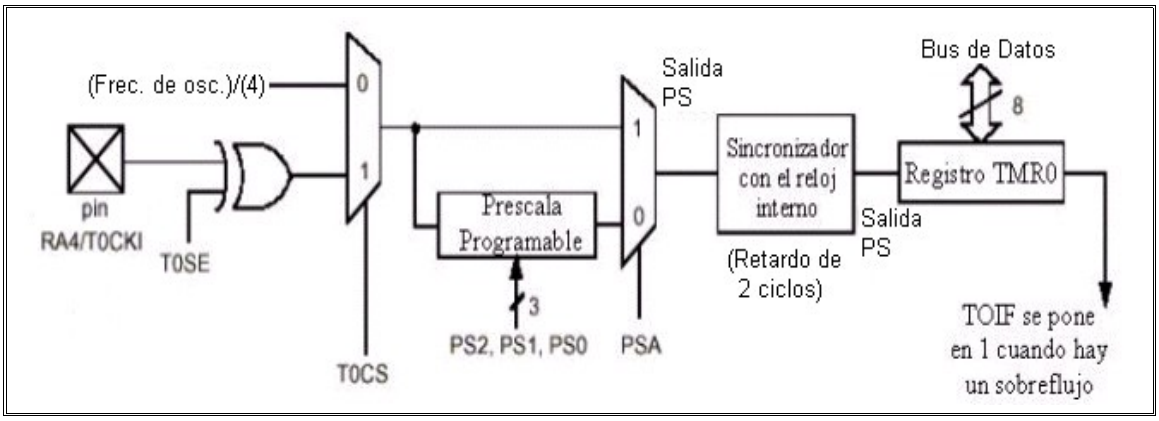


Figura 2.5.1. Diagrama simplificador del temporizador TMR0.

En realidad, los PIC16X8X y los de la gama baja disponen de dos temporizadores, el TMR0 y el Perro Guardián (watchdog). El primero actúa como principal y sobre él recae el control de tiempos y el conteo de pulsos. El otro vigila que el programa no se «cicle», y para ello cada cierto tiempo comprueba si el programa se está ejecutando normalmente. En caso contrario, si el control está detenido en un bucle infinito a la espera de algún acontecimiento que no se produce, el perro guardián «interrumpe», lo que se traduce en un reset que reinicializa todo el sistema.

A menudo el TMR0 y el Perro Guardián precisan controlar largos intervalos de tiempo y necesitan aumentar la duración de los impulsos de reloj que les incrementa. Para cubrir este requisito se dispone de un circuito programable denominado divisor de frecuencia, que divide la frecuencia utilizada por diversos rangos.

Para programar el comportamiento del TMR0, el Perro Guardián (WDT) y el divisor de frecuencia se utilizan algunos bits del registro OPTION y de la palabra de

configuración, que se aplicará más adelante. En la figura 2.5.2 se proporciona un esquema simplificado de la arquitectura del circuito de control de tiempos usado en los PIC16X8X.

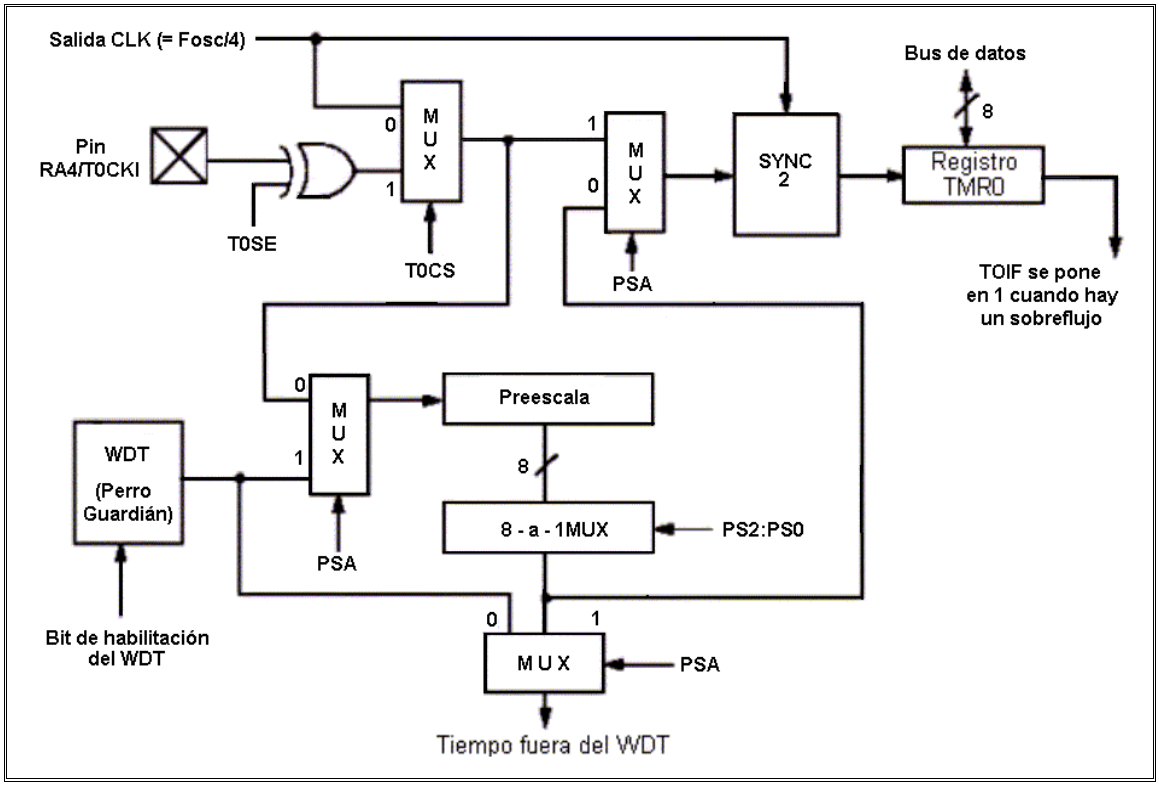


Figura 2.5.2. Esquema simplificado del circuito de control de tiempos usado en los PIC16X8X

El divisor de frecuencia puede usarse con el TMR0 o con el WDT. Con el TMR0 actúa como *pre-divisor,* es decir, los impulsos pasan primero por el divisor y luego se aplican al TMR0, una vez aumentada su duración. Con el perro guardián actúa después, realizando la función de *Post-divisor.*

Los pulsos, que divide por un rango el divisor de frecuencia, pueden provenir de la señal de reloj interna (Fosc/4) o de los que se aplican a la terminal T0CK1.

El TMR0 se comporta como un registro de propósito especial (SFR) ubicado en la dirección 1 del banco 0 de la memoria de datos. En igual dirección, pero en el banco 1, se

halla el registro OPTION. TMR0 puede ser leído y escrito en cualquier momento al estar conectado al bus de datos. Funciona como un contador ascendente de 8 bits.

Cuando funciona como temporizador conviene cargarle con el valor de los impulsos que se quiere temporizar, pero expresados en complemento a 2. De esta manera, al llegar el número de impulsos deseados se desborda y al pasar por 00h se activa el señalizador TOIF y/o se produce una interrupción.

Para calcular los tiempos a controlar con TMR0 se utilizan las siguientes fórmulas prácticas.

Temporización = 4 • Tosc • (valor cargado en TMR0) • (rango del divisor)

Valor a cargar en a TMR0 = (Temporización / 4) • Tosc • rango del divisor

En cualquier momento se puede leer el valor que contiene TMR0, sin detener su conteo. La instrucción adecuada al caso es *movf tmr0,w.*

En la figura 2.5.3 se muestra un esquema de funcionamiento del TMR0. Obsérvese que hay un bloque que retrasa 2 ciclos el conteo para sincronizar el momento del incremento producido por la señal aplicada en T0CKI con el que producen los impulsos externos de reloj. Cuando se escribe TMR0 se retrasa 2 ciclos su incremento y se pone a 0 el divisor de frecuencia.

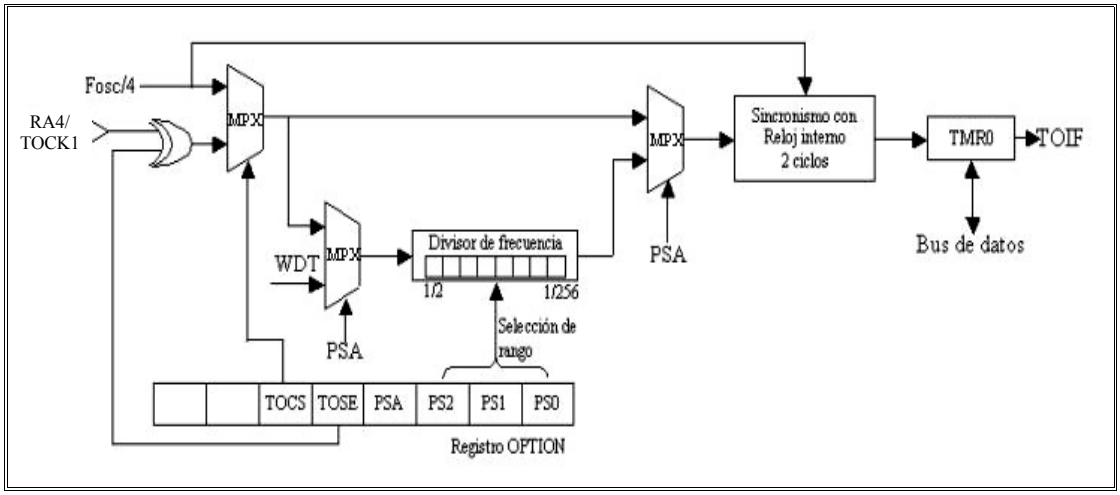


Figura 2.5.3. Esquema general del funcionamiento del TMR0.

La misión principal del registro OPTION es controlar TMR0 y el Divisor de frecuencia. Ocupa la posición 81h de la memoria de datos, que equivale a la dirección 1 del banco 1. El bit T0CS (Timer 0 Clock Edge Select) selecciona en el multiplexor MPX1 la procedencia de los impulsos de reloj, que pueden ser los del oscilador interno (Fosc/4) o los que se aplican desde el exterior por la terminal T0CK1. El bit T0SE (Timer 0 Clock Source Select) elige el tipo de flanco activo en los impulsos externos. Si T0SE = 1 el flanco activo es el descendente y si T0SE = 0 el flanco es ascendente.

El bit PSA del registro OPTION asigna el divisor de frecuencia al TMR0 (PSA = 0) o al WDT (PSA = 1). Los 3 bits de menos peso de OPTION seleccionan el rango por el que divide el Divisor de frecuencia los pulsos que se le aplican en su entrada. La figura 2.5.3 muestra la distribución de los bits del registro OPTION.

El bit 6 INTEDG (*Interrupt Edge*) sirve para determinar el flanco activo que provocará una interrupción externa al aplicarse a la terminal RB0/INT. Un 1 si es ascendente y un 0 descendente.

El bit 7 RBPU# (RB Pull-Up) activa, si sale 0 o desactiva, cuando vale 1, las resistencias Pull-Up que pueden conectarse en las líneas de el puerto B.

2.5.1 Registro OPTION

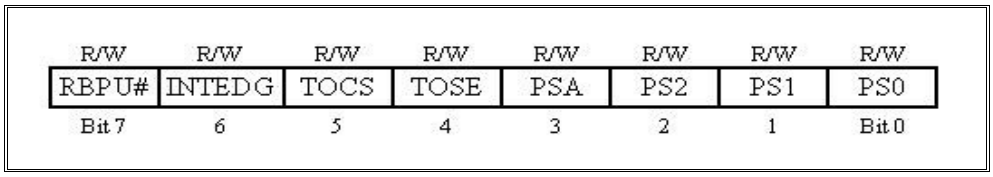


Figura 2.5.1.1. Mapeo de bits del registro OPTION.

A continuación se describen los bits del registro OPTION que se muestran en la figura 2.5.1.1.

RBPU: Resistencia Pull-Up Puerta B

1 = Desactivadas

0 = Activadas

INTEDG: Flanco activo interrupción externa

1 = Flanco ascendente

0 = Flanco descendente

TOCS: Tipo de reloj para el TMR0

1 = Pulsos introducidos a través de T0CKl (contador)

0 = Pulsos de reloj interno Fosc/4 (temporizador)

TOSE: Tipo de flanco en TOCKl

1 = Incremento de TMR0 cada flanco descendente

0 = Incremento de TMR0 cada flanco ascendente

PSA: Asignación del Divisor de frecuencia

1 = El Divisor de frecuencia se le asigna al WDT

0 = El Divisor de frecuencia se le asigna al TMR0

PS2-PS0: Prescaler Value (Valor con el que actúa el Divisor de frecuencia, el cual se muestra en la tabla 2.5.1.1)

Tabla 2.5.1.1. Distribución y asignación de funciones de los bits del registro OPTION

PS2	*PS1*	*PS0*	*Divisor del TMR0*	*Divisor del WDT*
0	0	0	1 : 2	1 : 1
0	0	1	1 : 4	1 : 2
0	1	0	1 : 8	1 : 4
0	1	1	1 : 16	1 : 8
1	0	0	1 : 32	1 : 16
1	0	1	1 : 64	1 : 32
1	1	0	1 : 128	1 : 64
1	1	1	1 : 256	1 : 128

2.6 Memoria EEPROM

El microcontrolador PIC16F84 tiene implementada una memoria de datos de tipo EEPROM, con una capacidad de 64 posiciones de 8 bits cada una. Su principal inconveniente es que la duración del ciclo de escritura o borrado de una posición es muy lento, comparado con la velocidad del procesador, ya que puede llegar a los 10 milisegundos, en los que un PIC16F84, trabaja a 10 Mhz, ejecuta más instrucciones que las 1024 que caben en su memoria de programa. Además, este tiempo es critico y hay que esperar a que termine completamente la operación para iniciar otra nueva.

Los ciclos de lectura y escritura de la EEPROM se realizan sobre la posición de un byte. El ciclo de escritura conlleva el borrado previo y automático de la información que contenía la posición accedida. El tiempo del ciclo de escritura está controlado por un temporizador integrado en el chip. El gran problema es que dicho tiempo no sólo es largo, unos 10ms como máximo, sino que es variable según la temperatura y el voltaje aplicado. Como este tiempo es critico, el final de la operación está controlado por un señalizador (EEIF), que puede causar una interrupción al completarse la escritura de una posición.

Si con los bits correspondientes de la palabra de configuración se protege el código del microcontrolador, se puede continuar leyendo y escribiendo la EEPROM.

2.6.1 Mapa de memoria EEPROM

Aunque la EEPROM es una parte de la memoria de datos, sus posiciones no están mapeadas junto a los registros SFR y GPR de la RAM, que es volátil. Ocupa un espacio de memoria independiente, tal y como se muestra en la figura 2.6.1.1.

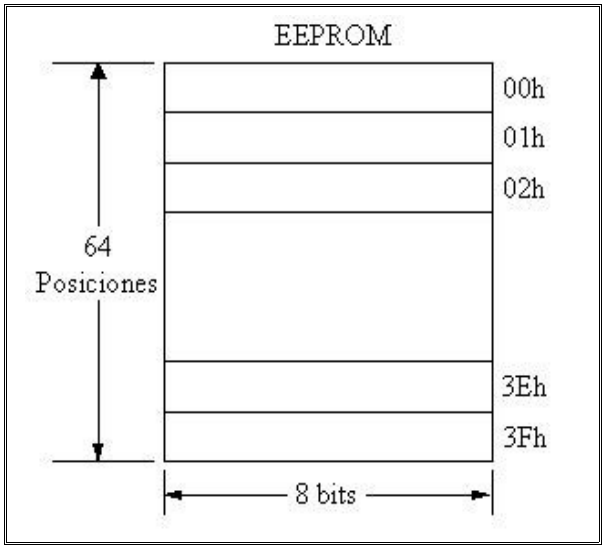


Figura 2.6.1.1. Mapeo de memoria EEPROM del PIC16F84

2.6.2 Manejo de la EEPROM

Para manipular la lectura y escritura de las 64 posiciones de la EEPROM del PIC16F84 se emplean cuatro registros del área de SFR:

- EECON1
- EECON2
- EEDATA
- EEADR

Para manejar el espacio de la EEPROM hay que utilizar dos registros de control. Se trata de los registros EEADR y EEDATA. El registro EEADR ocupa la posición 9 del banco 0 de la RAM, mientras que EEDATA se ubica en la dirección 8 del banco 0, como se muestra gráficamente en la figura 2.6.2.1.

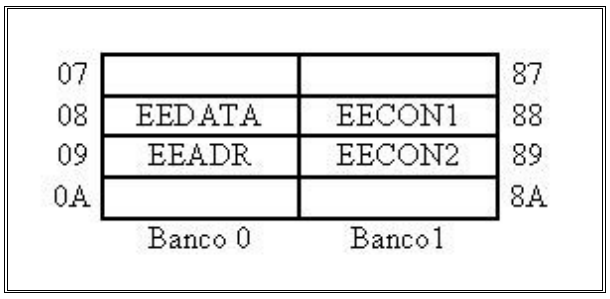


Figura 2.6.2.1. Ubicación de los registros EEDATA, EEADR, EECON1 y EECON2

En la figura 2.6.2.1 se muestra la ubicación de los registros EEADR y EEDATA que se emplean en el manejo del espacio de memoria de la EEPROM. También aparecen los registros de control EECON1 y EECON2.

Ocupando una posición correlativa a EEDATA, pero en el banco 1, se encuentra el registro de control de la EEPROM, EECON1. De manera similar, el registro de control EECON2 ocupa una posición correlativa a EEADR, en el banco 1.

Para acceder a una posición de la EEPROM hay que cargar en el registro EEADR la dirección de la misma, mientras que en EEDATA se carga el dato a escribir, o bien, se deposita el dato leído en él.

Como los registros EEADR y EEDATA son de 8 bits, los dos bits de más peso del registro EEADR siempre valen cero, ya que el mayor valor que se puede cargar es 3Fh.

En la posición relativa 8 del banco 1 de la zona SFR de la RAM está situado el registro EECON1, cuyos bits tiene la misión de controlar las operaciones de la EEPROM. Los tres bits de más peso de este registro no tienen asignada ninguna función.

Los bits que controlan la lectura y escritura son RD y WR respectivamente. Estos bits no pueden limpiarse, solo ponerse en alto por software. Estos bits se limpian por hardware después del proceso de lectura y escritura.

En la figura 2.6.2.2 se describe cada uno de los bits del registro EECON1.

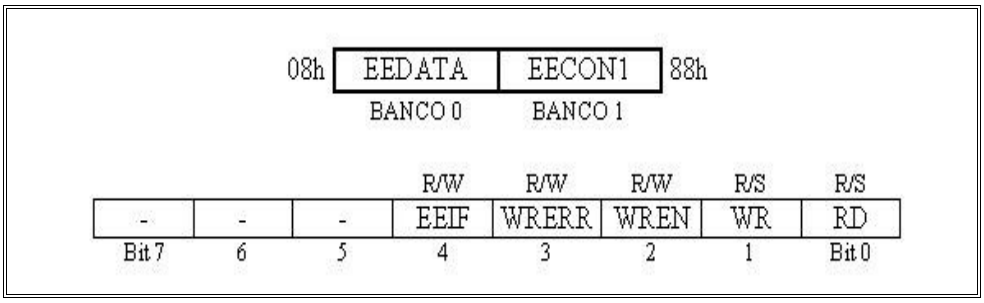


Figura 2.6.2.2. Bits correspondientes al registro EECON1

Los bits 5, 6 y 7 no están implementados, son leídos como ‘0’.

A continuación se describe la función de cada uno de los bits que integran el registro EECON1.

EEIF: Bandera de interrupción de la operación de escritura de la EEPROM

1 = El proceso de escritura se ha completado (debe ser limpiado por software).

0 = El proceso de escritura no se ha completado o no ha empezado.

WRERR: Bandera de error de escritura en la EEPROM

1 = El proceso de escritura ha terminado prematuramente.

0 = La operación de escritura se ha completado correctamente.

WREN: Permiso de escritura de la EEPROM

1 = Permite la escritura de la EEPROM

0 = Inhabilita la escritura en la EEPROM

WR: Bit de control de escritura

1 = Se inicializa el ciclo de escritura (El bit es limpiado por hardware una vez completada la escritura). El bit solo puede ponerse en 1 (no en cero) por software).

0 = Se ha completado el ciclo de escritura de la EEPROM. (Cuando se completa el ciclo pasa ‘0’ automáticamente).

RD: Bit de control de lectura

1 = Inicializa el ciclo de lectura de la EEPROM (Toma un ciclo de lectura. RD se limpia por hardware). El bit RD solo puede ponerse en '1' (no puede limpiarse) por software.

0 = No se ha iniciado la lectura de la EEPROM (cuando se ha completado el ciclo de escritura de la EEPROM este bit se pone automáticamente en cero).

El registro de control EECON2 en realidad no está físicamente implementado en la RAM. Se utiliza como un dispositivo de seguridad para validar la operación de escritura de la EEPROM y evitar interferencias que pudieran producirse en el largo intervalo de tiempo que dura esta operación. Si se lee todos sus bits valen 0.

2.6.3 Operación de lectura en la EEPROM

La lectura de una de las posiciones de la EEPROM comienza cargando en EEADR la dirección a acceder. Luego se pone a 1 el bit RD del registro EECON1. En el siguiente ciclo el dato leído estará disponible en el registro EEDATA. Dicho dato permanecerá en este registro hasta que se realice otra nueva operación de lectura o escritura en la EEPROM.

El siguiente programa lee la dirección 0Ch de la EEPROM y la deposita en el registro W.

```
STATUS      equ             0x03        ;STATUS = 0x03
EEADR       equ             0x09        ;EEADR = 0x09
EECON1      equ             0x08        ;EECON1 = 0x08

bcf         STATUS,rp0                  ;rp0 = 0 para acceder al banco 0
movlw       0x0C                        ;w = 0C
movwf       EEADR                       ;w = EEADR
bsf         STATUS,rp0                  ;rp0 = 1 para acceder al banco 1
bsf         EECON1,rd                   ;rd = 1 para lectura
bcf         STATUS,rp0                  ;rp0 = 0 para acceder al banco 0
movf        EECON1,w                    ;EEDATA = w
```

2.6.4 Operación de escritura en la EEPROM

Para escribir un dato en una posición de la EEPROM hay que seguir una secuencia de instrucciones en la que participa de una manera especial el registro EECON2. Este registro, que en realidad no está implementado físicamente, sólo asume una función de seguridad al cargarse con los valores 55h y AAh, sucesivamente. Esto evita interferencias durante la larga operación de escritura, que a veces llega a los 10 ms.

El ciclo comienza cargando en EEADR la dirección de la posición y en EEDATA el byte a escribir. Si durante el proceso de escritura no se desea admitir interrupciones hay que poner el bit de permiso global de interrupciones GIE = 0. Así se evita que durante la operación de escritura la CPU se desvíe a otra rutina si ocurre una interrupción.

Después hay que autorizar la escritura poniendo el bit WREN = 1 en el registro EECON1. A continuación hay que incluir en el programa una secuencia de instrucciones que cargan en primer lugar el valor de 55h en EECON2 y luego AAh en el mismo registro. Una vez ejecutada esta secuencia se pone el bit WR = 1 en EECON1 y se inicia la larga operación de escritura.

Cuando finaliza la escritura el bit WR es igual a 0 y el señalizador EEIF a 1 en el registro EECON1. Este señalizador se pone a 1 automáticamente en cuanto finaliza el ciclo de escritura, y leyendo puede saberse si el acontecimiento ha sucedido. Luego debe ser el programa el que pase a 0 el bit EEIF.

Detrás de cada ciclo de escritura es recomendable comprobar que la posición de la EEPROM se ha grabado correctamente.

El siguiente programa escribe el valor 0xFF en la dirección 0x32 de la EEPROM.

```
STATUS    equ           0x03    ;STATUS = 0x03
EECON1    equ           0x08    ;EEADR = 0x09
EECON2    equ           0x09    ;EECON1 = 0x08

bcf       STATUS,rp0            ;rp0 = 0 para acceder al banco 0
movlw     0x32                  ;w = 0x32
movwf     EECON2                ;w = EEADR
movlw     0xff                  ;w = 0xff
movwf     EECON1                ;w = EEDATA
bsf       STATUS,rp0            ;rp0 = 0 para acceder al banco 1
bcf       INTCON,gie            ;gie = 0 para prohibir interrupciones
bsf       EECON1,wren           ;wren = 1 para permitir escritura en EEPROM
```

Secuencia de seguridad

```
movlw     0x55                  ;w = 0x55
movwf     EECON2                ;w = EECON2
movlw     0xAA                  ;w = 0xAA
movwf     EECON2                ;w = EECON2
bsf       EECON1,wr             ;wr = 1 y comienza el ciclo de escritura
```

2.7 Interrupciones

Las interrupciones constituyen el procedimiento más rápido y eficaz del microcontrolador para el tratamiento de los acontecimiento del mundo exterior. Si algún parámetro o circunstancia tiene influencia en el desarrollo del programa, la interrupción es el medio por el cual el microcontrolador lo atiende inmediatamente. Al programa especial que trata dicho acontecimiento se le llama “rutina de interrupción”.

Las interrupciones son desviaciones del flujo de control del programa originadas asincrónicamente por diversos sucesos que no se hallan bajo la supervisión de las instrucciones. Dichos sucesos pueden ser externos al sistema, como la generación de un flanco o nivel activo en una terminal del microcontrolador, o bien internos, como el desbordamiento de un contador.

El comportamiento del microcontrolador ante la interrupción es similar al de la instrucción CALL de llamada a subrutina. Cuando se utiliza la instrucción CALL, se detiene la ejecución del programa en curso, se guarda la dirección actual del PC en la pila y éste se carga con una nueva dirección que se especifica en dicha instrucción.

En el caso de una interrupción, el proceso inicia de la misma forma; se detiene la ejecución del programa en curso, se guarda la dirección actual del PC en la pila y éste se

carga con una dirección «reservada» de la memoria de programa (0004h), llamada Vector de Interrupción.

En los PIC16X8X el Vector de interrupción se halla situado en la dirección 0004h, en donde comienza la Rutina de Servicio a la Interrupción (RSI). En general, en dicho Vector se suele colocar una instrucción de salto incondicional (GOTO), que traslada el flujo de control a la zona de la memoria de código destinada a contener la rutina de atención a la interrupción.

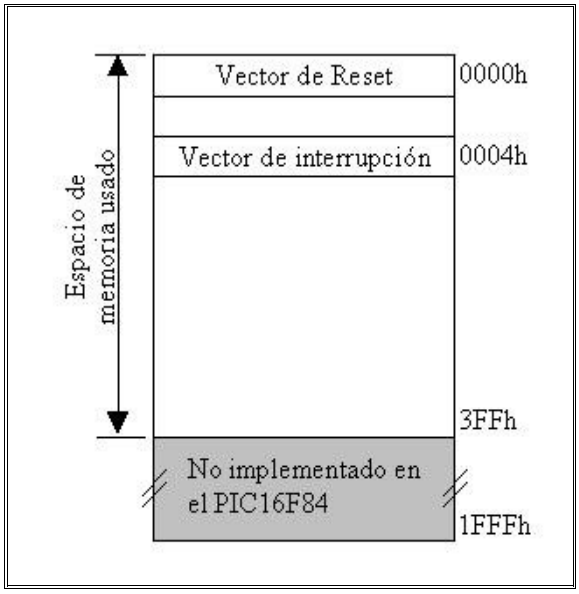


Figura 2.7.1. Localización del vector de interrupción.

En la figura 2.7.1, se muestra la dirección del vector de interrupción que está localizada en la posición 0004h de la memoria de programa del PIC16F84.

La RSI suele comenzar guardando en la memoria de datos algunos registros específicos del procesador. Concretamente aquellos que la RSI va a emplear y va a alterar su contenido. Antes del retorno al programa principal se recuperan los vectores guardados y se restaura completamente el estado del procesador.

Algunos procesadores salvan estos registros en la pila, pero los PIC no disponen de instrucciones para introducir (push) y extraer (pop) información de la pila, utilizando para este fin registros de propósito general de la memoria de datos.

Los PIC 16X8X pueden ser interrumpidos por 4 causas diferentes, pero todas ellas desvían el flujo de control a la dirección 0004h, por lo que otra de las operaciones iniciales

de la RSI es averiguar cuál de las posibles causas ha sido la responsable de la interrupción en curso. Para ello se exploran los señalizadores de las fuentes de interrupción.

Otro detalle importante de la RSI de los PIC 16X8X es que estos microcontroladores poseen un bit GIE (Global Interrupt Enable) que cuando vale 0 inhabilita todas las interrupciones. Pues bien, al comenzar la RSI dicho bit GIE se pone automáticamente a 0, con objeto de no atender nuevas interrupciones hasta que se termine la que ha comenzado. En el retorno final de la interrupción, GIE pasa a valer automáticamente 1 para volver a tener en cuenta las interrupciones. Dicho retorno de interrupción se realiza mediante la instrucción RETFIE. Antes del retorno conviene borrar el señalizador de la causa de interrupciones que se ha atendido, porque si bien los señalizadores se ponen a 1 automáticamente en cuanto se produce la causa que indican, la puesta a 0 se hace por programa, la figura 2.7.2 muestra un organigrama de las fases más importantes que se desarrollan durante el proceso de ejecución de una interrupción.

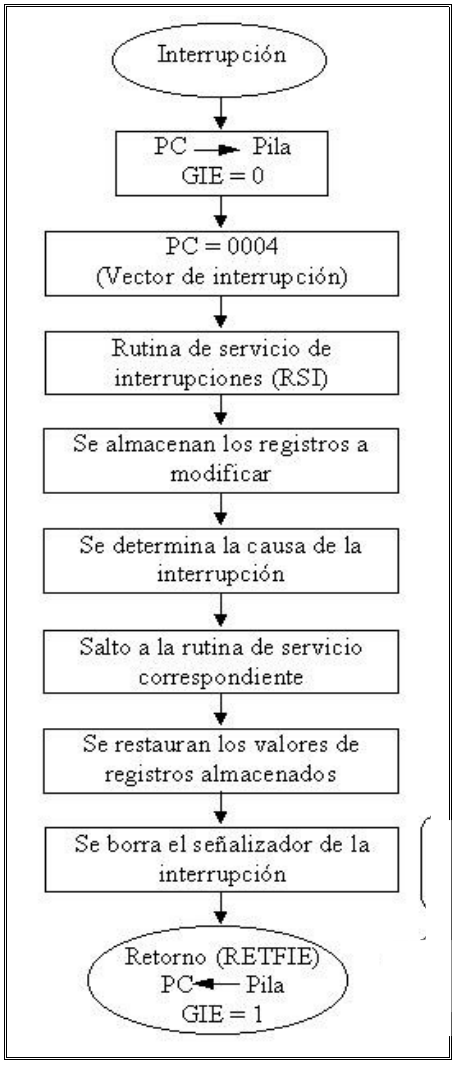


Figura 2.7.2. Organigrama del desarrollo de una interrupción, teniendo en cuenta el papel del bit GIE

2.7.1 Causas de una interrupción

Hay cuatro causas que pueden ocasionar una interrupción del PIC16F84:

- Activación del pin RB0/INT
- Desbordamiento del temporizador TMR0
- Cambio de estado de uno de los 4 pines de mas peso del PUERTO B (RB7:RB4)
- Finalización de la escritura en la EEPROM de datos

Una dificultad inicial que presentan estas interrupciones es que sea cual sea la que ha originado la interrupción, la rutina de atención siempre empieza en la dirección del vector de interrupción que es la 0004h.

Otros procesadores disponen de un vector de interrupción para cada causa que la produce. Con el PIC16F84 no queda otro remedio que comenzar averiguando cuál ha sido la causa de la interrupción, probando los señalizadores correspondientes, y desviando hacia el procedimiento particular que atiende a cada una.

Cuando ocurre cualquiera de las 4 sucesos indicados, se origina una petición de interrupción, que si se acepta y se atiende comienza depositando el valor del PC actual en la pila, poniendo el bit GIE = 0 y cargando en el PC el valor 0004h, que es el vector de interrupción donde se desvía el flujo de control.

Cada fuente de interrupción dispone de un señalizador o flag, que es un bit que se pone automáticamente a 1 cuando se produce. Además cada fuente de interrupción tiene otro bit de permiso, que según su valor permite o prohíbe la realización de dicha interrupción.

2.7.2 Registro de control de interrupciones INTCON

A veces, muchos programas no precisan la ayuda de las interrupciones y no las usan. Por este motivo debe existir una llave general que permita o prohíba las interrupciones en su conjunto. En el PIC16F84 existe un registro de control en el área SFR de la RAM, llamado INTCON, que regula el funcionamiento de las interrupciones, la dirección que ocupa dicho registro es la 0Bh del banco 0 y la dirección 8Bh del banco 1, justo como se muestra en la figura 2.7.2.1 . El bit de más peso de INTCON (INTCON<7>) se denomina GIE (Permiso Global de Interrupciones). Si se escribe este bit y se pone GIE = 0, todas las interrupciones están prohibidas y el procesador no admite ninguna. En cambio, si escribimos GIE = 1 el procesador admite cualquier interrupción que se produzca y que el programador haya habilitado para que actúe. Es decir, cada interrupción dispone de un bit particular de permiso que también debe habilitarse si se quiere que cuando la causa de la interrupción se produzca el procesador lance las fases de su desarrollo.

El registro INTCON es un registro de lectura/escritura que contiene los bits de habilitación de las fuentes de interrupción. Las banderas de interrupción se ponen en alto cuando una interrupción ocurre, sin tener en cuenta el estado de su correspondiente bit de habilitación o el permiso global de interrupciones, GIE (INTCON<7>).

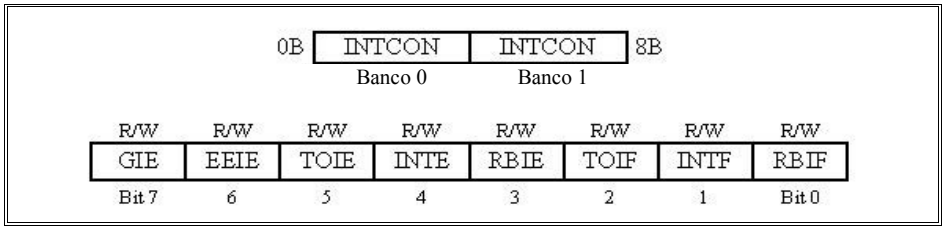


Figura 2.7.2.1. Bits que conforman el registro INTCON.

A continuación se describen los bits que conforman el registro INTCON y que se muestran en la figura 2.7.2.1.

GIE: Habilitador global de interrupciones (Global Interrupt Enable)

1 = Habilita todas las interrupciones

0 = Deshabilita todas las interrupciones

EEIE: Habilitación de interrupción por escritura de la EEPROM (EEPROM Write Interrupt Enable)

1 = Habilita la interrupción EEIE

0 = Deshabilita la interrupción EEIE

TOIE: Habilitación de la interrupción del temporizador TMRO (TMR0 Interrupt Enable)

1 = Habilita la interrupción TOIE

0 = Deshabilita la interrupción TOIE

INTE: Habilitación de la interrupción INT (INT interrupt Enable)

1 = Habilita la interrupción INTE

0 = Deshabilita la interrupción INTE

RBIE: Habilitación de la interrupción RBIF (RBIF Interrupt Enable)

1 = Habilita la interrupción RBIE

0 = Deshabilita la interrupción RBIE

TOIF: Bandera de interrupción por sobreflujo del TMR0 (TMR0 Overflow Interrupt Flag)

1 = Ha ocurrido un sobreflujo en el TMR0 (solo es limpiado por software)

0 = No hay sobreflujo en el TMR0

INTF: Bandera de interrupción INT (RB0/INT Interrupt Flag)

1 = Ha ocurrido la interrupción externa RB0/INT

0 = No hay interrupción externa por RB0/INT

RBIF: Bandera de interrupción por cambio en el puerto B (RB Port Change Interrupt Flag)

1 = Indica el cambio de estado de los bits RB7:RB4 del puerto B (se limpia por software)

0 = No hay cambios de estado en los bits RB7:RB4 del puerto B.

En el organigrama de la figura 2.7.2 se amplía la operatividad del desarrollo de la interrupción al tener en cuenta el papel del bit GIE. Obsérvese que en cuanto se admite una interrupción el procesador pone automáticamente GIE = 0, para evitar que cuando está atendiéndose a una interrupción se produzcan otras nuevas.

Al completarse la rutina de atención a la interrupción y ejecutarse la última instrucción RETFIE, que es la de retorno de la rutina de interrupción, el bit GIE vuelve a tomar el valor 1 de manera automática, para que al retomar el programa principal queden habilitadas las interrupciones y puedan atenderse.

Cabe señalar que los señalizadores de interrupción deben ponerse a 0 por programa antes del retorno de la interrupción y son operativos aunque la interrupción esté prohibida con su bit de permiso correspondiente. En la figura 2.7.2.2 se ofrece el esquema de la lógica de control que origina la interrupción.

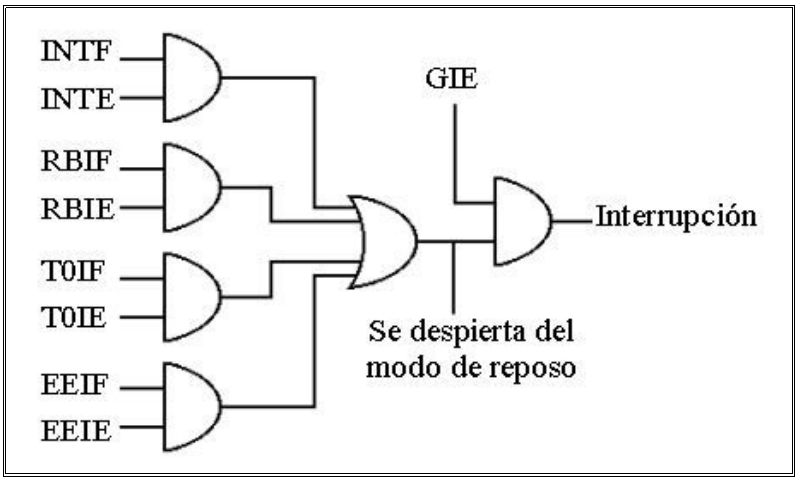


Figura 2.7.2.2. Lógica de control para la generación de una interrupción en los PIC16X8X

El siguiente programa muestra una forma de explorar las diversas causas de interrupción.

```
int     btfsc  INTCON,intf ;Explora el bit intf de intcon y si vale 0 hay brinco
        goto   int_RB0/INT ;Salto para atender la interrupción externa INT
portb   btfsc  INTCON,rbif ;Explora el bit rbif de intcon y si vale 0 hay brinco
        goto   int_portb   ;Salto para atender la interrupción por cambio de
                           ;estado en las líneas RB7:RB4 del puerto B
tmr0    btfsc  INTCON,t0if ;Explora el bit toif de intcon y si vale 0 hay brinco
        goto   int_TMR0    ;Salto para atender la interrupción por
                           ;desbordamiento del TMR0
eeprom  btfsc  EECON1,eeif ;Explora el bit eeif de eecon1 y si vale 0 hay brinco
        goto   int_EEPROM  ;Salto para atender la interrupción por finalización
                           ;de la escritura en la EEPROM
        retfie
```

2.7.3 Interrupción externa INT

De las cuatro posibles causas de interrupción que admite el PIC16F84, hay una que le permite estar en contacto permanente con los acontecimientos que ocurren en el mundo exterior.

Esta interrupción está soportada por el pin 6 del microcontrolador, y se llama RB0/INT. Se trata de un pin multifunción que puede actuar de dos formas diferentes. En primer lugar puede funcionar como línea de entrada/salida digital (RB0), también puede actuar como línea de petición de interrupción externa (INT).

Para que este pin pueda recibir las peticiones externas de interrupción hay que habilitarla para tal trabajo, para lo cual hay que comenzar poniendo a 1 el bit de permiso global de interrupciones GIE. Además, también habrá que habilitar esta interrupción poniendo a 1 el bit INTE. Tanto GIE como INTE se ubican en el registro INTCON.

Mediante el bit 6, llamado INTEG, del registro OPTION se puede seleccionar cuál será el flanco activo en RB0/INT. Si se desea en flanco ascendente se escribe un 1 en dicho bit, y si se desea que sea el flanco descendente se escribe un 0.

A continuación se describe con algunos comentarios un pequeño programa para admitir la interrupción externa con flanco descendente como activo:

```
bst     ESTADO,rp0        ;Selección del banco 1
bcf     OPTION,intdeg     ;INTDEG = 0, flanco descendiente
bcf     ESTADO,rp0        ;Banco 0
bsf     INTCON,gie        ;GIE = 1, Permiso global de interrupciones
bsf     INTCON,inte       ;INTE = 1, Permiso de interrupción externa
bcf     INTCON,intf       ;INTF = 0, Se asegura que el flag empiece a 0
```

Dentro de la rutina de interrupción y antes de retornar al programa principal, el programador debe encargarse de poner el señalizador INTF = 0, pues de lo contrario, al ejecutarse la instrucción de retorno RETFIE volveríamos a repetir el proceso de interrupción, ya que el flag INTF se encontraría a 1.

2.7.4 Interrupción por desbordamiento del TMR0

El temporizador TMR0 es un contador ascendente de 8 bits. Cuando TMR0 se desborda y pasa del valor FFh al 00h, el señalizador T0IF se pone automáticamente a 1.

En el caso del TMR0 su valor se incrementa al ritmo de los impulsos de reloj que se le aplican por el pin RA4/T0CKI o desde el oscilador principal interno con la frecuencia Fosc/4. Cuando el valor del TMR0 pasa de FFh a 00h el señalizador T0IF, que es el bit 2 de INTCON, se pone a 1 automáticamente.

Para que el procesador sepa cuando se desborda el TMR0 debe explorar cada poco tiempo el estado del señalizador T0IF. El chequeo de dicho bit supone una constante atención y pérdida de rendimiento. Para conocer inmediatamente cuándo se ha desbordado el TMR0 es mucho más eficaz generar una interrupción cuando ese hecho ocurra. Para ello es necesario habilitar la interrupción del temporizador, lo cual implica inicialmente poner el bit GIE = 1 , que es el permiso global de interrupciones, y después activar el bit de permiso de la interrupción del TMR0, o sea, escribir un 1 en T0IE.

En la figura 2.7.4.1 se muestra el diagrama a bloques de la interrupción por desbordamiento del temporizador TMR0.

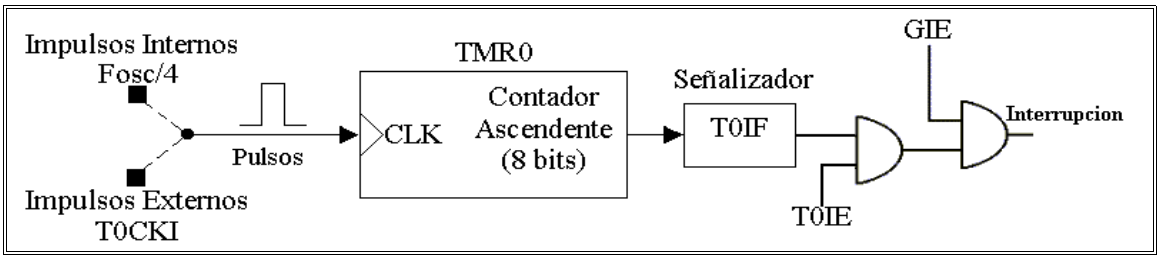


Figura 2.7.4.1. Diagrama a bloques de la Interrupción por desbordamiento del TMR0

Si no se recarga el TMR0 cuando se desborda, sigue contando desde 00h a FFh. En cualquier momento se puede leer y escribir este registro, pero cada vez que se escribe se pierden dos ciclos de reloj para la sincronización.

Cuando se carga inicialmente TMR0 con el valor de N_{10}, cuenta 256 – N impulsos, siendo el tiempo que tarda en hacerlo el que expresa la siguiente formula:

Temporización = 4 * Tosc * (256 – N_{10}) * Rango del divisor de frecuencia

Para entender lo anterior se propondrá un ejemplo:

Calcular el tiempo para que el TMR0 genere una interrupción si se le carga continuamente con el valor 40_{10}, el divisor de frecuencia está programado en el rango 1:8 y el microcontrolador dispone de un cristal de 1 MHz.

Tosc = 1/1MHz

Tosc = 1μs

Temporización = 4 * Tosc * (256 – N_{10}) * Rango del divisor de frecuencia

Temporización = 4 * 1μs * (256 – 40) * 8 = 6.912 μs

2.7.5 Interrupción por cambio de estado en RB7:RB4

Una de las cuatro causas que producen la interrupción del PIC16F84 es el cambio del estado lógico en alguna de las cuatro líneas de más peso del puerto B (RB7:RB4).

Si en un momento una de las mencionadas líneas, que actúan como entradas digitales, cambia el estado lógico que recibe del mundo exterior, el señalizador RBIF se pone a 1, y si el bit de permiso de esta interrupción también está activo (RBIE = 1) se genera una interrupción, para que esta ocurra es necesario que el bit de permiso global de interrupciones, GIE valga 1, y además, que el bit de permiso particular de la interrupción por cambio de estado en RB7:RB4 (RBIE) también esté a 1.

La razón por la que existe esta extraña causa de interrupción es la de atender a los teclados matriciales, tan usuales en las aplicaciones con microcontroladores. El teclado es un periférico de entrada de información fácil de manejar, de reducido tamaño y economía, tal como se muestra en la figura 2.7.5.1.

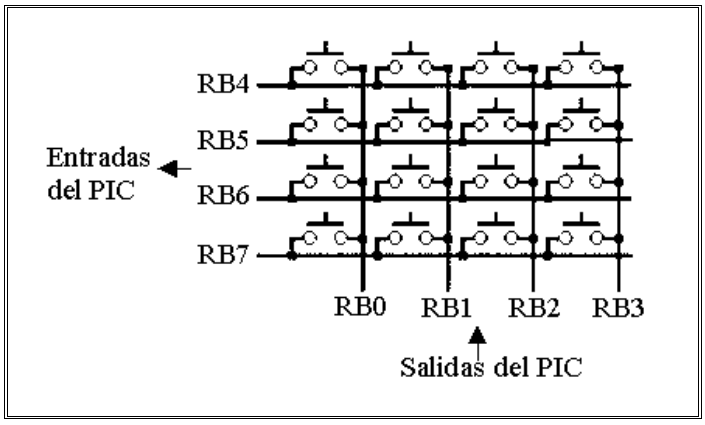


Figura 2.7.5.1. Estructura y conexionado del teclado matricial a los pines del PUERTO B del PIC

2.7.6 Interrupción por finalización de la escritura en la EEPROM

El tiempo típico que tarda en desarrollarse una operación de escritura en la EEPROM de datos de los PIC16X8X es de 10 ms, que es considerable comparado con la velocidad a la que el procesador ejecuta instrucciones. Para asegurarse que se ha

completado la escritura y puede continuarse con el flujo de control del programa es aconsejable manejar la interrupción que se origina al finalizar la escritura, que pone automáticamente el señalizador EEIF a 1, y se autoriza siempre que los bits de permiso EEIE = 1 y GIE = 1.

Cuando se describió el proceso de escritura de la EEPROM de datos se indicó que se usaba un registro no real para asegurar la misma. Se trataba del EECON2, en el que se grababan dos valores, el 55h y el AAh. Durante la escritura de este registro debe prohibirse la aceptación de interrupciones para salvaguardar la operación de escritura, por eso en ese módulo se pone GIE = 0, tal como se indica en el siguiente segmento de programa orientado a escribir la memoria EEPROM. Se supone que la dirección a acceder ya se ha cargado en el registro EEADR y el dato a escribir en EEDATA.

```
bsf     STATUS,rp0      ;Selecciona el banco 1
bcf     INTCON,gie      ;GIE = 0. Prohíbe interrupciones
bsf     INTCON,eeie     ;Permite la interrupción de la EEPROM
bsf     EECON1,wren     ;Permiso de escritura de la EEPROM
movlw   0x55
movwf   EECON2          ;Se escribe en EECON2 el dato 55h
movlw   0xAA
movw    EECON2          ;Se escribe AAh en EECON2
bsf     EECON1,wr       ;Comienza el proceso de escritura
bsf     INTCON,gie      ;Permiso de interrupciones
```

En los PIC16C84 y PIC16F8X se puede leer y escribir la EEPROM de datos aunque se haya protegido el código. En los PIC16CF8X, que disponen de memoria ROM para el código, existen dos bits para el código de protección: uno dedicado a la ROM de código y el otro a la EEPROM de datos.

2.8 El perro guardián (WDT)

El perro guardián, llamado abreviadamente WDT (Watchdog Timer) es un contador ascendente de 8 bits que tiene la propiedad especial de producir el Reset al microcontrolador cada vez que se desborda. A diferencia del TMR0, el Perro guardián no dispone de señalizadores, ni tampoco genera interrupción en el desbordamiento, lo único que hace es provocar un Reset que inicializa el procesador y comienza a ejecutar el programa principal desde la primera instrucción que se halla en la dirección 0000h de la memoria de programa (Vector Reset).

El objetivo del Perro Guardián es comprobar el correcto procesamiento de las instrucciones del programa. Si todo va bien, el programa proporciona, antes de terminar su temporización, una instrucción que borra e inicia de nuevo el conteo, algo así como si se "refrescara". Si el programa se ha colgado y no se han ejecutado las instrucciones según las previsiones, el programa no suministrará la instrucción de refresco, con lo que se desbordará el WDT y se generará un Reset. Con esta acción se pretende que el microcontrolador comience de nuevo el programa y evita la causa que lo bloqueó, por ejemplo, entra en bucle infinito, o porque precisa una señal para salir de él que no llega, etc.

Para evitar que se desborde el Perro Guardián hay que refrescarle previamente. En realidad este refresco consiste en ponerle a cero mediante las instrucciones *clrwdt* y *sleep*. El programador debe analizar las instrucciones de la tarea y situar alguna de esas dos en sitios estratégicos por los que pase el flujo de control antes de que transcurra el tiempo asignado al WDT. De esta manera si el programa se «cicla» no se refresca al Perro Guardián y se produce la reinicialización del sistema.

La instrucción *clrwdt* borra al WDT y reinicia su cuenta. Sin embargo, la instrucción *sleep*, además de borrar WDT detiene al sistema y lo mantiene en un estado de «reposo» o «de bajo consumo». Si no se desactiva el Perro Guardián al entrar en el modo de reposo, al completar su ciclo de espera provocará un Reset y sacará al microcontrolador del modo de bajo consumo. Para desactivar el Perro Guardián hay que escribir un 0 en el bit 2 (WDTE) de la Palabra de Configuración.

En el registro ESTADO existe un bit denominado $\overline{TO}$ que toma el valor de 0 después del desbordamiento del WDT.

El Perro Guardián se activa y se desactiva con el valor del bit WDTE de la palabra de configuración. La frecuencia de trabajo de este temporizador siempre es interna e independiente, originada por un oscilador RC propio. Cuando se asigna el divisor de frecuencia al Perro Guardián actúa como postdivisor, al revés de cómo actuaba con el TMR0. De esta manera, los impulsos que genera el WDT pasan al divisor de frecuencia que los divide por el rango que seleccionamos con los tres bits de menos peso del registro OPTION.

Si el bit PSA es 1, en el registro OPTION, el Divisor de frecuencia se aplica al WDT. Es curioso apreciar que el Perro Guardián no dispone de ningún registro en el área SFR de la RAM. La razón se debe a que no se carga con ningún valor inicial, empieza desde 0 hasta desbordarse. Cada vez que se refresca con *clrwdt* vuelve a ponerse a 0 y a reiniciar el conteo. En el registro STATUS existe un bit llamado $\overline{TO}$ que toma el valor de 0 cuando se desborda el WDT, así discrimina la forma en que se ha realizado el Reset. En la figura 2.8.1 se muestra un diagrama a bloques que describe el funcionamiento del WDT.

El temporizador WDT se halla preparado para que controle un tiempo de 18ms, pero se puede actuar sobre el divisor de frecuencia para que dicho tiempo se eleve hasta un máximo de 2.3 segundos.

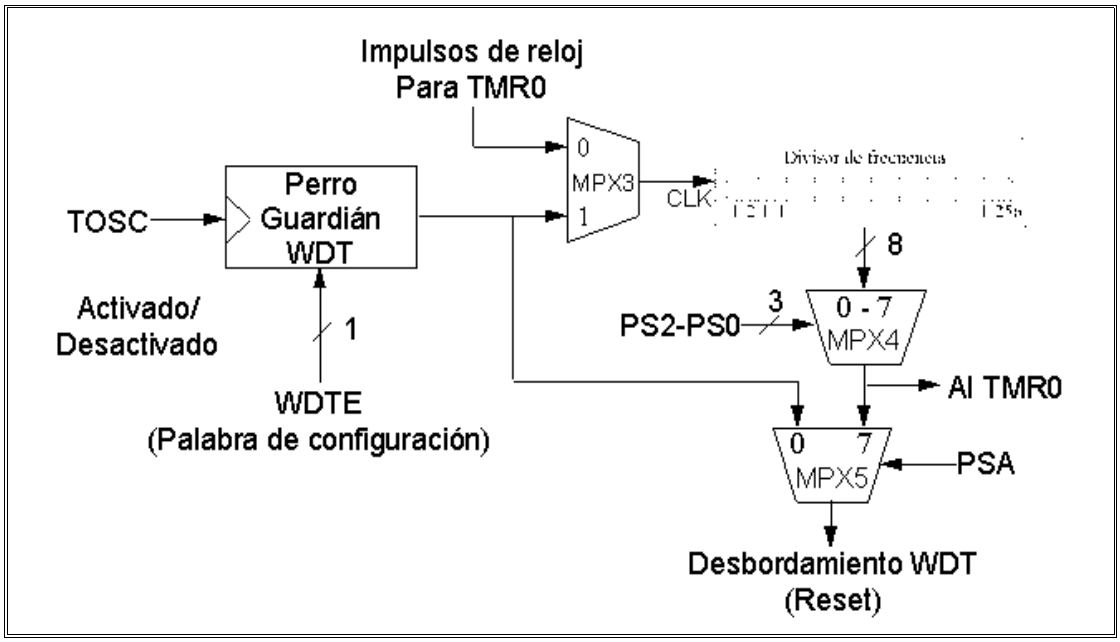


Figura 2.8.1. Diagrama a bloques de la operación del Perro guardián (WDT).

2.9 Reinicialización o reset

Cuando se aplica un nivel bajo durante el tiempo suficiente al pin $\overline{MCLR}$, se produce la reinicialización, que conlleva la ejecución de dos acciones importantes.

- La puesta a cero del Contador de Programa, que así pasa a direccionar la primera instrucción del programa.

- Colocar en un determinado estado la mayoría de los bits de los registros de control del procesador.

En la figura 2.9.1 se muestra un sencillo esquema, a base de un pulsador, que provoca el Reset externo.

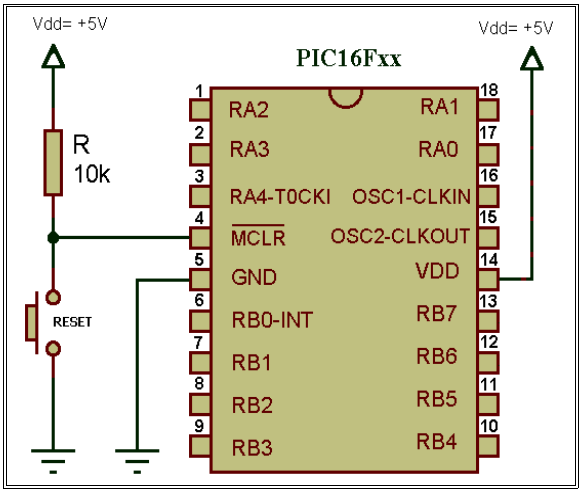


Figura 2.9.1. Diagrama de conexión para controlar el reset del PIC.

Cuando se presiona el pulsador de Reset el pin $\overline{MCLR}$ pasa a nivel lógico bajo

Otro esquema habitual para el pulsador de Reset incluye una resistencia de absorción de 10K, un diodo que impide la circulación de corriente hacia el pin 4 y otra pequeña resistencia de 100Ω, como se muestra en la figura 2.9.2.

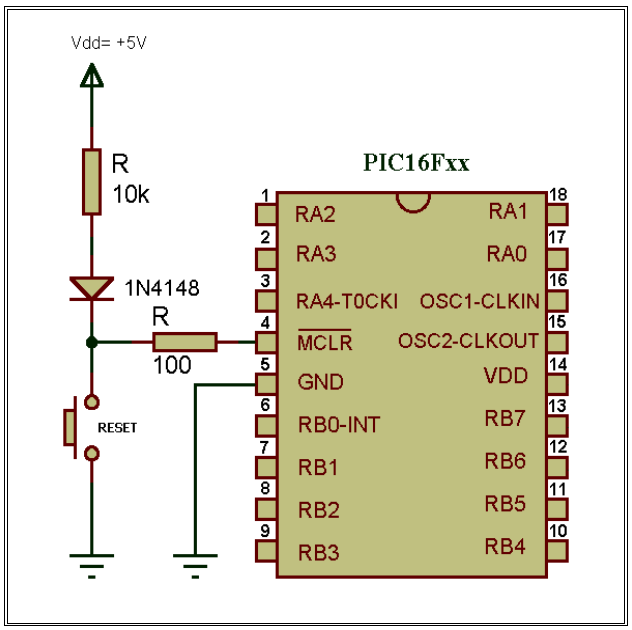


Figura 2.9.2. El diodo impide que circule corriente del positivo al interior del microcontrolador

2.9.1 Causas que provocan el reset

El Reset pone a cero el PC y coloca en un estado conocido la mayoría de los bits de los registros de control. Existen tres causas principales que originan un Reset:

- Conexión de la alimentación (por: Power On Reset). Se producen al detectarse la subida de la tensión de alimentación desde 1.2 a 1.7 V.
- Activación del pin $\overline{MCLR}$. Esto puede suceder tanto cuando el PIC funciona normalmente como cuando esté es estado de reposo.
- Desbordamiento de Perro Guardián (WDT). Igualmente puede ocurrir en el modo de funcionamiento normal del PIC y en su estado de reposo.

En cualquiera de las cinco posibilidades después del Reset el PC se carga con cero, excepto cuando se desborda el WDT estando el PIC en estado de reposo, donde el PC se incrementa una unidad para ejecutar la siguiente instrucción a la que lo introdujo en ese

estado de bajo consumo. En la tabla 2.9.1.1 se presenta el valor que toma cada uno de los bits de los registros SFR tras cada una de las posibles causas de Reset.

Tabla 2.9.1.1. Valores que toman los bits de los registros SFR tras los posibles Reset.

Registro	Dirección	Conexión de la alimentación	Desbordamiento perro guardián modo normal	Desbordamiento perro guardián modo reposo	$\overline{MCLR}$ modo normal	$\overline{MCLR}$ modo reposo
W	- - - -	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
INDIR	00 h	- - - -	- - - -	- - - -	- - - -	- - - -
TMR0	01 h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PCL	02 h	0000 h	0000 h	PC + 1	0000 h	0000 h
STATUS	03 h	0001 1xxx	0000 1uuu	uuu 0 0uuu	000u uuuu	0001 0uuu
FSR	04 h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PORT A	05 h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PORT B	06 h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TRIS A	85 h	- - -1 1111	- - -1 1111	----u uuuu	- - -1 1111	- - -1 1111
TRIS B	86 h	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
OPTION	81 h	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
EEDATA	08 h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
EEADR	09 h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
EECON 1	88 h	- - -0 0000	- - -0 ?000	- - -u uuuu	- - -0 ?000	- - -0 ?000
EECON 2	89 h	- - - -	- - - -	- - - -	- - - -	- - - -
PCLATH	0A h	- - -0 0000	- - -0 0000	- - -u uuuu	- - -0 0000	- - -0 0000
INTCON	0B h	0000 000x	0000 000u	uuuu uuuu	0000 000u	0000 0000

u=No cambia x=Independiente - - - -=No implementado ?=Dependiente de otras condiciones

2.9.2 Esquema eléctrico del reset

En la figura 2.9.2.1 se ofrece el esquema eléctrico que controla la actividad del Reset. En él destacan los temporizadores OST y PWRT.

El temporizador Power-up Timer (PWRT) añade 72 milisegundos que mantienen al microcontrolador en la situación de Reset, para dar tiempo a que se estabilice el voltaje de alimentación. Para su funcionamiento hay que poner a nivel bajo el bit de permiso $\overline{\text{PWRTE}}$, que reside en la palabra de configuración.

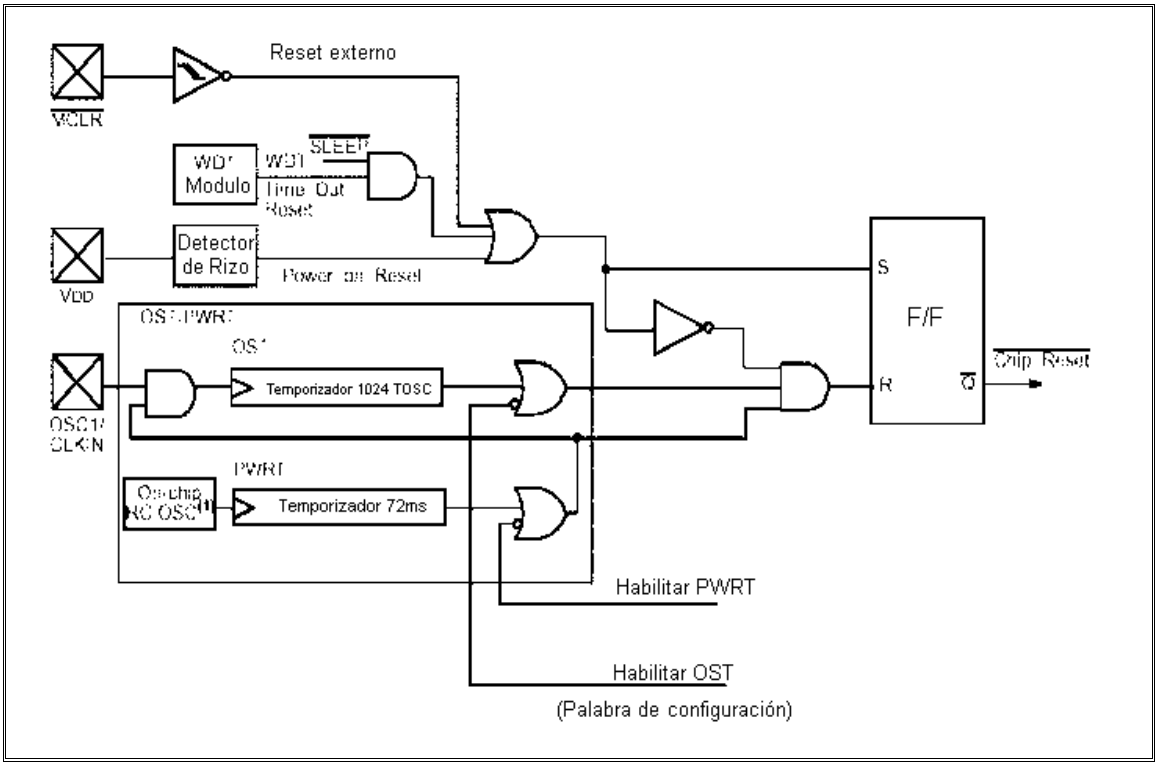


Figura 2.9.2.1. Diagrama simplificado a bloques del Reset.

El otro temporizador, OST, proporciona un retardo de 1,024 Tosc y tiene la misión de asegurar que el oscilador principal del procesador estabilice su frecuencia. OST comienza a funcionar cundo finaliza la temporización de PWRT.

2.9.3 Determinación del origen del reset

Dados los diferentes efectos que provoca el Reset según la causa que lo ha originado, es muy importante averiguar cuál es ésta. Para ello se utilizan dos bits del registro ESTADO, que ocupa la dirección 3 de los dos bancos de la RAM. Se trata del bit $\overline{TO}$ (Timer Out) que se activa con el desbordamiento del WDT y del bit $\overline{PD}$ (Power Down) que se activa cuando el PIC está en estado de reposo. En la tabla 2.9.3.1 se muestran los estados de estos dos bits, al aplicarse algún tipo de reset .

Tabla 2.9.3.1. Los bits $\overline{TO}$ y $\overline{PD}$ del registro de STATUS determinan la causa que ha originado el Reset

$\overline{TO}$	$\overline{PD}$	*Condición de Reset*
1	1	Por (Reset por conexión V_{DD})

0	1	Desbordamiento WDT en funcionamiento normal
0	0	Desbordamiento WDT en estado de reposo
1	1	Activación $\overline{MCLR}$ en funcionamiento normal
1	0	Activación $\overline{MCLR}$ en reposo

2.9.4 Reset por fallo en la alimentación

A veces es muy importante inicializar el procesador cuando se produce un fallo en la alimentación y el voltaje desciende por debajo de un valor determinado sin llegar a cero, recuperándose después. Para dicho proceso se propone el circuito de la figura 2.9.4.1.

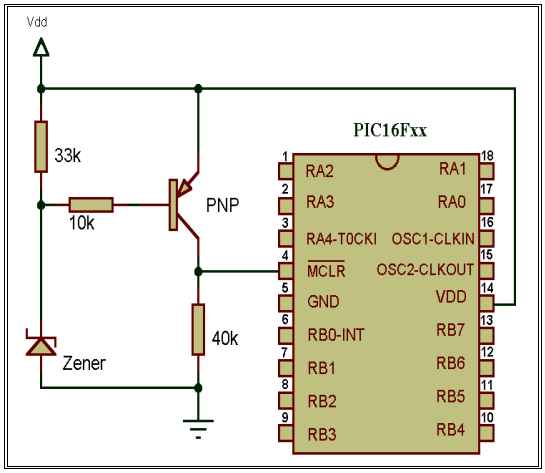


Figura 2.9.4.1. Circuito auxiliar de Reset

Cuando la tensión de alimentación desciende por debajo del valor Vzener + 0.7V, se produce un Reset por la activación del pin $\overline{MCLR}$.

2.10 Modo de reposo o bajo consumo de energía (Sleep)

El microcontrolador PIC16F84 puede estar mucho tiempo consumiendo una cantidad apreciable de energía eléctrica. El modo de reposo o de bajo consumo de energía consiste en reducir la corriente de alimentación, que típicamente oscila alrededor de los 2 miliamperios, y que puede llegar a menos de 10 microamperios, lo que permite alimentarlo con una pequeña pila durante casi dos años. Esta característica lo hace muy recomendable en aquellas aplicaciones en las que hay largos periodos de inactividad a la espera de que suceda algún acontecimiento asíncrono, como la pulsación de una tecla. En dicho estado se dice que el microcontrolador se ha "dormido".

Para dormir al microcontrolador basta ejecutar la instrucción SLEEP, entonces el primer elemento que se detiene es el oscilador principal, dejando de producirse los impulsos de trabajo a la frecuencia Fosc/4. En estas circunstancias el procesador no ejecuta instrucciones. Tampoco funciona el TMR0 al no recibir los impulsos internos. Para que el TMR0 no funcione con impulsos externos se lleva el pin RA4/T0CLKI al positivo. En este estado hay que destacar que el Perro Guardián sí queda en funcionamiento al entrar en el modo de reposo, se refresca o inicializa, pero sigue funcionando al disponer de un oscilador propio e independiente. En este caso los señalizadores del registro ESTADO quedarán $\overline{TO} = 1$ y $\overline{PD} = 0$.

Las líneas de E/S mantienen el estado anterior a la entrada en el modo de reposo, y las que no se hallan conectadas a periféricos y actúan como entradas de alta impedancia se recomienda conectarlas a positivo o a tierra, para evitar posibles fugas de corriente. Véase la tabla 2.10.1.

Tabla 2.10.1. Consecuencias del Modo de reposo

Estado Recurso	*Consecuencia*
Oscilador principal detenido	-No se ejecutan instrucciones -No funciona el TMR0 (RAM/T0CKI conectado a V_{DD})
WDT	-Se resetea pero sigue funcionando
Líneas E/S congeladas	-Mantienen el estado previo
Consumo mínimo	-Pasa de 2mA a menos de 10μA

2.10.1 Salida del estado de reposo

El PIC se “duerme” al ejecutarse la instrucción SLEEP en el programa y puede ser “despertado” para reanudar su funcionamiento normal por varias causas:

Por la activación del pin $\overline{MCLR}$

Por el desbordamiento del perro guardián

Cuando se genera una interrupción

Cuando el PIC16F84 está “dormido” y se activa el pin $\overline{MCLR}$, el microcontrolador se “despierta” y comienza a ejecutar la primera instrucción del programa, que reside en el Vector de Reset. Lo mismo sucede cuando se desborda el WDT: si el microcontrolador estaba dormido se despierta y se reinicializa.

Para determinar cuál de las dos causas ha originado el Reset se exploran los señalizadores $\overline{TO}$ y $\overline{PD}$ del registro STATUS, de acuerdo con los valores que se indican en la tabla 2.10.1.1.

Tabla 2.10.1.1. STATUS y significado de los señalizadores $\overline{PD}$ y $\overline{TO}$.

Valor del señalizador	*Indicación*
$\overline{PD}$ = 0	- Ejecución SLEEP. Modo de reposo
$\overline{PD}$ = 1	- Se ha despertado
$\overline{TO}$ = 0	- Se ha desbordado WDT
$\overline{TO}$ = 1	- No se ha desbordado WDT

Cuando se despierta el PIC se desarrolla la secuencia del temporizador OST que retarda 1024 Tosc el paso al funcionamiento normal, con objeto de dar tiempo a que se estabilice la frecuencia de funcionamiento.

2.10.2 Salida del estado de reposo mediante una interrupción

La tercera causa que puede hacer despertar al PIC sucede cuando se produce una interrupción. De las cuatro posibles interrupciones que admite el PIC16F84 la correspondiente al TMR0 no es posible, al carecer de entrada de impulsos, por lo tanto sólo son validas las otras tres interrupciones:

- Activación del pin RB0/INT
- Cambio de estado de uno de los pines RB7:RB4
- Fin de la escritura de la EEPROM

Cuando se ha ejecutado SLEEP, el PC se queda cargado con la dirección de la siguiente instrucción (PC + 1).

Para que se admita una de las tres interrupciones es necesario que sus bits de permiso (INTE, RBIE y EEIE) estén a 1. En este caso cuando se activa uno de los señalizadores de interrupción se explora el valor del bit GIE, que es el de Permiso Global de Interrupciones. Si GIE = 1 y se permiten globalmente las interrupciones, se despierta el PIC, se ejecuta la instrucción (PC + 1), que es la siguiente a SLEEP y, finalmente, se carga en el PC el valor 0004h, saltando a la rutina de interrupción. Como a veces no se desea ejecutar ninguna instrucción después de SLEEP y antes de saltar a la rutina de interrupción, se coloca una instrucción NOP.

Si se produce una interrupción permitida individualmente, pero el bit GIE = 0 (Prohibición Global), se despierta el PIC del estado de reposo y continúa la ejecución del programa a partir de la siguiente instrucción a SLEEP, como se muestra en la tabla 2.10.2.1.

Tabla 2.10.2.1. Instrucciones que ejecuta el PIC cuando "despierta" por una interrupción según el valor de bit GIE.

Estado de Bits	*Siguiente Instrucción*
GIE = 0 (Prohibición) RBIE = INTE = EEIE = 1	PC + 1 (Siguiente a SLEEP) PC + 2 (Siguiente)
GIE = 1 (Permiso) RBIE = INTE = EEIE = 1	PC + 1 (Siguiente a SLEEP) 0004 (Vector de interrupción)

Cuando GIE = 0 y se activa la bandera de una interrupción que tiene a 1 su bit individual de permiso, puede suceder uno de estos dos acontecimientos:

- Si la interrupción ocurre antes de la ejecución de la instrucción SLEEP, ésta se completará como si se tratase de una instrucción NOP. No se borra ni el WDT ni su divisor de frecuencia. El bit $\overline{TO}$ no se pone a 1 y el bit $\overline{PD}$ no se borra.

- Si la interrupción ocurre durante o después de la instrucción SLEEP, el microcontrolador se despertará inmediatamente, una vez ejecutada SLEEP. Se borra el WDT y su divisor de frecuencia y el bit $\overline{TO}$ = 1 y el bit $\overline{PD}$ = 0.

Para asegurar el borrado del WDT cuando se ejecuta una instrucción SLEEP se recomienda poner delante de ella una instrucción CLR WDT.

En la figura 46 se presenta un cronograma que muestra la forma de despertarse del PIC16F84 a través de una interrupción.

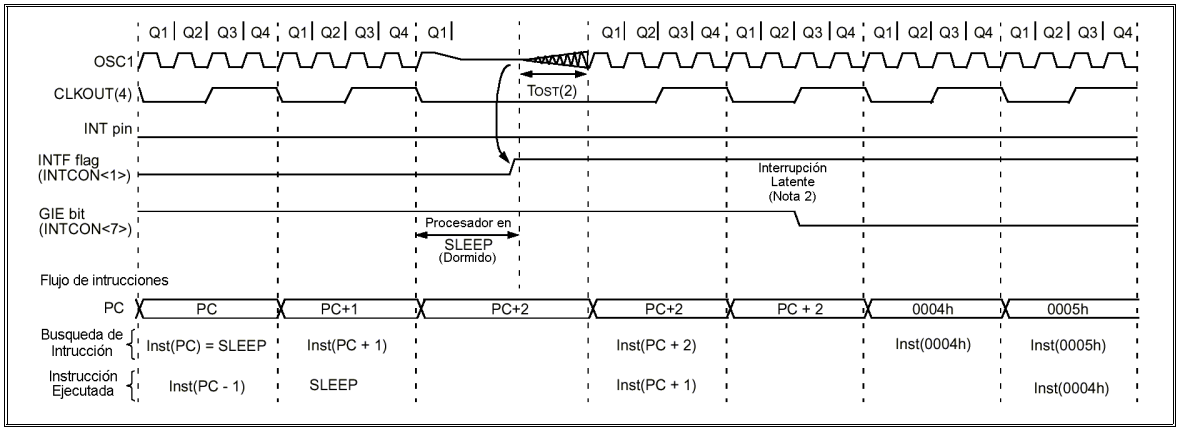


Figura 2.10.2.1. Cronograma del despertar del PIC al producirse una interrupción.

Notas:

1.- XT, HS o LP son los modos asumidos del oscilador
2.- TOST = 1024 TOST. Este retardo (delay) no sucede en modo RC.
3.- Si se supone que GIE = 1, al despertar el procesador salta a la rutina de interrupción. Si GIE = 0 la ejecución continua detrás de SLEEP.
4.- CLKOUT no se precisa en estos modos, pero se usa como referencia.

2.11 Tipos de osciladores

Los microcontroladores PIC disponen de un oscilador interno que es el que determina la frecuencia de los impulsos de reloj. Sin embargo, para controlar el valor de la frecuencia y estabilizarla, es necesario colocar algunos elementos externos en los pines 15 y 16 (OSC1/CLKIN y OSC2/CLKOUT). Según el tipo de circuitería externa los osciladores se clasifican en cuatro tipos:

- Oscilador RC
- Oscilador LP
- Oscilador XT
- Oscilador HS

Para iniciar el trabajo del microcontrolador hay que especificar el tipo de oscilador que se ha implementado. Para este fin existen dos bits FOSC1 y FOSC2 en una posición de la memoria de programa, llamada Palabra de Configuración, especificación que debe elegirse con la combinación correspondiente.

2.11.1 Oscilador RC

En este caso se coloca en el pin 16 del PIC16F84 (OSC1/CLKIN) una resistencia y un condensador. Este procedimiento es muy barato, pero la estabilidad de la frecuencia es mediocre y sólo se recomienda en sistemas en los que el tiempo no sea un factor crítico y la economía sea un objetivo prioritario.

Por el pin 15 (OSC2/CLKOUT) sale la cuarta parte de la frecuencia de oscilación (Fosc/4) que es el tiempo del ciclo de instrucción normal. La figura 2.11.1.1, muestra un oscilador de tipo RC.

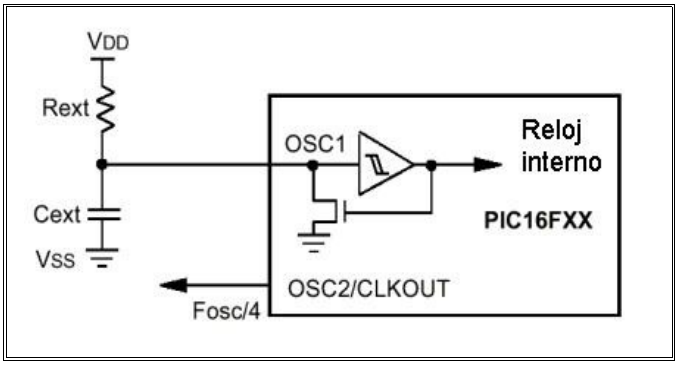


Figura 2.11.1.1. Oscilador de tipo RC.

En la tabla 2.11.1.1, se especifican los valores de capacitor y resistencia, recomendados por el fabricante para obtener una determinada frecuencia.

Valores recomendados: $5K\Omega \leq Rext \leq 100K\Omega$

$Cext > 20pF$

Tabla 2.11.1.1 . Valores recomendados para los componentes del oscilador RC con respecto a la frecuencia deseada

Fosc	Rext	Cext
625 KHz	10 KΩ	20pF
80 KHz	10 KΩ	220pF
80 Hz	10 KΩ	0.1pF

2.11.2 Osciladores de cristal / Resonadores cerámicos

En los modos XT, LP, y HS debe conectarse un cristal o un resonador cerámico en los pines OSC1/CLKIN y OSC2/CLKOUT para establecer la oscilación.

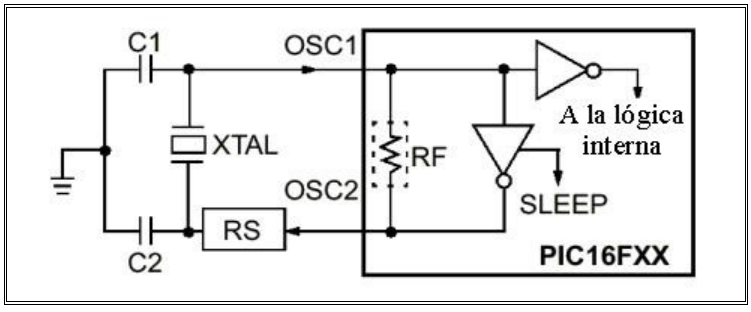


Figura 2.11.2.1. Oscilador de tipo HS, XT ó LP.

Oscilador de cristal/Resonador cerámico (configuración de oscilador XT, LP o HS). En la figura 2.11.2.1 se muestra el esquema de los osciladores HS, XT y LP que usan un cristal de cuarzo y dos condensadores de desacoplo, cuyos valores especifica el fabricante en función de la frecuencia.

Según el tipo de Oscilador y la frecuencia de trabajo, se emplean diferentes valores en los condensadores C1 y C2 que acompañan al cristal de cuarzo. Para frecuencias comprendidas entre 4 MHz y 10 MHz, los condensadores tienen una capacidad de 15 pF a 33 pF. La resistencia RS sólo es necesaria en algunas versiones tipo HS.

En la tabla 2.11.2.1, se muestran los valores de cristal y de capacitor especificado para cada tipo de oscilador.

Tabla 2.11.2.1. Tabla de selección de cristal y capacitor

Modo	*Frecuencia*	*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
	20 MHz	15 – 33 pF	15 – 33 pF

El oscilador diseñado requiere el uso de un cristal de corte paralelo. El uso de una serie de corte de cristal puede dar una frecuencia fuera de la especificada por el fabricante. En los modos XP, LP, HS, el dispositivo puede tener una fuente del reloj externa para manejar el pin OSC1/CLKIN, justo como se observa en la figura 2.11.2.2.

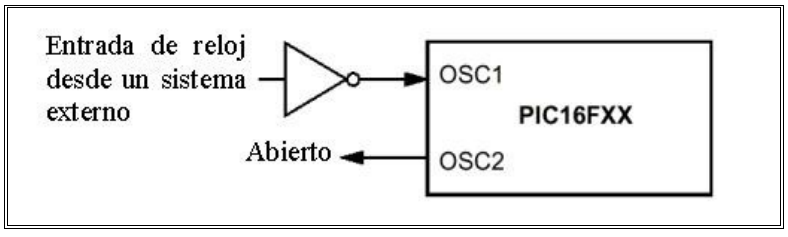


Figura 2.11.2.2. Entrada de reloj externa (configuración de oscilador HS, XT o LP)

2.11.3 Oscilador LP

Entre los pines 15 y 16 del microcontrolador se coloca un cristal de cuarzo de la frecuencia a la que se desee funcione. También puede instalarse en lugar del cristal de cuarzo un resonador cerámico. En cada uno de dichos pines se sitúa un condensador de desacoplo, Este tipo de oscilador está preparado para trabajar con bajas frecuencias, comprendidas entre 35 y 250 KHz. Su principal interés radica en el bajo consumo que necesita el PIC para trabajar a estas frecuencias tan bajas, detalle muy importante en los sistemas que se alimentan con pilas.

2.11.4 Oscilador XT

En este tipo de oscilador se coloca un cristal de cuarzo o un resonador cerámico entre los pines 15 y 16, junto a los correspondientes condensadores de desacoplo, cuyos valores se encuentran en la tabla 17, y dependen de la frecuencia seleccionada. El oscilador XT está preparado para trabajar en un rango de frecuencia de 100 KHz – 4 MHz. En la figura 2.11.4.1 se muestra la configuración típica para este tipo de oscilador

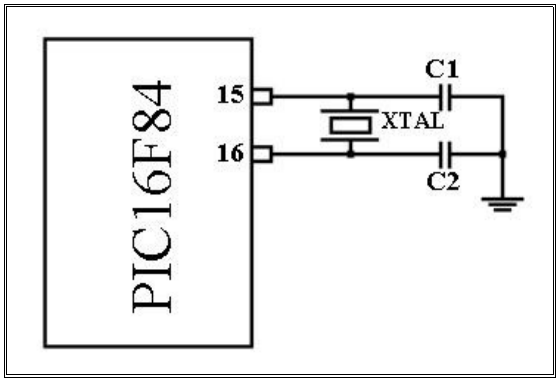


Figura 2.11.4.1. Circuito típico para el oscilador XT

2.11.5 Oscilador HS

Es un oscilador de alta velocidad y muy estable. Funciona en frecuencias comprendidas entre 4 y 20 MHZ. Utiliza un cristal de cuarzo o un resonador cerámico entre los pines 15 y 16, junto a dos condensadores de desacoplo. No todos los modelos de PIC admiten este tipo de oscilador. Debido su alta frecuencia de oscilación, el costo es muy elevado.

2.11.6 Otros circuitos osciladores

Empleando algunas puertas lógicas TTL muy sencillas pueden construirse osciladores de un inmejorable comportamiento y amplio rango de frecuencias. Hay dos tipos de osciladores con cristal: de resonancia en paralelo y de resonancia en serie

2.11.6.1 Osciladores de resonancia en paralelo

El circuito oscilador con resonancia en paralelo, como se muestra en la figura 2.11.6.1.1, se usa para la frecuencia del cristal que incorpora. La puerta inversora 74AS04 realiza una inversión de fase de 180°

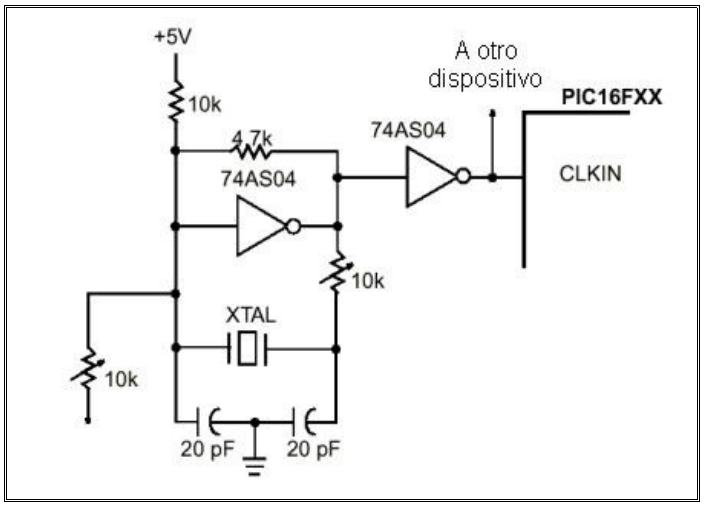


Figura 2.11.6.1.1. Oscilador de cristal y resonancia en paralelo

2.11.6.2 Oscilador de resonancia serie

El circuito oscilador resonante serie que se muestra en la figura 2.11.6.2.1 permite una enorme precisión en la frecuencia que, en general, es la que determina el cristal de cuarzo.

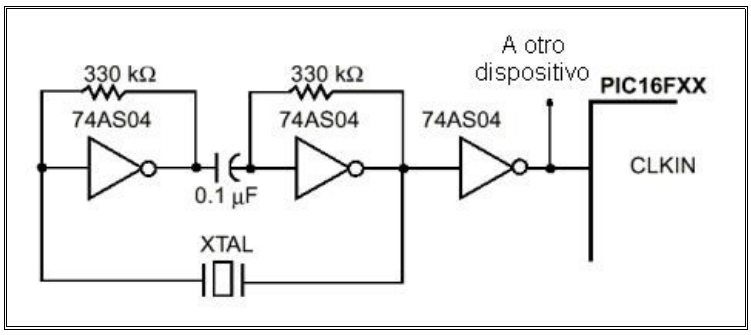


Figura 2.11.6.2.1. Oscilador de cristal y resonancia en serie

2.12 Palabra de configuración

La palabra de configuración ocupa una posición especial de la memoria de programa FLASH, la dirección 2007h, cuyos 14 bits regulan algunas funciones vitales del procesador. La organización de esta posición de la memoria se presenta en la figura 2.12.1.

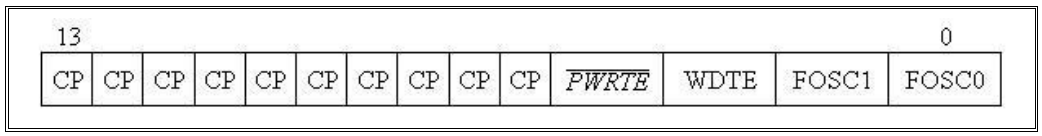


Figura 2.12.1. Estructura interna de la palabra de configuración

Los 10 bits de más peso, denominados todos ellos CP (Código de protección), se encargan de proteger el acceso a la memoria de código. Si CP es 0 el programa de instrucción no puede leerse, por el contrario, si CP es 1 el programa grabado en la FLASH se puede leer. Además el código de protección evita que pueda ser accedida la EEPROM de datos y, finalmente, si se modifica el bit CP de 0 a 1, se borra completamente la EEPROM.

Con los dos bits de menos peso, FSOC0 y FOSC1, se determina el tipo de oscilador que emplea el procesador, según la siguiente codificación mostrada en la tabla 2.12.2.

El bit WDTE es el que habilita o inhabilita el funcionamiento del perro guardián. Si WDTE es 1 el perro guardián entra en funcionamiento. Por último, el bit $\overline{PWRTE}$ es el de habilitación del temporizador PWRT, que retarda 72 ms el Reset para dar tiempo a estabilizar el voltaje de alimentación. Si $\overline{PWRTE}$ es 0, actúa el temporizador.

A continuación se muestra la descripción de cada uno de los bits que conforman la palabra de configuración del PIC16F84A.

CP: Bits de protección de la memoria de código

1 = No protegida

0 = El programa no se puede leer, evitando copias.

PWRTE: Activación del temporizador <<POWER-UP>>

1 = Desactivada

0 = Activado

WDTE: Activación del perro guardián (WDT)

1 = Activado el WDT

0 = Desactivado

FOSC1 – FOSC0: Selección del oscilador utilizado, el cual se muestran en la tabla 2.12.2

Tabla 2.12.2. Combinaciones para elegir el tipo de oscilador

FOSC1	*FOSC2*	*Oscilador*
1	1	RC
1	0	HS
0	1	XT
0	0	LP

Una de las pocas diferencias que tiene el PIC16C84 con el PIC16F84 es la actuación del bit PWRTE de la Palabra de Configuración, que tiene invertida su función en ambos. En el primero cuando este bit vale 1 el temporizador de Power-Up está activado,

mientras que sucede lo contrario en el segundo. En ambos microcontroladores el temporizador retrasa 72 ms el funcionamiento del microcontrolador al conectarle la alimentación para asegurar la estabilidad del voltaje aplicado.

2.13 Palabra de identificación (ID)

Son cuatro posiciones de la memoria FLASH que ocupan las direcciones comprendidas entre la 2000h y 2003h. Sólo son accesibles en lectura y escritura durante el proceso de grabación del programa.

De los 14 bits de las palabras de identificación sólo son validos los 4 de menos peso, y en ellos el diseñador puede escribir códigos de identificación, números de serie, datos sobre la fabricación, modelos, etc. En la figura 2.13.1 se destacan los bits válidos de cada posición.

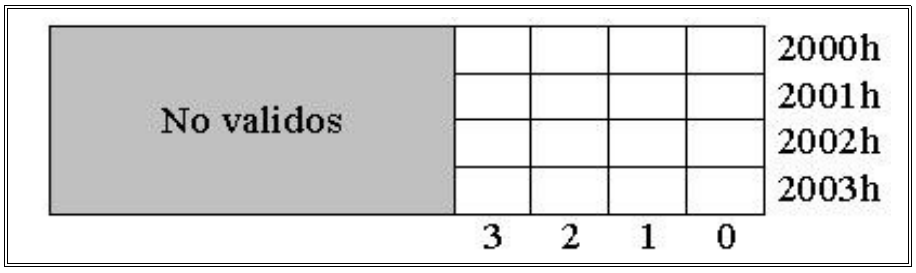


Figura 2.13.1. Palabras de identificación. Solo son validos los 4 bits de menos peso.

2.14 Circuitería fija

Para comenzar a funcionar el PIC16F84 precisa de tres recursos:

- Voltaje de alimentación
- Frecuencia de referencia
- Reinicialización o Reset

El voltaje de alimentación es imprescindible para el funcionamiento de todos los componentes electrónicos, y su valor puede estar comprendido entre +2 y +6 VDC, siendo +5 V la tensión típica, compatible con el resto de la circuitería del sistema. Este voltaje se aplica a los pines 15 y 5, positivo y tierra, respectivamente. El consumo de corriente depende de la frecuencia de funcionamiento y del voltaje de alimentación, no sobrepasando los 10 mA. La frecuencia de referencia sirve para estabilizar y fijar la frecuencia del oscilador interno, que determina la de trabajo del procesador. Se obtiene de un cristal de cuarzo situado entre los pines 16 (OSC1/CLKIN) y 15 (OSC2/CLKOUT).

Por último un microcontrolador debe disponer de diversas formas de inicializar su programa de trabajo. Para provocar un Reset externo se utiliza el pin 4 ($\overline{MCLR}/V_{pp}$), que también tiene la función de introducir la tensión Vpp de 12 a 14 VDC que se necesita durante el proceso de grabación del programa. En la figura 2.14.1 se muestran los cinco pines del PIC16F84 que soportan los tres recursos principales

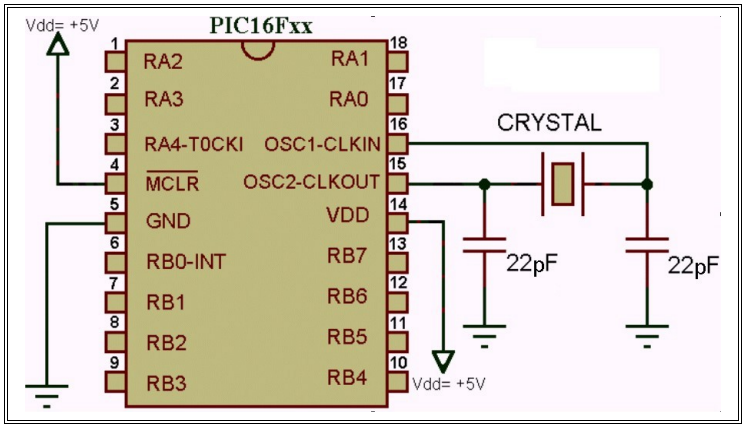


Figura 2.14.1. Se resaltan los cinco pines del PIC16F84 que soportan las tres funciones principales: alimentación, frecuencia de referencia y Reset.

2.15 Juego de instrucciones

Todos los modelos de PIC responden a la arquitectura RISC, que significa «Computadora de Juego de Instrucciones Reducido». No sólo implica que el número de instrucciones máquina que es capaz de interpretar y ejecutar el procesador es pequeño, como sucede con los PIC16X8X que constan de 35, sino también que posee las siguientes características. Las instrucciones son simples y rápidas. La falta de complejidad en la operación que realizan las instrucciones de los procesadores RISC permite que sean ejecutadas, mayoritariamente, en un solo ciclo de instrucción. Los PIC's tardan un ciclo en ejecutar la mayoría de las instrucciones, excepto las de salto, que tardan el doble. Las instrucciones son ortogonales. Apenas tienen restricciones en el uso de operando. Cualquier instrucción puede usar cualquier operando. La longitud de los datos y las instrucciones es constante. Todas las instrucciones tienen la misma longitud, 14 bits en los PIC16X8X, y todos los datos de un byte. La arquitectura Harvard del procesador aísla la memoria de instrucciones de la de datos, pudiendo tener diferente tamaño sus palabras.

2.15.1 Clasificación de formatos

Las instrucciones de los PIC de la gama media, entre los que se encuentran los modelos PIC16X8X, tienen 14 bits de longitud. Dicho formato se divide en diferentes campos de bits, cada uno de los cuales referencia a operandos o elementos que maneja la instrucción en la operación que realiza el procesador. A continuación se describen dichos campos.

a) *Campo de código OP*

   Los bits de este campo sirven para definir la operación que realiza la instrucción.

b) Campo de los operandos fuente (f) y destino (d)

Estos campos de bits definen los registros que actúan como operandos en la instrucción. Suelen referenciar la dirección que ocupan en la memoria de datos.

c) Campo de operando inmediato o literal (k)

Es un campo de bits que contiene el valor del operando inmediato.

d) Campo de referencia a un bit (b)

Suele ser un campo de 3 bit que indica la posición de un bit concreto dentro de un registro de 8 bits.

e) Campo de la dirección de salto

En las instrucciones de salto CALL y GOTO hay un campo de bits que contiene la dirección de la siguiente instrucción que hay que ejecutar. Dicho campo de bits se carga en el PC en las instrucciones de salto incondicional.

La clasificación de los formatos se muestra en la tabla 2.15.1.1.

Tabla 2.15.1.1. Clasificación de formatos

Símbolo	*Descripción*
f	Dirección del registro
W	Registro de trabajo
k	Campo que contiene un valor inmediato, que puede ser un operando (8 bits) o una dirección para el PC (11 bits).
x	Valor indeterminado de un bit. Puede ser 1 o 0.
d	Selección del destino. Cuando d=0, el resultado se almacena en W. Si d=1, el resultado se almacena en el archivo de registro f. Por default d=1.
label	Nombre de etiqueta
PC	Contador de Programa
PCLATH	Retenedor de Contador de Programa
GIE	Permiso global de interrupciones (Global Interrupt Enable bit)
WDT	Temporizador Perro Guardián (Watchdog Timer/Counter)
$\overline{TO}$	Bit Time-out
$\overline{PD}$	Bit Power-down
[ ]	Opciones
( )	Contenido
$\rightarrow$	Asignado a
< >	Campo de un bit de un registro. Por ejemplo: ESTADO <5>.
$\in$	En el conjunto de ejemplo: d $\in$ [0,1]

2.15.2 Instrucciones del PIC16F84A

En la tabla 2.15.2.1 se muestra el juego de 35 instrucciones del PIC16F84A.

Tabla 2.15.2.1. Juego de Instrucciones del PIC16F84A

Mnemonicos, Operandos		Descripción	Ciclos	Opcode 14 bits MSb LSb	Banderas Afectadas
ADDLW	k	Suma la literal con W	1	11 111x kkkk kkkk	C,DC,Z
ADDWF	f,d	Suma W y f	1	00 0111 dfff ffff	C, DC, Z
ANDLW	k	AND entre la literal y W	1	11 1001 kkkk kkkk	Z
ANDWF	f,d	AND W con f	1	00 0101 dfff ffff	Z
BCF	f,d	Limpia el bit f	1	01 00bb bfff ffff	
BSF	f,d	Pone a 1 el bit f	1	01 01bb bfff ffff	
BTFSC	f,d	Explora un bit de f y brinca si vale 0	1 (2)	01 10bb bfff ffff	
BTFSS	f,d	Explora un bit de f y brinca si vale 1	1 (2)	01 11bb bfff ffff	
CALL	k	Llamada a Subrutina	2	10 0kkk kkkk kkkk	
CLRF	f	Borra f	1	00 0001 1fff ffff	Z
CLRW	-	Borra W	1	00 0001 0xxx xxxx	Z
CLRWDT	k	Limpia Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$, $\overline{PD}$
COMF	f,d	Complementa f	1	00 1001 dfff ffff	Z
DECF	f,d	Decrementa f	1	00 0011 dfff ffff	Z
DECFSZ	f,d	Decrementa f y salta si es 0	1 (2)	00 1011 dfff ffff	
GOTO	k	Salto incondicional	2	10 1kkk kkkk kkkk	
INCF	f,d	Incrementa f	1	00 1010 dfff ffff	Z
INCFSZ	f,d	Incrementa f y salta si es 0	1 (2)	00 1111 dfff ffff	
IORLW	k	OR inmediata con W	1	11 1000 kkkk kkkk	Z
IORWF	f,d	Or entre W y f	1	00 0100 dfff ffff	Z
MOVF	f,d	Mueve f	1	00 1000 dfff ffff	Z
MOVLW	k	Mueve la literal a W	1	11 00xx kkkk kkkk	
MOVWF	f	Mueve W a f	1	00 0000 1fff ffff	
NOP	-	No opera	1	00 0000 0xx0 0000	
RETFIE	k	Retorno de interrupción (GIE = 1)	2	00 0000 0000 1001	
RETLW	k	Retorno de subrutina y carga W = k	2	11 01xx kkkk kkkk	
RETURN	k	Retorno de subrutina	2	00 0000 0000 1000	
RLF	f,d	Rota f a la izq. A través del acarreo	1	00 1101 dfff ffff	C
RRF	f,d	Rota f a la der. A través del acarreo	1	00 1100 dfff ffff	C
SLEEP	k	Paso a modo SLEEP	1	00 0000 0110 0011	$\overline{TO}$, $\overline{PD}$
SUBLW	k	Resta W de la literal	1	11 110x kkkk kkkk	C,DC,Z
SUBWF	f,d	Resta W y f	1	00 0010 dfff ffff	C,DC,Z
SWAPF	f,d	Intercambia nibbles en f	1	00 1110 dfff ffff	
XORLW	k	OR exclusiva entre la literal y W	1	11 1010 kkkk kkkk	Z
XORWF	f,d	OR exclusiva W con f	1	00 0110 dfff ffff	Z

2.15.2.1 Instrucciones que manejan registros

Responden a la sintaxis nemónico *f,d*, siendo *f* y *d* los dos operandos fuente y destino que se hallan implementados por los registros de 8 bits de la memoria de datos.

El registro *f* viene referenciado por la dirección de 7 bits que ocupa, mientras que el destino sólo por 1, que si vale 0 es el W y si vale 1 es el fuente. En la tabla 2.15.2.1.1 se muestran las instrucciones de este grupo con sus características más interesantes. Se han excluido las de salto condicional que tienen el mismo formato.

Tabla 2.15.2.1.1. Principales características de las instrucciones de los PIC16X8X .

INSTRUCCIONES QUE MANEJAN REGISTROS				
SINTAXIS	OPERACIÓN	CICLOS	FORMATO 14 BITS	SEÑALIZADORES
ADDWF f,d	Suma W y *f*	1	00 0111 dfff ffff	C,DC,Z
ANDWF f,d	AND W y f	1	00 0101 dfff ffff	Z
CLRF f	Borra f (pone todos los bits a 0)	1	00 0001 dfff ffff	Z
CLRW ---	Borra W	1	00 0001 0xxx xxxx	Z
COMF f,d	Complementa f (Invierte)	1	00 1001 dfff ffff	Z
DECF f,d	Decrementa f	1	00 0011 dfff ffff	Z
INCF f,d	Incrementa f	1	00 1010 dfff ffff	Z
IORWF f,d	OR entre W y f	1	00 0100 dfff ffff	Z
MOVF f,d	Mueve f	1	00 1000 dfff ffff	Z
MOVWF f	Mueve f	1	00 0000 dfff ffff	----
NOP ---	No opera	1	00 0000 0xx0 0000	----
RLF f,d	Rota f a la izq. A través del acarreo	1	00 1101 dfff ffff	C
RRF f,d	Rota fa la der. A través del acarreo	1	00 0100 dfff ffff	C
SUBWF f,d	Resta W a f	1	00 0110 dfff ffff	C,DC,Z
SWAPF f,d	Intercambia nibbles	1	00 1110 dfff ffff	----
XORWF	XOR de W con f	1	00 0110 dfff ffff	Z

2.15.2.2 Instrucciones que manejan bits

Sólo hay dos instrucciones en este grupo, pero son muy flexibles. Una de ellas pone a 1 (*bsf*) cualquier bit de un registro, mientras que la otra pone a 0 (*bcf),* tal como se muestra en la tabla 2.15.2.2.1.

Tabla 2.15.2.2.1. Características más importantes de las dos instrucciones que manejan un bit determinado de un registro.

INSTRUCCIONES QUE MANEJAN BITS				
SINTAXIS	OPERACIÓN	CICLOS	FORMATO 14 BITS	SEÑALIZADORES
BCF f,d	Borra bit de f	1	01 00bb bfff ffff	----
BSF f,d	Pone a 1 el bit de f	1	01 01bb bfff ffff	----

2.15.2.3 Instrucciones de brinco (Skip)

Sólo hay 4 instrucciones de salto condicional en los PIC de la gama media. Dos de ellas examinan un bit de un registro y según valga 1 o 0, brincan o no. Recuérdese que un brinco es un «salto» pequeño, sólo se salta la instrucción siguiente a la condicional. Las otras dos instrucciones incrementan o decrementan un registro y la posibilidad del brinco se efectúa si con esa operación el valor del registro ha llegado a cero. Cuando estás instrucciones no brincan por que no se cumple la condición, tardan un ciclo de instrucción en ejecutarse. Para el caso en que brinquen tardan el doble, tal como se muestra en la tabla 2.15.2.3.1.

Tabla 2.15.2.3.1. Características más relevantes de las cuatro instrucciones condicionales de brinco.

INSTRUCCIONES DE BRINCO				
SINTAXIS	**OPERACIÓN**	**CICLO**	**FORMATO 14 BITS**	**SEÑALIZADORES**
BTFSC f,d	Explora un bit de f y brinca si vale 0	1(2)	01 10bb bfff ffff	----
BTFSS f,d	Explora un bit de f y brinca si vale 1	1(2)	01 11bb bfff ffff	----
DECFSZ f,d	Decrementa f y si es 0, brinca	1(2)	00 1011 dfff ffff	----
INCFSZ f,d	Decrementa f y si es 1, brinca	1(2)	00 1111 dfff ffff	----

2.15.2.4 Instrucciones que manejan operandos inmediatos

Se trata de media docena de instrucciones que realizan una operación con un valor inmediato de 8 bits que se proporcionan dentro del formato de la instrucción, el cual sólo tiene dos campos: el del Código OP (6 Bits) y el de operando inmediato (8 bits), como se puede observar en la tabla 2.15.2.4.1.

Tabla 2.15.2.4.1. Características más importantes de las instrucciones que manejan operandos inmediatos (k).

INSTRUCCIONES QUE MANEJAN OPERANDOS INMEDIATOS				
SINTAXIS	**OPERACIÓN**	**CICLOS**	**FORMATO 14 BITS**	**SEÑALIZADORES**
ADDLW k	Suma inmediata con W	1	11 111x kkkk kkkk	C,DC,Z
ANDLW k	AND inmediato con W	1	11 1001 kkkk kkkk	Z
IORLW k	OR inmediato con W	1	11 1000 kkkk kkkk	Z
MOVLW k	Mueve a W un valor inmediato	1	11 00xx kkkk kkkk	----
SUBLW k	Resta W de un inmediato	1	11 110x kkkk kkkk	C,DC,Z
XORLW k	OR exclusiva con W	1	11 1010 kkkk kkkk	----

2.15.2.5 Instrucciones de control y especiales

En este grupo se incluyen las instrucciones que rompen la secuencia normal del programa porque alteran el contenido del PC y también las instrucciones especiales.

La instrucción de salto incondicional *goto* carga en el PC la dirección de la nueva instrucción. La instrucción *call* de llamada a Subrutina, antes de cargar el PC con la dirección de la instrucción a saltar, salva la dirección de la partida guardando en la cima de la pila el valor actual del PC. De esta manera, al retornar de la subrutina se saca de la pila la dirección de regreso del programa principal.

Para realizar un retorno de una subrutina se pueden emplear dos instrucciones. La más habitual es *return,* que se limita a extraer de la cima de la pila el valor que carga en el PC. Otra más completa es *retlw k*, que además de hacer lo mismo que *return*, carga en W el valor inmediato que contiene *k*. Es decir, devuelve un parámetro desde la subrutina.

Para el final de las interrupciones hay otra instrucción cuyo nemónico es *retfie*. La operatividad de esta instrucción consiste en cargar en el PC el contenido de la cima de la pila y poner el bit GIE = 1, pues al comenzar la interrupción este bit se pone automáticamente a 0 para evitar que cuando se atiende una interrupción se produzca otra. GIE es el bit de permiso de todas las interrupciones.

En cuanto a las interrupciones especiales, se han incluido dos en este grupo: *clrwdt* y *slepp*. La primera pone a 0 el contenido del Perro Guardián, es decir, lo refresca o lo reinicializa. El Perro Guardián si se desborda (pasa de 0xff a 0x00) provoca un Reset. La instrucción *clrwdt* hay que colocarla estratégicamente en ciertos puntos del programa para evitar la reinicialización.

La instrucción *slepp* introduce al procesador en un modo de funcionamiento que se llama de Reposo o de Bajo Consumo. Detiene el oscilador y el procesador queda congelado, no ejecutando instrucciones y manteniendo el mismo valor los Puertos de E/S. También pone los bits $\overline{PD}$ = 0 y $\overline{TO}$ = 1 y borra al Perro Guardián y al Divisor de frecuencia, como se muestra en la tabla 2.15.2.5.1.

Es recomendable incluir la instrucción *clrwdt* para que no se desborde el WDT (Watch Dog Timer o Perro Guardián). Las instrucciones de salto cuentan el doble de ciclos que las instrucciones normales.

Tabla 2.15.2.5.1. Principales características de las instrucciones del control del flujo del programa y de las especiales.

INSTRUCCIONES QUE MANEJAN OPERANDOS INMEDIATOS				
SINTAXIS	**OPERACIÓN**	**CICLOS**	**FORMATO 14 BITS**	**SEÑALIZADORES**
CALL k	Llamada a subrutina	2	10 0kkk kkkk kkkk	TO#,PD#
CLRWDT	Borra o refresca el PeroGuardián	1	00 0000 0110 0100	-----
GOTO k	Salto incondicional	2	10 1kkk kkkk kkkk	-----
RETFIE	Retorno de interrupción (GIE = 1)	2	00 0000 0000 1001	-----
RETLW k	Retorno subrutina y carga W = k	2	11 01xx kkkk kkkk	-----
RETURN	Retorno de subrutina	2	00 0000 0000 1000	-----
SLEPP	Pasa al modo de reposo	1	00 0000 0110 0011	TO#,PD#

CAPÍTULO 3. COMPILACIÓN Y SIMULACIÓN DE PROGRAMAS

3.1 Uso del MPLAB

3.1.1 Instalación

El primer paso a seguir es poseer el programa MPLAB, para esto, es necesario descargar el archivo de instalación de www.microchip.com . Todas las utilidades y herramientas de trabajo estan contenidas en un mismo archivo de instalación, sin embargo, existe la posibilidad de descargar este mismo paquete en varios archivos comprimidos para facilitar su transportación en discos flexibles.

Los requerimientos mínimos para la instalación de este programa son:

- Procesador 386, 486 o Pentium.
- Windows 3.1/ 95/ 98, Windows NT 3.51/4.0, Windows 2000 ,MAC OS 7.0, ó Unix compatible OS.
- 16 MB de memoria RAM para sistema con Windows 95.
- 24 MB de RAM para Windows NT.
- 32 MB para sistemas con Windows 2000.

En caso de que se descargue el programa en varios archivos comprimidos para su transferencia a Disquettes, se debe descomprimir cada uno de ellos con la herramienta Winzip y colocarse en un mismo directorio, por ejemplo C:\MPLAB. Dicho directorio o carpeta podrá ser eliminado por completo cuando la instalación se ha terminado.

Una vez que se ha descargado el archivo completo o que se han descomprimido todos los archivos del conjunto de instalación, se debe dar doble click al archivo de nombre "MPxxxxx.exe" para ejecutar la aplicación que nos guiará a través del proceso. A continuación, aparecerá un cuadro de diálogo que indica la bienvenida a la aplicación en curso y la versión del programa a instalar, tal como se muestra en la figura 3.1.1.1.

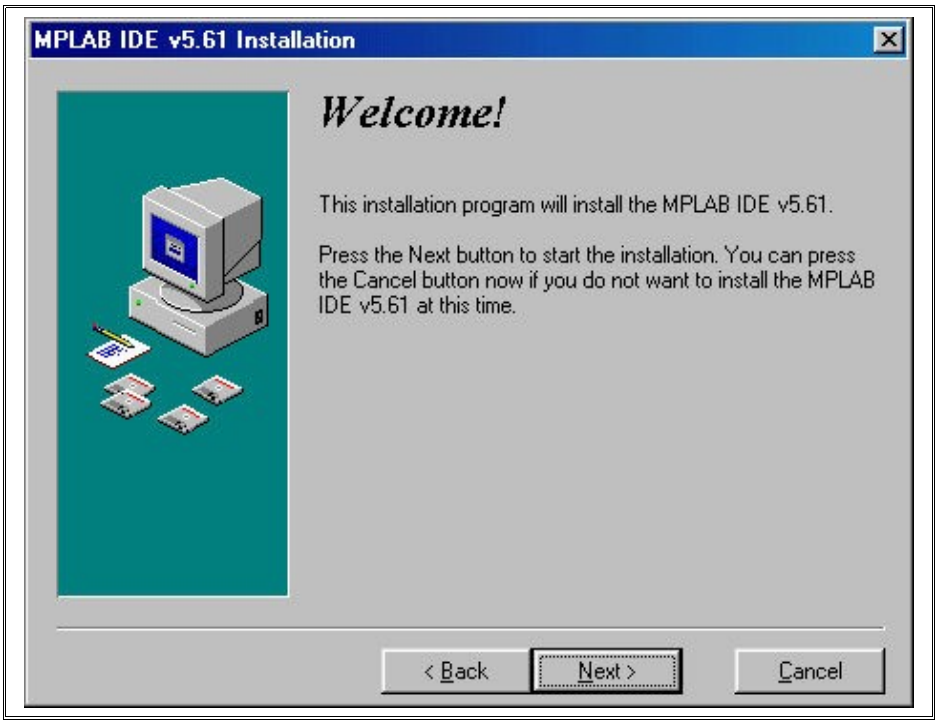


Figura 3.1.1.1. Mensaje de bienvenida a la instalación del MPLAB.

Después de dar un click en el botón etiquetado como "Next" aparecerá el cuadro de diálogo que muestra las términos y condiciones que el fabricante impone para ser aceptadas si se desea instalar el paquete del MPLAB, tal como se muestra en la figura 3.1.1.2. Para proseguir con la instalación se debe activar la opción "I agree", seguido de un click en el botón identificado por la etiqueta "Next".

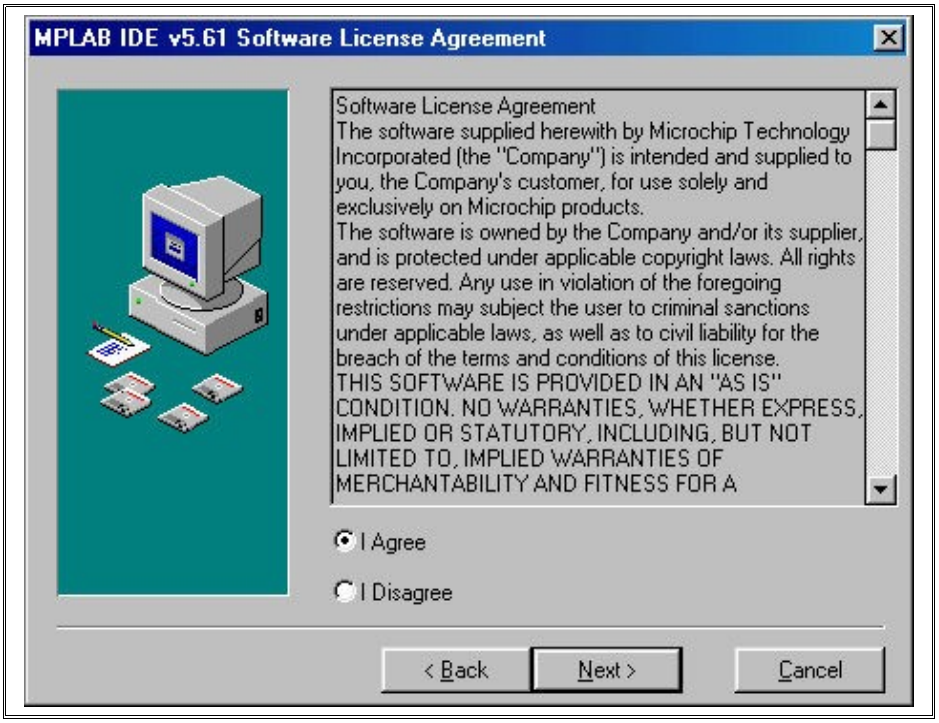


Figura 3.1.1.2. Términos y condiciones impuestas por Microchip.

A continuación aparecerá el cuadro de diálogo correspondiente a la selección de los módulos del MPLAB, tal como se muestra en la figura 3.1.1.3; No es necesario instalar todos los módulos, ya que algunos corresponden al soporte a ciertos dispositivos programadores y herramientas de desarrollo como PICSTART, PICMASTER, MPLAB-ICE, ICEPIC ó SIMICE;

Por ejemplo, el módulo llamado “MPLAB IDE Files” contiene los archivos relacionados con el entorno de desarrollo en sí, que es a lo que hacen referencia las siglas IDE (Integrated Development Enviroment);

El módulo “MPASM/MPLINK/MPLIB Files” contiene el ensamblador, cuya función es traducir los programas al leguaje del microcontrolador; el “linker”, que permite que los subprogramas ubicados en varios archivos funcionen como una sola unidad coherente y las librerías del MPLAB que facilitan la labor del programador al proporcionar fragmentos de programas ya elaborados y muy utilizados que se pueden incluir dentro de un determinado código.

El simulador que proporciona Microchip está contenido en el Módulo "MPLAB-SIM", el cual es una herramienta para comprobar el correcto funcionamiento de las instrucciones de un programa al ser ejecutadas.

Los archivos de soporte necesarios para la función de depuración están contenidos en el módulo "MPLAB-ICD Debugger Suport Files". Dicha función permite ver los resultados finales del programa y acceder a los registros del sistema según se va ejecutando la aplicación o bien, ejecutar las instrucciones paso a paso.

El módulo "Help Files" contiene los archivos necesarios para obtener ayuda de las funciones que integran el MPLAB, como son los menús, las ventanas, etc.

Para continuar el proceso de instalación, se debe dar click en el botón "Next".

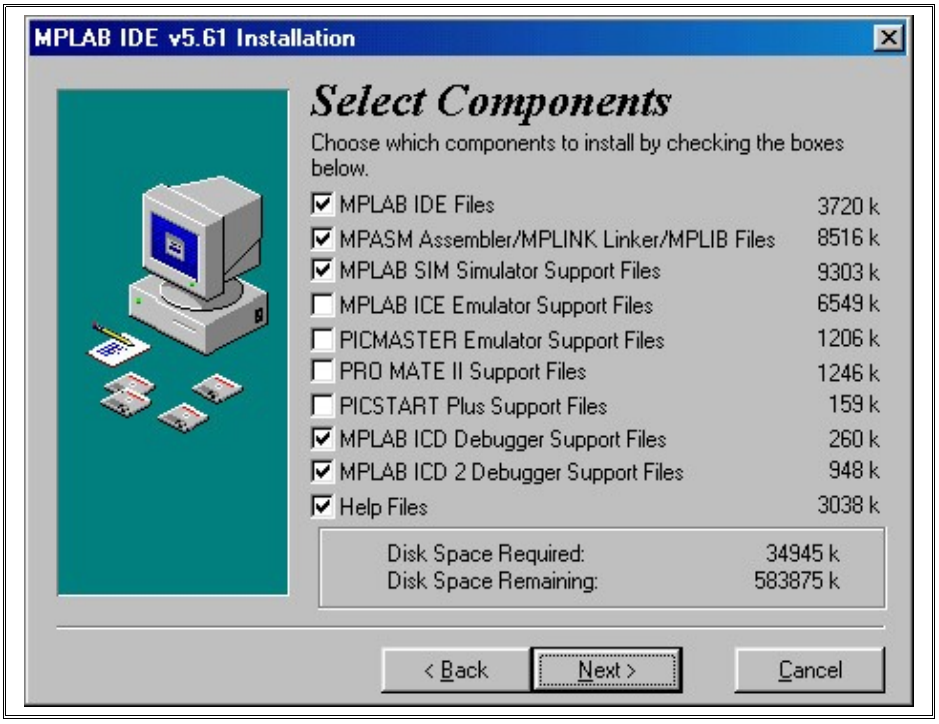


Figura 3.1.1.3. Selección de los Módulos del MPLAB

El siguiente cuadro de diálogo corresponde a la selección de componentes del lenguaje de compilación del MPLAB. Se recomienda dejar activadas las casillas que se muestran en la figura 3.1.1.4, ya que no se trabajará en un sistema operativo inferior a Windows 95. A continuación se debe dar click en el botón "Next".

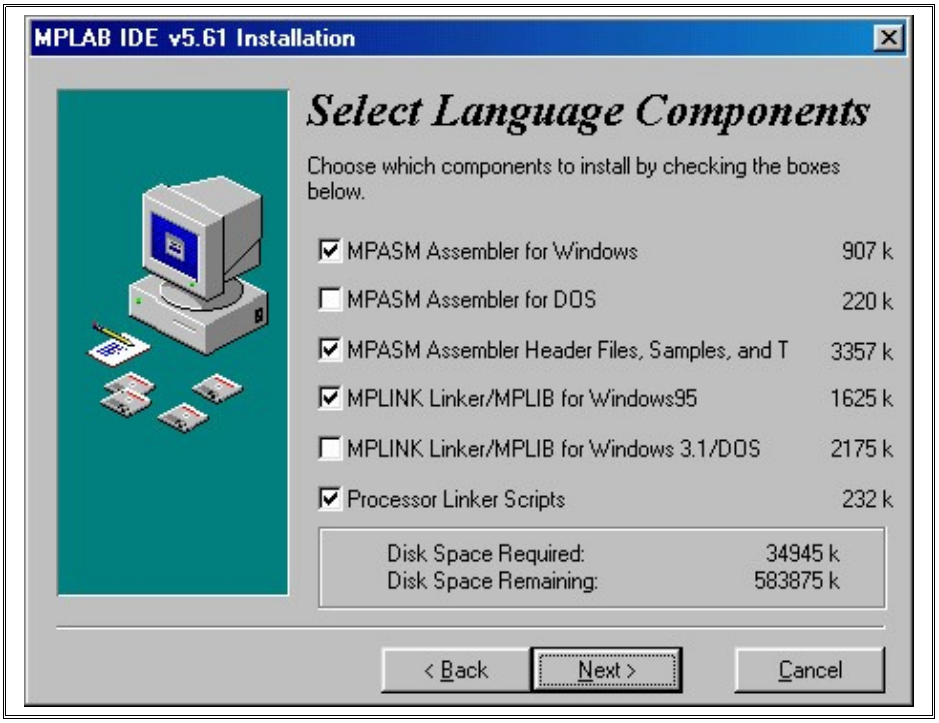


Figura 3.1.1.4. Selección de los componentes del lenguaje MPLAB.

Uno de los últimos pasos de este proceso consiste en especificar el directorio donde se desea instalar el programa MPLAB como se observa en la figura 3.1.1.5;

Se recomienda instalar el paquete en la ruta que se sugiere por defecto, la cual es C:\Archivos de programa\MPLAB.

En dicho directorio se podrán almacenar los proyectos creados, así como los programas construidos en cada uno de ellos. Al igual que en los cuadros anteriores, se debe dar click en el botón “Next” para continuar con la instalación.

En la siguiente pantalla se ofrece la opción de hacer una copia de seguridad de los archivos existentes en el directorio y que posiblemente sean reemplazados, para continuar con la instalación se debe elegir la opción más conveniente y dar click en el botón “Next”.

La opción de introducir accesos directos del programa al menú de inicio y al escritorio se presenta en la siguiente pantalla; en este caso, el usuario deberá elegir la opción que más le convenga y dar click en “Next”.

Figura 3.1.1.5. Selección del directorio de instalación

Una vez instalado el MPLAB, se puede ejecutar este programa dando un click al Botón de "Inicio", después al submenú "Programas", a continuación al Submenú "Microchip MPLAB" y por último al acceso directo del programa, etiquetado como MPLAB.

La forma más fácil y directa de ejecutar el programa es dando un doble click al icono del MPLAB que representa su acceso directo y que está situado en el escritorio de Windows, dicho icono se muestra en la figura 3.1.1.6.

Figura 3.1.1.6. Icono del MPLAB

La apariencia del programa MPLAB es similar a cualquier otro programa diseñado para el ambiente gráfico de Windows. Contiene una barra de título que muestra el icono de la aplicación y el nombre del programa, una barra de menú, una barra de herramientas, botones de control y una barra de estado, tal como se muestra en la figura 3.1.1.7.

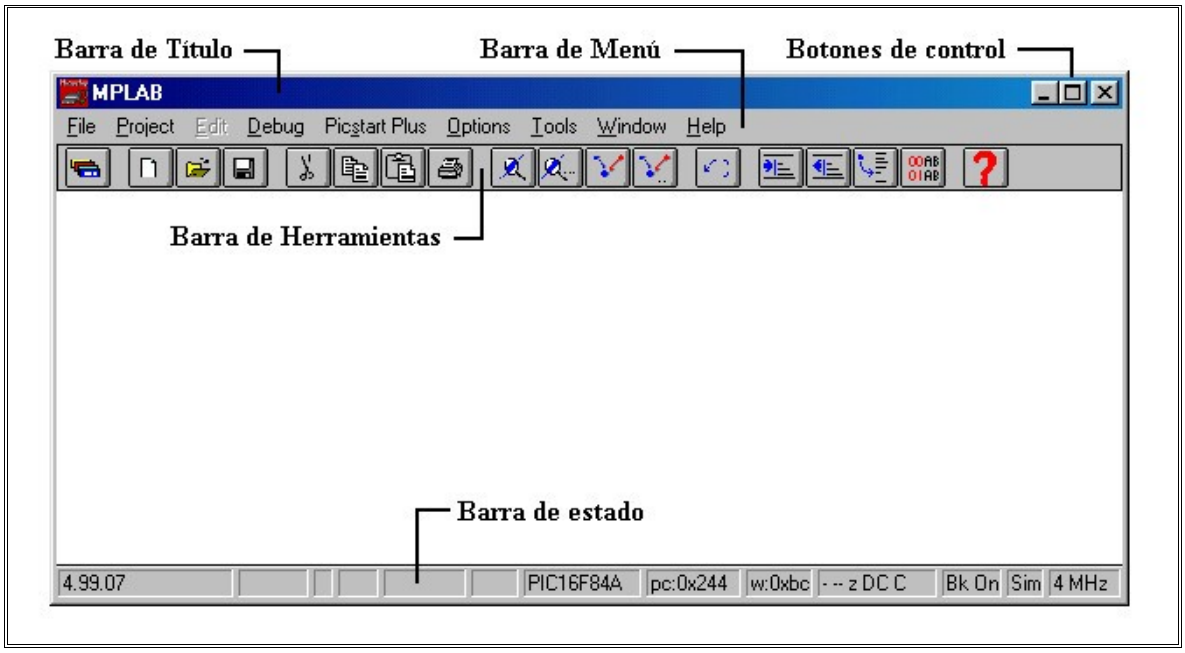


Figura 3.1.1.7. Aspecto de la aplicación MPLAB

3.1.2 Edición de programas

Antes de editar un programa es necesario configurar el modo de desarrollo, lo cual se puede hacer accediendo al menú Opciones (options) y a continuación dando un click en "Development Mode" (Modo de Desarrollo), apareciendo un cuadro de diálogo como el que se muestra en la figura 3.1.2.1.

Mediante la pestaña "Tools" del cuadro de diálogo "Development Mode" es posible configurar al MPLAB como solo editor o como Simulador. En ambos caso se debe seleccionar el tipo de microcontrolador que se va utilizar de acuerdo al programa a desarrollar.

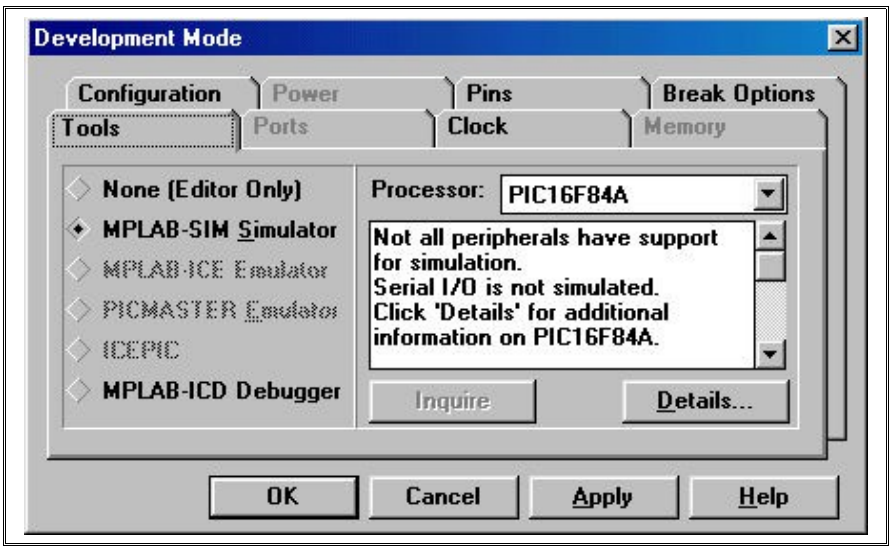


Figura 3.1.2.1. Configuración del modo de desarrollo

La función de Simulador deberá ser seleccionada, así como el PIC16F84A ubicado en la lista desplegable correspondiente a los dispositivos que soporta el programa MPLAB. Una vez que se ha especificado el modo de operación y el dispositivo a emplear, se debe dar click en el botón etiquetado como "OK".

El MPLAB, al igual que muchos otros programas, trabaja mediante proyectos, por lo tanto, para trabajar con un archivo, es necesario que esté incluido en un proyecto.

El primer paso para editar en proyecto es accediendo al menú "Project" y seleccionando la opción "New Project".

A continuación aparecerá un cuadro de diálogo como el que se muestra en la figura 3.1.2.2; en el cuadro de texto "File Name" se debe escribir el nombre del proyecto que se desea crear.

En la estructura de carpetas de la parte derecha se debe indicar la ubicación en el disco duro donde se guardará el nuevo proyecto. Nótese que la extensión de un proyecto de MPLAB es "*.pjt". A continuación, se debe dar click en el botón "OK".

Después de esto, aparecerá el cuadro de diálogo de "Edit Project", al cual por el momento no se le modificará ninguna opción, solo deberá de darse click en el botón "OK" para culminar la creación de este nuevo proyecto.

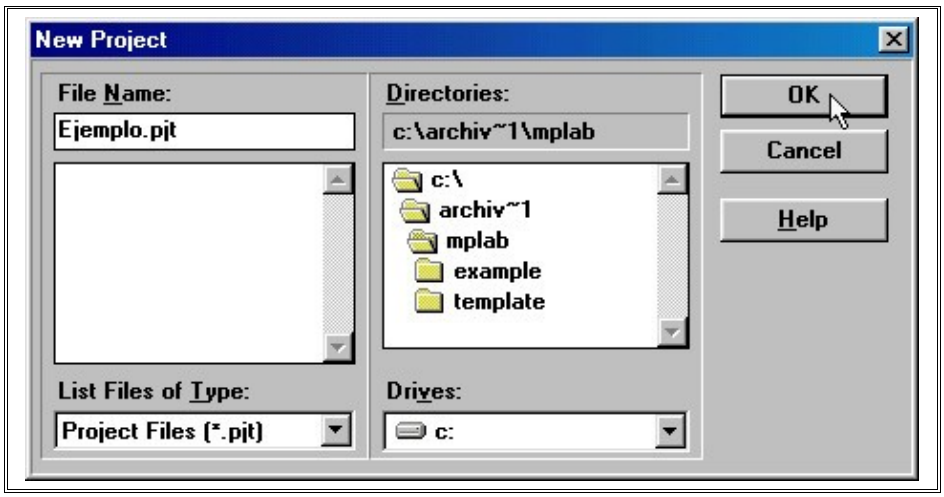


Figura 3.1.2.2. Creación de un nuevo proyecto

Una vez que se ha creado un proyecto, se puede crear un nuevo archivo mediante el menú “File” y seleccionando la opción “New”. A continuación aparecerá un cuadro de texto en el cual se podrá editar el programa deseado, justo como se muestra en la figura 3.1.2.3.

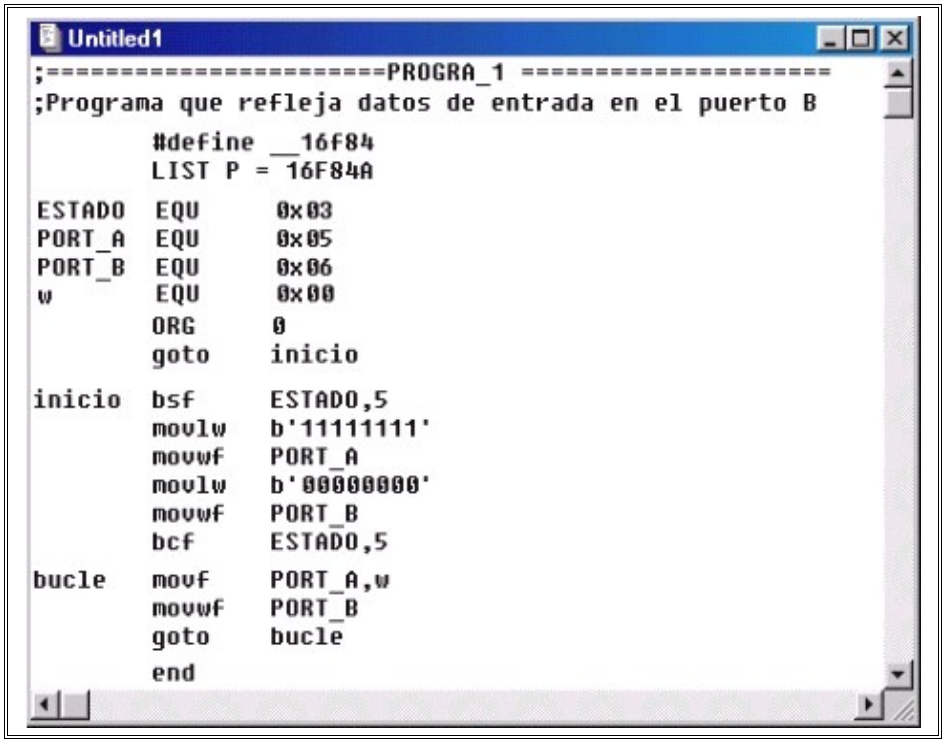


Figura 3.1.2.3. Ventana de edición para el programa.

Se debe tener en cuenta que la primera columna del cuadro de texto está reservada para las etiquetas que son nombres simbólicos que se asignan a una dirección de memoria.

Dichas etiquetas deben empezar por una letra, deben empezar en la primera posición de la columna uno, pueden incluir caracteres alfanuméricos, líneas de subrayado e interrogación, además su longitud no debe exceder de 31 caracteres y no pueden usarse expresiones que ya utiliza el ensamblador, tales como instrucciones, directivas del propio ensamblador, nombres de registros especiales (SFR) o el nombre de cada uno de los bits de los registros especiales.

Por ejemplo, cuando se escribe la palabra “inicio”, se está asignando el símbolo de inicio a un valor propio del microcontrolador, que es la dirección de memoria correspondiente donde quedará grabada esa instrucción.

En la segunda columna se puede comenzar a escribir la operación o el mnemónico de la instrucción del microcontrolador o las directivas del ensamblador; normalmente se ocupa una tabulación mediante la tecla "Tab" para que el cursor salte a la segunda columna y se pueda empezar a escribir la instrucción.

En la tercera columna se escribe o indican los operandos de la instrucción, que son los registros o cantidades sobre las que se realizan las operaciones o instrucciones. Un operando puede ser un registro de la memoria de datos o un valor constante que comúnmente se conoce como literal. Cuando se utilizan dos operandos, el primero es el operando fuente y el segundo es el operando destino.

Cuando se desea realizar un comentario, se debe escribir un símbolo de punto y coma (";") seguido de la indicación deseada. El punto y coma es un delimitador y hace que el ensamblador ignore la línea de texto que se encuentra a la derecha de él.

Una vez que se ha completado el programa, o cuando el usuario lo decida, se debe guardar el archivo mediante el menú “File”, y la opción “Save As”.

Después de esto, aparecerá el cuadro de diálogo de “Save file as”. En el cuadro de texto “File Name” se debe especificar el nombre con que se desea guardar el archivo. Nótese que la extensión del archivo en ensamblador debe ser “*.asm”. En la estructura de carpetas de la derecha se debe elegir la ubicación en la que se podrá encontrar el archivo para modificaciones posteriores. A continuación se debe dar un click en el botón “OK”, justo como se muestra en la figura 3.1.2.4.

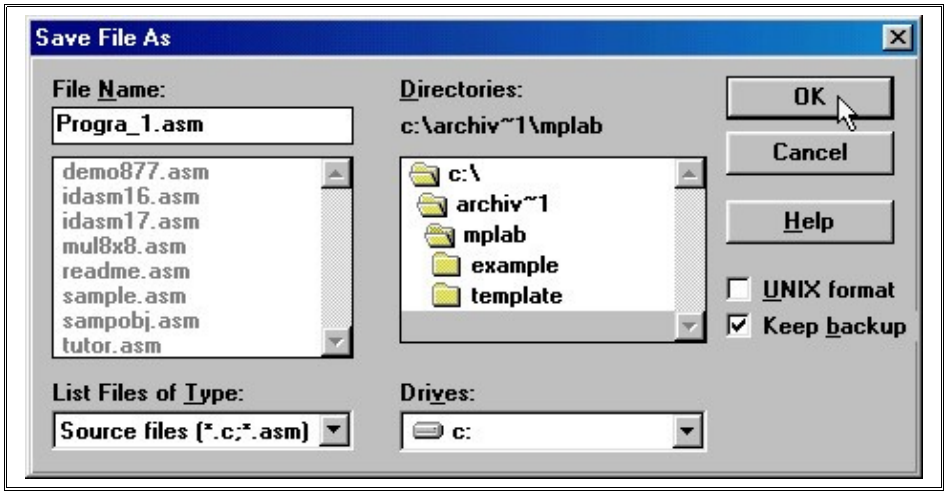


Figura 3.1.2.4. Cuadro de diálogo para guardar un archivo

Una vez que el archivo se ha guardado, es necesario incorporarlo al proyecto de la siguiente manera. Mediante el menú "Project" y la opción "Edit Project", debe accederse al cuadro de diálogo de "Edit Project" como se muestra en la figura 3.1.2.5. El siguiente paso es dar un click en el botón etiquetado como "Add Node ..." que nos permitirá incorporar al proyecto el archivo que se acaba de editar.

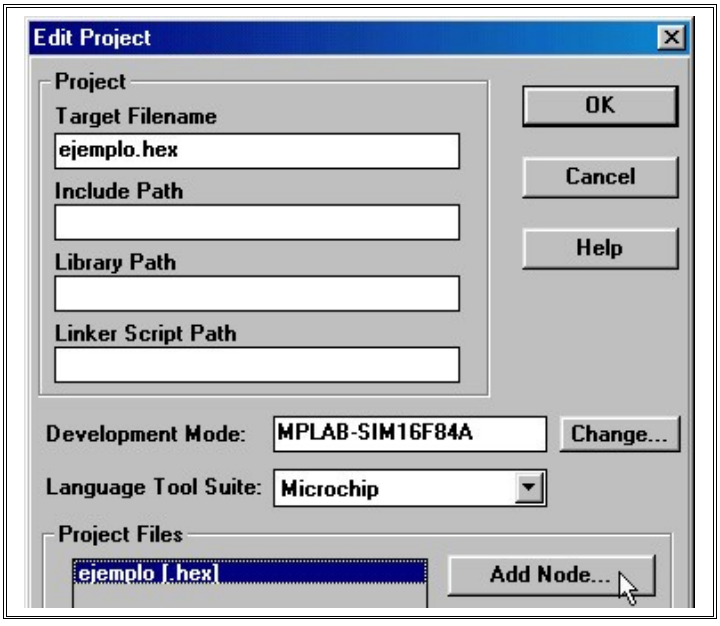


Figura 3.1.2.5. Incorporación de un archivo a un proyecto

El siguiente paso es seleccionar el archivo que contiene el programa editado y dar click en el botón "Aceptar" del cuadro de diálogo "Add Node" que se muestra en la figura 3.1.2.6. Después de esto, aparecerá automáticamente el cuadro de diálogo "Edit Project".

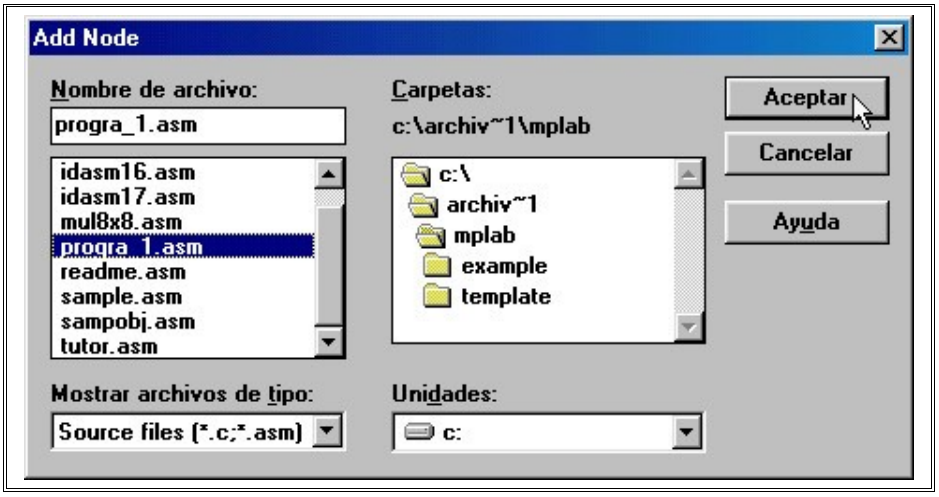


Figura 3.1.2.6. Cuadro de diálogo "Add Node"

Una vez que se ha agregado el archivo que contiene el programa al proyecto, se visualizará su nombre en el margen etiquetado como "Project Files", para finalizar se deberá dar click en el botón "OK", justo como en la figura 3.1.2.7.

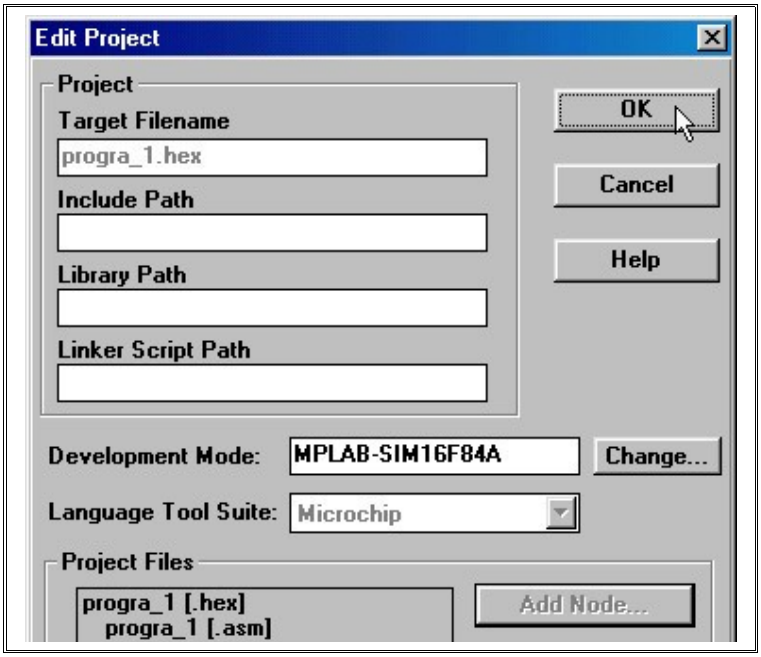


Figura 3.1.2.7. Adición de un archivo al proyecto

3.1.3 Compilación de programas

Compilar un programa consiste en un proceso de conversión desde un código fuente a un lenguaje máquina, que en este caso corresponde al microcontrolador PIC16F84A.

Para realizar la compilación se necesita abrir el proyecto que contiene el archivo a compilar mediante el menú “Project” y la opción “Open Project”.

Una vez definidos los archivos del proyecto, para compilarlos en el MPLAB se deberá elegir el menú “Project” y dar click en la opción “Make Project”; con esta opción, si no ha habido errores, se generará un archivo *.hex a partir de nuestro código fuente ubicado en el archivo *.asm. Mientras el compilador revisa las instrucciones escritas en el programa, aparece un cuadro de diálogo como el que se muestra en la figura 3.1.3.1.

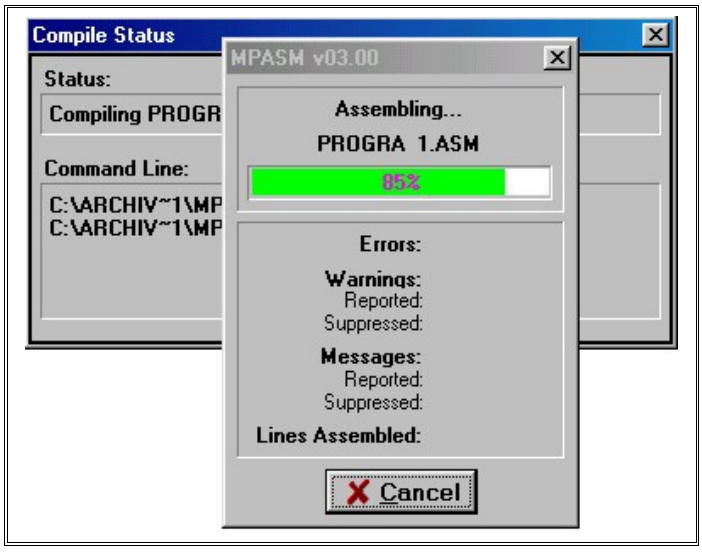


Figura 3.1.3.1. Compilación de un programa.

Cuando el MPLAB intenta generar el archivo *.hex puede avisar de tres tipos de eventos: los errores (errors), los avisos (warnings) y mensajes (messages), los cuales pueden imposibilitar que se cree dicho archivo. En la ayuda del MPLAB se describen uno a uno estos eventos.

El archivo *.hex generado es almacenado en el mismo directorio donde fue guardado el archivo *.asm.

Cuando la compilación resultó exitosa y no ocurrió ningún error, se muestra una ventana que indica la construcción completa del archivo *.hex, justo como se muestra en la figura 3.1.3.2.

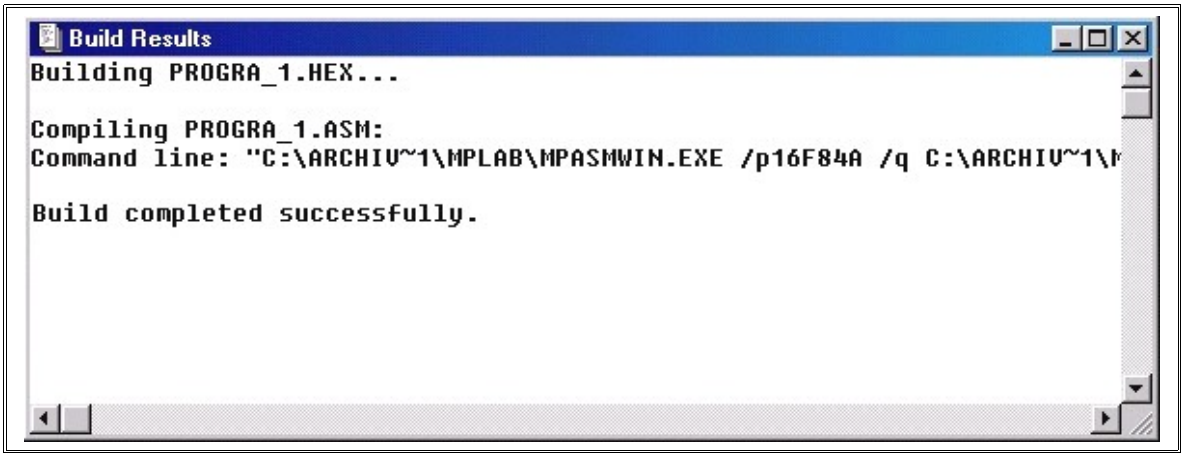


Figura 3.1.3.2. Ventana que indica una correcta compilación

Una vez que se ha creado el archivo *.hex, es posible observar el código generado que se grabará en el microcontrolador mediante el menú “Window”, y la opción “Program Memory”, tal como se muestra en la figura 3.1.3.3.

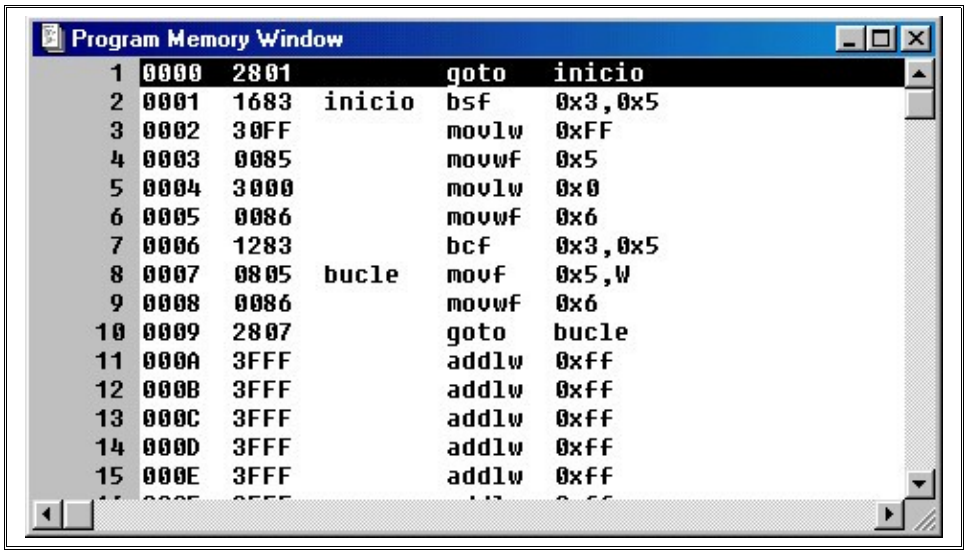


Figura 3.1.3.3. Ventana que muestra el código correspondiente a la memoria de programa

3.1.4 Simulación de programas

Como primer paso para la simulación, se debe de asegurar que el modo de desarrollo del MPLAB está en simulador y que el dispositivo que se especifica es el correcto. Dicha acción se puede realizar mediante el menú ”Options” y la opción “Development Mode”.

En la pestaña “Clock” del cuadro de diálogo “Development Mode” es posible especificar la frecuencia a la cual trabajará físicamente el microcontrolador, dicha frecuencia estará en función al tipo de oscilador, mismo que también deberá ser indicado en esta sección, justo como se muestra en la figura 3.1.4.1.

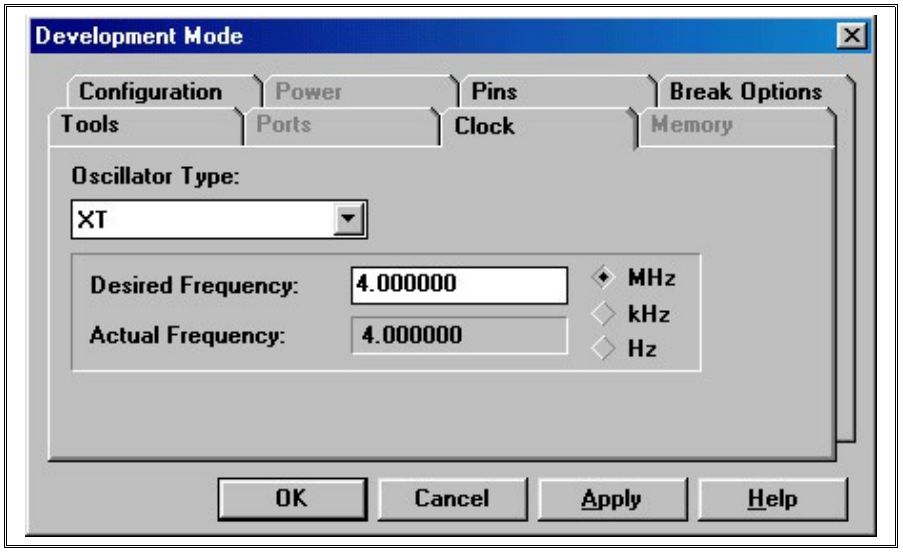


Figura 3.1.4.1. Configuración del oscilador y la frecuencia de operación.

Dentro de este mismo cuadro de diálogo “Development Mode”, es posible habilitar o deshabilitar el Watch Dog Timer ó Perro Guardián, desde la pestaña “Configuration”, tal como se muestra en la figura 3.1.4.2.

Un simulador permite ejecutar en una computadora el programa que se pretende grabar en el microcontrolador.

Para simular un programa en el MPLAB, es necesario abrir el proyecto que contiene el archivo con el programa a comprobar, esto se hace mediante el menú “Project” y la opción “Open Project”.

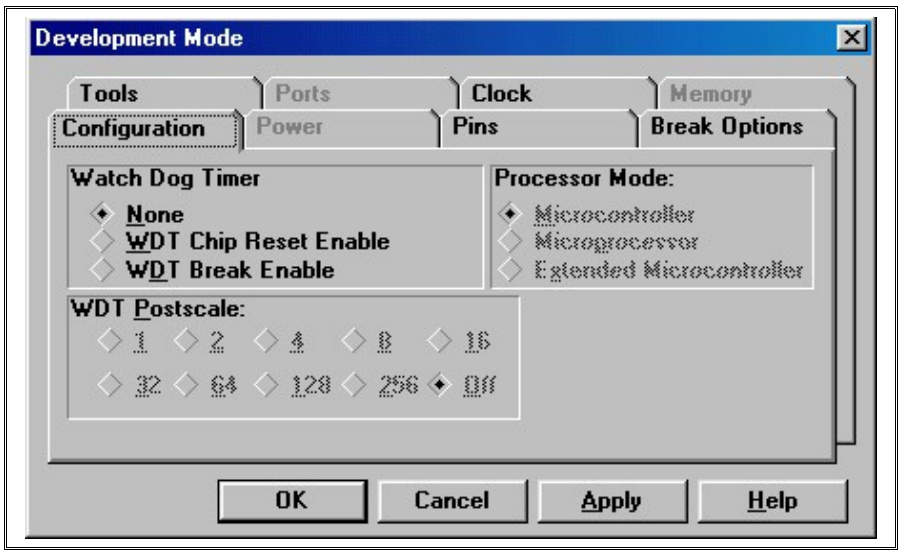


Figura 3.1.4.2. Configuración del Perro Guardián.

Una condición inviolable para simular el programa es que el archivo *.asm debe de estar compilado, de otra forma, aparecerá un mensaje indicando que no hay información de depuración disponible en el proyecto.

Una vez que se tiene abierto el proyecto, se debe abrir el archivo *.asm que se desea simular. Cuando el programa ya ha sido compilado no es necesario hacerlo de nuevo, ya que el MPLAB informa de la existencia el archivo *.hex.

Durante la simulación del programa puede ser interesante ver en qué posiciones se alojan las instrucciones que están ejecutándose. Esto es posible a través de la ventana de "Memoria de programa", la cual puede ser mostrada mediante el menú "Window" y la opción "Program Memory". La información que se muestra en esta ventana puede visualizarse en diferentes formatos, los cuales pueden ser elegidos al dar un click en el icono de "Program Memory" ubicado en esquina superior izquierda de la barra de título de esta misma ventana. La ventana "Program Memory" se muestra en la figura 3.1.4.3.

Estos formatos hacen que se muestre el código de instrucciones en hexadecimal o tal como se ha escrito, facilitando así su comprobación. Además aparece información adicional sobre el número de cada instrucción y su posición dentro de la memoria de programa.

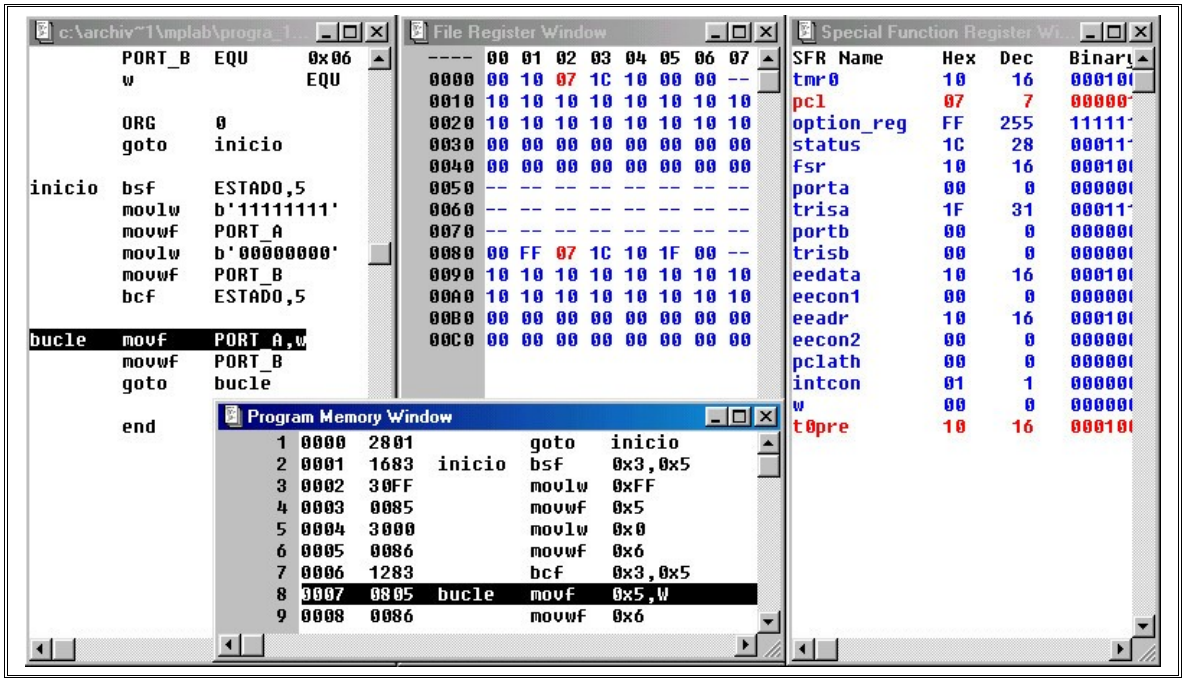


Figura 3.1.4.3. Ventanas útiles al realizar simulaciones.

La ventana general de registros (File Register Window) y la de los registros especiales (Special Function Register Window) también suelen ser muy útiles a la hora de hacer una simulación. Los registros especiales son los específicos y los generales reúnen tanto a los específicos como a los generales. Dichas ventanas se muestran en la parte media superior y superior izquierda de la figura 3.1.4.3.

Si el programa que se ha diseñado trabaja con la memoria EEPROM de datos, es posible visualizar los valores que contiene en la ventana “EEPROM Window”. Dicha ventana se muestra al acceder al menú “Window” y al elegir la opción “EEPROM Memory”. La apariencia de la ventana “EEPROM Window” se muestra en la figura 3.1.4.4.

Otra alternativa es decidir si el programa que se va a simular debe mostrarse tal y como se ha editado o con información adicional que proporciona el simulador al ensamblarlo, como las direcciones de cada instrucción, etiquetas definidas, número de posiciones de la memoria de instrucciones sin utilizar, etc. Esta información está disponible en la ventana de listado, misma que puede visualizarse mediante el menú Window y la opción “Absolute Listing”. Dicha ventana se muestra en la figura 3.1.4.5.

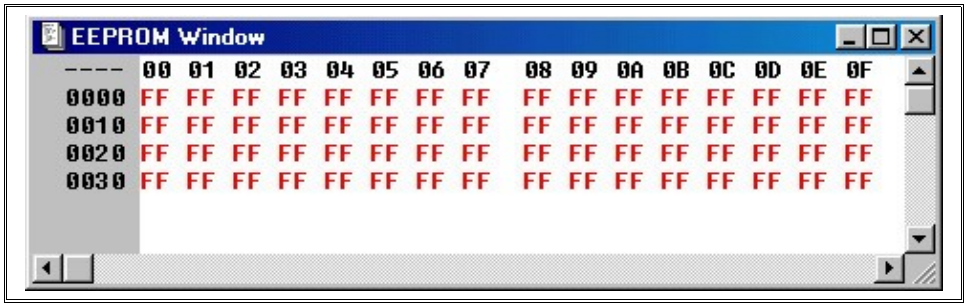


Figura 3.1.4.4. Ventana de la memoria de datos EEPROM.

Una vez que se tienen disponibles varias ventanas es necesario ordenarlas para su visualización óptima. Una opción para ordenar un número considerable de ventanas es acceder al menú “Window” y seleccionar la opción “Title Horizontal”, que como su nombre lo indica colocará las ventanas en un orden horizontal.

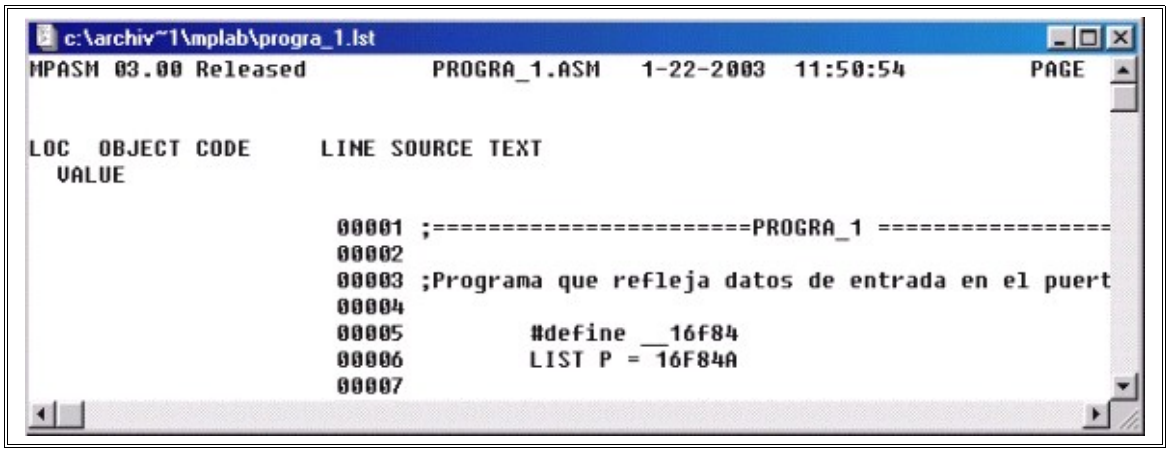


Figura 3.1.4.5. Ventana de listado

Para simular un programa es necesario elegir una opción de esta función, dicha modalidad deberá ser elegida de acuerdo a lo que desea observar en el programa. Puede ser que solo se quiera observar el resultado final de un registro, o quizás ver el desarrollo de un segmento del programa del cual se desconoce si realmente funciona, o bien visualizar paso a paso todo el programa o parte de él.

Antes de ejecutar la simulación debe hacerse un reset general del sistema mediante el menú “Debug”, luego el submenú “Run” y finalmente la opción “Reset”. Cada vez que se realice una simulación, y se quiera volver a empezar debe de hacerse un reset.

Para comenzar la simulación de debe dar click en el menú “Debug”, entrar al submenú “Run” y seleccionar la opción “Run”. En esta opción de simulación solo se pueden observar visualmente el valor de los registros y las posiciones de memoria cuando se detenga la simulación, ya que el sistema pretende ejecutar el programa a la velocidad especificada del oscilador perteneciente al microcontrolador.

Para detener la simulación se debe entrar al menú “Debug”, acceder al submenú “Run” y seleccionar la opción “Halt”.

El siguiente tipo de simulación es de mucha utilidad, ya que mientras se ejecutan las instrucciones, una franja negra de selección ilumina la sentencia próxima a procesar. Dicha opción de simulación puede ejecutarse accediendo al menú “Debug”, luego al submenú “Run” y seleccionando la opción “Animate”. La diferencia de esta modalidad de simulación es que en el modo “Run” no se ve cómo se van modificando los registros, aunque internamente el proceso es el mismo, y una vez ejecutado el código, el contador de programa sigue con las siguientes posiciones de la memoria de programa.

Otra forma de simular el programa en el MPLAB, es la modalidad de paso a paso. La simulación paso a paso se puede ejecutar entrando al menú “Debug”, luego al submenú “Run” y seleccionando la opción “Step”. La principal ventaja de este modo de ejecución es que se ilumina la instrucción a ejecutar, pero no se realiza dicha instrucción hasta que se oprime la tecla rápida de esta función, que es “F7”. De esta manera, cada vez que se oprima “F7” se ejecutará una única instrucción, y es posible observar el valor que cada registro toma tras cada instrucción.

Además, los cambios realizados se van resaltando en color rojo.

Si en un momento determinado se necesita solo ver qué sucede paso a paso en una parte del programa y no en el programa completo, se puede modificar el valor del contador del programa (PC) para que comience donde se requiere. Esto se realiza accediendo al menú “Debug”, luego al submenú “Run” y finalmente mediante la opción “Change Program Counter”. El uso de esta opción suele ser típico en programas muy largos donde la simulación completa en el modo paso a paso sería demasiado tardada o complicada.

En todos los modos de simulación es posible modificar el valor de una posición de memoria o de un registro mediante el menú “Window” y la opción “Modify”. Tanto en la modalidad de “Run” como en la de “Animate”, es necesario detener la simulación para hacer una modificación a un registro. En la opción de simulación “Step” no es necesario detener la ejecución del programa, ya que por defecto está temporalmente detenida hasta que no se oprima la tecla “F7”.

Una vez que aparece el cuadro de diálogo “Modify”, primero se selecciona el área de memoria donde se encuentra la posición o el registro que se desea cambiar en el margen etiquetado como “Memory Area”, posteriormente se indica el sistema de numeración en el que se especificará el nuevo valor en el margen llamado “Radix”, a continuación se escribe la cantidad que se desea asignar en el cuadro de Texto “Data/Opcode” y por último la dirección o registro en el que se desea escribir en la lista desplegable “Address”. Para grabar dicha información solo debe darse un click en el botón etiquetado como “Write”. El cuadro de diálogo “Modify” se muestra en la figura 3.1.4.6. Si se desea escribir en un rango de memoria, solo se especifica la dirección final en la lista desplegable “End Address”y se da un click en “Fill Range”.

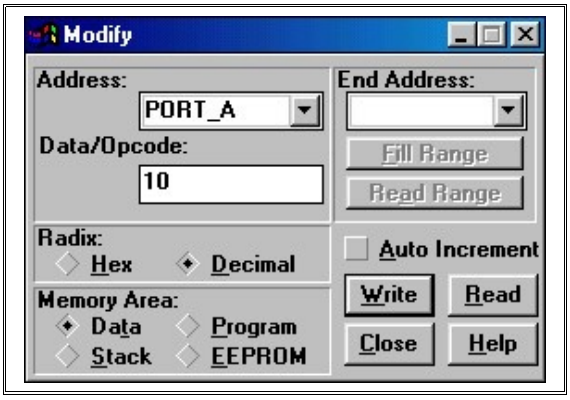


Figura 3.1.4.6. Modificación del valor de un registro.

CAPÍTULO 4. PROGRAMADORES Y PROGRAMACIÓN DEL PIC

4.1 Uso del programador NOPPP

El NOPPP fue creado por Michael A. Covington de la empresa Covington Innovations y está disponible de forma gratuita el la dirección electrónica http://www.mindspring.com/~covington/noppp . El NOPPP emplea el puerto paralelo de una computadora para enviar los datos al microcontrolador.

4.1.1 Circuito eléctrico del grabador

El programador de pics de pocos componentes (No-Parts Pic Programer), como su nombre lo dice, es un programador simple para los microcontroladores PIC16C84, PIC16F83 y PIC16F84. El NOPPP al igual que todos los programadores está compuesto por un software que controla un circuito electrónico (hardware) para programar el PIC. El programa contenido en el archivo *.asm y compilado en formato *.hex por el MPLAB, es enviado por medio del software NOPPP al circuito programador, que a su vez se encarga de cargar los datos en el dispositivo microcontrolador.

Este programador es muy sencillo y emplea componentes que no son difíciles de conseguir. El NOPPP está integrado por partes que regularmente se tienen a la mano. El circuito de este programador de PIC's se muestra en la figura 4.1.1.1 La fuente de alimentación propuesta se muestra en la figura 4.1.1.2.

Lista de materiales para el programador NOPPP:

- R1, R7, R8 = 4.7kΩ
- R2, R3, R4 = 1kΩ
- R5 = 2.2kΩ
- C1, C2 = 0.1µF
- D1, D2 = 1N34
- 1 Zócalo para C.I. de 18 pines

- 1 Conector DB-25

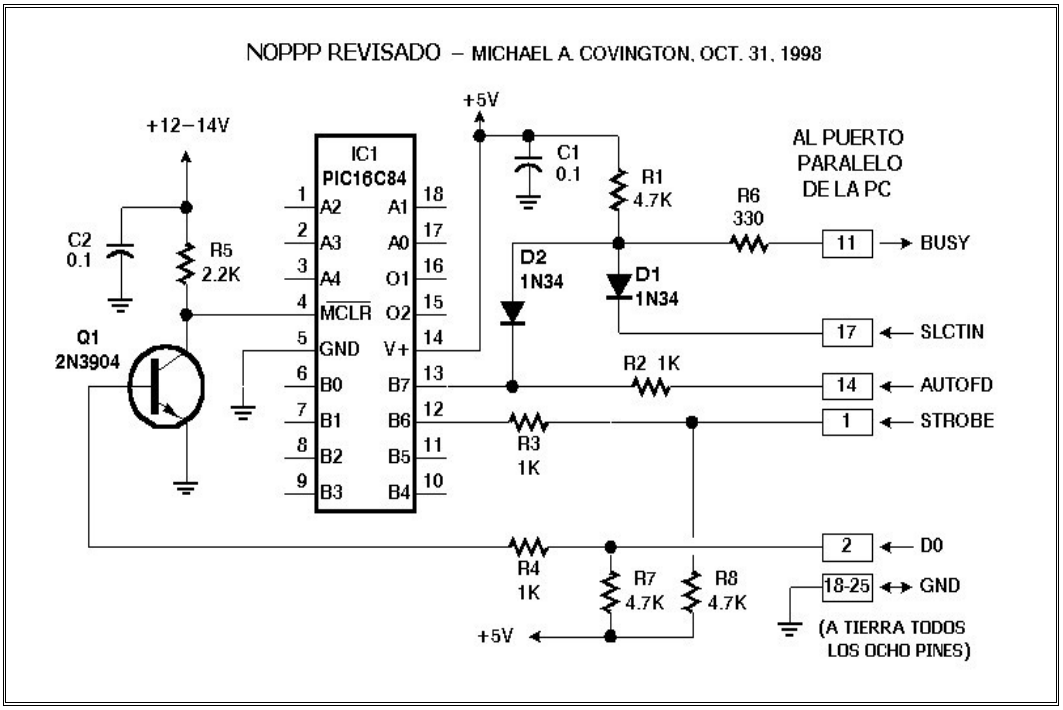


Figura 4.1.1.1. Diagrama eléctrico del programador NOPPP.

Lista de materiales para la fuente de alimentación:

- 1 Transformador (18-25V)
- 1 puente de diodos
- C 1µF
- D 1N914
- Reguladores LM7812 y LM7805

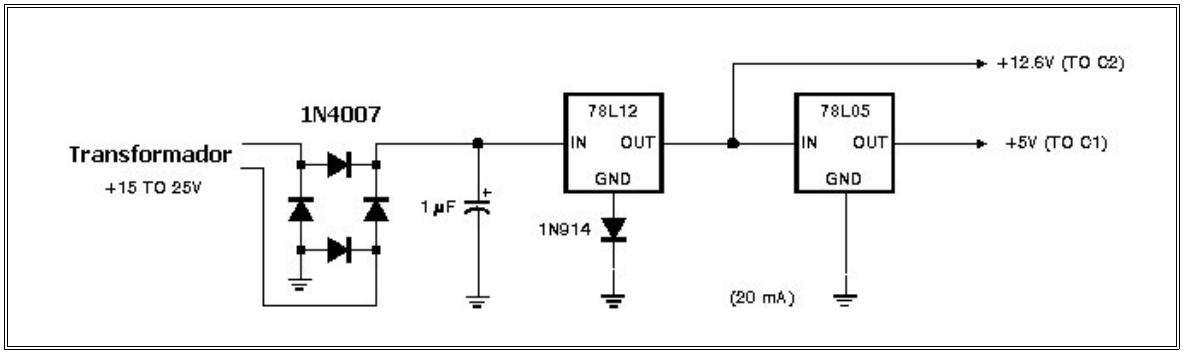


Figura 4.1.1.2. Diagrama de la fuente de alimentación sugerida para el NOPPP.

4.1.2 Grabación de un programa en el microcontrolador

El "No-Parts Pic Programer" ha sido probado exitosamente en 15 diferentes puertos paralelos, así como en computadoras que poseen un procesador 8088 a 4.77MHz hasta un Pentium II a 300MHz, sin embargo no es recomendable ejecutarlo en sistemas operativos como Windows XP, ya que el manejo de puertos es diferente a todas sus versiones anteriores.

El programa NOPPP corre en un entorno de MS-DOS.

Una vez que se tiene el archivo *.hex, producto de la compilación del archivo *.asm, el primer paso para grabar el programa en el microcontrolador consiste en ejecutar el archivo Noppp.exe y especificar el puerto paralelo que se va a utilizar.

Después de especificar el puerto, que normalmente corresponde a LPT1, el siguiente paso consiste en encender la alimentación del circuito eléctrico correspondiente al NOPPP.

A continuación de debe elegir el dispositivo a emplear por medio de una letra que corresponde a cada uno de ellos que soporta este programador.

El siguiente paso es colocar el microcontrolador en el zócalo del circuito eléctrico del NOPPP.

Como penúltimo paso, aparecerá un menú que indica todas las posibilidades del software NOPPP y su respectivo programador.

Para cargar el pic con el programa diseñado se elige la opción de "Load HEX file" y como paso final se indica la ruta donde se encuentra el archivo *.hex, como por ejemplo: C:\program\prueba.hex

Una vez que se ha cargado el programa, el microcontrolador PIC se encuentra listo para realizar las funciones a las que deberá responder de acuerdo al programa diseñado.

4.1.3 Grabación de la palabra de configuración

Cabe mencionar que el NOPPP no posee una utilidad para fijar la palabra de configuración del PIC, por lo tanto es necesario que ésta se especifique en el mismo programa contenido en el archivo *asm.

Antes del cuerpo del programa y después de haber indicado el tipo de dispositivo y las librerías a usar, se escribe el código (__config H'XXX'). Dicha instrucción empieza con un doble guión bajo y las tres equis representan el código en hexadecimal que resulta del número binario formado por la palabra de configuración del PIC. En este caso, la palabra de configuración del PIC16F84 es la que se muestra en la figura 4.1.3.1.

Los bits cero (menos significativo) y uno corresponden a la selección del tipo de oscilador (RC, HS, XT, LP). El bit número 2 está destinado a la habilitación del temporizador del perro guardián (WDT). El bit número tres corresponde a la habilitación del temporizador de energización (PWRTE). Para el PIC16F84, el bit número cuatro determina la configuración del código de protección. El número en binario que resulta de estos bits se convierte a hexadecimal y es el que se escribe en la instrucción "Config".

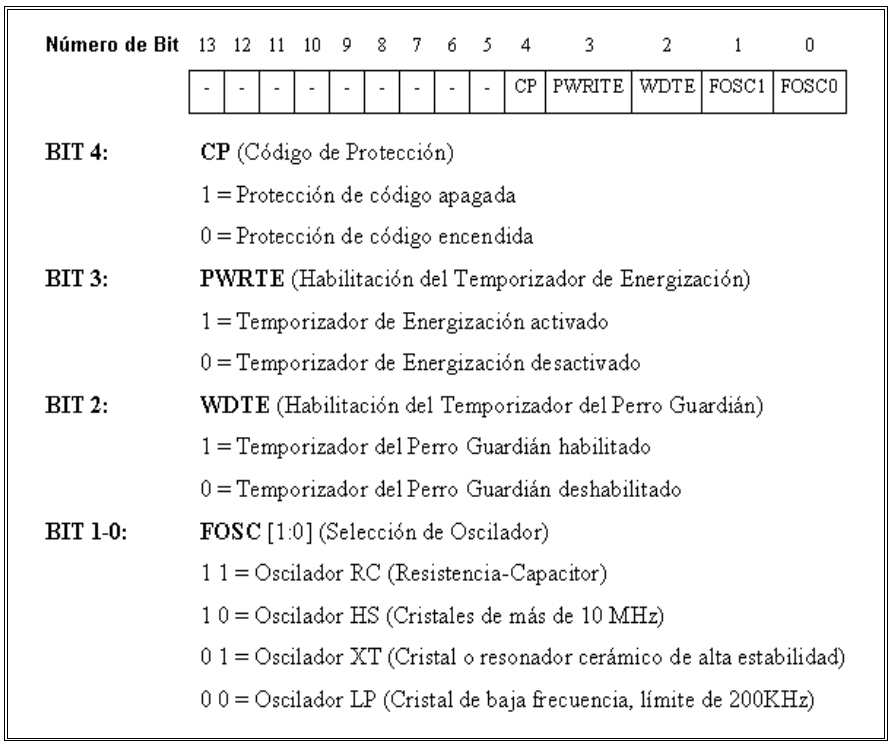


Figura 4.1.3.1. Palabra de configuración del PIC16F84 compuesta por cinco bits

De esta manera, si se desea encender la protección de código, habilitar el temporizador de energización, deshabilitar el Perro Guardián y utilizar un oscilador del tipo XT, el código resultante será 01001, y que equivale en hexadecimal a H“09”, con lo que la instrucción “config” quedaría de la siguiente forma.

```
#define     _16F84
list        p=16F84
include     <p16f84.inc>
__config    H”09”
```

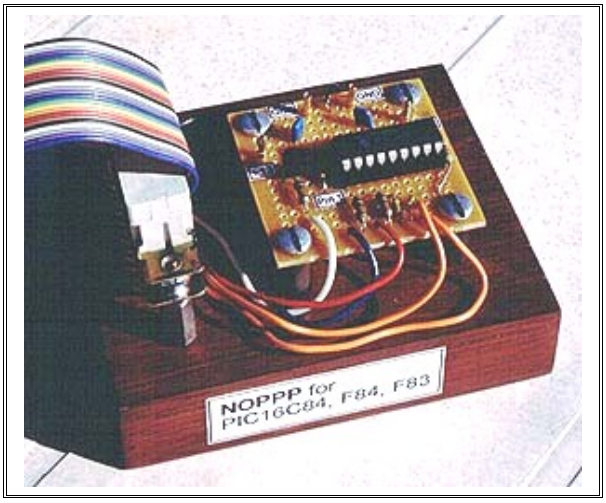


Figura 4.1.3.2. Aspecto físico del programador NOPPP.

El programador de pocos componentes NOPPP es una herramienta de desarrollo muy simple y sencilla para el microcontrolador PIC16F84 de microchip. Su principal característica consiste en la cantidad reducida de componentes que requiere el circuito eléctrico para ensamblar el programador, sin embargo, esta cantidad reducida de elementos determina las limitadas opciones que ofrece el software que controla al programador de PIC’s; no obstante, es una muy buena opción para empezar a programar sin necesidad de grandes inversiones. El aspecto físico de un programador NOPPP se muestra en la figura 4.1.3.2.

4.2 Uso del programador PonyProg2000

Este programa, a diferencia del NOPPP, posee varias utilidades como la lectura de los datos, los programa guardados en el PIC o bien la determinación de la palabra de configuración, sin necesidad de incluirla en el programa en forma de código.

El PonyProg es un software libre y fue creado por Claudio Lanconelli. Dicho programa puede obtenerse gratuitamente en la dirección electrónica www.lancos.com

El PonyProg emplea el puerto serie de una computadora para enviar los datos al microcontrolador.

4.2.1 Instalación

La instalación del PonyProg depende del sistema donde se desea usar el programa. Para los sistemas operativos Windows 95 / 98 / Me ó NT / 2000, solo debe ejecutarse el archivo de instalación que se muestra en la figura 4.2.1.1.

Figura 4.2.1.1. Icono correspondiente al archivo de instalación del PonyProg2000

Una vez iniciada la utilidad de instalación, aparecerá una ventana mostrando la bienvenida a dicho proceso. Para continuar, se deberá dar un click en el botón etiquetado como “Next”, justo como se muestra en la figura 4.2.1.2.

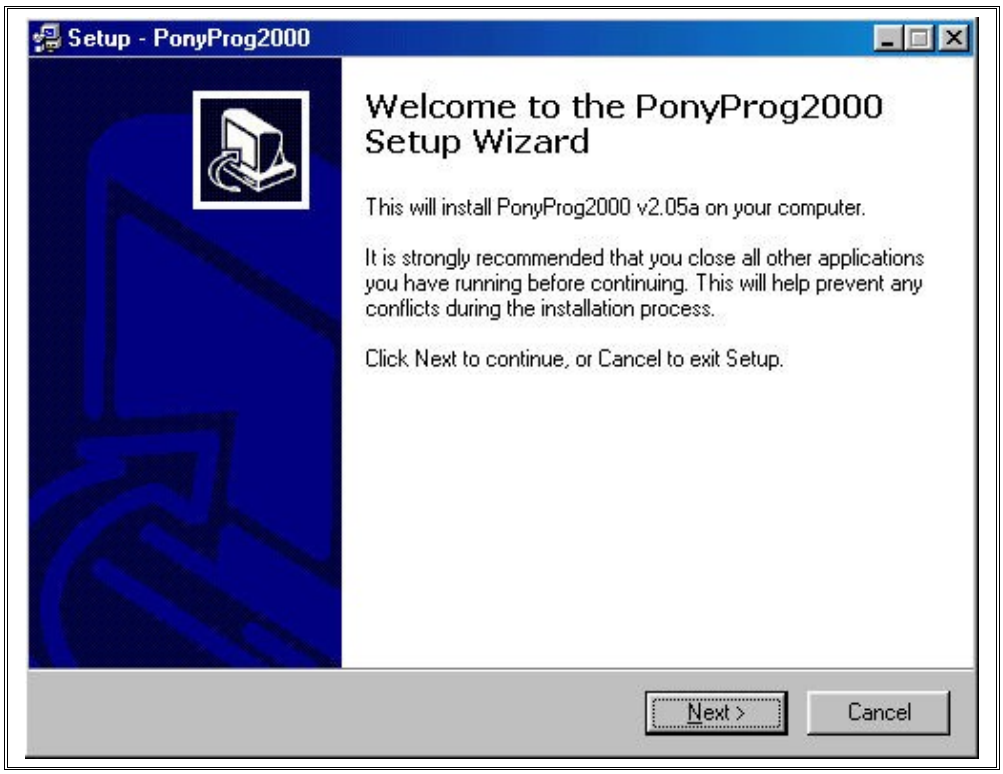


Figura 4.2.1.2. Bienvenida al proceso de instalación.

Después del paso anterior, la siguiente ventana requiere que se especifique la ubicación física en el disco duro, donde se desean guardar los archivos correspondientes a la aplicación PonyProg 2000.

Esta ventana también muestra el espacio libre en MB que el programa requiere en el disco duro. Una vez especificada la ruta de instalación, se deberá dar click al botón "Next", tal como se puede apreciar en la figura 4.2.1.3.

Una vez realizado el paso anterior, solo es necesario seguir las instrucciones restantes de la aplicación en curso.

Cuando la instalación ha finalizado, es recomendable reiniciar la computadora para completar el proceso de registro en la configuración del sistema operativo.

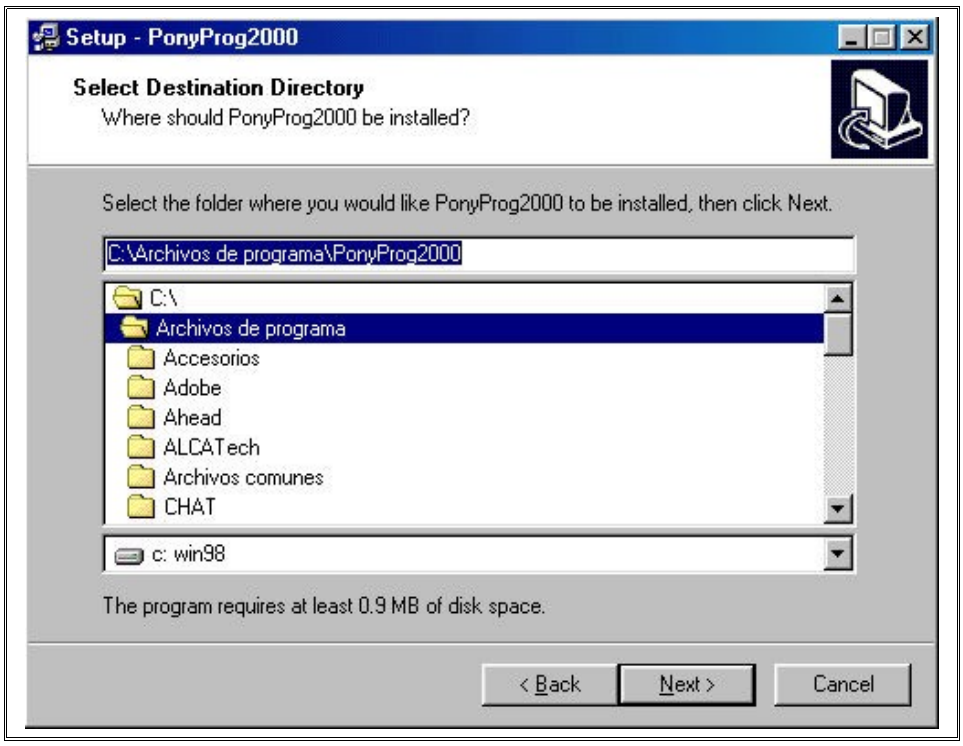


Figura 4.2.1.3. Selección de la ruta de instalación.

Cuando la instalación ha finalizado, el PonyProg2000 es agregado al menú de programas con el mismo nombre.

La forma más fácil de abrir el PonyProg2000 consiste en dar doble click al icono que representa su acceso directo y que se encuentra situado en el escritorio de windows. Dicho icono se muestra en la figura 4.2.1.4.

Figura 4.2.1.4. Acceso directo al PonyProg2000

Cuando la aplicación se ha iniciado, la ventana principal del PonyProg2000 luce como se muestra en la figura 4.2.1.5. Dicha ventana contiene todos los componentes que poseen los programas diseñados para windows.

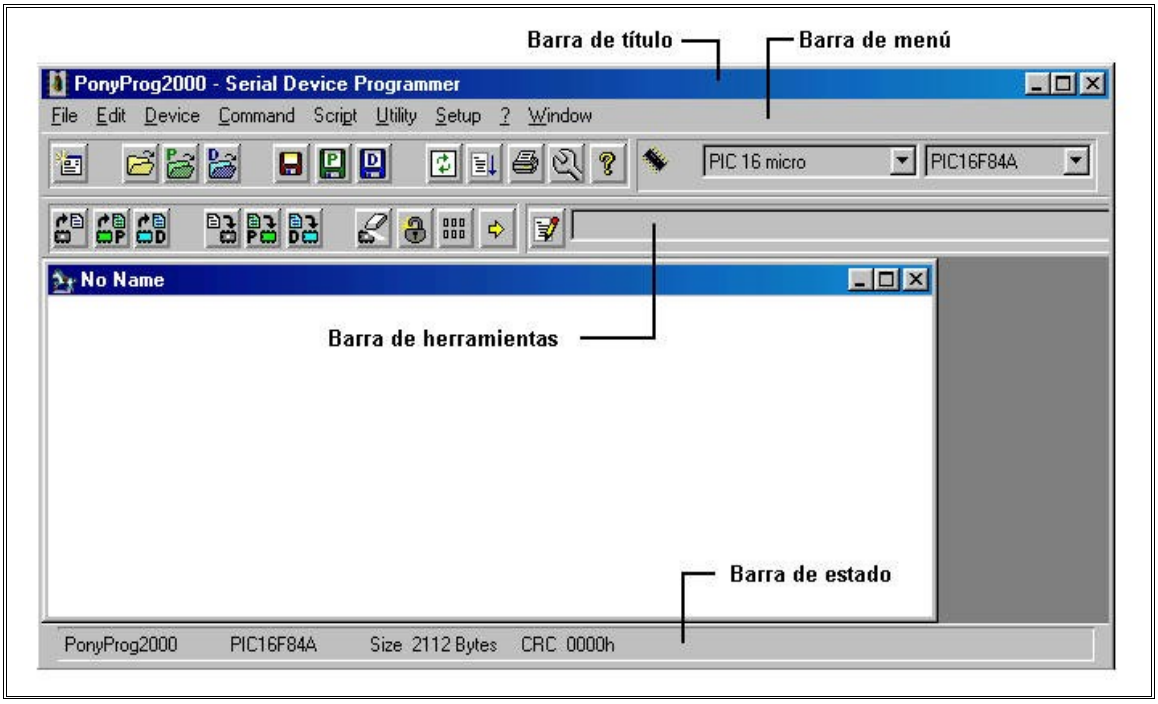


Figura 4.2.1.5. Ventana principal del PonyProg2000

La primera actividad dentro de este programador por el puerto serie, es precisamente especificar qué puerto se va a utilizar; el más común es el COM1, sin embargo, esto depende de cada sistema. Esto puede realizarse a través del menú “Setup” y de la opción “Interfase Setup”.

A continuación aparecerá una ventana donde se deberá elegir el tipo de interfase “SI Prog I/O” y el puerto que se va a emplear, justo como se muestra en la figura 4.2.1.6. Si no se elige el puerto correcto, el programador no funcionará y nunca se podrá realizar cualquier tipo de operación con el microcontrolador. Una vez definida la configuración del puerto, se deberá dar click en el botón etiquetado como “Ok”.

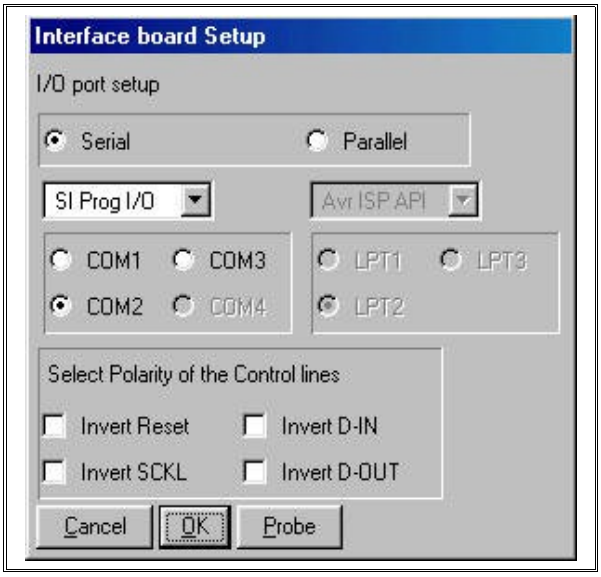


Figura 4.2.1.6. Configuración del puerto.

Siempre que se encienda la computadora y se pretenda trabajar con el PonyProg2000, es necesario realizar una calibración del puerto. Dicho ajuste se hace debido a que el sistema operativo pudiera emplear el puerto al mismo tiempo que el programador serial, esta situación se evita mediante el menú “Setup” y la opción “Calibration” . A continuación aparecerá un cuadro de diálogo que indica la calibración del temporizador del bus, tal como se muestra en la figura 4.2.1.7. Para continuar con el proceso, es necesario dar un click en el botón llamado “Yes”

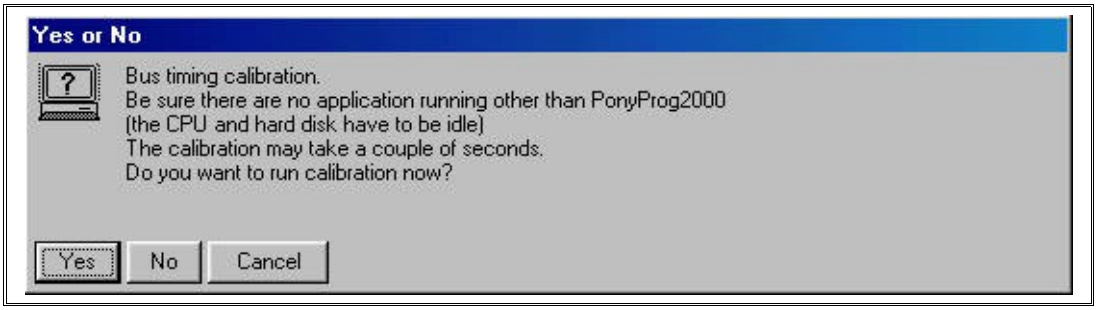


Figura 4.2.1.7. Calibración del temporizador del bus serial.

Cuando la calibración ha terminado, el PonyProg 2000 muestra un mensaje informando al usuario de este acontecimiento, justo como se muestra en la figura 4.2.1.8. Para seguir trabajando solo se deberá dar un click al botón etiquetado como “Ok”.

Figura 4.2.1.8. Finalización de la calibración

Es indispensable que cada vez que se empieza a trabajar con el PonyProg2000, se especifique el tipo de dispositivo que se va a emplear, ya que dicho programa soporta una gran variedad de microcontroladores y memorias.

Esta acción se realiza seleccionando la familia de microcontroladores y a continuación, el tipo de dispositivo en las listas desplegables ubicadas en el lado derecho de la barra de herramientas; en este caso se debe seleccionar la familia “PIC MICRO” y posteriormente el “PIC16F84A”, tal como se muestra en la figura 4.2.1.9.

Figura 4.2.1.9. Selección del tipo de dispositivo.

4.2.2 Circuito eléctrico del grabador

El hardware encargado de recibir los datos del PonyProg es el comisionado para transmitir dicha información al microcontrolador, por lo que cualquier conexión errónea en el circuito, causará un error en cualquier proceso relacionado con el dispositivo.

La lista de materiales del PonyProg es mucho más extensa que la del programador NOPPP. La razón reside en que el primero contiene más utilidades.

Lista de materiales:

- R1, R2, R3, R6 = 4.7KΩ
- R4, R9 = 1KΩ
- R5 = 2.2KΩ
- R7 = 10KΩ
- R8 = 100KΩ
- C1, C2, C3 = 100µF
- C4, C5, C6, C7 = 100nF
- D1,D2 = 1N4007
- D3, D4, D5 = Zener 5.1V ½ W
- Q1, Q2 = BC548
- Q3 = BC558
- LM7812
- LM7805
- Puente de diodos
- Transformador a 24Vca – 1 A
- Conector DB-9 Hembra
- Zócalo para Circuito integrado de 18 pines

El diagrama eléctrico del programador PonyProg2000 para el microcontrolador PIC16F84A se muestra en la figura 4.2.2.1. Nótese que se incluye la fuente de alimentación; sin embargo si ya se posee una fuente de 13 Volts y una de 5 Volts, solo se debe tomar en cuenta el diagrama a partir de la salida de los reguladores.

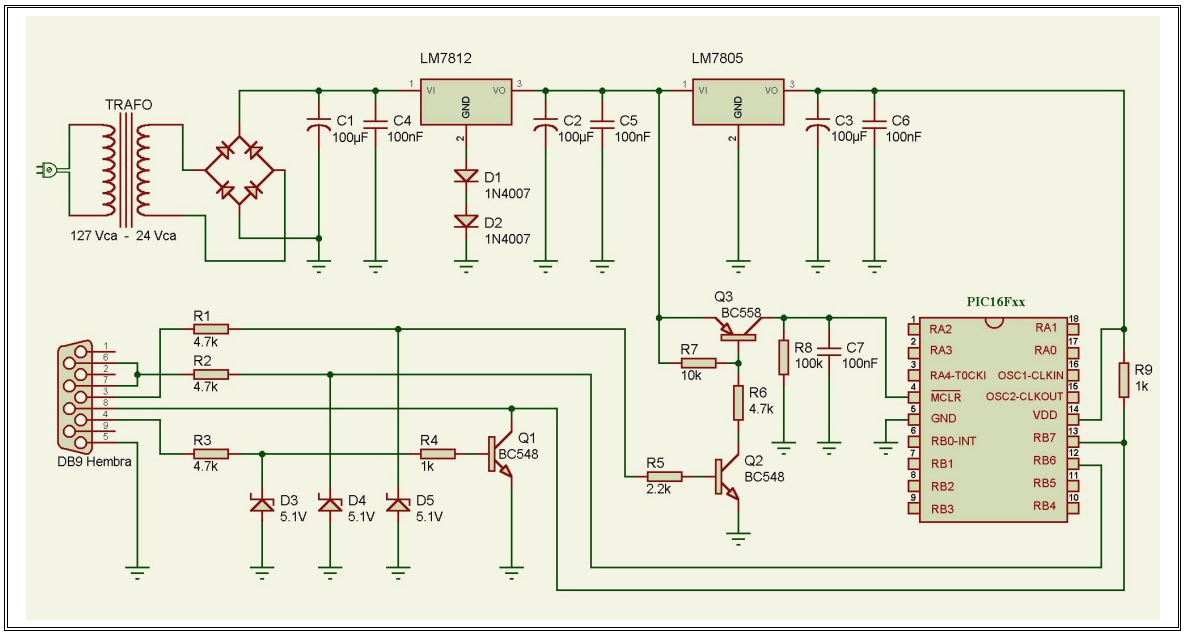


Figura 4.2.2.1. Diagrama eléctrico del PonyProg 2000

Para realizar cualquier procedimiento que involucre la transferencia de datos entre el programador PonyProg 2000 y el hardware (circuito del grabador), es necesario mantener situado al dispositivo microcontrolador en su zócalo y por supuesto, tener alimentación en el grabador, así como una conexión física entre éste y el puerto serial de la computadora. Dicha conexión debe hacerse mediante un cable donde cada terminal se adapte a la forma donde debe ser enchufado.

También es importante poner especial atención en que el programador establece una conexión por medio del puerto serial, lo cual significa que el sistema debe ser capaz de ser controlado mediante un ratón tipo PS-2, ya que es muy complejo operar en PonyProg2000 por medio del teclado.

4.2.3 Procedimiento para leer el microcontrolador

Para leer el microcontrolador se debe acceder al menú “Command” y dar click a la opción “Read All”, con lo que aparecerá una barra de proceso como la que se muestra en la figura 4.2.3.1. Después de esta acción, aparecerá en la pantalla principal el programa que contiene el microcontrolador. Dicho programa se puede almacenar mediante el menú “File” y la opción “Save Device File As ...” para posteriores grabaciones en otros dispositivos similares.

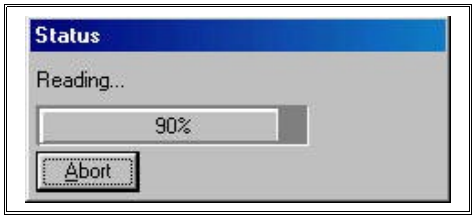


Figura 4.1.3.1. Proceso de lectura del PIC

4.2.4 Procedimiento para leer la palabra de configuración (Fusibles)

La palabra de configuración, mejor conocida como fusibles del PIC, se puede leer accediendo al menú “Command” y la opción “Security and Configuration Bits”, con lo que aparecerá un cuadro de diálogo al cual se deberá dar un click en el botón etiquetado como “Read” para visualizar el estado de los fusibles. Dicha ventana se muestra en la figura 4.2.4.1.

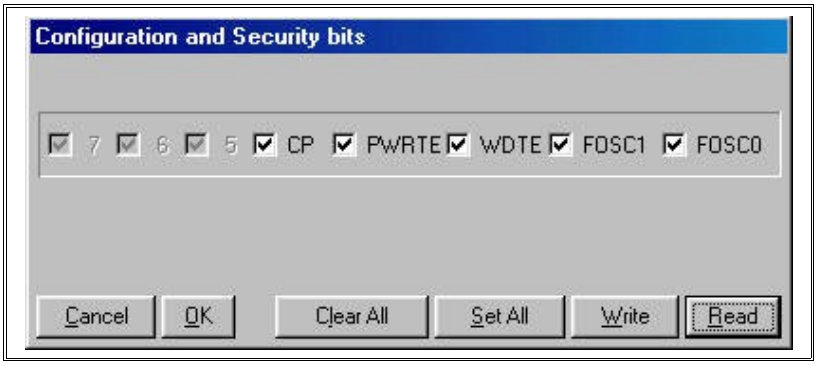


Figura 4.2.4.1. Lectura de la palabra de configuración

La interpretación de las casillas de verificación, del cuadro de diálogo de la palabra de configuración es la siguiente.

- CP: Si está activada la casilla, el código de protección está activado.
- PWRTE: Si está activada la casilla, el temporizador de encendido está activado.
- WDTE: Si está activada la casilla, el WDT está desactivado.
- FOSC1,FOSC0: Bits de selección del oscilador

Casilla desactivada,	Casilla desactivada:	Oscilador RC
Casilla desactivada,	Casilla activada:	Oscilador HS
Casilla activada,	Casilla desactivada:	Oscilador XT
Casilla activada,	Casilla activada:	Oscilador LP

4.2.5 Grabación de un programa en el microcontrolador

Para grabar un programa, es necesario poseer el archivo *.hex, el cual se deberá abrir por medio del menú File, y la opción “Open Device File”. A continuación aparecerá una ventana en la que deberá escogerse la ruta y el archivo a utilizar, tal como se muestra en la figura 4.2.5.1.

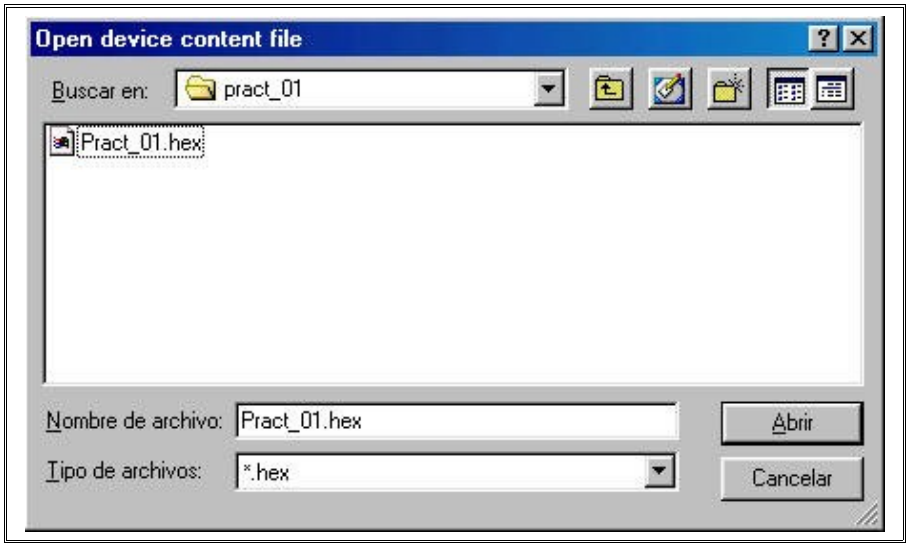


Figura 4.2.5.1. Selección del programa a grabar en el PIC.

Una vez que se ha abierto el archivo *.hex, para grabar en el PIC el programa que contiene este archivo, es necesario acceder al menú “Comand” y posteriormente a la opción”Write All”.

A continuación aparecerá una advertencia preguntado si se está seguro de escribir en el dispositivo, ya que de aceptar, se perderán los datos que éste contenga. Para continuar con el proceso de escritura, es necesario dar un click al botón etiquetado como “Yes”, justo como se muestra en la figura 4.2.5.2.

Figura 4.2.5.2. Advertencia al escribir en el PIC.

Después de indicar que se desea escribir el programa cargado por el PonyProg2000 en el dispositivo, aparecerá una barra de proceso que indica el porcentaje realizado del mismo, tal como se muestra en la figura 4.2.5.3.

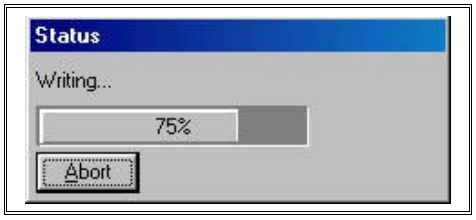


Figura 4.2.5.3. Proceso de escritura.

4.2.6 Grabación de la palabra de configuración en el PIC

Para grabar la palabra de configuración del PIC, es necesario acceder a la ventana que muestra el estado de los fusibles mediante el menú “Command” y ejecuntando la opción “Security and Configuration Bits”. Después de esta acción aparecerán las casillas de verificación de cada bit , las cuales deben ser llenadas de acuerdo a la conveniencia del usuario. Para escribir la palabra de configuración en el dispositivo solo es necesario dar click en el botón “Write” después de haber hecho las modificaciones pertinentes; justo como se muestra en la figura 4.2.6.1.

Figura 4.2.6.1. Escritura de los fusibles del microcontrolador

4.2.7 Procedimiento para borrar el PIC

Para eliminar por completo todos los datos del microcontrolador, solo es necesario acceder al menú “command” y elegir la opción “Erase”.

Se debe tener especial cuidado con esta opción, ya que no se muestra ningún mensaje posterior a la ejecución de esta función y se borra inmediatamente el dispositivo.

Cuando el borrado se está realizando, se muestra una barra de proceso que indica el porcentaje completado de esta acción, tal como se muestra en la figura 4.2.7.1.

Una vez que se ha completado la eliminación de datos, aparece un mensaje que indica la culminación de dicho proceso, tal como se muestra en la figura 4.2.7.1.

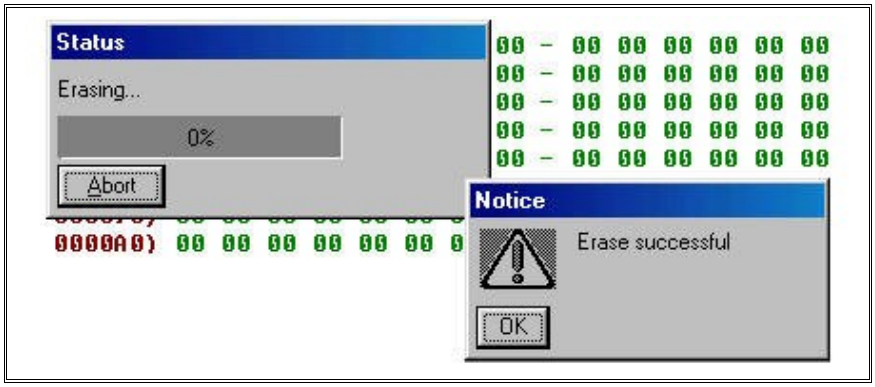


Figura 4.2.7.1. Mensajes que indican la eliminación de la información contenida en el PIC.

CAPÍTULO 5. PRÁCTICAS SUGERIDAS

5.1 Práctica No. 1 Manejo de puertos (Como entrada y salida de datos)

5.1.1 Objetivo

Que el lector aprenda a utilizar los puertos del PIC, así como los mnemónicos utilizados para configurarlos.

5.1.2 Descripción

Se configura el puerto A como entrada de datos y el puerto B como salida. Una vez que se introduce un dato en el puerto A, dicho dato será reflejado en el puerto B mediante el encendido de los Leds.

5.1.3 Diagrama de flujo

El diagrama de flujo de la práctica número 1 se muestra en la figura 5.1.3.1.

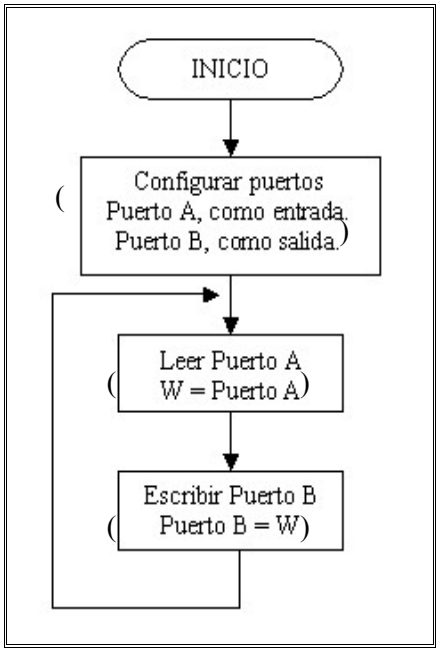


Figura 5.1.3.1. Diagrama de flujo de la práctica 1.

5.1.4 Material y equipo a utilizar

- Computadora personal (características mínimas necesarias para ejecutar las aplicaciones MPLAB y PonyProg2000)
- Software MPLAB
- Software PonyProg
- Circuito Grabador de PIC's PonyProg
- Fuente de alimentación de 5V.
- 1 PIC16F84A
- 1 Cristal Oscilador a 4 MHz
- 2 Capacitores de 22 pF
- 5 Resistencias de 10 KΩ
- 5 Resistencias de 560Ω
- 1 Resistencia de 100Ω
- 4 Led's
- 5 Interruptores miniatura

5.1.5 Procedimiento

Como primer paso a seguir se debe copiar el siguiente programa a un editor de texto sin formato como el "Block de Notas" de Windows, o el editor del MPLAB.

```
;============================ Práctica_01 ==========================


; Se colocan cuatro interruptores en las líneas RA0, RA1, RA2, RA3,
; del puerto A del PIC16F84A y cuatro diodos Led en las líneas
; RB0, RB1, RB2, RB3, del puerto B.

; Mediante los interruptores se introduce un número binario de cuatro
; bits, el cual deberá ser reflejado en los Led's del puerto B.

;============================== INICIO ================================

         #define __16f84
         LIST P = 16F84A           ;Comando que indica el pic usado

;============================  Etiquetas ==============================

         ESTADO   EQU   0x03       ;Dirección del reg. Estado en hex
         PORT_A   EQU   0x05       ;Dirección del Puerto A en hex
         PORT_B   EQU   0x06       ;Dirección del Puerto B en hex
         w        EQU   0x00       ;w = 0

;============================== Programa ==============================

         ORG      0                ;Dirección del vector de reset
         goto     inicio           ;Salto a inicio

inicio   bsf      ESTADO,5         ;Se pone en 1 el bit 5 del reg. ESTADO
         movlw    b'11111111'      ;w = b'11111111'
         movwf    PORT_A           ;Se configura el puerto A como entrada
         movlw    b'00000000'      ;w = b'00000000'
         movwf    PORT_B           ;Se configura el puerto B como entrada
         bcf      ESTADO,5         ;Se pone en 0 el bit 5 del reg. ESTADO

bucle    movf     PORT_A,w         ;Mueve el valor del puerto A a w
         movwf    PORT_B           ;Mueve w al puerto B
         goto     bucle            ;Salto a bucle

;================================ Fin ================================

         end

;Nota: para este programa no importa la frecuencia de oscilación
```

Una vez que se ha editado todo el programa, es necesario guardarlo en un archivo que puede ser llamado Practica_01.asm

El siguiente paso consiste en compilar el programa que se ha creado mediante el MPLAB para generar el archivo Practica_01.hex

Mediante el PonyProg2000 debe grabarse el contenido del archivo Practica_01.hex en el microcontrolador. Una vez hecho esto, se debe armar el circuito de aplicación para esta práctica, el cual se muestra en la figura 5.1.5.1.

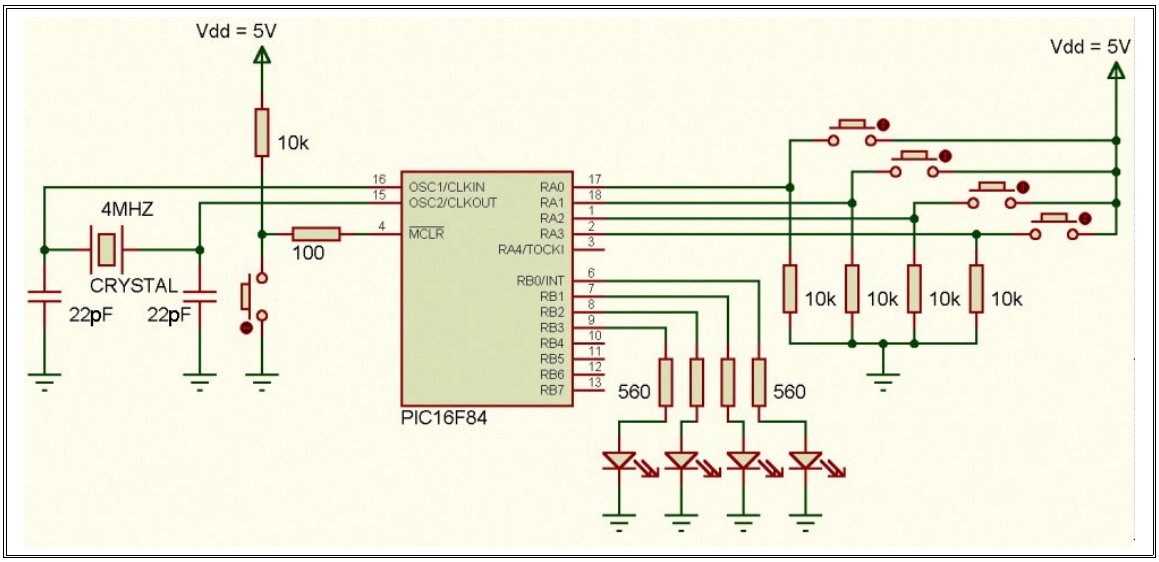


Figura 5.1.5.1. Circuito de aplicación para la práctica 1.

Cuando el circuito de aplicación esté terminado y el PIC ya tenga grabado el programa correspondiente, se inserta el dispositivo en dicho circuito y después se enciende la fuente de alimentación para probar su correcto funcionamiento.

5.1.6 Indicaciones

Para esta práctica, la palabra de configuración del PIC debe ser establecida de la siguiente forma. Código de protección deshabilitado, WDTE deshabilitado, PWRTE habilitado, oscilador XT. Los pines que no sean empleados en esta práctica, deben conectarse a Vdd por medio de una resistencia de 10KΩ.

5.2 Práctica No. 2 Encendido y apagado de un led.

5.2.1 Objetivo

En esta práctica el lector aprenderá a utilizar ciclos anidados para crear un retardo de encendido y apagado de un led.

5.2.2 Descripción

Para la realización de este programa se implementó la utilización de ciclos anidados, los cuales ocasionarán una pérdida de tiempo en el PIC, debido a su alta frecuencia de operación. Esto dará un efecto de parpadeo en la salida del puerto B.

5.2.3 Diagrama de flujo

El diagrama de flujo de la práctica número 2 se muestra en la figura 5.1.3.1.

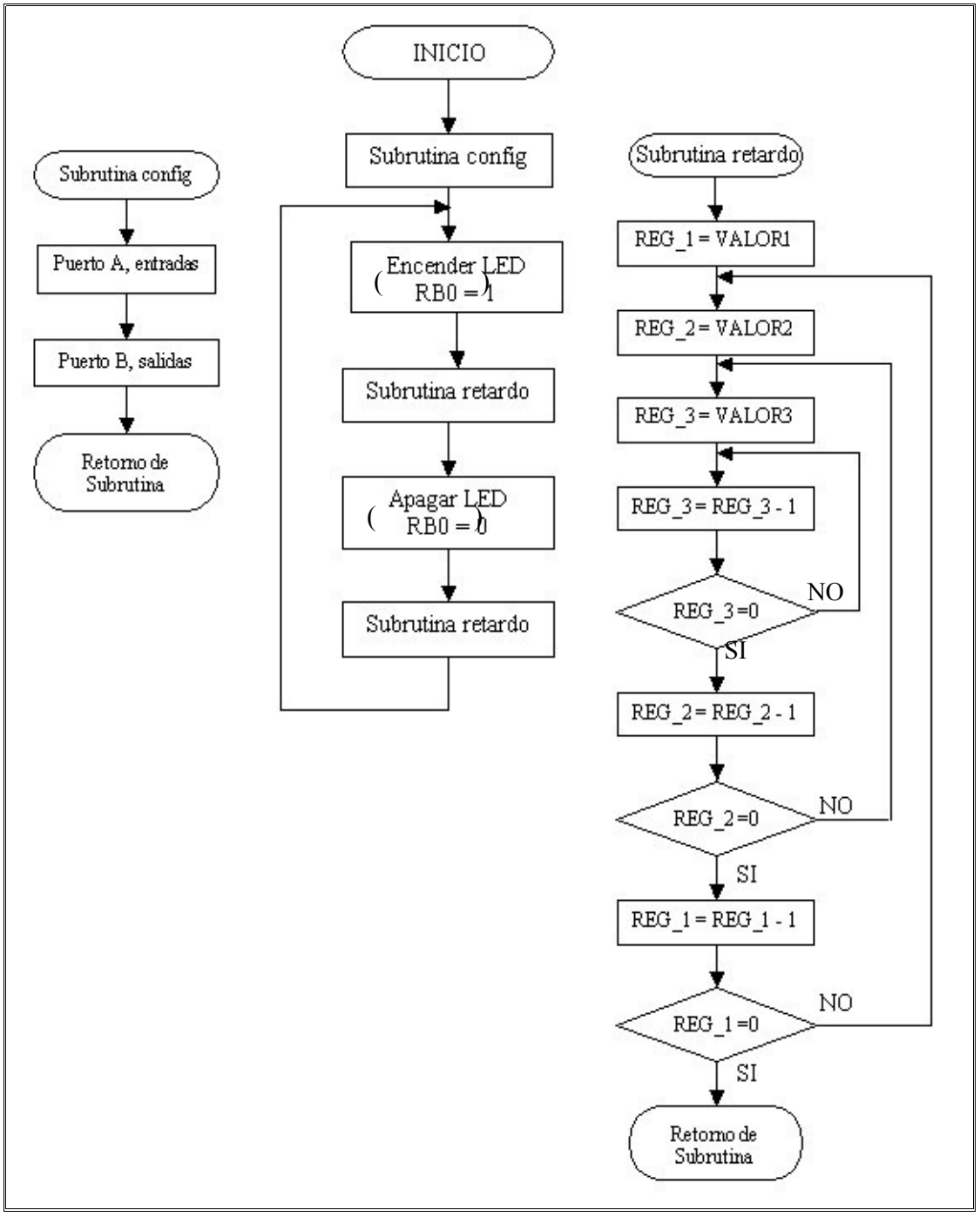


Figura 5.2.3.1. Diagrama de flujo de la práctica 2.

5.2.4 Material y equipo a utilizar

- Computadora personal (características mínimas necesarias para ejecutar las aplicaciones MPLAB y PonyProg2000)
- Software MPLAB
- Software PonyProg ó NOPPP
- Circuito Grabador de PIC's PonyProg ó NOPPP
- Fuente de alimentación de 5V.
- 1 PIC16F84A
- 1 Cristal Oscilador a 4 MHz
- 2 Capacitores de 22 pF
- 1 Resistencias de 10 KΩ
- 1 Resistencias de 560Ω
- 1 Resistencia de 100Ω
- 1 Led
- 1 Interruptor miniatura

5.2.5 Procedimiento

Como primer paso a seguir se debe copiar el siguiente programa a un editor de texto sin formato como el "Block de Notas" de Windows, o el editor del MPLAB.

```
;============================ Práctica_02 ============================

;Programa que prende y apaga un led a una determinada frecuencia
;El led debe estar conectado en el pin RB0 del puerto B del PIC

;============================ INICIO ==============================

        #define __16f84
        LIST P = 16F84A       ;Comando que indica el pic usado

;============================ Etiquetas ============================

        ESTADO  EQU       0x03      ;Dirección del reg. Estado en hex
        PC      EQU       0X02      ;Dirección del reg. PC en hex
        PORT_A  EQU       0x05      ;Dirección del puerto A en hex
        PORT_B  EQU       0x06      ;Dirección del puerto B en hex
        w EQU   0x00                ;w = 0
        REG_1   EQU       0x0C      ;Dirección del registro REG_1
        REG_2   EQU       0x0D      ;Dirección del registro REG_2
        REG_3   EQU       0x0E      ;Dirección del registro REG_3
        VALOR1  EQU       0x30      ;Valor que se asigna a VALOR1
        VALOR2  EQU       0x40      ;Valor que se asigna a VALOR2
        VALOR3  EQU       0x50      ;Valor que se asigna a VALOR3

;============================ Programa ============================

        ORG     0             ;Vector de reset
        goto    inicio        ;Salto a inicio
        ORG     5

inicio  call    config        ;Llamada a la rutina config
bucle   movlw   0x00          ;Mueve 0x01 a w
        movwf   PORT_B        ;Mueve w al puerto B (apaga Led)
        call    retardo       ;Llamada a la rutina retardo
        movlw   0x01          ;Mueve 0x02 a w
        movwf   PORT_B        ;Mueve w al puerto B (prende Led)
        call    retardo       ;Llamada a la rutina retardo
        goto    bucle         ;Salto a bucle

;============================ Rutinas ============================

config  bsf     ESTADO,5      ;Se pone en 1 el bit 5 del reg. estado
        movlw   b'11111111'   ;w = b'11111111'   w = 0XFF
        movwf   PORT_A        ;Se configura el puerto A como entradas
        movlw   b'00000000'   ;w = b'00000000'   w = 0X00
        movwf   PORT_B        ;Se configura el puerto B como salidas
        bcf     ESTADO,5      ;Se pone en 0 el bit 5 del reg. estado
        return
```

```
retardo movlw   VALOR1          ;Carga w con el número 30 (VALOR1)
        movwf   REG_1           ;Mueve w al registro REG_1
tres    movlw   VALOR2          ;Carga w con el número 40 (VALOR2)
        movwf   REG_2           ;Mueve w al registro REG_2
dos     movlw   VALOR3          ;Carga w con el número 50 (VALOR3)
        movwf   REG_3           ;Mueve w al registro REG_3
uno     decfsz  REG_3           ;Decrementa el valor de REG_3 en 1
        goto    uno             ;Salto a uno
        decfsz  REG_2           ;Decrementa el valor de REG_2 en 1
        goto    dos             ;Salto a dos
        decfsz  REG_1           ;Decrementa el valor de REG_1 en 1
        goto    tres            ;Salto a tres
        return                  ;Retorno desde subrutina

;================================ Fin ================================

        end                     ;Fin del programa

;Nota: La frecuencia externa debe ser de 4MHz (Cristal de 4MHz)
```

Una vez que se ha editado todo el programa, es necesario guardarlo en un archivo que puede ser llamado Practica_02.asm

El siguiente paso consiste en compilar el programa que se ha creado mediante el MPLAB para generar el archivo Practica_02.hex

Mediante el PonyProg2000 debe grabarse el contenido del archivo Practica_02.hex en el microcontrolador. Una vez hecho esto, se debe armar el circuito de aplicación para esta práctica, el cual se muestra en la figura 5.2.5.1.

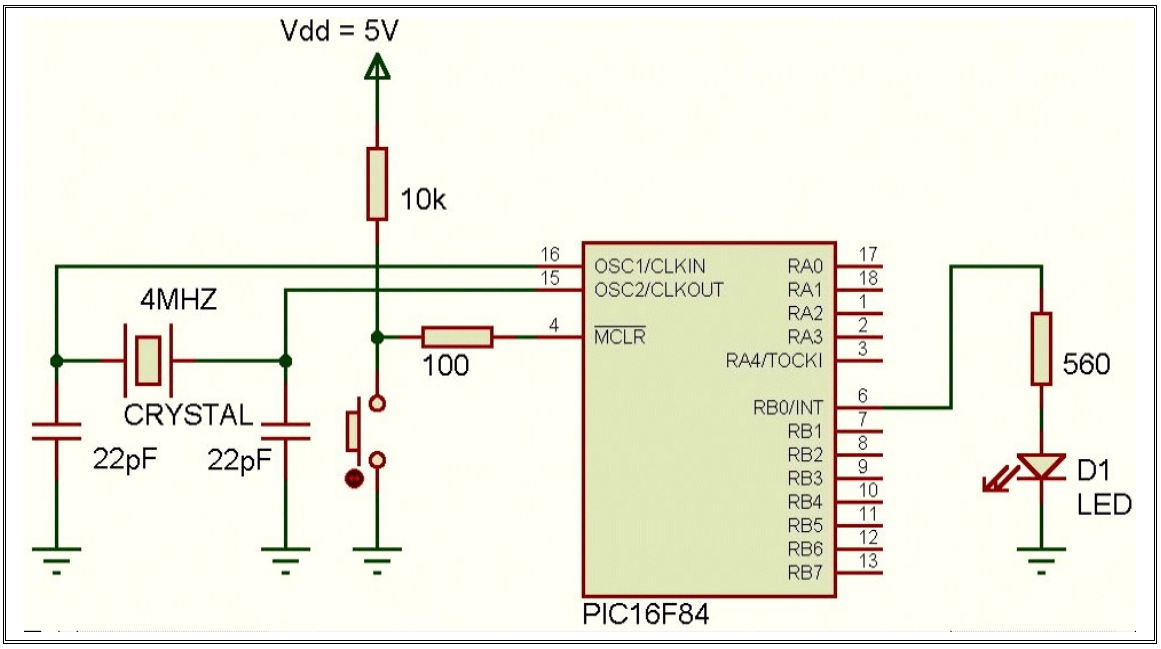


Figura 5.2.5.1. Circuito de aplicación para la práctica 2.

Cuando el circuito de aplicación esté terminado y el PIC ya tenga grabado el programa correspondiente, se inserta el dispositivo en dicho circuito y después se enciende la fuente de alimentación para probar su correcto funcionamiento.

5.2.6 Indicaciones

Para esta práctica, la palabra de configuración del PIC debe ser establecida de la siguiente forma. Código de protección deshabilitado, WDTE deshabilitado, PWRTE habilitado, oscilador XT. Los pines que no sean empleados en esta práctica, deben conectarse a Vdd por medio de una resistencia de 10KΩ.

5.3 Práctica No. 3 Secuencia de leds

5.3.1 Objetivo

Que el lector pueda realizar una sencilla secuencia de leds con la utilización del mnemotécnico LRF.

5.3.2 Descripción

Este simple circuito muestra una llamativa secuencia de encendido de leds, los cuales dan la sensación de un corrimiento hacia la izquierda, debido a la rotación generada por el mnemónico LRF. La experiencia de la práctica anterior servirá para la utilización de un tiempo de retardo.

5.3.3 Diagrama de flujo

El diagrama de flujo de la práctica número 3 se muestra en la figura 5.3.3.1.

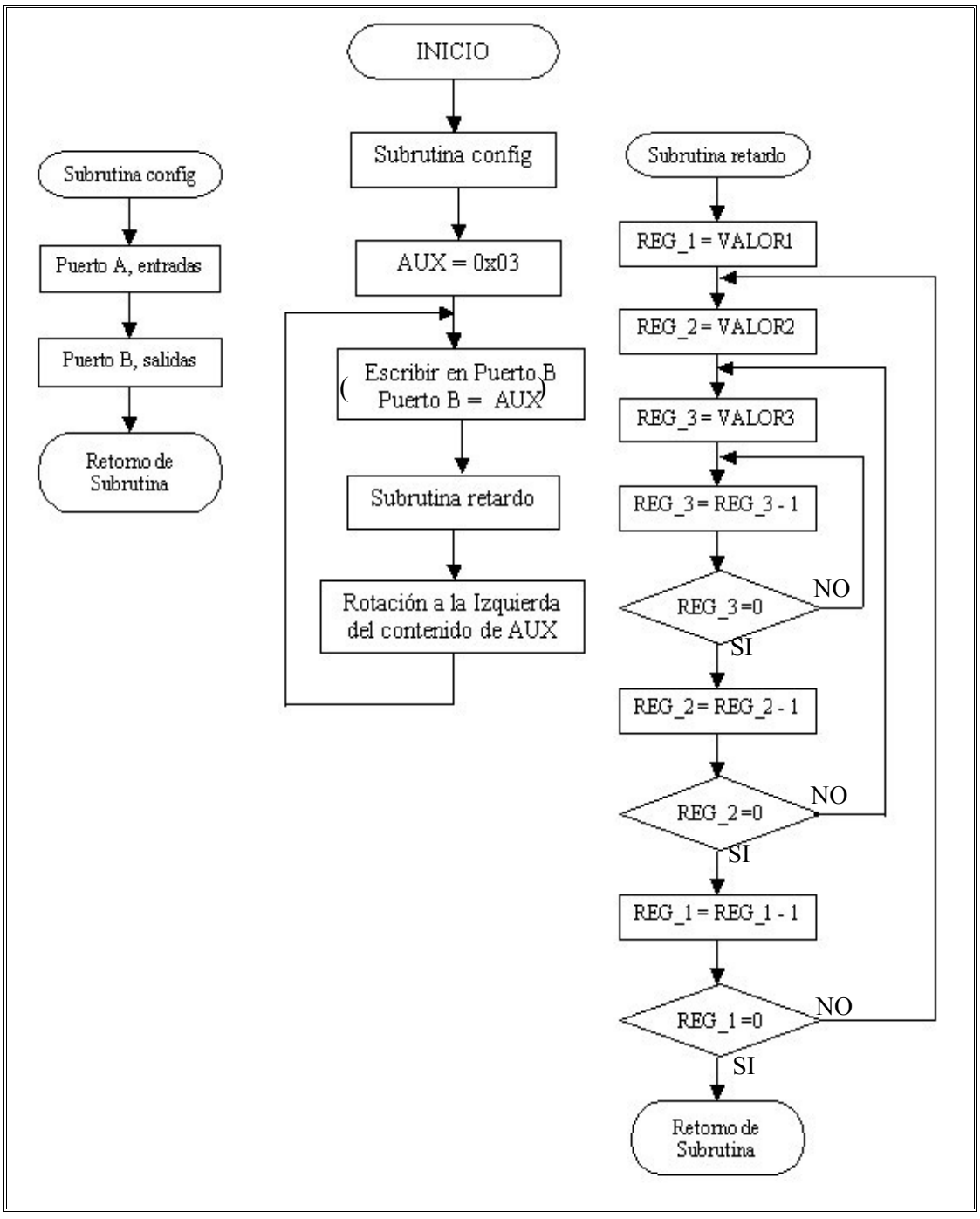


Figura 5.3.3.1. Diagrama de flujo de la práctica 3.

5.3.4 Material y equipo a utilizar

- Computadora personal (características mínimas necesarias para ejecutar las aplicaciones MPLAB y PonyProg2000)
- Software MPLAB
- Software PonyProg
- Circuito Grabador de PIC's PonyProg
- Fuente de alimentación de 5V.
- 1 PIC16F84A
- 1 Cristal Oscilador a 4 MHz
- 2 Capacitores de 22 pF
- 1 Resistencias de 10 KΩ
- 8 Resistencias de 560Ω
- 1 Resistencia de 100Ω
- 8 Led's
- 1 Interruptores miniatura

5.3.5 Procedimiento

Como primer paso a seguir se debe copiar el siguiente programa a un editor de texto sin formato como el "Block de Notas" de Windows, o el editor del MPLAB.

```
;========================== Práctica_03 ==============================

;Programa que realiza una determinada secuencia de Led's.

;Los led's deben conectarse en los pines del puerto B (RB0, RB1, RB2,
;RB3, RB4, RB5, RB6, RB7).

;Se deben conectar los led's con su respectiva resistencia limitadora
;de corriente.

;============================ INICIO ================================

        #define __16f84
        LIST P = 16F84A           ;Comando que indica el pic usado

;============================ Etiquetas ==============================

        ESTADO   EQU   0x03  ;Dirección del reg. ESTADO
        PORT_A   EQU   0x05  ;Dirección del puerto A
        PORT_B   EQU   0x06  ;Dirección del puerto B
        w        EQU   0x00  ;w = 0
        REG_1    EQU   0x0C  ;Dirección del registro REG_1
        REG_2    EQU   0x0D  ;Dirección del registro REG_2
        REG_3    EQU   0x0E  ;Dirección del registro REG_3
        VALOR1   EQU   0x30  ;Valor que se asigna a VALOR1
        VALOR2   EQU   0x40  ;Valor que se asigna a VALOR2
        VALOR3   EQU   0x50  ;Valor que se asigna a VALOR3
        AUX      EQU   0x0F  ;Dirección del registro AUX

;============================ Programa ===============================

        ORG      0           ;Vector de reset
        goto     inicio      ;Salto a inicio
        ORG      7

inicio  call     config      ;Llamada a la rutina config
        movlw    0x03        ;w = 0x03
        movwf    AUX         ;Carga w con el dato 0x03
bucle   movf     AUX,w       ;w = AUX
        movwf    PORT_B      ;Mueve w al puerto B
        call     retardo     ;Llamada a la rutina retardo
        rlf      AUX,1       ;Rotacion a la izquierda, usando carry
        goto     bucle       ;Salto a bucle

;============================ Subrutinas =============================

config  bsf      ESTADO,5    ;Se pone en 1 el bit 5 del reg. ESTADO
        movlw    b'11111111' ;Carga w con el dato b'11111111'
        movwf    PORT_A      ;Configura el puerto A como entradas
        movlw    b'00000000' ;Carga w con el dato b'00000000'
        movwf    PORT_B      ;Configura el puerto B como salidas
        bcf      ESTADO,5    ;Se pone en 0 el bit 5 del reg. ESTADO
        return               ;Retorno desde subrutina
```

```
retardo  movlw    VALOR1      ;Carga w con el número 30 (VALOR1)
         movwf    REG_1       ;Mueve w al registro REG_1
tres     movlw    VALOR2      ;Carga w con el número 40 (VALOR2)
         movwf    REG_2       ;Mueve w al registro REG_2
dos      movlw    VALOR3      ;Carga w con el número 50 (VALOR3)
         movwf    REG_3       ;Mueve w al registro REG_3
uno      decfsz   REG_3       ;Decrementa el valor de REG_3 en 1
         goto     uno         ;Salto a uno
         decfsz   REG_2       ;Decrementa el valor de REG_2 en 1
         goto     dos         ;Salto a dos
         decfsz   REG_1       ;Decrementa el valor de REG_1 en 1
         goto     tres        ;Salto a tres
         return               ;Retorno desde subrutina

;============================== Fin ==================================

         end
```

Nota: La frecuencia externa debe ser de 4MHz (Cristal de 4MHz).

Una vez que se ha editado todo el programa, es necesario guardarlo en un archivo que puede ser llamado Practica_03.asm

El siguiente paso consiste en compilar el programa que se ha creado mediante el MPLAB para generar el archivo Practica_03.hex

Mediante el PonyProg2000 debe grabarse el contenido del archivo Practica_03.hex en el microcontrolador. Una vez hecho esto, se debe armar el circuito de aplicación para esta práctica, el cual se muestra en la figura 5.3.5.1.

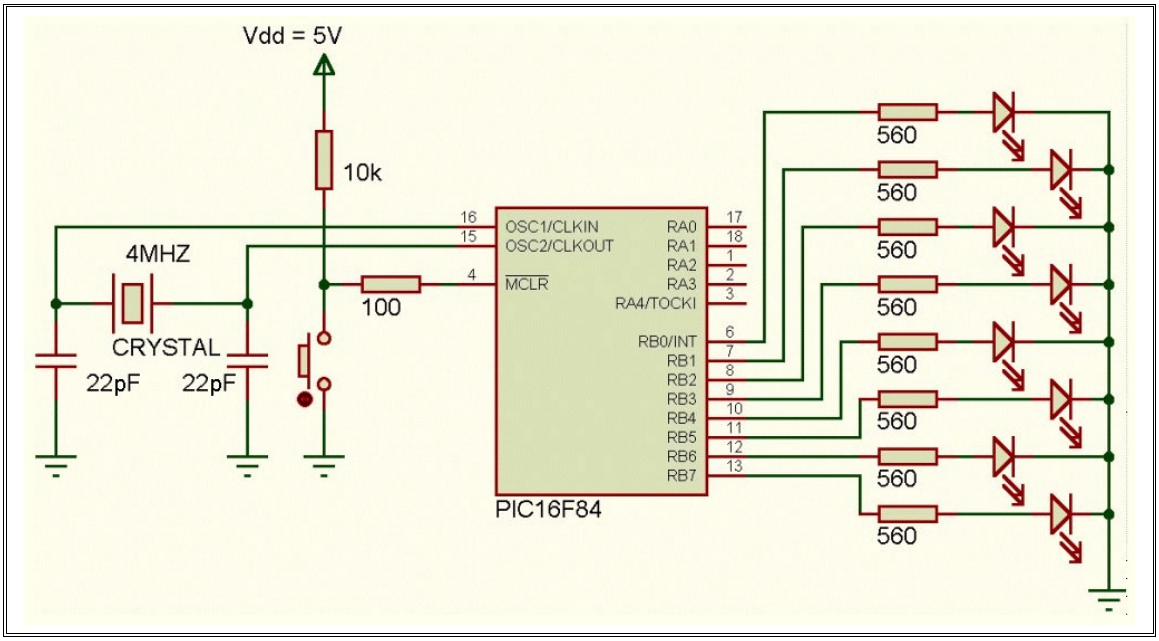


Figura 5.3.5.1. Circuito de aplicación para la práctica 3.

Cuando el circuito de aplicación esté terminado y el PIC ya tenga grabado el programa correspondiente, se inserta el dispositivo en dicho circuito y después se enciende la fuente de alimentación para probar su correcto funcionamiento.

5.3.6 Indicaciones

Para esta práctica, la palabra de configuración del PIC debe ser establecida de la siguiente forma. Código de protección deshabilitado, WDTE deshabilitado, PWRTE habilitado, oscilador XT. Los pines que no sean empleados en esta práctica, deben conectarse a Vdd por medio de una resistencia de 10KΩ.

5.4 Práctica No. 4 Secuencia de 0 a 9

5.4.1 Objetivo

En esta práctica se observará un conteo decimal de 0 a 9.

5.4.2 Descripción

Este circuito mostrará un conteo de 0 a 9. Esto es posible debido a la implementación de secuencias de salida. Dichas secuencias corresponden a cada uno de los números en decimal, por lo que el PIC mostrará un dato de salida que será representado por el display como un número. Después de esto existirá un tiempo de espera y se mostrará el siguiente número.

5.4.3 Diagrama de flujo

El diagrama de flujo de la práctica número 4 se muestra en la figura 5.4.3.1.

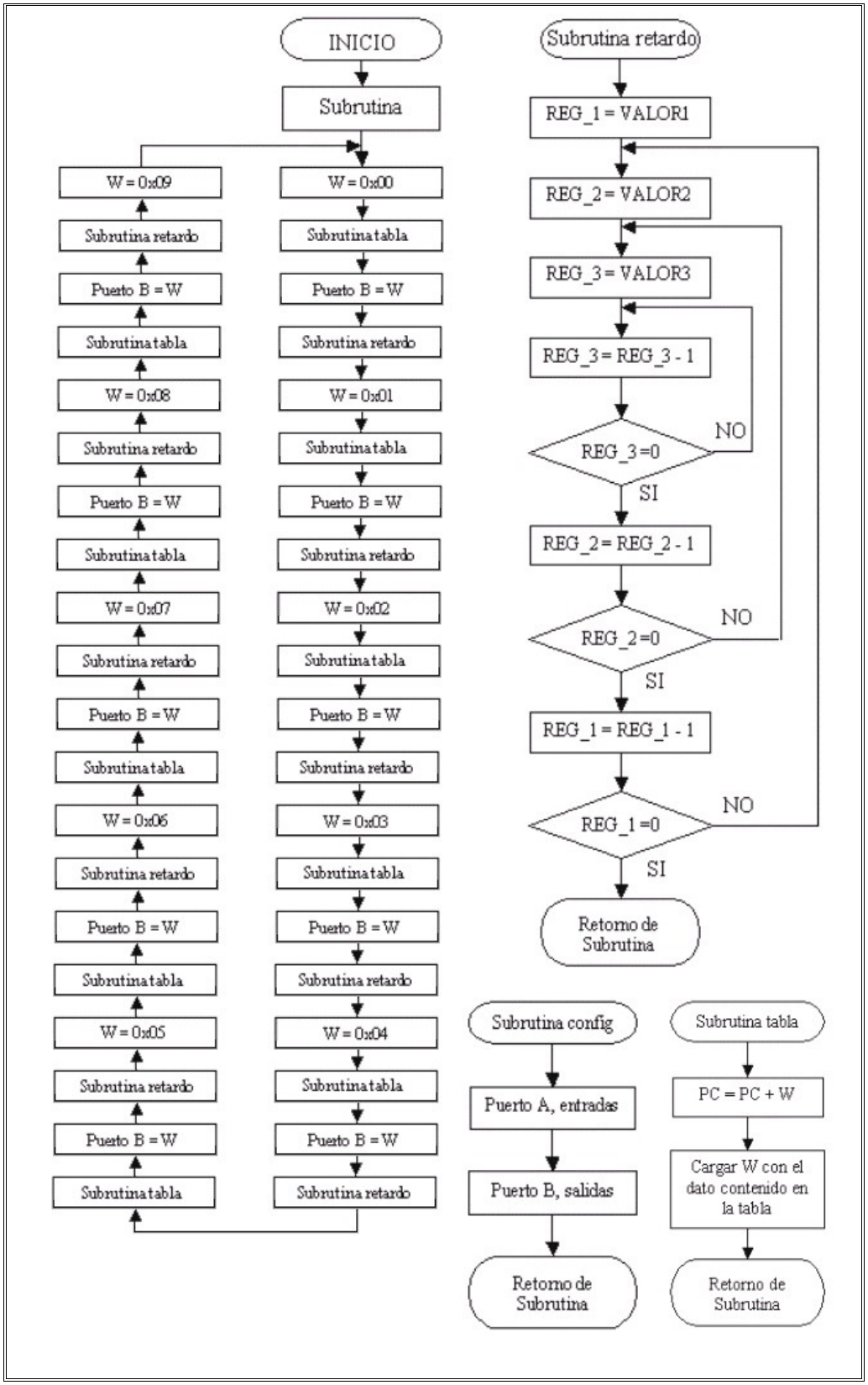


Figura 5.4.3.1. Diagrama de flujo de la práctica 4.

5.4.4 Material y equipo a utilizar

- Computadora personal (características mínimas necesarias para ejecutar las aplicaciones MPLAB y PonyProg2000)
- Software MPLAB
- Software PonyProg
- Circuito Grabador de PIC's PonyProg
- Fuente de alimentación de 5V.
- 1 PIC16F84A
- 1 Cristal Oscilador a 4 MHz
- 2 Capacitores de 22 pF
- 1 Resistencias de 10 KΩ
- 7 Resistencias de 560Ω
- 1 Resistencia de 100Ω
- 1 Display de cátodo común.
- 1 Interruptor miniatura

5.4.5 Procedimiento

Como primer paso a seguir se debe copiar el siguiente programa a un editor de texto sin formato como el "Block de Notas" de Windows, o el editor del MPLAB.

```
;========================== Práctica_04 ==============================

;Programa que muestra en un display de 7 segmentos un conteo del 0 - 9

;El display de 7 segmentos debe conectarse en los pines del puerto B de
;la siguiente manera:

;RB0-a, RB1-b, RB2-c, RB3-d, RB4-e, RB5-f, RB6-g.

;Recuerda que el PIC no puede entregar demasiada corriente, por eso,
;cuando conectes el display de 7 segmentos debes colocar resistencias
;limitadoras de corriente (iguales o mayores a 560 ohms)

;============================ INICIO ================================

        #define __16f84
        LIST P = 16F84A          ;Comando que indica el pic usado

;============================ Etiquetas =============================

        PC      EQU   0X02       ;Dirección del reg. PC en hex
        ESTADO  EQU   0x03       ;Dirección del reg. ESTADO
        PORT_A  EQU   0x05       ;Dirección del puerto A
        PORT_B  EQU   0x06       ;Dirección del puerto B
        w       EQU   0x00       ;w = 0
        REG_1   EQU   0x0C       ;Dirección del registro REG_1
        REG_2   EQU   0x0D       ;Dirección del registro REG_2
        REG_3   EQU   0x0E       ;Dirección del registro REG_3
        VALOR1  EQU   0x30       ;Valor que se asigna a VALOR1
        VALOR2  EQU   0x40       ;Valor que se asigna a VALOR2
        VALOR3  EQU   0x50       ;Valor que se asigna a VALOR3
        AUX     EQU   0x0F       ;Dirección del registro AUX

;============================ Programa ==============================

        ORG     0                ;Vector de reset
        goto    inicio           ;Salto a inicio
        ORG     5

inicio  call    config           ;Llamada a la rutina config
bucle   movlw   0x00             ;Carga w con el dato 0x00
        call    tabla            ;Llamada a la rutina tabla
        movwf   PORT_B           ;Se envian datos al puerto B
        call    retardo          ;Llamada a la rutina retardo
        movlw   0x01             ;Carga w con el dato 0x01
        call    tabla            ;Llamada a la rutina tabla
        movwf   PORT_B           ;Se envian datos al puerto B
        call    retardo          ;Llamada a la rutina retardo
        movlw   0x02             ;Carga w con el dato 0x02
        call    tabla            ;Llamada a la rutina tabla
        movwf   PORT_B           ;Se envian datos al puerto B
        call    retardo          ;Llamada a la rutina retardo
        movlw   0x03             ;Carga w con el dato 0x03
        call    tabla            ;Llamada a la rutina tabla
        movwf   PORT_B           ;Se envian datos al puerto B
```

```
        call    retardo         ;Llamada a la rutina retardo
        movlw   0x04            ;Carga w con el dato 0x04
        call    tabla           ;Llamada a la rutina tabla
        movwf   PORT_B          ;Se envian datos al puerto B
        call    retardo         ;Llamada a la rutina retardo
        movlw   0x05            ;Carga w con el dato 0x05
        call    tabla           ;Llamada a la rutina tabla
        movwf   PORT_B          ;Se envian datos al puerto B
        call    retardo         ;Llamada a la rutina retardo
        movlw   0x06            ;Carga w con el dato 0x06
        call    tabla           ;Llamada a la rutina tabla
        movwf   PORT_B          ;Se envian datos al puerto B
        call    retardo         ;Llamada a la rutina retardo
        movlw   0x07            ;Carga w con el dato 0x07
        call    tabla           ;Llamada a la rutina tabla
        movwf   PORT_B          ;Se envian datos al puerto B
        call    retardo         ;Llamada a la rutina retardo
        movlw   0x08            ;Carga w con el dato 0x08
        call    tabla           ;Llamada a la rutina tabla
        movwf   PORT_B          ;Se envian datos al puerto B
        call    retardo         ;Llamada a la rutina retardo
        movlw   0x09            ;Carga w con el dato 0x09
        call    tabla           ;Llamada a la rutina tabla
        movwf   PORT_B          ;Se envian datos al puerto B
        call    retardo         ;Llamada a la rutina retardo
        goto    bucle           ;Salto a bucle

;============================ Subrutinas ==============================

config  bsf     ESTADO,5        ;Se pone en 1 el bit 5 del reg. ESTADO
        movlw   b'11111111'     ;Carga w con el dato b'11111111'
        movwf   PORT_A          ;Configura el puerto A como entradas
        movlw   b'00000000'     ;Carga w con el dato b'00000000'
        movwf   PORT_B          ;Configura el puerto B como salidas
        bcf     ESTADO,5        ;Se pone en 0 el bit 5 del reg. ESTADO
        return                  ;Retorno desde subrutina

retardo movlw   VALOR1          ;Carga w con el número 30 (VALOR1)
        movwf   REG_1           ;Mueve w al registro REG_1
tres    movlw   VALOR2          ;Carga w con el número 40 (VALOR2)
        movwf   REG_2           ;Mueve w al registro REG_2
dos     movlw   VALOR3          ;Carga w con el número 50 (VALOR3)
        movwf   REG_3           ;Mueve w al registro REG_3
uno     decfsz  REG_3           ;Decrementa el valor de REG_3 en 1
        goto    uno             ;Salto a uno
        decfsz  REG_2           ;Decrementa el valor de REG_2 en 1
        goto    dos             ;Salto a dos
        decfsz  REG_1           ;Decrementa el valor de REG_1 en 1
        goto    tres            ;Salto a tres
        return                  ;Retorno desde subrutina
```

```
tabla    addwf    PC,1              ;pc= PC+w
                  ; .gfedcba        Codigo para 7seg catodo comun
         retlw    b'00111111'       ;7 segmentos para el 0
         retlw    b'00000110'       ;7 segmentos para el 1
         retlw    b'01011011'       ;7 segmentos para el 2
         retlw    b'01001111'       ;7 segmentos para el 3
         retlw    b'01100110'       ;7 segmentos para el 4
         retlw    b'01101101'       ;7 segmentos para el 5
         retlw    b'01111101'       ;7 segmentos para el 6
         retlw    b'00000111'       ;7 segmentos para el 7
         retlw    b'01111111'       ;7 segmentos para el 8
         retlw    b'01101111'       ;7 segmentos para el 9

;================================ Fin ===================================

         end

;Notas:

;La frecuencia externa debe ser de 4MHz (Cristal de 4MHz).

;Si se desea usar un display de 7 segmentos de ánodo comun, entonces
;se deberan complementar los datos de la tabla.
```

Una vez que se ha editado todo el programa, es necesario guardarlo en un archivo que puede ser llamado Practica_04.asm

El siguiente paso consiste en compilar el programa que se ha creado mediante el MPLAB para generar el archivo Practica_04.hex

Mediante el PonyProg2000 debe grabarse el contenido del archivo Practica_04.hex en el microcontrolador. Una vez hecho esto, se debe armar el circuito de aplicación para esta práctica, el cual se muestra en la figura 5.4.5.1.

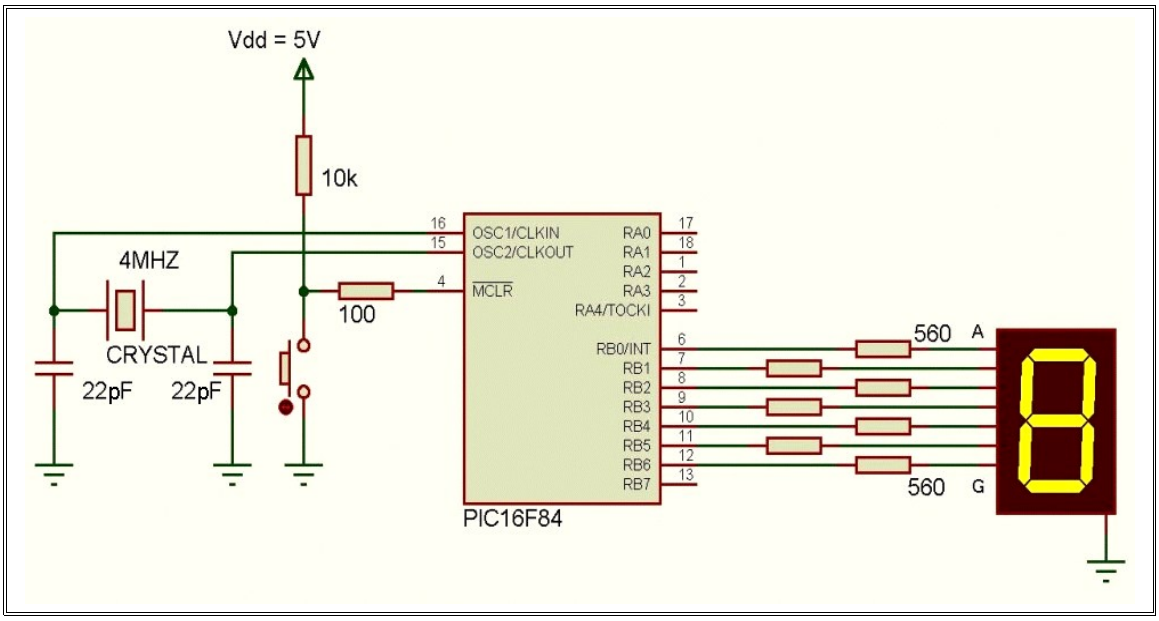


Figura 5.4.5.1. Circuito de aplicación para la práctica 4.

Cuando el circuito de aplicación esté terminado y el PIC ya tenga grabado el programa correspondiente, se inserta el dispositivo en dicho circuito y después se enciende la fuente de alimentación para probar su correcto funcionamiento.

5.4.6 Indicaciones

Para esta práctica, la palabra de configuración del PIC debe ser establecida de la siguiente forma. Código de protección deshabilitado, WDTE deshabilitado, PWRTE habilitado, oscilador XT. Los pines que no sean empleados en esta práctica, deben conectarse a Vdd por medio de una resistencia de 10KΩ.

5.5 Práctica No. 5 Decodificador BCD a 7 segmentos

5.5.1 Objetivo

Que el lector sea capaz de implementar un driver decodificador de BCD a 7 segmentos. Así mismo el lector podrá darse cuenta del manejo de tablas.

5.5.2 Descripción

Las señales de entrada serán proporcionadas por medio de interruptores que se encuentran conectados en el puerto A. La salida será visualizada en un display de 7 segmentos conectado al puerto B. En este programa la conversión de BCD a 7 segmentos se realiza por medio de tablas.

5.5.3 Diagrama de flujo

El diagrama de flujo de la práctica número 5 se muestra en la figura 5.5.3.1.

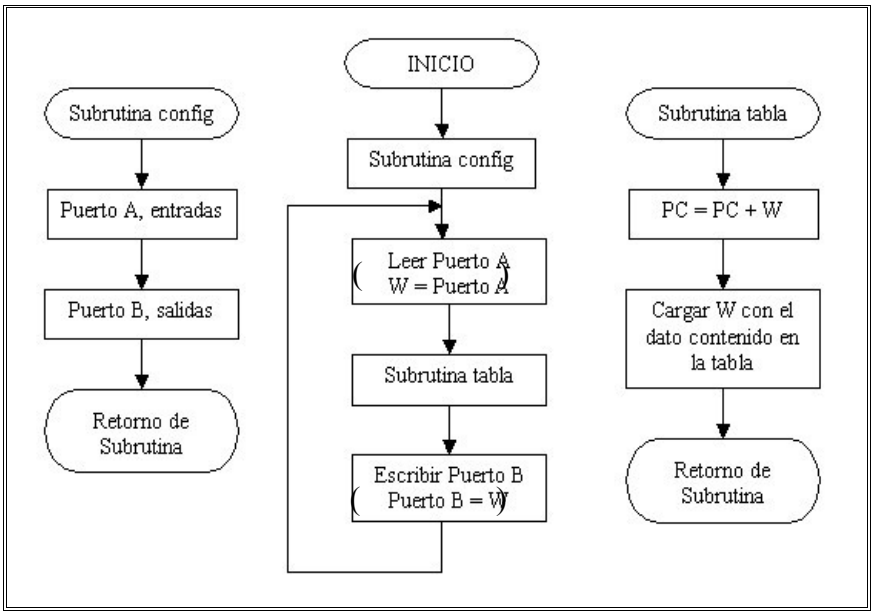


Figura 5.5.3.1. Diagrama de flujo de la práctica 5.

5.5.4 Material y equipo a utilizar

- Computadora personal (características mínimas necesarias para ejecutar las aplicaciones MPLAB y PonyProg2000)
- Software MPLAB
- Software PonyProg
- Circuito Grabador de PIC's PonyProg
- Fuente de alimentación de 5V.
- 1 PIC16F84A
- 1 Cristal Oscilador a 4 MHz
- 2 Capacitores de 22 pF
- 7 Resistencias de 10 KΩ
- 5 Resistencias de 560Ω
- 1 Resistencia de 100Ω
- 1 Display de cátodo común
- 5 Interruptores miniatura

5.5.5 Procedimiento

Como primer paso a seguir se debe copiar el siguiente programa a un editor de texto sin formato como el "Block de Notas" de Windows, o el editor del MPLAB.

```
;============================ Práctica_05 ============================


;Se implementará el PIC16F84A como un codificador para un display de 7
;segmentos.

;Se colocan cuatro interruptores en las líneas RA0, RA1, RA2, RA3,
;del puerto A y un display de 7 segmentos en el puerto B.

;El display de 7 segmentos debe conectarse en los pines del puerto B de
;la siguiente manera:

;RB0-a, RB1-b, RB2-c, RB3-d, RB4-e, RB5-f, RB6-g.

;Recuerda que el PIC no puede entregar demasiada corriente, por eso,
;cuando conectes el display de 7 segmentos, debes colocar resistencias
;limitadoras de corriente (iguales o mayores a 560 ohms)

;Mediante los interruptores se introduce un número binario de cuatro
;bits, dicho número deberá ser presentado en el display de siete
;segmentos, pero en hexadecimal

;============================ INICIO ============================

        #define  __16f84
        LIST P = 16F84A     ;Comando que indica el pic usado

;============================ Etiquetas ============================

        ESTADO   EQU   0x03  ;Dirección del reg. Estado en hex
        PC       EQU   0X02  ;Dirección del reg. PC en hex
        PORT_A   EQU   0x05  ;Dirección del puerto A en hex
        PORT_B   EQU   0x06  ;Dirección del puerto B en hex
        w        EQU   0x00  ;w = 0

; ============================ Programa ============================

        ORG      0
        goto     inicio
        ORG      5

inicio  call     config      ;Llamada a la rutina config

bucle   movf     PORT_A,w    ;w = puerto A
        call     tabla       ;Llamada a la rutina tabla
        movwf    PORT_B      ;Se envian datos al puerto B
        goto     bucle       ;Salto a inicio
```

```
; ============================== Rutinas ==============================

config   bsf      ESTADO,5          ;Se pone en 1 el bit 5 del reg. estado
         movlw    b'11111111'       ;w = b'11111111'   w = 0XFF
         movwf    PORT_A            ;Se configura el port A como entradas
         movlw    b'00000000'       ;w = b'00000000'   w = 0X00
         movwf    PORT_B            ;Se configura el port B como salidas
         bcf      ESTADO,5          ;Se pone en 0 el bit 5 del reg. estado
         return

tabla    addwf    PC                  ;pc= PC+w
                  ; .gfedcba   Codigo para 7seg catodo comun
         retlw    b'00111111'         ;7 segmentos para el 0
         retlw    b'00000110'         ;7 segmentos para el 1
         retlw    b'01011011'         ;7 segmentos para el 2
         retlw    b'01001111'         ;7 segmentos para el 3
         retlw    b'01100110'         ;7 segmentos para el 4
         retlw    b'01101101'         ;7 segmentos para el 5
         retlw    b'01111101'         ;7 segmentos para el 6
         retlw    b'00000111'         ;7 segmentos para el 7
         retlw    b'01111111'         ;7 segmentos para el 8
         retlw    b'01101111'         ;7 segmentos para el 9
         retlw    b'01110111'         ;7 segmentos para el A
         retlw    b'01111100'         ;7 segmentos para el b
         retlw    b'00111001'         ;7 segmentos para el C
         retlw    b'01011110'         ;7 segmentos para el d
         retlw    b'01111001'         ;7 segmentos para el E
         retlw    b'01110001'         ;7 segmentos para el F

; ============================== Fin ================================

         end                  ;Fin del programa

; Notas:

;Si se desea usar un display de 7 segmentos de ánodo comun,
;entonces se deberan complementar los datos de la tabla.

;Para este programa no importa la frecuencia de oscilación.
```

Una vez que se ha editado todo el programa, es necesario guardarlo en un archivo que puede ser llamado Practica_05.asm

El siguiente paso consiste en compilar el programa que se ha creado mediante el MPLAB para generar el archivo Practica_05.hex

Mediante el PonyProg2000 debe grabarse el contenido del archivo Practica_05.hex en el microcontrolador. Una vez hecho esto, se debe armar el circuito de aplicación para esta práctica, el cual se muestra en la figura 5.5.5.1.

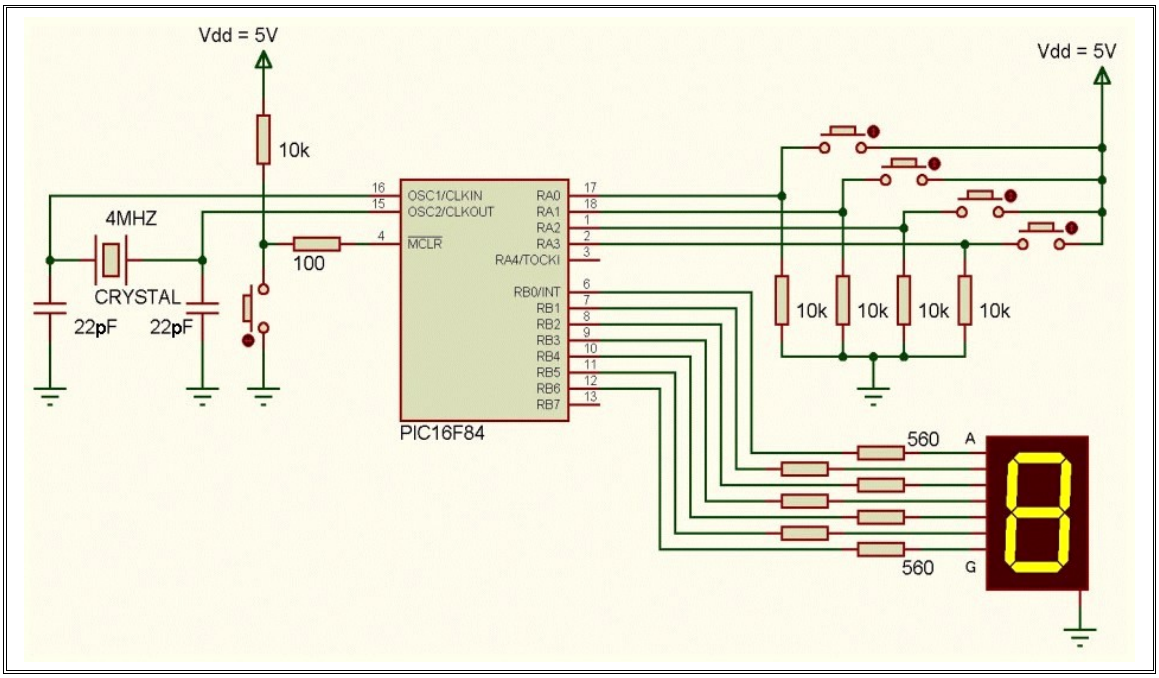


Figura 5.5.5.1. Circuito de aplicación para la práctica 5.

Cuando el circuito de aplicación esté terminado y el PIC ya tenga grabado el programa correspondiente, se inserta el dispositivo en dicho circuito y después se enciende la fuente de alimentación para probar su correcto funcionamiento.

5.5.6 Indicaciones

Para esta práctica, la palabra de configuración del PIC debe ser establecida de la siguiente forma. Código de protección deshabilitado, WDTE deshabilitado, PWRTE habilitado, oscilador XT. Los pines que no sean empleados en esta práctica, deben conectarse a Vdd por medio de una resistencia de 10KΩ, a excepción de RA4 que debe ser conectada a tierra.

5.6 Práctica No. 6 Contador de pulsos externos

5.6.1 Objetivo

Con esta práctica, el lector será capaz de implementar un circuito contador de pulsos. Dicho conteo será de 0 a 9 y es mostrado en un display de 7 segmentos.

5.6.2 Descripción

La señal externa de pulsos es proporcionada por un pulsador, el cual puede ser reemplazado por un optoacoplador, una fotorresistencia, fototransistor, fotodiodo o cualquier otro dispositivo que proporcione una variación de voltaje. El conteo se lleva a cabo mediante el incremento de un registro, el cual después será procesado por una tabla para posteriormente ser codificado y finalmente mostrado en un display de 7 segmentos.

5.6.3 Diagrama de flujo

El diagrama de flujo de la práctica número 6 se muestra en la figura 5.6.3.1.

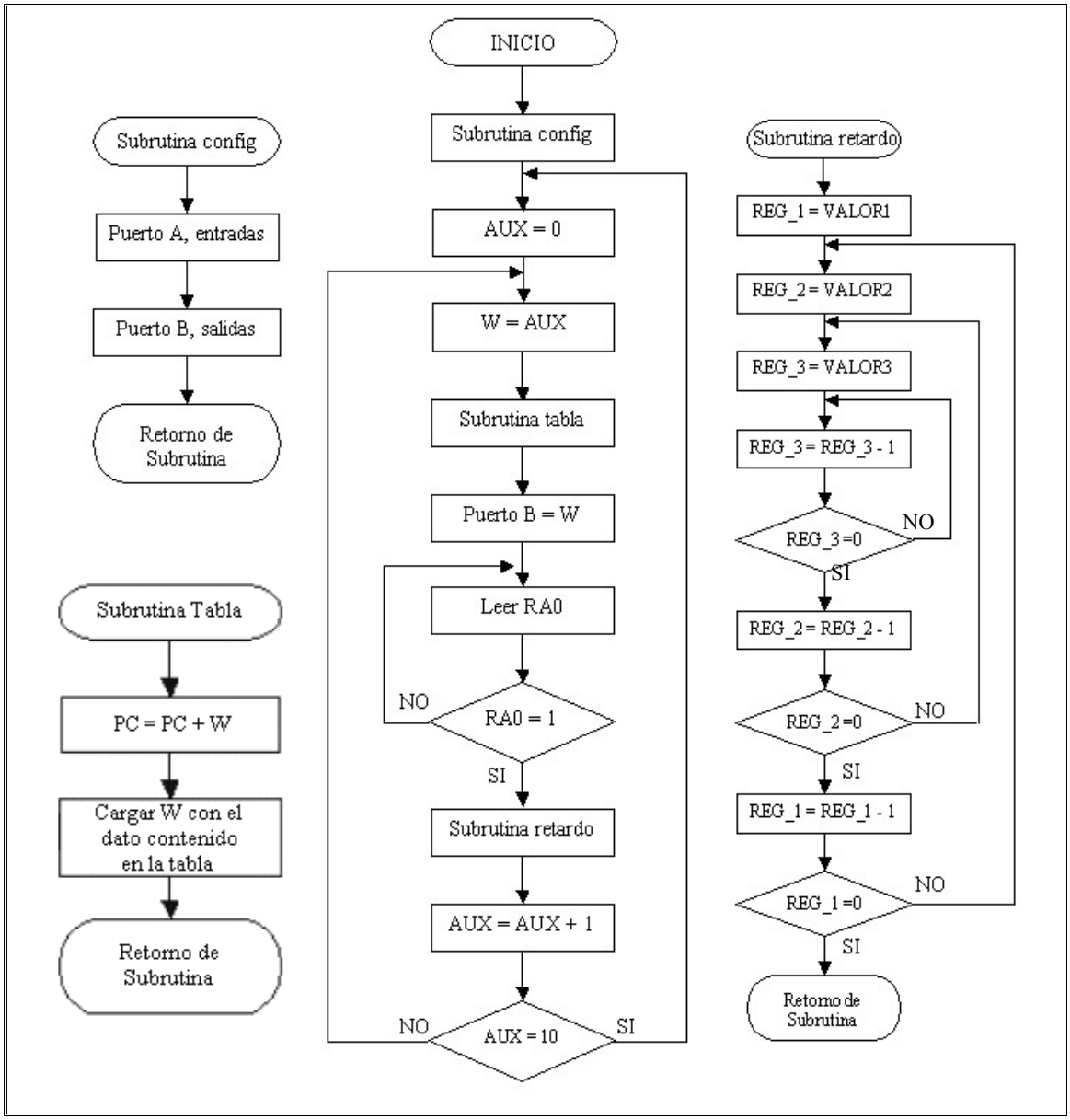


Figura 5.6.3.1. Diagrama de flujo de la práctica 6.

5.6.4 Material y equipo a utilizar

- Computadora personal (características mínimas necesarias para ejecutar las aplicaciones MPLAB y PonyProg2000)
- Software MPLAB
- Software PonyProg
- Circuito Grabador de PIC's PonyProg
- Fuente de alimentación de 5V.
- 1 PIC16F84A
- 1 Cristal Oscilador a 4 MHz
- 2 Capacitores de 22 pF
- 2 Resistencias de 10 KΩ
- 7 Resistencias de 560Ω
- 1 Resistencia de 100Ω
- 1 Display de cátodo común
- 2 Interruptores miniatura

5.6.5 Procedimiento

Como primer paso a seguir se debe copiar el siguiente programa a un editor de texto sin formato como el "Block de Notas" de Windows, o el editor del MPLAB.

```
;============================ Práctica_06 ================================

;Se programara el PIC para que funcione como un contador de pulsos
;externos (del 0 - 9).
;El conteo se podrá visualizar en un display de 7 segmentos.
;El display de 7 segmentos debe conectarse en los pines del puerto B de
;la siguiente manera:

;RB0-a, RB1-b, RB2-c, RB3-d, RB4-e, RB5-f, RB6-g.

;Recuerda que el PIC no puede entregar demasiada corriente, por esto,
;cuando conectes el display de 7 segmentos debes colocar resistencias
;limitadoras de corriente (iguales o mayores a 560 ohms).

;============================ INICIO ================================

        #define __16f84
        LIST P = 16F84A          ;Comando que indica el pic usado

;============================ Etiquetas ================================

        PC       EQU   0X02      ;Dirección del reg. PC en hex
        ESTADO   EQU   0x03      ;Dirección del reg. ESTADO
        PORT_A   EQU   0x05      ;Dirección del puerto A
        PORT_B   EQU   0x06      ;Dirección del puerto B
        w        EQU   0x00      ;w = 0
        REG_1    EQU   0x0C      ;Dirección del registro REG_1
        REG_2    EQU   0x0D      ;Dirección del registro REG_2
        REG_3    EQU   0x0E      ;Dirección del registro REG_3
        VALOR1   EQU   0x10      ;Valor que se asigna a VALOR1
        VALOR2   EQU   0x20      ;Valor que se asigna a VALOR2
        VALOR3   EQU   0x20      ;Valor que se asigna a VALOR3
        AUX      EQU   0x0F      ;Dirección del registro AUX
        DIEZ     EQU   0x0A

;============================ Programa ================================

        ORG      0               ;Vector de reset
        goto     inicio          ;Salto a inicio
        ORG      5

inicio  call     config          ;Llamada a la rutina config
borra   movlw    0x00            ;Carga w con 0x00
        movwf    AUX             ;Carga AUX con w
bucle   movf     AUX,w           ;Mueve AUX a w
        call     tabla           ;Llamada a la rutina tabla
        movwf    PORT_B          ;Mueve w al puerto B
boton   btfss    PORT_A,0        ;¿El boton esta oprimido?
        goto     boton           ;No (salto a boton)
        call     retardo         ;Si (Llamada a la rutina retardo)
boton2  btfss    PORT_A,0        ;¿El boton sigue oprimido?
        goto     sigue           ;No (salto a sigue)
        goto     boton2          ;Si (salto a boton2)
```

```
sigue    call     retardo          ;Salto a la rutina retardo
         incf     AUX,1            ;Incrementa AUX (AUX = AUX + 1)
         movf     AUX,w            ;Mueve AUX a w
         xorlw    DIEZ             ;Funcion Xor (¿es igual a 10?)
         btfsc    ESTADO,2         ;¿AUX = DIEZ? (ESTADO<2> = 0)
         goto     borra            ;Salto a borra (ESTADO<2> <> 0)
         goto     bucle            ;Salto a bucle (ESTADO<2> =  0)

;============================ Subrutinas ==============================

config   bsf      ESTADO,5         ;Se pone en 1 el bit 5 del reg. ESTADO
         movlw    b'00011111'      ;Carga w con el dato b'11111111'
         movwf    PORT_A           ;Configura el puerto A como entradas
         movlw    b'00000000'      ;Carga w con el dato b'00000000'
         movwf    PORT_B           ;Configura el puerto B como salidas
         bcf      ESTADO,5         ;Se pone en 0 el bit 5 del reg. ESTADO
         return                    ;Retorno desde subrutina

retardo  movlw    VALOR1           ;Carga w con el número 30 (VALOR1)
         movwf    REG_1            ;Mueve w al registro REG_1
tres     movlw    VALOR2           ;Carga w con el número 40 (VALOR2)
         movwf    REG_2            ;Mueve w al registro REG_2
dos      movlw    VALOR3           ;Carga w con el número 50 (VALOR3)
         movwf    REG_3            ;Mueve w al registro REG_3
uno      decfsz   REG_3            ;Decrementa el valor de REG_3 en 1
         goto     uno              ;Salto a uno
         decfsz   REG_2            ;Decrementa el valor de REG_2 en 1
         goto     dos              ;Salto a dos
         decfsz   REG_1            ;Decrementa el valor de REG_1 en 1
         goto     tres             ;Salto a tres
         return                    ;Retorno desde subrutina

tabla    addwf    PC,1             ;pc= PC+w
                  ; .gfedcba       Codigo para 7seg catodo comun
         retlw    b'00111111'      ;7 segmentos para el 0
         retlw    b'00000110'      ;7 segmentos para el 1
         retlw    b'01011011'      ;7 segmentos para el 2
         retlw    b'01001111'      ;7 segmentos para el 3
         retlw    b'01100110'      ;7 segmentos para el 4
         retlw    b'01101101'      ;7 segmentos para el 5
         retlw    b'01111101'      ;7 segmentos para el 6
         retlw    b'00000111'      ;7 segmentos para el 7
         retlw    b'01111111'      ;7 segmentos para el 8
         retlw    b'01101111'      ;7 segmentos para el 9

;============================== Fin ================================

         end
```

Notas:

```
;La frecuencia externa debe ser de 4MHz (Cristal de 4MHz).

;Si se desea usar un display de 7 segmentos de ánodo común, entonces
;Se deberan complementar los datos de la tabla.
```

Una vez que se ha editado todo el programa, es necesario guardarlo en un archivo que puede ser llamado Practica_06.asm

El siguiente paso consiste en compilar el programa que se ha creado mediante el MPLAB para generar el archivo Practica_06.hex

Mediante el PonyProg2000 debe grabarse el contenido del archivo Practica_06.hex en el microcontrolador. Una vez hecho esto, se debe armar el circuito de aplicación para esta práctica, el cual se muestra en la figura 5.6.5.1.

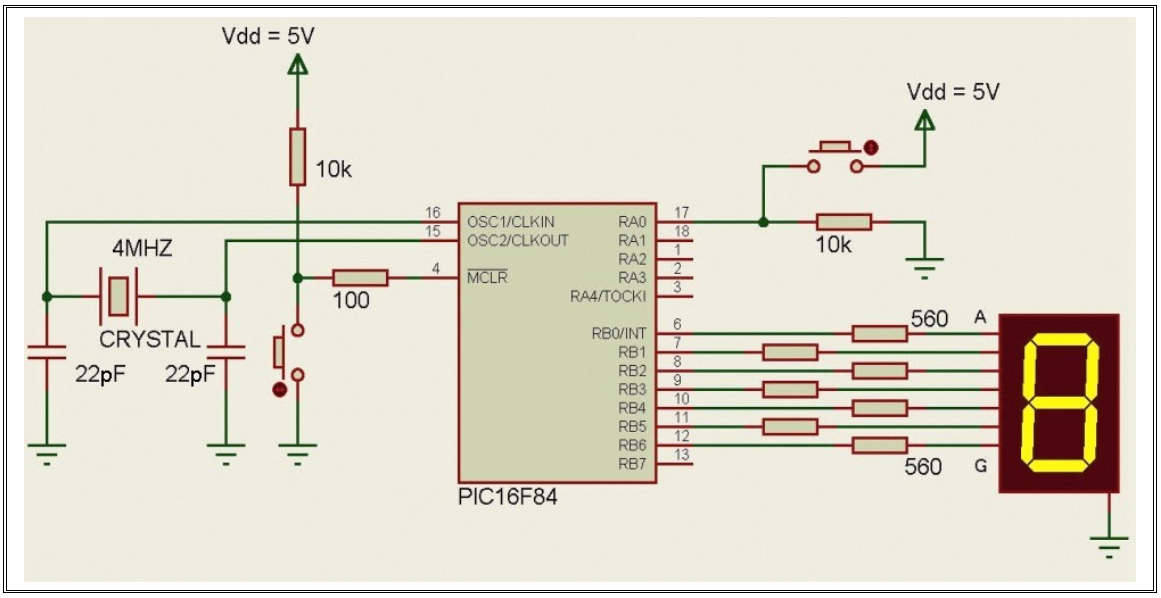


Figura 5.6.5.1. Circuito de aplicación para la práctica 6.

Cuando el circuito de aplicación esté terminado y el PIC ya tenga grabado el programa correspondiente, se inserta el dispositivo en dicho circuito y después se enciende la fuente de alimentación para probar su correcto funcionamiento.

5.6.6 Indicaciones

Para esta práctica, la palabra de configuración del PIC debe ser establecida de la siguiente forma. Código de protección deshabilitado, WDTE deshabilitado, PWRTE habilitado, oscilador XT. Los pines que no sean empleados en esta práctica, deben conectarse a Vdd por medio de una resistencia de 10KΩ.

5.7 Práctica No. 7 Escritura en la memoria de datos EEPROM

5.7.1 Objetivo

Se podrá apreciar la escritura de datos en la memoria EEPROM, la cual podrá leerse posteriormente con la utilización del PonyProg2000.

5.7.2 Descripción

Se desea almacenar el dato 0x44 en las posiciones 31 y 32 de la memoria de datos EEPROM. Una vez logrado dicho proceso, se encenderá un LED conectado a RB0, indicando que dicho proceso ha terminado; posteriormente, el pic será colocado en el PonyProg, para que el dispositivo sea leido. En él se prodrá apreciar la memoria de programa y la memoria de datos EEPROM que se encuentra al final de la ventana y resalta por tener otro color. En dicha memoria se podrá apreciar que dos casillas poseen el dato 0x44 en formato hexadecimal.

5.7.3 Diagrama de flujo

El diagrama de flujo de la práctica número 7 se muestra en la figura 5.7.3.1.

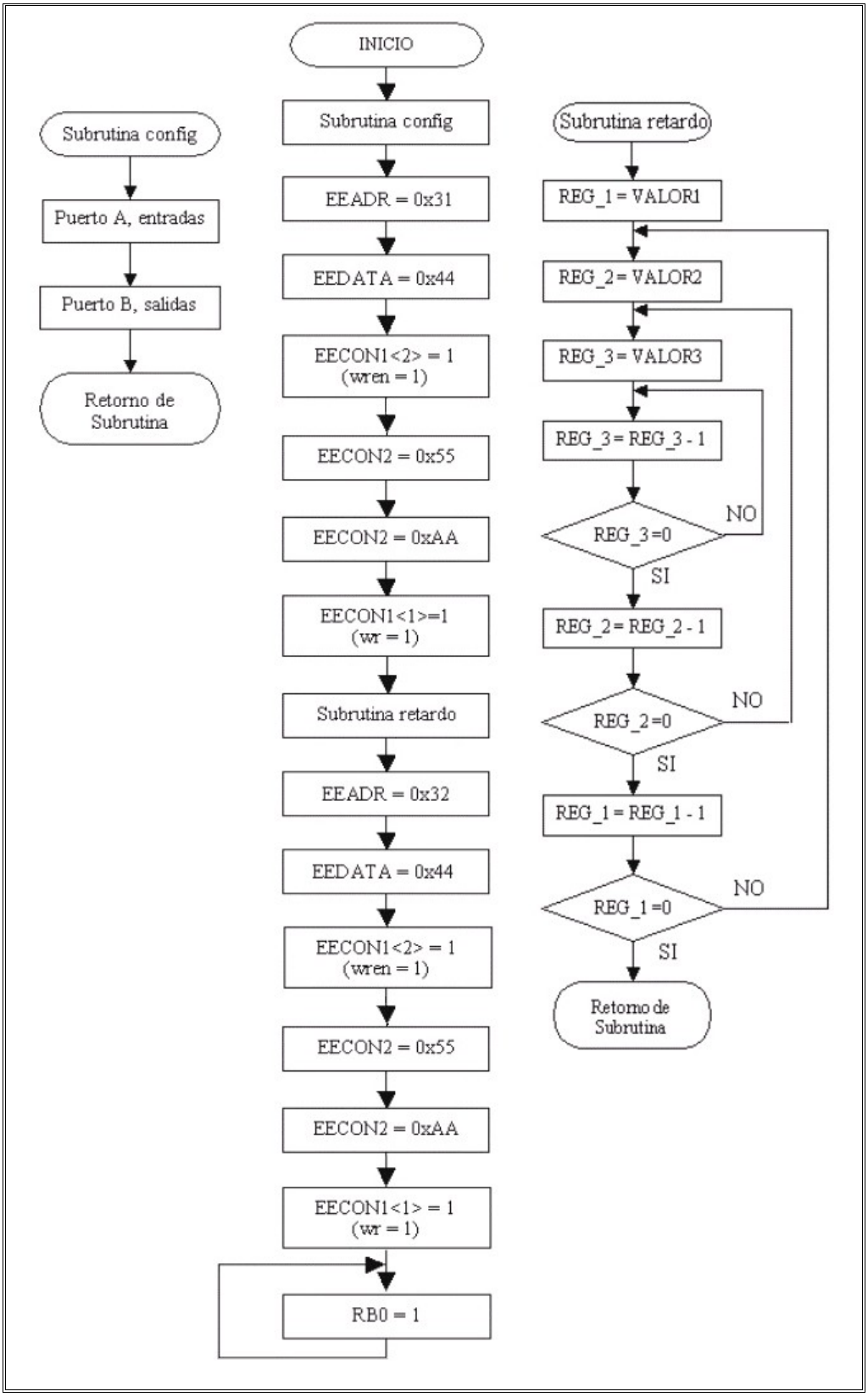


Figura 5.7.3.1. Diagrama de flujo de la práctica 7.

5.7.4 Material y equipo a utilizar

- Computadora personal (características mínimas necesarias para ejecutar las aplicaciones MPLAB y PonyProg2000)
- Software MPLAB
- Software PonyProg
- Circuito Grabador de PIC's PonyProg
- Fuente de alimentación de 5V.
- 1 PIC16F84A
- 1 Cristal Oscilador a 4 MHz
- 2 Capacitores de 22 pF
- 1 Resistencia de 10KΩ
- 1 Resistencia de 560Ω
- 1 Resistencia de 100Ω
- 1 Led

5.7.5 Procedimiento

Como primer paso a seguir se debe copiar el siguiente programa a un editor de texto sin formato como el "Block de Notas" de Windows, o el editor del MPLAB.

```
;========================== Práctica_7 ==============================

;Se desea almacenar el dato 0x44 en las posiciones 31 y 32 de la
;memoria de datos EEPROM. Una vez logrado dicho proceso, se encenderá
;un LED conectado a RB0, indicando que dicho proceso ha terminado.
;Posteriormente, el pic será colocado en el PonyProg, para que el
;dispositivo sea leido. En él se podrá apreciar la memoria de programa
;y la memoria de datos EEPROM que se encuentra al final de la ventana y
;resalta por tener otro color. En dicha memoria se podrá apreciar que
;dos casillas poseen el dato 0x44 en formato hexadecimal.


;============================ INICIO ================================

        #define __16f84
        LIST P = 16F84A          ;Comando que indica el pic usado

;============================ Etiquetas =============================

        w       EQU   0x00       ;w = 0
        STATUS  EQU   0x03       ;Dirección del reg. ESTADO
        PORT_A  EQU   0x05       ;Dirección del puerto A
        PORT_B  EQU   0x06       ;Dirección del puerto B
        EEADR   EQU   0x09       ;Dirección del reg. EEADR
        EEDATA  EQU   0x08       ;Dirección del reg. EEDATA
        EECON1  EQU   0x08       ;Dirección del reg. EECON1
        EECON2  EQU   0x09       ;Dirección del reg. EECON2
        INTCON  EQU   0x0B       ;Dirección del reg. INTCON
        REG_1   EQU   0x0C       ;Dirección del registro REG_1
        REG_2   EQU   0x0D       ;Dirección del registro REG_2
        REG_3   EQU   0x0E       ;Dirección del registro REG_3
        VALOR1  EQU   0x40       ;Valor que se asigna a VALOR1
        VALOR2  EQU   0x40       ;Valor que se asigna a VALOR2
        VALOR3  EQU   0x50       ;Valor que se asigna a VALOR3

;============================ Programa ==============================

        ORG     0                ;Vector de reset
        goto    inicio           ;Salto a inicio
        ORG     5

inicio  call    config           ;Llamada a la rutina config
        clrf    PORT_B           ;PORT_B = 0x00
        bcf     STATUS,5         ;Pasa al banco 0
        movlw   0x31             ;w = 0x31
        movwf   EEADR            ;EEADR = w
        movlw   0x44             ;w = 0x44
        movwf   EEDATA           ;EEDATA = w
        bsf     STATUS,5         ;Pasa al banco 1
        bcf     INTCON,7         ;GIE=0 para prohibir interrupciones
        bsf     EECON1,2         ;wren=1 para permitir escritura en
                                 ;EEPROM
```

```
;Secuencia de seguridad
         movlw    0x55              ;w = 0x55
         movwf    EECON2            ;EECON2 = w
         movlw    0xAA              ;W = 0xAA
         movwf    EECON2            ;EECON2 = w
         bsf      EECON1,1          ;wr=1 y comienza el ciclo de escritura
         call     retardo
         bcf      STATUS,5          ;Pasa al banco 0
         movlw    0x32              ;w = 0x32
         movwf    EEADR             ;EEADR = w
         movlw    0x44              ;w = 0x44
         movwf    EEDATA            ;EEDATA = w
         bsf      STATUS,5          ;Pasa al banco 1
         bcf      INTCON,7          ;GIE=0 para prohibir interrupciones
         bsf      EECON1,2          ;wren=1 para permitir escritura en
                                    ;EEPROM
;Secuencia de seguridad

         movlw    0x55              ;w = 0x55
         movwf    EECON2            ;EECON2 = w
         movlw    0xAA              ;W = 0xAA
         movwf    EECON2            ;EECON2 = w
         bsf      EECON1,1          ;wr=1 y comienza el ciclo de escritura
         call     retardo
         bcf      STATUS,5
         movlw    b'00000001'
ciclo    movwf    PORT_B
         goto     ciclo
;============================ Subrutinas ==============================

config   bsf      STATUS,5          ;Se pone en 1 el bit 5 del reg. ESTADO
         movlw    b'11111111'       ;Carga w con el dato b'11111111'
         movwf    PORT_A            ;Configura el puerto A como entradas
         movlw    b'00000000'       ;Carga w con el dato b'00000000'
         movwf    PORT_B            ;Configura el puerto B como salidas
         bcf      STATUS,5          ;Se pone en 0 el bit 5 del reg. ESTADO
         return                     ;Retorno desde subrutina
retardo  movlw    VALOR1            ;Carga w con el número 30 (VALOR1)
         movwf    REG_1             ;Mueve w al registro REG_1
tres     movlw    VALOR2            ;Carga w con el número 40 (VALOR2)
         movwf    REG_2             ;Mueve w al registro REG_2
dos      movlw    VALOR3            ;Carga w con el número 50 (VALOR3)
         movwf    REG_3             ;Mueve w al registro REG_3
uno      decfsz   REG_3             ;Decrementa el valor de REG_3 en 1
         goto     uno               ;Salto a uno
         decfsz   REG_2             ;Decrementa el valor de REG_2 en 1
         goto     dos               ;Salto a dos
         decfsz   REG_1             ;Decrementa el valor de REG_1 en 1
         goto     tres              ;Salto a tres
         return                     ;Retorno desde subrutina
;============================== Fin ================================
         end

;Notas: La frecuencia externa debe ser de 4MHz (Cristal de 4MHz).
```

Una vez que se ha editado todo el programa, es necesario guardarlo en un archivo que puede ser llamado Practica_7.asm

El siguiente paso consiste en compilar el programa que se ha creado mediante el MPLAB para generar el archivo Practica_7.hex

Mediante el PonyProg2000 debe grabarse el contenido del archivo Practica_7.hex en el microcontrolador. Una vez hecho esto, se debe armar el circuito de aplicación para esta práctica, el cual se muestra en la figura 5.7.5.1.

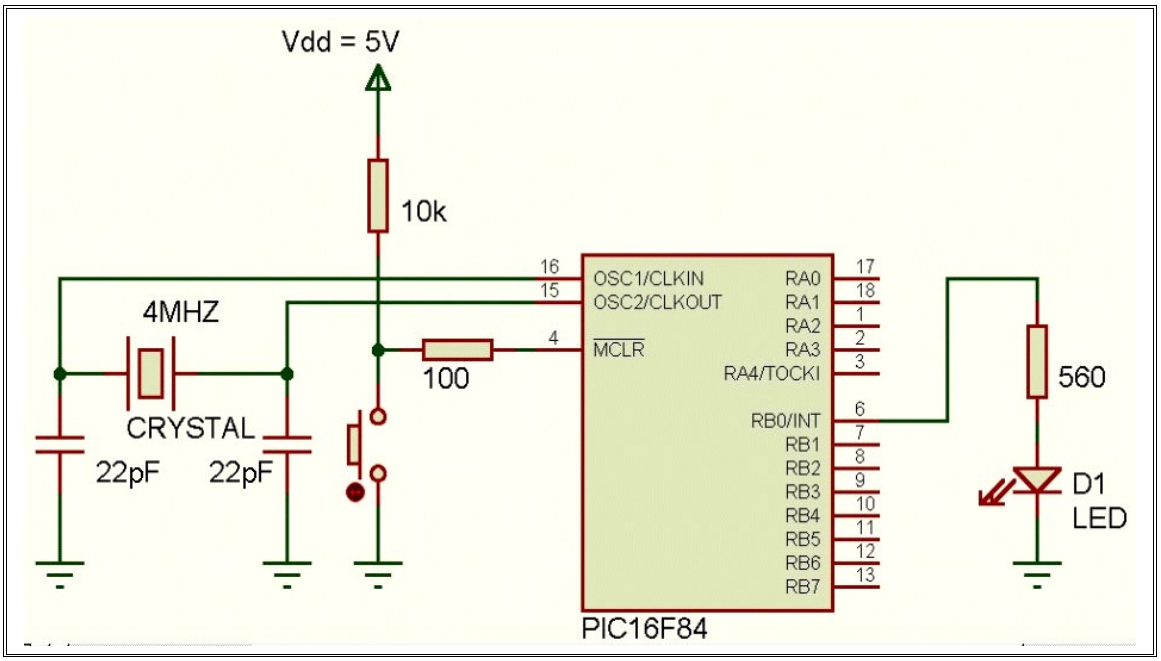


Figura 5.7.5.1. Circuito de aplicación de la práctica 7.

5.7.6 Indicaciones

Para esta práctica, la palabra de configuración del PIC debe ser establecida de la siguiente forma. Código de protección deshabilitado, WDTE deshabilitado, PWRTE habilitado, oscilador XT. Los pines que no sean empleados en esta práctica, deben conectarse a Vdd por medio de una resistencia de 10KΩ.

5.8 Práctica No. 8 Parpadeo de un led utilizando la interrupción por desbordamiento del temporizador TMR0

5.8.1 Objetivo

Se observará el encendido y apagado de un Led a una frecuencia de 1 Hz. Dicha temporización es lograda debido a la interrupción por desbordamiento del temporizador TMR0.

5.8.2 Descripción

Se encuentra un Led conectado al pin RB0 del puerto B, el cual se encontrará parpadeando a la frecuencia de 1 Hz. Esto es posible gracias a la utilización de la interrupción por desbordamiento del temporizador TMR0, el cual cada vez que se desborde desviará el flujo del programa principal y decrementará el contenido de un registro auxiliar 12 veces, ocasionando el encendido y apagado del Led.

5.8.3 Diagrama de flujo

El diagrama de flujo de la práctica número 8 se muestra en la figura 5.8.3.1.

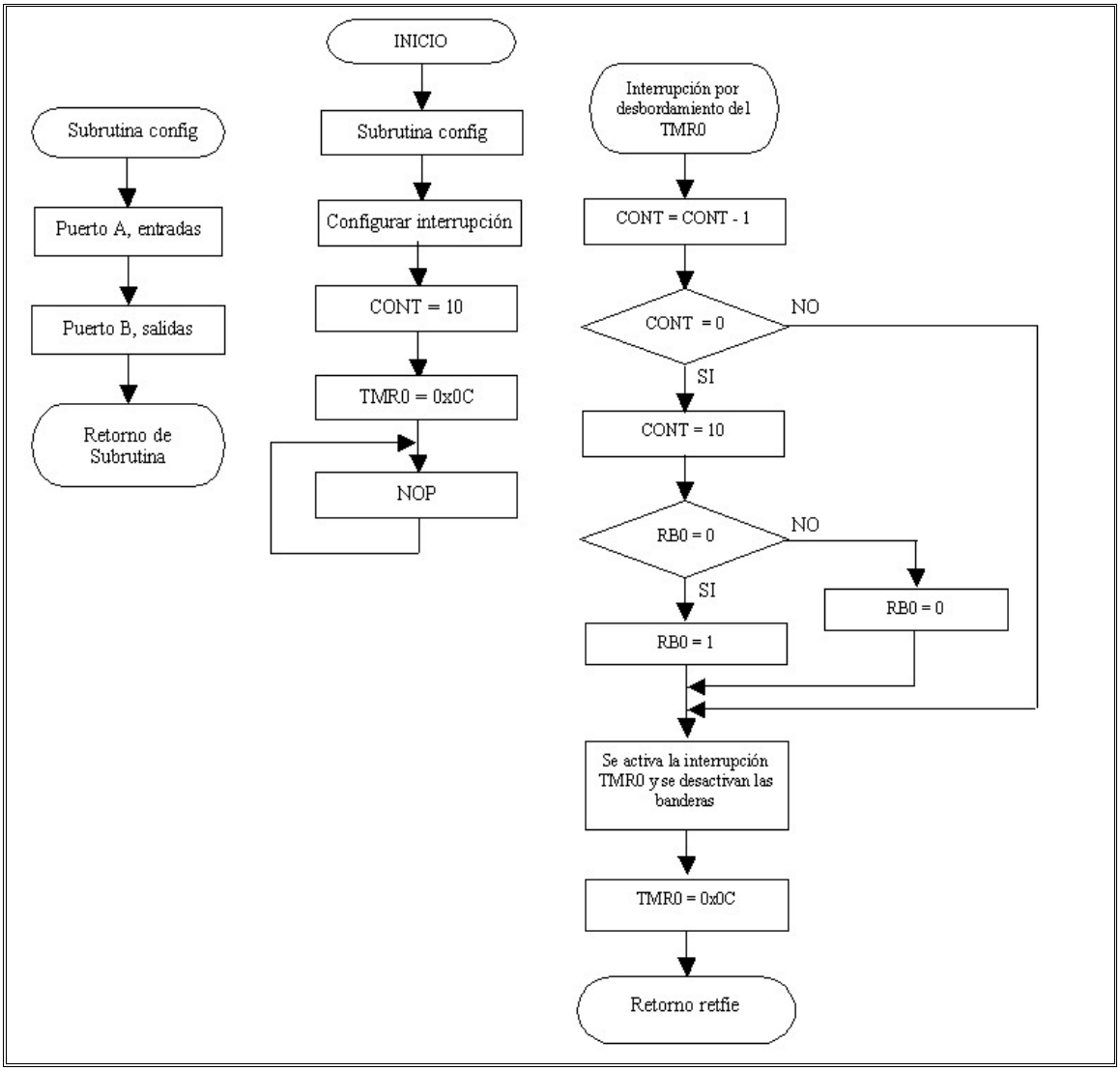


Figura 5.8.3.1. Diagrama de flujo de la práctica 8.

5.8.4 Material y equipo a utilizar

- Computadora personal (características mínimas necesarias para ejecutar las aplicaciones MPLAB y PonyProg2000)
- Software MPLAB
- Software PonyProg
- Circuito Grabador de PIC's PonyProg
- Fuente de alimentación de 5V.
- 1 PIC16F84A
- 1 Cristal Oscilador a 4 MHz
- 2 Capacitores de 22 pF
- 1 Resistencia de 10KΩ
- 1 Resistencia de 560Ω
- 1 Resistencia de 100Ω
- 1 Led

5.8.5 Procedimiento

Como primer paso a seguir se debe copiar el siguiente programa a un editor de texto sin formato como el "Block de Notas" de Windows, o el editor del MPLAB.

```
;============================ Práctica_8 ==============================
;En esta practica se propone realizar un circuito electrónico que
;encienda y apague un led a una frecuencia aproximad de 1 segundo.

;El led esta conectado a RB0 mediante una resistencia limitadora.

;Esta practica utiliza la interrupción por desbordamiento del TMR0.

;============================ INICIO ==================================

        LIST P=16F84A           ;Comando que indica el pic usado

;============================ Etiquetas ===============================

        w       EQU   0x00
        OPT     EQU   0x01      ;Dirección del reg. OPTION
        TMR0    EQU   0x01      ;Dirección del reg. TMR0
        STATUS  EQU   0x03      ;Dirección del reg. ESTADO
        PORT_A  EQU   0x05      ;Dirección del puerto A
        PORT_B  EQU   0x06      ;Dirección del puerto B
        INTCON  EQU   0x0B      ;Dirección del reg. INTCON
        CONT    EQU   0x10      ;Dirección del reg. CONT

;============================ Programa ================================

        ORG     0               ;Vector de reset
        goto    inicio          ;Salto a inicio
        ORG     4               ;Vector de interrupciones
        goto    inter           ;Salto a inter
        ORG     5

inicio  call    config
        clrf    PORT_A          ;Se limpia el puerto A
        clrf    PORT_B          ;Se limpia el puerto b
        movlw   b'10100000'     ;Carga w con b'10100000'
        movwf   INTCON          ;Permiso de interrupción TMR0
        movlw   0x10
        movwf   CONT            ;Se carga CONT con 0x10
        movlw   0x0c
        movwf   TMR0            ;Se carga TMR0 con 0x0c
ciclo   nop
        goto    ciclo           ;Salto a ciclo

;============================ Subrutinas ==============================

config  bsf     STATUS,5        ;STATUS<5>=1, se pasa al banco 1
        clrf    PORT_A          ;PORT_A como salidas
        clrf    PORT_B          ;PORT_B como salidas
        movlw   b'00000111'     ;Configura el registro OPTION
        movwf   OPT
        bcf     STATUS,5        ;Selecciona el banco 0
        return
```

```
;========================== Interrupciones ============================

inter    decfsz   CONT,1           ;Decrementa CONT y brinca si vale 0
         goto     seguir           ;Salto a seguir
cont_0   movlw    0x10             ;Si CONT = 0, se carga con 16
         movwf    CONT
         btfsc    PORT_B,0         ;Si RB0 = 0, brinca
         goto     rb7_1
         bsf      PORT_B,0         ;Si RB0 = 0 se invierte el valor
         goto     seguir
rb7_1    bcf      PORT_B,0         ;Si RB0 = 0 se invierte el valor
seguir   movlw    b'10100000'      ;Se restaura INTCON, desactivar inte.
         movwf    INTCON
         movlw    0x0C             ;Se recarga TMR0 con 12 en decimal
         movwf    TMR0
         retfie                    ;Retorno de interrupción
;============================== Fin ===================================
         end
```

Nota: La frecuencia del cristal debe ser de 4 MHz

Una vez que se ha editado todo el programa, es necesario guardarlo en un archivo que puede ser llamado Practica_8.asm

El siguiente paso consiste en compilar el programa que se ha creado mediante el MPLAB para generar el archivo Practica_8.hex

Mediante el PonyProg2000 debe grabarse el contenido del archivo Practica_8.hex en el microcontrolador. Una vez hecho esto, se debe armar el circuito de aplicación para esta práctica, el cual se muestra en la figura 5.8.5.1.

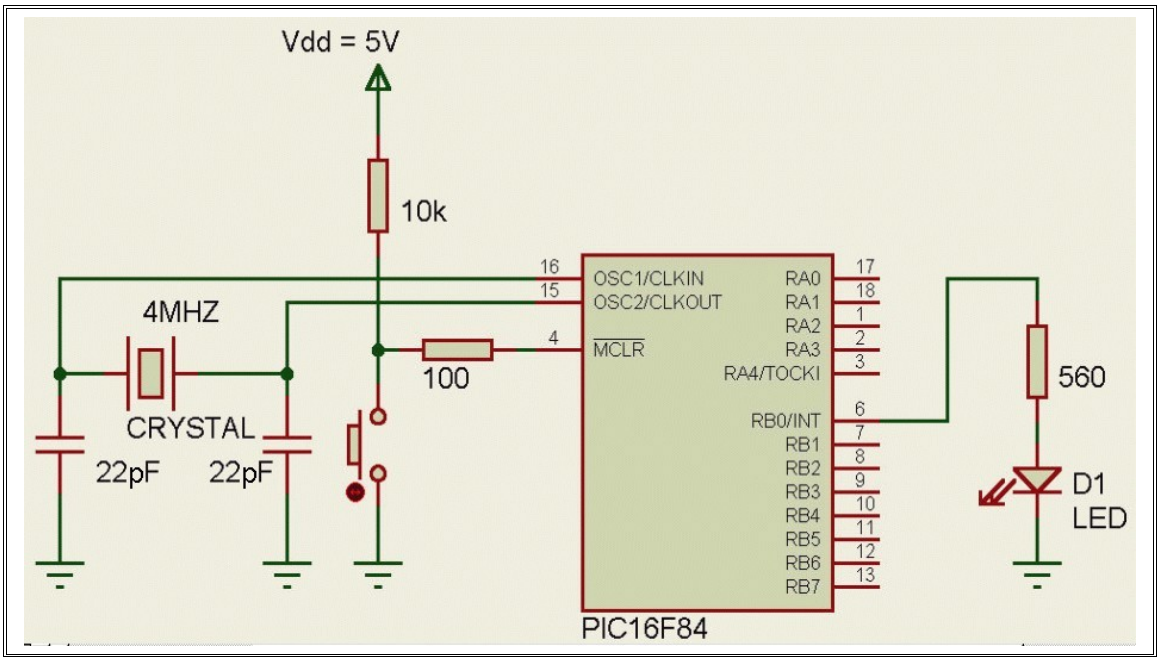


Figura 5.8.5.1. Circuito de aplicación de la práctica 8.

5.8.6 Indicaciones

Para esta práctica, la palabra de configuración del PIC debe ser establecida de la siguiente forma. Código de protección deshabilitado, WDTE deshabilitado, PWRTE habilitado, oscilador XT. Los pines que no sean empleados en esta práctica, deben conectarse a Vdd por medio de una resistencia de 10KΩ.

5.9 Práctica No. 9 Controlador para un motor a pasos

5.9.1 Objetivo

El lector implementará un driver para un motor a pasos de cinco hilos, mediante una secuencia predefinida en el microcontrolador.

5.9.2 Descripción

El PIC será programado para que funcione como un driver para un motor a pasos. El motor debe estar conectado a los arreglos de los transistores, los cuales serán controlados por las señales de salida del puerto B. En el puerto A, están conectados dos pulsadores en los pines RA0 y RA1, los cuales se utilizarán para indicar el sentido de giro que se desea obtener. Si se presiona RA0, el motor girará en sentido de las manecillas del reloj, y si se presiona RA1, girará en sentido inverso a las manecillas.

5.9.3 Diagrama de flujo

El diagrama de flujo de la práctica número 9 se muestra en la figura 5.9.3.1. La subrutina retardo que se menciona en este diagrama es similar a la subrutina retardo de los programas de las prácticas anteriores.

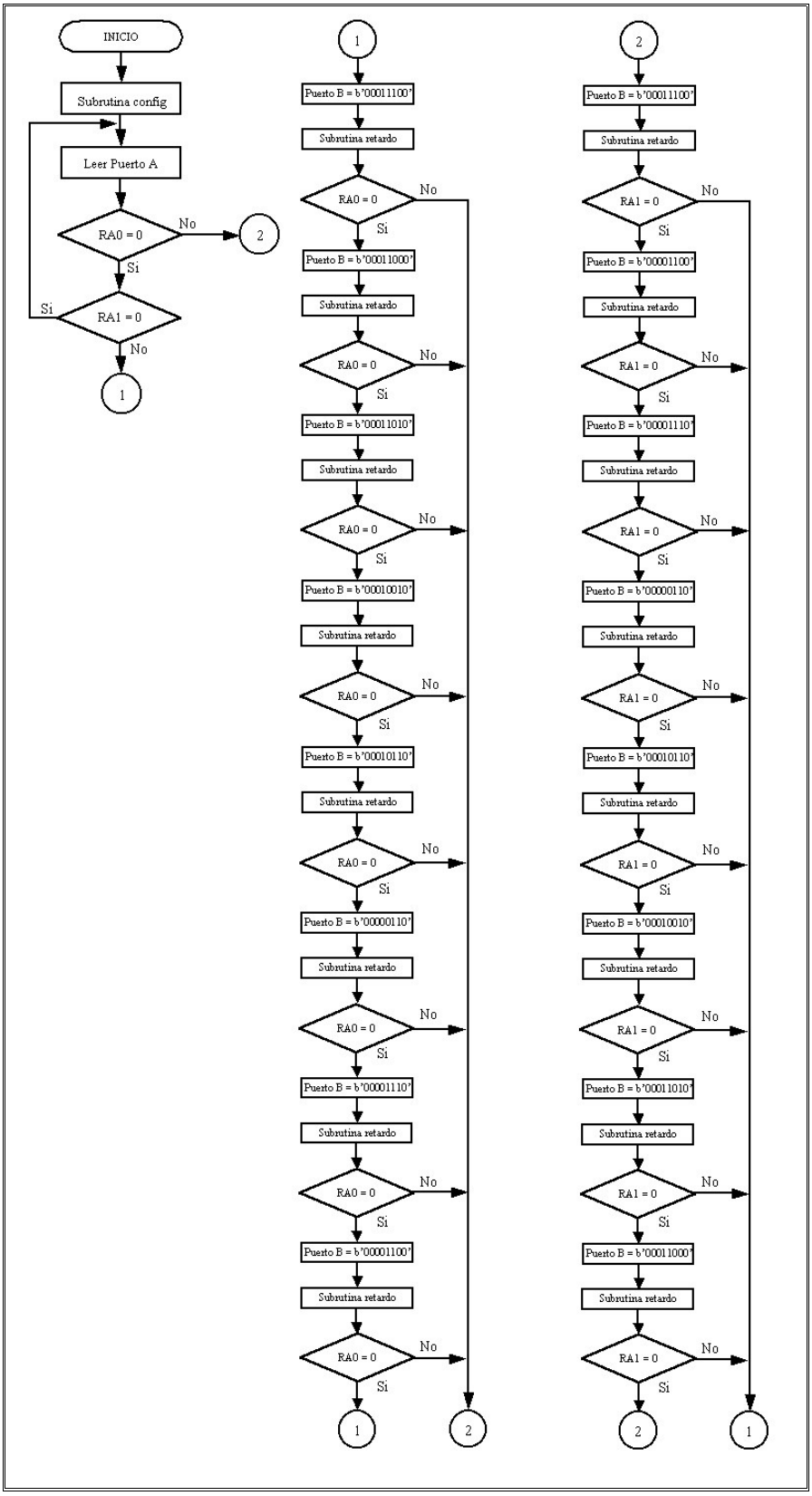


Figura 5.9.3.1. Diagrama de flujo de la práctica 9.

5.9.4 Material y equipo a utilizar

- Computadora personal (características mínimas necesarias para ejecutar las aplicaciones MPLAB y PonyProg2000)
- Software MPLAB y PonyProg
- Circuito Grabador de PIC's PonyProg
- Fuente de alimentación de 5V.
- 1 PIC16F84A
- 1 Cristal Oscilador a 4 MHz
- 2 Capacitores de 22 pF
- 3 Resistencias de 10 KΩ
- 4 Resistencias de 1KΩ
- 1 Resistencia de 100Ω
- 1 Motor Paso a Paso
- 3 Interruptores miniatura
- 4 Transistores BD135
- 4 Transistores BD136

5.9.5 Procedimiento

Como primer paso a seguir se debe copiar el siguiente programa a un editor de texto sin formato como el "Block de Notas" de Windows, o el editor del MPLAB.

```
;============================ Práctica_09 ==============================

;Se implementa el PIC16F84A como un dispositivo que controle el sentido
;de giro de un motor a pasos de 5 hilos.

;Los pines RB1, RB2, RB3, RB4, estan conectados a una configuración de
;transistores, los cuales estan conectados al motor a pasos

;Los pines RA0, RA1 estan conectados a unos pulsadores los cuales al ser
;presionados determinan el sentido del giro del motor a pasos

;============================ INICIO ================================
        #define __16f84
        LIST P = 16F84A          ;Comando que indica el pic usado
;============================ Etiquetas ==============================

        ESTADO   EQU   0x03      ;Dirección del reg. ESTADO
        PORT_A   EQU   0x05      ;Dirección del puerto A
        PORT_B   EQU   0x06      ;Dirección del puerto B
        w        EQU   0x00      ;w = 0
        REG_1    EQU   0x0C      ;Dirección del registro REG_1
        REG_2    EQU   0x0D      ;Dirección del registro REG_2
        REG_3    EQU   0x0E      ;Dirección del registro REG_3
        VALOR1   EQU   0x10      ;Valor que se asigna a VALOR1
        VALOR2   EQU   0x20      ;Valor que se asigna a VALOR2
        VALOR3   EQU   0x30      ;Valor que se asigna a VALOR3

;============================ Programa ===============================
        ORG      0               ;Vector de reset
        goto     inicio          ;Salto a inicio
        ORG      6
inicio  call     config          ;Llamada a la rutina config

ciclo   btfsc    PORT_A,0        ;¿PORT_A<0> = 0?
        goto     derecha         ;No, salto a derecha
        btfsc    PORT_A,1        ;Si, ¿PORT_A<1> = 0?
        goto     izquier         ;No, salto a izquier
        goto     ciclo           ;Si, salto a ciclo

derecha movlw    b'00011100'     ;w = b'00011100'
        movwf    PORT_B          ;PORT_B = w
        call     retardo         ;Llamada a retardo
        btfsc    PORT_A,1        ;¿PORT_A<1> = 0?
        goto     izquier         ;No, salto a izquier

        movlw    b'00011000'     ;Si, mueve b'00011000' a w
        movwf    PORT_B          ;Mueve w a PORT_B, PORT_B = w
        call     retardo         ;Llamada a la subrutina retardo
        btfsc    PORT_A,1        ;¿PORT_A<1> = 0?
        goto     izquier         ;No, salto a izquier

        movlw    b'00011010'
        movwf    PORT_B
        call     retardo
        btfsc    PORT_A,1
        goto     izquier
```

```
          movlw    b'00010010'
          movwf    PORT_B
          call     retardo
          btfsc    PORT_A,1
          goto     izquier

          movlw    b'00010110'
          movwf    PORT_B
          call     retardo
          btfsc    PORT_A,1
          goto     izquier

          movlw    b'00000110'
          movwf    PORT_B
          call     retardo
          btfsc    PORT_A,1
          goto     izquier

          movlw    b'00001110'
          movwf    PORT_B
          call     retardo
          btfsc    PORT_A,1
          goto     izquier

          movlw    b'00001100'
          movwf    PORT_B
          call     retardo
          btfsc    PORT_A,1
          goto     izquier          ;Salto a izquierda

          goto     derecha          ;Salto a derecha

izquier   movlw    b'00011100'      ;w = b'00011100'
          movwf    PORT_B           ;PORT_B = w
          call     retardo          ;Llamada a retardo
          btfsc    PORT_A,0         ;¿PORT_A<0> = 0?
          goto     derecha          ;No, salto a derecha

          movlw    b'00001100'      ;Mueve b'00001100' a w
          movwf    PORT_B           ;Mueve w a PORT_B, PORT_B = w
          call     retardo          ;Llamada a la subrutina retardo
          btfsc    PORT_A,0         ;¿PORT_A<0> = 0?
          goto     derecha          ;No, salto a derecha

          movlw    b'00001110'
          movwf    PORT_B
          call     retardo
          btfsc    PORT_A,0
          goto     derecha

          movlw    b'00000110'
          movwf    PORT_B
          call     retardo
          btfsc    PORT_A,0
          goto     derecha
```

```
         movlw    b'00010110'
         movwf    PORT_B
         call     retardo
         btfsc    PORT_A,0
         goto     derecha

         movlw    b'00010010'
         movwf    PORT_B
         call     retardo
         btfsc    PORT_A,0
         goto     derecha

         movlw    b'00011010'
         movwf    PORT_B
         call     retardo
         btfsc    PORT_A,0
         goto     derecha

         movlw    b'00011000'
         movwf    PORT_B
         call     retardo
         btfsc    PORT_A,0
         goto     derecha          ;salto a derecha

         goto     izquier          ;Salto a izquier

;============================== Subrutinas ==============================

config   bsf      ESTADO,5         ;Se pone en 1 el bit 5 del reg. ESTADO
         movlw    b'11111111'      ;Carga w con el dato b'11111111'
         movwf    PORT_A           ;Configura el puerto A como entradas
         movlw    b'00000000'      ;Carga w con el dato b'00000000'
         movwf    PORT_B           ;Configura el puerto B como salidas
         bcf      ESTADO,5         ;Se pone en 0 el bit 5 del reg. ESTADO
         return                    ;Retorno desde subrutina

retardo  movlw    VALOR1           ;Carga w con el número 30 (VALOR1)
         movwf    REG_1            ;Mueve w al registro REG_1
tres     movlw    VALOR2           ;Carga w con el número 40 (VALOR2)
         movwf    REG_2            ;Mueve w al registro REG_2
dos      movlw    VALOR3           ;Carga w con el número 50 (VALOR3)
         movwf    REG_3            ;Mueve w al registro REG_3
uno      decfsz   REG_3            ;Decrementa el valor de REG_3 en 1
         goto     uno              ;Salto a uno
         decfsz   REG_2            ;Decrementa el valor de REG_2 en 1
         goto     dos              ;Salto a dos
         decfsz   REG_1            ;Decrementa el valor de REG_1 en 1
         goto     tres             ;Salto a tres
         return                    ;Retorno desde subrutina

;================================ Fin ================================

         end
```

Nota: La frecuencia externa debe ser de 4MHz (Cristal de 4MHz).

Una vez que se ha editado todo el programa, es necesario guardarlo en un archivo que puede ser llamado Practica_09.asm

El siguiente paso consiste en compilar el programa que se ha creado mediante el MPLAB para generar el archivo Practica_09.hex

Mediante el PonyProg2000 debe grabarse el contenido del archivo Practica_09.hex en el microcontrolador. Una vez hecho esto, se debe armar el circuito de aplicación para esta práctica, el cual se muestra en la figura 5.9.5.1.

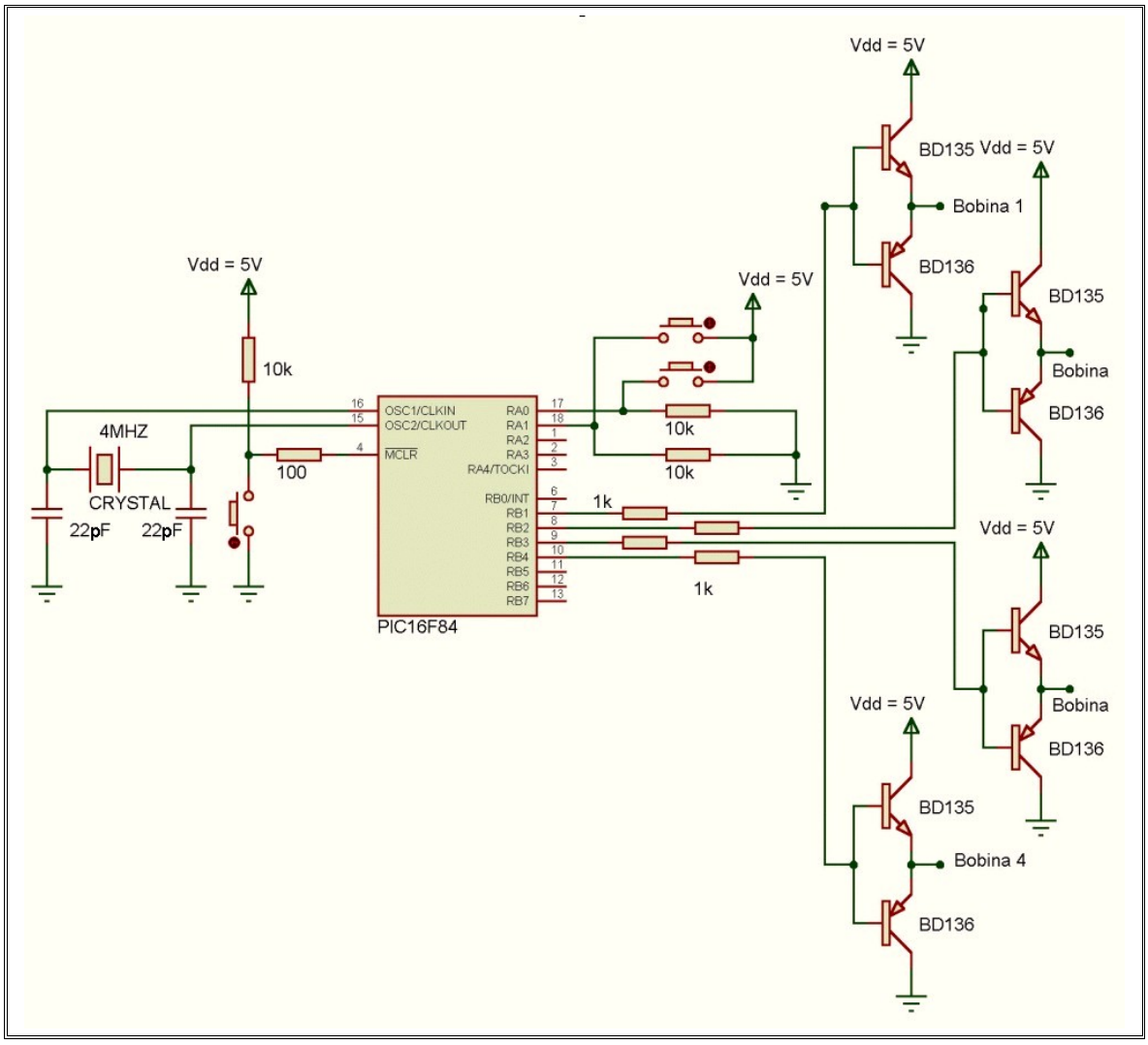


Figura 5.9.5.1. Circuito de aplicación para la práctica 9.

Cuando el circuito de aplicación esté terminado y el PIC ya tenga grabado el programa correspondiente, se inserta el dispositivo en dicho circuito y después se enciende la fuente de alimentación para probar su correcto funcionamiento.

5.9.6 Indicaciones

Para esta práctica, la palabra de configuración del PIC debe ser establecida de la siguiente forma. Código de protección deshabilitado, WDTE deshabilitado, PWRTE habilitado, oscilador XT. Los pines que no sean empleados en esta práctica, deben conectarse a Vdd por medio de una resistencia de 10KΩ..

5.10 Práctica No. 10 Control de giro de un motor de CD

5.10.1 Objetivo

Implementar un microcontrolador para el control de giro y conteo de vueltas de un motor de corriente directa, así como la utilización del modo de bajo consumo de energía SLEEP.

5.10.2 Descripción

Se utilizará el microcontrolador para generar una señal de mando a un puente H (arreglo de transistores), el cual controlará el giro de un pequeño motor de CD. También se utilizará un pequeño disco perforado en una orilla, el cual está fijado al rotor del motor de CD, esto con el objetivo de que un optoacoplador sea el elemento que detecte el número de vueltas que realiza el disco al interrumpirse el haz de luz. Al no haber luz en el receptor del optoacoplador mandará un cero lógico al pin RA4 del puerto A. Cuando exista el paso de luz en el receptor, se producirá un “uno” lógico que se enviará al pin RA4. El motor dará cien vueltas hacia la derecha, se detendrá un pequeño instante y dará cien vueltas hacia la izquierda. Cuando el proceso termine, entrará al modo de SLEEP.

5.10.3 Diagrama de flujo

El diagrama de flujo de la práctica número 10 se muestra en la figura 5.10.3.1. Cabe mencionar que la subrutina retardo en los diagramas de flujo es la misma que en todas las prácticas sugeridas.

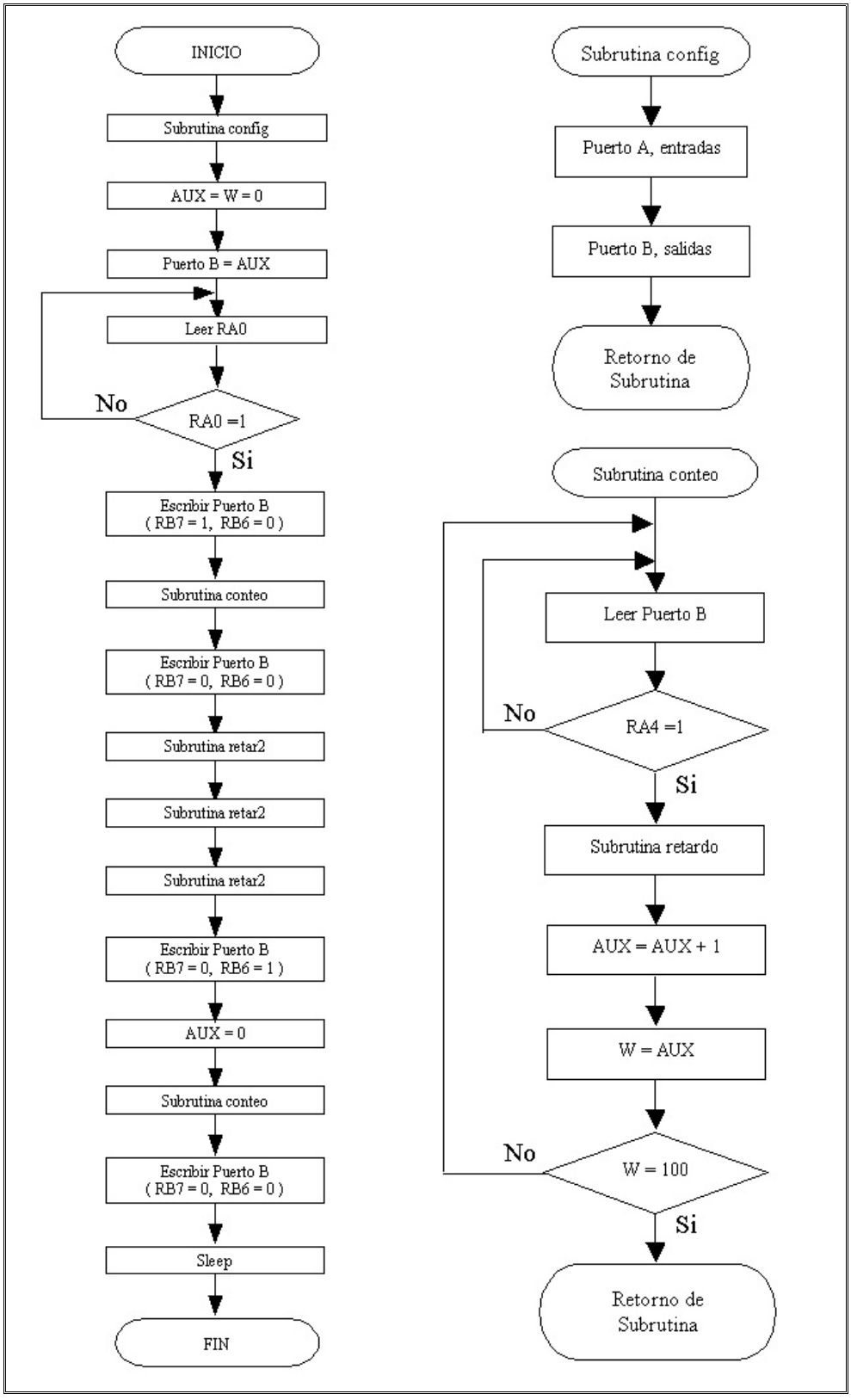


Figura 5.10.3.1. Diagrama de flujo de la práctica 10.

5.10.4 Material y equipo a utilizar

- Computadora personal (características mínimas necesarias para ejecutar las aplicaciones MPLAB y PonyProg2000)
- Software MPLAB
- Software PonyProg
- Circuito Grabador de PIC's PonyProg
- Fuente de alimentación de 5V.
- 1 PIC16F84A
- 1 Cristal Oscilador a 4 MHz
- 2 Capacitores de 22 pF
- 4 Resistencias de 10 KΩ
- 1 Resistencia de 100Ω
- 2 Interruptores miniatura
- 1 Motor de CD.

5.10.5 Procedimiento

Como primer paso a seguir se debe copiar el siguiente programa a un editor de texto sin formato como el "Block de Notas" de Windows, o el editor del MPLAB.

```
;========================== Práctica_10 ==============================

;Mediante la implementación del PIC16F84A se diseñará un circuito que
;controle el giro y número de vueltas de un motor de CD.

;El motor de CD debe girar 100 vueltas hacia la derecha, esperar un
;instante y girar 100 vueltas hacia de izquierda y quedar en modo de
;sleep.

;Al presionar un pulsador conectado a RA0 el motor empezará a funcionar

;Una perforación en un disco conectado al rotor del motor de CD,
;servirá como medio para interrumpir y dejar pasar la luz de un
;optocoplador el cual mandara un pulso positivo al pin RA4 y de
;esta forma se podra realizar el conteo.

;============================ INICIO ================================

        #define __16f84
        LIST P = 16F84A     ;Comando que indica el pic usado

;============================ Etiquetas ==============================

        PC      EQU   0X02  ;Dirección del reg. PC en hex
        ESTADO  EQU   0x03  ;Dirección del reg. ESTADO
        PORT_A  EQU   0x05  ;Dirección del puerto A
        PORT_B  EQU   0x06  ;Dirección del puerto B
        w       EQU   0x00  ;w = 0
        REG_1   EQU   0x0C  ;Dirección del registro REG_1
        REG_2   EQU   0x0D  ;Dirección del registro REG_2
        REG_3   EQU   0x0E  ;Dirección del registro REG_3
        VALOR1  EQU   0x05  ;Valor que se asigna a VALOR1
        VALOR2  EQU   0x05  ;Valor que se asigna a VALOR2
        VALOR3  EQU   0x05  ;Valor que se asigna a VALOR3
        AUX     EQU   0x0F  ;Dirección del registro AUX
        DIEZ    EQU   d'100'

;============================ Programa ===============================

        ORG     0           ;Vector de reset
        goto    inicio      ;Salto a inicio
        ORG     5

inicio  call    config      ;Llamada a la rutina config
off     movlw   0x00        ;w = 0x00
        movwf   AUX         ;AUX = w, AUX = 0x00
        movwf   PORT_B      ;PORT_B = 0x00
pulsa   btfsc   PORT_A,0    ;¿PORT_A<0> = 0?
        goto    arranca     ;Si, salto a arranca
        goto    pulsa       ;No, salto a pulsa
```

```
arranca  movlw    b'10000000' ;Carga w con b'10000000'
         movwf    PORT_B      ;PORT_B = w

         call     conteo      ;Llamada a la rutina conteo

         movlw    0x00        ;w = 0x00
         movwf    PORT_B      ;PORT_B = w, PORT_B = 0x00
         call     retar2      ;Llamada a la rutina retar2
         call     retar2      ;Llamada a la rutina retar2
         call     retar2      ;Llamada a la rutina retar2
         movlw    b'01000000' ;w = b'01000000'
         movwf    PORT_B      ;PORT_B = w, PORT_B = b'01000000'
         clrf     AUX         ;Borra el contenido de AUX
         call     conteo      ;Llamada a la rutina conteo

         movlw    0x00        ;w = 0x00
         movwf    PORT_B      ;PORT_B = w, PORT_B = 0x00
         sleep                ;El PIC se duerme

;============================ Subrutinas ==============================

config   bsf      ESTADO,5    ;Se pone en 1 el bit 5 del reg. ESTADO
         movlw    b'00011111' ;Carga w con el dato b'00011111'
         movwf    PORT_A      ;Configura el puerto A como entradas
         movlw    b'00000000' ;Carga w con el dato b'00000000'
         movwf    PORT_B      ;Configura el puerto B como salidas
         bcf      ESTADO,5    ;Se pone en 0 el bit 5 del reg. ESTADO
         return               ;Retorno desde subrutina

retardo  movlw    VALOR1      ;Carga w con el número 30 (VALOR1)
         movwf    REG_1       ;Mueve w al registro REG_1
tres     movlw    VALOR2      ;Carga w con el número 40 (VALOR2)
         movwf    REG_2       ;Mueve w al registro REG_2
dos      movlw    VALOR3      ;Carga w con el número 50 (VALOR3)
         movwf    REG_3       ;Mueve w al registro REG_3
uno      decfsz   REG_3       ;Decrementa el valor de REG_3 en 1
         goto     uno         ;Salto a uno
         decfsz   REG_2       ;Decrementa el valor de REG_2 en 1
         goto     dos         ;Salto a dos
         decfsz   REG_1       ;Decrementa el valor de REG_1 en 1
         goto     tres        ;Salto a tres
         return               ;Retorno desde subrutina

retar2   call     retardo     ;Llamada a la rutina retardo
         call     retardo     ;Llamada a la rutina retardo
         call     retardo     ;Llamada a la rutina retardo
         call     retardo     ;Llamada a la rutina retardo
         call     retardo     ;Llamada a la rutina retardo
         call     retardo     ;Llamada a la rutina retardo
         return               ;Retorno desde subrutina
```

```
conteo   btfss    PORT_A,4    ;¿El boton esta oprimido?
         goto     conteo      ;No (salto a boton)
         call     retardo     ;Si (Llamada a la rutina retardo)
boton2   btfss    PORT_A,4    ;¿El boton sigue oprimido?
         goto     sigue       ;No (salto a sigue)
         goto     boton2      ;Si (salto a boton2)
sigue    incf     AUX,1       ;Incrementa AUX (AUX = AUX + 1)
         movf     AUX,w       ;Mueve AUX a w
         xorlw    DIEZ        ;Funcion Xor (¿es igual a 100?)
         btfsc    ESTADO,2    ;¿AUX = DIEZ? (ESTADO<2> = 0)
         goto     para        ;Salto a borra (ESTADO<2> <> 0)
         goto     conteo      ;Salto a bucle (ESTADO<2> =  0)
para     return

;============================== Fin ================================

         end
```

Notas:

```
;La frecuencia externa debe ser de 4MHz (Cristal de 4MHz).
```

Una vez que se ha editado todo el programa, es necesario guardarlo en un archivo que puede ser llamado Practica_10.asm

El siguiente paso consiste en compilar el programa que se ha creado mediante el MPLAB para generar el archivo Practica_10.hex

Mediante el PonyProg2000 debe grabarse el contenido del archivo Practica_10.hex en el microcontrolador. Una vez hecho esto, se debe armar el circuito de aplicación para esta práctica, el cual se muestra en la figura 5.10.5.1.

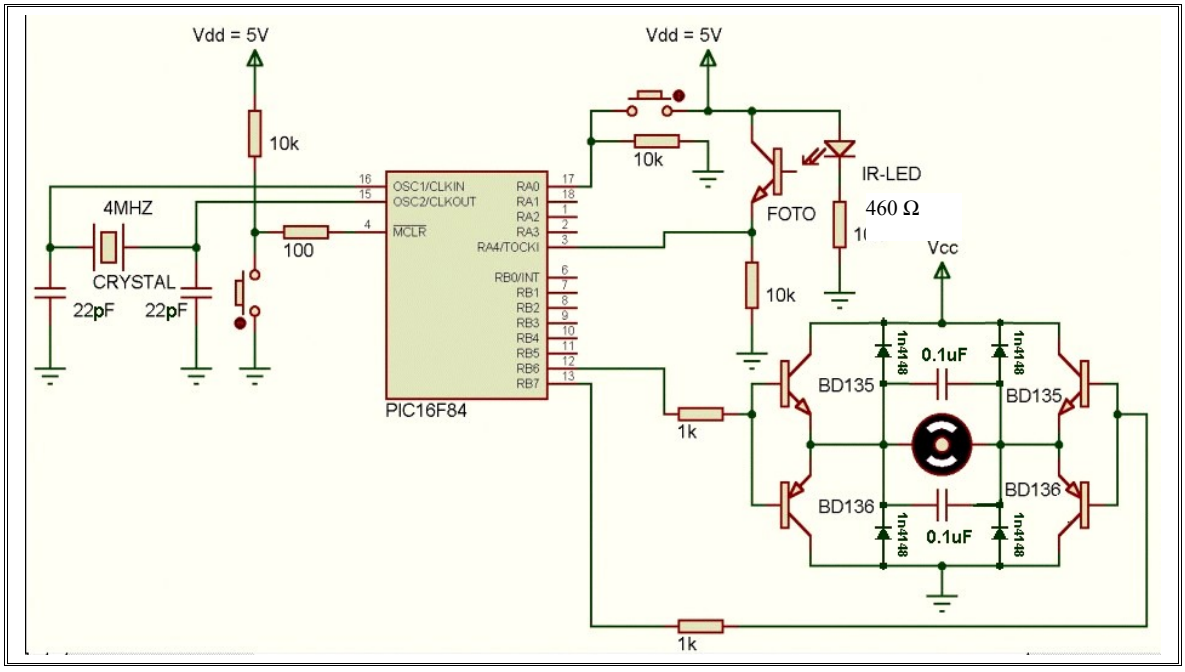


Figura 5.10.5.1. Circuito de aplicación para la práctica 10.

Cuando el circuito de aplicación esté terminado y el PIC ya tenga grabado el programa correspondiente, se inserta el dispositivo en dicho circuito y después se enciende la fuente de alimentación para probar su correcto funcionamiento.

5.10.6 Indicaciones

Para esta práctica, la palabra de configuración del PIC debe ser establecida de la siguiente forma. Código de protección deshabilitado, WDTE deshabilitado, PWRTE habilitado, oscilador XT. Los pines que no sean empleados en esta práctica, deben conectarse a Vdd por medio de una resistencia de 10KΩ.

5.11 Práctica No. 11 Control de giro de un motor de CD mediante la interrupción externa INT

5.11.1 Objetivo

Implementar un microcontrolador para el control de giro y conteo de vueltas de un motor de corriente directa. También se utilizará la interrupción externa INT como una forma de adquisición de datos, además se empleará el modo de bajo consumo de energía SLEEP, así como la forma de salir de este estado a través de una interrupción (INT).

5.11.2 Descripción

Esta práctica es muy parecida a la práctica anterior, pues también utliza el motor de Cd, el disco y el optoacoplador, pero la diferencia reside en la utilización de la interrupción externa INT, que se encuentra en el pin 6 (RB0). Cuando se produce a un flanco ascendente en el pin RB0/INT, debido al paso de la luz por el orificio del disco se origina una interrupción, desviando el flujo del programa hacia el vector de interrupción, el cual atenderá a la interrupción e incrementará un registro auxiliar. El motor girará hacia la derecha, luego se detendrá un instante y empezará a girar hacia la izquierda; una vez terminado el proceso, entrará en modo SLEEP. Para salir de dicho estado, bastará con girar el disco para dejar pasar luz hacia el optoacoplador para activar la bandera de interrupción externa, la cual despertará al PIC y provocará que se continúe con el resto del programa, despertando al PIC del modo de ahorro de energía.

5.11.3 Diagrama de flujo

El diagrama de flujo de la práctica número 11 se muestra en la figura 5.11.3.1.

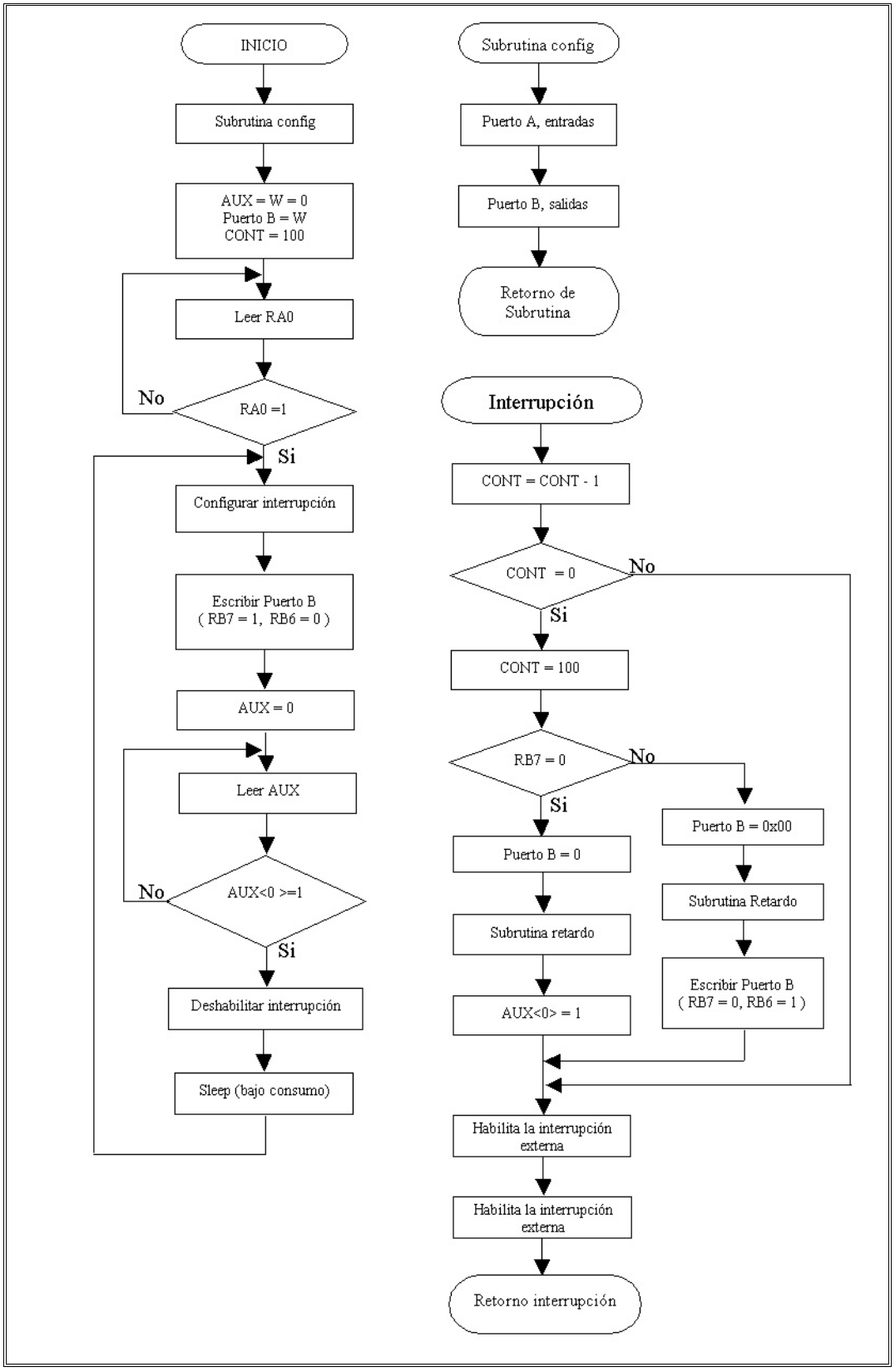


Figura 5.11.3.1. Diagrama de flujo de la práctica 11.

5.11.4 Material y equipo a utilizar

- Computadora personal (características mínimas necesarias para ejecutar las aplicaciones MPLAB y PonyProg2000)
- Software MPLAB
- Software PonyProg
- Circuito Grabador de PIC's PonyProg
- Fuente de alimentación de 5V.
- 1 PIC16F84A
- 1 Cristal Oscilador a 4 MHz
- 2 Capacitores de 22 pF
- 4 Resistencias de 10 KΩ
- 1 Resistencia de 100Ω
- 2 Interruptores miniatura
- 1 Motor de CD.

5.11.5 Procedimiento

Como primer paso a seguir se debe copiar el siguiente programa a un editor de texto sin formato como el "Block de Notas" de Windows, o el editor del MPLAB.

```
;========================== Práctica_11 ==============================

;Esta práctica es similar a la práctica 10 pero en esta se utiliza la
;interrupción externa INT, la cual tambien servirá para sacar al
;PIC del modo de SLEEP
;============================ INICIO ================================

        LIST P=16F84A           ;Comando que indica el pic usado

;========================== Etiquetas ============================

        OPT     EQU   0x01      ;Dirección del reg. OPTION
        STATUS  EQU   0x03      ;Dirección del reg. ESTADO
        PORT_A  EQU   0x05      ;Dirección del puerto A
        PORT_B  EQU   0x06      ;Dirección del puerto B
        INTCON  EQU   0x0B      ;Dirección del reg. INTCON
        AUX     EQU   0x0C      ;Dirección del reg. AUX
        AUX1    EQU   0x0D      ;Dirección del reg. AUX1
        AUX2    EQU   0x0E      ;Dirección del reg. AUX2
        AUX3    EQU   0x0F      ;Dirección del reg. AUX3
        CONT    EQU   0x10      ;Dirección del reg. CONT
        VALOR_1 EQU   0x40      ;Valor asignado a VALOR_1
        VALOR_2 EQU   0x40      ;Valor asignado a VALOR_2
        VALOR_3 EQU   0x50      ;Valor asignado a VALOR_3

;========================== Programa ============================

        ORG     0               ;Vector de reset
        goto    inicio          ;Salto a inicio
        ORG     4               ;Vector de interrupciones
        goto    inter           ;Salto a inter
        ORG     5

inicio  clrf    PORT_A          ;Borra el reg. PORT_A
        clrf    PORT_B          ;Borra el reg. PORT_B
        clrf    CONT            ;Borra el reg. CONT
        clrf    AUX             ;Borra el reg. AUX
        call    config          ;Llamada a la rutina config
        movlw   d'100'          ;Carga w con 100
        movwf   CONT            ;Carga CONT con w, CONT = 100

boton   btfss   PORT_A,0        ;¿PORT_A<0> = 1?
        goto    boton           ;No, salto a boton
ciclo   bsf     INTCON,7        ;Si, pone en 1 el bit 7 de INTCON
        movlw   b'10000000'     ;Carga w con b'10000000'
        movwf   PORT_B          ;Carga PORT_B con w
        clrf    AUX             ;Borra el reg. AUX
bucle_1 btfss   AUX,0           ;¿AUX<0> = 1?
        goto    bucle_1         ;No, salto a bucle_1

        bcf     INTCON,7        ;INTCON<7>=0 desabilita interrupciones
        sleep                   ;Si, duerme el PIC
        nop                     ;No opera
        goto    ciclo           ;Salto a ciclo
```

```
;============================ Subrutinas ==============================

config   bsf      STATUS,5          ;STATUS<5>=1, se pasa al banco 1
         movlw    b'11111111'       ;w = b'11111111'
         movwf    PORT_A            ;PORT_A = w, se configuran con
entradas
         movlw    b'00000001'       ;w = b'00000001'
         movwf    PORT_B            ;PORT_B = w, se configura en puerto B
         movlw    b'00010000'       ;w = b'00010000'
         movwf    INTCON            ;INTCON = w, se configura INTCON
         bsf      OPT,6             ;OPTION<6>=1, flanco ascendente
         bcf      STATUS,5          ;STATUS<5>=0, se pasa al banco 0
         return                     ;Retorno de subrutina

retardo  movlw    VALOR_1           ;Carga w con el valor de VALOR_1
         movwf    AUX1              ;AUX1 = w, AUX = 0x30
tres     movlw    VALOR_2           ;Carga w con rl valor de VALOR_2
         movwf    AUX2              ;AUX2 = w, AUX = 0x40
dos      movlw    VALOR_3
         movwf    AUX3
uno      decfsz   AUX3              ;Decrementa en 1 el valor de AUX3
         goto     uno               ;Salto a uno
         decfsz   AUX2              ;Decrementa en 1 el valor de AUX2
         goto     dos               ;Salto a dos
         decfsz   AUX1              ;Decrementa en 1 el valor de AUX1
         goto     tres              ;Salto a tres
         return                     ;Retorno de subrutina

;========================== Interrupciones ============================

inter    decfsz   CONT              ;Decrementa en 1 el valor de CONT
         goto     sigue             ;Salto a sigue
cont_0   movlw    d'100'            ;Carga w con 100
         movwf    CONT              ;Carga CONT con w, CONT = 100
         btfsc    PORT_B,7          ;¿PORT_B<7> = 0?
         goto     cambio            ;No, Salto a sigue
         movlw    b'00000000'       ;Carga w con b'00000000'
         movwf    PORT_B            ;Carga PORT_B con b'00000000'
         call     retardo           ;Llamada a retardo
         bsf      AUX,0             ;Pone en 1 en bit 0 de AUX, AUX<0> = 1
         goto     sigue             ;Salto a sigue
cambio   movlw    0x00              ;Carga w con 0x00
         movwf    PORT_B            ;Carga PORT_B con 0x00
         call     retardo           ;Llamada a la rutina retardo
         movlw    b'01000000'       ;Carga w con b'01000000'
         movwf    PORT_B            ;Carga PORT_B con b'01000000'
sigue    movlw    b'10010000'       ;Carga w con b'10010000'
         movwf    INTCON            ;Habilita la interrupcion externa
         retfie                     ;Retorno de interrupcion

;============================== Fin ===================================
         end
```

Nota: La frecuencia del cristal debe ser de 4 MHz

Una vez que se ha editado todo el programa, es necesario guardarlo en un archivo que puede ser llamado Practica_11.asm

El siguiente paso consiste en compilar el programa que se ha creado mediante el MPLAB para generar el archivo Practica_11.hex

Mediante el PonyProg2000 debe grabarse el contenido del archivo Practica_11.hex en el microcontrolador. Una vez hecho esto, se debe armar el circuito de aplicación para esta práctica, el cual se muestra en la figura 5.11.5.1.

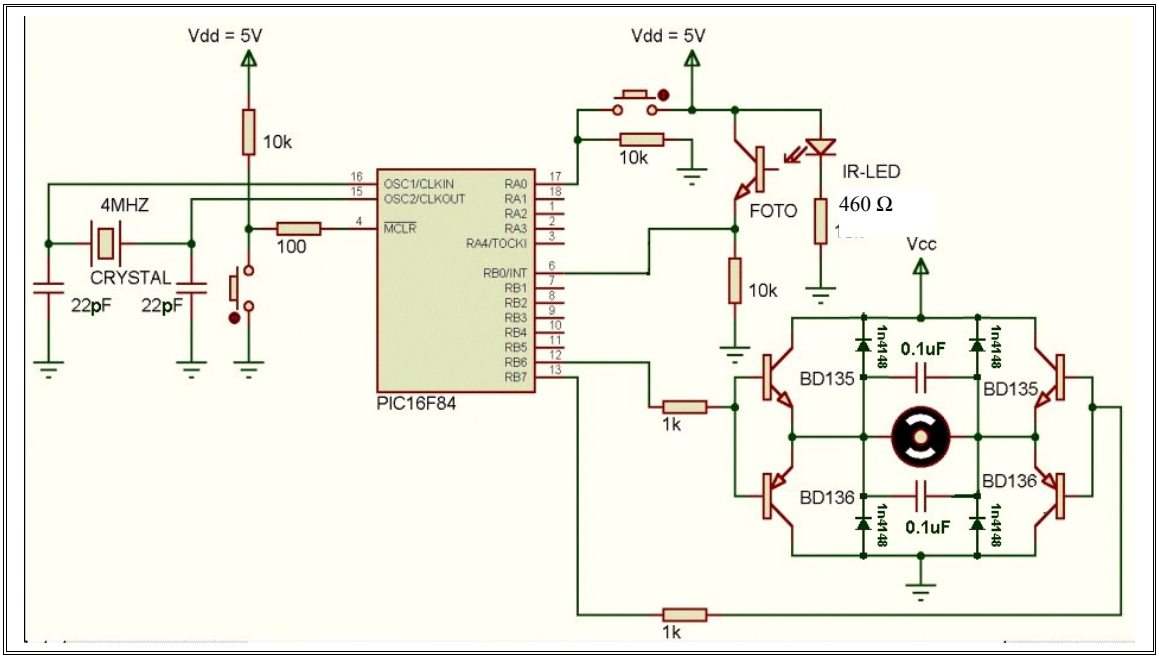


Figura 5.11.5.1. Circuito de aplicación para la práctica 11.

5.11.6 Indicaciones

Para esta práctica, la palabra de configuración del PIC debe ser establecida de la siguiente forma. Código de protección deshabilitado, WDTE deshabilitado, PWRTE habilitado, oscilador XT. Los pines que no sean empleados en esta práctica, deben conectarse a Vdd por medio de una resistencia de 10KΩ.

RECOMENDACIONES

Es muy recomendable que el usuario de los microcontroladores conozca muy bien las especificaciones de operación del dispositivo, las cuales están disponibles en el sitio web del fabricante; esto con la finalidad de darle un buen uso, de no sobrecargarlo o de operarlo en condiciones inadecuadas.

El PIC16F84A de Microchip está construido mediante tecnología CMOS, por lo que se recomienda manipularlo de forma cuidadosa y siempre con una pulsera antiestática u otra herramienta que ayude a la conservación de este dispositivo.

Cuando se pretende cargar un programa en un dispositivo microcontrolador, es necesario un circuito grabador, el cual se debe elegir de acuerdo a las necesidades del usuario. Actualmente en el mercado existen herramientas de desarrollo con un valor que oscila desde los $ 2,000.00 hasta los $15,000.00. Los programadores de uso libre son una excelente opción para todos aquellos usuarios que no requieren de un gran sistema para realizar prácticas y prototipos didácticos, por lo que se recomienda usar programas como el IC-Prog y el PonyProg2000.

Por lo general, todos los circuitos grabadores operan bajo ciertos niveles de señal procedentes de algún puerto de la computadora, por lo que de preferencia se debe trabajar con una computadora de escritorio, debido a que comúnmente se tiene el problema de que las computadoras portátiles no ofrecen el mismo voltaje a la salida de sus puertos; aunque al programar los microcontroladores realmente se requiere de un nivel muy bajo de corriente.

El simulador del programa MPLAB es una herramienta que debe ser usada para comprobar el correcto funcionamiento del programa diseñado, todo esto, con la finalidad de que no se grabe el pic con un programa inservible.

El tema de los microcontroladores es muy extenso, y aún cuando en esta guía se ha pretendido analizar y explicar todo lo relacionado con el dispositivo PIC16F84A, es necesario que el usuario investigue en otras fuentes y se actualice constantemente sobre la evolución de esta línea de productos PIC Micro de la empresa Microchip.

CONCLUSIONES

Durante el desarrollo de esta obra didáctica se presentó una situación poco peculiar, ya que en contraste con otros temas, acerca de los microcontroladores PIC16F84A, existe una gran cantidad de fuentes bibliográficas, por lo que fue necesario analizar minuciosamente cada fuente de información y posteriormente seleccionar solo el mejor material.

En lo que se refiere a la explicación de las características, arquitectura y operación de dicho microcontrolador, fue necesario hacer una trascripción de los términos que se encuentran en las diversas fuentes bibliográficas, ya que la mayoría de las obras relacionadas con este tema fueron desarrolladas en otros países con modismos y terminología diferente.

Los recursos del PIC16F84A son muy variados, por lo que requirieron de un fuerte análisis para poder ser explicados.

El manejo de los programas MPLAB, NOPPP y PonyProg2000 no se especifica correctamente en ninguna fuente de información, por lo que el desarrollo de estas secciones fue basado en la experimentación, la práctica y los resultados al operar dichas herramientas de desarrollo.

La parte más importante de esta guía son las prácticas propuestas, la cuales se desarrollaron en circuitos reales y se garantiza su correcto funcionamiento. Durante el desarrollo de este trabajo se adquirió una gran habilidad para implementar soluciones a problemas comunes y específicos mediante los microprocesadores PIC.

BIBLIOGRAFÍA

- Angulo, J.Ma. e Ignacio Angulo, *Microcontroladores PIC, Diseño práctico de aplicaciones,* Editorial McGraw Hill, 1997.

- Angulo, J. Ma., E. M. Cuenca y J. Angulo, *Microcontroladores PIC, la solución en un chip*, Editorial Paraninfo, 1997.

- Angulo, J. Ma., E. M. Cuenca y J. Angulo, *Aplicaciones de los microcontroladores PIC de Microchip*, Editorial McGraw Hill, 1998.

- Microchip Technology Inc, *Microchip PIC16F84 Microcontroller Data Book*, 1997.

- Joyanes, Luis A., *Problemas de metodología de la programación*, Editorial McGraw Hill, 1996.

- Duque, Edison C., *Curso Avanzado de Microcontroladores PIC,* CEKIT, 1998.

- Goñi, Julio, "*Robots. Construye tu microbot"* F&G Editores, 2000

SITIOS WEB

- www.redeya.com
- www.lancos.com
- www.microchip.com
- www.mindspring.com/~covington/noppp

ANEXOS

Hojas de Datos del PIC16F84A