879316

UNIVERSIDAD LASALLISTA BENAVENTE

ESCUELA DE INGENIERÍA EN COMPUTACIÓN

Con estudios incorporados a la
Universidad Nacional Autónoma de México
CLAVE: 8793-16

"ROBOT SEGUIDOR DE LINEA CON EL PIC16F84"

TESIS

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

PRESENTA:

MIGUEL ANTONIO MUÑOZ DOMÍNGUEZ

ASESOR: **ING. CARLOS ALFONSO HERNANDEZ VILLANUEVA**

Celaya, Gto. **Junio de 2004**

Índice

Introducción

La aplicación de los Microcontroladores se ha manifestado en un sin número de objetos que usamos día a día, aún cuando no se interactúa propiamente con ellos, se perciben con su eficiencia y su versatilidad de usos, lo que ha desencadenado cada vez mas su implementación en nuevas tareas del hombre.

Lo anterior se ve reflejado en que los microcontroladores son la representación palpable de la evolución tecnológica en el área de la arquitectura de computadoras, lo que a motivado al desarrollo de su diseño y fabricación.

El presente trabajo tiene como objetivo dar a conocer las generalidades de los Microcontroladores PIC, su funcionamiento, y la programación de los mismos, refiriéndose en particular el PIC16F84A del cual se propondrán ejemplos y un caso práctico de su uso en un robot autómata seguidor de línea, donde se expondrá el porque de su uso y selección, las razones con respecto a que se consigue una reducción de tamaño y una disminución de los costos del mecanismo.

En el capítulo 1: microcontroladores. El objetivo es conocer que son los microcontroladores, aprender en que nos sirven en la vida cotidiana, saber cuales son los mas destacados dentro de la variedad de microcontroladores, aprender a seleccionar el microcontrolador mas adecuado de acuerdo a los factores (costo y aplicación), y en caso de que se seleccionen los PIC'S para trabajar saber seleccionar el mas adecuado dentro de las diferentes gamas que existe. Saber las diferencias entre una microcontrolador con una CPU y un microprocesador.

En el capítulo 2: El PIC 16F84. El objetivo es dar a conocer al lector que es un microcontrolador PIC, más específicamente el PIC16F84A, como es que surge y como es que han evolucionado con el tiempo; la gama de PIC'S que existen y como están constituidos internamente (arquitectura, memoria, puertos, etc.).

En el capítulo3: PROGRAMACION DEL PIC16F84A. El objetivo es dar a conocer como se crea un código fuente para programar los PIC'S y como se hace para pasar en código fuente al PIC, además proveer al lector las instrucciones con las que cuenta el PIC16F84A y algunos ejemplos para su mejor comprensión, y así el lector se interese mas en el tema.

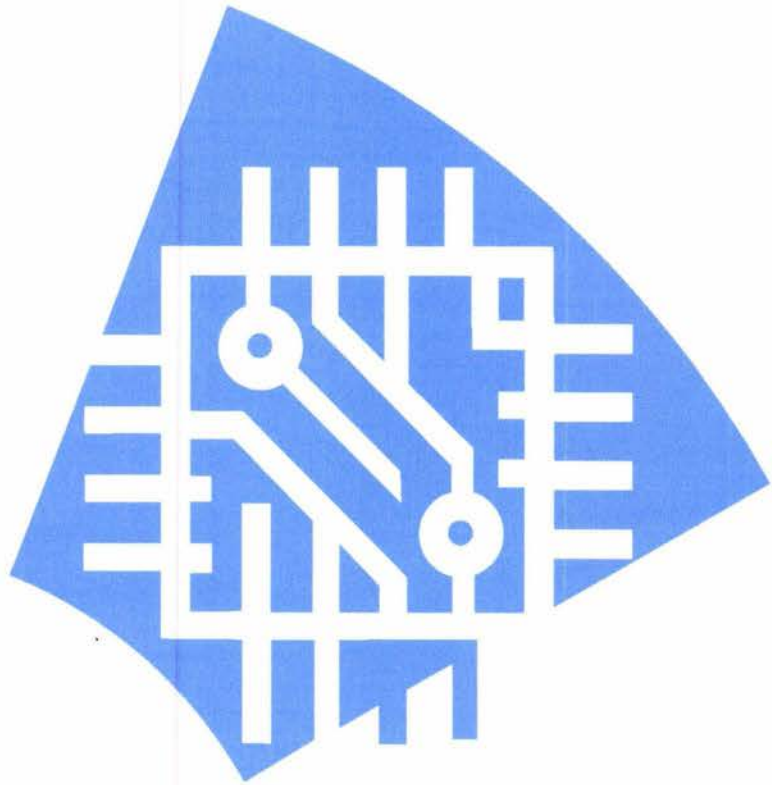
En el capítulo 4: Robot autómata seguidor de línea. El objetivo es llevar acabo los conocimientos adquiridos en el transcurso del trabajo y realizar una aplicación, la cual es un robot seguidor de línea con el microcontrolador PIC16F84A y así brindarle al lector un enfoque mayor de lo que se puede lograr al trabajar con el microcontrolador PIC16F84A.

Mucha gente cree que para realizar un proyecto, el cual involucre un control automatizado, como puede ser un semáforo, una máquina expendedora de refrescos, un microondas, etc. se tendría que invertir mucho dinero en la adquisición de una computadora para poder lograr esto. La intención de éste trabaja es dar a conocer una alternativa diferente a lo comúnmente utilizado, la cual es muy poderosa en comparación a su tamaño y se pueden logran un sinfín de cosas, dicha alternativa son los PIC'S.

Estos microcontroladores PIC son mas utilizados para interactuar con las máquinas, esto quiere decir, que una ves programado nuestro PIC nada más se tiene que crear la interfase con la que va a trabajar el PIC y una vez conectado éste, él se encargará de todo.

Capítulo 1

Los Microcontroladores



Contenido:

- **CPU**
- **Microcontrolador**
- **Diferencia entre microprocesador y microcontrolador**
- **¿Por qué pequeña computadora?**
- **Diversidad de Microcontroladores**
- **¿Qué Microcontrolador emplear?**
- **¿Qué Microcontrolador usaremos?**

1.1 CPU

Central Processing Unit (Unidad de procesamiento Central). Este término se refiere específicamente al circuito integrado (contenido dentro del gabinete de la computadora), que conforman las verdaderas computadoras. Sin embargo, a veces el término es usado (aunque incorrectamente) para incluir todo lo que está dentro del gabinete de una computadora, incluyendo el disco duro, la disquetera, el CD ROM, la fuente y la motherboard.[1]

1.2 MICROCONTROLADOR

Es un circuito integrado que contiene muchas de las mismas cualidades que una computadora de escritorio, tales como la CPU, la memoria, etc., pero no incluye ningún dispositivo de "comunicación con humanos", como monitor, teclados o mouse. Los microcontroladores son diseñados para aplicación de control de máquinas, más que para interactuar con humanos. Es decir, se usan en el control de los hornos de microondas, en juguetes en lo cuales intervienen procesos de toma de decisión, en los carros, etc.

Muchos de nosotros sabemos qué apariencia tiene una computadora. Usualmente tiene teclado, monitor, CPU (Unidad de Procesamiento Central), impresora y mouse. Este tipo de computadoras, como la Mac[2] o PC,[3] son diseñadas principalmente para comunicarse con humanos.

Manejo de base de datos, análisis financieros o incluso procesadores de textos, se encuentran todos dentro de una computadora que contiene: CPU, la memoria, el disco rígido, etc. El verdadero "cómputo", sin embargo, tiene lugar dentro de la CPU.

---

[1] Tarjeta madre.
[2] Computadora de la marca apple.
[3] Computadora personal.

Si se piensa sobre esto, el único propósito del monitor, teclado, mouse e incluso la impresora, es conectar a la CPU con el mundo exterior, para que los humanos puedan interactuar directamente y poder realizar diferentes procesos en la CPU por medio de la computadora con todos sus periféricos. "Pero, ¿usted sabía que hay computadoras alrededor de nosotros, corriendo programas y haciendo cálculos silenciosamente sin interactuar con ningún humano?"[4] Estas computadoras están en los autos, las cuales sirven para controlar el tacómetro digital, el nivel de gasolina, los kilómetros que va a rendir la gasolina, hasta la brújula digital con las que cuentan algunos carros entre otras cosas; en el trasbordador espacial, en un juguete, e incluso puede haber uno en la secadora de cabello.

Llamamos a estos dispositivos "microcontroladores". Micro porque son pequeños, y controladores, porque controlan máquinas o incluso otros controladores.

Los microcontroladores, por definición entonces, son diseñados para ser conectados más a máquinas que a personas. Son muy útiles porque se puede construir una máquina o artefacto, escribir programas para controlarlo, y luego dejarlo trabajar para usted automáticamente.

Hay un número infinito de aplicaciones para los microcontroladores. ¡Su imaginación es el único factor limitante!

Cientos (sino miles) de variaciones diferentes de microcontroladores están disponibles. Algunos son programados una vez y producidos para aplicaciones específicas, tales como controlar su horno microondas. Otros son "reprogramables", que quiere decir que pueden ser usados una y varias veces para diferentes aplicaciones. Los microcontroladores son increíblemente versátiles, el mismo dispositivo puede controlar un aeromodelo, una tostadora, o incluso el ABS de su auto (sistema antibloqueo).

[4] What's a microcontroller?

1.3 DIFERENCIA ENTRE MICROPROCESADOR Y MICROCONTROLADOR

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

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

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

1.4 ¿POR QUÉ PEQUEÑA COMPUTADORA?

A este chip se le conoce también como microcomputadora (véase en la Fig.1.1)[5], porque tiene todos los componentes y recursos necesarios para serlo, es decir:

Una CPU (Central Processor Unit o Unidad de Procesamiento Central) quien interpreta las instrucciones de programa.

[5] www.frino.com.ar/micros.htm

Una memoria PROM (Programmable Read Only Memory o Memoria Programable Solamente para Lectura) memoriza permanentemente las instrucciones de programa. Otros modelos de microcontroladores tienen memoria de programa de tipo EEPROM y otros de tipo FLASH

Una memoria RAM (Random Access Memory o Memoria de Acceso Aleatorio) utilizada para memorizar las variables utilizadas para el programa.

Una serie de LINEAS de E/S para controlar dispositivos externos o recibir pulsos de sensores, switches, etc.

Una serie de dispositivos auxiliares para su funcionamiento, como puede ser: generador de clock, bus, contador, etc.

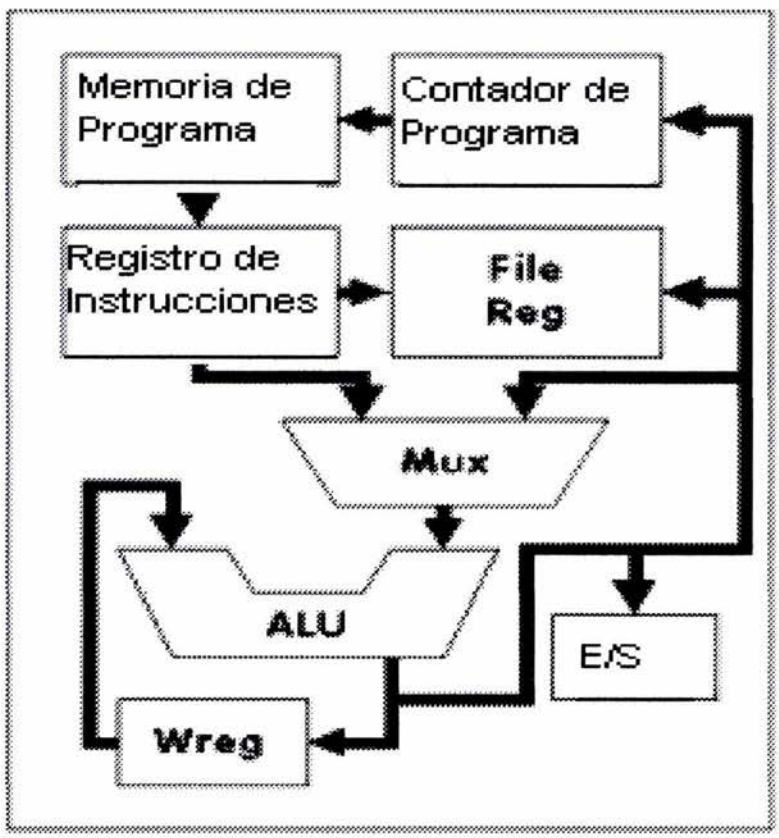


Fig. 1.1 Diagrama simplificado de una microcomputadora

El tener todos estos recursos en un solo chip hace ventajoso el uso de este dispositivo ya que en poco tiempo y con pocos componentes podemos hacer lo que sería trabajoso hacer con circuitos tradicionales.

1.5 DIVERSIDAD DE MICROCONTROLADORES

Existen cientos de fabricantes de microcontroladores, entre los que más destacan son:

- **8048 (Intel).** Es el padre de los microcontroladores actuales, el primero de todos. Su precio, disponibilidad y herramientas de desarrollo hacen que todavía sea muy popular.
- **8051 (Intel y otros).** Es sin duda el microcontrolador más popular. Fácil de programar, pero potente. Está bién documentado y posee cientos de variantes e incontables herramientas de desarrollo.
- **80186, 80188 y 80386 EX (Intel).** Versiones en microcontrolador de los populares microprocesadores 8086 y 8088. Su principal ventaja es que permiten aprovechar las herramientas de desarrollo para PC.
- **68HC11 (Motorola y Toshiba).** Es un microcontrolador de 8 bits potente y popular con gran cantidad de variantes.
- **683xx (Motorola).** Surgido a partir de la popular familia 68k, a la que se incorporan algunos periféricos. Son microcontroladores de altísimas prestaciones.
- **PIC (MicroChip).** Familia de microcontroladores que gana popularidad día a día. Fueron los primeros microcontroladores RISC.
- **Basic Stamp(Parallax)**

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

Por mencionar la existencia de diferentes tipos de microcontroladores que hay en el mercado, se describirá un poco el Basic Stamp de la empresa Parallax:

En 1979, Chip Gracey tuvo su primera introducción a la programación y la electrónica: la computadora Apple II.

Chip se interesó instantáneamente en la nueva máquina, escribiendo código BASIC para mostrar gráficos y quitando la tapa para mostrar los componentes electrónicos. Esta experiencia lo llevó rápidamente a desmantelar el código fuente de los videojuegos y electrodomésticos hogareños y a tratar de usar estos artefactos para propósitos distintos de los que originalmente tenían, transformando el hobby en un negocio. A la vez que se recibía en el colegio, Chip comenzaba un pequeño negocio desde su pieza.

Los colegios no ofrecían clases sobre montajes electrónicos ni programación en 1982, y cuando Chip se graduó en 1986 no se veía como el lugar correcto para comenzar un negocio. En lugar de eso, él y un amigo, Lance Walley comenzaron "Parallax" desde su apartamento. Los primeros productos incluyeron digitalizadores de sonido para la Apple II y programadores 8051.

El negocio creció lentamente hasta 1992 cuando Parallax realizó el primer BASIC Stamp. Parallax reconoció que el BASIC Stamp debería ser especial – era la herramienta que ellos necesitaban para hacer sus propios proyectos de hobby.

El hecho que el BASIC Stamp crearía su propia industria era probablemente desconocido por los fundadores de Parallax, pero rápidamente se volvió aparente que la pequeña computadora tenía su propio grupo de entusiastas. Este le permitía a la gente común programar un microcontrolador de una sola vez y les daba poderosos comandos de entrada-salida que lo hicieron fácil de conectar a otros componentes electrónicos.

Hacia el final de 1998 Parallax vendió más de 125,000 módulos de BASIC Stamp y distribuyó una serie completa de periféricos a través de 40 canales de venta mundiales.

¿Qué significa toda esta historia? Este está diseñado para introducir al uso de los microcontroladores usando circuitería simple, integrándola sin un gran esfuerzo.

Esta es la forma en que Chip aprendió microcontroladores y ganó suficiente experiencia en ingeniería para diseñar el BASIC Stamp.

El BASIC Stamp II (#BS2-IC) (véase en la Fig. 1,2)[6] es un pequeño que ejecuta programas en lenguaje PBASIC. El BS2-IC tiene 16 pines E/S (entrada/salida) que pueden ser conectados directamente a dispositivos digitales o de niveles lógicos, tales como botones, LEDs, altavoces, potenciómetros, y registros de desplazamiento. Además, con unos pocos componentes extra, estos pines de I/O pueden ser conectados a dispositivos que no manejen niveles TTL, tales como solenoides, relés, redes RS-232, y otros dispositivos de alta corriente o tensión.

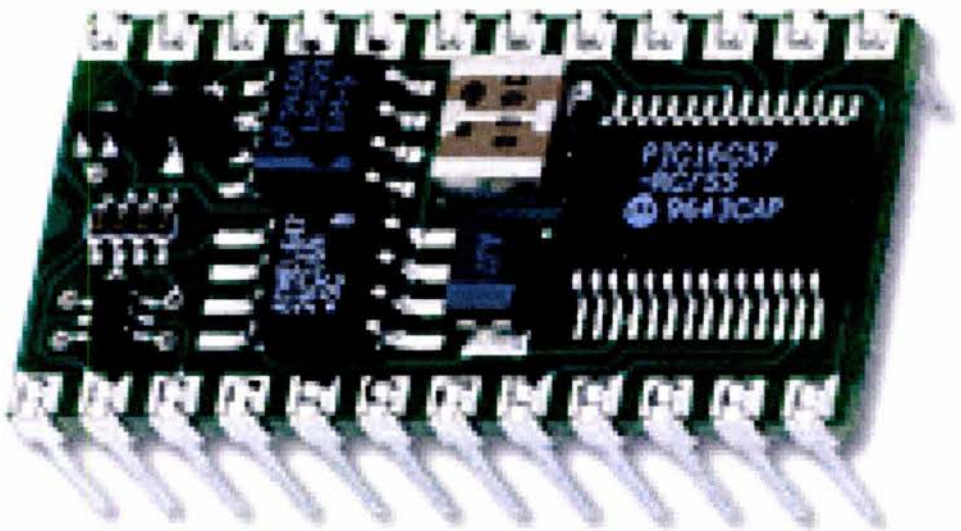


Fig.1.2 BASIC Stamp II (#BS2-IC)

[6] www.parallax.com

El OEM BASIC STAMP II circuito es una magnifica alternativa al BS2-IC (véase en la Fig. 1.3)[7], ya que con las mismas características que éste, permite ser programado directamente desde el PC, sin necesidad de la placa educacional. Puede ser *pinchado* en una placa protoboard desde la que tener acceso a todas sus señales de E/S y alimentación. Es un elemento muy práctico en la etapa de desarrollo de una aplicación y puede ser adquirido montado o en Kit.

Otra ventaja que presenta respecto al BS2-IC, es que su módulo intérprete así como la memoria puede ser fácilmente reemplazable en caso de una avería debida a un "error de usuario".

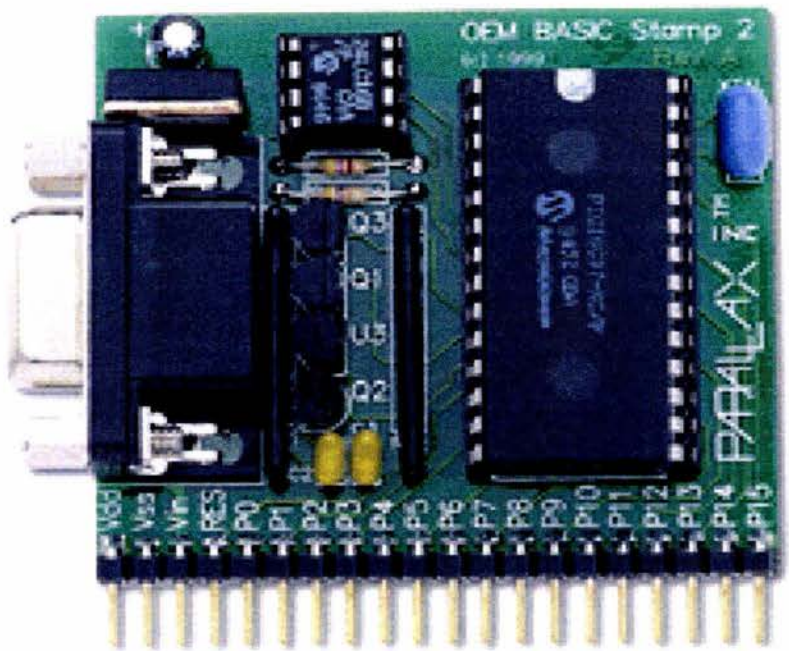


Fig. 1.3 OEM BASIC STAMP II

[7] IDEM

La tarjeta BASIC STAMP II CARRIER BOARD (véase en la Fig. 1.4)[8] dispone de un conector DIP de 24 pines que permite insertar el BS2-IC y a través de un cable serie (CAP9) programarlo desde el PC. Además dispone también de un área para prototipos, botón de reset y conector para pila de 9 Vdc.

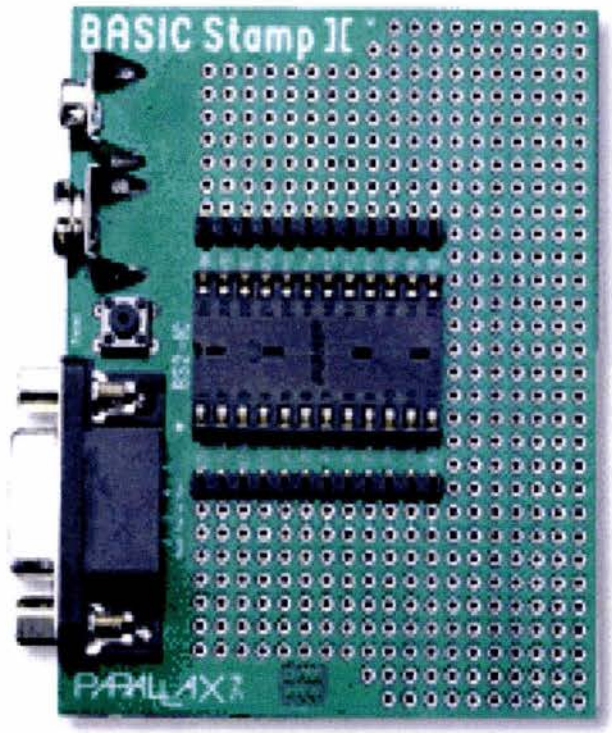


Fig. 1.4 TARJETA BASIC STAMP II CARRIER BOARD

La tarjeta SUPERCARRIER BOARD (véase en la Fig. 1.5)[9] dispone de conectores donde insertar tanto el BS1-IC como el BS2-IC permitiendo el volcado del programa desde el PC. Dispone además de un área para montaje de prototipos, regulador de tensión, conexión al puerto serie del PC (CAP9), botón reset, base de alimentación para fuente de VDC (6-30VDC) o conector para pila 9 Vdc.

[8] IDEM
[9] IDEM

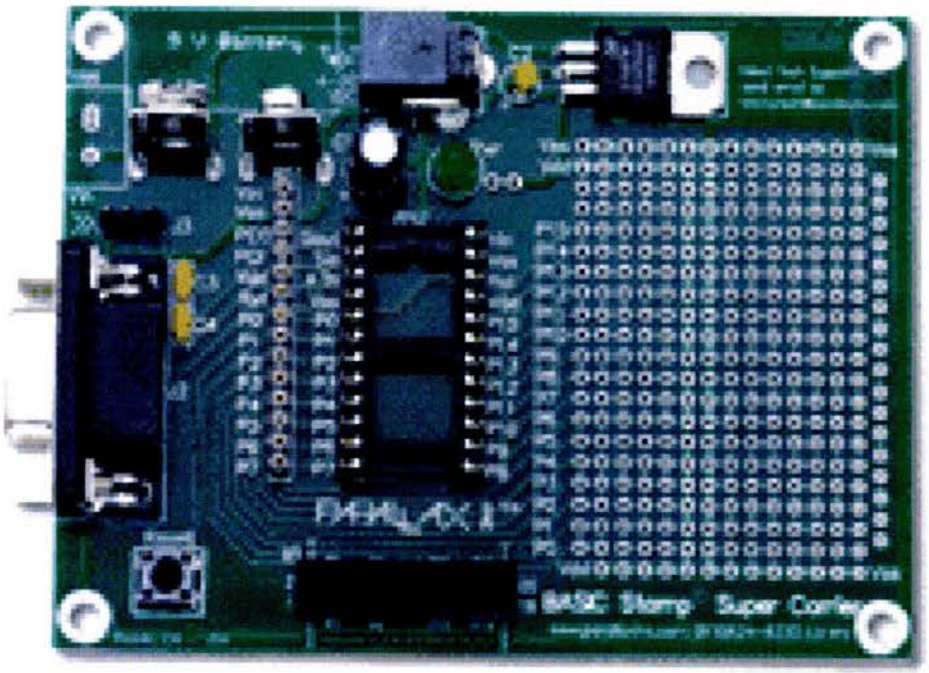


Fig. 1.5 TARJETA SUPERCARRIER BOARD

El diseño físico (véase en la Fig. 1.6)[10] consiste en un regulador de 5-volts, resonador, EEPROM serie, e intérprete PBASIC. Un programa simbolizado (o tokenizado) en PBASIC es almacenado en la EEPROM serie no volátil, desde donde el chip intérprete (que se puede adquirir por separado referencia PBASIC2C/P) grabado en el microcontrolador lee y escribe valores. Este chip intérprete ejecuta una instrucción cada vez, realizando la operación apropiada en los pines de E/S o en la estructura interna del intérprete.

Debido a que el programa PBASIC es almacenado en una EEPROM, puede ser reprogramado una cantidad cercana al millón de veces, sin la necesidad de borrar primero la memoria.

[10] IDEM

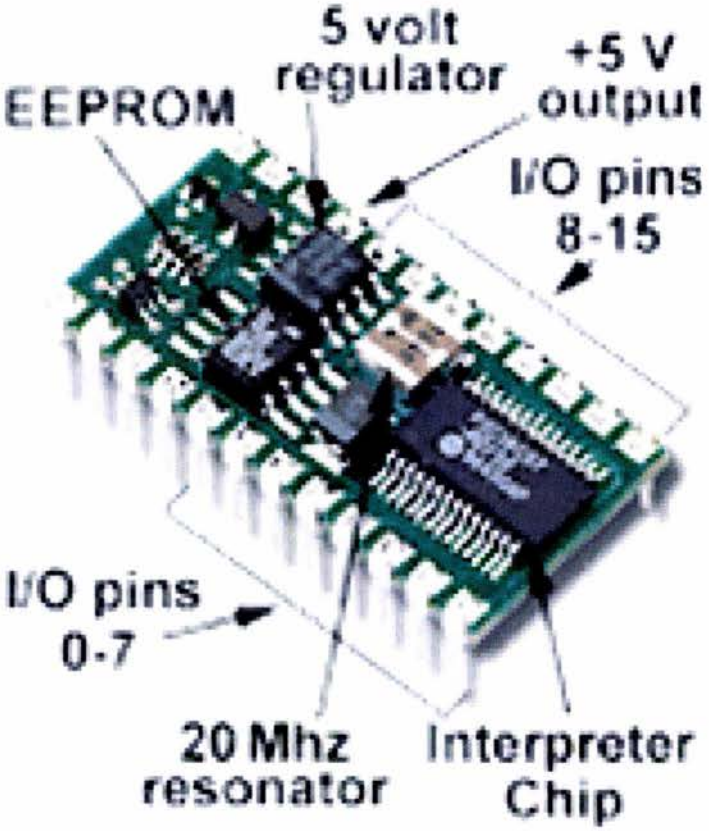


Fig. 1.6 BASIC STAMP II

La programación de la memoria se puede realizar directamente desde una computadora personal colocando el BS2-IC en la tarjeta de educación y volcando los programas desde el software proporcionado por Parallax, Inc.

El Basic Stamp II es capaz de almacenar entre 500 y 600 instrucciones de alto nivel (PBASIC) y ejecuta un promedio de 4000 instrucciones/segundo.

Para programar el BS2-IC, simplemente colóquelo en la tarjeta de Educación o en el prototipo, conéctelo a un PC, y ejecute el software editor para crear y descargar su programa, a través del cable serie.

1.6 ¿QUÉ MICROCONTROLADOR EMPLEAR?

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

Costos. Como es lógico, los fabricantes de microcontroladores compiten duramente para vender sus productos. Y no les va demasiado mal ya que sin hacer demasiada publicidad venden 10 veces más microcontroladores que microprocesadores. Para que nos demos una idea, para el fabricante que usa el microcontrolador en su producto una diferencia de precio en el microcontrolador de algunos pesos es importante (el consumidor deberá pagar además el costo del empaquetado, el de los otros componentes, el diseño del hardware y el desarrollo del software). Si el fabricante desea reducir costos debe tener en cuenta las herramientas de apoyo con que va a contar: emuladores, simuladores, ensambladores, compiladores, etc. Es habitual que muchos de ellos siempre se decanten por microcontroladores pertenecientes a una única familia.

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

- Procesamiento de datos: puede ser necesario que el microcontrolador realice cálculos críticos en un tiempo limitado. En ese caso se debe asegurar de seleccionar un dispositivo suficientemente rápido para ello. Por otro lado, habrá que tener en cuenta la precisión de los datos a manejar: si no es suficiente con un microcontrolador de 8 bits, puede ser necesario acudir a microcontroladores de 16 ó 32 bits, o incluso a

hardware de punto flotante. Una alternativa más barata y quizá suficiente es usar librerías para manejar los datos de alta precisión.

- Entrada/Salida: para determinar las necesidades de Entrada/Salida del sistema es conveniente dibujar un diagrama de bloques del mismo, de tal forma que sea sencillo identificar la cantidad y tipo de señales a controlar. Una vez realizado este análisis puede ser necesario añadir periféricos hardware externos o cambiar a otro microcontrolador más adecuado a ese sistema.

- Consumo: algunos productos que incorporan microcontroladores están alimentados con baterías y su funcionamiento puede ser tan vital como activar una alarma antirrobo. Lo más conveniente en un caso como éste puede ser que el microcontrolador esté en estado de bajo consumo pero que despierte ante la activación de una señal (una interrupción) y ejecute el programa adecuado para procesarla.

- Memoria: para detectar las necesidades de memoria de nuestra aplicación debemos separarla en memoria volátil (RAM), memoria no volátil (ROM, EPROM, etc.) y memoria no volátil modificable (EEPROM). Este último tipo de memoria puede ser útil para incluir información específica de la aplicación como un número de serie o parámetros de calibración. El tipo de memoria a emplear vendrá determinado por el volumen de ventas previsto del producto: de menor a mayor volumen será conveniente emplear EPROM, OTP y ROM. En cuanto a la cantidad de memoria necesaria puede ser imprescindible realizar una versión preliminar, aunque sea en pseudo-código, de la aplicación y a partir de ella hacer una estimación de cuánta memoria volátil y no volátil es necesaria y si es conveniente disponer de memoria no volátil modificable.

- ❖ Ancho de palabra: el criterio de diseño debe ser seleccionar el microcontrolador de menor ancho de palabra que satisfaga los requerimientos de la aplicación. Usar un microcontrolador de 4 bits supondrá una reducción en los costos importante, mientras que uno de 8 bits puede ser el más adecuado si el ancho de los datos es de un byte. Los microcontroladores de 16 y 32 bits, debido a su elevado costo, deben reservarse para aplicaciones que requieran sus altas prestaciones (Entrada/Salida potente o espacio de direccionamiento muy elevado).

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

1.7 ¿QUÉ MICROCONTROLADOR USAREMOS?

Usaremos el PIC (Peripheral Interface Controller), analizaremos para esta tesis el modelo PIC16F84, la letra F hace referencia al tipo de memoria de programa, en este caso FLASH, si fuera PIC16C84, la letra C hace referencia al modelo con memoria de programa de tipo EEPROM.

El PIC16F84 esta dotado de 18 pines (véase en Fig. 1.7)[11], 13 de los cuales son para E/S de datos, 2 para alimentación, 2 para clock y uno para reset. Luego se explicara que significa cada término. Veamos una figura con la nomenclatura de sus respectivos pines:

[11] www.frino com.ar/micros.htm

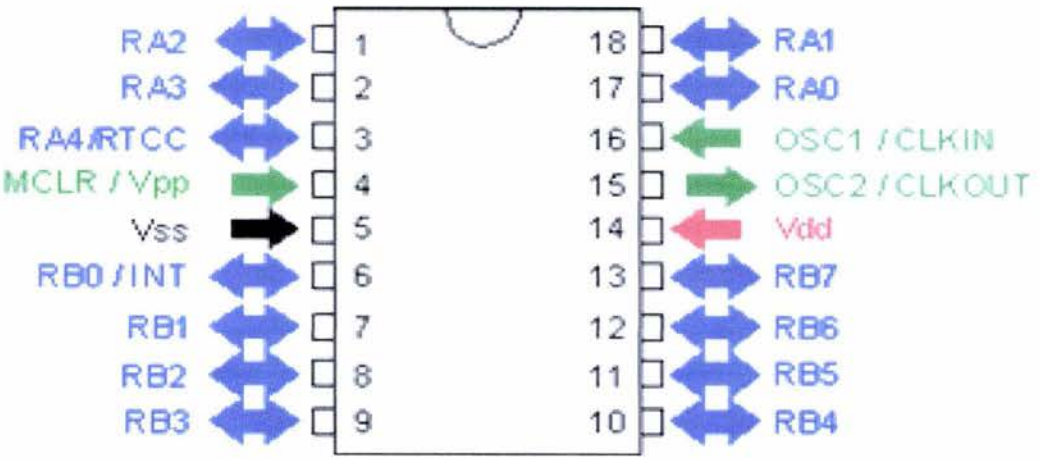


Fig. 1.7 PIC16F84

Como se puede ver, el PIC16F84 está dotado de un total de **18 pines** dispuestos en dos hileras paralelas de 9 pines cada una.

Los pines contrastados en AZUL representan las líneas de E/S (Entrada/Salida) disponibles para nuestra aplicación.

Los pines en ROJO y el **NEGRO** son los pines de alimentación,

Y los de VERDE son reservados para el funcionamiento del PIC (MCLR para el reset y OSC1-2 para el clock).

Capítulo 2

El PIC16F84

Contenido:

- **¿Qué es un PIC?**
- **Introducción histórica**
- **Tipos de PIC**
- **Familia PIC16F8X**
- **Arquitectura interna**
- **La unidad lógica aritmética (ALU)**
- **Diagrama de la estructura interna**
- **Organización de la memoria**
- **Memoria de programa**
- **Memoria de datos**
- **Puertos de E/S**

2.1 ¿QUÉ ES UN PIC?

Un PIC es un microcontrolador basado en memoria EPROM/FLASH desarrollado por Microchip Technology.

2.2 INTRODUCCIÓN HISTÓRICA

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 Division 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. Este fue un microprocesador razonable, pero no particularmente bueno manejando puertos de E/S. Para algunas aplicaciones muy específicas, GI diseñó un Controlador de Interface Periférico (PIC) entorno a 1975; 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 fuera, 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 por grandes fabricantes únicamente.

Durante los 80, 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 División 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.

2.3 TIPOS DE PIC

Una de las labores más importantes en el diseño es la elección del microcontrolador que mejor satisfaga las necesidades del proyecto con el mínimo presupuesto, como ya se vio en el capítulo anterior.

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

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

Gama baja con una memoria de programa (ROM o EPROM) de 12 bits y de 512, 1024 o 2048 palabras y una memoria de datos de 8 bits de 25, 72 o 73 bytes. Trabajan hasta 20 MHz y disponen de 12 o 20 líneas de E/S de alta corriente, un temporizador y de 33 instrucciones. Además, como el resto de los PICs disponen de perro guardián, Autoinicialización (POR o Power on Reset), modo de bajo consumo (SLEEP), reloj interno mediante cristal o red RC y protección contra lectura del código.

Gama media el ancho de la memoria de programa es de 14 bits, pudiendo ser ROM, EPROM o EEPROM. Incluyen interrupciones, conversores A/D, mayor número de temporizadores y otras características según el modelo. Su juego de instrucciones es de 35.

Gama alta estos microcontroladores poseen un verdadero bus de datos y direcciones, pudiendo trabajar con memoria externa. El ancho de la memoria de programa es de 16 bits y el número de instrucciones aumenta a 55 ó 58, según el modelo.

Gama enana con sólo 8 patillas, disponen de 6 líneas de E/S y una frecuencia máxima de 4 Mhz. Disponen de una red RC interna para el oscilador aprovechando al máximo las patillas disponibles. Algunos modelos incluyen conversores A/D y memoria EEPROM de datos. Esta gama es posterior a las anteriores para cubrir unas necesidades muy concretas, como es sustituir a un gran volumen de lógica cableada o implementar por hardware partes de sistemas microprocesadores más complejos.

2.4 FAMILIA PIC16F8X

Dispositivos integrantes de la familia:

FLASH-PROGRAMA	MEMORIA-DATOS	EEPROM-DATOS
PIC16F83 512 bytes	36 bytes	64 bytes
PIC16F84 1 K bytes	68 bytes	64 bytes

Adicionalmente, hay una familia derivada PIC16LF8X con voltaje extendido. Los PIC16F8X son iguales a los 16C8X salvo que sustituyen FLASH por EPROM para la memoria de programa.

2.4.1 Características de CPU

- CPU con núcleo RISC
- 35 instrucciones
- Ejecución de instrucciones en un ciclo (salvo saltos de programa en dos ciclos)
- Frecuencia máxima de trabajo: 10 MHz (400ns de ciclo de instrucción)
- 14 bits de tamaño de instrucción
- 8 bits de tamaño de datos
- 15 registros de función especial
- Stack de 8 niveles de profundidad
- Modos de direccionamiento directo, indirecto y relativo
- 4 fuentes de interrupción:
  - pin externo RB0/INT
  - exceso de temporizador TMR0
  - interrupción por cambio del puerto B (7:4)

- escritura completa de EEPROM

- ❖ 1 millón de ciclos de borrado/escritura en EEPROM
- ❖ Retención de datos en EEPROM > 40 años

2.4.2 Características de periferia

- ❖ 13 pins de E/S para control de direccionamiento individual
- ❖ Alta corriente de salida para manejar LEDs directamente:
- ❖ 25 mA de pico por pin
- ❖ 20 mA entregados por pin
- ❖ TMR0: contador/temporizador de 8 bits pre-escalar y programable

2.4.3 Características especiales del Microcontrolador

- ❖ Power on Reset (POR)
- ❖ Power por temporización (PWRT)
- ❖ Temporizador de comienzo por oscilador (OST)
- ❖ Temporizador de "perro guardián" o watchdog (WDT) con su propio oscilador RC para mejor operación
- ❖ Protección de código
- ❖ Modo SLEEP para ahorro de energía
- ❖ Programación del sistema por puerto serie de dos pines (los dispositivos ROM sólo soportan programación de datos en EEPROM)

2.4.4 Tecnología CMOS

- Tecnología de baja potencia y alta velocidad CDMOS Flash/EEPROM
- Diseño completamente estático
- Rango de voltajes de trabajo:
  - Comercial: 2.0V a 6.0V
  - Industrial: 2.0V a 6.0V
- Bajo consumo de potencia:
  - <2mA típicos a 5V y 4MHz
  - 15uA típicos a 2V y 32 KHz
- 1 uA típico de corriente de "standby" o espera a 2V

2.4.5 Diagrama Lógico

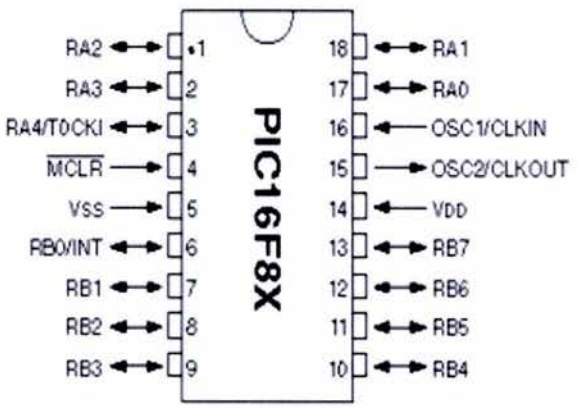


Fig. 2.1[12] Diagrama del PIC16F8X

[12] www.microchip.com

Nº pin	Nombre	Tipo (E/S/A)	Tipo de buffer	Descripción
16	OSC1/CLKIN	E	ST/CMOS (3)	entrada del cristal oscilador / entrada externa de reloj
15	OSC2/CLKOUT	S	-	Salida del cristal oscilador. En el modo de oscilación por cristal se conecta al cristal o resonador. En modo RC OSC2 proporciona CLKOUT (salida de reloj), que posee 1/4 de la frecuencia de OSC1, y representa el ciclo de instrucción.
4	MCLR'	E/A	ST	Entrada de reset / entrada de voltaje de programación. Este pin es un reset activo a nivel lógico bajo del dispositivo.
17 18 1 2 3	RA0 RA1 RA2 RA3 RA4/T0CKI	E/S E/S E/S E/S E/S	TTL TTL TTL TTL ST	PORTA es un puerto de E/S bidireccional Puede ser seleccionado también para ser la entrada de reloj al contador/temporizador TMR0. La salida es de colector abierto.

6	RB0/INT	E/S	TTL/ST	PORTB es un puerto de E/S
7	RB1	E/S	(1)	bidireccional que puede ser
8	RB2	E/S	TTL	programado para levantar
9	RB3	E/S	TTL	internamente todas las entradas.
10	RB4	E/S	TTL	
11	RB5	E/S	TTL	RB0/INT puede ser seleccionado
12	RB6	E/S	TTL	como un pin de interrupción externa
13	RB7	E/S	TTL/ST	
			(2)	RB4 a RB7 son pines de interrupción
			TTL/ST	por cambio de estado.
			(2)	RB6 es la entrada de reloj de programación y RB7 la entrada de programación serie.
5	Vss	A	-	Referencia de tierra (masa) para todos los pines lógicos de E/S.
14	Vdd	A	-	Alimentación positiva para todos los pines lógicos de E/S.

Leyenda y notas:

E=entrada S=salida E/S=entrada/salida A=alimentación

-=no empleado TTL=entrada TTL ST=entrada Schmitt

(1) Este buffer es una entrada Schmitt cuando se configura como interrupción externa

(2) Este buffer es una entrada Schmitt cuando se emplea en el modo de programación serie

(3) Este buffer es una entrada Schmitt cuando se configura en el modo de oscilación RC y como entrada CMOS

2.5 ARQUITECTURA INTERNA

Los microcontroladores PIC utilizan la arquitectura Harvard, la cual separa la memoria de programa de la memoria de datos. Esto hace que el dispositivo tenga un bus de datos y un bus de memoria de programa, hecho que permite acceder a ambos simultáneamente. En el caso de la memoria de programa es interna, ya que se encuentra en forma de EPROM o FLASH dentro del propio PIC.

Se puede observar claramente que las principales ventajas de esta arquitectura son:

a) Que el tamaño de las instrucciones no esta relacionado con el de los datos, y por lo tanto puede ser optimizado para que cualquier instrucción ocupe una sola posición de memoria de programa, logrando así mayor velocidad y menor longitud de programa.
b) Que el tiempo de acceso a las instrucciones puede superponerse con el de los datos, logrando una mayor velocidad de operación.

Una pequeña desventaja de los procesadores con arquitectura Harvard, es que deben poseer instrucciones especiales para acceder a tablas de valores constantes que pueda ser necesario incluir en los programas, ya que estas tablas se encontraran físicamente en la memoria de programa (por ejemplo en la EPROM de un microprocesador).

Otra característica reseñable es que las instrucciones en los PIC son de 14 bits y no de 8, permitiendo instrucciones de palabra única con operación y operando en la misma palabra.

Esta estructura, acompañada de un pipeline, permite que las instrucciones se ejecuten en un sólo ciclo de reloj, salvo en el caso de saltos/rupturas de programa. Lo que se hace internamente es que mientras se ejecuta la instrucción

actual se carga la siguiente instrucción en el pipeline, con lo que se alcanza una alta velocidad de ejecución.

En cuanto a los registros los PIC pueden direccionarse tanto directa como indirectamente. Todos los registros de función especial, incluyendo el contador de programa, están accesibles en la memoria de datos y pueden ser operados de cualquier forma y empleando cualquier modo de direccionamiento (ortogonalización).

Igualmente, todos los elementos del sistema (temporizadores, puertos de E/S, etc.) están implementados como registros.

2.6 LA UNIDAD LÓGICA ARITMÉTICA (ALU)

Los PIC poseen una ALU de 8 bits y un registro de trabajo (W) de 8 bits, pudiéndose efectuar operaciones aritméticas y booleanas entre el registro de trabajo y cualquier otro registro. Por naturaleza los datos se operan en complemento a 2, a menos que se diga lo contrario.

En función de las instrucciones ejecutadas la ALU puede afectar los valores de los siguientes registros:

Acarreo (C)

Acarreo de dígito (DC)

bit Cero (Z) del registro de estado.

2.7 DIAGRAMA DE LA ESTRUCTURA INTERNA

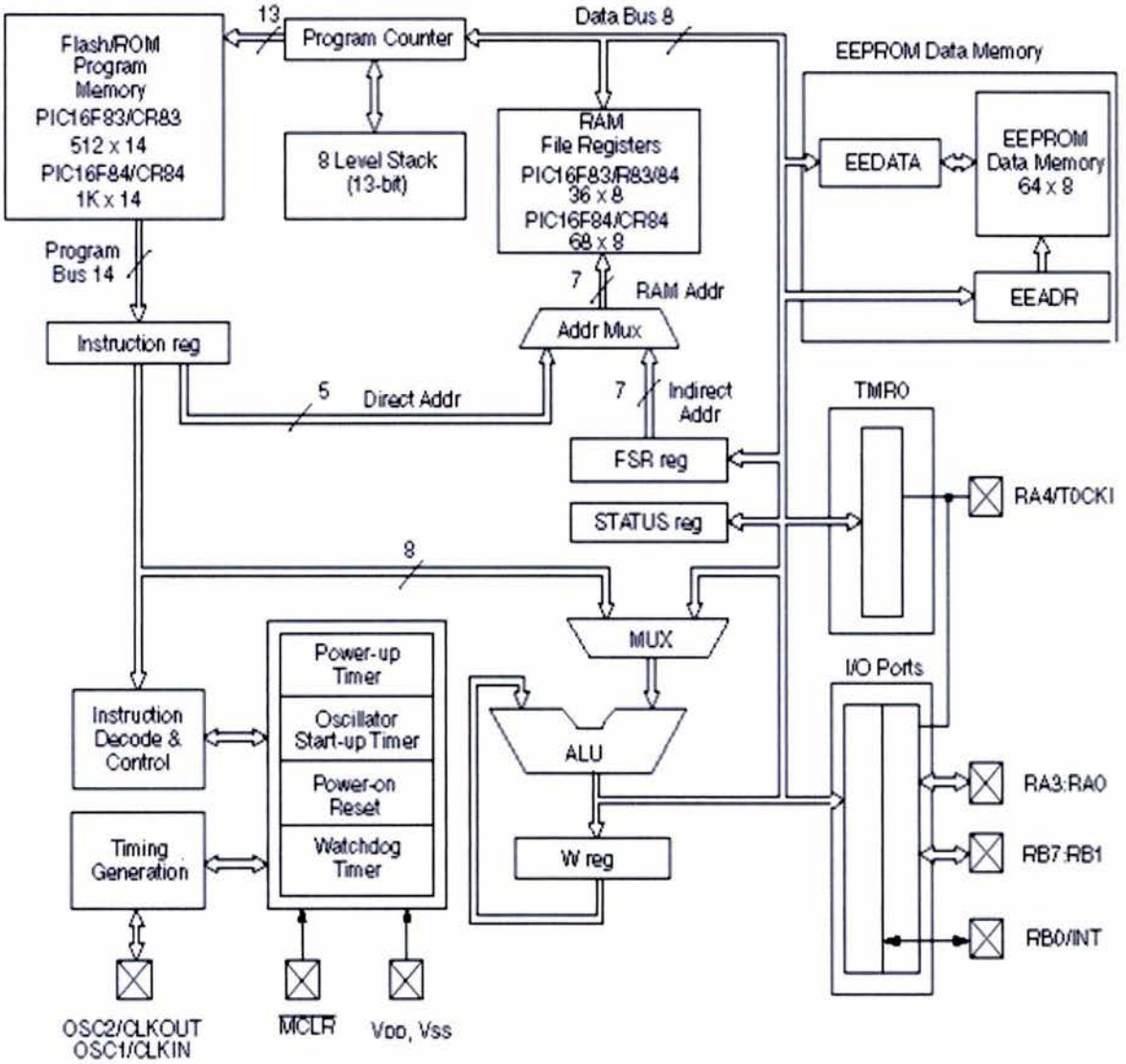


Fig. 2.2[13] Estructura interna del PIC16F84

[13] Progamacion Assembler de pic

2.8 ORGANIZACIÓN DE LA MEMORIA

Hay dos bloques de memoria en el PIC16F8x (véase en Fig. 2.3). Éstas son la memoria del programa y la memoria de datos. Cada bloque posee su propio bus, con la finalidad que el acceso para cada bloque pueda ocurrir durante el mismo ciclo del oscilador.

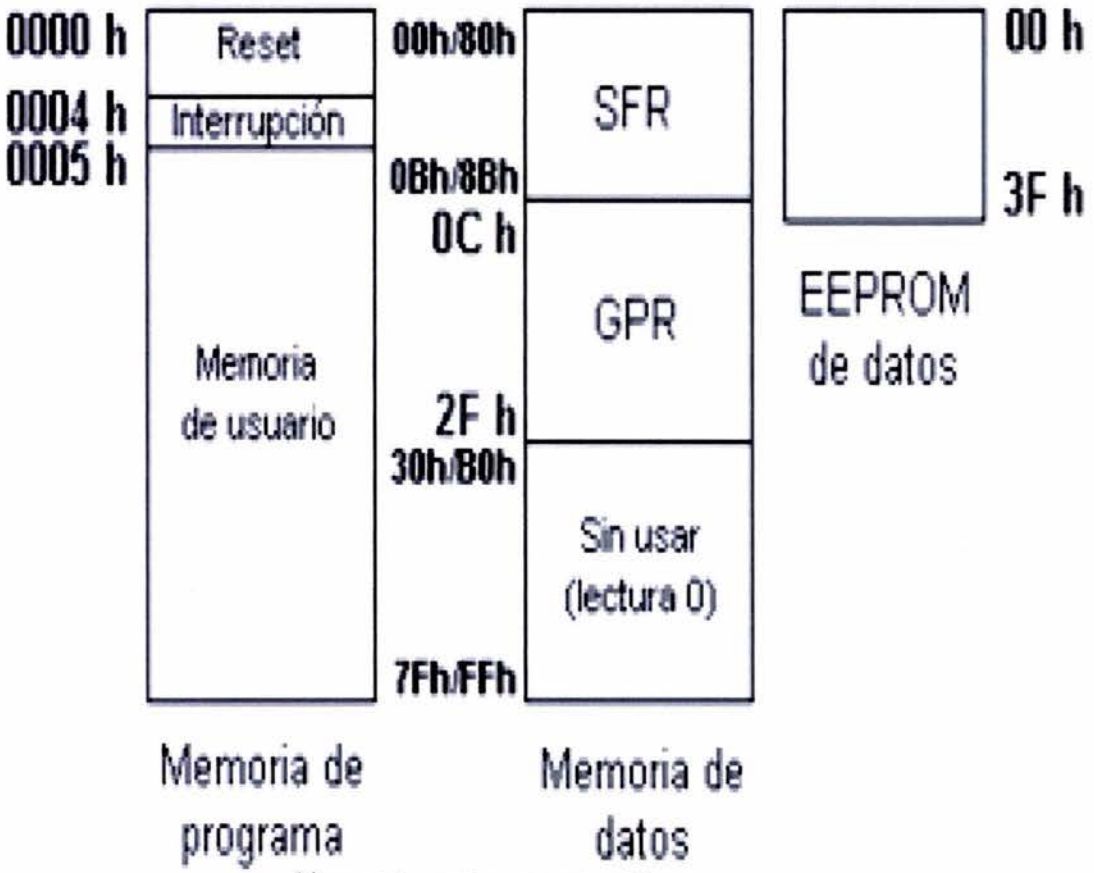


Fig. 2.3[14] BLOQUES DE MEMORIA

[14] Figura creada por el autor

2.9 MEMORIA DE PROGRAMA

La memoria de programa está organizada en palabras de 14 bits y es del tipo FLASH. Esta memoria es de sólo lectura y únicamente se ejecutará el código contenido en ella. El vector de reset se encuentra en la posición 0000h y el de interrupciones en la 0004h, por lo que la memoria de usuario se extiende desde la dirección 0005h.

2.10 MEMORIA DE DATOS

La memoria de datos está dividida en Registros de Propósito General (GPR) y los Registros (véase en Fig.2.4) de Función Especiales (SFR). Los SFR controlan la operación del dispositivo.

Para el estudio de esta memoria se suele dividir virtualmente la misma en dos bancos, tal y como podemos ver en el gráfico adjunto. De esta forma los registros GPR están agrupados entre 00h-0Bh y 80h-8Bh. Los registros de propósito general pueden ser accedidos desde 0Ch-2Fh o 8Ch-AFh, aunque se recomienda siempre el primer intervalo.

Las zonas de memoria 30h-7Fh y B0h-FFh no son empleadas y devuelven 0 en lectura.

El área de memoria de datos también contiene la memoria de datos EEPROM. Esta memoria no está directamente mapeada en la memoria de datos, pero está indirectamente mapeada. Esto es, un puntero de dirección indirecta especifica la dirección de la memoria de datos EEPROM para lectura/escritura. Los 64 bytes de memoria de datos EEPROM tienen el rango de dirección 00h-3Fh.

Para acceder a la EEPROM en lectura y escritura empleamos dos registros, que forman el puntero de dirección indirecta:

- EEDATA (0008h), para datos

❖ EEADR (0009h), para direcciones

Para definir el modo de funcionamiento se emplean los registros especiales:

- EECON1 (0088h)
- EECON2 (0089h)

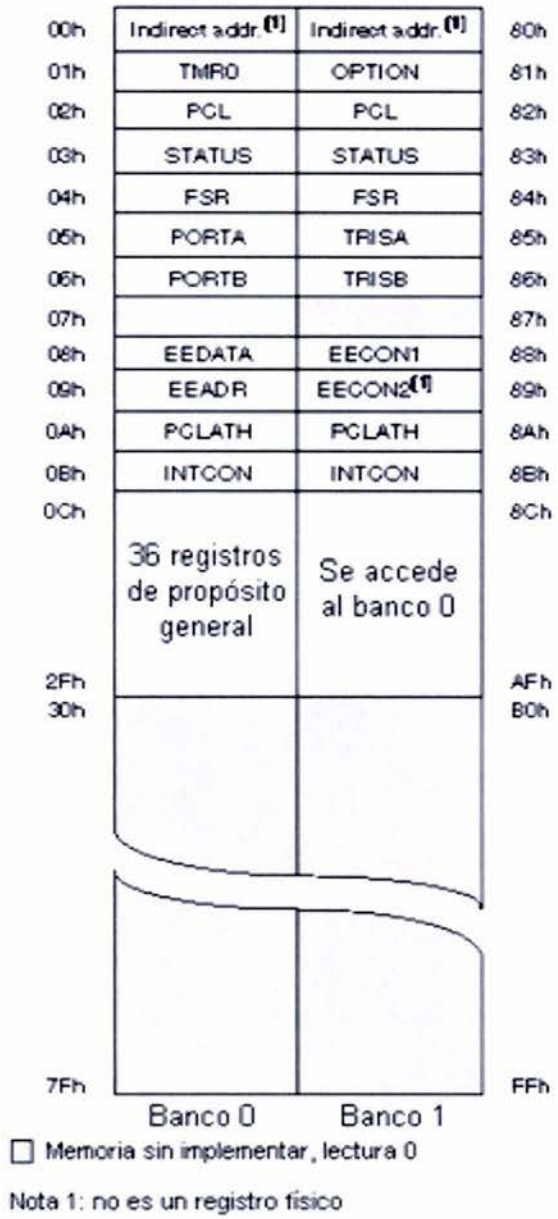
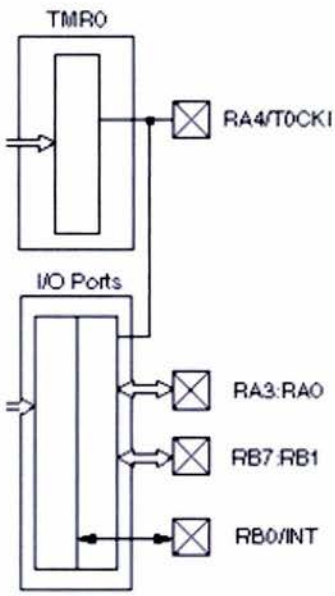


Fig.2.4 Registros del PIC16F84

2.11 PUERTOS DE E/S



Disponemos de dos puertos de entrada y salida (E/S).

Puerto A

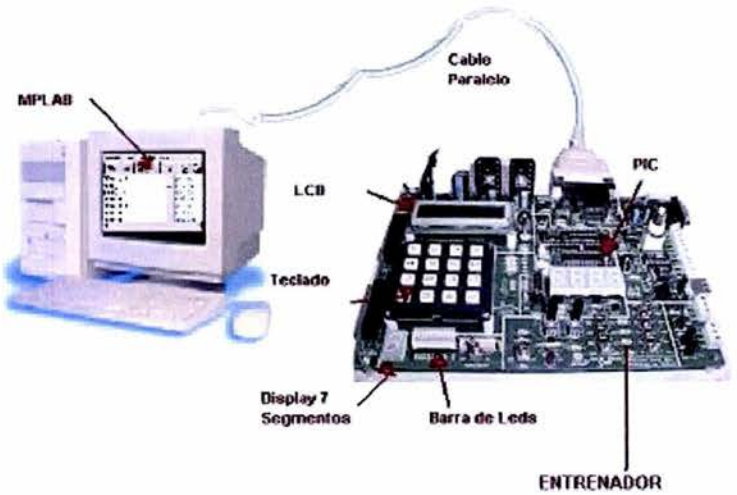
Posee 5 líneas (RA0 a RA4), en la que RA4 o T0CKI es compartida con la entrada para el temporizador 0 (TMR0).

Puerto B

Posee 8 líneas de E/S (RB0 a RB7), en la que la línea RB0 o INT es compartida con la entrada de interrupción externa.

Capítulo 3

PROGRAMACION DEL PIC16F84



Contenido:

- **Enfoque general de funcionamiento de los software de programación**
- **Funcionamiento en el interior del PIC**
- **Hardware y software necesario en toda aplicación**
- **Memoria de Programa Flash**
- **Memoria RAM**
- **¿Qué es un ciclo de instrucción?**
- **Introducción a los registros**
- **Instrucciones**
- **Temporizando**
- **Interrupciones**

3.1 ENFOQUE GENERAL DE FUNCIONAMIENTO DE LOS SOFTWARE DE PROGRAMACIÓN

Canales por los que llega el programa al PIC:

En primer lugar, necesitamos de un software de ambiente agradable al entorno Windows donde poder realizar y simular nuestros programas, para esto tenemos el IDE (Integrate Development Enviroment), es decir, Ambiente de Desarrollo Integrado más conocido como MPLAB. Este software será el encargado de compilar nuestro código fuente, que no es más que el conjunto de instrucciones que editemos en el MPLAB y que tienen extensión .asm.

MPLAB es un software proporcionado gratuitamente por la microchip, en éste editamos nuestras diferentes directivas e instrucciones de programa, es acá donde plasmamos nuestra capacidad e intuición de programación.

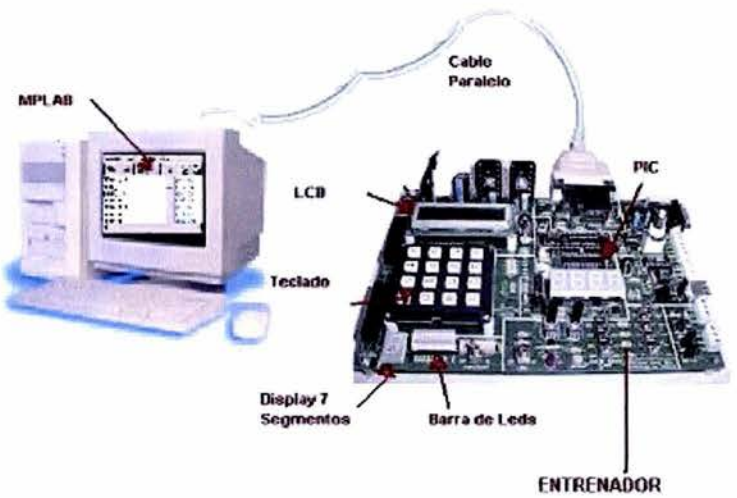


Fig. 3.1 Interfase PC y Programador

Una vez depurado nuestro programa, este genera un archivo de salida con extensión .hex.

Este archivo .hex se carga primero en una interface de usuario, la cual usaremos el IcProg, ya que es con el que programamos.

En el IcProg se configura tipo de puerto de comunicación, en nuestro caso el serial, el tipo de procesador que usamos, es decir, el modelo de PIC, tipo de oscilador que dará la frecuencia de reloj.

EL archivo .hex contiene el código de operación (Código OP) que será enviado a la memoria de programa (FLASH o EEPROM) dentro del PIC por medio del cable serial y del circuito de programación que forma parte del entrenador.

El archivo .hex no es un archivo en formato binario y no refleja directamente el contenido que deberá tener la memoria de programa (FLASH o EEPROM) del PIC. Pero los formatos reflejarán directamente cuando sean transferidos al PIC en forma de bajo nivel y con algunas instrucciones más.

Sin entrar en detalles, es útil saber que tal formato es directamente reconocido por el hardware del PIC que promueve durante la programación la conversión en binario del código de operación (código OP).

Si se pregunta ¿por qué código binario?, es simplemente porque como buena microcomputadora, al igual que cualquier computador, procesador y sistema digital, el PIC entiende solamente código binario. Es por esta razón que se necesita siempre de un software de aplicación que compila o traduce a binario la serie de instrucciones que editamos y conocemos como código fuente.

Asumiremos entonces que el Código de Operación, o Código OP, es simplemente el código de nuestras instrucciones en binario.

Ejemplo:

$00\ 0001\ 0000\ 0000^2$

En notación hexadecimal será:

0100H

Vemos que este código tiene una extensión de 14 bits, que es la extensión de la memoria de programa.

Este código, completamente sin sentido para los humanos, es lo que el PIC esta preparado para entender.

Para facilitar la comprensión del programador (nosotros), se recurre a un instrumento y convención para tornar la instrucción más comprensible.

La primera convención es la que asocia el código OP (un total de 35 para o PIC16x84; letra "x" puede ser F, C ó CR) a una sigla nemónica, o sea una inicial que nos permitirá recordar fácilmente el significado de la instrucción.

Volviendo a nuestro ejemplo, el opcode o código OP 0100H corresponde a la instrucción nemónica CLRW que es una forma abreviada de la instrucción CLEAR W REGISTER (veremos posteriormente lo que significa).

De esta manera, todos los nemónicos o siglas de las instrucciones de programación tienen su respectivo código OP.

3.2 FUNCIONAMIENTO EN EL INTERIOR DEL PIC

Viendo la arquitectura interna del PIC sus componentes principales son: el CPU, memoria RAM, memoria FLASH o EEPROM, puertos E/S.

Una vez que tenemos grabado el código de operación dentro de la memoria de programa o simplemente nuestro programa, el PIC, está listo para realizar su función encomendada.

Todo el funcionamiento interno del PIC se trata de **manejo y configuración de bits de archivos.**

Estos archivos se encuentran implementados en la memoria RAM y cada uno tiene una longitud de 8 bits, cada bit de cada archivo cumple una función; por esta razón a estos archivos que ocupan las primeras posiciones de la memoria RAM se les llama Registros de Función Específica o SFR (Special Function Register).

También existen posiciones de la memoria RAM que no están ocupados por los SFR, y nos sirven para implementar nuestros propios registros, por esto se les llama Registros de Propósito General GPR (General Purpose Register).

En resumen, existe un uso intensivo de los registros de la memoria RAM en su interacción con la memoria de programa y la CPU.

3.3 HARDWARE Y SOFTWARE NECESARIO EN TODA APLICACIÓN

3.3.1 Hardware

En Hardware tenemos 3 circuitos estarán presentes de la misma manera para todas las aplicaciones que hagamos: Esta misma configuración se seguirá para los otros modelos del PIC, en lo único que se diferencian son en los pines en los que se encuentran implementados: Por ejemplo:

En el PIC16F873 de 28 pines: pin 1 Reset o MCLR (Master CLear)
pines 9 y 10 Circuito oscilador
pines 8 y 19 para Vss pin 20 para Vdd.

En el PIC16F877 de 40 pines: pin 1 Reset o MCLR (Master CLear)
pines 13 y 14 Circuito oscilador
pines 12 y 31 para Vss pines 11 y 32 para Vdd.

- Circuito Oscilador: Este circuito (véase en Fig. 3.2) sirve para dar la frecuencia de operación del microcontrolador, y de esta manera establecer la velocidad de ejecución de las instrucciones, En la figura se observa un circuito oscilador formado por un cristal de cuarzo y 2 condensadores del orden de los picofaradios. También se puede hacer

el circuito oscilador de tipo RC, pero es más usado el cristal de cuarzo de 2MHz a 20MHz.

- Circuito de Reset: Este circuito (véase en Fig. 3.2) es simplemente para reinicializar el programa dentro del PIC, al igual que una PC común que tiene su Reset.

Circuito de alimentación: El microcontrolador trabaja a una tensión nominal de 5V, y le corresponde al pin 14, tomar este valor. También se tiene naturalmente la referencia a tierra, siempre importante en cualquier circuito eléctrico, sea por motivos de protección o medición.

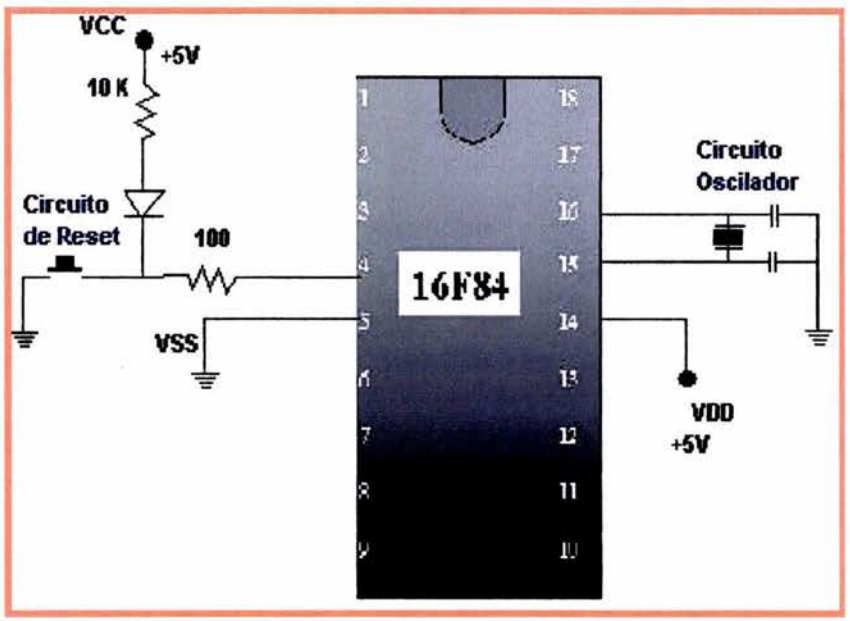


Fig. 3.2 HARDWARE NECESARIO EN TODA APLICACIÓN

3.3.2 Software

Más específicamente en el MPLAB, existen directivas generales que se repetirán en la confección de todos nuestros programas.

Este es el entorno de trabajo en el que trabajaremos. Todo lo que va seguido de ";" no es tomado en cuenta a la hora de compilar el programa, por lo tanto esto nos permite colocar una breve descripción de lo que hará nuestro programa, fecha, autor, y cualquier comentario que queramos hacer a lo largo de las líneas de programa.

LIST, nos especifica el tipo o modelo de PIC que se va a usar.

RADIX, nos especifica el tipo de numeración, en este caso hexadecimal. En la zona de etiquetas definimos algunos registros FSR que formaran parte de nuestro programa (PORTA Y TRISA en nuestro ejemplo) así como también los registros creados por nosotros (mi_registro1 y mi_registro2) y sus posiciones en la memoria RAM (véase la Fig. 3.3).

Fig. 3.3 AMBIENTE MPLAB

Equ es una directiva que significa igual, y 0xNumero, es la dirección, en notación hexadecimal, del registro. La directiva **equ** es muy importante cuando se trata de definir con ella una constante simbólica dentro de nuestro código.

La directiva **org** es una directiva que permite definir la dirección en la cual queremos que el compilador inicie el alojamiento de las instrucciones en el área de la memoria de programa (FLASH o EEPROM). Desde este punto en adelante colocaremos las siglas de las instrucciones que el compilador deberá convertir en su respectivo código OP (Código de Operación) del PIC.
goto INICIO, nos vamos al programa principal.

La directiva **ORG 0x05,** salta la posición 0x04 que es la que ocupa el vector de interrupción.

La directiva **include**, hace referencia a un archivo, e incluye en nuestro programa todo el contenido de este archivo, el contenido del archivo **p16f84.inc** es la definición de todas las posiciones de los registros FSR y de sus respectivos bits, por esta razón, muchas veces se omite especificar las posiciones de los FSR en la zona de etiquetas, siendo necesario solamente colocar nuestros registros.

En resumen, tenemos dos vectores implementados en la memoria de programa y son; el vector del reset que corresponde a la primera posición de la memoria de programa y donde se inicia e alojamiento de nuestro código de operación y el vector de interrupción que en este caso se salta.

También podemos incluir cualquier archivo de extensión **.inc.**, y todo su contenido automáticamente formará parte de nuestro programa.

Estos archivos **include** son muy usados en aplicaciones complejas y en las que se necesitarán de archivos auxiliares, para no saturar nuestro programa.

Hay que respetar el orden de las columnas como esta escrito el código, las etiquetas siempre van en el extremo izquierdo, luego las instrucciones en el medio y a la derecha, la referencia a la cual están dirigidas las instrucciones, que son normalmente los registros.

3.4 MEMORIA DE PROGRAMA FLASH

Esta memoria es usada para almacenar el programa. Una palabra esta compuesto por 14 bits y están instalados 1K de palabras (1024 palabras). Aunque se desconecte la alimentación el contenido almacenado en la memoria flash no desaparece.

El contenido de la memoria puede ser reescrito, existen aparatos llamados grabadores de microcontroladores con los que se hace esta operación de borrado, lectura y escritura, pero el numero de reescrituras es limitado, en este tipo de memoria solo permite 10000.

3.5 VECTOR DEL RESET (0000h)

Es la primera posición de la memoria de programa, cuando el reset es ejecutado por encendido, o terminado el tiempo del perro guardián (whatchdog), u otro factor, el programa empieza desde esta dirección, después de ocurrido el reset.

3.6 VECTOR DE INTERRUPCIÓN (0004h)

Cuando se habilita la interrupción y esta ocurre por algún evento interno del microcontrolador o se hace una interrupción externa, el programa empieza desde esta dirección.

3.7 PALABRA DE CONFIGURACIÓN (2007h)

Es un espacio de la memoria reservado para configurar el tipo de oscilador, habilitar el temporizador watchdog, etc. Esta área no puede ser fijada por el

programa. Es necesario escribirlo usando el interfaz de usuario junto con el grabador.

El resto del espacio es para que el usuario pueda escribir su programa, incluyendo el vector del reset.

Como se puede observar (véase la Fig. 3.4), solo están implantados 1K de memoria, es decir 1024 posiciones equivalente a 03FFh en notación hexadecimal.

Fig. 3.4 REGISTROS DE CONFIGURACIÓN DEL PIC16F84

3.8 MEMORIA RAM

Esta memoria está dividida en dos áreas llamados bancos y tiene una capacidad de 80 bytes (00h - 4Fh), por banco.

Cada banco contiene espacios de memoria de 8 bits de longitud en los que se encuentran los SFR y GPR (véase la Fig. 3.5).

Los 12 primeros bytes (00h - 0Bh) de cada banco son llamados Registros de Función Específica (SFR) y son usados de manera permanente por el programa, para configurar los puertos, establecer el estado de operación, y otras condiciones que el usuario decida.

Los otros 68 bytes (0Ch - 4Fh), son llamados Registros de propósito general (GPR) y son usados por el usuario para implementar registros

Todos los SFR tienen un contenido por defecto después de conectar la alimentación, cuando empieza a correr nuestro programa dentro de la memoria flash, el contenido de los SFR y los GPR que hayamos creado están continuamente cambiando al ritmo con que se esta ejecutando cada línea de programa.

Cuando se desconecta la alimentación, el contenido actual de los SFR y GPR se pierden.

La zona de blanco de la figura que corresponde al banco 1, significa que el contenido es el mismo que el del banco 0, es decir, está mapeado.

Otra memoria es la EEPROM de datos que también tiene una longitud de 8 bits y se usa para memorizar variables de manera permanente.

Notamos pues que estas dos importantes memorias tienen longitudes diferentes, RAM de 8 bits y FLash de 14 bits, esta configuración es propio de la arquitectura Harvard, en la que la RAM y el Flash están conectados a la CPU con buses diferentes.

La ventaja de esta arquitectura es que el tamaño de las instrucciones no está relacionado con la de los datos de la RAM, por lo tanto, puede ser optimizada para que cualquier instrucción ocupe una sola posición de la memoria de programa logrando mayor velocidad y menor longitud de programa.

Dirección	Banco 0	Banco 1	Dirección
00h	INDF	←	80h
01h	TMR0	OPTION_REG	81h
02h	PCL	←	82h
03h	STATUS	←	83h
04h	FSR	←	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	Unimplemented	←	87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2	89h
0Ah	PCLATH	←	8Ah
0Bh	INTCON	←	8Bh
0Ch - 4Fh	GPR	←	8Ch - CFh

Fig. 3.5 MEMORIA RAM

Para finalizar, se puede señalar que, de todos estos registros SFR, solamente 6 contienen los bits de control, señalización y configuración. Veamos, grosso modo, lo que hacen:

- STATUS:(direcciones 03h y 83h) Contiene el estado aritmético de la ALU, el estado del reset y los bits de selección de banco a acceder en la memoria de datos.
- OPTION:(dirección 81h) Configuran predivisor de frecuencia a TMR0 o Watchdog, interrupción externa, resistencia pull-up sobre el puerto B.
- INTCON: Contiene los bits de activación de todas las fuentes de interrupción.
- EECON1:Bits de configuración para lectura y escritura de la EEPROM de datos.
- TRISA y TRISB: Contiene bits de configuración de pines como entradas o salidas digitales.

3.9 DETALLES DE FUNCIONAMIENTO INTERNO

Las instrucciones del programa residentes en la memoria Flash o EEPROM, hacen referencia a los bits de cada registro de la memoria RAM y dependiendo del tipo de aplicación, se ordena sacar por los puertos a través de la CPU las señales deseadas.

La Unidad Central de Proceso **(CPU)** está formado por el contador de programa **(PC),** el registro de instrucciones **(RI),** el decodificador y control de instrucciones **(DCI),** la unidad aritmética lógica **(ALU)** y el registro de trabajo **(W).**

3.10 ¿QUÉ ES UN CICLO DE INSTRUCCIÓN?

Es el tiempo que demora en ejecutarse una instrucción, es decir el tiempo que demora en pasar de una línea a otra línea de programa, dentro de la memoria Flash.

Un ciclo de instrucción es igual a 4 ciclos de reloj, y el ciclo de reloj lo define el oscilador externo al PIC, por ejemplo, si tenemos como oscilador un cristal de cuarzo de frecuencia de 4Mhz, entonces el ciclo de reloj será 1/4 = 0.25 microsegundos. Por lo tanto el ciclo de instrucción será 4x0.25 = 1 microsegundo.

Estos 4 ciclos de reloj que se demora un ciclo de instrucción se divide en dos fases: **búsqueda y ejecución.**

Es decir mientras esta ejecutando una instrucción, esta buscando la siguiente instrucción, y es por esto que se le atribuye la característica de procesador de tipo segmentado y es lo que lo diferencia de la arquitectura de los microprocesadores.

3.11 ¿QUÉ ES Y QUÉ HACE EL CONTADOR DE PROGRAMA (PC)?

El PC forma parte del equipo de la CPU, y su función es la de iniciar todo el proceso de un ciclo de instrucción.

Todo empieza con la fase de búsqueda de la dirección en la que cual se encuentra la instrucción, (código OP de la instrucción), dentro de la memoria de programa (véase la Fig. 3.6).

En esta fase de búsqueda, el PC facilita el direccionamiento de las posiciones de las instrucciones que residen en la memoria de programa.

En otras palabras, la información que nos ofrece el PC es simplemente la dirección actual de la instrucción que se esta ejecutando durante el flujo normal del programa. El modelo PIC16F84, tiene implementado 1 K de memoria de programa, es decir 1024 direcciones o sea 1024 posiciones para que puedan ser ocupadas por nuestras instrucciones.

El PC tiene una longitud de 13 bits, es decir pueden direccionarse 4 K de memoria de programa, pero como el modelo PIC16F84 solo tiene implementado 1 K es suficiente 10 bits para direccionar toda esta memoria, por lo tanto los 3 bits de mas peso (MSB), no se consideran.

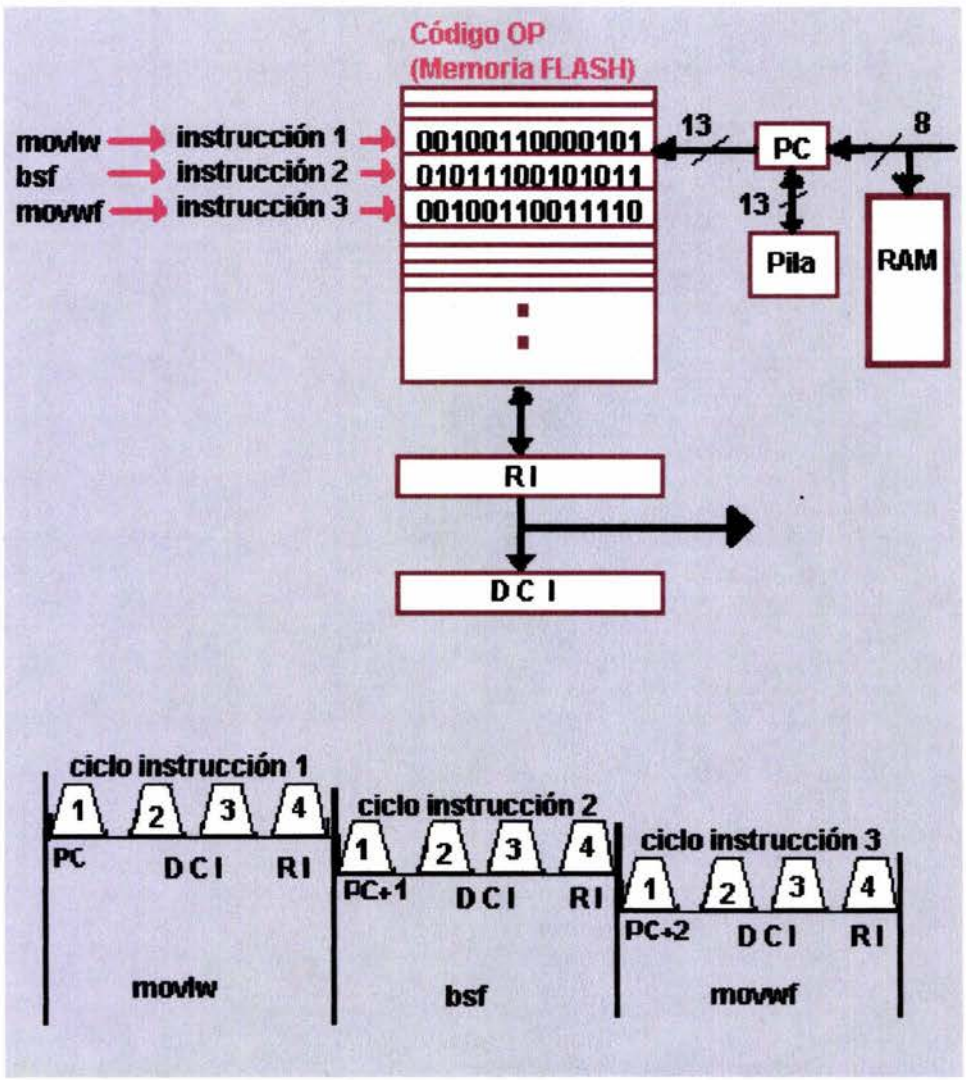


Fig. 3.6 CICLO DE INSTRUCCIÓN

3.12 ANÁLISIS DE UN CICLO DE INSTRUCCIÓN

Durante el primer ciclo de reloj se incrementa el PC (como se muestra en la Fig. 3.6); si la instrucción 1 tiene la dirección PC, que puede ser por ejemplo 0008h, en el siguiente ciclo de instrucción durante el primer ciclo de reloj, la dirección a la que se apunta será PC+1, es decir, 0009h, y así sucesivamente.

Durante los ciclos 2 y 3 de reloj, se produce la decodificación y ejecución de la instrucción, **(DCI)**, luego la **ALU** es la encargada de realizar la operación lógica aritmética que implica la instrucción decodificada

Durante el ciclo 4 de reloj se busca el código de la siguiente instrucción y se carga en el registro de instrucciones **(RI),** y así sucesivamente

Recordemos que cada línea de programa dentro de la memoria, en este caso Flash, siempre esta haciendo referencia a los archivos de la memoria RAM y a sus respectivos bits, de esta manera se tiene una ínteractuación permanente de todos los componentes de la arquitectura interna del microcontrolador.

La ínteractuación con uno u otro registro de la memoria RAM, dependerá del programa, es decir, es el programa el que especifica con qué registros de la memoria RAM está trabajando y esto depende directamente de la aplicación que estemos realizando.

Más adelante veremos el funcionamiento de un programa contrastado con la arquitectura interna del PIC y poco a poco iremos definiendo que función cumple cada bit de cada registro SFR.

3.13 INTRODUCCIÓN A LOS REGISTROS

Suponiendo que hemos definido los pines del micro que actuarán como entradas y salidas, empezamos con los registros STATUS, TRISA, TRISB y enseguida los registros OPTION e INTCON.

Veremos paralelamente algunas instrucciones y registros, y más adelante se explicará con más detenimiento cada instrucción.

3.13.1 Selección de banco de memoria y configuración de puertos

Los registros TRISA y TRISB contienen los bits que me configuran los pines como entradas o salidas

Observando el mapa de memoria de la RAM, vemos que estos registros TRISA y TRISB se encuentran en las direcciones 85h y 86h respectivamente, lo que es equivalente a decir que están en las direcciones 05h y 06h pero en el banco1.

Cuando se usa un microcontrolador por primera vez, por defecto se posiciona en el banco 0 de la memoria de datos RAM, y como los registros de configuración de puertos TRIS A y TRIS B se encentran en el banco 1 entonces ¿cómo nos cambiamos al banco 1?

Para acceder al banco 1 usamos dos bits del registro de estado (STATUS) el RP0 y el RP1.

3.13.2 STATUS (Registro de estado)

IRP	RP1	RP0	/TO	/PD	Z	DC	C

- **C: Acarreo en el 8º bit**.

  1 = acarreo en la suma y no en la resta. 0 = acarreo en la resta y no en la suma.

- **DC: Acarreo en el 4º bit de menor peso.**

  Igual que C.

- **Z: Zero.**

  1 = El resultado de alguna operación es 0. 0 = El resultado es distinto de 0.

- **/PD: Power Down.**

  1 = Recién encendido o tras CLRWDT. 0 = Tras ejecutar una instrucción SLEEP.

- **/TO: Timer Out.**

  1 = Recién encendido, tras CLRWDT, o SLEEP. 0 = Saltó el WDT.

- **RP1-RP0.**

  Bits de Selección de Banco de Registro, usado para direccionamiento directo.

- **IRP.**

  No es usado por el PIC16F84A.

La combinación binaria de los bits RP0 y RP1, se pueden definir 4 bancos, pero como solo tenemos 2 en el caso del PIC16x84, sólo usamos la combinación binaria 00 y 01.

La combinación 00, es decir RP1=0 y RP0=0 me posicionan en el banco 0 (00h – 7Fh).

La combinación 01, es decir RP1=0 y RP0=1 me posicionan en el banco 1 (80h - FFh).

Entonces veamos las instrucciones que me ponen los bits a cero o uno.

Instrucción	Significado	Definición
bcf	Bit Clear f:	Pone a 0 el bit del registro F
bsf	Bit Set f:	Pone a 1 el bit del registro F

bcf y bsf son instrucciones orientadas a bits, es decir, modifican el estado de un bit de cualquier registro

Para configurar los pines de los puertos como entradas o salidas, usamos dos instrucciones orientadas a bytes.

- **movlw** Mover valor literal a registro de trabajo W
- **movwf** Mover valor cargado en W a registro F
- **clrf** Pone a 0 todos los bits el registro F
- **clrw** Pone a 0 todos los bits el registro W

Ejemplo:

```
inicio  bsf     STATUS,RP0      ;Cambio al banco 1
        clrf    TRISA           ;Puerto A como salida
        movlw   b'00111111'     ;Mueve valor literal a  W
        movwf   TRISB           ;RB0 - RB5 como entradas
                                ;RB6 - RB7 como salidas
        bcf     STATUS,RP0      ;Cambio al banco 0
```

Una vez en el banco uno, podemos manipular los registros de configuración de puertos TRISA y TRISB. En este caso todos los pines del puerto A son salidas, llenando de ceros los 8 bits del registro TRISA.

El registro TRISB tiene tanto ceros como unos, los ceros establecen pines de salida y los unos establecen pines de entrada de datos.

3.13.3 Option (Registro de opciones)

RBU	INTDEG	T0CS	T0SE	PSA	PS2	PS1	PS0

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

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

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

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

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

- **PSA2:PSA0: Rango con el que actúa el divisor de frecuencia.**

PS2 - PS0	División TMR0	División WDT
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4

011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

3.13.4 INTCON (Registro de Interrupciones)

GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

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

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

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

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

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

- **T0IF: Indicador de TMR0 se ha desbordado.**
  1 = TMR0 desbordado. Borrar por software. 0 = No se ha desbordado.
- **INTF: Bit de RB0/INT Interrupt Flag.**
  1 = Ocurrió un RB0/INT Interrupt. 0 = No ocurrió un RB0/INT Interrupt.

- **RBIF: Bit de RB Port Change Interrupt Flag.**
  1 = Cuando por lo menos una de las patas de RB7 a RB4 cambió (debe ser limpiado en software). 0 = Ninguna de las patas RB7:RB4 cambió.

3.14 INSTRUCCIONES

3.14.1 Definiciones y abreviaturas

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

Abreviatura	Descripción
PC	Contador de Programa que direcciona la memoria de instrucciones.
TOS	Cima de la pila, con 8 niveles.
WDT	Perro guardián (Watchdog).
W	Registro W, similar al acumulador.
F	Suele ser un campo de 5 bits (fffff) que contiene la dirección del banco de registros, que ocupa el banco 0 del área de datos. Direcciona uno de esos registros.
D	Bit del código OP de la instrucción, que selecciona el destino. Si d=0, el destino es W, y si d=1 el destino es f.
Dest	Destino (registro W o f).
TO	Bit "Time Out" del registro de estado.
PD	Bit "Power Down" del registro de estado.

b	Suele ser un campo de 3 bits (bbb) que determinan la posición de un bit dentro de un registro de 8 bits.
k	Se trata, normalmente, de un campo de 8 bits (kkkkkkkk) que representa un dato inmediato. También puede constar de 9 bits en las instrucciones de salto que cargan al PC.
x	Valor indeterminado (puede ser un 0 o un 1). Para mantener la compatibilidad con las herramientas software de Microchip conviene hacer x = 0.
Label	Nombre de la etiqueta.
[]	Opciones.
()	Contenido.
→	Se asigna a.
<>	Campo de bits de un registro.
∈	Pertenece al conjunto.
Z	Señalizador de cero en W. Pertenece al registro de estado.
C	Señalizador de acarreo en el octavo bit del W. Pertenece al registro de estado.
DC	Señaliza el acarreo en el 4 bit del W. Pertenece al registro de estado.

3.14.2 Repertorio de instrucciones

En los microcontroladores PICs de tipo 16X84 cada instrucción tiene una longitud de 14 bits.

Los bits que actúan como datos de la memoria EPROM se reciben en el decodificador de instrucciones, y operan con el contador de programa y el

registro de trabajo W, para acceder a lugares específicos del microcontrolador, tales como la ALU, posiciones de memoria, registros, etc.

Como se ha comentado, los PICs 16X84, manejan un set reducido de instrucciones (35 instrucciones en lo que se denomina RISC) que presentan una codificación muy particular llamada formato de la instrucción.

Cada instrucción posee su formato y es totalmente definido por MICROCHIP.

Para poder analizar al conjunto de instrucciones que conforman el set RISC, se los suele agrupar teniendo en cuenta el tipo de operación que realizan, as es común que se las presente bajo cuatro posibles formas, a saber:

- Instrucciones orientadas a registros.
- Instrucciones orientadas a bits.
- Instrucciones con literales.
- Instrucciones de control y especiales.

3.14.3 Instrucciones orientadas a registros

ADDWF W + F	**ANDWF W AND F**
Sintaxis: [label] ADDWF f,d	**Sintaxis:** [label] ANDWF f,d
Operandos: $d \in [0,1]$, $0 \leq f \leq 127$	**Operandos:** $d \in [0,1]$, $0 \leq f \leq 127$
Operación: (W) + (f) $\rightarrow$ (dest)	**Operación:** (W) AND (f) $\rightarrow$ (dest)
Flags afectados: C, DC, Z	**Flags afectados:** Z
Código OP: 00 0111 dfff ffff	**Código OP:** 00 0101 dfff ffff
Descripción: Suma el contenido del registro W y el registro f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.	**Descripción:** Realiza la operación lógica AND entre los registros W y f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.
Ejemplo: ADDWF REG,0	**Ejemplo: :** ANDWF REG,0

Antes: W = 0x17., REG = 0xC2 Después: W = 0xD9, REG = 0xC2	Antes: W = 0x17., REG = 0xC2 Después: W = 0x17, REG = 0x02
CLRF Borra un registro **Sintaxis:** [label] CLRF f **Operandos:** $0 \leq f \leq 127$ **Operación: :** 0x00 → (f), 1 → Z **Flags afectados:** Z **Código OP:** 00 0001 1fff ffff **Descripción:** El registro f se carga con 0x00. El flag Z se activa. **Ejemplo: :** CLRF REG Antes: REG = 0x5A Después: REG = 0x00, Z = 1	**CLRW Borra el registro W** **Sintaxis:** [label] CLRW **Operandos:** Ninguno **Operación: :** 0x00 → W, 1 → Z **Flags afectados:** Z **Código OP:** 00 0001 0xxx xxxx **Descripción:** El registro de trabajo W se carga con 0x00. El flag Z se activa. **Ejemplo: :** CLRW Antes: W = 0x5A Después: W = 0x00, Z = 1
COMF Complemento de f **Sintaxis:** [label] COMF f,d **Operandos:** $d \in [0,1]$, $0 \leq f \leq 127$ **Operación: :** (/ f), 1 → (dest) **Flags afectados:** Z **Código OP:** 00 1001 dfff ffff **Descripción:** El registro f es complementado. El flag Z se activa si el resultado es 0. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.. **Ejemplo: :** COMF REG,0 Antes: REG = 0x13 Después: REG = 0x13, W = 0XEC	**DECF Decremento de f** **Sintaxis:** [label] DECF f,d **Operandos:** $d \in [0,1]$, $0 \leq f \leq 127$ **Operación: :** (f) – 1 → (dest) **Flags afectados:** Z **Código OP:** 00 0011 dfff ffff **Descripción:** Decrementa en 1 el contenido de f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. **Ejemplo: :** DECF CONT,1 Antes: CONT = 0x01, Z = 0 Después: CONT = 0x00, Z = 1
DECFSZ Decremento y salto	**INCF Incremento de f**

Sintaxis: [label] DECFSZ f,d

Operandos: d ∈ [0,1], 0 ≤ f ≤ 127

Operación: (f) -1 → d; Salto si R=0

Flags afectados: Ninguno

Código OP: 00 1011 dfff ffff

Descripción: Decrementa el contenido del registro f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Si la resta es 0 salta la siguiente instrucción, en cuyo caso costaría 2 ciclos.

Ejemplo: DECFSC REG,0

GOTO NO_ES_0

SI_ES_0 Instrucción

NO_ES_0 Salta instrucción anterior

Sintaxis: [label] INCF f,d

Operandos: d ∈ [0,1], 0 ≤ f ≤ 127

Operación: : (f) + 1 → (dest)

Flags afectados: Z

Código OP: 00 1010 dfff ffff

Descripción: Incrementa en 1 el contenido de f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.

Ejemplo: : INCF CONT,1

Antes: CONT = 0xFF, Z = 0

Después: CONT = 0x00, Z = 1

INCFSZ Incremento y salto

Sintaxis: [label] INCFSZ f,d

Operandos: d ∈ [0,1], 0 ≤ f ≤ 127

Operación: (f) -1 → d; Salto si R=0

Flags afectados: Ninguno

Código OP: 00 1111 dfff ffff

Descripción: Incrementa el contenido del registro f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Si la resta es 0 salta la siguiente instrucción, en cuyo caso costaría 2 ciclos.

Ejemplo: INCFSC REG,0

GOTO NO_ES_0

SI_ES_0 Instrucción

IORWF W AND F

Sintaxis: [label] IORWF f,d

Operandos: d ∈ [0,1], 0 ≤ f ≤ 127

Operación: (W) OR (f) → (dest)

Flags afectados: Z

Código OP: 00 0100 dfff ffff

Descripción: Realiza la operación lógica OR entre los registros W y f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.

Ejemplo: : IORWF REG,0

Antes: W = 0x91, REG = 0x13

Después: W = 0x93, REG = 0x13

NO_ES_0 Salta instrucción anterior	
MOVF Mover a f **Sintaxis:** [label] MOVF f,d **Operandos:** d ∈ [0,1], 0 ≤ f ≤ 127 **Operación:** (f) → (dest) **Flags afectados:** Z **Código OP:** 00 1000 dfff ffff **Descripción:** El contenido del registro f se mueve al destino d. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Permite verificar el registro, puesto que afecta a Z. **Ejemplo:** MOVF REG,0 Después: W = REG	**MOVWF Mover a f** **Sintaxis:** [label] MOVWF f **Operandos:** 0 ≤ f ≤ 127 **Operación:** W → (f) **Flags afectados:** Ninguno **Código OP:** 00 0000 1fff ffff **Descripción:** El contenido del registro W pasa el registro f. **Ejemplo:** MOVWF REG,0 Antes: REG = 0xFF, W = 0x4F Después: REG = 0x4F, W = 0x4F
NOP No operar **Sintaxis:** [label] NOP **Operandos:** Ninguno **Operación:** No operar **Flags afectados:** Ninguno **Código OP:** 00 0000 0xx0 0000 **Descripción:** No realiza operación alguna. En realidad consume un ciclo de instrucción sin hacer nada. **Ejemplo: :** CLRWDT Después: Contador WDT = 0, Preescales WDT = 0, /TO = 1, /PD = 1	**RLF Rota f a la izquierda** **Sintaxis:** [label] RLF f,d **Operandos:** d ∈ [0,1], 0 ≤ f ≤ 127 **Operación:** Rotación a la izquierda **Flags afectados:** C **Código OP:** 00 1101 dfff ffff **Descripción:** El contenido de f se rota a la izquierda. El bit de menos peso de f pasa al carry (C), y el carry se coloca en el de mayor peso. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. **Ejemplo:** RRF REG,0 Antes: REG = 1110 0110, C = 0

	Después: REG = 1110 0110, W = 1100 1100, C = 1
RRF Rota f a la derecha **Sintaxis:** [label] RRF f,d **Operandos:** d ∈ [0,1], 0 ≤ f ≤ 127 **Operación:** Rotación a la derecha **Flags afectados:** C **Código OP:** 00 1100 dfff ffff **Descripción:** El contenido de f se rota a la derecha. El bit de menos peso de f pasa al carry (C), y el carry se coloca en el de mayor peso. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. **Ejemplo:** RRF REG,0 Antes: REG = 1110 0110, C = 1 Después: REG = 1110 0110, W = 01110 0011, C = 0	**SUBWF Resta f – W** **Sintaxis:** [label] SUBWF f,d **Operandos:** d ∈ [0,1], 0 ≤ f ≤ 127 **Operación:** (f) – (W) → (dest) **Flags afectados:** C, DC, Z **Código OP:** 00 0010 dfff ffff **Descripción:** Mediante el método del complemento a dos el contenido de W es restado al de f. . Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. **Ejemplos:** SUBWF REG,1 Antes: REG = 0x03, W = 0x02, C = ? Después:REG=0x01, W = 0x4F, C=1 Antes: REG = 0x02, W = 0x02, C = ? Después:REG=0x00, W =0x02, C= 1 Antes: REG= 0x01, W= 0x02, C= ? Después:REG=0xFF, W=0x02, C= 0 (Resultado negativo)

3.14.4 Instrucciones orientadas a bits

BCF Borra un bit **Sintaxis:** [label] BCF f,b **Operandos:** 0 ≤ f ≤ 127, 0 ≤ b ≤ 7 **Operación:** : 0 ⇒ (f<b>)	**BSF Activa un bit** **Sintaxis:** [label] BSF f,b **Operandos:** 0 ≤ f ≤ 127, , 0 ≤ b ≤ 7 **Operación:** 1 ⇒ (f<b>)

Flags afectados: Ninguno **Código OP:** 01 00bb bfff ffff **Descripción:** Borra el bit b del registro f **Ejemplo: :** BCF REG,7 Antes: REG = 0xC7 Después: REG = 0x47	**Flags afectados:** Ninguno **Código OP:** 01 01bb bfff ffff **Descripción:** Activa el bit b del registro f **Ejemplo: :** BSF REG,7 Antes: REG = 0x0A Después: REG = 0x8A
BTFSC Test de bit y salto **Sintaxis:** [label] BTFSC f,d **Operandos:** d ∈ [0,1], 0 ≤ f ≤ 127 **Operación:** Salto si (f<b>) = 0 **Flags afectados:** Ninguno **Código OP:** 01 10bb bfff ffff **Descripción:** Si el bit b del registro f es 0, se salta una instrucción y se continúa con la ejecución. En caso de salto, ocupará dos ciclos de reloj. **Ejemplo:** BTFSC REG,6 GOTO NO_ES_0 SI_ES_0 Instrucción NO_ES_0 Instrucción	**BTFSS Test de bit y salto** **Sintaxis:** [label] BTFSS f,d **Operandos:** d ∈ [0,1], 0 ≤ f ≤ 127 **Operación:** Salto si (f<b>) = 1 **Flags afectados:** Ninguno **Código OP:** 01 11bb bfff ffff **Descripción:** Si el bit b del registro f es 1, se salta una instrucción y se continúa con la ejecución. En caso de salto, ocupará dos ciclos de reloj. **Ejemplo:** BTFSS REG,6 GOTO NO_ES_0 SI_ES_0 Instrucción NO_ES_0 Instrucción

3.14.5 Instrucciones con literales y de control

ADDLW Suma un literal **Sintaxis:** [label] ADDLW k **Operandos:** 0 ≤ k ≤ 255	**ANDLW W AND literal** **Sintaxis:** [label] ANDLW k **Operandos:** 0 ≤ k ≤ 255

Operación: : (W) + (k)⇒ (W) **Flags afectados:** C, DC, Z **Código OP:** 11 111x kkkk kkkk **Descripción:** Suma el contenido del registro W y k, guardando el resultado en W. **Ejemplo:** ADDLW 0xC2 Antes: W = 0x17 Después: W = 0xD9	**Operación:** : (W) AND (k)⇒ (W) **Flags afectados:** Z **Código OP:** 11 1001 kkkk kkkk **Descripción:** Realiza la operación lógica AND entre el contenido del registro W y k, guardando el resultado en W. **Ejemplo:** ADDLW 0xC2 Antes: W = 0x17 Después: W = 0xD9
CALL Salto a subrutina **Sintaxis:** [label] CALL k **Operandos:** 0 ≤ k ≤ 2047 **Operación:** PC ⇒ Pila; k ⇒ PC **Flags afectados:** Ninguno **Código OP:** 10 0kkk kkkk kkkk **Descripción:** Salto a una subrutina. La parte baja de k se carga en PCL, y la alta en PCLATCH. Ocupa 2 ciclos de reloj. **Ejemplo:** ORIGEN CALL DESTINO Antes: PC = ORIGEN Después: PC = DESTINO	**CLRWDT Borra el WDT** **Sintaxis:** [label] CLRWDT **Operandos:** Ninguno **Operación:** 0x00 ⇒ WDT, 1 ⇒ /TO 1 ⇒ /PD **Flags afectados:** /TO, /PD **Código OP:** 00 0000 0110 0100 **Descripción:** Esta instrucción borra tanto el WDT como su preescaler. Los bits /TO y /PD del registro de estado se ponen a 1. **Ejemplo:** : CLRWDT Después: Contador WDT = 0, Preescales WDT = 0, /TO = 1, /PD = 1
GOTO Salto incondicional **Sintaxis:** [label] GOTO k **Operandos:** 0 ≤ k ≤ 2047	**IORLW W OR literal** **Sintaxis:** [label] IORLW k **Operandos:** 0 ≤ k ≤ 255

Operación: k ⇒ PC <8:0> **Flags afectados:** Ninguno **Código OP:** 10 1kkk kkkk kkkk **Descripción:** Se trata de un salto incondicional. La parte baja de k se carga en PCL, y la alta en PCLATCH. Ocupa 2 ciclos de reloj. **Ejemplo:** ORIGEN GOTO DESTINO Antes: PC = ORIGEN Después: PC = DESTINO	**Operación: :** (W) OR (k)⇒ (W) **Flags afectados:** Z **Código OP:** 11 1000 kkkk kkkk **Descripción:** Se realiza la operación lógica OR entre el contenido del registro W y k, guardando el resultado en W. **Ejemplo: :** IORLW 0x35 Antes: W = 0x9A Después: W = 0xBF
MOVLW Cargar literal en W **Sintaxis:** [label] MOVLW f **Operandos:** 0 ≤ f ≤ 255 **Operación:** (k) ⇒ (W) **Flags afectados:** Ninguno **Código OP:** 11 00xx kkkk kkkk **Descripción:** El literal k pasa al registro W. **Ejemplo:** MOVLW 0x5A Después: REG = 0x4F, W = 0x5A	**RETFIE Retorno de interrup.** **Sintaxis:** [label] RETFIE **Operandos:** Ninguno **Operación: :** 1 ⇒ GIE; TOS⇒PC **Flags afectados:** Ninguno **Código OP:** 00 0000 0000 1001 **Descripción:** El PC se carga con el contenido de la cima de la pila (TOS): dirección de retorno. Consume 2 ciclos. Las interrupciones vuelven a ser habilitadas. **Ejemplo: :** RETFIE Después: PC = dirección de retorno GIE = 1
RETLW Retorno, carga W **Sintaxis:** [label] RETLW k **Operandos:** 0 ≤ k ≤ 255 **Operación: :** (k)⇒ (W); TOS⇒PC **Flags afectados:** Ninguno	**RETURN Retorno de rutina** **Sintaxis:** [label] RETURN **Operandos:** Ninguno **Operación: :** TOS ⇒ PC **Flags afectados:** Ninguno

Código OP: 11 01xx kkkk kkkk

Descripción: El registro W se carga con la constante k. El PC se carga con el contenido de la cima de la pila (TOS): dirección de retorno.

Consume 2 ciclos.

Ejemplo: : RETLW 0x37

Después: PC = dirección de retorno

W = 0x37

Código OP: 00 0000 0000 1000

Descripción: El PC se carga con el contenido de la cima de la pila (TOS): dirección de retorno.

Consume 2 ciclos.

Ejemplo: : RETURN

Después: PC = dirección de retorno

SLEEP Modo bajo consumo

Sintaxis: [label] SLEEP

Operandos: Ninguno

Operación: 0x00⇒WDT, 1 ⇒ / TO

0 ⇒ WDT Preescaler, 0 ⇒ / PD

Flags afectados: / PD, / TO

Código OP: 00 0000 0110 0011

Descripción: El bit de energía se pone a 0, y a 1 el de descanso. El WDT y su preescaler se borran. El micro para el oscilador, yendo al modo "durmiente".

Ejemplo: : SLEEP

Preescales WDT = 0,

/TO = 1, /PD = 1

SUBLW Resta Literal - W

Sintaxis: [label] SUBLW k

Operandos: $0 \leq k \leq 255$

Operación: (k) - (W) ⇒ (W)

Flags afectados: Z, C, DC

Código OP: 11 110x kkkk kkkk

Descripción: Mediante el método del complemento a dos el contenido de W es restado al literal. El resultado se almacena en W.

Ejemplos: SUBLW 0x02

Antes:W=1,C=?. Después: W=1, C=1

Antes:W=2,C=?. Después: W=0, C=1

Antes:W=3,C=?.Después:W=FF,C=0

(El resultado es negativo)

XORLW W OR literal

Sintaxis: [label] XORLW k

Operandos: $0 \leq k \leq 255$

Operación: : (W) XOR (k)⇒ (W)

Flags afectados: Z

Código OP: 11 1010 kkkk kkkk

Descripción: Se realiza la operación lógica XOR entre el contenido del registro W y k, guardando el resultado en W.

Ejemplo: : XORLW 0xAF

Antes: W = 0xB5

Después: W = 0x1A

3.15 ENCENDIENDO UN LED

Ya que hemos visto como seleccionar bancos de memoria RAM y configurar pines de puertos como entradas o salidas, ahora haremos unas rutinas de programación para encendido de un led y utilizaremos más instrucciones de las que ya se vieron anteriormente.

En el siguiente diagrama se ha tratado de simular un programa de aplicación completa:

Observemos primero el diagrama (véase la Fig. 3.7), tenemos el programa en asembler como se ve en el MPLAB, también con su grupo de registros SFR en el área superior derecha.

En el área inferior derecha encontramos el micro y su arquitectura interna con sus componentes principales RAM, FLASH, CPU y puertos A y B con cada uno de sus pines.

Tenemos que aclarar que lo que esta dentro del recuadro de memoria flash son las direcciones en las que se encuentran nuestro código de operación y no el código mismo

3.15.1 Análisis explicativo

La idea esencial es ver como cambia el flujo de programa al encender el interruptor.

El estado inicial es un nivel lógico alto en el pin RA0 configurado como entrada.

El led se encenderá con un nivel lógico bajo en el pin de entrada RA0 al cual está conectado el interruptor.

La instrucción responsable de que cambie el estado del led de apagado a encendido es BTFSC.

BTFSC: Es otra instrucción orientada a bit
Bit Test file skip if Clear: Explora bit y salta si es 0:

```
btfsc   PORTA, RA0
goto    TEST
bsf     PORTB,RB0
```

Mientras se mantiene un nivel lógico alto en el pin RA0, la siguiente instrucción a ejecutarse será goto.

Cuando se acciona el interruptor y se vuelve a explorar el pin RA0, éste está ahora en nivel lógico bajo, por lo tanto salta a la instrucción goto, y se va a la siguiente que es bsf, siendo ésta la instrucción que encenderá el led.

Luego de encender el led, nos vamos otra vez a TEST, donde se encuentra la instrucción bcf que apaga el led, pero sabemos que la exploración de línea a línea de programa es de 1 microsegundo, y hasta que explore el pin RA0 y encienda nuevamente el led no son mas que 2 microsegundos, imposible que el ojo humano pueda notar que se apaga el led por 2 microsegundos, por lo tanto el efecto final y real es que vemos el led encendido permanentemente.

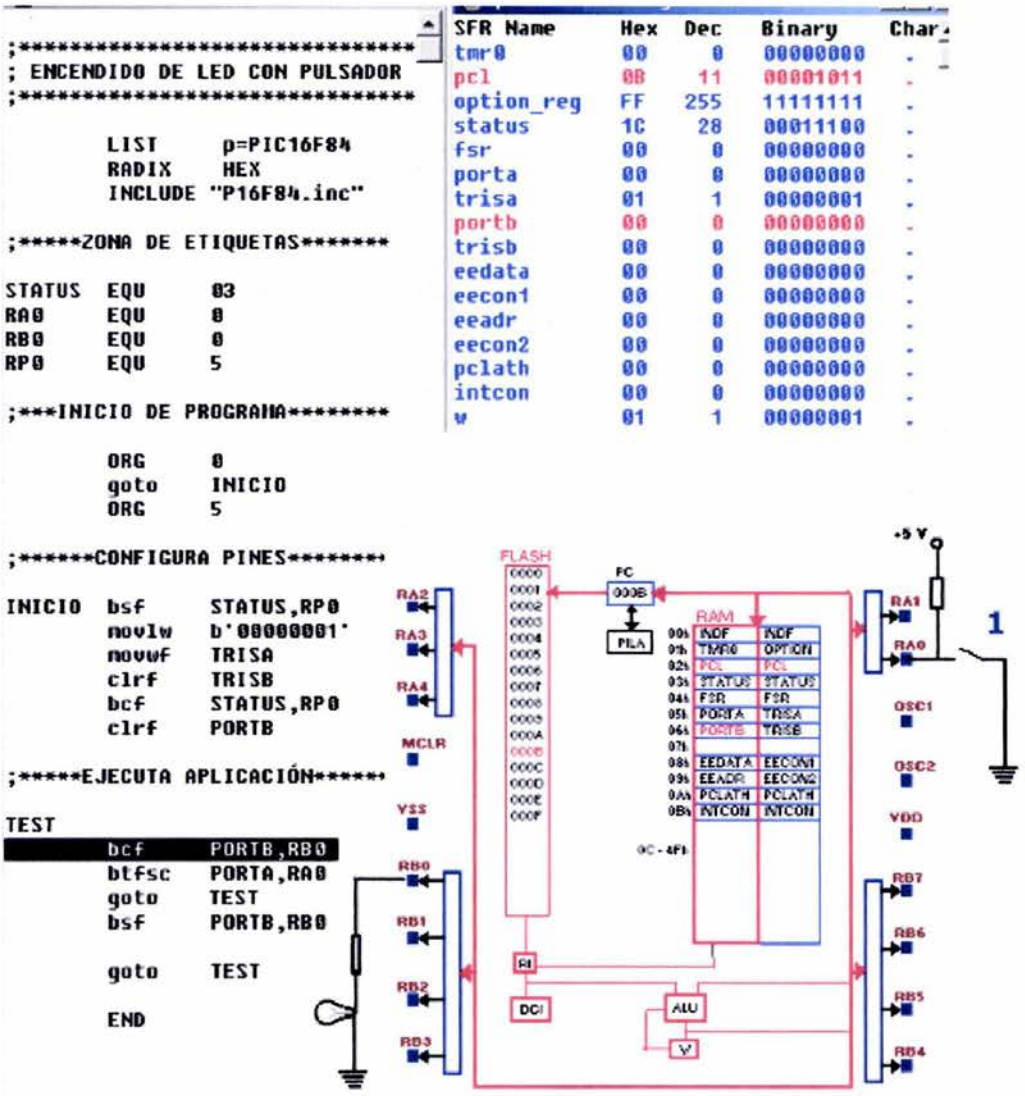


Fig. 3.7 DIAGRAMA DE CÓDIGO, REGISTROS Y ARQUITECTURA

3.16 TEMPORIZANDO

El uso de la temporización se usa en casi todas las aplicaciones, semáforos, alarmas, acceso con clave, ascensores, comunicación, etc.

Recordando nuestro programa de encendido de LED, se menciona que este se apagaba por 2 microsegundos pero que era imposible detectarlo, pues bien en

las siguientes sesiones vamos a temporizar para que el tiempo de apagado sea 1 segundo y de esta manera apreciar la temporización.

Al igual que teníamos dos registros TRISA y TRISB que configuraban los pines de los puertos como entradas o salidas. En la temporización tenemos el registro OPTION en el banco 1, que me configurará el registro TMR0 del banco 0, ya sea como temporizador o contador

La medida de tiempo en un micro, está sujeta a tres temas:

- Oscilador externo
- Registro TMR0
- Divisor de Frecuencia

El oscilador externo, es el que define la velocidad del ciclo de instrucción, y cuando usamos un oscilador de cristal de cuarzo (XT) de 4MHz, tenemos 1 microsegundo de ciclo de instrucción

El registro TMR0 esta implementado físicamente en el pin RA4 del PORTA, este registro de 8 bits se configura como temporizador para determinar intervalos concretos de tiempo o como contador de impulsos externos.

EL divisor de frecuencia, como su nombre lo dice divide la frecuencia de reloj, que es lo mismo decir que aumenta la duración de los impulsos de reloj.

La combinación binaria de los bits PS0, PS1 y PS2, del registro OPTION definen el rango del divisor de frecuencias.

Fórmula para calcular el tiempo

$$T = CI \times TMR0 \times DF$$

CI = Ciclo de instrucción

TMR0 = Valor cargado en este registro.

DF = Rango escogido en el divisor de frecuencia

El valor cargado en el TMR0, es aquel valor que le falta a este registro para que llegue a 255 (valor máximo para un registro de 8 bits)

EJEMPLO:

Si por software cargo un valor de 125 en el registro TMR0, entonces el valor que interviene en la fórmula es 130, valor que falta para que se desborde (llegue a 255) el registro TMR0.

Si ponemos al máximo los valores de la fórmula, rango de divisor de 256, TMR0 con 255 y un oscilador de 4 Mhz, obtenemos un valor máximo de temporización de 65.28 milisegundos, el cual es un valor evidentemente insuficiente si se quiere que el pulso dure varios segundos o minutos, tiempo necesario para la mayoría de aplicaciones, como pueden ser, relojes digitales, semáforos, etc. la solución sencilla es crear un archivo en la RAM y usarlo como archivo auxiliar para lograr la temporización deseada.

Tenemos dos alternativas para alargar tiempos, crear bucles anidados de retardo, en este caso no se usa el registro TMR0, y la otra alternativa es usando este registro.

La ventaja de usar este registro es que se pueden generar condiciones de interrupción, habilitando el tipo de interrupción por desbordamiento de registro TMR0.

La habilitación de este tipo de interrupción, se hace mediante el registro INTCON y sus respectivos bits.

3.16.1 Ejemplo de temporización

En este ejemplo contemplaremos la utilidad del temporizador, y el respectivo señalizador T0IF que se activa por desbordamiento del registro TMR0 (véase la Fig. 3.8).

EL valor que se carga e el registro OPTION, corresponde a la configuración del registro TMR0 como temporizador, un predivisor de frecuencia con un rango de 256 y asignado al TMR0.

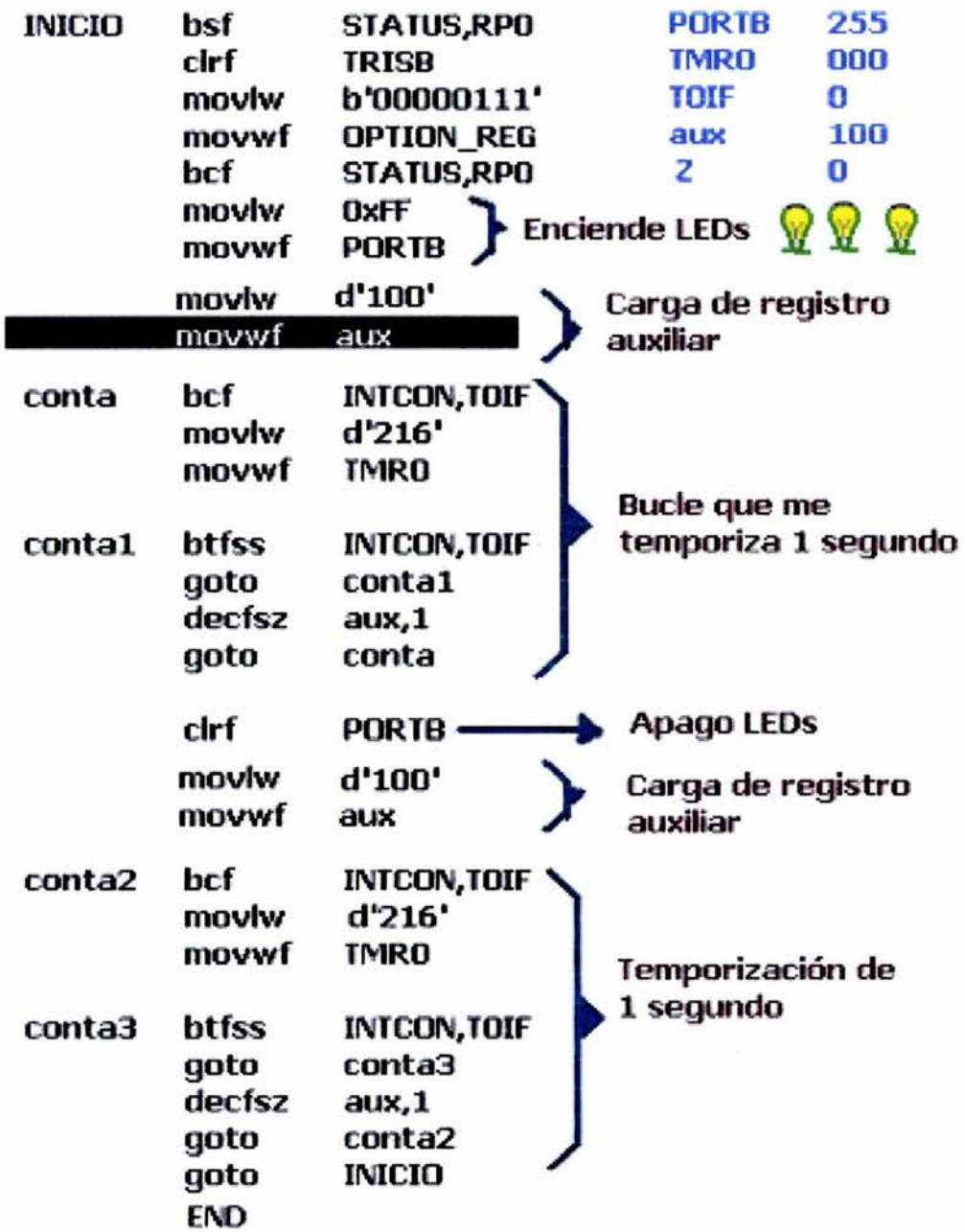


Fig. 3.8 CÓDIGO DEL PROGRAMA EN ASEMBLER

Analicemos

Este ejercicio pretende temporizar un segundo, de tal manera que cada segundo se apaguen y se enciendan leds conectados al puerto B.

En el programa cargamos al registro TMR0 con cualquier valor, en este caso con un valor decimal de 216; entonces en la fórmula de temporización tendremos un valor de 39, que es el valor que le falta el TMR0 para desbordarse (llegar a 255).

Configurado el predivisor con un rango de 256, solamente haría falta un registro auxiliar **aux** cargado con un valor de 100 para alcanzar el segundo.

Comprobando tenemos: 100x39x256 = 0.99seg, aproximadamente 1 segundo.

Cada vez que se desborda el TMR0, se activa el señalizador T0IF (bit 2 del registro INTCON), y explorando la instrucción **btfss** se salta a la instrucción:

decfsz aux,1

Esta instrucción como se menciona anteriormente hace que decremente el registro f y saltar si Z=1. Es decir, decrementa una unidad al registro **aux** y el nuevo valor se deposita en el mismo registro **aux.**

Si fuera:

decfsz aux,0

EL valor decrementado no se depositaría en **aux**, sino en el registro de trabajo W.

Cada 39x256 veces se decrementa en uno el valor de aux

En el preciso instante en que **aux = 0**; se activa el **bit Z** (bit 2 del registro de estado STATUS).

El bit Z se pone a 1 cuando una operación de la ALU es 0.

3.17 INTERRUPCIONES

AL igual que la instrucción call, una interrupción genera un desvío del flujo normal del programa, pero la diferencia sustancial es que al generarse la interrupción, el PC no va a cualquier dirección de la memoria FLASH, sino a la posición 4, que es la posición reservada para la llamada al tratamiento de la interrupción.

3.17.1 ¿Qué es lo que genera interrupciones?

En este tipo de microcontrolador se tienen 4 fuentes de interrupción, es decir son 4 las razones por las que puede ocurrir una interrupción, estas razones son:

1. Desbordamiento del registro TMR0.
2. Cambio de estado en los pines RB4 - RB7 del puerto B
3. Fin de escritura de la EEPROM de datos
4. Por causa de una interrupción externa, pin RB0/INT

Primero tenemos que habilitar el GIE para poder hacer cualquier tipo de interrupción, luego consideramos las interrupciones pertinentes a nuestra aplicación con los correspondientes bits de permiso.

Los bits de señalización son los encargados de avisar que evento de interrupción ha ocurrido poniéndose a 1.

Para volver a ponerlo a 0 estos bits de señalización, debemos programar por software, porque de no ser así no se podrá salir de ellos.

Durante el tiempo que se esta ejecutando la interrupción, GIE se pone a cero para evitar tener en cuenta cualquier otro tipo de interrupción.

Si se quiere retornar de la interrupción con el GIE nuevamente habilitado, se pone la instrucción **RETFIE** que pondrá a GIE automáticamente a 1.

3.17.2 Ejemplo de Interrupción

Vamos a hacer un programa que provoque una interrupción por desbordamiento del registro TMR0 configurado como temporizador y con divisor de frecuencia en un rango de 128.

Se pone a disposición el programa completo listo para ser probado.

Posiblemente se obtenga errores al compilar este código tal como está, así que para que funcione perfectamente se tiene que manipular el archivo p16f84.inc, este archivo se encuentra dentro del MPLAB, sólo se tiene que buscar, abrirlo, y hacerle un par de cambios:

OPTION_REG EQU 0081 por OPTION EQU 0001

TRISB EQU 0086 por TRISB EQU 0006

En este archivo puedes hacer el cambio que quieras, al momento de incluir este archivo en nuestro proyecto, el compilador considerará los cambios realizados.

3.17.3 Código fuente del ejercicio

```
;*********************************************************
;          INTERRUPCIÓN POR DESBORDAMIENTO DEL TMR0
;*********************************************************

      LIST        P=16F84
      INCLUDE     "P16F84a.inc"

;*** Definición de Registros
PORTB  EQU   0x06    ; Dirección puerto B en banco 0
TRISB  EQU   0x06    ; Dirección TRISB en banco 1
STATUS EQU   0x03    ; Dirección de registro de estado
INTCON EQU   0x0B    ; Registro controlador de interrupciones
```

```
OPTION EQU    0x01     ; Registro OPTION
TMR0   EQU    0x01

;*** Definición de Bits
banco  EQU    0x05
Z      EQU    0x02
T0IF   EQU    0x02
T0IE   EQU    0x05

;*** Definición de Constantes
W      EQU    0
F      EQU    1

;*** Definición de Variables
contador EQU  0x0C
;*** Comienzo del Programa
       org    0x00          ;
       goto   INICIO
       org    0x04          ;
       goto   RSI

;*** Configuración de registros

       org        0X05

INICIO bsf    STATUS,banco  ; Seleccionamos el banco 1
       movlw  b'00000110'   ; En binario 0000 0110
       movwf  OPTION        ; Ponemos el predivisor a 128

       ; ****Bits del registro OPTION ****************************
       ; bit8 = 0 Resistencias de polarización deshabilitadas
       ; bit7 = 0 Interrupción externa por flanco bajada (no se usa)
```

```
        ; bit6 = 0 Fuente de reloj es el interno (Usamos TMR0 como
temporizador)
        ; bit5 = 0 Flanco de señal externa (no lo usamos)
        ; bit4 = 0 Divisor asignado al TMR0
        ; bit3 = 1 bit2 = 1 bit1 = 0 División por 128
        ;
**********************************************************

        clrf    TRISB           ; Todo puerto B configurado como salida
        bcf     STATUS,banco    ; Volvemos al banco 0
        clrf    PORTB           ; Borramos todos los LEDS
        clrf    contador                ; Contador = 0
        movlw   0xB2            ; Cargamos el TMR0 con 78 decimal (0xB2)
        movwf   TMR0
        movlw   0xA0            ; 1010 0000 en binario
        movwf   INTCON          ; Habilitamos GIE y T0IE (interrupción del
TMR0)

;****Ejecución del ejemplo
;**** Mira si hay 20 cuentas de 10 ms
;**** Y, si las hay, cambia el LED

BUCLE   movf    contador,W      ; Se incrementa  cada 10 ms en uno al
                                ; producirse la interrupción
        xorlw   0x14            ; Ha llegado a 200 ms si llevamos 20 (0x14)
cuentas
        btfss   STATUS,Z        ; Si es así, salta para cambiar el LED
        goto    BUCLE           ; Si no es así, prueba otra vez
        clrf    contador                ; El contador vuelve a 0 para iniciar
el nuevo ciclo
        btfss   PORTB,1                 ; Está encendido ? Si sí, apaga
        goto    Encien
```

```
Apaga   bcf     PORTB,1             ; Apaga el LED
        goto    BUCLE
Encien  bsf     PORTB,1             ; Enciende el LED
        goto    BUCLE

; **** RSI: Rutina de servicio de interrupción
; **** Salta al desbordarse el TMR0, cada 10 ms
RSI     btfss   INTCON,T0IF         ; Salta si la interrupción es TMR0
        retfie                      ; Interrupción desconocida, regresar
                                    ; (es un mecanismo de seguridad).
        incf    contador,F          ; El contador es incrementado cada 10 ms
        movlw   0xB2                ; Recarga el valor inicial del TMR0
        movwf   TMR0
        bcf     INTCON,T0IF         ; Borra la bandera de interrupción
        bsf     INTCON,T0IE         ; Habilita de nuevo la interrupción
        retfie

        END
```

En la figura 3.9 se muestra la imagen del ejemplo mencionado.

Aplicando la fórmula de temporización nos da aproximadamente 10ms, entonces cada 10ms se desborda el TMR0, momento en el que ocurre una interrupción.

Notamos, pues, que primero se salta al vector de interrupción (posición 0004h de la memoria de programa) y desde acá se va a la Rutina de servicio a la interrupción etiquetado con RSI.

En RSI se incrementa el contador en una unidad, se reestablece valor de T0IF en 0 y se habilita interrupción por desbordamiento de TMR0, luego se retorna con el bit de interrupción general habilitado mediante la instrucción retfie.

Toda esta operación dura hasta que el contador llega al valor de 14H (20 decimal), es decir, después de 20 desbordamientos de TMR0 lo que da unos 2000ms.

Cuando el contador llega a 14H y se retorna de la interrupción, entonces se carga el registro W con este valor, y luego mediante la instrucción **xorlw** se compara con el valor literal 14H. Como son iguales, entonces el resultado de xorlw es 00h y provoca que el bit **Z** del registro de estado se ponga a 1, luego la instrucción **btfss** realiza un salto del bucle al notar que Z=1 y pone a 0 el contador, después de esto hace una exploración del estado de RB1, y enciende el led si se encuentra apagado o apaga el led si se encuentra encendido, el efecto final es un parpadeo del led con un intervalo de tiempo de 200ms.

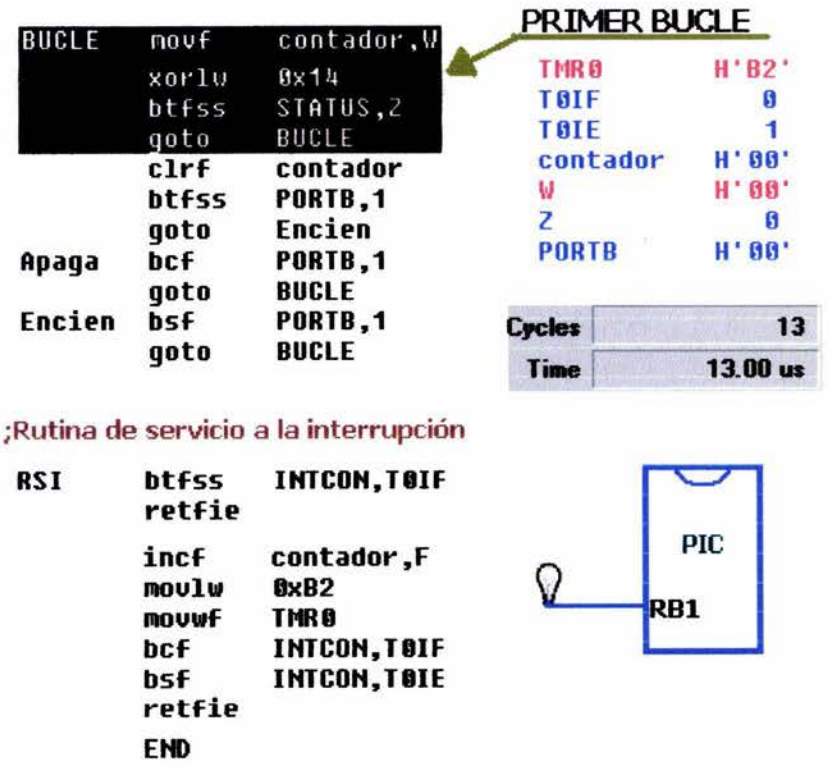
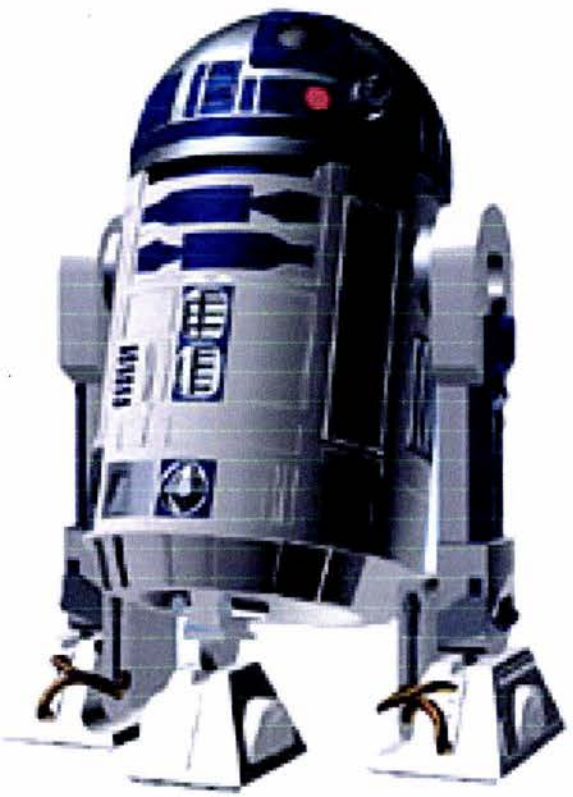


Fig. 3.9 CÓDIGO DEL EJEMPLO

Capítulo 4

Robot autómata seguidor de línea



Contenido:

- **Robot autómata**
- **Seguidor de línea con sensores enconor**
- **Seguidor de línea con el PIC16F84**

4.1 ROBOT AUTÓMATA

El robot móvil autónomo se caracteriza por una conexión *inteligente* entre las operaciones de percepción y acción, que define su comportamiento y le permite llegar a la consecución de los objetivos programados sobre entornos con cierta incertidumbre. El grado de autonomía depende en gran medida de la facultad del robot para abstraer el entorno y convertir la información obtenida en órdenes. De este modo, las dos grandes características que lo alejan de cualquier otro tipo de vehículo se relacionan a continuación:

- Percepción: Determina la relación del robot con su entorno de trabajo, mediante el uso de los sensores de a bordo.

- Razonamiento: Determina las acciones que se han de realizar en cada momento, según el estado del robot y su entorno, para alcanzar las metas asignadas.

De esta forma, la capacidad de razonamiento del robot autónomo móvil se traduce en la planificación de unas trayectorias seguras que le permitan la consecución de los objetivos encomendados. La ejecución de la tarea debe realizarla en bucle cerrado para adaptarse a la navegación sobre entornos no estructurados.

4.2 Seguidor de línea con sensores "enconor"

4.2.1 Sensores de Línea Negra "enconor"

La función básica de este sensor es la de adquirir una información mediante la emisión y recepción de un rayo luminoso que sale de cada una de las dos células que lo componen. El valor de dicha información dependerá del color con el que choque el rayo emisor.

Aprovechando la discriminación de colores que nos detecta este sensor y mediante la utilización de la controladora ENCONOR PLUS o ENCONOR 2 podemos desarrollar diversos proyectos en el aula de Tecnología.

4.2.2 Características del seguidor de línea enconor

- Tiene un tamaño muy reducido el sensor para poder instalarlo en cualquier móvil.
- El seguidor de línea negra ENCONOR se conecta a dos entradas analógicas de la controladora ENCONOR PLUS o ENCONOR 2 (por ejemplo EA1 y EA2).
- Su alimentación es de 5 voltios en continua (exterior o a través de la propia controladora).
- La línea que determina la información emitida por las células, puede ser negra en un fondo claro o clara en un fondo negro.
- La anchura de la línea determinará que el zig-zag que describe el móvil en su desplazamiento sea más o menos pronunciado. Está diseñado para una anchura óptima de línea correspondiente a la de una cinta aislante negra.
- Los valores de discriminación entre los colores claros y negros, puede llegar a ser hasta de 220 unidades si el negro es mate y de 175 si éste es con brillo.

4.2.3 Proyectos tipo

Los tipos de proyectos que se pueden realizar con este sensor, son todos aquellos en los cuales un móvil se desplace y la línea marque la trayectoria o los límites de su recorrido:

- Móvil que siga la trayectoria de una línea de cualquier tipo.
- Móvil que se mueva sin salirse de los límites marcados por una línea cerrada.
- Móvil que busque la zona de suelo de color oscuro.

4.2.4 Forma de uso

El sensor se colocará en la parte inferior del móvil, con el lado de los componentes hacia el suelo. La distancia hasta el suelo no es crítica, pero funciona mejor con distancias comprendidas entre 5 y 20 mm.

Para discriminar si se está dentro o fuera de la línea negra, se leerá el valor de las entradas analógicas a las que se ha colocado el sensor estando sobre el color negro, y fuera de él.

De esta forma se puede crear un programa que actúe en consecuencia teniendo en cuenta estos valores.

Por ejemplo, si sobre el color negro la lectura es de 75 y fuera de él es de 135, se podría hacer el siguiente programa:

Se colocará el sensor en un móvil con dos motores conectados a la salida digital 1 y 2 respectivamente. Si se conectan los dos motores el móvil se mueve en línea recta. Si se conecta solamente el derecho, el móvil gira a la izquierda. Si se conecta solamente el izquierdo, el móvil girará hacia la derecha.

De las dos salidas del detector de línea negra, la del sensor derecho se conectará a la entrada analógica 1 y la del sensor izquierdo se conectará a la entrada analógica 2.

El algoritmo será: Se pondrán en marcha los dos motores de un móvil. Cuando se detecte que se ha salido de la línea por la derecha, se parará el motor izquierdo, hasta detectar que se encuentra nuevamente en la línea. Cuando se detecte que se ha salido de la línea por la izquierda, se parará el motor derecho, hasta detectar que se encuentra nuevamente en la línea.

Diagrama del seguidor de línea con sensor "enconor"

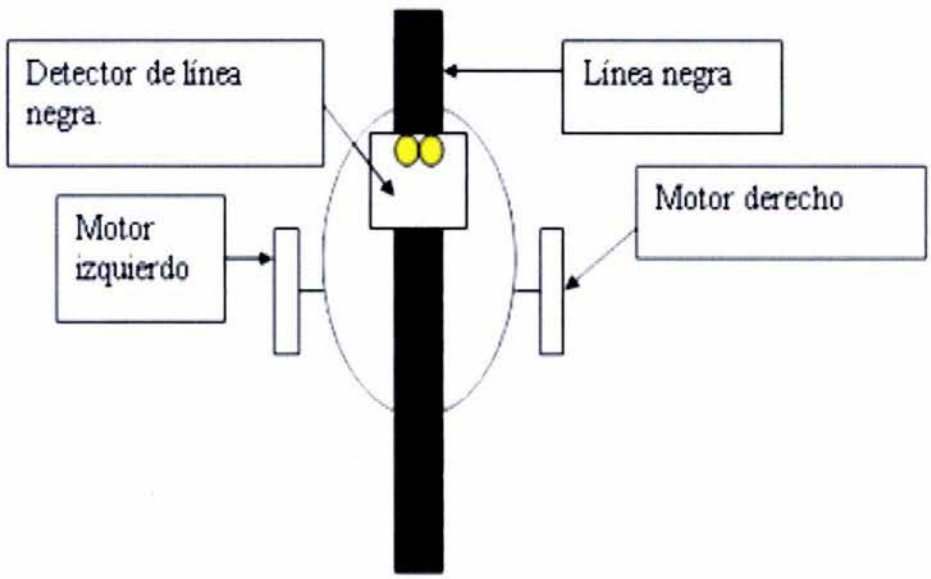


Fig. 4.1[15]

Para móvil
Conecta 1
Conecta 2
Siguelinea
Fin

[15] www.enconor.com

Para siguelinea

Sisino (leeanalogica 1 >100) [apaga 1][conecta 1]

Sisino (leeanalogica 2 >100) [apaga 2][conecta 2]

Siguelinea

fin

4.2.5 Conexión a la controladora "enconor plus"

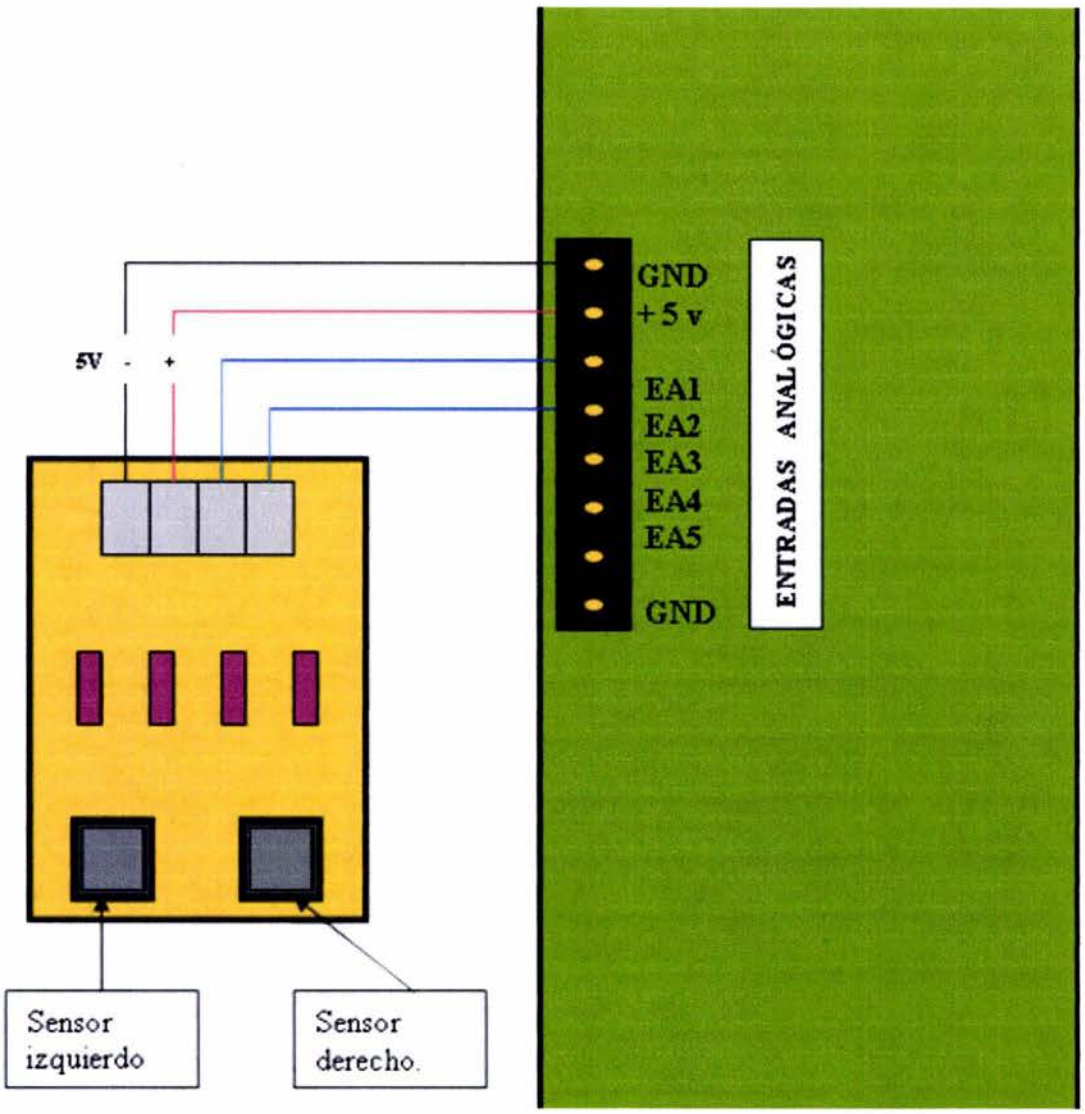


Fig. 4.2[16]

[16] IDEM

4.3 SEGUIDOR DE LÍNEA CON EL PIC16F84A

Es un robot que utiliza dos sensores, los cuales le sirven para ir siguiendo una línea blanca trazada en el piso. Estos sensores envían una señal al PIC16F84A (el cual actúa como cerebro de la aplicación), que lo guiaran para seguir una línea blanca.

Este robot seguidor de línea consta de una parte de software, la cual es el programa que con la que interpretara las señales emitidas por los sensores, y las transforma en ordenes a los motores; una parte mecánica-electrónica, la cual consta de dos motores y una rueda central libre; y una parte sensorial que será la que se encargara de enviar la información al PIC de la línea blanca (véase Fig. 4.3).

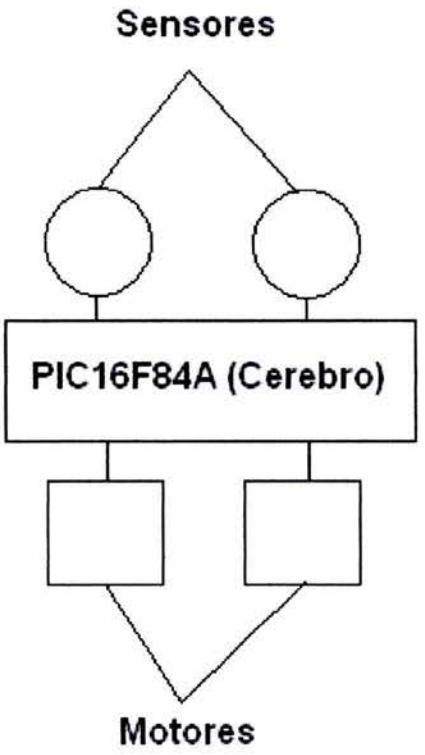


Fig. 4.3[17] DIAGRAMA DE PARTES DEL SEGUIDOR DE LÍNEA

[17] Imagen hecha por el autor

4.3.1 programa del seguidor de línea con el PIC16F84A

Para la explicación de la parte de software, se dividirá el programa en pequeños bloque para así poder explicar este parte por parte y se tenga una mejor visión de lo que se esta planteando.

Este primer bloque del programa es meramente informativo, el cual se usa para explicarle al lector de que forma están distribuidas la terminales del PIC, así como puede ver si son entradas (in) o salidas (out) y su función.

```
;*****************************************************************
*
;  PIC16F84A MAPA I/O                                            *
;*****************************************************************
*
;RA0 IN    SENSOR IZQUIERDO                                      *
;RA1 N/U                                                         *
;RA2 IN    SENSOR DERECHO                                        *
;RA3 OUT     LED                                                 *
;RA4 IN    SWITCH                                               *
;RBO N/U                                                         *
;RB1 N/U                                                         *
;RB2 N/U                                                         *
;RB3 N/U                                                         *
;RB4 OUT     MOTOR IZQUIERDO IN1                                 *
;RB5 OUT     MOTOR IZQUIERDO IN2                                 *
;RB6 OUT     MOTOR DERECHO IN1                                   *
```

```
;RB7 OUT     MOTOR DERECHO IN2                                              *
;                                                                          *
;***************************************************************************
```

Este bloque se encarga de la declaración de los registros que se utilizarán a través de todo el programa, ya que para la programación es mas sencillo y mas fácil de trabajar con etiquetas que con direcciones.

```
;***************************************************************************
   LIST P=16F84A
   #INCLUDE <P16F84A.INC>
;
;***************************************************************************
;
STATUS     EQU H'03'
PORTA      EQU H'05'
PORTB      EQU H'06'
SENSOR_IZQ     EQU H'00'
SENSOR_DER EQU H'02'
LED        EQU H'03'
SWITCH     EQU H'04'
LED_COUNTER    EQU H'10'
COUNTER1 EQU H'11'
COUNTER2 EQU H'12'
COUNTER3 EQU H'13'
PORTA_STATE    EQU H'16'
;
        ORG 0X00
        GOTO INICIALIZA
;
;***************************************************************************
```

En este bloque se encarga la configuración de los puertos, es decir se especifica si los puertos van a ser de entrada o de salida, para hacer esto primero se tiene que pasar al banco1 y configurar los puertos, ya una vez hecho esto se tiene que regresar al banco0 para empezar con la programación.

```
;*****************************************************************
INICIALIZA
;
   BSF STATUS,5      ;PASO AL BANCO1 PARA CONFIGURAR LOS PURTOS
   MOVLW 0X17        ;W <= B'10111' CARGO EL VALOR 10111 EN W PARA DESPUES
                          CONFIGURAR EL PUERTO
   MOVWF PORTA         ;RA4, RA2,RA1,RA0 ENTRADAS; RA3 SALIDA
   MOVLW 0X00        ;CARGO B'00000000' EN W PARA CONFIGURAR EL PUERTO B
   MOVWF PORTB         ;RB7:RB0 SALIDAS
   BCF STATUS,5      ;REGRESO AL BANCO0
;
;*****************************************************************
```

El bloque de rutina principal espera hasta que el bit de switch se active para poder continuar con la ejecución del programa.

```
;*****************************************************************
; RUTINA PRICIPAL
;
MAIN
;
   BCF PORTA,LED          ;LIMPIA RA3
   CLRF PORTB             ; LIMPIA TODO EL PORTB
;
   BTFSS PORTA,SWITCH     ;SALTA A CALL LED_FLICKER SI PORTA(4)=1
   GOTO MAIN              ; GOTO MAIN SI PORTA (4)=0
```

```
;
   CALL LED_FLICKER    ; LLAMA A LA SUBRUTINA LED_FLICKER
;
;
OPERACION
;
   MOVF PORTA,W     ;CARGA EN W LO QUE ESTA LEYENDO EL PORTA
   MOVWF PORTA_STATE   ;CARGA EN PORTA_STATE EL CONTENIDO DE W
;
;*****************************************************************
```

En el bloque de máscara del puerto A se crea una mascara con el numero 05 en hexadecimal, el cual en binario es 00000101, los unos indican el bit en el cual se encuentra instalado el sensor, para que una vez que el puertoA lea la información de los dos sensores haga una operación AND y posteriormente el programa ejecute la subrutina correspondiente a la señales que están mandando los sensores.

```
;********MASCARA DE PORA*************************************************
;
   MOVLW 0X05        ;
   ANDWF PORTA_STATE,1 ;
;
;*****************************************************************
```

El bloque de sensor izquierdo coteja el dato que viene de la mascara que hicimos y si corresponde a este manda a llamar a la subrutina IZQ_FRENTE y sino pasa al siguiente bloque.

```
;*******CHECA SENSOR IZQUIERDO************************************
;
   MOVLW 0X01              ; Carga W con un 1
   SUBWF PORTA_STATE,W ; Resta el contenido de de W a el contenido de
                           PORT_STATE y lo guarda en PORT_STATE
   BTFSC STATUS,2          ; checa la bandera de cero(Z) del registro de estado y
   CALL  IZQ_FRENTE        ; llama a IZQ_FRENTE si es 1 y salta si es 0
;
;*****************************************************************
```

En este bloque de checa sensor derecho hace algo muy similar al de checa sensor izquierdo, la única diferencia es que si al ejecutarse este bloque se manda a llamar a la subrutina de DER_FRENTE.

```
;*******CHECA SENSOR DERCHO*****************************************
;
   MOVLW 0X04              ; Carga W con un 04
   SUBWF PORTA_STATE,W ; Resta el contenido de de W a el contenido de
                           PORT_STATE y lo guarda en PORT_STATE
   BTFSC STATUS,2          ; checa la bandera de cero(Z) del registro de estado y
   CALL  DER_FRENTE        ; llama a DER_FRENTE    si es 1 y salta si es 0
;
;*****************************************************************
```

Este bloque de checa sensor izquierdo y derecho comprueba los 2 sensores simultáneamente por medio de la mascara que se hizo al principio del programa y si se activa éste manda a llamar a la subrutina de reversa para que así el carro camine hacia atrás y trate de encontrar de nuevo la línea.

```
;*******CHECA SENSOR IZQUIERDO Y DERECHO**************************
;
   MOVLW 0X05              ; Carga W con un 05
   SUBWF PORTA_STATE,W ; Resta el contenido de de W a el contenido de
                           PORT_STATE y lo guarda en PORT_STATE
   BTFSC STATUS,2          ; checa la bandera de cero(Z) del registro de estado y
   CALL  REVERSA           ; llama a REVERSA  si es 1 y salta si es 0
;
;*****************************************************************
```

En el bloque de memoria comprueba si no hay señal en los sensores, y de ser así quiere decir que el robot se encuentra dentro de la línea y mandará a llamar a la rutina FORWARD para que el robot siga hacia delante. Una vez ejecutada esta operación se manda el PC[18] a la etiqueta OPERACIÓN para que así se vuelva ejecutar todo el programa y el robot siga su camino.

```
;*******ACCION DE MEMORIA*****************************************
;
   MOVLW 0X00              ; Carga W con un 00
   SUBWF PORTA_STATE,W ; Resta el contenido de de W a el contenido de
                           PORT_STATE y lo guarda en PORT_STATE
   BTFSC STATUS,2          ; checa la bandera de cero(Z) del registro de estado y
   CALL  FORWARD           ; llama a FORWARD si es 1 y salta si es 0
;
;
   GOTO OPERACION          ;
;
;*****************************************************************
```

Este bloque de subrutina FORWARD es el que manda la señal a los motores para que sigan adelante.

[18] Contador de Programa

```
;******SUBRUTINA FORWARD*****************************************
;
FORWARD
;
   MOVLW 0XA0          ; carga W con 10100000 en binario
   MOVWF PORTB         ; el contenido de W lo manda al PuertoB
;
   RETURN
;
;*******************************************************************
```

Este bloque de subrutina REVERSA, como su nombre lo indica manda la señal a los motores para que giren hacia atrás.

```
;******SUBRUTINA REVERSA*****************************************
;
REVERSA
;
   MOVLW 0X50          ; carga W con 01010000 en binario
   MOVWF PORTB         ; el contenido de W lo manda al PuertoB
;
   RETURN
;
;*******************************************************************
```

El bloque de subrutina DER-FRENTE hace que el motor izquierdo camine hacia delante y el motor derecho se frene, para ocasionar que el robot gire hacia la derecha.

```
;******SUBRUTINA DER_FRENTE**************************************
;
DER_FRENTE
;
```

```
   MOVLW 0XE0          ; carga W con 11100000 en binario
   MOVWF PORTB         ; el contenido de W lo manda al PuertoB
;
   RETURN
;
;*********************************************************************
```

El bloque de subrutina IZQ-FRENTE hace que el motor izquierdo se frene y el motor derecho camine hacia delante, para ocasionar que el robot gire hacia la izquierda.

```
;******SUBRUTINA IZQ_FRENTE*****************************************
;
IZQ_FRENTE
;
   MOVLW 0XB0          ; carga W con 10110000 en binario
   MOVWF PORTB         ; el contenido de W lo manda al PuertoB
;
   RETURN
;
;*********************************************************************
```

El bloque de la subrutina LED_FLICKER ocasiona un parpadeo de un led cuando de activa el switch situado en RA4.

```
;******SUBRUTINA LED_FLICKER****************************************
;
;
LED_FLICKER
;
   MOVLW D'005'            ; carga el registro W con 5 decimal
   MOVWF LED_COUNTER   ; carga el contador led_counter con 5
;
```

```
LED_LOOP
;
   BSF PORTA,LED          ;  prende el led conectado a RA3
   CALL TIMER3            ;  manda a llamar un tiempo
   BCF PORTA,LED          ;  apaga el led conectado a RA3
   CALL TIMER3            ;  manda a llamar un tiempo
   DECFSZ LED_COUNTER     ;  Decrementa el contador led_counter y repite led_loop
   GOTO LED_LOOP          ;  hasta que el contador sea 0
;
   RETURN
;
;*****************************************************************
```

El bloque de timer1 es una subrutina la cual se encarga de decrementar un contador y hacer unos ciclos de más para ocasionar un tiempo, esta subrutina es llamada por otra subrutina de tiempo la cual es tiempo2.

```
;******SUBRUTINA TIMER1*********************************************
;
TIMER1
;
   MOVLW D'48'            ; carga el registro W con 48 decimal
   MOVWF COUNTER1         ; carga el contador counter1 con W
;
DELAY1
;
   GOTO $+1               ; hace un tiempo de (2ciclos*49)
   DECFSZ COUNTER1,1      ; decrementa counter1 y salta cuando este sea 0
   GOTO DELAY1            ; va a la etiqueta delay1
   GOTO $+1               ; un tiempo de 2ciclos
;
   RETURN
;
;*****************************************************************
```

El bloque de timer2 es una subrutina la cual se encarga de decrementar un contador para ocasionar un tiempo, esta subrutina es llamada por otra subrutina de tiempo, la cual es tiempo3, y esta a su vez manda a llamar la subrutina de tiempo timer1.

```
;******SUBRUTINA TIMER2************************************************
;
TIMER2
;
   MOVLW d'100'          ; carga el registro W con 100 decimal
   MOVWF COUNTER2        ; el contador counter2 se carga con W
;
DELAY2
;
   CALL TIMER1           ; manda a llamar a la subrutina timer1
   DECFSZ COUNTER2,1 ; decrementa el contador counter2 y salta si es 0
   GOTO DELAY2           ; va a la etiqueta delay2
;
   RETURN
;
;*********************************************************************
```

El bloque de subrutina timer3 es una subrutina, la cual se encarga de decrementar un contador para ocasionar un tiempo, y esta a su vez manda a llamar la subrutina de tiempo timer2.

```
;******SUBRUTINA TIMER3************************************************
;
;
TIMER3
;
   movlw d'100'          ; el registro W con 100 decimal
   movwf COUNTER3        ; carga el contador counter3 con W
;
```

```
DELAY3
;
   CALL TIMER2          ; manda a llamar a la subrutina timer2
   DECFSZ COUNTER3,1 ; decrementa el contador counter3 y salta si es 0
   GOTO DELAY3          ; va a la etiqueta delay3
;
   RETURN
;*****************************************************************************
   Este bloque indica el fin del programa.
;*****************************************************************************
   END                  ; fin del programa
;*****************************************************************************
```

4.3.2 Parte mecánica-electrónica del seguidor de línea

En la parte mecánica, se va a utilizar dos motores independientes conectados a una llanta cada uno por medio de un engranaje (véase Fig. 4.4), estos dos van a estar instalados en la parte trasera del robot, y en la parte delantera estará una rueda libre, para que se mueva esta según manden los dos motores traseros, los cuales son los que reciben la señal emitida por el PIC para ordenar en que dirección se moverá el robot.

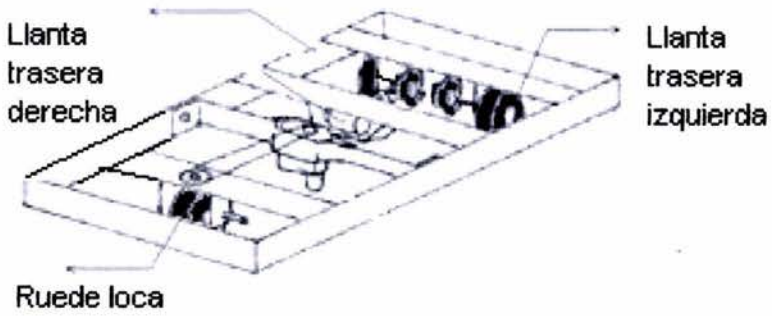


Fig. 4.4[19] CHASÍS DEL SEGUIDOR DE LÍNEA

[19] Imagen modificada por el autor, extraída de imágenes de google

El robot también consta de una parte electrónica, la cual se utiliza para que interprete la señal mandada por el PIC16F84A y la transforme en voltaje que direccionará a los motores, esto es gracias a un integrado de toshiba TA7257P. Este cuenta con 2 entradas y dos salidas, las entradas es la combinación de 1 y 0 emitidas por el PIC, y las salidas son voltaje o tierra que van conectados a los motores para su funcionamiento (véase Fig. 4.5).

La tabla de verdad de este Circuito integrado es la siguiente:

IN1	IN2	OUT1	OUT2
1	1	L	L
0	1	L	H
1	0	H	L
0	0	H	H

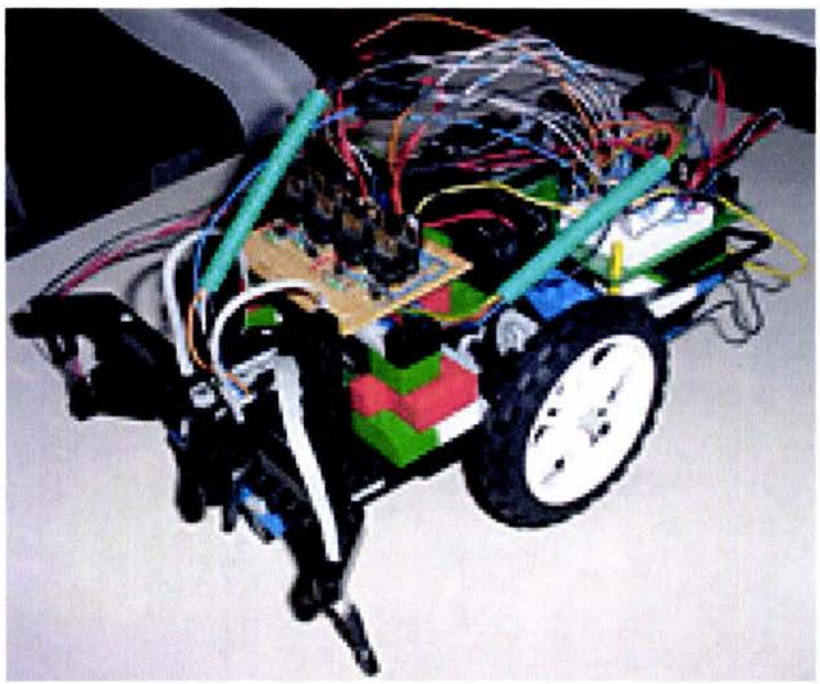


Fig. 4.5[20] ROBOT SEGUIDOR DE LÍNEA

[20] Figura extraída de imágenes de google

4.3.3 Parte sensorial del seguidor de línea

Un sensor es un detector que reacciona a determinados estímulos físicos (calor o frío, luz, velocidad, etc.) y los transforma en información apta para ser transmitida[21].

Los sensores que va a utilizar el seguidor de línea van a estar situados en la parte frontal del robot y muy cercanos al piso para que así puedan mandar una señal correcta de lo que están captando.

El sensor que se va a utilizar (véase Fig. 4.6) es el de tipo reflectivo donde el emisor de luz y el receptor están colocados en la misma dirección para detectar el cambio de reflexión de la luz utilizando la reflexión del infrarrojo sobre el suelo.

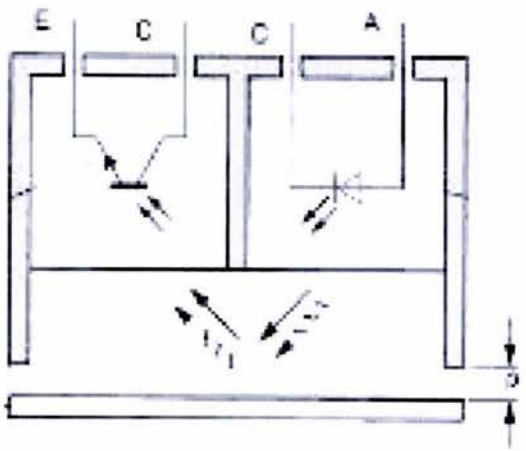


Fig.4.6 DIAGRAMA DE SENSOR REFLECTIVO

[21] Definición dada en enciclonet.com

Conclusión

Los PIC16F84 son unos microcontroladores hechos por microchip, estos son unos dispositivos, con los cuales se pueden desarrollar una gran variedad de aplicaciones, dichas aplicaciones son mas enfocadas a las máquinas, ya que los microcontroladores en general son mas enfocados a la interacción con las máquinas, porque su estructura de tipo lazo cerrado, esto quiere decir que internamente cuentan con memoria RAM, registros, etc. y no se les pude conectar mas dispositivos de este tipo a los microcontroladores.

El objetivo que se había fijado para este trabajo de dar a conocer las generalidades de los Microcontroladores PIC, su funcionamiento, y la programación de los mismos, refiriéndose en particular el PIC16F84A del cual se propuso ejemplos y un caso práctico de su uso en un Robot Autómata Seguidor de línea. Fue un éxito ya que aprendimos que existe una gran variedad de microcontroladores, de diferentes marcas y sobre todo aprendimos cómo seleccionar el microcontrolador más adecuado para la aplicación que se quiera realizar.

Se aprendió a utilizar y a comprender el microcontrolador PIC16F84A desde su estructura, la cual consta desde su tipo de memoria, familia lógica a la que pertenece, entre otras cosas. Y además identificar cuales son sus terminales para utilizarlo correctamente.

Una de las cosas más importantes de los microcontroladores PIC es saber cómo se va a programar el micro y qué herramientas utilizar para hacerlo, ya sea software o hardware. Tomando en cuenta la interacción que debe de haber para poder pasar del programa fuente al microcontrolador PIC. Una vez que se tiene el conocimiento de cómo utilizar estas herramientas, es necesario saber utilizar los diferentes registros y las instrucciones para poder así poder realizar una aplicación lo mejor posible.

Además se aprendió por medio de un ejemplo cómo conjuntar diferentes tipos de elementos como los cuales son mecánicos y eléctricos.

La factibilidad de este trabajo es muy alta, ya que el material para la realización de este proyecto no es muy caro en comparación de comprar una computadora, el único inconveniente es que para poder realizar este tipo de proyectos con microcontroladores, es necesario tener conocimientos previos de electrónica y de lenguaje ensamblador.

El manejo de este tipo de herramientas nos serán muy útiles en el área de desarrollo ya que estos PIC'S son muy versátiles y muy cómodos de manejar por su reducido tamaño, pero esto no involucra que son de poca capacidad, sino todo lo contrario, existen diversas gamas PIC'S que seguramente hay una, la cual se ajuste a nuestras necesidades.

BIBLIOGRAFÍA

ANGULO USATEGUI, José M.ª, Angulo Martínez, Ignacio. ***Microcontroladores PIC. Diseño práctico de aplicaciones,*** 2da ed., editorial Mac Graw Hill, México, 1999, 289 p.p.

KENNETH HINTZ, Daniel Tabak, ***Microcontrollers,*** editorial Mc Graw Gill, Estados Unidos Americanos, 2000, 483 p.p.

J. CRAIG John, ***Introduction to Robotics,*** 2da ed., Editorial Addison_Wesley Publishing Company, Estados Unidos Americanos, 1990.450 p.p.

Otras Fuentes

http://scmstore.com/acceso/sensores/altaCalidad/index.htm

http://www.microchip.com/

http://www.rentron.com/pic.htm

http://zeus.uam.mx/labre/microcontroladores.htm

http://www.frino.com.ar/micros.htm

http://www.enconor.com

http://www.google.com

http://www.micro1.com

http://www.paralax.com

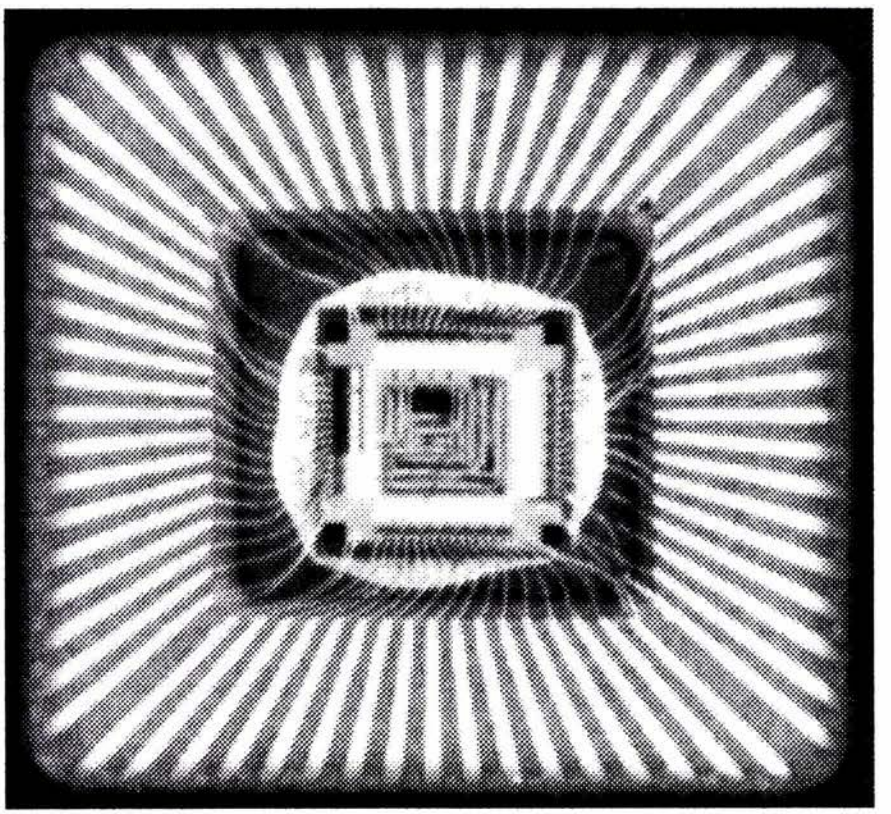
Foros de discusión de microcontroladores encontrados por google.

Manual PDF ***what's a microcontroller?***

Manual PDF ***PIC16F84A*** de Microchip.

Manual PDF ***Planificación de robots móviles.***

ANEXOS



MICROCHIP

PIC16F8X

8-Bit CMOS Flash/EEPROM Microcontrollers

Devices Included in this Data Sheet:

- PIC16F83
- PIC16CR83
- PIC16F84
- PIC16CR84
- Extended voltage range devices available (PIC16**LF**8X, PIC16**LCR**8X)

High Performance RISC CPU Features:

- Only 35 single word instructions to learn
- All instructions single cycle (400 ns @ 10 MHz) except for program branches which are two-cycle
- Operating speed: DC - 10 MHz clock input
  DC - 400 ns instruction cycle

Device	Memory			Freq Max.
	Flash	Data RAM	Data EEPROM	
PIC16F83	512 words	36	64	10 MHz
PIC16CR83	512 words	36	64	10 MHz
PIC16F84	1 K-words	68	64	10 MHz
PIC16CR84	1 K-words	68	64	10 MHz

F = Flash; CR = ROM

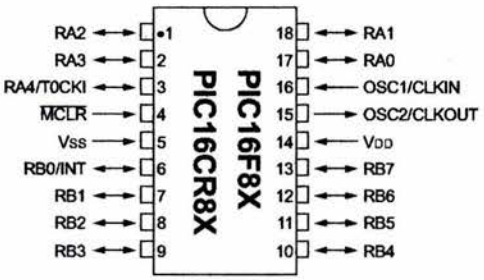
- 14-bit wide instructions
- 8-bit wide data path
- 15 special function hardware registers
- Eight-level deep hardware stack
- Direct, indirect and relative addressing modes
- Four interrupt sources:
  - External RB0/INT pin
  - TMR0 timer overflow
  - PORTB<7:4> interrupt on change
  - Data EEPROM write complete
- 1,000,000 data memory EEPROM ERASE/WRITE cycles
- EEPROM Data Retention > 40 years

Peripheral Features:

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

Pin Diagram

PDIP, SOIC



Special Microcontroller Features:

- Power-on Reset (POR)
- Power-up Timer (PWRT)
- Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Code-protection
- Power saving SLEEP mode
- Selectable oscillator options
- Serial In-System Programming - via two pins (ROM devices support only Data EEPROM programming)

CMOS Technology:

- Low-power, high-speed CMOS Flash/EEPROM technology
- Fully static design
- Wide operating voltage range:
  - Commercial: 2.0V to 6.0V
  - Industrial: 2.0V to 6.0V
- Low power consumption:
  - < 2 mA typical @ 5V, 4 MHz
  - 15 µA typical @ 2V, 32 kHz
  - < 1 µA typical standby current @ 2V

1.0 GENERAL DESCRIPTION

The PIC16F8X is a group in the PIC16CXX family of low-cost, high-performance, CMOS, fully-static, 8-bit microcontrollers. This group contains the following devices:

- PIC16F83
- PIC16CR83
- PIC16F84
- PIC16CR84

All PIC16/17 microcontrollers employ an advanced RISC architecture. PIC16CXX devices have enhanced core features, eight-level deep stack, and multiple internal and external interrupt sources. The separate instruction and data buses of the Harvard architecture allow a 14-bit wide instruction word with a separate 8-bit wide data bus. The two stage instruction pipeline allows all instructions to execute in a single cycle, except for program branches (which require two cycles). A total of 35 instructions (reduced instruction set) are available. Additionally, a large register set is used to achieve a very high performance level.

PIC16F8X microcontrollers typically achieve a 2:1 code compression and up to a 2:1 speed improvement (at 10 MHz) over other 8-bit microcontrollers in their class.

The PIC16F8X has up to 68 bytes of RAM, 64 bytes of Data EEPROM memory, and 13 I/O pins. A timer/counter is also available.

The PIC16CXX family has special features to reduce external components, thus reducing cost, enhancing system reliability and reducing power consumption. There are four oscillator options, of which the single pin RC oscillator provides a low-cost solution, the LP oscillator minimizes power consumption, XT is a standard crystal, and the HS is for High Speed crystals. The SLEEP (power-down) mode offers power saving. The user can wake the chip from sleep through several external and internal interrupts and resets.

A highly reliable Watchdog Timer with its own on-chip RC oscillator provides protection against software lock-up.

The devices with Flash program memory allow the same device package to be used for prototyping and production. In-circuit reprogrammability allows the code to be updated without the device being removed from the end application. This is useful in the development of many applications where the device may not be easily accessible, but the prototypes may require code updates. This is also useful for remote applications where the code may need to be updated (such as rate information).

Table 1-1 lists the features of the PIC16F8X, and Appendix D: list the features of all of the Microchip microcontrollers.

A simplified block diagram of the PIC16F8X is shown in Figure 3-1.

The PIC16F8X fits perfectly in applications ranging from high speed automotive and appliance motor control to low-power remote sensors, electronic locks, security devices and smart cards. The Flash/EEPROM technology makes customization of application programs (transmitter codes, motor speeds, receiver frequencies, security codes, etc.) extremely fast and convenient. The small footprint packages make this microcontroller series perfect for all applications with space limitations. Low-cost, low-power, high performance, ease of use and I/O flexibility make the PIC16F8X very versatile even in areas where no microcontroller use has been considered before (e.g., timer functions, serial communication, capture and compare, PWM functions and co-processor applications).

The serial in-system programming feature (via two pins) offers flexibility of customizing the product after complete assembly and testing. This feature can be used to serialize a product, store calibration data, or program the device with the current firmware before shipping.

1.1 Family and Upward Compatibility

Those users familiar with the PIC16C5X family of microcontrollers will realize that this is an enhanced version of the PIC16C5X architecture. Please refer to Appendix A: for a detailed list of enhancements. Code written for PIC16C5X can be easily ported to the PIC16F8X (Appendix B:).

1.2 Development Support

The PIC16CXX family is supported by a full-featured macro assembler, a software simulator, an in-circuit emulator, a low-cost development programmer and a full-featured programmer. A "C" compiler and fuzzy logic support tools are also available.

TABLE 1-1: PIC16F8X FAMILY OF DEVICES

	Clock	Memory					Peripherals		Features		
	Maximum Frequency of Operation (MHz)	Program Memory Flash	Program Memory EEPROM	Program Memory ROM	Data Memory (bytes)	Data EEPROM (bytes)	Timer Module(s)	Interrupt Sources	I/O Pins	Voltage Range (Volts)	Packages
PIC16C84	10	—	1K	—	36	64	TMR0	4	13	2.0-6.0	18-pin DIP, SOIC
PIC16F84(1)	10	1K	—	—	68	64	TMR0	4	13	2.0-6.0	18-pin DIP, SOIC
PIC16CR84(1)	10	—	—	1K	68	64	TMR0	4	13	2.0-6.0	18-pin DIP, SOIC
PIC16F83(1)	10	512	—	—	36	64	TMR0	4	13	2.0-6.0	18-pin DIP, SOIC
PIC16CR83(1)	10	—	—	512	36	64	TMR0	4	13	2.0-6.0	18-pin DIP, SOIC

All PIC16/17 family devices have Power-on Reset, selectable Watchdog Timer, selectable code protect, and high I/O current capability.

All PIC16C8X family devices use serial programming with clock pin RB6 and data pin RB7.

Note 1: Please contact your local sales office for availability of these devices.

2.0 PIC16F8X DEVICE VARIETIES

A variety of frequency ranges and packaging options are available. Depending on application and production requirements the proper device option can be selected using the information in this section. When placing orders, please use the "PIC16F8X Product Identification System" at the back of this data sheet to specify the correct part number.

There are four device "types" as indicated in the device number.

1. **F**, as in PIC16**F**84. These devices have Flash program memory and operate over the standard voltage range.
2. **LF**, as in PIC16**LF**84. These devices have Flash program memory and operate over an extended voltage range.
3. **CR**, as in PIC16**CR**83. These devices have ROM program memory and operate over the standard voltage range.
4. **LCR**, as in PIC16**LCR**84. These devices have ROM program memory and operate over an extended voltage range.

When discussing memory maps and other architectural features, the use of **F** and **CR** also implies the **LF** and **LCR** versions.

2.1 Electrically Erasable Devices

These devices are offered in the lower cost plastic package, even though the device can be erased and reprogrammed. This allows the same device to be used for prototype development and pilot programs as well as production.

A further advantage of the electrically erasable version is that they can be erased and reprogrammed in-circuit, or by device programmers, such as Microchip's PICSTART® Plus or PRO MATE® II programmers.

2.2 Quick-Turnaround-Production (QTP) Devices

Microchip offers a QTP Programming Service for factory production orders. This service is made available for users who choose not to program a medium to high quantity of units and whose code patterns have stabilized. The devices have all Flash locations and configuration options already programmed by the factory. Certain code and prototype verification procedures do apply before production shipments are available.

For information on submitting a QTP code, please contact your Microchip Regional Sales Office.

2.3 Serialized Quick-Turnaround-Production (SQTP℠) Devices

Microchip offers the unique programming service where a few user-defined locations in each device are programmed with different serial numbers. The serial numbers may be random, pseudo-random or sequential.

Serial programming allows each device to have a unique number which can serve as an entry-code, password or ID number.

For information on submitting a SQTP code, please contact your Microchip Regional Sales Office.

2.4 ROM Devices

Some of Microchip's devices have a corresponding device where the program memory is a ROM. These devices give a cost savings over Microchip's traditional user programmed devices (EPROM, EEPROM).

ROM devices (PIC16CR8X) do not allow serialization information in the program memory space. The user may program this information into the Data EEPROM.

For information on submitting a ROM code, please contact your Microchip Regional Sales Office.