TEMA – 6

MICROCONTROLADORES. PIC 16F84.

1. Introducción.

En las aplicaciones sencillas resultan preferibles las soluciones no programables que no requieren desarrollo de software. Escribir software consume mucho tiempo por lo que resulta más costoso y en aplicaciones sencillas y/o de poca tirada a menudo es más razonable efectuar tareas en hardware. Sin embargo, conforme aumenta la complejidad del sistema, aumentan las ventajas del uso de sistemas programables.

Una de las principales ventajas de los sistemas programables es su flexibilidad, lo que permite actualizar el funcionamiento de un sistema tan sólo mediante el cambio del programa sin tener que volver a diseñar el hardware. Esta flexibilidad es muy importante, al permitir que los productos se actualicen con facilidad y economía.

1.1. Referencia histórica.

En el año 1971 la compañía de semiconductores Intel lanzó al mercado el primer **microprocesador**, lo que supuso un cambio decisivo en las técnicas de diseño de los equipos de instrumentación y control. Este circuito integrado contenía todos los componentes de la unidad central de procesos (CPU) de una computadora dentro de un solo dispositivo. Los fabricantes, conscientes de la importancia de este mercado, crearon una amplia gama de estos circuitos integrados, constituyendo familias de microprocesadores.

En el año 1976, gracias al aumento de la densidad de integración de componentes en un circuito integrado, salió a la luz el primer ordenador en un chip, es decir se integraron junto con el microprocesador otros subsistemas que anteriormente formaban unidades independientes (memoria, entradas/salidas, etc.). A este nuevo integrado se le denominó **microcomputadora monopastilla**.

Cuando los sistemas basados en microprocesadores se especializan en aplicaciones industriales, aparece la versión industrial de la microcomputadora monopastilla, el **microcontrolador** (μC). Estos dispositivos producen un obvio beneficio en aplicaciones pequeñas. Su característica más sobresaliente es que son sistemas integrados.

1.2. ¿Qué es un microcontrolador?

Es un circuito integrado que contiene todos los componentes de un computador. Se emplea para controlar el funcionamiento de una tarea determinada y, debido a su reducido tamaño, suele ir incorporado en el propio dispositivo al que gobierna. Esta última característica es la que le confiere la denominación de «controlador incrustado» (*embedded controller*). Se dice que es “la solución en un chip” porque su reducido tamaño minimiza el número de componentes y el coste.

El microcontrolador es un computador dedicado. En su memoria sólo reside un programa destinado a gobernar una aplicación determinada; sus líneas de entrada/salida soportan el conexionado de los sensores y actuadores del dispositivo a controlar. Una vez programado y configurado el microcontrolador solamente sirve para gobernar la tarea asignada.

“Un microcontrolador es un computador completo, aunque de limitadas prestaciones, que está contenido en el chip de un circuito integrado y se designa a gobernar una sola tarea”.

El número de productos que funcionan en base a uno o varios microcontroladores aumenta de forma exponencial. Casi todos los periféricos del computador (ratón, teclado, impresora, etc.) son regulados por el programa de un microcontrolador. Los electrodomésticos de línea blanca (lavadoras, hornos, etc.) y de línea marrón (televisores, videos, aparatos de música, etc.) incorporan numerosos microcontroladores. Igualmente, los sistemas de supervisión, vigilancia y alarma en los edificios utilizan estos chips para optimizar el rendimiento de ascensores, calefacción, alarmas de incendio, robo, etc. Ofrecen la única solución práctica a muchos problemas de diversos campos:

1. Periféricos y dispositivos auxiliares de los computadores.
2. Electrodomésticos.
3. Aparatos portátiles y de bolsillo (tarjetas, monederos, teléfonos, etc.)
4. Máquinas expendedoras y juguetería.
5. Instrumentación.
6. Industria de automoción (Figura 6-1).
7. Control industrial y robótica.
8. Electromedicina.
9. Sistema de navegación espacial.
10. Sistemas de seguridad y alarma. Domótica en general.

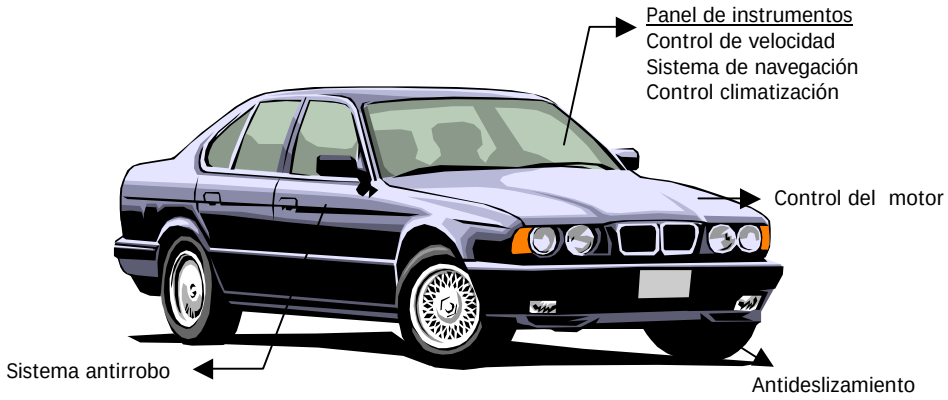

Figura 6-1. Aplicaciones de microcontroladores en un automóvil.

1.3. Diferencia entre microcontrolador y microprocesador.

El microprocesador es un circuito integrado que contiene la Unidad Central de Proceso (CPU), también llamada procesador de un computador. Al microprocesador se le conecta la Memoria y Módulos de E/S para configurar un computador implementado por varios circuitos integrados.

Un microprocesador es un sistema **abierto** (configuración variable) con el que puede construirse un computador con las características que se desee, acoplándole los módulos necesarios. Un microcontrolador es un sistema **cerrado** que contiene un computador completo y de prestaciones limitadas que no se pueden modificar.

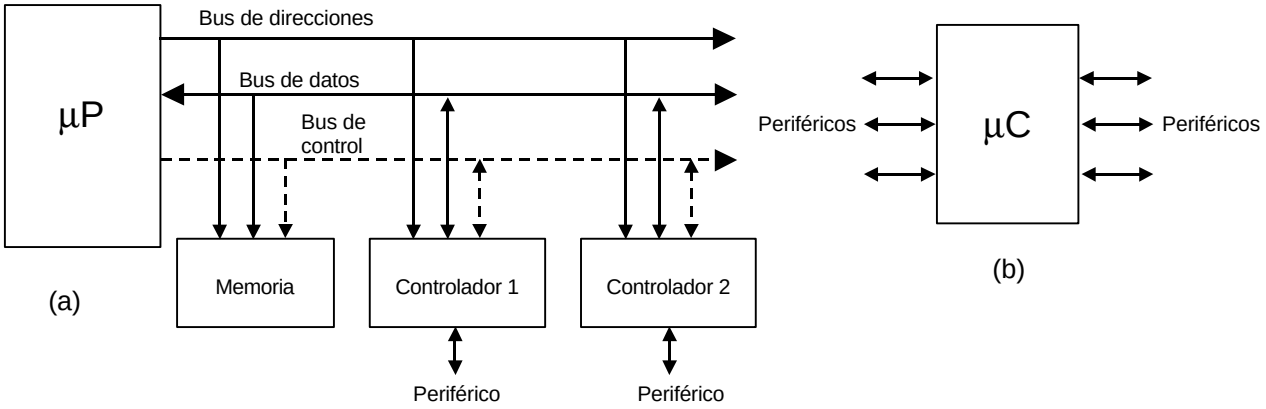

Figura 6-2 (a)Estructura de un sistema abierto basado en un microprocesador. (b) Microcontrolador.

Decidirse por construir el sistema con el microprocesador, o utilizar directamente el microcontrolador dependerá de la economía. Si el µC está limitado por su propia CPU, es necesario elegir un µP potente y añadir los buffers, drivers, decodificadores, memorias, etc. pertinentes. Generalmente, salvo que la aplicación exija grandes prestaciones, el µC será una solución válida, con la ventaja de que reduce el espacio y el hardware.

1.4. Fabricantes.

Gran parte de los fabricantes de circuitos integrados disponen de su propia línea de microcontroladores. En la tabla 6-1 se relacionan los fabricantes más destacados.

Tabla 6-1. Algunos fabricantes de µC con algunos modelos.

Fabricante	Modelos de µC
INTEL	8048, 8051, 80C196, 80186, 80188, 80386EX
MOTOROLA	6805, 68HC11, 68HC12, 68HC16, 683XX
HITACHI	HD64180
PHILIPS	Gama completa de clónicos del 8051
SGS-THOMSON (ST)	ST-62XX
MICROCHIP	PICs
NATIONAL SEMICONDUCTOR	COP8
ZILOG	Z8, Z80
TEXAS INSTRUMENTS	TMS370
TOSHIBA	TLCS-870
INFINEON	C500
DALLAS	DS5000
NEC	78K

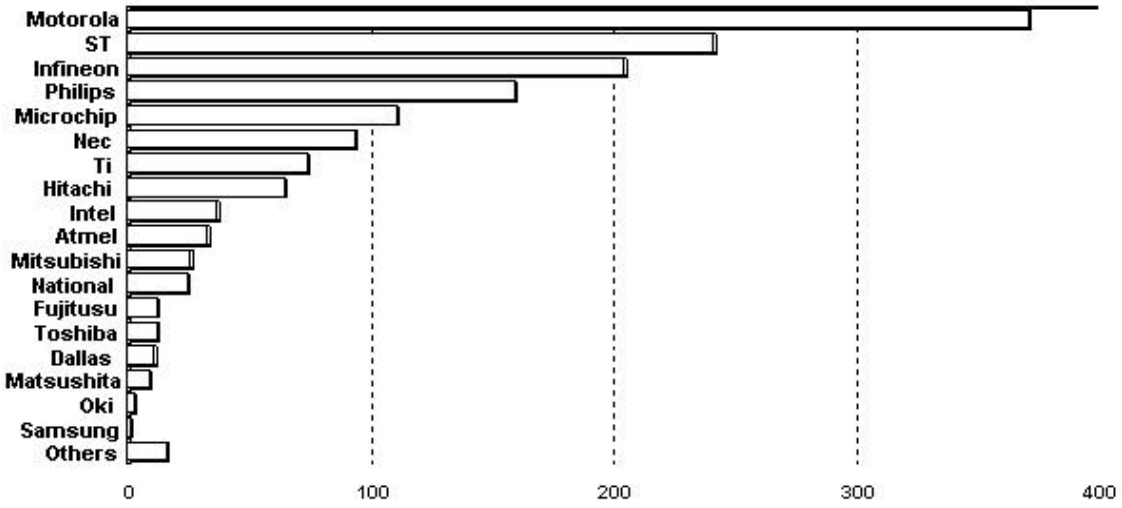

Figura 6-3. Ranking europeo de MCU de 8 bits.

Cada fabricante de microcontroladores oferta un elevado número de modelos diferentes, desde los más sencillos hasta los más poderosos, de forma que es posible seleccionar la capacidad de la memoria, el número de líneas de E/S, la cantidad y potencia de elementos auxiliares, la velocidad de funcionamiento, etc. En la Figura 6-3 se muestra el ranking de ventas en Europa de microcontroladores de 8 bits.

Se considera a **Intel** como el padre de los microcontroladores y al 8048 como el primer microcontrolador de 8 bits (fabricado por Intel en la década de los 70).

Otra de las principales empresas del mundo de dispositivos programables es **Motorola**, que dispone del potente microcontrolador 68HC11.

Los microcontroladores PIC de la empresa americana **Microchip** se emplean en la actualidad cada vez más debido a su reducido consumo, bajo coste, pequeño tamaño, facilidad de uso y la abundancia de información y herramientas de apoyo.

2. Arquitectura interna de un microcontrolador.

Un microcontrolador posee todos los componentes de un computador, pero con características fijas que no se pueden alterar.

Las partes principales son:

- Procesador o Unidad Central de Proceso (CPU).
- Memoria no volátil para contener el programa.
- Memoria de lectura y escritura para guardar los datos.
- Líneas de E/S para los periféricos (comunicación serie, paralela, etc.).
- Recursos auxiliares:
    - Circuito de reloj.
    - Temporizadores.
    - Perro guardián (*Watchdog*).
    - Convertidores A/D y D/A.

- Comparadores analógicos.
- Protección ante fallos de alimentación.
- Estado de reposo en bajo consumo.

La comunicación entre las principales secciones de un sistema computador (basado en microcontrolador o en microprocesador) tiene lugar sobre un cierto número de **buses**. Un bus está compuesto de líneas paralelas de datos que permiten flujo de información en uno o ambos sentidos (se pueden considerar un conjunto de conductores paralelos). La Figura 6-4 muestra la estructura de buses de un sistema típico. Se usan tres buses:

- **Bus de datos.** Se emplea para transferir datos. El número de líneas de este bus igual a la longitud de palabra del dispositivo.
- **Bus de direcciones.** Permite transferir información de direcciones. El número de líneas en el bus de direcciones determina el numero de posiciones de memoria que el procesador puede especificar. Un bus de direcciones de 8 líneas sería capaz de posicionar sólo 2^8 (256) direcciones.
- **Bus de control.** El procesador utiliza las líneas del bus de control para sincronizar operaciones con componentes externos.

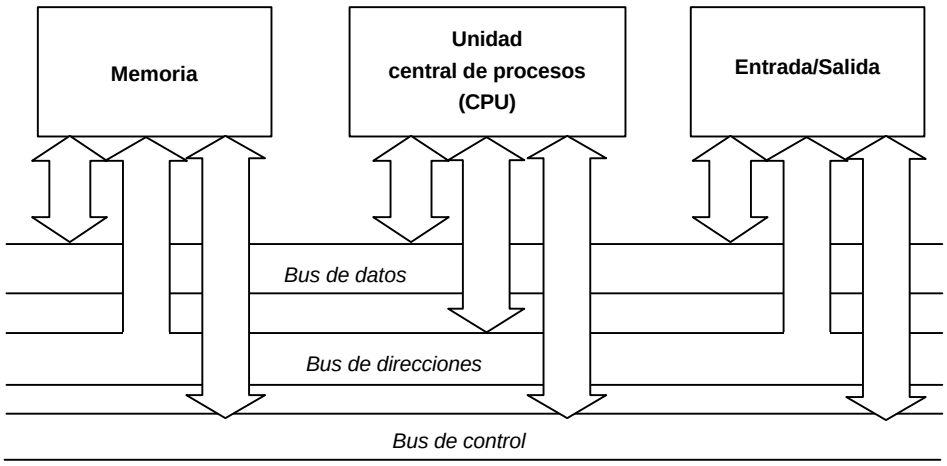

Figura 6-4 Sistema de buses.

Por ejemplo, si el procesador o microcontrolador deseara almacenar una palabra de datos en una ubicación de memoria particular, colocaría los datos en el bus de datos, la dirección en la que se van a almacenar los datos en el bus de direcciones y diversas señales de control para sincronizar la operación de almacenamiento en el bus de control.

2.1. Procesador.

Es el elemento más importante del microcontrolador tanto a nivel hardware como software. Se encarga de direccionar la memoria de instrucciones, recibir el código OP (opcode) de la instrucción en curso, su decodificación y la ejecución de la operación aritmética o lógica que implica la instrucción, así como la búsqueda de los operandos y el almacenamiento del resultado.

La necesidad de conseguir rendimientos elevados en el procesamiento de las instrucciones ha desembocado en el empleo generalizado de procesadores de arquitectura

Hardvard, frente a los tradicionales que seguían la arquitectura de **Von Neuman**. Esta última se caracteriza porque la CPU se conectaba a una memoria única donde coexistían datos e instrucciones a través de un sistema de buses. En la arquitectura **Harvard** son independientes la memoria de instrucciones y la memoria de datos y cada una dispone de su propio sistema de buses para el acceso. Esto propicia el paralelismo (Figura 6-5).

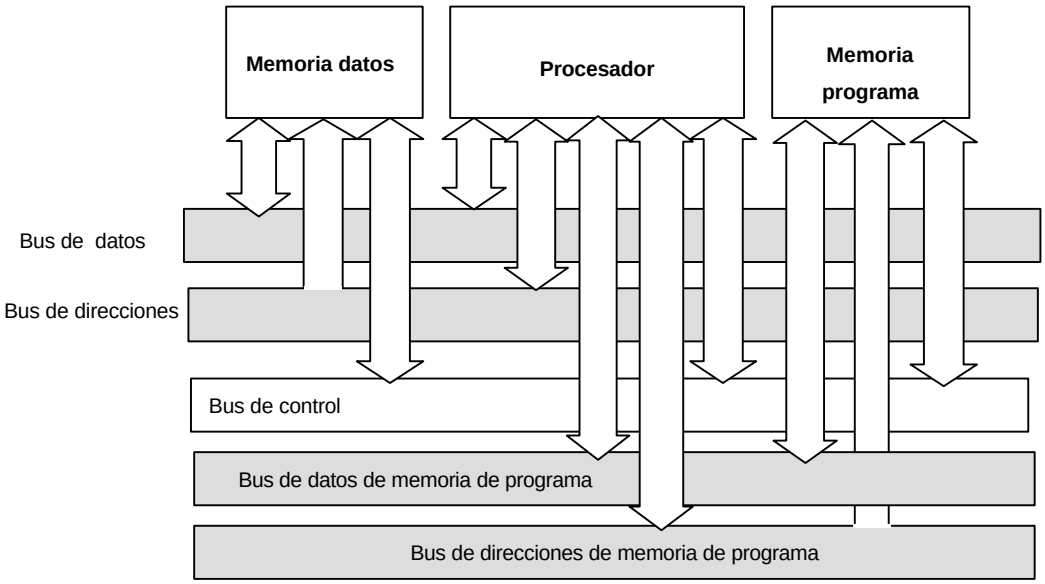

Figura 6-5 Arquitectura Hardvard.

2.1.1- Almacenamiento y ejecución del programa.

Un programa es una lista de instrucciones al procesador. Todos los μP y μC tiene un conjunto de instrucciones que pueden ejecutar. Cada tipo de procesador y controlador tiene su propio conjunto de instrucciones, y por lo general los programas escritos para una máquina no funcionarán en otra.

Un μC típico tendrá instrucciones para: transferir información entre registros y memoria; realizar operaciones aritméticas y lógicas; efectuar comparaciones y pruebas sobre el contenido de sus registros de memoria; controlar la secuencia de ejecución de programas.

Por lo general la operación que una instrucción ha de ejecutar está definida por un **código de operación,** (en μC sencillos generalmente de un solo byte) conocido también como **opcode**. Algunas instrucciones requieren además del opcode información extra (**operandos**). Por ejemplo, una instrucción para almacenar el contenido de un registro en una posición de memoria, necesitará incluir la dirección de memoria de destino.

La unidad de control y decodificación de instrucciones es el corazón del procesador. Es la encargada de extraer de forma secuencial las instrucciones de la memoria y luego ejecutarlas. Unido a la unidad de control se encuentra un **generador de reloj**, que utiliza un oscilador para producir una señal de reloj muy precisa. El tiempo se divide en un cierto número de ciclos de reloj.

El funcionamiento de la unidad de control y decodificación de instrucciones se puede dividir en dos partes (ambas duran varios ciclos de reloj):

- **Ciclo de búsqueda de instrucciones**. En él se transfiere la dirección de la siguiente instrucción al bus de direcciones, se envía una orden de lectura a los dispositivos de memoria a través del bus de control, se lee la información del bus de datos, y si se trata del primer byte de una instrucción, se transfiere este byte al registro de instrucciones.

- **Ciclo de ejecución de instrucciones.** En él se ejecuta la instrucción. El registro de instrucciones está conectado a un decodificador, que determina cuántos bytes de información adicionales se requieren. Éstos se cargan mediante ciclos de búsqueda, como se indicó con anterioridad.

Cuando la ejecución está completa, la máquina comienza automáticamente el ciclo de búsqueda de la siguiente instrucción del programa. La ejecución es por tanto, una secuencia continua de ciclos de búsqueda y ejecución.

El procesador de los modernos microcontroladores responde a la arquitectura **RISC** (Computadores de Juego de Instrucciones Reducido), que se identifica por tener un repertorio de instrucciones máquina pequeño y simple, de forma que la mayor parte de las instrucciones se ejecutan en un ciclo de instrucción.

2.2. Memoria de programa.

El microcontrolador está diseñado para que en su memoria de programa se almacenen todas las instrucciones del programa de control. Como el programa a ejecutar siempre es el mismo, debe estar grabado de forma permanente. Son posibles cinco tipos de memoria:

- **ROM de máscara**. Esta memoria se graba en el chip durante el proceso de fabricación. Los altos costes de diseño sólo aconsejan usarla cuando se precisan series grandes.
- **EPROM**. En la superficie de la cápsula del microcontrolador existe una ventana de cristal por la que se puede someter al chip a rayos ultravioletas para producir el borrado de la memoria y emplearla nuevamente. Su coste unitario es elevado.
- **OTP (*One Time Programmable*)**. Este modelo de memoria sólo se puede grabar una vez por parte del usuario. Su bajo precio y la sencillez de la grabación aconsejan este tipo de memoria para prototipos finales y series de producción cortas.
- **EEPROM**. La grabación es similar a la EPROM y OTP, pero el borrado es mucho más sencillo al poderse ejecutar eléctricamente las veces que se quiera.
- **FLASH**. Se trata de una memoria no volátil de bajo consumo que se puede escribir y borrar en circuito al igual que la EEPROM, pero suele disponer de mayor capacidad que estas últimas. El borrado sólo es posible de bloques completos y no se puede realizar de posiciones concretas. Por sus mejores prestaciones está sustituyendo a la memoria EPROM para contener instrucciones.

2.3. Memoria de datos.

Los datos que manejan los programas varía continuamente y eso exige que la memoria que los contiene deba ser de lectura y escritura. La memoria RAM estática (SRAM) es la más apropiada aunque sea no volátil. Hay microcontroladores que poseen como memoria de datos una memoria de escritura y lectura no volátil del tipo EEPROM. De este forma, un corte en el suministro de la alimentación no ocasiona la pérdida de la información.

2.4. Líneas de E/S para los controladores de periféricos.

A excepción de las patillas destinadas a recibir la alimentación, el cristal que regula la frecuencia de trabajo y el reset, las restantes patillas de un microcontrolador sirven para la comunicación con los periféricos externos.

2.5. Recursos auxiliares.

Según las aplicaciones cada modelo de microcontrolador incorpora una diversidad de complementos que refuerzan la potencia del dispositivo. Entre los más comunes se encuentran:

- Circuito de reloj, encargado de generar los impulsos que sincronizan el funcionamiento de todo el sistema.
- Temporizadores, orientados a controlar tiempos.
- Perro guardián (*watchdog*), destinado a provocar una reinicialización cuando el programa se queda bloqueado.
- Conversores A/D y D/A para poder recibir y enviar señales analógicas.
- Estado de reposo, en el que el consumo de energía se reduce al mínimo.

3. El microcontrolador PIC16F84.

Dado que las aplicaciones sencillas precisan pocos recursos y las aplicaciones más complejas requieren numerosos y potentes recursos, **Microchip** construye diversos modelos de microcontroladores orientados a cubrir las necesidades de cada proyecto. Siguiendo esta filosofía **Microchip** oferta diferentes gamas de microcontroladores:

- **PIC12CXXX** gama baja(8-pin, palabra de programa de 12 bits/14 bits):
  - Bajo consumo.
  - Memoria de datos EEPROM.
- **PIC16C5X**, gama baja o clásica (palabra de programa de 12 bits):
  - Encapsulados de 14, 18, 20 y 28 pines.
  - Óptimo para aplicaciones que trabajan con baterías (bajo consumo).
- **PIC16CXXX**, gama media (palabra de programa de 14 bits).
  - Convertidores A/D y puerto serie.
  - Encapsulados desde 18 a 68 pines.
- **PIC17CXXX**, gama alta (palabra de programa de 16 bits).
  - Arquitectura abierta, memoria ampliable.
- **PIC18XXX**, gama alta (palabra de programa de 16 bits).
  - Conjunto de instrucciones mejorado.
  - Detección de bajo voltaje programable (PLVD).

Dentro de cada gama se dispone de una gran variedad de modelos y encapsulados, pudiendo seleccionar el que mejor se adapte a cada proyecto (Tabla 6-2). En el año 2000 se comercializaron más de un centenar de modelos de PIC que cubren desde los "enanos" de ocho patillas y mínimos recursos hasta los "avanzados" de 84 patillas enormemente potentes.

Estos microcontroladores han logrado en pocos años ocupar un elevado puesto del ranking mundial de ventas de microcontroladores, dada su economía, fiabilidad, rapidez, abundante información técnica y el precio asequible de las herramientas que se precisan para desarrollar las aplicaciones.

La arquitectura y programación de todos los PIC es muy parecida. En este tema nos centremos en el microcontrolador PIC16F84, que es uno de los microcontroladores de la gama media. Sus características principales son las siguientes:

- Parte de la memoria de datos es de tipo EEPROM (64 registros de 8 bits).
- Memoria de programa (1024 registros de 14 bits) de tipo flash, de iguales prestaciones que la EEPROM pero mejor rendimiento.
- Dos temporizadores: TMR0 y *watchdog*. El TMR0 puede actuar como temporizador o como contador.
- Cuatro posibles fuentes de interrupción que pueden habilitarse o deshabilitarse por software.
- Reinicialización del sistema o *RESET* por cinco causas distintas.
- Estado de funcionamiento en bajo consumo o *Sleep*, con un consumo de 40 μA.
- Frecuencia de trabajo máxima puede ser de 10 MHz.

Tabla 6-2. Familias de microcontroladores PIC.

Familia	Descripción
PIC12C5XX	8-Pin, 8-Bit CMOS
PIC12CE5XX	8-Pin, 8-Bit CMOS con Memoria de Datos EEPROM
PIC12C67X	8-Pin, 8-Bit CMOS con Convertidor A/D
PIC12CE67X	8-Pin, 8-Bit CMOS con Convertidor A/D y Memoria de Datos EEPROM
PIC12F6XX	8-Pin, 8-Bit Flash con EEPROM y A/D
PIC16C5X PIC16HV540	EPROM/ROM 8-Bit CMOS
PIC16C55X	EPROM-Based 8-Bit CMOS
PIC16C6X	8-Bit CMOS
PIC16X62X	8-Pin EPROM-Based 8-Bit CMOS con Comparador
PIC16C64X PIC16C66X	8-Bit EPROM con Comparador analógico
PIC16CE62X	8-Bit CMOS con Comparador analógico y Memoria de Datos EEPROM
PIC16C7X	8-Bit CMOS con convertidor A/D.
PIC16C71X	18 Pin, 8-Bit CMOS con convertidor A/D.
PIC16C43X	18/20 Pin, 8-Bit CMOS con bus LIN.
PIC16C78X	Convertidor A/D, D/A, amplificador operacional, comparadores y PSMC.
PIC16C7X5	8-Bit CMOS con convertidor A/D , para aplicaciones de USB, PS/2 y dispositivo serie.
PIC16C77X	20-pin, 28-pin y 40-Pin, 8-Bit CMOS con convertidor A/D 12-bit.
PIC16F87X	28/40-Pin, 8-Bit CMOS FLASH con convertidor A/D 10-bit .
PIC16X8X	18-Pin, 8-Bit CMOS FLASH/EEPROM
PIC16F7X	28/40-pin 8-bit CMOS FLASH
PIC16C9XX	LCD, convertidor A/D 8 y 10-bit .
PIC17C4X	8-Bit CMOS EPROM/ROM, alto rendimiento.
PIC17C75X	8-Bit CMOS EPROM, alto rendimiento
PIC18CXXX	8-Bit, arquitectura mejorada.
PIC18F0XX	8-Pin, 8-Bit arquitectura mejorada FLASH con EEPROM, PLVD, BOR y PWM.
PIC18FXX2	Protección de código, 256 EEPROM de datos, Detección de bajo voltaje programable (LVD), Phase-locked Loop (PLL), modo SLEEP , multiplicador 8x8 , PSP, In-Circuit Debugging
PIC18FX32	28/40-Pin FLASH alto rendimiento, modo de bajo consumo, EEPROM de de 256 datos, PLVD y A/D 10-bit.
PIC18FXX31	28/40-Pin FLASH con EEPROM, PLVD, PBOR, A/D 10-bit y módulo PWM.
PIC18FXX5	28/40-Pin FLASH con USB y A/D 10-bit.
PIC18FXX30	18/20-pin FLASH con EEPROM, PBOR, A/D 10-bit y módulo PWM 3-fase.

3.1. Arquitectura interna.

Los microcontroladores PIC están basados en la arquitectura **Harvard** que posee buses y espacios de memoria diferenciados para los datos y las instrucciones. Gracias a ella se puede acceder de forma simultánea e independiente a la memoria de datos y a la memoria de instrucciones, por tanto son más rápidos que los microcontroladores basados en la arquitectura tradicional de **Von Neuman**.

Por otro lado, esta independencia entre datos e instrucciones permite que cada uno tenga el tamaño más adecuado. Así, los datos tienen una longitud de 8 bits, mientras que las instrucciones son de 14 bits (Figura 6-6).

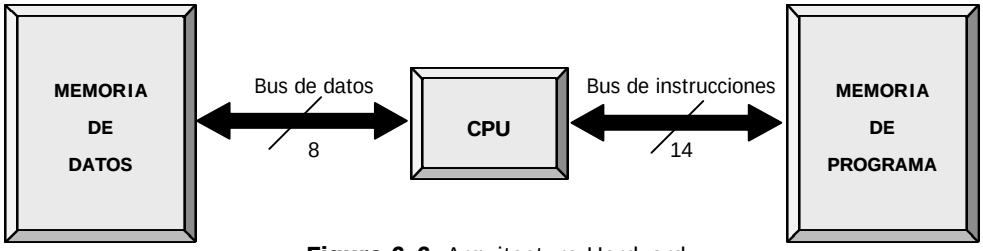

Figura 6-6. Arquitectura Hardvard.

Como se observa en la Figura 6-7 el PIC16F84 consta de un procesador con una ALU y un Decodificador de Instrucciones, una memoria de programa tipo FLASH de 1K palabras de 14 bits, una memoria de datos SRAM con 68 posiciones de 8 bits. También existe una zona de 64 bytes de EEPROM para datos no volátiles. Finalmente dispone de interrupciones, un temporizador, perro guardián y dos puertas A y B de entrada y salida de información digital.

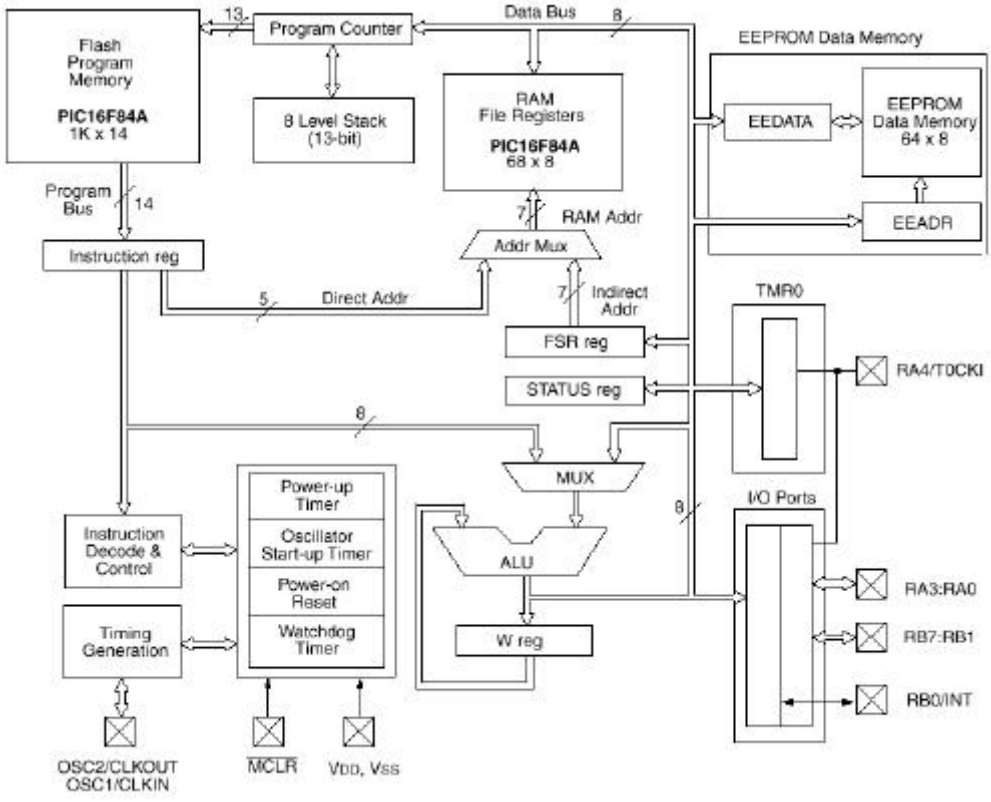

Figura 6-7. Diagrama de bloques del PIC16F84.

3.1.1. El procesador o CPU.

El procesador responde a la arquitectura RISC, que se identifica porque el juego de instrucciones se reduce a 35, donde la mayoría se ejecutan en un solo ciclo de reloj, excepto las instrucciones de salto que necesitan dos ciclos.

La ALU (*Arithmetic Logic Unit*) ubicada dentro del procesador realiza las operaciones lógicas y aritméticas con dos operandos, uno que recibe desde el registro **W** (registro de trabajo) y otro que puede provenir de cualquier registro interno (Figura 6-7).

3.1.2. Memoria de programa.

La memoria de programa es del tipo flash. La memoria flash es una memoria no volátil, de bajo consumo que se puede escribir y borrar eléctricamente. Es programable en el circuito como la EEPROM, pero de mayor densidad y más rápida.

El PIC16F84 posee una memoria de programa de 1K palabras, es decir, permite hasta 1024 (2^{10}) instrucciones de 14 bits cada una (de la 0000h a la 03FFh).

En la posición **0000h** (Figura 6-8) se encuentra el **vector de reset**. Cuando se produce un reset, el programa salta a dicha posición (0000h).

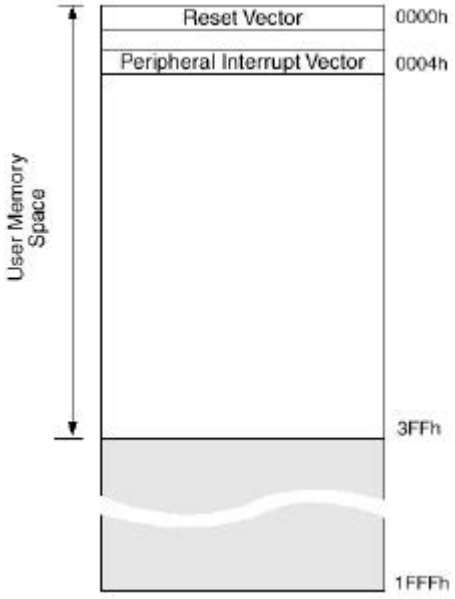

Figura 6-8. Mapa de la memoria de programa.

Las causas por las que se produce un reset son las siguientes:

- Conexión de la alimentación.
- Cuando durante el funcionamiento normal, o en el modo bajo consumo (s*leep*) se aplican cero voltios al terminal de $\overline{\text{MCLR}}$.
- Desbordamiento del perro guardián (*watchdog*) en funcionamiento normal o en funcionamiento en bajo consumo (s*leep*).

En la posición **0004h** (Figura 6-8) se encuentra el **vector de interrupción**, que es común a todas las interrupciones (las interrupciones pueden habilitarse o deshabilitarse de forma independiente por software). Las causas por las que se puede producir una interrupción son la siguientes:

- Activación de la patilla de interrupción externa INT.
- Desbordamiento del temporizador.
- Cambio de estado en alguno de los terminales de entrada del puerto B (RB4-RB7).
- Finalización del proceso de escritura de la EEPROM de datos.

Cuando se produce una interrupción se activan unos indicadores (flags) para identificar la causa de la interrupción, se detiene la ejecución del programa almacenando la posición de la instrucción actual, para que una vez finalizadas las operaciones asociadas a la interrupción, el programa pueda continuar por donde iba y el programa salta a la posición 0004h. A partir de aquí se ejecutará la rutina de interrupción. Con la última instrucción de la rutina de interrupción automáticamente el programa continuará por donde se había quedado.

3.1.3. Memoria de datos

Se encuentra en dos zonas bien diferenciadas:

a) Memoria tipo RAM (SRAM): Se divide en dos bancos o páginas de 128 registros de 8 bits cada uno, aunque sólo los 80 primeros de la página '0' (del 00h al 4Fh) y los 12 primeros de la página '1' (80h al 8Bh) se utilicen en el PIC16F84. Los primeros 12 registros de cada página son específicos (SFR) y los restantes (68 de la página '0') son de propósito general (GPR).

b) La memoria de datos del tipo EEPROM está compuesta por 64 registros de 8 bits cada uno. Este tipo de memoria es capaz de guardar la información más de 40 años. No podemos acceder directamente a estos registros, para ello hay que utilizar registros específicos (SFR).

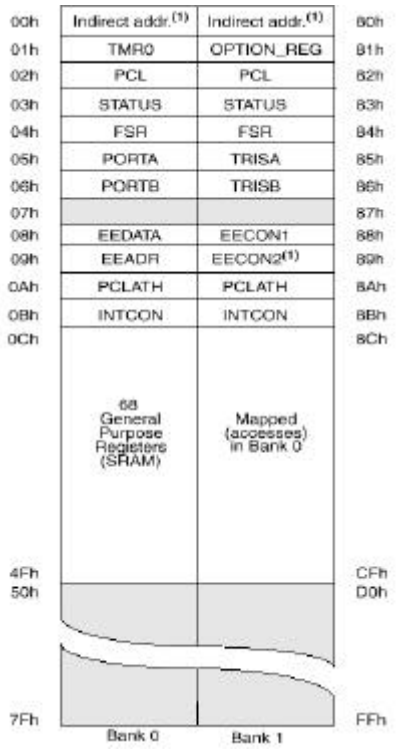

Figura 6-9. Mapa de la memoria de datos SRAM.

3.1.4. Puertas de entrada/salida

El microcontrolador PIC16F84 posee 13 patillas cada una de las cuales se puede configurar como entrada o como salida. Éstas se encuentran distribuidas en dos puertos: el puerto A con cinco patillas, desde RA0 a RA4; y el puerto B con ocho patillas, desde RB0 a RB7.

3.1.5. Recursos auxiliares.

El microcontrolador PIC16F84 dispone de los siguientes recursos adicionales:

- **Circuito de reloj**. Está incorporado en el microcontrolador y sólo se necesitan unos pocos componentes exteriores para seleccionar y estabilizar la frecuencia de trabajo. Dichos componentes suelen consistir en un cristal de cuarzo junto con elementos pasivos.

  La frecuencia del reloj principal determina el ciclo de instrucción (tiempo que tarda una instrucción en ejecutarse). En los PICs, un ciclo de instrucción emplea cuatro periodos de oscilación del reloj principal. Por ejemplo si la frecuencia del reloj principal es de 4 MHz, un ciclo de instrucción tardará en realizarse:

  T oscilación del reloj principal = **1/F** del reloj principal = 1/4 MHz = 250 ns

  Ciclo de instrucción = T oscilación x 4 = 250 ns x 4 = 1 μs.

  Si por ejemplo el programa tuviera 250 líneas de instrucciones, la ejecución completa tardaría : 250 x 1μs = 250 μs.

- **Estado de reposo.** Cuando el microcontrolador funciona en modo de reposo (*sleep*) la potencia necesaria es menor de 3 μA.
- **Temporizador.** Se incrementará en una unidad por cada pulso de reloj. Cuando llegue al valor máximo FFh (los 8 bits a '1'), se producirá el desbordamiento del registro asociado al temporizador (TMR0), se iniciará tomando el valor 00h y generará una interrupción (si se

había autorizado), repitiendo dicho ciclo indefinidamente. El temporizador puede recibir la señal de reloj que hace que se incremente de una señal externa a través de una patilla de entrada, o del reloj principal del microcontrolador (en ese caso dividida por cuatro). Se puede trabajar directamente con estas señales o bien pasarlas por un divisor, cuyo valor se puede seleccionar por software.

- **Perro guardián.** El temporizador perro guardián (*watchdog*) es independiente del reloj principal (posee su propio oscilador), por tanto, en el modo en bajo consumo puede seguir funcionando. Cuando llegue al valor máximo FFh, se produce el desbordamiento del perro guardián, se iniciará tomando el valor 00h y provocará un reset. El tiempo típico es de 18ms, pero utilizando un divisor de frecuencia (programable por software) se pueden alcanzar 2,5 s.

  La utilización del perro guardián permite controlar los posibles “cuelgues” del programa, esto es, si durante el programa hemos previsto el poner a cero el perro guardián para evitar que se genere un reset, en el momento que por un fallo no previsto el programa se quede en un bucle sin fin, al no poder poner a cero el perro guardián, éste generará un reset sacando al programa del bucle.

3.2. Distribución de los terminales.

El encapsulado del PIC16F84 es del tipo DIP de 18 pines (Figura 6-10) y está fabricado en tecnología CMOS, por lo que su consumo es muy reducido, 2 mA cuando trabaja a 4MHz. A continuación se describe la función de sus patillas:

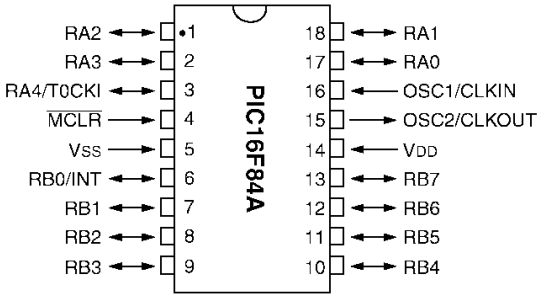

Figura 6-10 Disposición de pines.

- **V_{DD} y V_{SS}**
  Es la alimentación, que puede ser de 2 V a 6 V. V_{DD} es el terminal positivo y Vss el negativo. La tecnología CMOS permite que la tensión de alimentación pueda oscilar entre 2 y 6,25 V con un reducido consumo.

- **$\overline{\text{MCLR}}$**
  Es el reset del microcontrolador (*Master Clear*), el cual se produce cuando la tensión en dicho pin desciende entre 1,2V a 1,7V.
  Para la mayoría de aplicaciones bastará con conectarlo directamente a positivo.

  Cuando tengamos que utilizar un reset manual el fabricante recomienda el circuito de la Figura 6-11, con R_1 =100Ω y R =10KΩ.

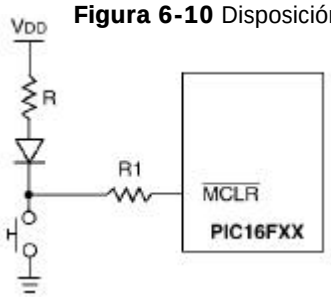

Figura 6-11 Circuito de reset.

- **RA0- RA4**
  Son los terminales de entrada/salida del puerto A. Los cuales pueden suministrar una corriente por cada pin de 20mA., pero la suma de las cinco líneas del puerto A no puede exceder de 50 mA. La corriente absorbida por cada pin puede ser de 25 mA, pero la suma de las cinco líneas no puede exceder de 80 mA.

  El pin RA4 tiene una doble función que se puede seleccionar por programa, y es la de ser la entrada del contador/temporizador TMR0, es decir T0CK1.

- **RB0-RB7**

  Son los terminales de entrada/salida del puerto B. Los cuales pueden suministrar una corriente por cada pin de 20 mA, pero la suma de las ocho líneas no puede superar los 100 mA. La corriente absorbida por cada pin puede ser de 25 mA, pero la suma de todas no puede exceder de 150 mA.

  El pin RB0 tiene una doble función seleccionable por programa, que es la de ser entrada de interrupción externa (INT). Los pines del RB4 al RB7 tienen una doble función que se puede seleccionar por programa, que es la de ser entrada de interrupción interna por cambio de estado.

- **OSC1/CLKIN OSC2/CLKOUT**

  Son los terminales para la conexión del oscilador externo que proporciona la frecuencia de trabajo o frecuencia del reloj principal.

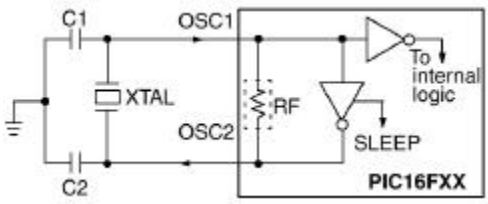

Figura 6-12. Circuito de reloj.

Frecuencia del cristal (XT)	Rango C_1	Rango C_2
100 KHz	68 – 100 pF	68 – 100 pF
2 MHz	10 – 22 pF	10 – 22 pF
4 MHz	10 – 22 pF	10 – 22 pF

3.2.1. Conexionado típico fijo para cualquier aplicación.

En los circuitos donde se utiliza el PIC16F84 es habitual emplear tensión de alimentación de +5V y como circuito de reloj externo uno del tipo XT a una frecuencia de 4MHz. Con esta configuración, el conexionado fijo para cualquier aplicación es el mostrado en la Figura 6-13.

Las patillas que no están conectadas son las dedicadas a transferir la información con los periféricos que utilice la aplicación.

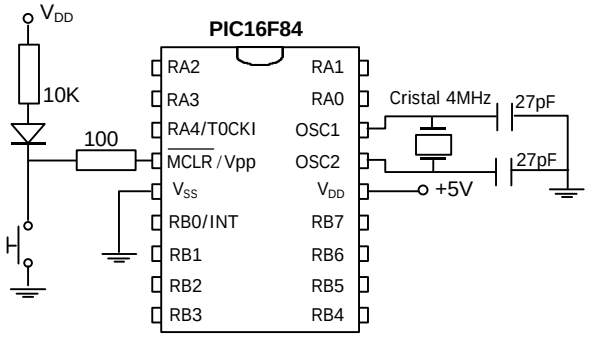

Figura 6-13. Conexionado fijo del PIC16F84.

3.3. Periféricos de E/S.

Los diseños reales utilizan diversos periféricos que hay que conectar a las patillas del microcontrolador que soportan las líneas de E/S.

3.3.1. Periféricos digitales de entrada.

- **Pulsadores.**

  Estos dispositivos permiten introducir un nivel lógico en el momento que se les acciona, pasando al nivel contrario cuando se deja de hacerlo (vuelven a la posición de reposo).

En el esquema de la izquierda de la Figura 6-14 la línea de entrada (IN) recibe un nivel lógico alto cuando el pulsador está reposo y un nivel lógico bajo cuando se acciona. El pulsador de la derecha funciona al revés.

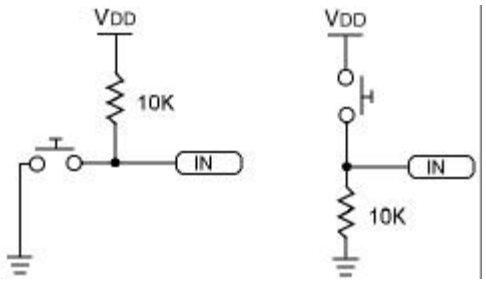

Figura 6-14. Dos posibles formas de conectar un pulsador.

Hay multitud de detectores, finales de carrera y sensores digitales que funcionan de la misma manera que los pulsadores.

- **Interruptores.**
  Los interruptores tienen dos estados estables y hay que accionarlos para cambiar de uno a otro. El interruptor admite el estado *abierto* y el estado *cerrado*. Las formas de conectar un interruptor a una entrada del microcontrolador son iguales a las de la figura 6-14, sustituyendo el pulsador por el interruptor.

Todos los circuitos electromecánicos (pulsadores, interruptores,...) originan un fenómeno denominado “rebotes”: las láminas se abren y se cierran varias veces en el momento de la transición (Figura 6-15). El efecto que produce es semejante a abrir y cerrar el interruptor o pulsador varias veces, por lo que puede provocar resultados erróneos.

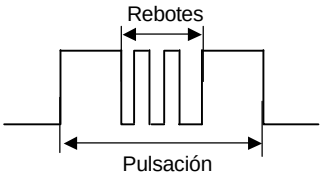

Figura 6-15 Efecto de los rebotes.

El efecto de los rebotes se puede solucionar bien mediante software, o bien por hardware. En la Figura 6-16 se muestran dos circuitos hardware antirrebotes. El circuito de la izquierda emplea un condensador y el de la derecha un flip-flop R-S.

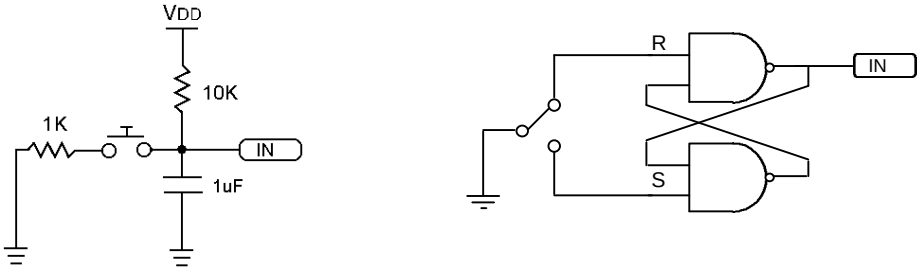

Figura 6-16. Esquemas para eliminar rebotes.

3.3.2. Periféricos digitales de salida

- **Diodos LED.**
  El diodo led es un elemento que se emplea como indicador luminoso. Cuando la diferencia de potencial entre su ánodo y su cátodo supere un determinado valor umbral el diodo led se encenderá. Las líneas de los PIC pueden suministrar suficiente corriente como para encender a un diodo led, por eso se pueden conectar directamente a través de una resistencia como muestra la Figura 6-17. Si empleamos la conexión de la izquierda de la figura, el diodo led se encenderá al poner a ‘1’ la salida del microcontrolador, mientras que con la conexión de la derecha lo hará cuando la salida se ponga a ‘0’.

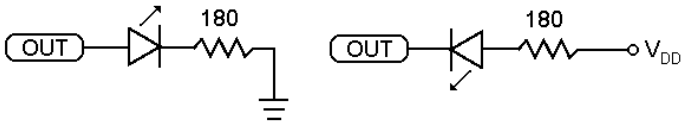

Figura 6-17. Dos posibles formas de conectar un led.

En ocasiones, los diodos u otro tipo de carga necesitan más corriente que la que pueden entregar las líneas de los PIC. En ese caso es necesario intercalar una etapa amplificadora.

- **Relés**

  La activación y desactivación de un relé brinda la oportunidad de poder controlar cargas mucho mayores (más corriente) porque pueden ser controladas por los contactos de dicho relé (Figura 6-18).

  Cuando la línea de salida, OUT, aplica un nivel alto a la base del transistor Darlington (etapa amplificadora) hace que conduzca y se active el relé. Al cerrarse los contactos del relé se controla una carga mayor. El valor de la resistencia depende del tipo de relé y del transistor.

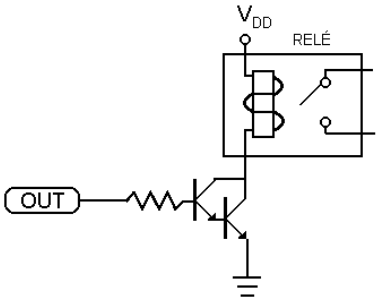

Figura 6-18. Esquema del control de un relé.

3.4. Programación.

Un programa es un conjunto ordenado de instrucciones que determinan el procesado de los datos y la obtención de los resultados. Cada procesador entiende un conjunto de instrucciones, que en el caso del PIC16F84 son 35. A dichas instrucciones se les llama "**instrucciones máquina**" y en este microcontrolador cada una de ellas consta de 14 bits con los cuales se indica el código de la operación.

Como sería muy complejo escribir las instrucciones con 14 bits, a cada una se la referencia con un **neumónico**, que es un conjunto de letras que expresan de forma resumida la operación que hace la instrucción. Al lenguaje que utiliza estos neumónicos se le llama **ensamblador**.

Programar en lenguaje ensamblador es programar con las mismas instrucciones que directamente puede ejecutar el procesador. Hay lenguajes de alto nivel en los que una instrucción equivale a muchas instrucciones máquina. Son más fáciles de utilizar, pero la utilización de los lenguajes más cercanos a la máquina (ensamblador) representan un considerable ahorro de código en la confección de los programas, lo que es muy importante dada la limitada capacidad de la memoria de instrucciones.

Entre los lenguajes de alto nivel que se pueden emplear con microcontroladores PIC se encuentran el lenguaje C y Basic.

Los compiladores son programas que se encargan de traducir el programa escrito en lenguaje ensamblador (u otro lenguaje) a código máquina, único lenguaje que el microcontrolador es capaz de entender. Tras la compilación, se grabará este código binario en la memoria de programa del microcontrolador.

3.4.1. Estructura general de un programa en ensamblador.

En una programa escrito en lenguaje ensamblador, además de las 35 instrucciones que interpreta el procesador también se colocan directivas, que son comandos para realizar ciertas operaciones con el programa. A continuación se comentan las partes que generalmente hay en un programa:

1º. **Modelo de procesador y sistema de numeración.**
Los programas comienzan con la directiva *list* que referencia el modelo de microcontrolador. También se suele especificar el tipo de numeración que se empleará con la directiva *radix*. Lo usual es emplear el sistema hexadecimal, en el que los valores se expresan precedidos de “0x”. En los ejemplos que se desarrollarán a lo largo del tema comenzaremos el programa ensamblador con las siguientes directivas (detrás del punto y coma se pueden añadir comentarios):

```
List  p=16F84      ;Se utiliza el microcontrolador PIC16F84
Radix hex          ; Se usará el sistema hexadecimal
```

2º. **Variables**.
Las posiciones de la memoria de datos se utilizan para guardar operandos y resultados, además de almacenar registros especiales.

Para que al programador le sea más sencillo confeccionar el programa, en lugar de hacer referencia a las posiciones de la memoria donde se encuentran los datos que va a emplear, a cada una de estas posiciones se le asocia un nombre. La directiva *equ* relaciona un nombre con la dirección que se asigna, así el programador trabaja con nombres y el compilador traduce automáticamente éstos a las direcciones correspondientes. Por ejemplo el registro que contiene la información de estado se encuentra en la dirección 0x03, el puerto de entrada A en 0x05, etc.. Si queremos emplear nombres de variables para estas direcciones de memoria escribiríamos:

```
ESTADO      equ 0x03   ;La etiqueta “ESTADO” está asociada a la dirección 0x03
PUERTAA     equ 0x05   ;La etiqueta “PUERTAA” está asociada a la dirección 0x05
```

3º. **Origen del programa**.
Antes de comenzar a escribir instrucciones máquina debe definirse la dirección de la memoria de programa a partir de la cual se desea comenzar a cargar el programa. Para ello se emplea la directiva *org.* En los PIC el origen del programa siempre se pone en la dirección 0x00 porque es donde comienza a ejecutarse el programa después de hacer un reset. Definiremos el origen de la siguiente manera:

```
org    0x00   ;Inicio de programa
```

Cuando el programa maneja interrupciones, no se comienza a cargar el programa desde la dirección 0x00, porque si se genera una interrupción el programa que la atiende comienza en la dirección 0x04 (vector de interrupción). En este caso lo que se suele hacer es poner en la dirección 0x00 un salto a una dirección de la memoria de programa posterior al vector de reset, por ejemplo saltaríamos a una posición etiquetada como INICIO que se encuentra en la dirección 0x05.

```
        org  0x00          ;La siguiente instrucción estará al inicio de la memoria
        goto INICIO        ;Salta a la dirección etiquetada con INICIO
        org  0x05          ;La siguiente instrucción estará en la dirección 0x05
INICIO
        -------
        -------
        end
```

4º. **Cuerpo del programa y final**.

Tras indicar la dirección donde se comenzará a cargar el programa, sigue el cuerpo del mismo compuesto por las instrucciones máquina y los operandos de éstas.

El código se estructura en columnas. La primera columna se utiliza para las etiquetas que se emplean para hacer referencia a partes del programa y nos permiten realizar saltos a estas partes (como `INICIO` en el ejemplo anterior). Las siguientes columnas contienen el campo de instrucciones, el campo de datos y el campo de comentarios. Los comentarios comienzan con ;).

Al final del programa se coloca la directiva *end*.

3.4.2. Ejemplos de programación.

Ejemplo 6.1. Sumar el contenido de la posición de memoria 0Ch (5) con el contenido de la posición de memoria 0Dh (2) y almacenar el resultado en la posición de memoria 0Eh.

El sumador (ALU) del PIC es capaz de sumar dos datos de 8 bits cada uno, pero debido a su configuración uno de los sumandos debe proceder del registro de trabajo W (Figura 6-7).

El ejemplo maneja tres posiciones de la memoria de datos de propósito general (figura 6-19). En la posición 0Ch se colocará el primer operando con el valor 5; en la posición 0Dh el segundo con el valor 2 y el resultado se almacenará en 0Eh. Como se vio en la Figura 6-9 las direcciones 0Ch, 0Dh y 0Eh son las tres primeras posiciones de la memoria de datos RAM (banco 0) que el usuario puede emplear para fines propios.

Antes de exponer el código del programa se explican las instrucciones que se van a utilizar:

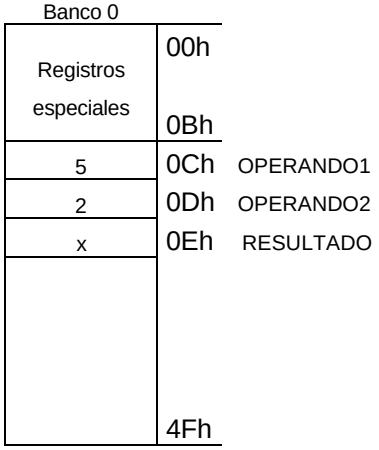

Figura 6-19. Mapa de memoria del ejemplo1.

- Instrucción ***mov***
  Permite transferir el contenido de un registro fuente *f* a un registro destino *d*. En los PIC todos los datos residen en posiciones de la memoria de datos a excepción del registro W. La instrucción *mov* puede mover tres tipos fundamentales de operandos:
  1. El contenido del registro W.
  2. El contenido de una posición de memoria de datos.
  3. Un literal o valor.
- ***movf f,d*** : mueve el contenido del operando fuente *f* (posición de la memoria de datos) al destino *d* (puede ser W o la propia fuente).

- ***movwf f*** : mueve el contenido del registro W a la posición de memoria especificada por *f.*
- ***movlw k*** : mueve el valor *k* al registro W.
- ***addwf f,d*** : suma el contenido del registro W con el de *f* y deposita el resultado en W si *d* = 0 (o W), o en *f* si *d* = 1.
- ***addlw k*** : suma al contenido del registro W el valor *k* y deposita el resultado en W.

A continuación se expone el código ensamblador del programa y el diagrama de flujo (Figura 6-20).

```
          List  p=16f84

OPERANDO1 equ   0x0C
OPERANDO2 equ   0x0D
RESULTADO equ   0x0E

          org   0x00          ;origen
INICIO
          movlw 0x05          ;W ← '5'
          movwf OPERANDO1     ;OPERANDO1←W
          movlw 0x02          ;W ← '2'
          movwf OPERANDO2     ;OPERANDO2←W
          addwf OPERANDO1,W   ;OPERANDO1+W →W
          movwf RESULTADO     ;RESULTADO ←W
          end
```

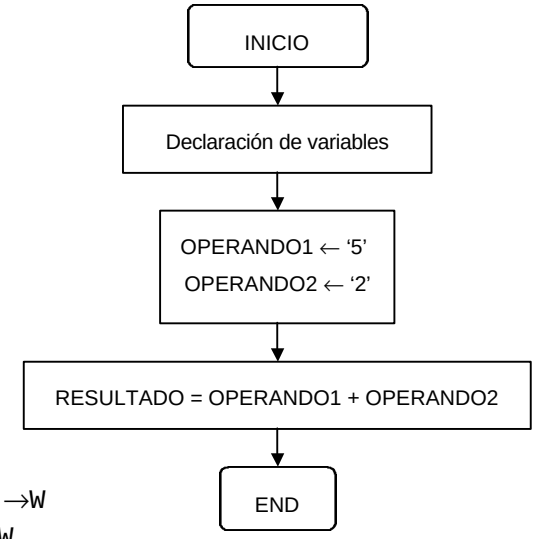

Figura 6-20. Diagrama de flujo del ejemplo 1.

Ejemplo 6.2. Confeccionar un programa para el microcontrolador PIC 16F84 que lea el número binario introducido por los interruptores y lo visualice en los diodos led (Figura 6-21).

El PIC16F84 dispone de dos puertas, PUERTA A (RA0-RA4) y PUERTA B (RB0-RB7). Cada una de las líneas de estas puertas puede actuar como entrada o como salida. El valor de los datos que entran o salen por cada puerta están materializados en dos posiciones de la RAM, PORTA para la puerta A y PORTB para la puerta B (figura 6.21). Existen otros dos registros de la memoria RAM (TRISA y TRISB) que se emplean para configurar las líneas de PORTA y PORTB como entrada o salida, de tal forma que cuando en uno de los bits se graba un '1', la línea correspondiente de la puerta afectada actúa como entrada, mientras que si se graba un '0', actúa como salida.

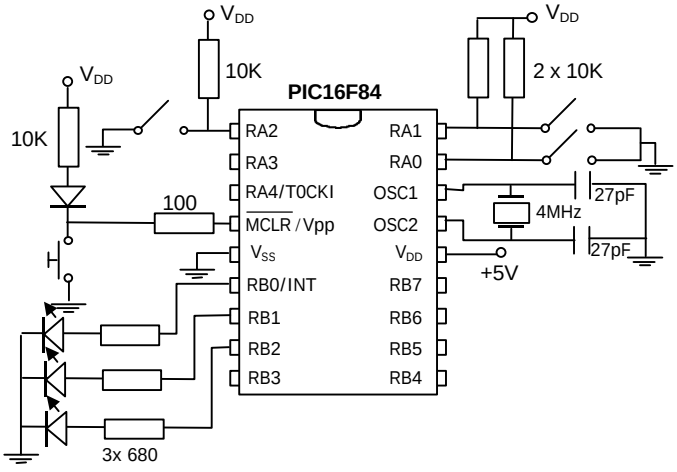

Figura 6-21. Esquema de conexión del ejemplo 2.

En las posiciones 5 y 6 (Figura 6-22) del banco 0 se ubican los registros de datos de las puertas y en las mismas posiciones de la memoria, pero en el banco 1, se encuentran los registros de configuración correspondientes. Si se desea acceder a posiciones del banco 1 hay que poner a '1' el bit 5 (RP0) de un registro llamado STATUS.

Por ejemplo si se quisiera configurar como entradas todas las líneas de la puerta A y como salidas las de la puerta B, habría que direccionar el banco 1 (bit 5 de STATUS='1'), cargar con '1' los bits de TRISA y con '0' todos los bits de TRISB. Una vez configuradas las puertas habrá que volver a poner a '0' el bit 5 del registro STATUS (volver al banco 0) para poder leer la información introducida por las líneas configuradas como entradas o enviar por las líneas configuradas como salidas.

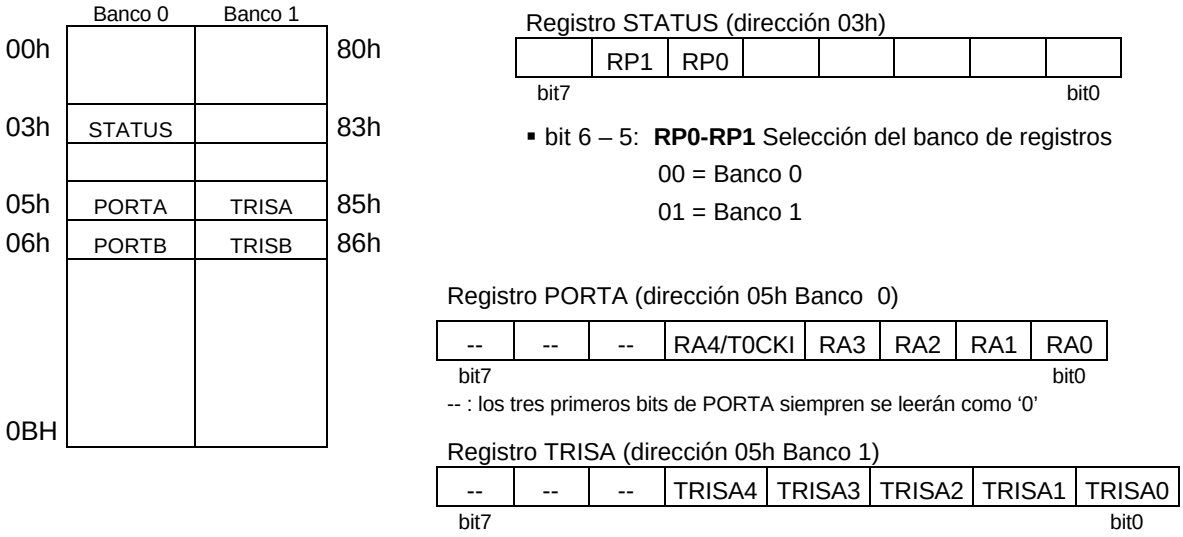

Figura 6-22. Registros STATUS, PORTA y TRISA.

En el esquema de conexión mostrado en la figura 6.20 se puede observar que cuando se cierre un interruptor se aplicará un '0' a la entrada correspondiente del microcontrolador y, si este valor se traslada a la salida el diodo led unido a la misma se apagará. Cuando se abra un interruptor, se aplicará un '1' en la entrada y el diodo led se encenderá.

Aparecen en este ejemplo algunas instrucciones nuevas:

- ***bsf f,d*** : pone a '1' el bit *b* del registro *f* .
- ***bcf f,d*** : pone a '0' el bit *b* del registro *f* .
- ***goto etiqueta*** : salta hasta la instrucción que va precedida por la etiqueta.

A continuación se expone el código ensamblador del programa y el diagrama de flujo (Figura 6-23).

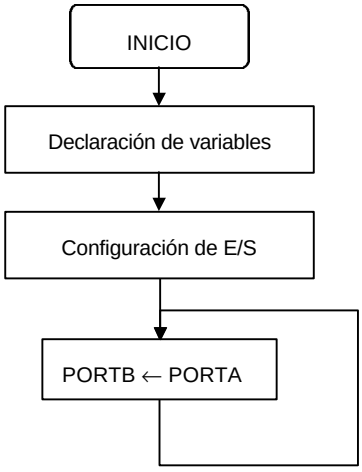

Figura 6-23. Diagrama de flujo del ejemplo 2.

```
       List  p=16f84
STATUS equ   0x03
PORTA  equ   0x05
PORTB  equ   0x06
       org   0x00
       goto  INICIO
       org   0x05        ;salta el vector de interrupción
INICIO
       bsf   STATUS,5    ;cambia al banco1
       movlw b'00000000' ;W ← '00'
       movwf PORTB       ;TRISB←W (PORTB salidas)
       movlw b'00011111' ;W ← '1F'
       movwf PORTA       ;TRISA←W (PORTA entradas)
       bcf   STATUS,5    ;cambia al banco0
BUCLE
       movf  PORTA,W     ;W ← PORTA
       movwf PORTB       ;PORTB ← W
       goto  BUCLE
       end
```

Los valores codificados en binario se escriben entre comilla simple y precedidos de la letra *b*.

Ejemplo 6.3. Confeccionar un programa para el PIC 16F84 que explore el estado de un interruptor conectado a la línea RA0 del puerto A (Figura 6-24) y que ilumine el diodo led conectado a la línea RB0 del puerto B cuando el interruptor se halle cerrado.

Este ejemplo incorpora la instrucción:

- ***btfsc f,d*** : salta una instrucción si el bit *b* del registro *f* es '0', si no, sigue por la siguiente instrucción.
- ***clrf f*** : borra el contenido del registro *f*.

A continuación se expone el código ensamblador del programa y el diagrama de flujo (Figura 6-25).

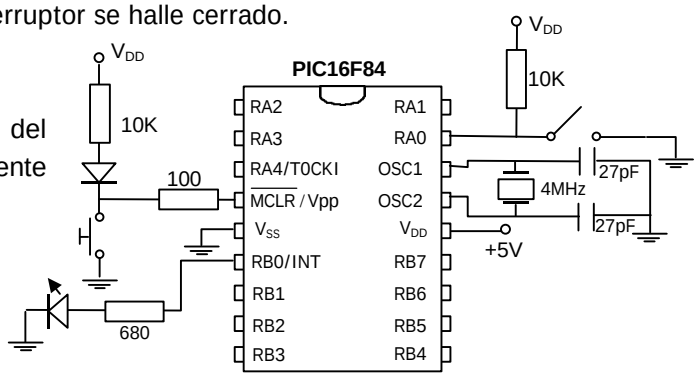

Figura 6-24. Esquema de conexión del ejemplo 3.

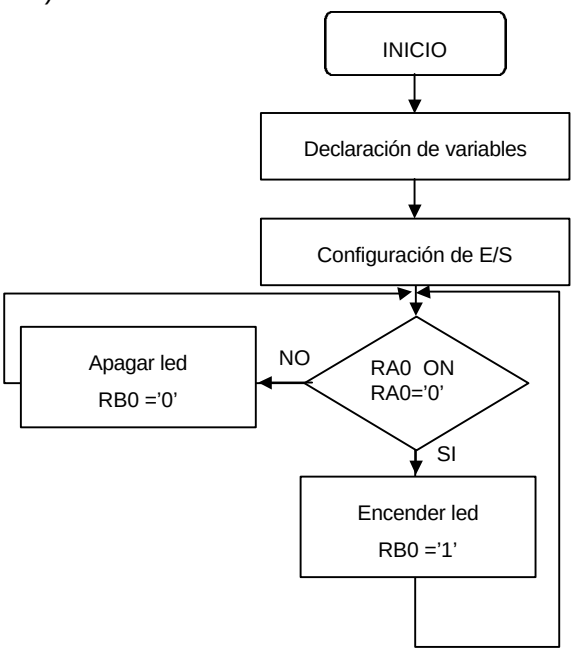

Figura 6-25. Diagrama de flujo del ejemplo 3.

```
       List  p=16f84
STATUS equ   0x03
PORTA  equ   0x05
PORTB  equ   0x06
       org   0x00
       goto  INICIO
       org   0x05          ;salta el vector de interrupción
INICIO
       bsf   STATUS,5      ;cambia al banco1
       clrf  PORTB         ;Puerto B configurado como salidas
       movlw b'00011111'   ;Puerto A configurado como entradas
       movwf PORTA
       bcf   STATUS,5      ;cambia al banco0
BUCLE
      btfsc  PORTA,0       ;¿RA0=0? salta si RA0 es '0'
      goto   RA0_es_1
      bsf    PORTB,0       ;pone a '1' RB0, enciende el led
      goto   BUCLE
RA0_es_1
      bcf    PORTB,0       ;pone a '0' RB0, apaga el led
      goto   BUCLE
      end
```

Ejemplo 6.4. Confeccionar un programa para el PIC 16F84 que comience poniendo a cero un contador y lo vaya incrementando de unidad en unidad hasta alcanzar el valor 5Fh (95), momento en el cual se encenderá un led conectado a RB0 y terminará el programa.

No existe ninguna instrucción que nos permita comparar el contenido de los registros de la memoria. Para solucionar este problema lo que haremos será restar al contenido del registro el valor con el que queremos compararlo y analizar el resultado de dicha operación.

Si el resultado de la resta es cero los dos valores son iguales y si por el contrario el resultado es distinto de cero los valores comparados son distintos. Existe un bit (Z) en el registro STATUS (figura 6.25) para indicar cuando una operación aritmética o lógica realizada en la ALU da como resultado cero.

Registro STATUS (dirección 03h)

					Z		
bit7							bit0

- bit 2 : **Z** bit de cero
  1 = El resultado de la operación es cero
  0 = El resultado de la operación no es cero

Figura 6-26. Bit Z del registro STATUS.

Las instrucciones nuevas que se van a emplear son:

- ***incf f,d*** : incrementa en una unidad el contenido del registro *f* y deposita el resultado en W si *d* = 0 (o W), o en *f* si *d* = 1.
- ***subwf f,d*** : resta el contenido de W al contenido de *f* y deposita el resultado en W si *d* = 0 (o W), o en *f* si *d* = 1.
- ***sublw k*** : resta el valor de *k* al contenido de W y almacena el resultado en W.

A continuación se expone el código ensamblador del programa y el diagrama de flujo (Figura 6-27).

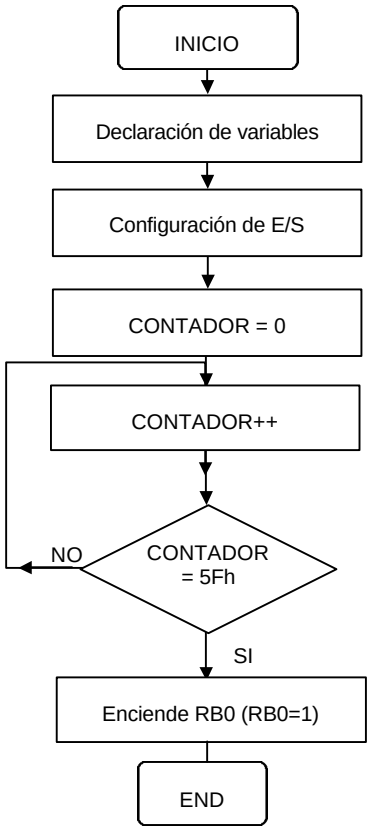

Figura 6-27. Diagrama de flujo del ejemplo 4.

```
List      p=16f84

STATUS    equ   0x03
PORTB     equ   0x06
CONTADOR  equ   0x0C

          org   0x00
          goto  INICIO
          org   0x05        ;salta el vector de interrupción

INICIO
          bsf   STATUS,5    ;cambia al banco1
          bcf   PORTB,0     ;RB0 salida, TRISB.0=0
          bcf   STATUS,5    ;cambia al banco0
          bcf   PORTB,0     ;apaga el led (RB0=0)
          clrf  CONTADOR    ;inicializa CONTADOR =0

BUCLE
          incf  CONTADOR,1  ;incrementa CONTADOR

          movwl 0x5f        ;W ← ‘5F’
          subwf CONTADOR,W  ;resta el valor del contador a W
          btfss STATUS,2    ;salta si Z=1(el resultado de la resta es 0)
          goto  BUCLE       ;salta a BUCLE si no son iguales
          bsf   PORTB,0     ;enciende el led (RB0=1)
          end
```

Ejemplo 6.5. Confeccionar un programa para el PIC 16F84 trabajando a 4MHz que produzca una señal cuadrada de periodo 50 ms (25 ms a nivel alto y 25 ms a nivel bajo) en el pin RB0.

Para generar la señal cuadrada en la patilla del microcontrolador RB0 procederemos de la siguiente manera: pondremos el bit RB0 a '1' ejecutaremos un rutina (subprograma) encargada de realizar un retardo de 25 ms; pondremos de nuevo el bit RB0 a '0' y volveremos a ejecutar la subrutina anterior. Este proceso se repetirá indefinidamente.

Una de las funciones más habituales en los programas de control suele ser determinar intervalos concretos de tiempo. También suele ser frecuente contar impulsos producidos en el exterior del sistema. En el microcontrolador PIC16F84 estas funciones la realiza un temporizador/contador de 8 bits, llamado TMR0 que actúa de dos maneras distintas:

- Como contador de sucesos, representados por los impulsos que se aplican a la patilla RA4/T0CKI. Al llegar al valor FFh se desbordará el contador y, con el siguiente impulso pasa a 00h, advirtiendo esta circunstancia activando un señalizador y/o provocando una interrupción.
- Como temporizador, se incrementa con cada ciclo de instrucción ($4/F_{reloj}$),o divisores del mismo, hasta que se desborda (pasa de 00h a FFh) y avisa poniendo a '1' un señalizador y/o provocando una interrupción.

Para que el TMR0 funcione como contador de impulsos aplicados en RA4/T0CKI hay que poner a '1' el bit T0CS, que es el que ocupa la posición 5 del registro OPTION (figura 6.27). Para que el TMR0 funcione como temporizador el bit T0CS debe ponerse a '0'.

TMR0 es un registro de propósito especial ubicado en la posición 01h del banco 0 de la memoria de datos RAM. En igual dirección pero en el banco 1 se encuentra el registro de configuración OPTION.

El tiempo de la temporización se calcula a partir del periodo de la señal de reloj (T_{reloj}), el valor de un divisor de frecuencia definido en el registro OPTION y el valor del temporizador TMR0.

$$\text{Temporización} = 4 \cdot T_{reloj} \cdot (255 - \text{TMR0}) \cdot \text{Divisor}$$

$$255 - \text{TMR0} = \text{Temporización} / (4 \cdot T_{reloj} \cdot \text{Divisor})$$

Si se desea temporizar 25ms (25000 μs) y asignamos al divisor de frecuencia el valor 128 (PSA ='0', PS2 = '1', PS1 ='1', PS0 = '0'), el temporizador TMR0 deberá contar 195 eventos:

$$255 - \text{TMR0} = 25 \cdot 10^{-3} / (1 \cdot 10^{-6} \cdot 128) = 195{,}3$$

El valor que hay que cargar en el temporizador TMR0 será el complemento de 195, es decir, 255-195= 60, que equivale al código hexadecimal 0x3c.

Registro OPTION (dirección 01h, Banco1)

		T0CS	T0SE	PSA	PS2	PS1	PS0
bit7							bit0

- bit 5 : **T0CS** bit de selección de la fuente de reloj
  1 = Transición en el pin RA4/T0CKI
  0 = Ciclo de instrucción interno
- bit 3 : **PSA** bit de asignación del divisor
  1 = Divisor asignado al WDT (watch dog)
  0 = Divisor asignado a TMR0
- bit 2-0 : **PS2:PS0** bit de selección del divisor

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

Figura 6-28. Registro OPTION.

Al cargar en el registro TMR0 el valor 60, éste se incrementará cada 128µs (ciclo definido por: Divisor · 4 · T_{reloj}). Cuando llegue al final de la cuenta (255), se habrá incrementado 195 veces, generando un retardo de aproximadamente 25ms (195 · 128µs). En ese instante se pondrá ‘1’ el bit T0IF del registro INTCON (registro de propósito especial 0x0B del banco 0, Figura 6-29).

Registro INTCON (dirección 0Bh, Banco 0)

GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit7							bit0

- bit 2 : **T0IF** flag de desbordamiento de TMR0
  1 = Desbordamiento de TMR0
  0 = TMR0 no se ha desbordado

Figura 6-29. Registro INTCON.

Aparecen en el código las siguientes instrucciones:

- ***call k*** : guarda la dirección de la instrucción actual y salta a la dirección *k* (donde se encuentra la subrutina).
- ***return*** :retorna a la dirección almacenada. Una vez ejecutada la subrutina el programa vuelve a la posición de la memoria de programa posterior a la llamada *call*.

A continuación se expone el código ensamblador del programa y el diagrama de flujo (Figura 6-30).

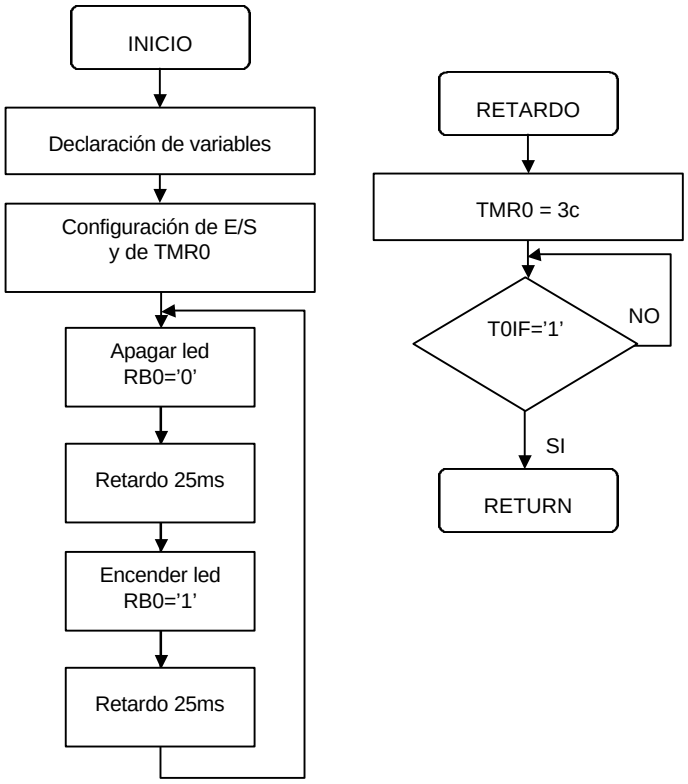

Figura 6-30. Diagrama de flujo del ejemplo 5.

```
       List  p=16f84
STATUS equ   0x03
PORTB  equ   0x06
TMR0   equ   0x01
INTCON equ   0x0B
       org   0x00
       goto  INICIO
       org   0x05

INICIO
       bsf   STATUS,5     ;cambia al banco1
       clrf  PORTB        ;PORTB salidas
       movlw b’00000110   ;configuración OPTION
       movwf TMR0
       bcf   STATUS,5     ;cambia al banco0
BUCLE
       bcf   PORTB,0      ;RB0 = ‘0’
       call  RETARDO      ;retardo
       bsf   PORTB,0      ;RB0 = ‘1’
       call  RETARDO      ;retardo
       goto  BUCLE

RETARDO
       bcf  INTCON,2      ;desconecta el flag de desbordamiento
       movlw 0x3c
       movwf TMR0         ;carga TMR0
EXPLORA
       btfss INTCON,2     ;¿Se ha desbordado TMR0?
       goto  EXPLORA      ;bucle hasta que TMR0 llegue a 256
       return
       end
```