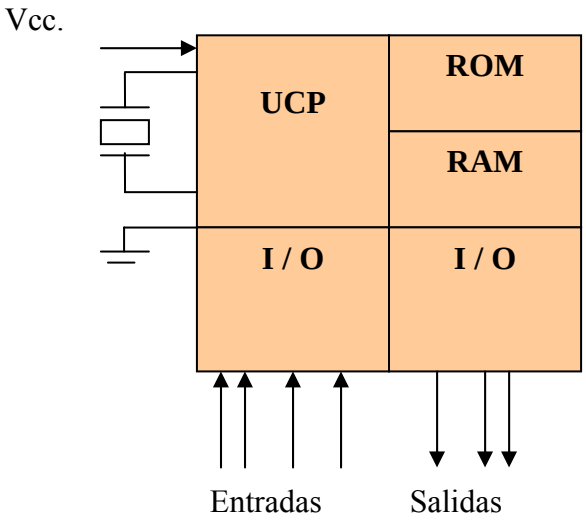
CAPITULO 1: Generalidades, especificaciones y conexionados externos del microcontrolador

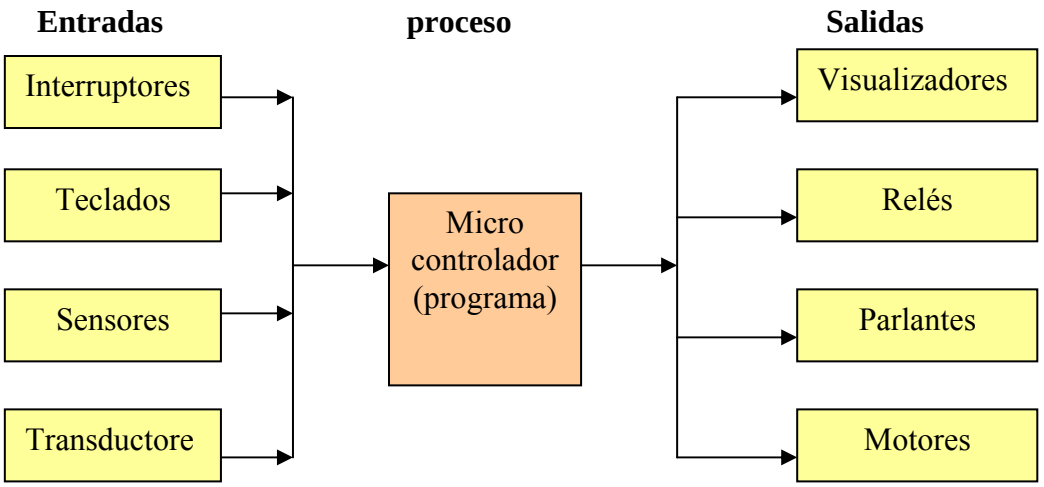
INTRODUCCION

Los microcontroladores son circuitos integrados “programables”, que contienen todos los elementos de un computador. Este componente electrónico, ha revolucionado, en los últimos años, las técnicas de diseño, en lo referente a “sistemas de control industrial”.
Está diseñado para controlar sistemas que realizan una tarea especifica. Como esta integrado en una sola pastilla (chips), de reducido tamaño, suele estar incorporado al propio dispositivo que gobierna. Podemos decir que es un “computador completo”, con limitaciones en sus prestaciones.
Aplicaciones comunes: Hornos microondas, lavarropas, sistema de inyección de automóviles, teclados de PC, impresoras, videos, sistemas de comunicaciones, procesos industriales etc.

Esquema general interno



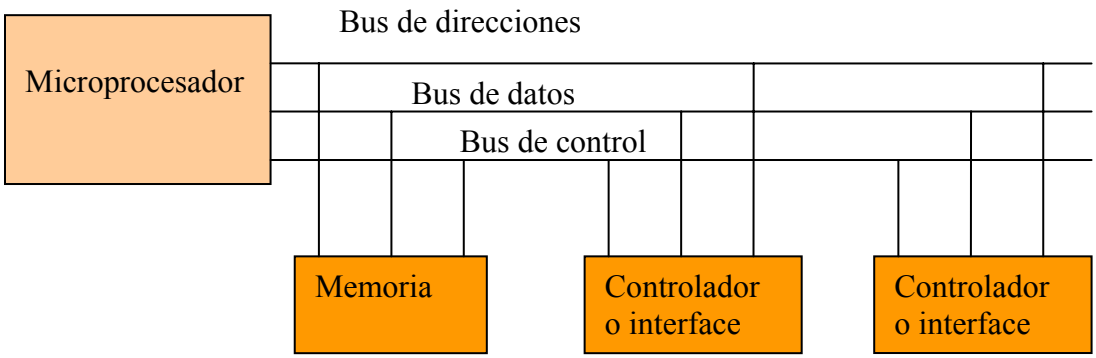
Esquema general de un sistema con microcontrolador



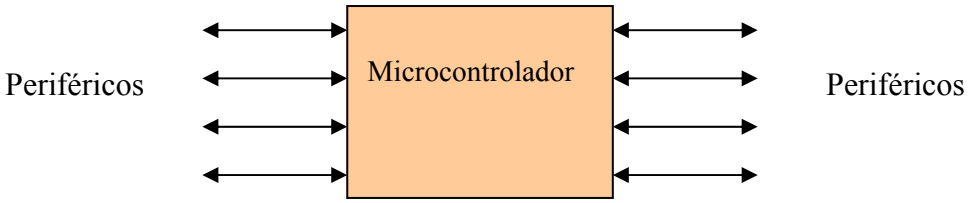
Diferencias entre sistemas basados con microprocesador y con microcontrolador:

El microprocesador, es un circuito integrado que contiene a "la unidad central de proceso(UCP) de un computador. Decimos entonces que el microprocesador es un "sistema abierto", con el que puede construirse, un computador, con las caracteristicas que se desee, acoplando los modulos necesarios, para el sistema a controlar.El microprocesador puede sacar al exterior, las lineas de sus buses de direcciones, datos y control.
El microcontrolador, es un "sistema cerrado", dado que no son accesibles las líneas de los buses de direcciones, datos y control (salvo casos especiales de microcontroladores). El microcontrolador, tiene todos los elementos de un computador, con prestaciones limitadas que no se pueden modificar. Las entradas y salidas de un microcontrolador, están adaptadas eléctricamente a los respectivos periféricos.

Sistema con microprocesador



Sistema con microcontrolador



Tenemos algunas diferencias importantes entre los sistemas basados con microprocesador y microcontrolador. Cada uno tiene sus ventajas y desventajas, dependiendo de las necesidades de cada aplicación. Enumeraremos algunas de ellas:

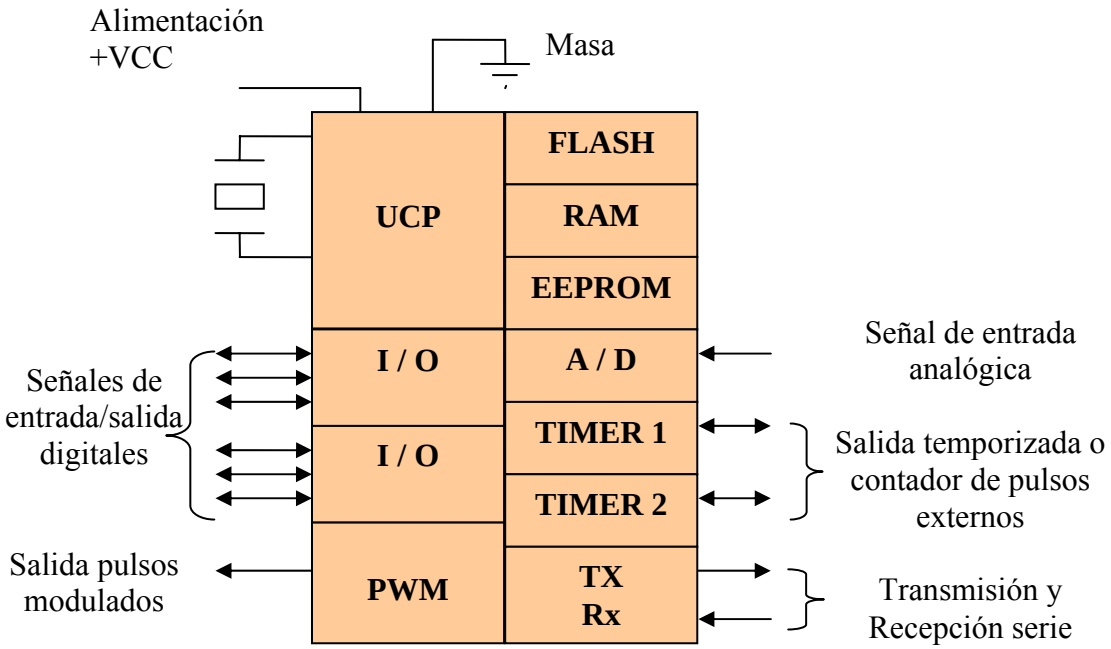
1)- **La UCP de los microcontroladores es más simple** y sus instrucciones están orientadas, fundamentalmente, a la operación de cada una de las líneas de entrada / salida.
2)- **La memoria RAM de datos de los microcontroladores, es de baja capacidad**. La razón es que para aplicaciones de control e instrumentación normales, no se necesita almacenar gran cantidad de información temporal. En cambio en los microprocesadores, pueden acceder a través de los buses, a grandes bancos de memoria RAM externa de acuerdo a las necesidades del sistema.
3)- **En los microcontroladores, la memoria ROM de programa, es limitada**. Por lo general no mayor a 4 K x instrucciones. En un sistema con microprocesador, se pueden tener ROM externas de acuerdo a las necesidades del sistema.
4)- **Con los microcontroladores, no es necesario diseñar circuitos complejos decodificadores, porque el mapa de memoria y de puertos I / O, están incluidos internamente.**

5)- **La mayoría de los microcontroladores, no tienen accesibles al usuario, los buses de direcciones, datos y control de la UCP**. Algunos modelos, lo pueden hacer a través de los puertos I / O, para construir expansiones de memoria RAM y ROM. En los microprocesadores, la expansión es más fácil.
6)- **La velocidad de operación de los microcontroladores es más lenta**, de la que se puede operar con los sistemas con microprocesadores. Sin embargo, actualmente existen microcontroladores que operan por encima de los 50 MHZ.
7)- De manera similar a los sistemas basados con microprocesadores, para escribir, ensamblar y depurar programas en lenguaje de maquina, los microcontroladores necesitan un sistema de desarrollo para cada familia de microcontroladores. Éstos, están compuestos por un paquete "software" con editor de textos, ensamblador y simulador de programas y al mismo tiempo, se necesita de un "hardware", para poder almacenar el "programa " de aplicación", en la memoria ROM del microcontrolador.
Resumiendo, podemos decir que algunas de las principales ventajas de los microcontroladores son:
a)- El circuito impreso es más pequeño dado que muchos de los componentes se encuentran dentro del circuito integrado.
b)- El costo del sistema es reducido, dado que es reducido él número de componentes.
c)- Los problemas de ruido eléctricos que pueden afectar a los sistemas con microprocesador, se eliminan, debido a que todo el sistema principal, se encuentra en un solo encapsulado.
d)- El tiempo de desarrollo de un sistema con microcontrolador, se reduce notablemente. Cuando una aplicación sobrepasa las características del microcontrolador como capacidad de memoria, velocidad de proceso, número de entradas y salidas, etc., se debe recurrir a un sistema con microprocesador o una computadora completa.

Recursos disponibles de los microcontroladores

Existen muchas aplicaciones que requieren solamente entradas y salidas de tipo digital. Por ello, muchos de los microcontroladores, disponen internamente de algunos circuitos especiales, para atender a diversas situaciones y naturaleza de las entradas y salidas:
a)- Si los fenómenos que se necesitan medir o controlar, son de naturaleza analógica, como los casos de temperatura, presión, voltaje, etc... Se debe entonces disponer de un conversor analógico/digital de varios canales.
b)- Si es necesario medir periodos de tiempo, o generar temporizaciones en las salidas, tonos o frecuencias, se debe contar con uno o más contadores programables (timer).
c)Si la información obtenida en un proceso de medida o control, o los resultados de los cálculos del programa del programa, se deben enviar a otro microcontrolador o a una computadora o a una red, es conveniente que el microcontrolador posea un circuito de comunicaciones (RS232, I2C, USB, etc.).
d)- Existen sistemas que requieren sistemas de control por ancho de pulso PWM como el caso de motores, cargas resistivas etc. Para este caso, hay disponibles microcontroladores con módulos PWM.
e)- Para aquellos eventos que actúan en tiempo real o existen procesos que no dan "espera", se deben utilizar la técnica llamada "interrupciones". Cuando una señal externa activa una línea de interrupción, el microcontrolador deja de lado la tarea que se encuentra ejecutando, para atender una situación especial y luego puede regresar a continuar con la labor que esta realizando.

Bloques internos principales y auxiliares de los microcontroladores

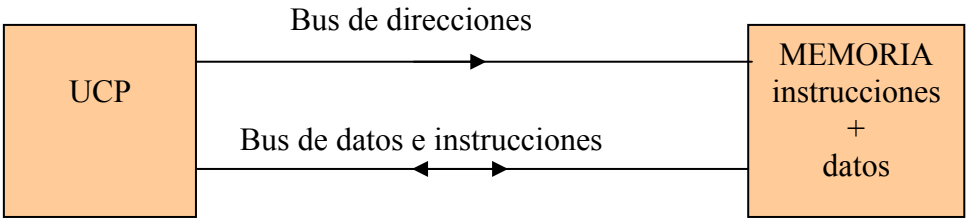


Partes principales:
-Unidad central de proceso (UCP)
-Memoria no volátil para guardar el programa, por ejemplo EPROM (hay varios tipos)
-Memoria no volátil para guardar datos, por ejemplo EEPROM.
-Memoria de lectura / escritura para guardar datos.
-Registros generales y especiales para programación.
-Líneas de entrada / salida para los controladores periféricos con comunicación paralela
-Líneas de entrada / salida para comunicación serie con periféricos (232C.I2C,USB,etc.)

Recursos auxiliares:
-Circuito reloj (oscilador para sincronismo)
-Temporizadores (timer)
-Perro guardián (watch dog)
-Conversores analógico / digital (A /D) y viceversa (D/A).
-Comparadores analógicos.
-Protección ante fallas de alimentación.
-Estado de bajo consumo o reposo.

Arquitecturas empleadas en los microcontroladores

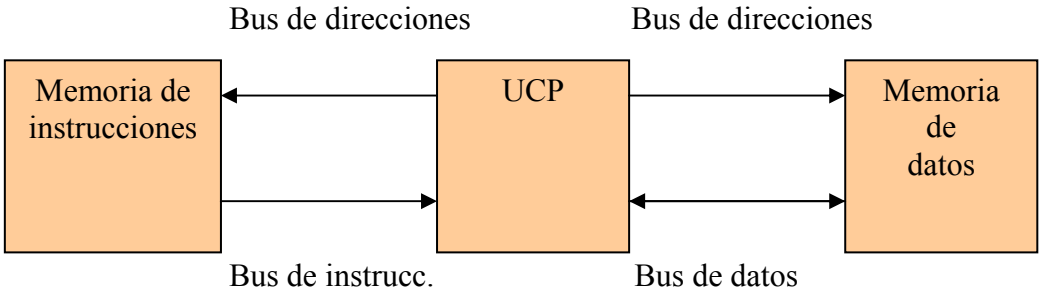
Arquitectura de Von Neuman.



Ésta es la arquitectura de los grandes computadores y las PC. Tenemos un bus de control, un bus de “datos e instrucciones” que es compartido por los datos y las instrucciones del programa

en ejecución. Además, tanto los datos como las instrucciones, comparten el uso de la memoria principal, lógicamente en distintas áreas.
Esta arquitectura se utilizó en los primeros microcontroladores.

Arquitectura Harvard



Esta arquitectura, es la utilizada por los modernos microcontroladores. En ella, son independientes la memoria de instrucciones y la memoria de datos. Cada una, dispone de su propio sistema de bus de direcciones.
Otra característica, esta en la UCP. Ésta responde a la arquitectura RISC (computadoras con juego de instrucciones reducido), identificada por poseer un juego de instrucciones de máquina pequeña y simple, de tal forma que la mayor parte de las instrucciones, se ejecuta en un ciclo de instrucción.
Otra característica es la segmentación del procesador (pipe line) que permite procesar en etapas, las instrucciones para trabajar con varias a la vez.
El alto rendimiento y elevada velocidad de los modernos microcontroladores, se deben a la aplicación de las arquitecturas Harvard, Risc y a la segmentación (pipe line) de las instrucciones.

La memoria de programa

El microcontrolador, esta diseñado para que en su memoria de programa, se almacenen todas las instrucciones del programa de control. En Gral., no se pueden utilizar memorias externas para su ampliación. Como el programa es siempre el mismo, éste se graba en forma permanente.
Los tipos de memoria que admiten la retención de lo grabado, son cinco versiones:
1)- ROM con mascara: Se graba el chip durante el proceso de fabricación; se justifica ésta memoria cuando se deben grabar grandes cantidades de microcontroladores.
2)- EPROM: Se graba con un dispositivo físico (circuito electrónico) gobernado por una PC, denominado “grabador”. En la superficie de la cápsula, tiene una ventana de cristal para borrarla con rayos ultravioleta y volverla a utilizar.
3)- OTP: Estas memorias se graban una sola vez, por el usuario, y no se pueden borrar. Son de bajo precio y fáciles de grabar. Se justifica para prototipos finales y series cortas.
4)- EEPROM: Se graban en forma similar a las OTP y EPROM y se borran en forma similar a la grabación, o sea eléctricamente sobre el mismo zócalo del grabador. Puede ser programada y borrada aprox. 1.000.000 de veces. La capacidad de memoria es limitada, con tiempo de grabado relativamente alto y elevado consumo de energía. Por ejemplo el microcontrolador PIC 16C84 puede almacenar en su memoria de programa EEPROM, 1 K de palabras de 14 bits y algunos bytes de datos, sin pérdida de la información cuando se interrumpe la tensión de alimentación.
5)- FLASH: Es una de las últimas versiones de memoria no borrables. Es de bajo consumo con posibilidad de escribir y borrar (aprox. 1000 veces) como la EEPROM pero de mayor capacidad. Por sus mejores prestaciones, están desplazando a las EEPROM.
Son recomendables en aplicaciones que sea necesario modificar el programa a lo largo de la vida del producto a controlar sea por desgaste, optimización etc. Por ejemplo la empresa Microchip T. Comercializa los microcontroladores PIC. Dentro de esta familia estén los PIC 16C84 con memoria de programa EEPROM y los PIC 16F84 con memoria FLASH.
Ambos microcontroladores, son similares en sus prestaciones. La memoria FLASH, es una variante de las EEPROM.

La memoria de datos
Ésta memoria debe ser de lectura / escritura (L/E) por lo que la memoria "RAM estática" (SRAM), es la mas adecuada aunque sea volátil al eliminar la tensión de alimentación. Hay microcontroladores que utilizan para los datos dos memorias: una EEPROM y otra SRAM. Por ejemplo el PIC 16F84 tiene 68 bytes de memoria SRAM para datos y 64 bytes de memoria EEPROM, también para datos. La memoria de programa para estos chips, es de 1 K x 14 bits. De tipo EEPROM.

Líneas o puertos de entradas y salidas (I/O)
Están destinadas a soportar los periféricos exteriores que controlan. Son de ambos sentidos, es decir que pueden actuar como entradas o salidas según se las programe y se adaptan con los periféricos, manejando información paralela; se agrupan generalmente en grupos de 8 bits, denominándose el conjunto "Puertas". La actuación de estas puertas es la de suministrar corriente eléctrica en el estado binario alto, con el nivel de tensión aprox. Al de la fuente de alimentación, y absorber corriente en el estado binario bajo.
Existen modelos que soportan comunicación serie, otros disponen de líneas para diversos protocolos de comunicación como I2C, USB etc.
Otros terminales de un microcontrolador son dos entradas para alimentación de energía eléctrica (VDD (+) y Vss (-); una entrada para el "reinicio" o "reset"(MCLR#) y dos entradas para el oscilador externo (osc1/CLKIN y osc2/CLKOUT); una entrada para interrupción.

PRINCIPALES FAMILIAS DE MICROCONTROLADORES

Detallaremos brevemente las principales o más renombradas familias de microcontroladores. Existen en el mercado varias marcas reconocidas por sus características, comercialización, soporte técnico, difusión, usos en la industria etc. Entre ellas tenemos INTEL, MOTOROLA, MICROCHIP, PHILLIPS, NATIONAL y ATMEL.

Familia Intel 8051: El primer microcontrolador fue el 8048 con 8bits de datos, con RAM interna, pero la memoria de programa era externa. En los años 80 nació el 8051, siendo el más difundido a nivel mundial. El 8051 tiene 4 Kbytes de ROM que deben programarse durante su construcción. El 8751 reemplazo la ROM por una EPROM. El 8031 no tiene ROM interna; el programa reside en memoria externa. Para la comunicación con la memoria, utiliza 3 de los cuatro puertos entrada / salida. Esta posibilidad de expansión es característica de esta familia.

Familia Motorola: Derivaron del microprocesador 6800, siendo optimizados para aplicaciones de control especializado, formando parte de aparatos de producción masiva como juguetes, equipos de video, impresoras, electrodomésticos y tienen amplia aplicación en la industria automotriz. Existen cinco familias principales: La 68H05, 68HC08 y 68HC11 de 8 bits; la 68HC12 y 68HC16 son de 16 bits, cada una de ellas con diferente UCP. Por ejemplo la 68hc05,representa a mas de 30 microcontroladores distintos con la misma UCP y de 8 bits.. Éstos incluyen RAM, ROM, puertos I/O, temporizadores, convertidores A/D y memorias PROM o EPROM.

Familia Microchip: Estos microcontroladores tienen arquitectura Hardvard. Se clasifican en tres grupos, dependiendo de la longitud de palabra de instrucción que pueden manejar (12,14 o 16 bits), tomando las referencias 12XXX, 16XXXX, 17XXX y 18XXX. Los fabricantes los definen a los PIC como microcontroladores de 8 bits tipo RISC. Son de bajo costo poco consumo y alta velocidad de operación.

Familia ATMEL: Manejan 3 grandes grupos de microcontroladores RISC, cuyas UCP, llegan hasta los 32 bits. El 1° grupo tiene la arquitectura basada en el 8051 con memoria de programa FLASH. El 2° grupo es el AT91, los cuales soportan compilados en lenguaje "C", ensamblador etc. El 3° grupo, AVR", son arquitectura RISC y UCP de 8 bits y módulos de comunicación USART, SIP, ADC, etc.

Microcontroladores Basic Stamp: Toman como base el microcontrolador PIC los cuales forman un sistema soportados en una placa principal, que les permite programarlos en lenguaje

"Basic Stamp", siendo éste más sencillo que otros (lenguaje de alto nivel).El fabricante de estos sistemas es PARALLAX INC.

CARACTERISTICAS ESPECIFICAS DEL MICROCONTROLADOR PIC16X84

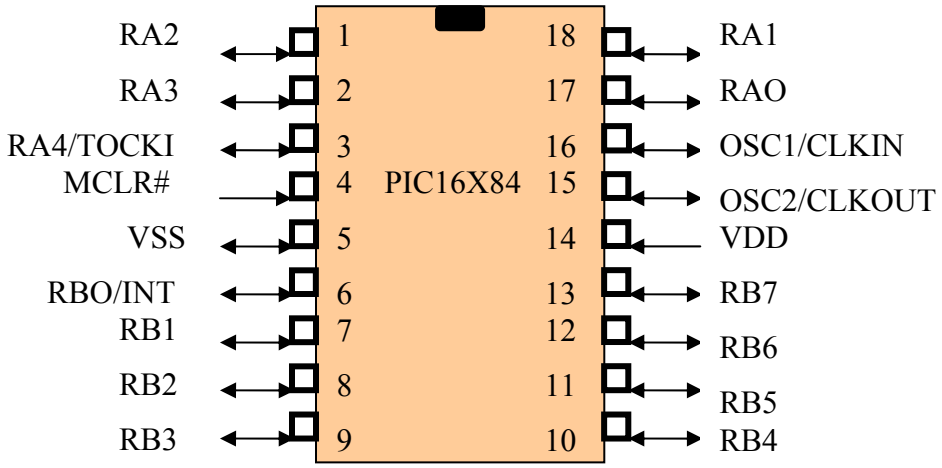
El desarrollo de este curso sobre introducción a los microcontroladores, tomara como base al microcontrolador PIC16X84, fabricado por la empresa Microchip. Su elección, esta basada en las siguientes consideraciones:
-Sencillez de su manejo
-Abundante información técnica de aplicación
-Buen promedio en los parámetros: velocidad, consumo, tamaño, alimentación, código compacto, etc.
Microchip, clasifica a sus microcontroladores en cuatro gamas a saber:

1º **Gama básica**: familias PIC 125xx y PIC 16C5x, con un repertorio de 33 instrucciones de 12 bits y dos niveles de pila
2º **Gama media**: Familias PIC 12C6xx, PIC 16Cxx y PIC 16F87x con 8 niveles de pila, 1 vector interrupción y 35 instrucciones de 14 bits.
3º **Gama alta**: PIC 17Cxx con 16 niveles de pila, 4 vectores de interrupción y 58 instrucciones de 16 bits.
4º **Gama mejorada**: PIC 18Cxxx con 32 niveles de pila, 4 vectores de interrupción y 77 instrucciones de 16 bits.
Para el caso especifico del PIC 16F84 que vamos a trabajar, dispone de 8 niveles de pila, 1 vector de interrupción y 35 instrucciones de 14 bits (gama media)

Partes principales y características eléctricas generales del PIC16X84

-**Tecnología de fabricación:** Circuito integrado CMOS, técnica epitaxial
-**Encapsulado** plástico con 18 terminales.
-**Unidad central de proceso.**
-**Memoria de programa**: 1K x 14 bits EEPROM (PIC16C84) y FLASH (PIC16F84)
-**Memoria de datos** RAM (GPR): 36 bytes (PIC16C84) y 68 bytes (PIC16F84)
-**Memoria de datos** EEPROM: 64 bytes para ambos modelos.
-**PILA** (stack): de 8 niveles (Memoria para subrutinas anidadas)
-**Interrupciones**: 4 tipos diferentes con 1 vector de interrupción.
-**Temporizadores**: uno solo, el TMRO, que puede actuar como temporizador de eventos o como contador de pulsos externos.
-**Perro guardián** (WDT) actúa para evitar que el microcontrolador quede "colgado" ante una falla temporal en la ejecución de las instrucciones.
Líneas de entrada / salida digitales: 13 en total, 5 en Puerta A y 8 en Puerta B.
-**Juego de instrucciones**: 35 (de 14 bits)
-**Corriente máxima absorbida**: 80 mA en Puerta A y 150 mA en Puerta B.
-**Corriente máxima suministrada**: 50 mA en Puerta A y 100 mA en puerta B.
-**Corriente máxima absorbida por línea**: 25 mA
-**Corriente máxima suministrada por línea**: 20 mA
-**Voltaje de alimentación**: (V_{DD}): de 2 a 6 volt. CC
-**Voltaje de grabación**: (V_{PP}): de 12 a 14 volt. CC
- **Protección contra fallo de alimentación**.
-**Entrada para RESET**.
- **Entradas para osciladores externos**.
-**Estado de reposo o bajo consumo**.
- **Registro de trabajo W.**
-**Registros de propósitos especial** (SFR) : Total 22 ubicados en la memoria RAM.
-**Direccionamiento directo e indirecto** de la memoria RAM

DIAGRAMA DE CONEXIONES

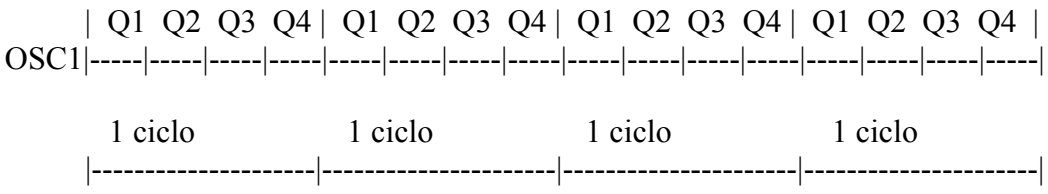


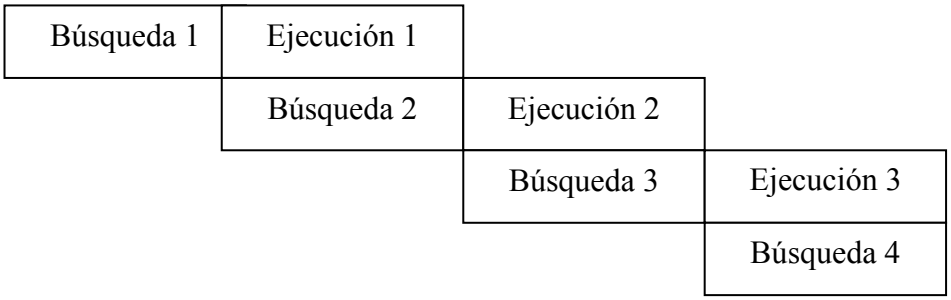
OSC1 / CLKIN: Entrada externa de los impulsos reloj o conexión con el cristal de cuarzo.
OSC2 /CLKOUT: Salida de Fosc/4 en modo osc. RC o conexión con el cristal de cuarzo.
MCLR#: En modo grabación se introduce la tensión VPP (12 a 14 V DC.).
En funcionamiento normal, es la entrada del "reset" del PIC.
RA0- RA3: Líneas de E / S de la puerta A (puerto A)
RA4 / TOCKI: Línea de E / S de la puerta A o entrada de impulsos de reloj para TMR0.
RB0 / INT: Línea de E / S de la puerta B (o puerto B) o de pedido de interrupción.
RB1-RB7: Líneas de E / S de la puerta B. (o puerto B)
VDD: Entrada tensión de alimentación (+)
VSS: Entrada tensión de alimentación (-)

Para la grabación de las instrucciones en la memoria de programa (EEPROM o FLASH), se ingresa la tensión de grabación (VPP=12 a 14 volt.) por el Terminal MCLR#, la señal reloj del sistema grabador por RB6 y los bits de las instrucciones en serie por la entrada RB7.

EL CIRCUITO OSCILADOR EN LOS MICROCONTROLADORES PIC

Como los microcontroladores son sistemas síncronos programables, necesitan una señal eléctrica con una frecuencia de funcionamiento fija, provista por un oscilador. Esta señal, ingresa a través del pin OSC1/CLKIN. Los pulsos que ingresan, se dividen internamente por cuatro, dando lugar a las señales Q1, Q2, Q3 y Q4. Las instrucciones del programa, requieren de estos cuatro periodos para ejecutarse, denominándose éste tiempo, periodo (ciclo) de instrucción. Por ejemplo para una frecuencia reloj de 10 MHZ el periodo resulta T =100 ns y el ciclo de instrucción es 4 x 100 = 400 ns.



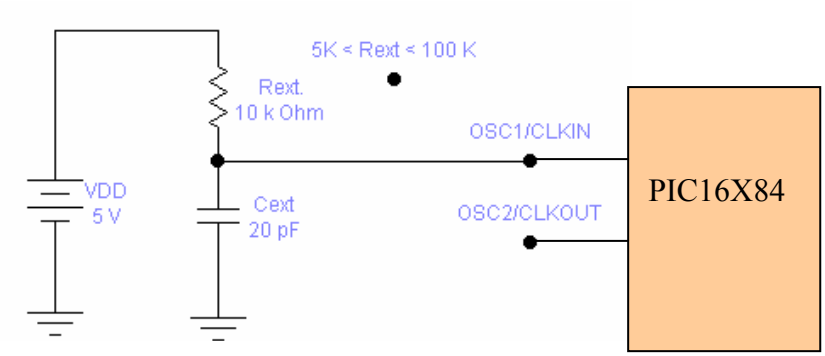


Las instrucciones simples, requieren para cumplirse de dos ciclos de instrucción. Las instrucciones de salto, necesitan cuatro ciclos.
Q1: Durante este tiempo, se incrementa el contador de programa.
Q2: Se busca el código de instrucción en la memoria de programa y se carga en el registro de instrucciones.
Q3 – Q4: Se produce la decodificación y la ejecución de la instrucción.
Como los microcontroladores PIC aplican la técnica de segmentación (pipe-line), que consiste en realizar en paralelo las dos fases que comprenden cada instrucción (búsqueda y ejecución), podemos decir que cada instrucción simple, se ejecuta en un tiempo de 1 ciclo de instrucción y las de salto, en 2 ciclos.

Tipos de osciladores

Los microcontroladores PIC admiten cinco tipos de osciladores externos para aplicarle la señal con la frecuencia de funcionamiento. El tipo de oscilador empleado, debe especificarse en dos bits (FOSC1 Y FOSC2) de la denominada “palabra de configuración” o registro de configuración, durante el proceso de grabación del programa, en la memoria de instrucciones del micro.
Los tipos de osciladores son los siguientes;
1)- Oscilador tipo RC.
2)- Oscilador RC interno (INTRC)
3)- Oscilador tipo LP
4)- Oscilador tipo XT
5)- Oscilador tipo HS

Oscilador tipo RC:
Este oscilador es de bajo costo, proporcionando una estabilidad en frecuencia mediocre. Se lo utiliza para aquellos casos donde los tiempos de funcionamiento (temporizaciones) no son exigentes.



Ejemplo de frecuencias de oscilación:

Fosc	Rex	Cex
625 Khz	10 K	20 pF
80 Khz	10 K	220 pF
80 Hz	10 K	0.1 uF

La resistencia exterior (Rex) varía entre 5 K y 100 K. Para valores menores de 5 K la oscilación se hace inestable y puede detenerse; para valores mayores de 100 K, se hace susceptible al ruido y la humedad.
Por el terminal OSC2/CLKOUT podemos obtener la frecuencia del oscilador dividida por cuatro para sincronizar dispositivos externos.

Oscilador RC interno (INTRC)

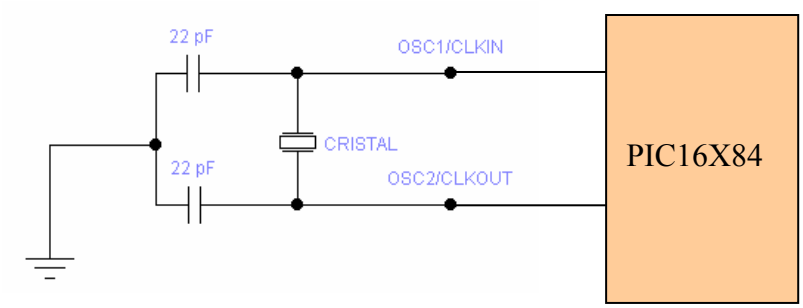
Es la solución más económica, no siempre disponible en todas las familias de PIC. En los PIC 16X84 no existe esta variante. La frecuencia de oscilación se genera internamente sin elementos externos.

Oscilador tipo LP:
Es un oscilador de bajo consumo y baja frecuencia con cristal de cuarzo o resonador cerámico. Esta diseñado para trabajar en frecuencias de 35 a 200 KHZ.

Oscilador tipo XT:
Es un oscilador de cristal de cuarzo o resonador cerámico tipo estándar para frecuencias comprendidas entre 100 KHZ y 4 MHZ. Tiene un consumo de energía medio.

Oscilador tipo HS:
Es un oscilador de cristal de cuarzo resonador cerámico con alta frecuencia, comprendida entre 4 y 10 MHZ. Tiene un consumo de energía alto.

Conexión del oscilador basado en cristal o resonador cerámico



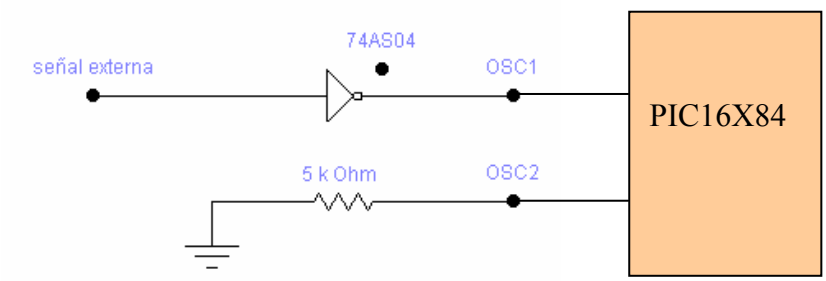
RESONADOR

MODO	FREC.	C1/C2
XT	455 KHZ 2,0 MHZ 4,0 MHZ	22 – 100 pf 15 – 68 pf 15 – 68 pf
HS	8,0 MHZ 16 MHZ	10 – 68 pf 10 –22 pf

CRISTAL

MODO	FRECUENCIA	C1	C2
LP	32 KHZ 200 KHZ	68 –100 pf 15 –30 pf	68 –100 pf 15 – 30 pf
XT	100 KHZ 2 MHZ 4 MHZ	68 – 150 pf 15 – 30 pf 15 –30 pf	150 – 200 pf 15 – 30 pf 15 –30 pf
HS	8 MHZ 10 MHZ 20 MHZ	15 –30 pf 15 – 30 pf 15 – 30 pf	15 – 30 pf 15 – 30 pf 15 –30 pf

Cuando el microcontrolador se configura en los modos LP, XT o HS, se puede utilizar una fuente externa para los pulsos reloj adaptada mediante una compuerta lógica y conectada al pin OSC1. Al pin OSC2 se le suele colocar una resistencia a masa para disminuir ruidos del sistema, pero a costa de incrementar la corriente del sistema.

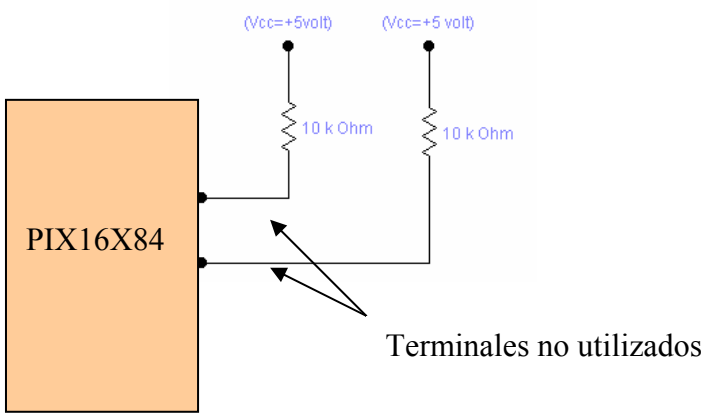


Caracteristicas de los puertos de entrada/salida de los microcontroladores PIC

Como lo hemos mencionado, el puerto “A” tiene 5 líneas conectadas a 5 terminales del ” chip” y el puerto “B” tiene 8 líneas conectadas a 8 terminales. Cada línea, puede ser configurada, por el programa grabado, como entrada o salida. Cada Terminal, tiene un resistor interno, conectado a la fuente de alimentación (pull-up) que puede ser conectado o desconectado, por el programa. Estos resistores se desconectan automáticamente, si un terminal se predispone como terminal de salida. Esto es asi debido a que las salidas tienen la posibilidad de actuar como fuente de corriente (entregan corriente) o como sumidero (absorben corriente). Todos los resistores de “pull-up” se conectan o desconectan al mismo tiempo (no existe un comando que los conecte independientemente.
Como salida, un terminal del puerto “A”, puede absorber 25 mA del circuito exterior o entregar 20 mA al circuito exterior, pero en total, no se debe exceder de 80 mA absorbidos y 50 mA entregados.
Para el puerto “B”, las características son similares por Terminal individual, pero en total no se puede exceder de los 150 mA absorbidos y 100 mA entregados.

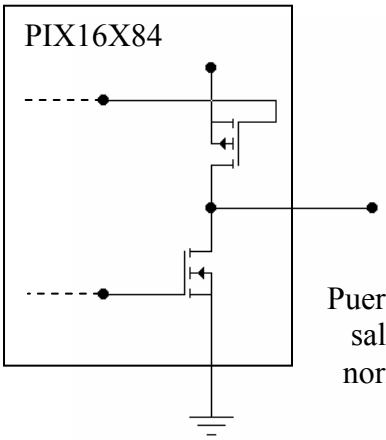
Terminales no utilizados

Los terminales de los puertos no utilizados, siempre se deben conectar a la fuente de alimentación (+5 volt) mediante un resistor de 10 KΩ, debido a que se trata de un dispositivo CMOS, caso contrario podría deteriorarse por captación electrostática.

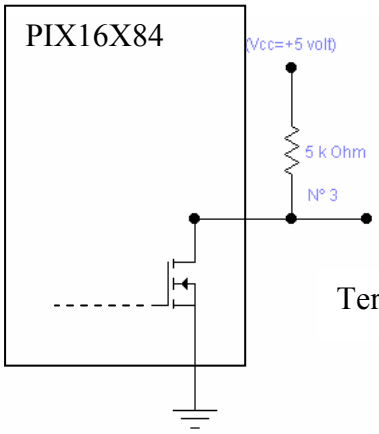


El Terminal nº 3

El Terminal nº 3 del circuito integrado, denominado RA4/TOCKI, puede ser configurado como entrada/ salida o como de arranque de un temporizador/contador. Cuando se programa como entrada, este Terminal funciona como un disparador de Schimitt trigger ideal para reconocer señales distorsionadas o de crecimiento lento. Cuando trabaja como salida lo hace “colector abierto” (drenador abierto), es decir que no se puede utilizarlo como fuente de corriente, en este caso siempre se debe colocar un resistor externo entre la fuente de alimentación y el Terminal, según se muestra en el dibujo:



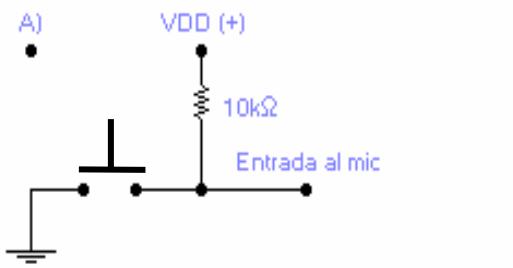
Puerto de salida normal

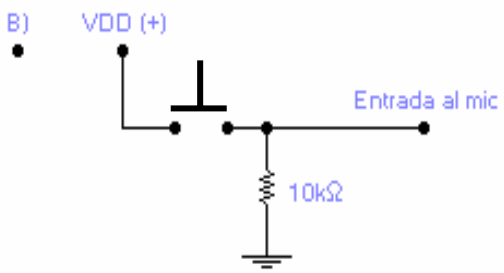


Terminal 3

PERIFÉRICOS DIGITALES PARA LAS ENTRADAS Y SALIDAS

Entradas:
En el primer programa que desarrollamos, introducimos los datos directamente. En la práctica el microcontrolador se comunica con el mundo exterior, a través de señales externas digitales o analógicas. De igual forma, son las señales de salida, que gobiernan el proceso controlado. Solamente analizaremos las señales digitales para el PIC16X84.

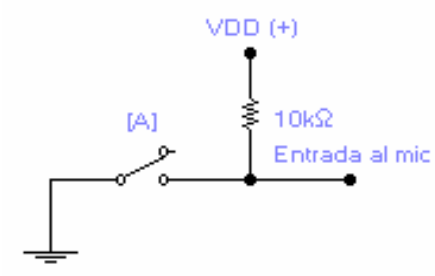


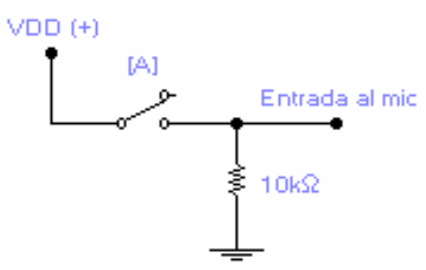


A) Sin pulsar: entrada en VDD (+) = 1
Pulsando: entrada a cero volt. = 0

B) Sin pulsar: entrada en cero volt. = 0
Pulsando: entrada en VDD (+) =0

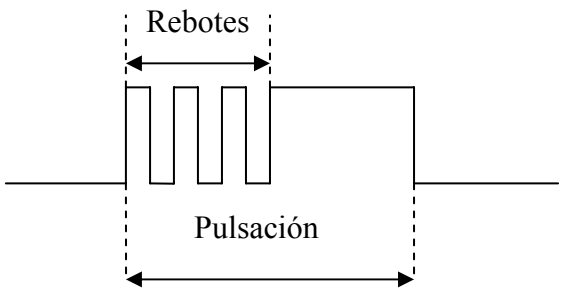
Contactos o interruptores:





El análisis de los circuitos, es similar al de los pulsadores

Circuitos de entradas antirrebotes:



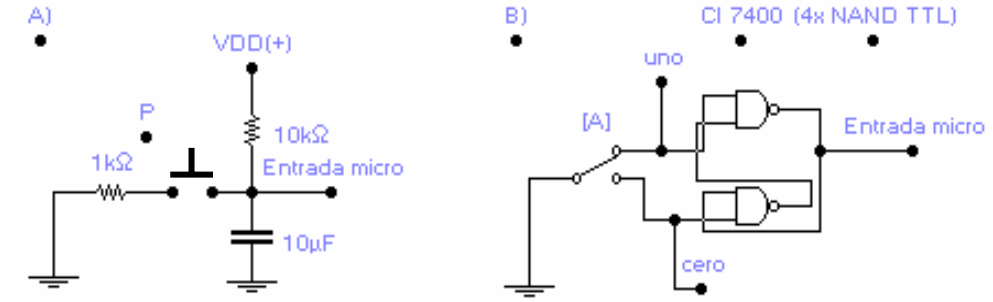
Los dispositivos electromecánicos, al cerrar, provocan rebotes que pueden durar algunos milisegundos. Si no se toma ninguna acción, pueden provocar inestabilidad, en la mayoría de los circuitos digitales.

En el caso de los microcontroladores, tenemos dos tipos de soluciones:

1)- Solución por programa:

Consiste en identificar el primer flanco de la señal de entrada, luego se pasa a un programa de "rutina de retardo" de varios milisegundos antes de pasar a detectar si se ha producido el flanco contrario.

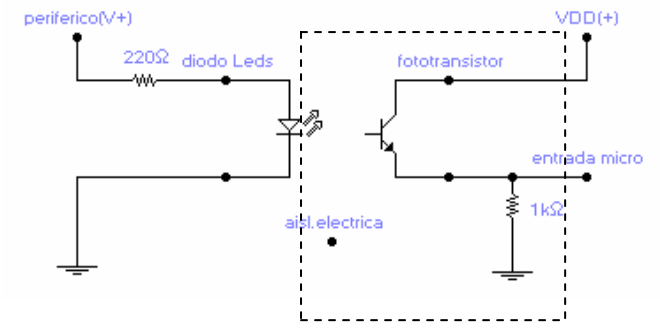
2)- **Solución por circuito**:



Circuito (A): Al cerrar "P" el capacitor se descarga a través de la resistencia de 1K y suministra un cero (0). Cuando se desactiva "P", el capacitor se carga a través de la resistencia de 10 K con una determinada cte de tiempo, que dependen de los valores del capacitor y resistencia de carga, suministrando un uno (1).

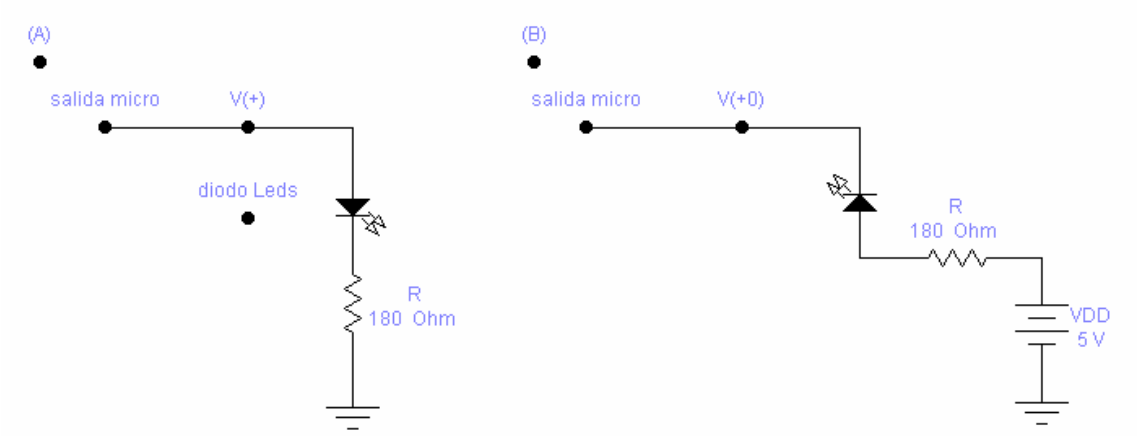
Circuito (B): El circuito representa un flip flop tipo RS con puertas NAND, de manera tal que este circuito cambia su salida, apenas detecta el 1° flanco.

Acoplamiento óptico de entradas digitales:



Los opto acopladores, son encapsulados de cuatro terminales, disponiendo en su interior, de un diodo Leds (emisor de luz) y un fototransistor (receptor de luz). Ambos dispositivos se encuentran aislados eléctricamente.
El periférico, cuando desea introducir un "uno lógico", al microcontrolador, aplica una tensión positiva al ánodo del diodo. La corriente circulante, provoca una emisión de luz, que es captada por el fototransistor. Este último, al tener aplicada una tensión eléctrica en su terminal colector, conduce corriente que circula por la resistencia de 1K, provocando una caída de tensión en sus extremos, que es captada por el microcontrolador, interpretándola, como un "uno lógico" en su entrada.

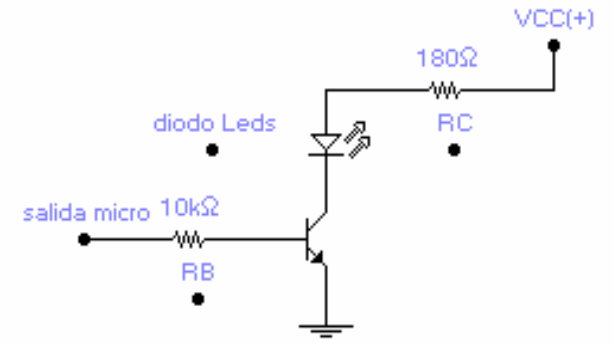
Circuitos conectados a las salidas del microcontrolador
Diodos Leds:



Circuito (A): Cuando la salida del microcontrolador es una tensión positiva (uno lógico), entrega una corriente al diodo Leds, limitada por la resistencia eléctrica. El diodo Leds, emite luz.

Circuito (B): Cuando la salida del microcontrolador provee una tensión baja (0+), por el diodo Leds circula una corriente, dado que su ánodo, tiene aplicada una tensión positiva respecto a la masa o terminal común.

Utilización de transistores y diodos Leds:



Este circuito, se utiliza para amplificar la corriente de salida del microcontrolador. El transistor trabaja al corte y saturación. La resistencia en colector limita la corriente que se entrega al diodo Leds. La resistencia eléctrica en la base limita la corriente en la base del transistor y la de salida del microcontrolador.
Cuando el microcontrolador entrega una tensión positiva (uno lógico), suministra corriente a la base del transistor. Este, pasa a la saturación, dando lugar a la corriente de colector que a su vez alimenta al diodo leds. Este último, emite luz.

Las formulas de cálculo son las siguientes:

RC = (VCC – Vcsat. – VD) / Icsat.

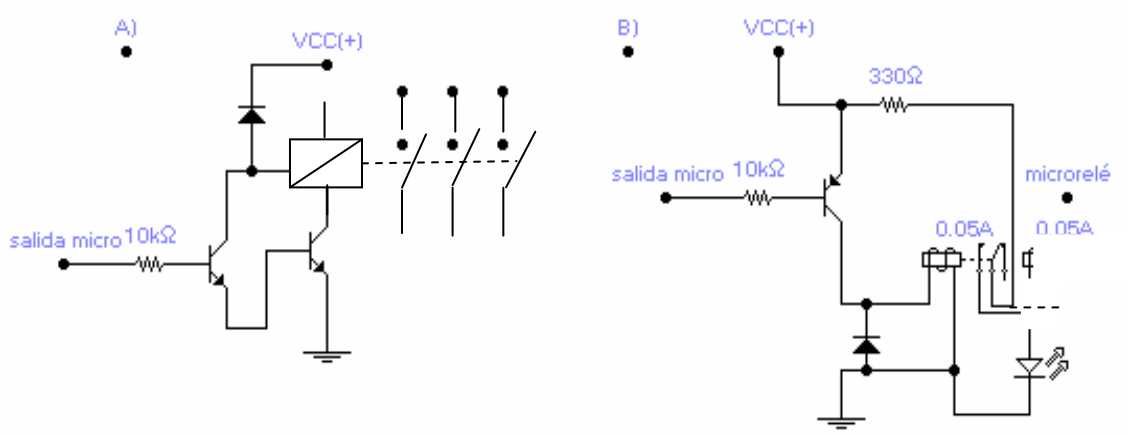
IBsat = Icsat/ βsat

RB = (V (+) – Vbsat)/ IBsat

βsat: ganancia de corriente de saturación del Transistor

V (+)= VDD =+5 volt. : tensión de salida del Microcontrolador

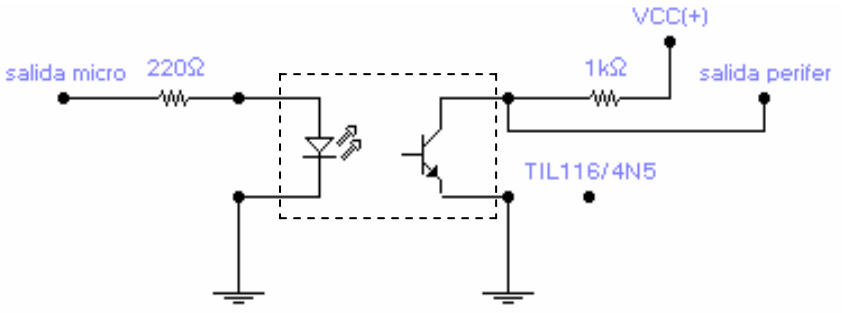
Activación por relés y microrelés:



A)- Actuando las salidas sobre relés, nos permite controlar cargas mucho mayor dado que las corrientes de carga pasarán por los contactos del relé. Por ejemplo poner en marcha un motor eléctrico a través de un contactor. Un “uno” en la salida del micro (V+) produce el accionamiento del relé. Un “cero” (0+), el relé esta desactivado.

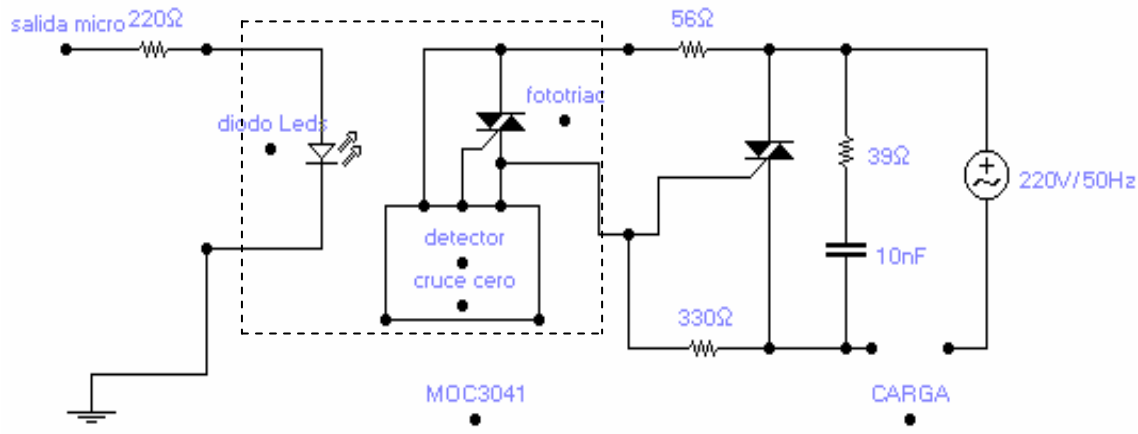
B) Este es el caso de activación por microrelés con doble contacto. En este caso, el micro relé se activa con un “cero” en la salida del microcontrolador y se desactiva con un “uno lógico”. En este circuito, se utiliza un diodo Leds para indicar la activación del micro relé; el otro se utiliza para la aplicación.

Salidas opto acopladas:



Cuando la salida del microcontrolador vale “1” (V+), el Leds del opto acoplador se enciende y activa al fototransistor a la saturación, entregando entonces un “0” (0+) lógico al periférico. Cuando la salida del microcontrolador vale “0” (0+), el Leds esta apagado y el fototransistor esta desactivado (corte); se entrega un “1” (VCC+) al periférico.

Control de cargas eléctricas alternas con triac:

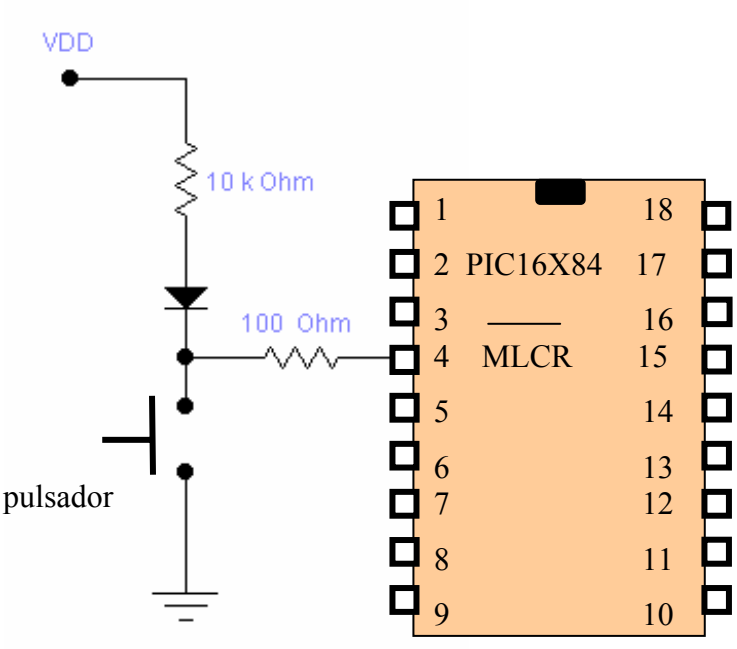


Los TRIAC son dispositivos electrónicos que dejan pasar una parte del semiciclo positivo o negativo, en función de un impulso de disparo aplicado a su compuerta. En el caso del circuito del ejemplo, resulta un control de carga eléctrica "por ciclos enteros".
Cuando la salida del microcontrolador vale "1", el diodo Leds se enciende ; el fototriac se activa recién cuando la tensión alterna de la carga pasa por cero, y de esta manera le inyecta un impulso de corriente a la puerta del triac de potencia que controla la carga. El resistor 39 ohm y capacitor conectado a el (10nf), protegen al triac frente a sobre tensiones y dv/dt.

Otras aplicaciones:
Existen una gran variedad de aplicaciones conectadas a las salidas del microcontrolador como ser activación de displays de 7 segmentos, pantallas de cristal líquido LCD, zumbadores, comunicaciones digitales bajo la norma RS-232 (previo desarrollo de un programa de comunicación y circuito especial adaptador como el MAX232), control de motores paso a paso, etc.

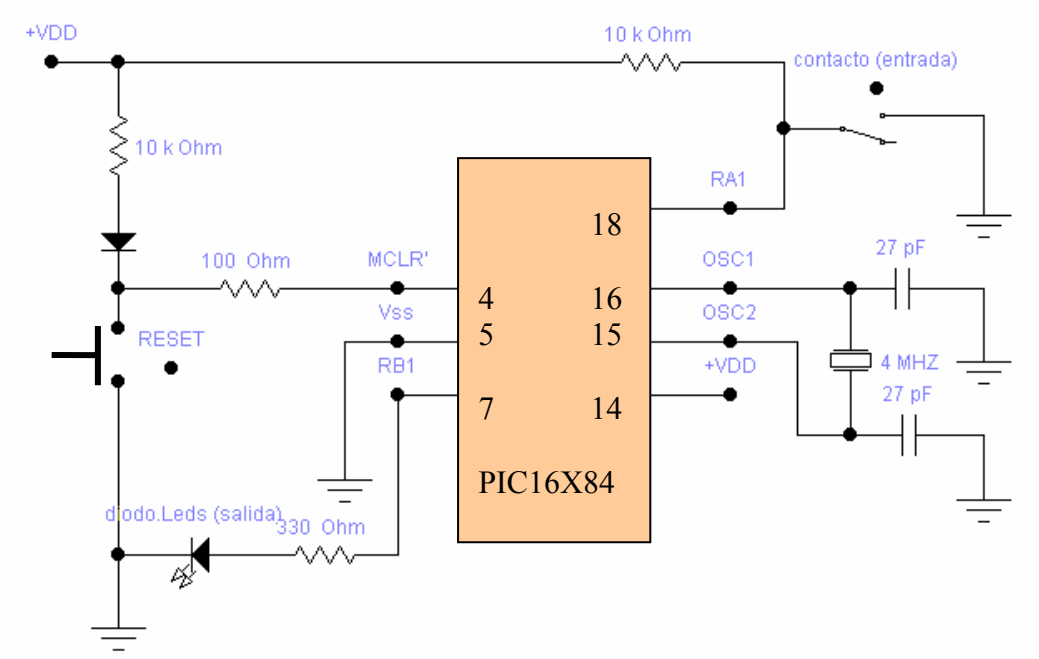
Circuito de reinicializacion o reset

En los microcontroladores, se requiere un Terminal para reiniciar el funcionamiento del sistema cuando sea necesario, ya sea por una falla que se presente o porque así fue diseñado. Este Terminal se denomina "Master Clear", abreviadamente MCLR.
La acción de provocar un "reset" en el microcontrolador, produce dos efectos importantes:
a)- El contador de programa (que me indica la próxima dirección de la instrucción a ejecutar) se carga con la dirección 0x 00 (00000000), apuntando a la primera dirección de la memoria de programa (vector reset) en donde deberá estar situada la primera instrucción del programa de aplicación.
b)- La mayoría de los registros de estado y control del procesador, toman un estado conocido y determinado.
En el PIC 16X84, el Terminal de "reset" esta ubicado en el pin nº4 denominado $\overline{\text{MCLR}}$.
Este microcontrolador, admite cinco diferentes tipos de reset:
1)- **Reset al encendido "POR**" (Power On Reset), hasta estabilizar VDD y el oscilador; Si esta habilitado, se logra conectando el terminal de reset (MCLR#) con el terminal de la entrada de la tensión de alimentación VDD.(se conecta a través de una resistencia eléctrica)
2)- **Reset por pulsación externa** (Master clear); se logra, llevando a masa el terminal de reset.
3)- **Reset por pulsación externa** (Master Clear), cuando el microcontrolador esta en el estado de bajo consumo (modo sleep). Se logra de la misma forma que el reset nº2
4)- **Actuación del circuito de vigilancia** "perro guardián" (watchdog) durante la operación normal (si esta habilitado).
5)- **Actuación del circuito de vigilancia** "perro guardián" durante el modo de reposo (modo slepp), si esta habilitado.
Un circuito sencillo que admite un reset al encendido" (si esta habilitado) y reset por pulsación externa (2 y 3), es el siguiente:

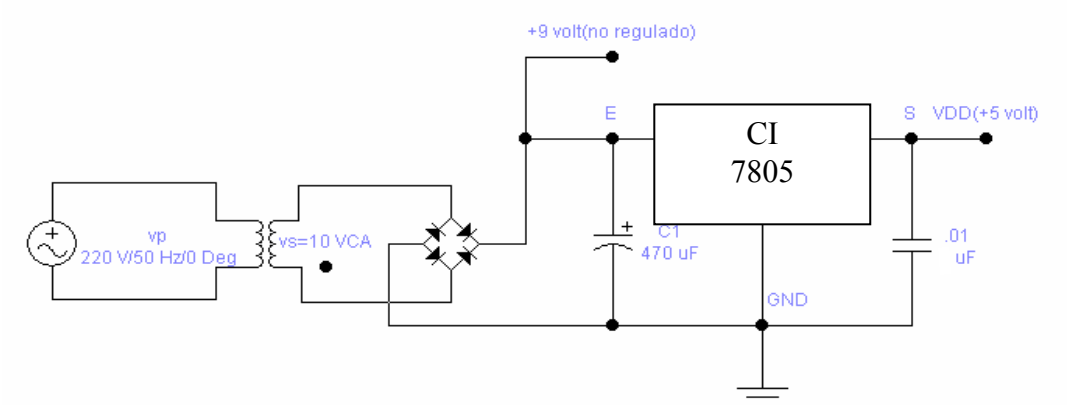


Circuito práctico

El siguiente circuito práctico muestra un conexionado común en casi todas las aplicaciones. Dispone de un circuito de reset, un oscilador a cristal con una frecuencia de oscilación de 4 MHZ (tipo XT), una entrada a contacto (RA1) y una salida con un diodo LEDS (RB1). La tensión de alimentación, es de 5 volt., que es un valor normal de aplicación para los microcontroladores PIC. Para una mayor estabilidad de funcionamiento resulta conveniente que la tensión de alimentación sea provista por una fuente regulada, como podría ser con el CI 7805



Esquema de la fuente de alimentación +VDD



CAPITULO 2: Métodos para el diseño y programación de los microcontroladores

PROGRAMACIÓN DE LOS MICROCONTROLADORES

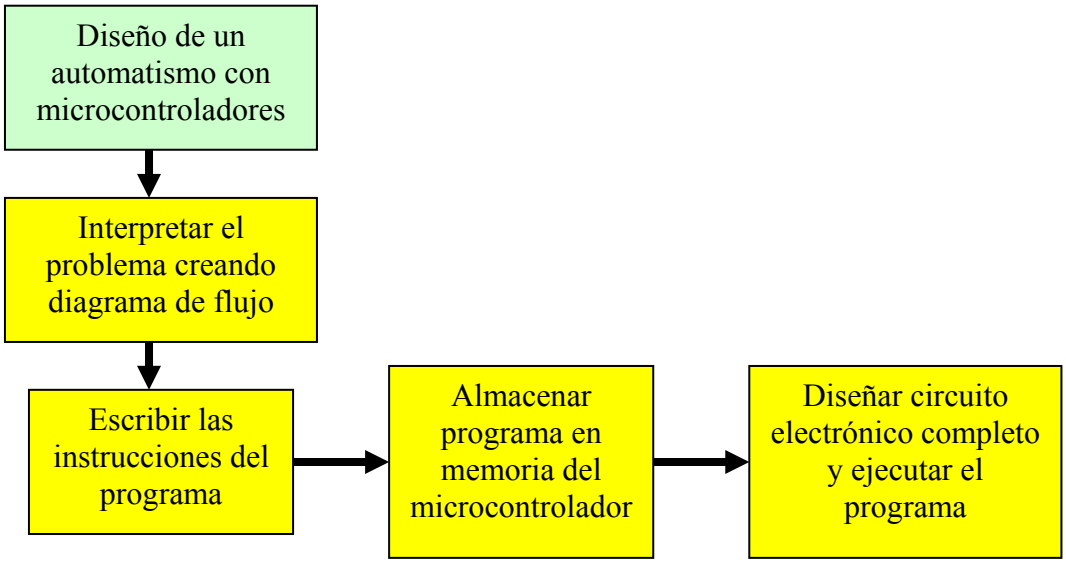
Introducción:

Cuando se decide realizar un automatismo o un subsistema de un sistema más complejo (por ejemplo formando parte de un circuito de un aparato electrónico), se debe establecer la combinación adecuada del "hardware" (circuitos) y del "software" (programa). Estos puntos, son los que involucran el diseño con microcontroladores.

Los microcontroladores, permiten configurar un sistema que cumpla con los requisitos del problema a resolver, gracias a una característica fundamental que comparten con las computadoras convencionales: que son "programables". **Por ello, diseñar sistemas de control con microcontroladores, exige el dominio de dos especialidades fundamentales: la primera es la especialidad o destreza para seleccionar y conectar componentes electrónicos (diseñar y realizar el circuito), y la segunda, es el conocimiento de las técnicas de programación.** Ambas especialidades, logran que el microcontrolador actúe según los requisitos que el problema a resolver propone.

Un aspecto importante que tenemos que tener siempre presente, cuando realicemos el programa, es que todos los sistemas programables, no procesan la información en forma continua (como los sistemas analógicos), sino que lo hacen en pequeños periodos de tiempo, por lo que deben organizar sus tareas en forma secuencial en el tiempo.

Los pasos básicos en la creación y ejecución de un programa, en un sistema programable en Gral., son los indicados en el diagrama en bloques de la siguiente figura:



Estas acciones a resolver, involucran a especialistas en el tema y son los denominados "PROGRAMADORES".

Lenguajes de programación:

Dado que programar en lenguaje de maquina (de unos y ceros) resulta muy complicado, es conveniente utilizar lenguajes nemotécnicos, más fáciles de entender. Existen varios lenguajes que utilizan las computadoras modernas. Algunos de ellos se utilizan para resolver problemas de carácter administrativo, como lo es el lenguaje COBOL. Otros lenguajes, ayudan a crear programas de utilidad para Ingeniería, como FORTRAN, PASCAL etc.
Cuando se trata de resolver problemas de "control industrial" con microcontroladores, cuya capacidad de memoria de programa resulta restringida, conviene utilizar lenguajes de bajo nivel o más cercano al dispositivo. Él más conveniente, por requerir menos instrucciones para ejecutar tareas especificas, es el" lenguaje ensamblador o Assembler". Este lenguaje esta compuesto por un conjunto de palabras sencillas, que permiten describir las acciones básicas, que ejecuta la UCP del microcontrolador.
Uno de los inconvenientes de este lenguaje, es que cada familia de microcontroladores, tiene su propio lenguaje ensamblador. No obstante esta dificultad, aprendiendo a programar en "ensamblador" para un determinado tipo de microcontrolador, le permite transferir esta especialidad, a otro diferente.
Otros lenguajes de alto nivel que se utilizan en la programación de microcontroladores son el lenguaje "C" y el lenguaje "Basic".
Cuando se utiliza uno de estos lenguajes, es necesario otro programa de computadora para que lo traduzca al sistema binario, de manera tal que se pueda introducir en la memoria de instrucciones del microcontrolador. Estos programas se denominan "ensambladores" o "compiladores" y sirven para el microcontrolador específico o para una determinada familia de microcontroladores. En el caso específico del microcontrolador PIC tenemos:

Lenguaje ensamblador > ensamblador MPASM.

Lenguaje C > compilador PCM.

Lenguaje Basic > compilador PBASIC.

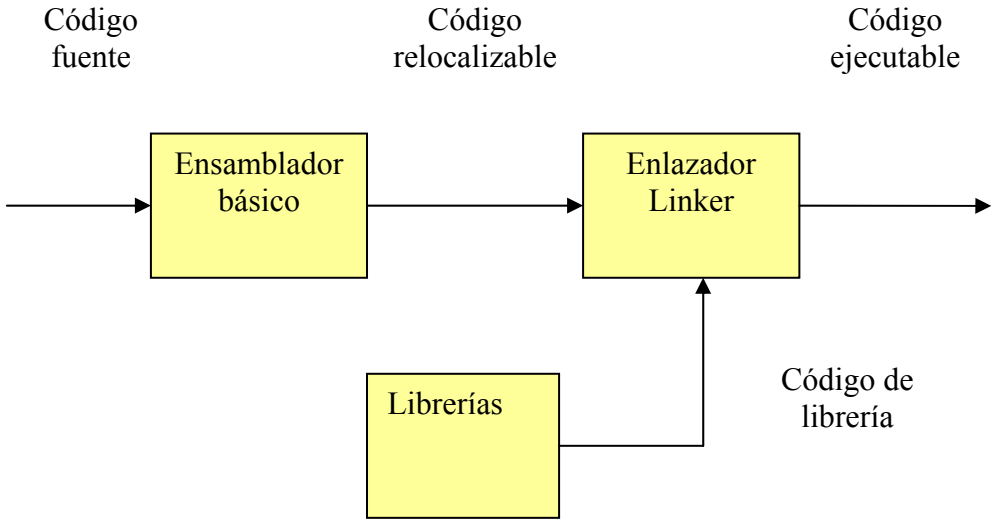
Descripción del programa ensamblador:
El "programa ensamblador", es aquel que recibe el "código fuente" de un "programa de usuario" y genera un archivo binario ejecutable. Éste código, se almacena en la memoria de instrucciones y es ejecutado por la UCP de microcontrolador.

Definimos como "**programa de usuario**" aquel que es escrito por el programador, en el lenguaje "**ensamblador**", utilizando un editor de texto de PC.
El programa ensamblador, esta conformado por varios módulos independientes, cada uno de los cuales, cumple una función especifica. Los módulos más importantes son:

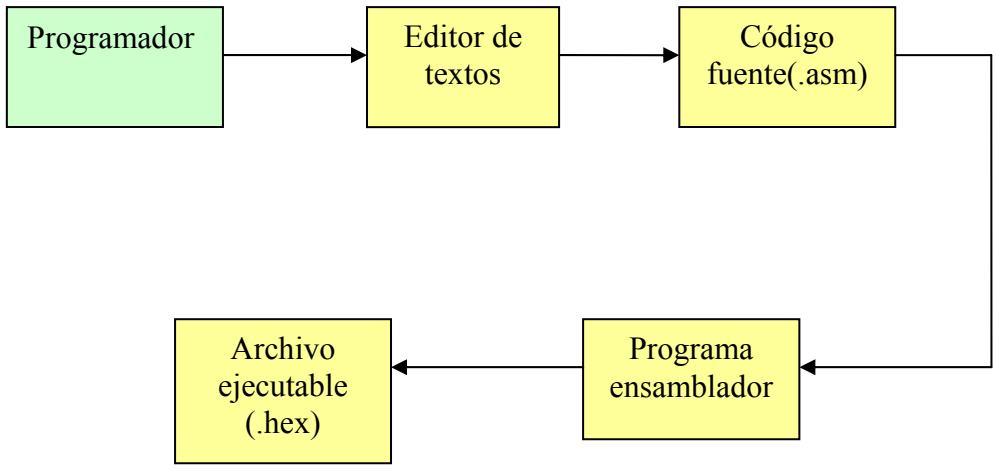
a)-Ensamblador básico: Genera, a partir del código fuente, un archivo "relocalizable"

b)- Enlazador: Crea, a partir del archivo relocalizable y otros archivos del módulo" control de librerías"(Lib.), un archivo binario ejecutable. Éste código, es el que ejecuta directamente el microcontrolador.

c)- Control de librerías (Lib.): Éste módulo permite crear archivos binarios que pueden ser unidos (enlazados) con otros bloques de código binario, lo que facilita la reutilización de partes de programas generados en otros proyectos. El uso de librerías simplifica el desarrollo de programas de gran tamaño y complejidad.

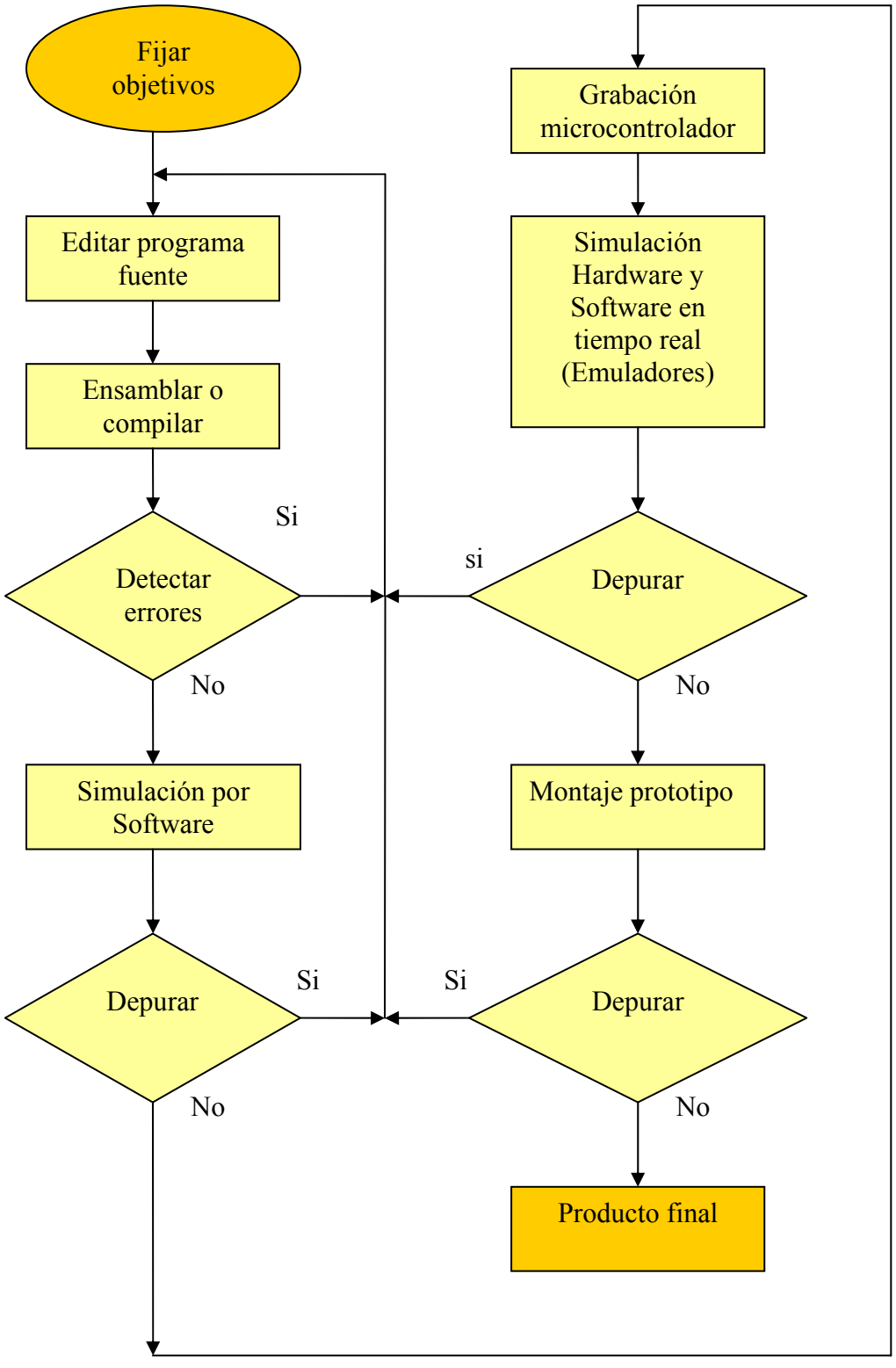


Un esquema más simple y más cercano a la realidad cuando se programa un microcontrolador, es el que se muestra en la figura:



Estos son los pasos concretos que debemos seguir para obtener el programa ejecutable que luego debe grabarse en la memoria del microcontrolador.

Diagrama de flujo de las fases de diseño con microcontrolador



Archivos generados por el programa ensamblador:

Además del código binario ejecutable, el programa ensamblador genera un conjunto de archivos adicionales utilizables para controlar la evolución del proyecto. La extensión que

acompaña a cada nombre de de archivo de un punto y tres letras, indica cual es el tipo de información que contiene el archivo. Un ejemplo tomado de un ensamblador típico, es el siguiente:

TIPO DE ARCHIVO	EXTENSIÓN	EJEMPLO	COMENTARIO
Código fuente programa programador	ASM	prueba.asm	programa escrito por
Código binario ejecutable microcont.	HEX	prueba.hex	Archivo ejecutable
Listado del programa proceso ensamb.	LST	prueba.lst	salida formateada
Depuración depuración	COD	prueba.cod	archivo para
Errores generados	ERR	prueba.err	Listado de errores

Estructura del programa escrito en lenguaje ensamblador con editor de textos de PC

El programa escrito usando el lenguaje ensamblador (o lenguaje de las instrucciones nemotécnicas), debe organizarse según un diseño basado en columnas y líneas, de manera tal que el programa ensamblador, lo pueda interpretar.
La figura, muestra la estructura de las columnas que debemos respetar:

ETIQUETAS	INSTRUCCIONES	OPERANDOS	COMENTARIOS
Inicio	org	0	; Comienzo programa
	Movwf	0x0d	; muevo W a 0c (Hex)

Etiqueta : Es un nombre con el cual se identifica una posición de memoria del microcontrolador, y sirve para marcar puntos específicos dentro del programa.
Toda etiqueta debe escribirse en la primera columna de la línea y su longitud, no debe superar, usualmente los 31 caracteres. Los caracteres pueden ser los alfanuméricos, el carácter de subrayado (_) y el símbolo de interrogación (?).
Instrucción: Hace referencia a una de las operaciones básicas que puede realizar el microcontrolador; por ejemplo “movwf” significa cargar al registro “f” el contenido del registro “w”, también llamado registro de trabajo.
Operando: Es un elemento utilizado por una instrucción. En el caso del ejemplo 0x0d él operando es la dirección en hexadecimal (0x) de un registro de propósito general (0d). Algunas instrucciones no tienen operando. Otras, tienen dos operandos, para cumplir su objetivo: el primer operando, cuando esta definido, se denomina “operando fuente”. El segundo operando, complemento del anterior, recibe el nombre de “operando destino”. La información fluye desde él operando fuente hacia él operando destino.
Comentario: Un comentario es un texto que le sirve al programador para documentar el programa. Para que el ensamblador lo ignore, debe ir precedido con el carácter punto y coma (;).

Estructuras de las líneas que se incluyen dentro del programa

NOMBRE DEL ARCHIVO
PROBLEMA PLANTEADO
ENCABEZADO
CONSTANTES
DEFINICIÓN DE ORIGEN
INSTRUCCIONES
FINAL DE PROGRAMA

Analizaremos a continuación el significado de cada uno de ellos.

Nombre del archivo: Resulta conveniente colocar como titulo, precedido de ";" el nombre del archivo con su extensión (.ASM), si se lo imprime, con la finalidad de poder localizarlo posteriormente. No es una línea obligatoria.
Problema planteado: Resulta también conveniente, definir en forma concisa las características del problema planteado a solucionar con el sistema programable, para la mejor comprensión del desarrollo del programa. Esta línea no es obligatoria y como el caso anterior, también va precedida del punto y coma, para que el ensamblador lo tome como un comentario.
Encabezado: Es el primer componente del programa en si, y en el se definen algunas directivas de tipo Gral. de tal forma que modifican el funcionamiento del ensamblador. La directiva a colocar es "list = tipo de microcontrolador". Por ejemplo si vamos a programar el microcontrolador PIC 16F84, colocaremos en esta primer línea "list=16F84". Esta directiva es obligatoria colocarla.
Constantes: En esta línea/s definen constantes que son reconocidas en cualquier punto del programa. Usar constantes simplifica la lectura del programa, ya que en vez de referirse por ejemplo a valores hexadecimales, se utiliza el nombre de una constante.
Ejemplo: puertoA equ 05, estamos reemplazando el valor hexadecimal del registro "05" por la constante "puertoA".
Estas líneas no son obligatorias para confeccionar el programa.
Definición de origen: Cuando realizamos el programa, debemos indicarle explícitamente, en que dirección de la memoria, se debe almacenar el código binario del inicio del programa que va a ser ensamblado.
Ejemplo : org 0 esta línea le esta indicando al programa ensamblador que deberá almacenar el programa desde la dirección "0" de la memoria de programa del microcontrolador.
Instrucciones: En esta sección, se colocaran las instrucciones según los siguientes campos ya definidos:

ETIQUETAS	CÓDIGO DE INSTRUCCIÓN	OPERANDOS	; COMENTARIOS

A excepción del campo del campo correspondiente al" código de instrucción", los restantes campos pueden o no, aparecer dentro de la línea.

Final de programa: Esta línea, contiene una instrucción simple que indica el final del programa. El ensamblador MPASM para los PIC, utiliza la instrucción "end".

PROGRAMACIÓN DEL MICROCONTROLADOR PIC

Antes De comenzar a elaborar un programa, debemos primero conocer la estructura lógica del microcontrolador en particular. Dada su versatilidad, arquitectura simple y revolucionaria, reducido set de instrucciones, etc., tomaremos como desarrollo el PIC 16X84 de Microchip. Los conceptos que desarrollaremos se pueden aplicar en su gran mayoría a los otros modelos de PIC.

Elementos del PIC16X84 para su programación:

-**El registro de trabajo W.**
-El registro de estado. (ESTADO)
-La memoria de programa. (EEPROM para 16C84 y FLASH para 16F84)
-la memoria de datos. (EEPROM)
-Los registros de propósito especial (SFR) (RAM)
-Los registro de propósito general (GPR) o memoria de datos RAM
-Los registros de pila (stack)
-El puerto A.
-El puerto B
-Set de instrucciones.
Estos son todos los elementos o componentes que necesitamos para desarrollar los programas de aplicación con microcontrolador PIC 16X84.

Analizaremos ahora, en forma general, cada uno de estos elementos. El detalle de los mismos lo tenemos en el apéndice “GUÍA RÁPIDA DEL PIC 16X84” y su aplicación, en los programas que desarrollaremos mas adelante, paso a paso.

El registro de trabajo W:
El registro W, es de 8 bits y sirve para almacenar un dato, generalmente en forma temporal, cuyo valor se utilizará posteriormente en una operación matemática o lógica, o en la transferencia entre registros y la memoria. Razón por la cual, toda la información pasa por este registro, es de suma importancia para la mayoría de las instrucciones del microcontrolador.

El registro de Estado:
Este registro consta de 8 bits y el valor que toma cada bit (1 ó 0), nos determina el estado de los componentes internos del microcontrolador. Durante la ejecución del programa, mediante instrucciones, los bits de este registro son consultados y su valor puede modificar el desenvolvimiento del programa en ejecución, por medio de rutinas preparadas a ese fin Estos bits se pueden modificar mediante instrucciones.

IRP	RPI	RPO	TO#	PD#	Z	DC	C

C: Bit de acarreo en el bit más significativo de un resultado (carry)
DC: Bit de acarreo en el tercer bit de un resultado. (Operaciones en BCD)
Z : Bit de cero. (Toma valor 1, si el resultado de la operación es cero)
PD#: Bit que indica el estado de bajo consumo y actuación perro guardián (power down).
PO#: Indica el final de tiempo del temporizador perro guardián (Timer out)
RPO – RPI: Selección de bancos de memoria direccionamiento directo
IRP: Selección de bancos de memoria direccionamiento indirecto

La memoria de programa:

0000 H	VECTOR RESET
0004 H	VECTOR DE INTERRUPCIONES
0005 H ---------- ---------- 03FF H	MEMORIA DE PROGRAMA (1 K)

En el PIC16F84, los programas de usuario, se almacenan en una zona de memoria de tamaño de 1 K palabras (14 bits).Esta cantidad es suficiente para solucionar gran parte de los problemas de control automático.
En esta memoria tenemos dos direcciones de interés, cuando debemos realizar el programa: la 0000 H (vector reset) y la 0004 H (vector interrupciones). La primera es la dirección de inicio de todo programa; la segunda corresponde al inicio de la rutina de servicio de interrupciones. Detallaremos cada una de estas direcciones:

Vector reset: Cuando se aplica un nivel bajo a la línea de reset (patilla n°4 Vpp /MCCLR#), el contador de programa toma el valor 0000 H y el programa comienza a ejecutar la primera instrucción del programa. Esta situación también se presenta cuando se conecta la tensión de alimentación del microcontrolador.

Vector de interrupciones: Los programas con un cierto grado de complejidad, admiten la aplicación de señales externas, denominadas interrupciones, que alertan la sobre la aparición de condiciones que deben ser atendidas sin ninguna demora por parte del sistema. Las

interrupciones también pueden producirse como consecuencia de variaciones en el estado de temporizaciones internas o debido al cambio en el contenido de ciertos registros.
La dirección de memoria 0004 H es el punto de inicio de la rutina de servicio de la interrupción. La porción de programa que debe ejecutarse cuando ocurra la interrupción, debe guardarse a partir de la dirección 0004 H. Como veremos mas adelante, en esa posición de memoria, se coloca una instrucción denominada “de salto incondicional”.
Si el programa no contiene una rutina de manejo de interrupciones, se puede utilizar toda la memoria de forma lineal.

La memoria de datos en RAM
La memoria de datos (RAM) del PIC16F84, esta compuesta por los registros de propósitos específicos (SFR) y los registros de propósitos general (GPR). La figura muestra un esquema simplificado de esta memoria:

Registros específicos (SFR) y de propósito general (GPR) PIC 16 F84

DIREC.	BANCO 0	BANCO 1
00 H	INDF	INDF
01 H	TMRO	TMR0
02 H	PCL	PCL
03 H	ESTADO	ESTADO
04 H	FSR	FSR
05 H	PUERTO A	TRIS A
06 H	PUERTO B	TRIS B
07 H	////////////////////	////////////////////
08 H	EEDATA	EECON1
09 H	EEADR	EECON2
0A H	PCLATH	PCLATH
0B H	INTCON	INTCON
0C H ..... 4F H	68 REGISTROS PROPÓSITO GENERAL	MAPEADOS EN BANCO CERO

El PIC16F84 se caracteriza por disponer de dos bancos de memoria de datos RAM: El banco 1 y el banco 0. Los registros de propósito especial, se encuentran en esta memoria, alguno de ellos repetidos en los dos bancos, como se muestra en la figura anterior. Su aplicación la veremos en los ejemplos de programas que desarrollaremos mas adelante. Daremos solamente su significado:

INDF: Direccionamiento indirecto
TMRO: Temporizador / Contador
OPTION: programación temporizador
PCL: Parte baja del contador de programa
ESTADO: Registro de estado
FSR: Selector de registros
PUERTO A: entrada o salida de datos
TRIS A: Configuración puerto A
PUERTO B: Entrada o salida datos
TRIS B: configuración puerto B
EEDATA, EECON1, EEADR, EECON2: Acceso a memoria de datos EEPROM
PCLATH: Parte alta del contador de programa
INTCON: Control de interrupciones.

De la misma forma, los bits de estos registros especiales, tienen distintas funciones, que las explicaremos, mas adelante.

Los registros de propósito general, son en total 68 para el PIC16F84 y están mapeados sobre el banco cero. Se los utiliza para guardar temporalmente datos que ingresan de los puertos o resultados de operaciones de la UAL.

La memoria de datos EEPROM

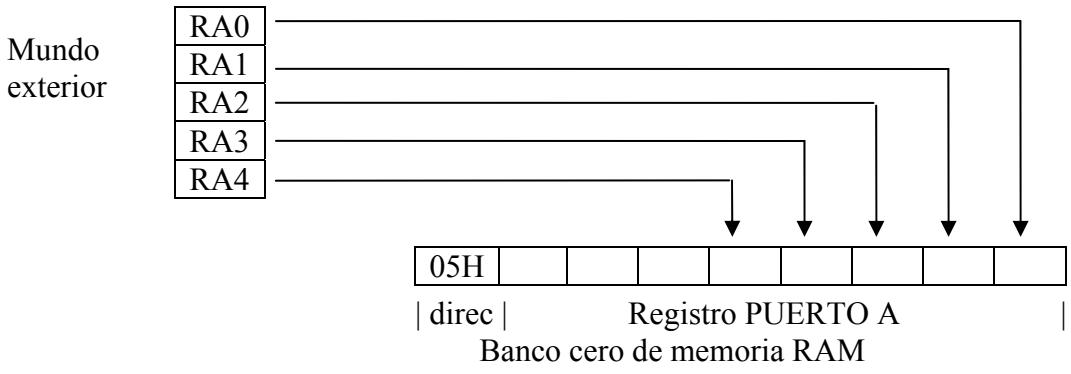
Por ejemplo el PIC16F84 dispone de una memoria EEPROM con una capacidad de almacenamiento de 64 bytes, comprendidas entre las direcciones 00 H hasta la 3F H. Para acceder a estos registros, hay que utilizar los registros especiales de control **EEDATA, EEADR, EECON1 EECON2**. El proceso de escritura de esta memoria es lento y dura unos 10 ms. Mas adelante veremos la aplicación de estos registros.

Los registros de pila (stack)

Cuando se desarrolla un programa de aplicación, en muchos casos resulta conveniente dividir un programa en pequeñas porciones de "subprogramas", los cuales cumplen un propósito especial. El programa principal, o sea aquel que será ejecutado cuando el microcontrolador reciba una señal de reset, o cuando sea energizado, efectuara llamadas a estos subprogramas en diferentes puntos, de acuerdo con las necesidades establecidas.
Para llamar a un subprograma, se lo hace a través de la instrucción "**CALL"** seguida de la etiqueta del subprograma. En el momento de ejecutar esta instrucción, el microcontrolador guarda la dirección de retorno al programa principal, desde el punto donde fue llamado el subprograma, en una memoria denominada "pila". En principio la "pila" no es mas que un deposito de datos (direcciones) en donde él último dato en entrar, es el primero en salir.
Cuando la subrutina concluye, lo hace siempre con una instrucción de retorno "**RETURN**".Esta instrucción saca la ultima dirección almacenada en la "pila" y la coloca en el contador de programa retornando al programa principal, desde la dirección donde fue llamado el subprograma. Un subprograma puede contener a su vez, otros subprogramas. Como la "pila "del PIC16F84 puede contener hasta 8 direcciones, se pueden producir "anidamientos" entre subprogramas, en una cantidad no mayor a ocho.
Los procesos relacionados con el manejo de la "pila", son transparentes al programador, es decir, ocurren de manera automática en el interior del microcontrolador.

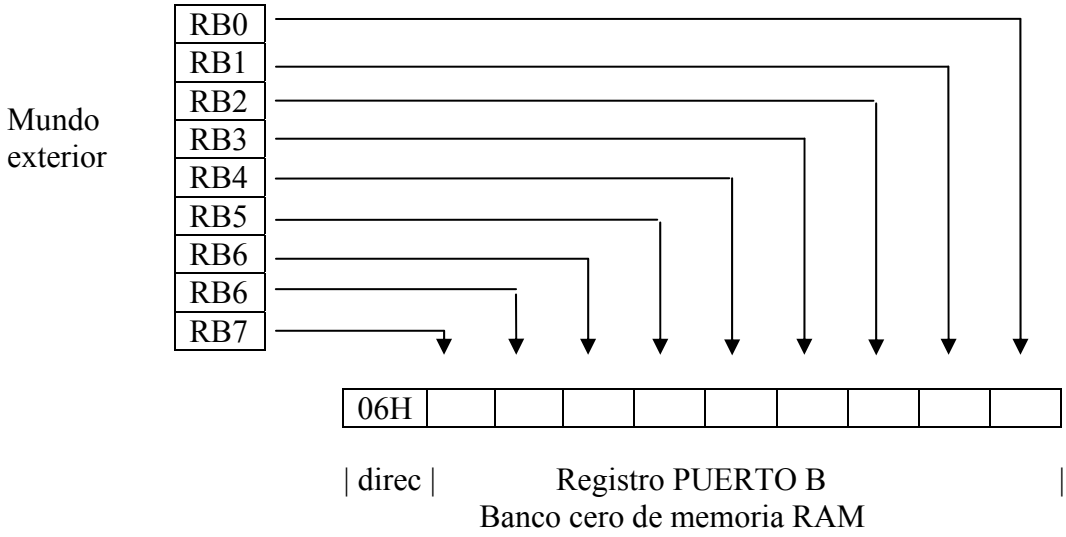
Los puertos entrada / salida del microcontrolador
Los puertos, son los elementos por los cuales se introduce o extrae información del microcontrolador. En el caso del PIC 16F84, se disponen de dos puertos denominados A y B. Pasamos a detallar, brevemente, cada uno de estos puertos.

Puerto A: Este puerto consta de 5 líneas que pueden utilizarse como entradas o como salidas, dependiendo del tipo de aplicación. Se denominan RA0, RA1, RA2, RA3, RA4.
La línea RA4 puede utilizarse también, como entrada de pulsos de reloj aplicados al temporizador / contador interno TMR0.



El programador, puede acceder al contenido de este puerto, en una operación de “lectura de las entradas”, o escribir, en una operación de “salida de dato al exterior”, mediante el registro “PUERTO A”, ubicado en la dirección 05 H del banco cero de la memoria RAM.

Puerto B:
El puerto B esta conformado por ocho líneas que pueden configurarse como entrada o como salida, en forma individual, denominadas RA0, RA1, RA2, RA3, RA4, RA5, RA6, RA7. Estas líneas, también pueden cumplir otras misiones como generar interrupciones o grabar el programa de aplicación, en la memoria del microcontrolador



De la misma forma que el puerto A, el puerto B esta disponible al programador por medio del registro PUERTO B en la dirección 06 H del banco cero de la memoria RAM.

Configuración de los puertos A y B :
Éstos se configuran como entradas o salidas mediante los registros TRISA y TRISB en las direcciones 05 h y 06 h respectivamente, del banco uno de la memoria RAM. Colocando un uno (1) o un cero en los bits de estos registros, las líneas se configuran como “entradas” o “salidas” respectivamente. E: entrada, S : salida

PUERTO A

			S	E	E	S	E

			0	1	1	0	1

TRISA

PUERTO B

E	E	E	S	E	E	S	S

1	1	1	0	1	1	0	0

TRISB

Set de instrucciones de los microcontroladores PIC

Los modernos microcontroladores PIC, responden a la arquitectura “RISC”, significando esto que son computadores con juego de instrucciones reducido. Disponen de un conjunto de

instrucciones maquina pequeño y simple, de forma que la mayor parte de las instrucciones se ejecutan en un ciclo de instrucción.
Por ejemplo la familia de microcontroladores PIC 16X84, dispone de un conjunto de 35 instrucciones clasificadas de la siguiente forma:
a)- Instrucciones que manejan registros. Cantidad: 16
b)- Instrucciones que manejan bits. Cantidad: 2
c)- Instrucciones de "salto". Cantidad: 4
d)- Instrucciones que manejan operandos inmediatos. Cantidad: 6
e)- Instrucciones de control y especiales. Cantidad: 7

Las características en particular, del accionar de estas instrucciones, las veremos mas adelante cuando desarrollemos los programas de aplicación, donde explicaremos paso a paso, el desarrollo y efectos de estas instrucciones en el entorno del microcontrolador..
El repertorio de estas instrucciones con su sintaxis, operación, ciclos, formato y actuación de señalizadores, lo podemos ver en el apéndice Guía rápida del PIC 16X84.

LA PALABRA DE CONFIGURACIÓN

Se trata de una posición reservada de la memoria de programa situada en la dirección 2007 H y accesible solamente durante el proceso de grabación. Al escribirse el programa de la aplicación, es necesario grabar el contenido de esta posición de acuerdo con las características del sistema. Veamos la distribución de los bits de la palabra de configuración:

CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	PWRTE#	WDTE	FOSC1	FOSC0

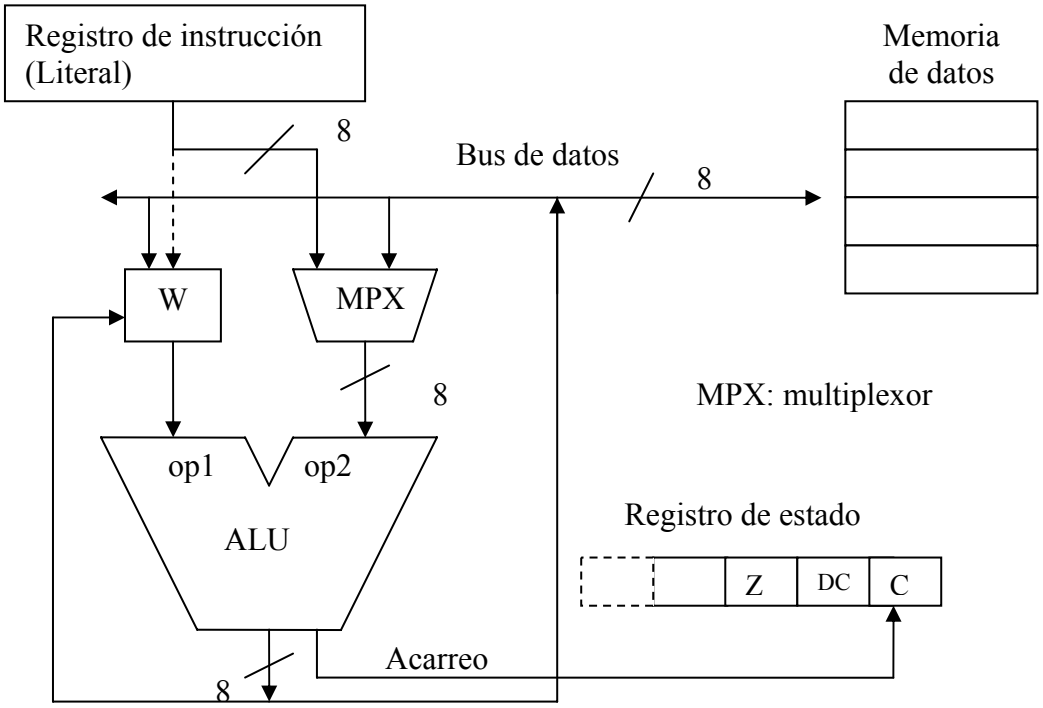
CP : Bits de protección de la memoria de código
Si colocamos un "1" la memoria no esta protegida.
Si colocamos un "0", el programa no se puede leer, evitando copias. Tampoco se puede sobrescribir. Además evita que pueda ser accedida la EEPROM de datos y, finalmente si se modifica el bit CP de "0" a "1", se borra completamente la EEPROM.
PWRTE: Activación del temporizador "Power-up". Este temporizador retrasa 72 ms la puesta en marcha o reset que se produce al conectar la alimentación al PIC, para garantizar la estabilidad de la tensión de alimentación aplicada.
Si colocamos un "0", la temporización se activa. Si colocamos un "1", se desactiva.
WDTE: Activación del "perro guardián" Se denomina de esta manera a una seguridad que dispone el microcontrolador en el caso de que la ejecución del programa quede "colgado". Este dispositivo es en definitiva un temporizador programado que si esta en activación, produce un reset, cuando finaliza su tiempo, volviendo el programa a su estado inicial.
Cuando colocamos un "1" el "perro guardián esta activado". Si colocamos un "0", esta desactivado. Cuando esta activado, para evitar que produzca reset cuando el programa funciona correctamente, es necesario anular el final de la temporización, con instrucciones al efecto, ubicadas en puntos estratégicos del programa.
FOSC1 – FOSC2 : Selección del oscilador utilizado. Estos bits deben ser cargados con "1" 0 "0" según el tipo de oscilador que se va a utilizar para generar los pulsos reloj, necesarios para el funcionamiento del PIC.
1-1 : Oscilador RC
1-0 : Oscilador HS.
0-1 : Oscilador XT
0-0 : Oscilador LP

PALABRA DE IDENTIFICACIÓN (ID)

Son cuatro posiciones de memoria de programa ubicadas en las direcciones 2000 H- 2003 H que no son accesibles, en el funcionamiento normal del microcontrolador, y solo pueden ser leídas y escritas durante el proceso de grabación.
Solo se utilizan los 4 bits de menor peso de cada palabra de identificación (ID), en donde se almacena un valor que puede consistir en un número de serie, código de identificación, numeraciones secuenciales o aleatorias etc.

MODO DE TRABAJO DEL MICROCONTROLADOR

Antes de proseguir con aspectos prácticos y funcionales de este dispositivo, realizaremos como primer experiencia, un programa para sumar o restar dos operandos en forma binaria en la "ALU" del microcontrolador PIC 16X84, guardando el resultado en la memoria de datos RAM ,específicamente, en los registros de propósito gral (GPR). Veamos los registros involucrados para esta operación:



W, es el registro de trabajo o registro acumulador. Como hemos dicho anteriormente es el más utilizado y ocupa un lugar físico especial. Los demás registros especiales del microcontrolador, se los localiza en la memoria de datos RAM, en los bancos 0 y 1 entre las direcciones 0x 00 y 0x 0B.
0x: indica que la dirección esta en hexadecimal, o sea 0x 0B > 00001011(binario),
El programa que vamos a realizar, requerirá de tres instrucciones que son mover, sumar y restar. A continuación analizaremos cada una de ellas en su lenguaje nemotécnico.

Movimiento de datos: En asembler, es "mov" y puede mover el contenido del registro W, el contenido de una posición de la memoria de datos al registro W, o a si mismo un valor literal o valor inmediato que se incluye en el código de maquina de la propia instrucción.
Veamos a continuación las instrucciones de código de maquina escritas en asembler.

movf f,d : mueve el contenido del operando "fuente" o sea "f" y que es una posición de la memoria de datos y lo deposita en W si d=0 o´ en el propio fuente si d = 0.

Cuando se realiza esta operación, se activa el señalizador de cero FZ (ó Z) ubicado en el registro especifico (SFR) ESTADO, en el bit n°2. Si Z = 1 el resultado es cero. Si Z =0 el resultado es distinto de cero. En este caso moverse a si mismo cuando d = 1, se utiliza para saber si el contenido del registro fuente, vale cero o no.

movwf : mueve el contenido de W al registro o posición de memoria de datos "f" o sea W => f. Cuando se realiza esta operación, W queda con su valor anterior, es decir que no se borra su contenido, salvo que por otra instrucción, le carguemos otro valor. Lo mismo ocurre con la primera instrucción.

movlw k : mueve el literal k al registro W. Esta instrucción debe tener cargado el literal en la misma instrucción.

addwf f,d : suma el contenido del registro W con el de "f" y deposita el resultado en W si d = 0, mientras que lo deposita en "f" si d = 1.

addlw k : suma al contenido del registro W el literal que acompaña a la instrucción y deposita el resultado en W (W + k => W).

subwf f,d : Idem a la suma pero la operación es de resta.

sublw k : Idem a la suma pero la operación es de resta.

El la operación de suma, también interviene el bit nº0 (C), que indica si hubo acarreo. Si C=1, hubo acarreo; caso contrario, no lo hubo.
En la operación de resta, el bit "C" también actúa como señalizador de "llevada". En este caso la correspondencia es inversa (si vale 1 no hay llevada y si vale 0 si).

A continuación realizaremos este programa con variantes, utilizando las instrucciones para "mover" y para sumar.

```
; PRIMERO1.ASM: Este programa suma el contenido de las posiciones 0x0c
; y 0x0d de memoria y almacena el resultado en la posición 0e.


        LIST p= 16F84           ; Indica el modelo de PIC que se usa
                                ;Es una directiva del ensamblador.

; -----------------------------------------------------------------
                                ;Zona para etiquetas.
        OPERANDO1 EQU 0x0c      ;Define la posici¢n del operando1
        OPERANDO2 EQU 0x0d      ;Define la posición del operando2
        RESULTADO EQU 0x0e      ;Define la posición del resultado

;-----------------------------------------------------------------

        ORG 0                  ;Comando que indica al Ensamblador
                               ;la dirección de la memoria de programa
                               ;donde situar  la siguiente instrucción

;-----------------------------------------------------------------

        movlw   05             ; 5 -> W (Primera instrucción)
        movwf   OPERANDO1      ; W -> OPERANDO1
        movlw   02             ; 2 -> W
        movwf   OPERANDO2      ; W -> OPERANDO2
        movfw   OPERANDO1      ; OPERANDO1 -> W
        addwf   OPERANDO2,0    ; W + OPERANDO2 -> W
        movwf   RESULTADO      ; W -> RESULTADO

        END                    ; Directiva de fin de programa
```

```
;PRIMERO2.ASM Optimización del programa Primero.asm que calcula la
;suma de 2 posiciones de memoria y deja el resultado en una tercera.
```

```
;Estas posiciones son 0x0c para operando1, 0x0d para operando2 y 0x0e
; para el resultado.
;La optimización consiste en ahorrar una instrucción al aprovechar el
;hecho de que la suma es una operación conmutativa.De esta manera tras
;cargar en W el operando 2, se puede realizar directamente la suma.


        LIST p= 16F84           ;Indica el modelo de PIC que se usa
                                ;Es una directiva del ensamblador.

; -----------------------------------------------------------------------
                                ;Zona para etiquetas.
        OPERANDO1 EQU 0x0c      ;Define la posición del operando1
        OPERANDO2 EQU 0x0d      ;Define la posición del operando2
        RESULTADO EQU 0x0e      ;Define la posición del resultado

;-----------------------------------------------------------------------

        ORG 0                   ;Comando que indica al Ensamblador
                                ;la dirección de la memoria de
programa
                                ;donde situar  la siguiente
instrucción

;-----------------------------------------------------------------------

        movlw   05              ;5 --> W (primera instrucción)
        movwf   OPERANDO1       ;W --> Operando1
        movlw   02              ;2 --> W
        movwf   OPERANDO2       ;W --> Operando2
        addwf   OPERANDO1,0     ;W + operando1 --> W
        movwf   RESULTADO       ;W --> resultado

        END                     ;directiva de fin del programa


;PRIMERO3.ASM.     Este programa suma el contenido de las posiciones
;0c y 0d de memoria y almacena el resultado, en la misma posición 0d.


        LIST    p= 16F84     ; Para PIC 16F84

;-----------------------------------------------------------------------

        OPERANDO1  EQU  0x0C ; Define la posición del operando 1
        OPERANDO2  EQU  0x0D ; Define la posición del operando 2
                             ; y del resultado

;-----------------------------------------------------------------------

        ORG     0            ; Dirección de inicio del programa

;-----------------------------------------------------------------------

        movlw   02           ; 2 -> W
        movwf   OPERANDO2    ; W -> OPERANDO2
        movlw   05           ; 5 -> W
```

```
        movwf   OPERANDO1     ; W -> OPERANDO1 ( Operando1 esta  en W y
.                             ; en 0x0C)
        addwf   OPERANDO2,1   ; OPERANDO2 + W -> Operando2

        END                   ; Directiva de fin de programa
```

Cuando el programa contiene pocas instrucciones, es posible editar el archivo de texto, sin definir etiquetas ni comentarios, simplemente se coloca la dirección, en hexadecimal, del registro que acompaña a la instrucción, como lo muestra el siguiente ejemplo:

```
;                PRIMERO4.ASM

        LIST     16F84

        ORG      0
        movlw    0x05
        movwf    0x0c
        movlw    0x02
        movwf    0x0d
        addwf    0x0c,1

        END
```

PROGRAMACIÓN DEL PIC CON ENTRADAS Y SALIDAS EXTERIORES

A continuación vamos a realizar programas de aplicación que nos permitan ingresar “variables lógicas de entrada”, se ejecute un determinado algoritmo de control y luego los resultados, se presenten como “variables lógicas de salida”, en los respectivos pines del microcontrolador.
El PIC16X84 tiene dos puertas (puertos) de entrada / salida, denominadas puerta A y puerta o puerto B. La puerta A tiene 5 líneas de entrada / salida (RA0, RA1, RA2, RA3, RA4). La puerta B tiene 8 líneas de entrada / salida (RB0.....RB7). Cualquiera de estas líneas puede ser entrada o salida.
Todos los recursos del PIC se manejan como registros de 8 bits que están implementados en la memoria de datos RAM, denominados, registros específicos.
El valor de los datos o variables que entran o salen por las puertas PA y PB, están materializados en dos posiciones de la memoria RAM que en este caso en particular, están en la dirección 5 y 6 del banco cero. Como hemos mencionado anteriormente la memoria de datos esta dividida en dos bancos, banco 0 y banco 1.

MEMORIA RAM

DIREC	BANCO 0	BANCO1
00		
01		
02		
03	ESTADO	ESTADO
04		
05	PUERTA A	TRIS A
06	PUERTA B	TRIS B
07 ----- ----- 7F		

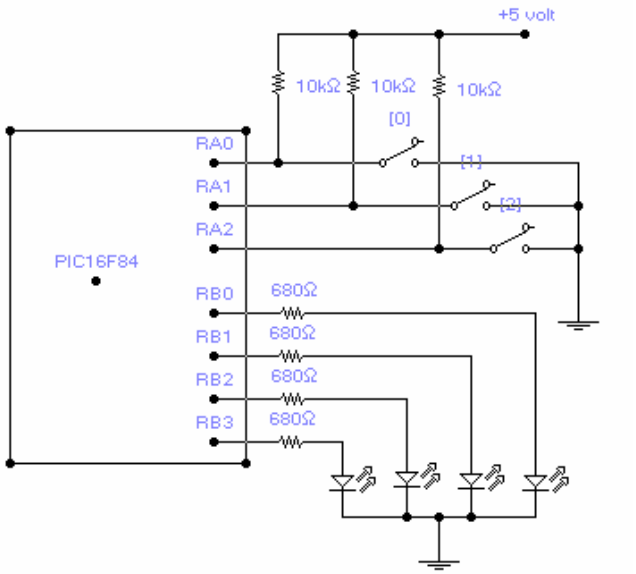
Para configurar las líneas de estos puertos como entradas o salidas, existen dos registros TRIS A y TRIS B, que se encuentran en la misma dirección de los registros PUERTA A y PUERTA B, pero en el banco 1 de la memoria de datos.
Colocando un "uno"(1) en los bits de TRIS A o TRIS B, se configuran como "entradas" las líneas de PA o PB. Serán "entradas" aquellas líneas que tengan el bit en "1" y "salidas", las que tengan en "0", en los registros TRIS A y TRIS B.
Cuando se conecta la alimentación del PIC, o se reinicializa su funcionamiento mediante un "RESET", automáticamente se tiene acceso al banco cero. Para pasar al banco 1, debemos poner a "1" el bit 5 del llamado "REGISTRO DE ESTADO", que se encuentra duplicado en los dos bancos, en la dirección o posición de memoria 03 (Hex).
Por ejemplo, si queremos que todas las líneas del puerto A sean entradas, debemos cargar con "1" todos los bits del registro TRIS A; y si queremos que sean salidas todas las líneas del puerto B, debemos cargar con "0" todos los bits del registro TRIS B.
Como aplicación, vamos a desarrollar un programa, donde intervienen variables exteriores tanto de entrada como de salida del microcontrolador.
Para ello vamos a introducir tres nuevas instrucciones a saber:

bsf f,b: Pone a "1" el bit "**b**" del operando fuente "**f**", que es una posición de la memoria de datos o registros.
bcf f, b: Pone a "0" el bit "**b**" del operando fuente "**f**"

goto etiqueta : Provoca un salto incondicional en la ejecución del programa hasta la instrucción que vaya precedida por el nombre de la etiqueta. Esta instrucción carga al contador de programa con la dirección de la instrucción que esta referenciada con la etiqueta, provocando un cambio de la secuencia normal del programa.

PROBLEMA : SEGUNDO.ASM

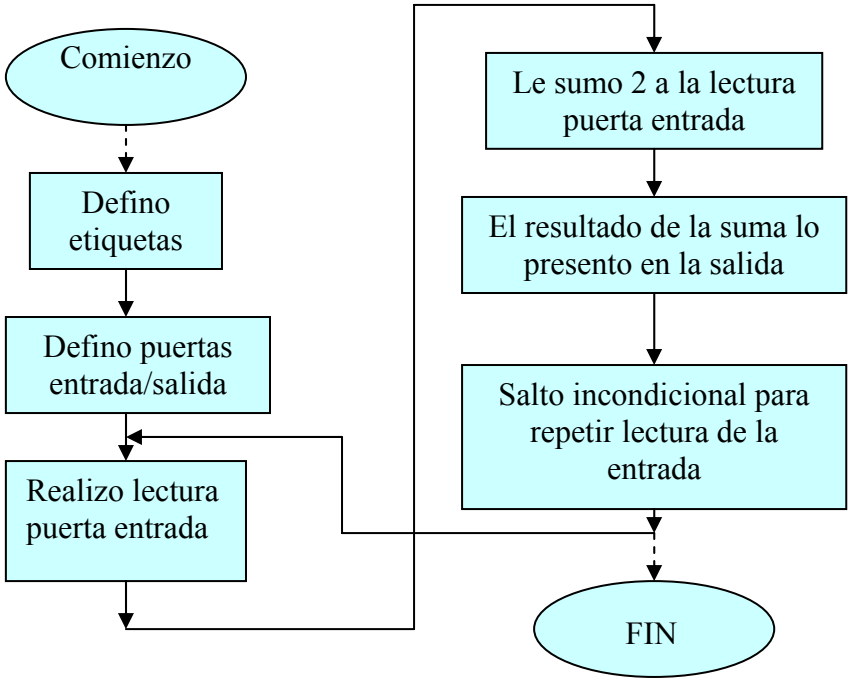
Se colocan tres interruptores en las líneas RA0, RA1 y RA2 de la puerta PA de un PIC16F84 y cuatro diodos Leds. En las lineas RB0, RB1, RB2, y RB3 de la puerta PB como muestra la figura



Mediante los tres interruptores, se introduce un numero binario de tres bits , de forma que si el interruptor esta abierto, coloca un "1" y si esta cerrado coloca un "0".
Realizar un programa con las instrucciones del PIC (nemónico), denominado "SEGUNDO:ASM" que comienza leyendo el número binario introducido por los interruptores, luego suma 2 unidades a este valor y visualiza en los diodos Leds el resultado binario de la operación.
Los diodos Leds apagados representan un "0" y encendidos un "1".

Solución:
Resulta siempre conveniente en todos los casos, realizar el diagrama de flujo, que nos permita presentar el desarrollo del programa especialmente cuando este presenta bifurcaciones como ser saltos condicionales, incondicionales, interrupciones, etc.
Luego con un editor de textos (utilizaremos el DOS del sistema operativo de la PC), crearemos un archivo de textos, con nombre "SEGUNDO.ASM" .
En este archivo se debe indicar el tipo de PIC que se va a utilizar y el sistema de representación de números que se va a emplear, para que el "programa ensamblador" pueda interpretarlo también se debe indicar, donde se colocara la primera instrucción dentro de la memoria de instrucciones del PIC. Se colocaran etiquetas, si se quiere trabajar con ellas y luego se desarrollara el programa en el lenguaje ensamblador, resultando conveniente agregar a cada instrucción, el comentario de su finalidad, dentro del programa.

Diagrama de flujo de "SEGUNDO:ASM"




```
;SEGUNDO.ASM Programa que lee el numero binario introducido mediante 3
;interruptores conectados a la puerta PA (RA0,RA1,RA2), luego suma 2
;unidades a ese valor y visualiza el resultado mediante 4 diodos led
;conectados a la puerta PB (RB0, RB1,RB2,RB3).

;-----------------------------------------------------------------------

        LIST    P=16C84         ;Comando que indica el PIC usado
        RADIX   HEX             ;Los valores se representar en hexade-
                                ; cimal
;-----------------------------------------------------------------------

        PUERTAA EQU     0X05    ;La etiqueta "PUERTAA" queda
                                ;identificada
                                ;la dirección 0x05, que si corresponde
                                ; con el banco 0 es el valor de la
                                ;puerta A
                                ; y si es del banco 1 con el de trisA
        PUERTAB EQU     0X06    ;Equivalencia de la etiqueta PUERTAB
        ESTADO  EQU     0X03    ;Estado corresponde con el valor 0x03.
        W       EQU     0       ;Identifica W con el valor 0.
```

```
;-------------------------------------------------------------------

        ORG     0               ;Comando que indica al Ensamblador la
                                ;dirección de la memoria donde se
                                ;situar  la instrucción siguiente

;-------------------------------------------------------------------


        bsf     ESTADO,5        ;Pone a 1 el bit 5 de ESTADO para
                                ;direccionar
                                ;la pagina 1 de la memoria de datos.
        movlw   0xff            ;W <-- FF(Hex)
        movwf   PUERTAA         ;W --> TRISA
        movlw   0x00            ;W <-- 0
        movwf   PUERTAB         ;W -->TRISB (líneas salida puerto B)
        bcf     ESTADO,5        ;Pone a 0 el bit 5 de ESTADO pasando a
                                ;acceder al banco 0.

inicio  movf   PUERTAA,W        ;W <-- PUERTAA. Se introduce el valor
                                ;binario de los interruptores.
        addlw   0x02            ;W <-- W + 2
        movwf   PUERTAB         ;W --> PUERTAB. El valor de W sale por
                                ; por las líneas de PB a los led.
        goto    inicio          ;Salta a la instrucción precedida por
                                ;la etiqueta de inicio.
        END
```

RESOLUCIÓN DE AUTOMATISMOS COMBINACIONALES

Tenemos tres métodos prácticos para confeccionar el programa que resuelva este tipo de automatismo.
El primero método, consiste en resolver la función lógica que cumple con el automatismo propuesto, utilizando instrucciones que realizan operaciones lógicas entre registros (AND ,OR y NOT).
El segundo método, consiste en guardar en la memoria de datos, “la tabla de la verdad” del automatismo combinacional. Utilizando el direccionamiento indexado e indirecto, los valores lógicos de las variables de entrada, se convierten en direcciones, que direccionan la “tabla de la verdad” y presentan su contenido en las salidas.
El tercer método, consiste en realizar la tabla de la verdad, mediante la llamada a una rutina con la instrucción “**call**” y su retorno, con el valor lógico de la salida ,mediante la instrucción “**retlw**”.

Resolución por el 1° método:

Para ello, utilizaremos las siguientes instrucciones del repertorio disponible del PIC 16X84.

Andwf f,d : operación AND entre W y f; resultado en W si d=0. Resultado en f sí d=1.

Comf f,d : operación not entre W y f; resultado en W si d=0. Resultado en f si d=1.

Iorwf f,d: Operación OR entre Wy F; resultado en W si d=0. Resultado en f si d=1.

Xorwf f,d : Operación XOR entre W y f ; resultado en W si d=0. Resultado en f si d=1.

Nota: Las operaciones lógicas presentadas, se realizan entre los registros direccionados (f) y el registro de trabajo W encolumnados bit a bit.

Rrf f,d : Rotación hacia la derecha, (a través del señalizador de acarreo C), del registro f. El resultado queda en W si d=0. El resultado queda en f si d=1.

Rlf f, d: Es similar al caso anterior pero rotación hacia la izquierda.

Nota : El señalizador de acarreo se encuentra en el bit 0 del registro de Estado (0x 03 de ambos bancos de memoria).

Problema:

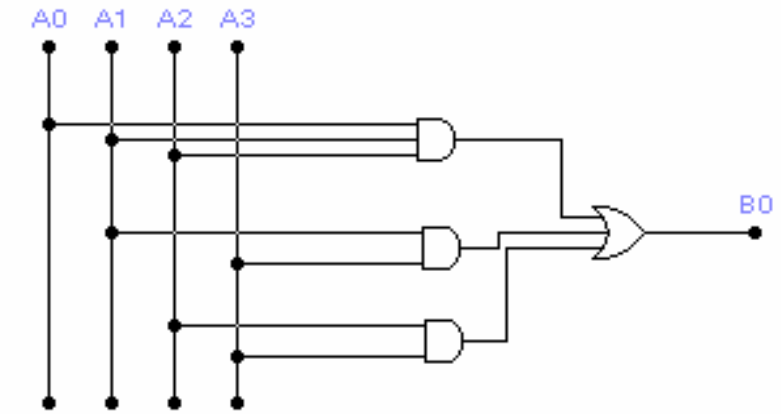
Se dispone de cuatro motores conectados a una misma barra de alimentación de energia eléctrica. Por razones de limitación de potencia conectada, se deberá realizar un automatismo (enclavamiento) que actúe sobre sobre los contactores de los motores, o activar una alarma, que indique, que la potencia conectada supera los 18 KVA.

Solución: Este problema ha sido resuelto en el apunte "Sistemas lógicos digitales" pagina 36, con las siguientes variables y solución de la función lógica.

A = A0 = 4 KVA
B = A1 = 6 KVA
C = A2 = 8 KVA
D = A3 = 12 KVA

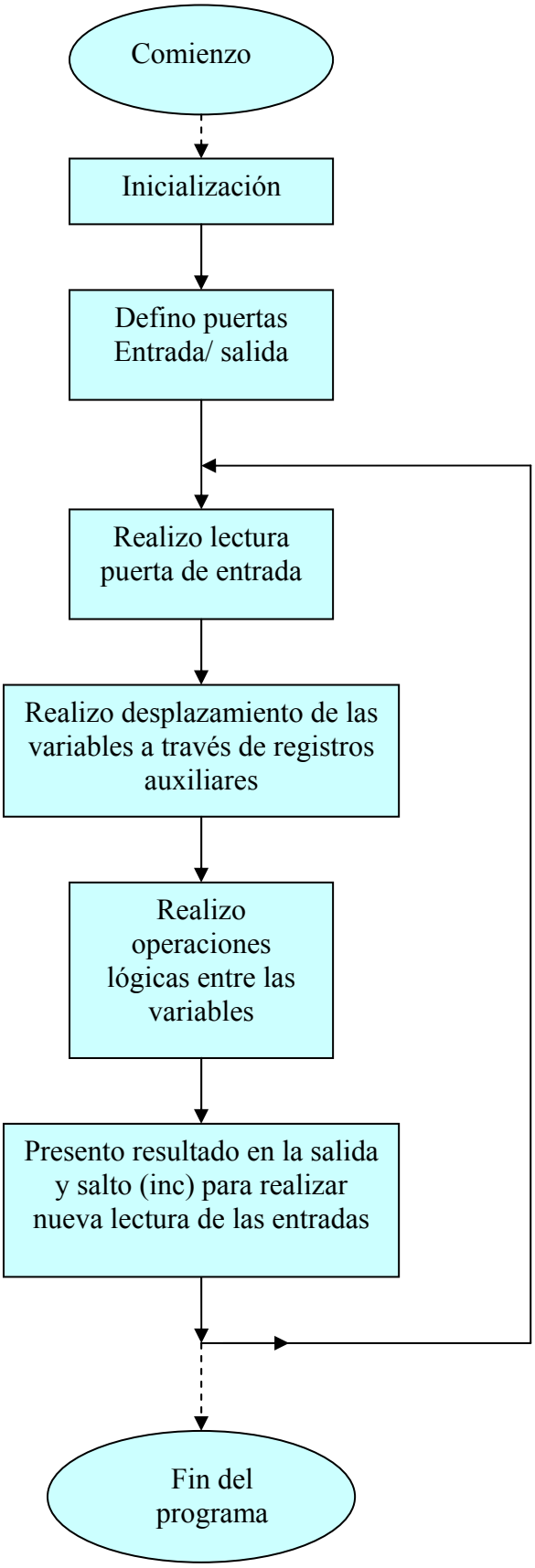
Y = B0 = 1 si potencia conectada > 18 KVA
Y = B0 = 0 " " " < 18 KVA

B0 = A0. A1. A2 + A1. A3 + A2. A3



Para resolver, por programa la función lógica " B0", debemos tener en cuenta que las variables lógicas A0....A3 se encuentran en distintas columnas (orden de bit) por lo cual para poder realizar las operaciones AND y OR , deberemos realizar los corrimientos de esta variables, para que queden encolumnadas. Para ello deberemos ingresarlas en registros auxiliares, previamente desplazadas por ejemplo a la columna de la variable A0 y luego proceder con las operaciones lógicas que indique "la función lógica " a resolver.
Como primer paso a la resolución, plantearemos el diagrama de flujo que nos permitirá posteriormente desarrollar el programa para ser ejecutado por el microcontrolador
El programa, lo desarrollaremos sin definir etiquetas.

Diagrama de flujo de "MOTORES1.ASM"



Desarrollamos a continuación el programa, sin definir etiquetas:

```
;                MOTORES1.ASM
;                ============
;
; B0 = A0.A1.A2 + A1.A3 +A2.A3
;Programa que permite controlar la cantidad de motores que se
;conectan a una barra de alimentación eléctrica, que tiene
;limitaciones respecto a la máxima potencia eléctrica entregada.


            LIST    P=16F84
            RADIX   HEX

            ORG     0
            goto    INICIO
            ORG     5


   INICIO   clrf    0x05        ;llevo a cero r05 (entradas)
            clrf    0x06        ;llevo a cero r06 (salidas)
            bsf     0x03,5      ;selecciono el banco uno
            movlw   0xff        ;ff>w
            movwf   0x05        ;w>trisa A son entradas
            clrf    0x06        ;B son salidas
            bcf     0x03,5      ;selecciono el banco cero


   BUCLE    movf    0x05,0      ;entradas A>w
            movwf   0x0C        ;w>0C direcc. Memoria datos. Entrada
                                     ;"Ao"
            movwf   0x0D        ;w>0D     "        "     "
            rrf     0x0D,1      ;desplazo A1 a la columna Ao y lo
                                ;deposito
                                ;en 0D.Entrada "A1"
            rrf     0x0D,0      ;desplazo A2 a la columna A0
.                               ;resultado >w
            movwf   0x0E        ;w>0E direcc.mem datos Entrada A2
            rrf     0x0E,0      ;desplazo A3 a la columna
                                 ;resultado>w
            movwf   0x0F        ;w>0F direcc.mem datos Entrada A3
            andwf   0x0E,0      ;A2.A3>w
            movwf   0x10        ;w>10 direcc.mem datos producto "A2.A3"
            movf    0x0F,0      ;0F>w
            andwf   0x0D,0      ;A1.A3>w
            movwf   0x11        ;w>11 direcc.mem datos producto "A1.A3"
            movf    0x0C,0      ;0C>w
            andwf   0x0D,0      ;Ao.A1>w
            andwf   0x0E,0      ;Ao.A1.A2>w
            iorwf   0x11,0      ;Ao.A1.A2+A1.A3>w
            iorwf   0x10,0      ;Ao.A1.A2+A1.A3+A2.A3>w
            andlw   0x01        ;10 producto logico con w resultado wo
            movwf   0x06        ;w>06 puerta B salida
            goto    BUCLE
            end                 ;fin del programa
```

Para resolver por el 2º método, con la tabla de la verdad, debemos previamente , tratar el tema relacionado al direccionamiento de la memoria de datos.

CAPITULO 3 : Direccionamiento, instrucciones de salto condicional y rutinas

DIRECCIONAMIENTO DE LA MEMORIA DE DATOS

Para los microcontroladores PIC de la gama media, la memoria de datos esta organizada para alojar un máximo de 4 bancos de 128 bytes cada uno. Los bits RP1 y RP0 del registro de ESTADO, se utilizan para seleccionar el banco y se necesitan otros 7 bits, para elegir una de las 128 posiciones del banco seleccionado.
Para el caso del PIC 16F84, solamente tiene disponibles dos bancos; el banco 00 y el 01. (Banco 0 y banco 1). El banco 0 tiene en sus primeras direcciones, los "**registros de propósito especifico**" **SFR**, en una cantidad de 11, ubicados entre las direcciones 0x 00 y 0x 0b (la dirección 0x 07, no es operativa). Desde la dirección 0x 0c hasta la 0x 4f dispone de 68"**registros de propósito general**" **GPR**. Todos estos registros son de 1 byte.
El banco 1 dispone también de la misma cantidad de "**SFR**" y los "**GPR**" están mapeados sobre el banco 0, es decir que si nos encontramos en el banco 1 y queremos direccionar un "**GPR**" desde este banco, accederemos a los "GPR" del banco 0. Como conclusión El PIC16F84, dispone solamente de 68 registros de propósito general., accesibles desde el banco 0 ó banco 1.
Como este PIC, solamente dispone de dos bancos, siempre RP1= 0 y RP0 = 0 para trabajar sobre el banco 0 y RP0 = 1 para pasar al banco 1. Veamos un esquema simplificado de esta memoria para el PIC16F84 (memoria de datos RAM Volátil)

(R. ESTADO)

RP1	RP0

Dirección
Del banco

BANCO

	00	01	10	11
00 0b	SFR 11 registros	SFR 11 registros	No implementado	No implementado
0c 4f	GPR 68 bytes	Mapeado en el banco cero	No implementado	No implementado
50 7f	No implementado	No implementado	No implementado	No implementado

Direccionamiento directo:

En este caso el operando que utiliza la instrucción en curso, se referencia mediante su dirección, que viene incluida en el código de operación de la misma, concretamente en los 7 bits de menos peso. Para Seleccionar el banco, como dijimos, lo hacemos con los bits RP0 y RP1 del registro de ESTADO.

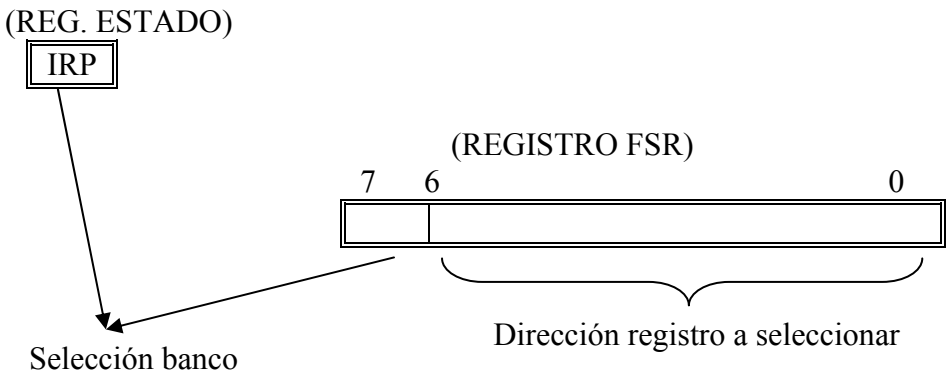
FORMATO DE LA INSTRUCCIÓN (14 BITS)

13 … 8	7	6 … 0
CÓDIGO OPERACIÓN	D	F(DIRECCIÓN DEL REGISTRO)

D= 1 El registro destino es f
D= 0 El registro destino es W
Ejemplo: ***add f,d*** como ser: ***add 0x0c,1***
add: *Codigo de operación (6 bits)* , sumar a W el contenido del registro fuente 0x0c
f≡ 0x0c:*direccion del registro fuente (7 bits)*, en la memoria de datos (GPR)
d≡1: registro destino (1bit) , en este caso el resultado se deposita en 0x0c

Direccionamiento indirecto:

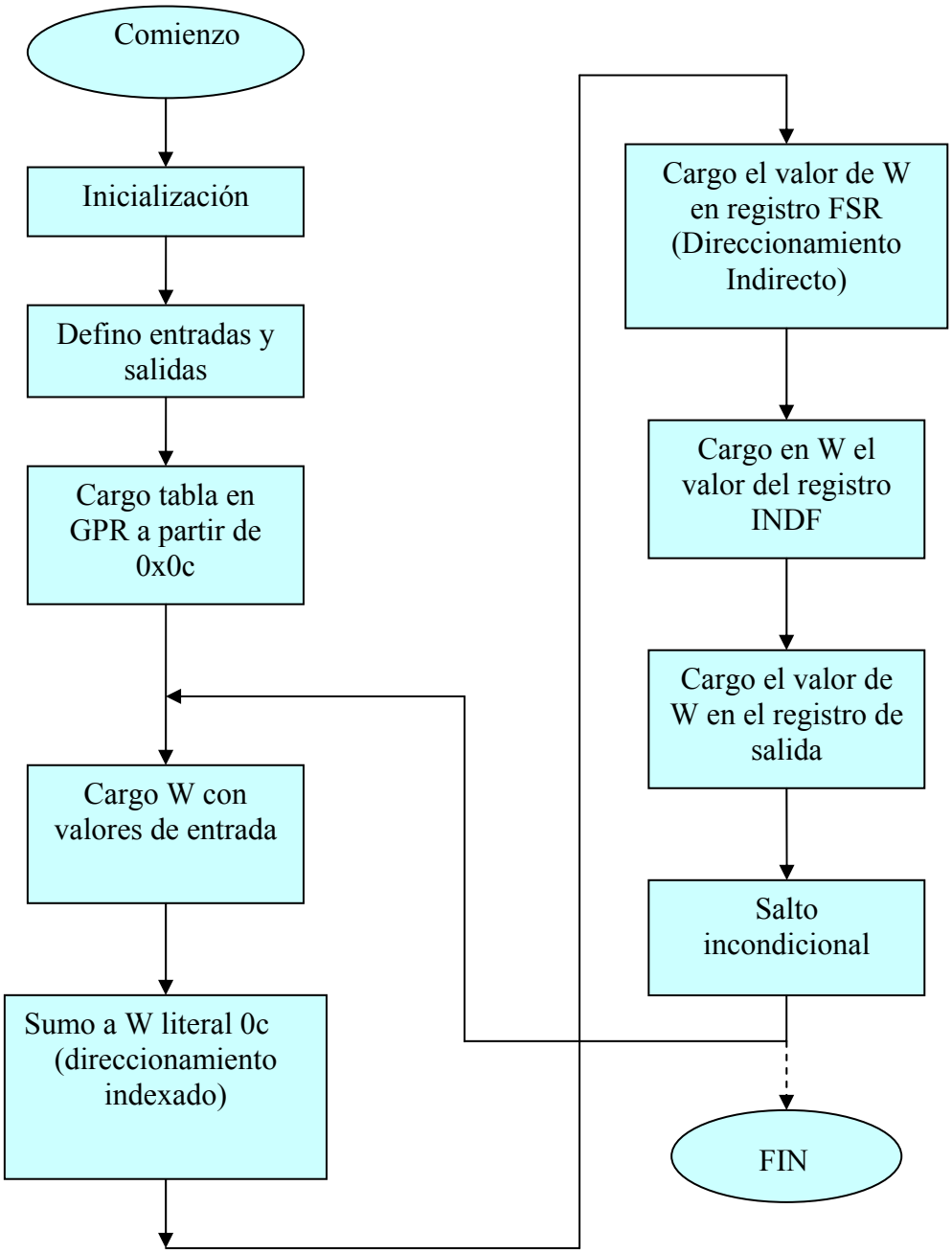
Este modo de direccionamiento se usa cuando en una instrucción se utiliza como operando el registro INDF, que ocupa la dirección 0 de ambos bancos. En realidad, el registro INDF no esta implementado físicamente y cuando se le hace referencia, se accede a la dirección de un banco especificada con los 7 bits de menos peso del registro FSR. El bit de mayor peso de FSR junto al bit IRP del registro de ESTADO se encarga de seleccionar el banco a acceder, mientras que los 7 bits de menor peso, apuntan a la posición. Como en los PIC16X84 solamente tenemos dos bancos el bit IRP= 0 siempre.



Como conclusión para este tipo de direccionamiento, podemos decir lo siguiente: S i queremos incorporar un dato sobre un registro “GPR”, debemos mediante una instrucción incorporar el dato en el registro INDF y colocar la dirección del “GPR” en el registro FSR.
De la misma manera procederemos a rescatar un dato que está en un “GPR”. Apuntaremos a la dirección de ese “GPR” incorporándola en el registro FSR y rescataremos el dato en el registro INDF. En todos los casos, incorporando o retirando datos o direcciones de los registros, se deben realizar mediante instrucciones de “movimiento de datos”.
Veremos a continuación, una aplicación de direccionamiento indirecto.

Resolución automatismo MOTOREST.ASM:

El procedimiento que aplicaremos, será grabar la “tabla de verdad” de la funcion logica del automatismo en la memoria de datos, utilizando para ello, los registros de propósito general “GPR”, a partir de la dirección 0x0c. Luego convertiremos los valores lógicos de las variables de entrada, mediante “**indexacion**”, en direcciones que apunten a la “tabla de la verdad” . A continuación, por medio del direccionamiento indirecto, obtendremos los datos de la tabla, para posteriormente, presentarlo en la salida del microcontrolador. Veamos el diagrama de flujo:



A continuación desarrollaremos en archivo de texto, el programa en lenguaje asembler , Correspondiente a MOTOREST.ASM

```
                ;MOTOREST.ASM


LIST        P=16F84 ;defino el tipo de micro
RADIX       HEX     ; escritura en hexadecimal

ORG         0       ; 1øinstrucc en 0x00
goto        INICIO;salto incondicional
ORG         5       ; proxima inst. en 0x05
```

```
INICIO   bsf        0x03,5;paso al banco 1
         movlw      0xfe  ;cargo literal fe en W
         movwf      0x06  ; W=>tris a defino entradas
         movlw      0x1f  ;cargo literal 1f
         movwf      0x05  ;W=>tris b defino salidas
         bcf        0x03,5;paso al banco 0

         movlw      0x00  ;cargo la tabla de la verdad
         movwf      0x0c  ;de la funcion logica del
         movwf      0x0d  ;automatismo, desde la direcc
         movwf      0x0e  ;0x0c hasta la 0x1b,cargando
         movwf      0x0f  ;en W los valores 0 y 1,despla-
         movwf      0x10  ;zandolos a las respectivas
         movwf      0x11  ;direcciones de la tabla.
         movwf      0x12
         movwf      0x14
         movwf      0x15

         movlw      0x01
         movwf      0x13
         movwf      0x16
         movwf      0x17
         movwf      0x18
         movwf      0x19
         movwf      0x1a
         movwf      0x1b


BUCLE    movf       0x05,0 ;cargo entradas en W
         andlw      0x0f   ;enmascaro valor de W para
                           ;evitar errores en la suma
         addlw      0x0c   ;sumo a W el literal 0c
         movwf      0x04   ;direcc. indirecto de la tabla
         movf       0x00,0 ;obtengo el dato de la tabla
         movwf      0x06   ;presento el dato en la salida
         goto       BUCLE  ;salto incondicional a entrada
         end
```

Mas adelante, veremos otro método alternativo (3°), para resolver automatismos combinacionales, con instrucciones especiales como son las llamadas a rutinas y retornos (instrucciones ***call y retlw k***).

PROGRAMAS CON INSTRUCCIONES DE SALTO CONDICIONAL

Para los microcontroladores PIC de la gama media tenemos solamente cuatro instrucciones de salto condicional. Dos de ellas, testean un bit de un registro y según valga 1 ó 0, saltan la instrucción siguiente a la condicional. Las otras dos instrucciones incrementan o decrementan un registro y la posibilidad de salto se efectúa si con esa operación, el valor del registro es 1 ó 0. El salto en estas dos últimas instrucciones tambien es de una instrucción posterior a la condicional.
Estas instrucciones, si no producen el salto, tardan un ciclo de instrucción en ejecutarse; si producen el salto, tardan el doble. Veamos estas instrucciones.

Btfsc f,d: Explora un bit(d) de f y salta si vale cero.

Btfss f,d: Explora un bit (d) de f y salta si vale uno.

Decfsz f,d: Decrementa el registro f; el resultado lo deposita en f si d=1 y en W si d=0 y salta si f es igual a cero.

Incfsz f,d: incrementa el registro f (una unidad); el resultado lo deposita en f si d=1 y en W si d=0. Salta si f es igual a uno.

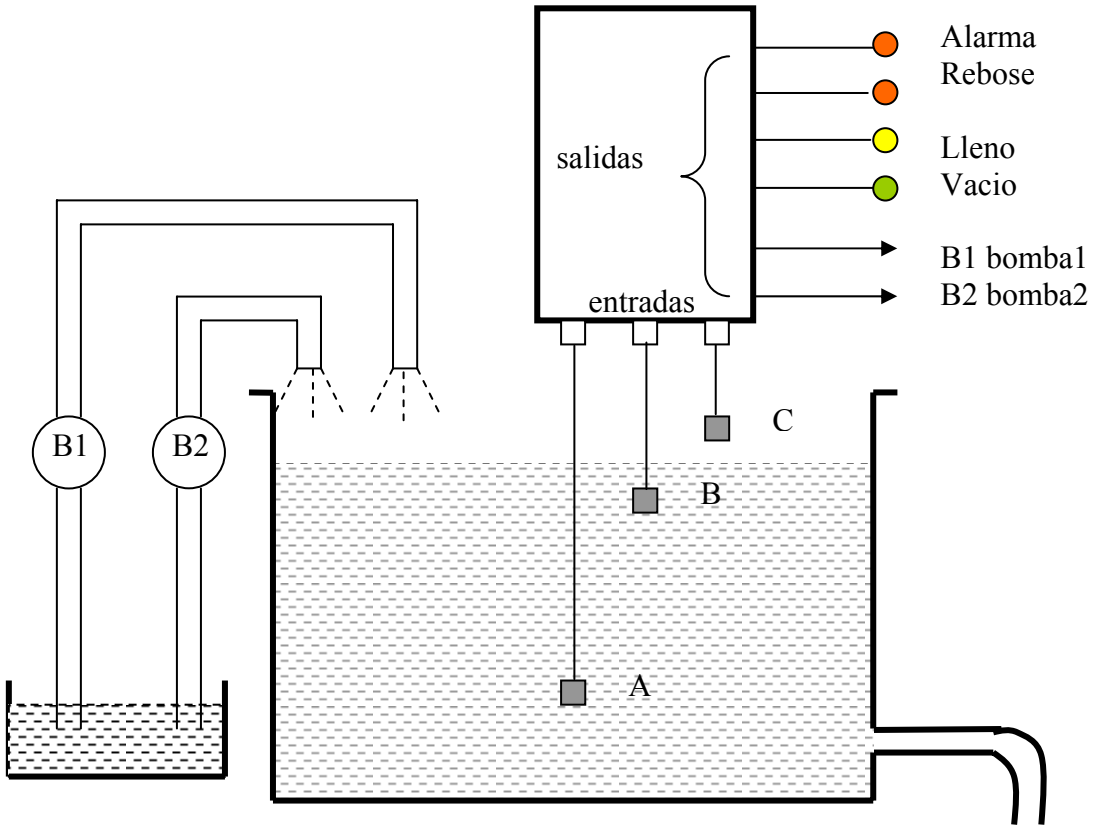
Como una primera aplicación utilizando instrucciones de salto condicional, realizaremos una porción de programa, de uno mas amplio. En este programa, utilizando el direccionamiento indirecto, la instrucción de incremento de registro "incf f,d" y la de salto condicional "btfss f,d", nos permite por ejemplo borrar los "GPR", desde la dirección 0x0c hasta la dirección 0x20.

```
                    ;INCF~IND.ASM
;programa que permite, utilizando el direccionamiento indirecto,
;borrar los registros de proposito general "GPR"
;desde las direcciones 0x0c hasta la 0x20 inclusive.
;(21 registros)
          LIST      P=16F84

          RADIX     HEX
          ORG       0

          movlw     0x0c    ;cargo en W literal 0c
          movwf     0x04    ;cargo el valor de W en 0x04(FSR)
                            ;direccionamiento indirecto
          movlw     0x00    ;cargo en W el literal 0x00
          movwf     0x00    ;cargo indirectamente a través
                            ;del registro INDF el valor 0x00
  BUCLE   incf      0x04    ;incremento en una unidad FSR
          clrf      0x00    ;borro, a través de INDF la direcc.
                            ;apuntada por FSR
          btfss     0x04,5  ;salto condicional. reviso el bit 5
                            ;de FSR y salto si vale 1
          goto      BUCLE   ;salto incondicional a la etiqueta
                            ;indicada BUCLE
          end
```

RESOLUCIÓN DE UN AUTOMATISMO LÓGICO SECUENCIAL

En este ejercicio realizaremos el control de un proceso industrial relativamente simple, utilizando la CPU del microcontrolador para la toma de desiciones, su memoria FLASH para guardar las instrucciones del programa, su memoria RAM volátil para almacenar datos de entradas, y los puertos, uno de lectura para los sensores de entrada (RA0, RA1, RA2) y otro de escritura para enviar las señales a los actuadores y señalizadores (RB0.....RB5).
Este ejercicio ha sido resuelto en el libro "SISTEMAS MICROPROCESADORES" del autor J.M.ANGULO USATEGUI. También fue resuelto en el libro " INTRODUCCIÓN A LOS MICROCONTROLADORES" del autor J. A. GONZALEZ VASQUEZ, usando los microcontroladores 8052/8051 (INTEL 8 bits). Para nuestro caso , lo resolveremos con los PIC16X84.
Este automatismo, es muy conocido por los profesionales de la especialidad, como así también muy usado y resuelto por medios convencionales.
El automatismo en sí es del tipo combinacional; nosotros para mejorar su perfomance también le daremos características de tipo secuencial.
El método que emplearemos para desarrollar el programa, será el de escritura en la memoria de datos de la "tabla de la verdad" y direccionamiento indexado e indirecto.

Esquema del control



Proceso :
Se trata de controlar el nivel de líquido de un depósito, utilizando tres sondas detectores de niveles A, B y C y dos bombas B1 y B2, con indicadores de nivel y alarma por falla en los detectores. En este proceso, tenemos desde el punto de vista con el mundo exterior, los tres tipos de señales principales de un automatismo lógico:
-Entrada de la información, a través de los “SENSORES”.
-Actuación sobre los elementos finales denominados “ACTUADORES”.
-Señalizadores del proceso, denominados “INDICADORES”.

Los sensores (sondas):
Son la entrada de la información al proceso de automación o etapa de “decisión”, resuelto por el microcontrolador.
La “sonda “A” detecta el mínimo nivel de líquido. Por debajo de éste nivel, se indicará “VACIO” en el cuadro de indicadores.
La sonda “B” señala el nivel óptimo y cuando se alcance, se indicará “LLENO”.
La sonda “C”, señala el nivel más alto; por encima de éste nivel el tanque se desbordará.
Se indicará “REBOSE” en el cuadro de indicadores.
Estas entradas con circuitos antirrebotes si provienen de contactos secos de lámina, ingresarán al puerto RA del microcontrolador, definido por programa como puerto de entrada. C => RA0; B => RA1 ; A => RA2.

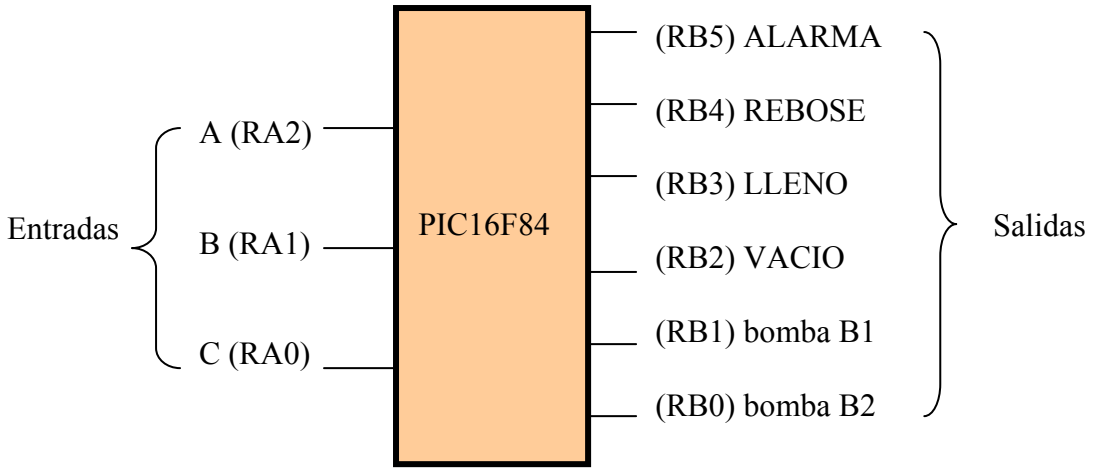
Actuadores e indicadores:
Estos representan las salidas del microcontrolador que actuarán sobre las bombas de líquidos electromecánicas, que se activarán o desactivarán en las siguientes condiciones propuestas:

Nivel del líquido en ascenso

-Si el nivel del líquido no supera la sonda “A”, las dos bombas estarán activadas y se indicará “VACIO”.
-Cuando se supere el nivel “A” desaparecerá la indicación “VACIO”; las dos bombas seguirán funcionando.
-Cuando se llegue al nivel “B”, la bomba “B2” se desactivará y se indicará “LLENO”.
La bomba “B1” seguirá activada.
-Cuando se llegue al nivel “C”, la bomba “B1” se desactivará, quedando ambas bombas fuera de servicio. El indicador señalizará “REBOSE”.

Nivel del líquido en descenso

Cuando el nivel del líquido comience a descender, desde el nivel de “REBOSE”, se modificará la lógica de control, con la finalidad de evitar continuos arranque y parada de la bomba “B1” a consecuencia de la variación del nivel del liquido cuando se encuentre próximo al “REBOSE” (sonda “C”). Las condiciones serán las siguientes:
-Cuando el nivel del líquido en bajada se encuentre entre los niveles “C” y “B”, ninguna de las dos bombas estarán activadas. El indicador señalizará “LLENO”.
-Cuando se supere el nivel “B” en descenso, se activará la bomba “B1”; la bomba “B1” seguirá desactivada y desaparecerá la indicación de “LLENO”.
-Cuando el nivel llegue al nivel “A”, se activara la bomba “B2”, se indicará “VACIO” y se modificara la secuencia lógica de control pasando a la correspondiente al ascenso del nivel del líquido.
- Cuando se produzca un fallo en las sondas de entrada por “discordancia”, ambas bombas se desactivaran y se indicará “ALARMA”.
Salidas: RB0 => B2 ; RB1 => B1; RB2 => VACIO ; RB3 => LLENO
RB4 => REBOSE ; RB5 => ALARMA



A continuación realizaremos las tablas de la verdad para las dos condiciones: en subida y en bajada.

Tabla de la verdad en subida

|direc |entradas | salidas |

HEX	A	B	C	AL	REB	LLE	VAC	B1	B2	OBSERVACIONES
0C	0	0	0	0	0	0	1	1	1	B1-B2 activa. Señala VACIO
0D	0	0	1	1	0	0	0	0	0	ALARMA
0E	0	1	0	1	0	0	0	0	0	ALARMA
0F	0	1	1	1	0	0	0	0	0	ALARMA
10	1	0	0	0	0	0	0	1	1	B1-B2 activa. No señala alarma
11	1	0	1	1	0	0	0	0	0	ALARMA
12	1	1	0	0	0	1	0	0	1	B1 desactiva-B2 activa. Señala LLENO
13	1	1	1	0	1	1	0	0	0	B1-B2 desactiva. LLENO Y REBOSE

Cuando se produce la condición de “REBOSE” se pasa a la secuencia “líquido en descenso”, cuya tabla de la verdad es la que sigue a continuación

Tabla de la verdad en descenso

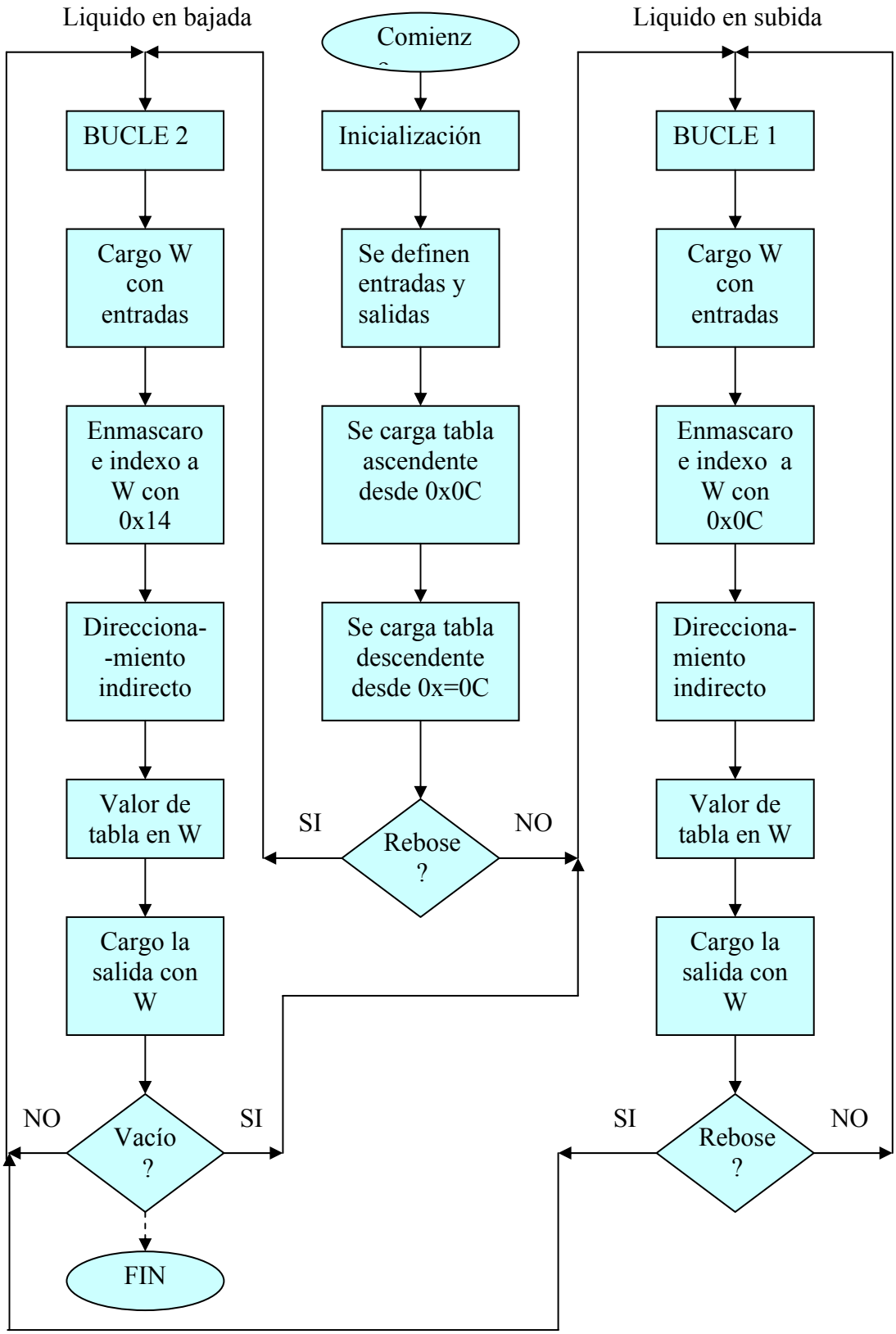
Direc | Entradas | Salidas |

HEX	A	B	C	AL	REB	LLE	VAC	B1	B2	OBSERVACIONES
14	0	0	0	0	0	0	1	1	1	B1-B2 activas. Señala VACIO
15	0	0	1	1	0	0	0	0	0	ALARMA
16	0	1	0	1	0	0	0	0	0	ALARMA
17	0	1	1	1	0	0	0	0	0	ALARMA
18	1	0	0	0	0	0	0	0	1	B1 desactiva.-B2 activa. Sin señalización
19	1	0	1	1	0	0	0	0	0	ALARMA
1A	1	1	0	0	0	1	0	0	0	B1-B2 desactivas. Señala LLENO
1B	1	1	1	0	1	1	0	0	0	B1-B2 desactivas. LLENO y REBOSE

Con señalización “VACIO” , se pasa al secuencia “líquido en ascenso

DIAGRAMA DE FLUJO

Realizaremos a continuación, el diagrama de flujo que nos permitirá determinar el programa que, deberá ejecutar la CPU del microcontrolador, para cumplir con los objetivos del automatismo.



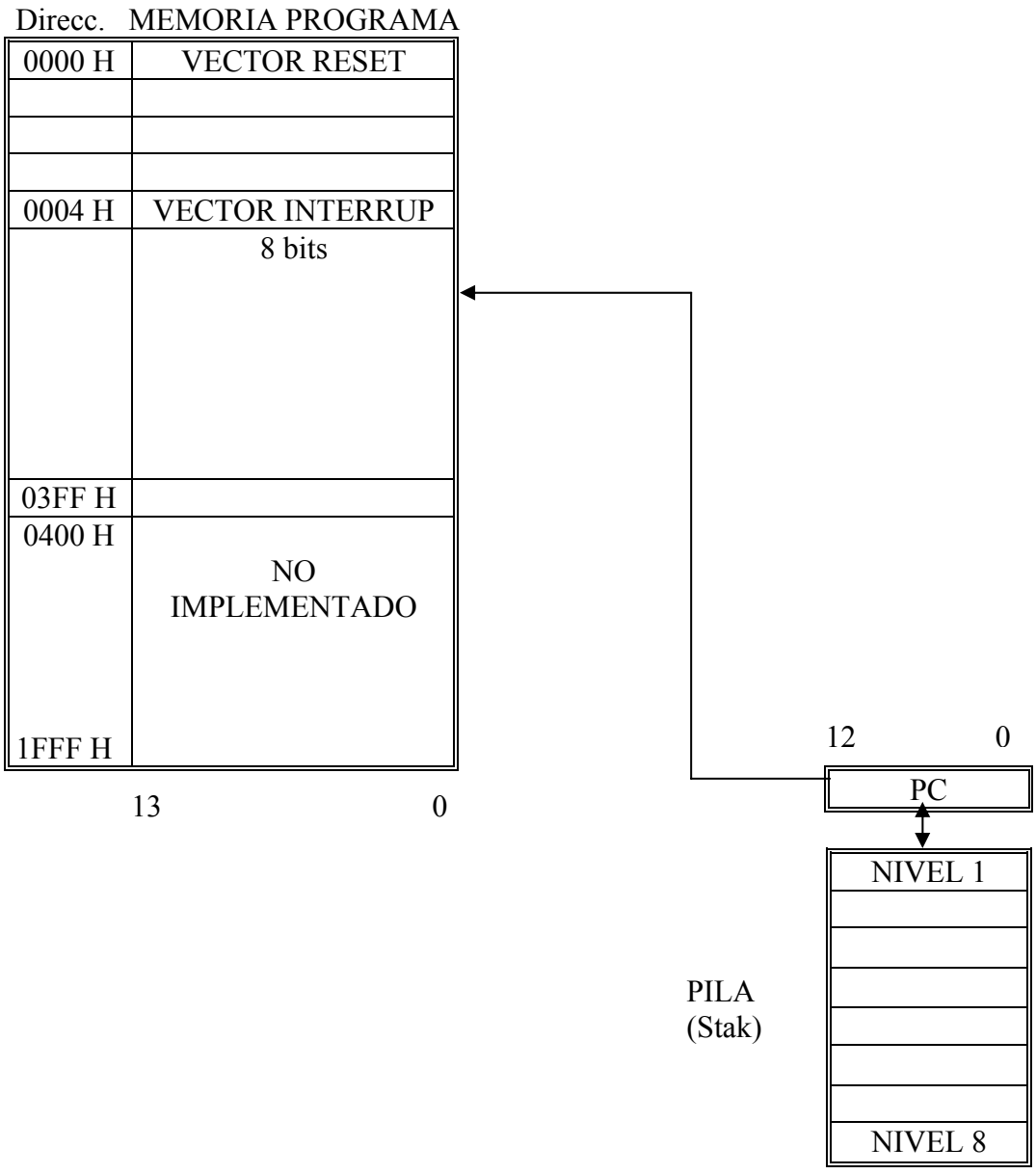
Desarrollamos a continuación el programa, en archivo de texto, denominado "NIVELTAN.ASM", escrito en lenguaje Asembler, sin definir etiquetas.

```
                    ;NIVEL TAN.ASM

;Programa que controla el nivel de líquido de un tanque a través
;de tres sondas "A"(VACIO), "B"(LLENO), "C"(REBOSE),con acciona-
;miento de dos bombas electromecánicas B1 y B2.Además con indi-
;caciones de "alarma","rebose","lleno" y "vacío"

                 LIST       P=16F84

                RADIX       HEX

                ORG         0       ;coloco en el vector reset la
                                    ;instrucción que sigue abajo
                goto        INICIO  ;salto incondicional a INICIO
                ORG         5       ;coloco en la dirección 0x05
                                    ;la próxima instrucción.

                INICIO   bsf          0x03,5 ;paso al banco 1
                movlw       0xff    ;cargo literal 0xff en W
                movwf       0x05    ;W=0x05.RA0..RA4 son entradas
                movlw       0xc0    ;cargo literal 0xc0 en W
                movwf       0x06    ;W=>0x06. RB0...RB5 son entradas
                bcf         0x03,5  ;paso al banco cero
                clrf        0x05    ;coloco en 0 entradas
                clrf        0x06    ;coloco en 0 salidas

              ; TABLA ASCENDENTE
              ; ----------------
                movlw       0x07    ;Guardo tabla Nø1 ascendente
                movwf       0x0c    ;a partir de la direcc. 0x0c
                movlw       0x20    ;hasta la direcc. 0x13
                movwf       0x0d
                movwf       0x0e
                movwf       0x0f
                movwf       0x11
                movlw       0x03
                movwf       0x10
                movlw       0x09
                movwf       0x12
                movlw       0x18
                movwf       0x13

              ;TABLA DESCENDENTE
              ;------------------

                movlw       0x07    ;Guardo tabla Nø2 descendente
                movwf       0x14    ;a partir de la direcc. 0x14
                movlw       0x20    ;hasta la direcc. 0x1B
                movwf       0x15
                movwf       0x16
                movwf       0x17
                movwf       0x19
                movlw       0x01
                movwf       0x18
                movlw       0x08
                movwf       0x1a
                movlw       0x18
                movwf       0x1b

                btfss       0x06,4 ;Salto condicional revisando
```

```
                              ;si se produjo REBOSE en salida
                              ; despues de actuar el V.reset
            goto      BUCLE1 ;salto condicional a BUCLE1
            goto      BUCLE2 ;salto condicional a BUCLE2

BUCLE1      movf      0x05,0 ; cargo entradas en W
            andlw     0x07   ; enmascaro W con literal 0x07
            addlw     0x0c   ;obtengo direccion indexada para
                             ;tabla ascendente. W+0x0c
            movwf     0x04   ;W=>FSR direccionamiento indirecto
            movf      0x00,0 ;obtengo dato tabla ascendente
                             ;a través del registro INDF
            movwf     0x06   ;W=>0x06 presento dato en salida
            btfss     0x06,4 ;salto condicional revisando si
                             ;se produjo alarma REBOSE en salida
            goto      BUCLE1 ;salto incondicional a BUCLE1
            goto      BUCLE2 ;salto incondicional a BUCLE2

BUCLE2      movf      0x05,0 ;cargo valor de entradas en W
            andlw     0x07   ;enmascaro W con literal 0x07
            addlw     0x14   ;obtengo dirección indexada para
                             ;tabla descendente W+0x14
            movwf     0x04   ;W=>FSR direccionamiento indirecto
            movf      0x00,0 ;obtengo dato tabla descendente
                             ;a través del registro INDF
            movwf     0x06   ;presento dato en salida
            btfsc     0x06,2 ;salto condicional revisando si
                             ;se produjo alarma VACIO en salida
            goto      BUCLE1 ;salto incondicional a BUCLE1
            goto      BUCLE2 ;salto incondicional a BUCLE2
            end              ;fin del programa
```

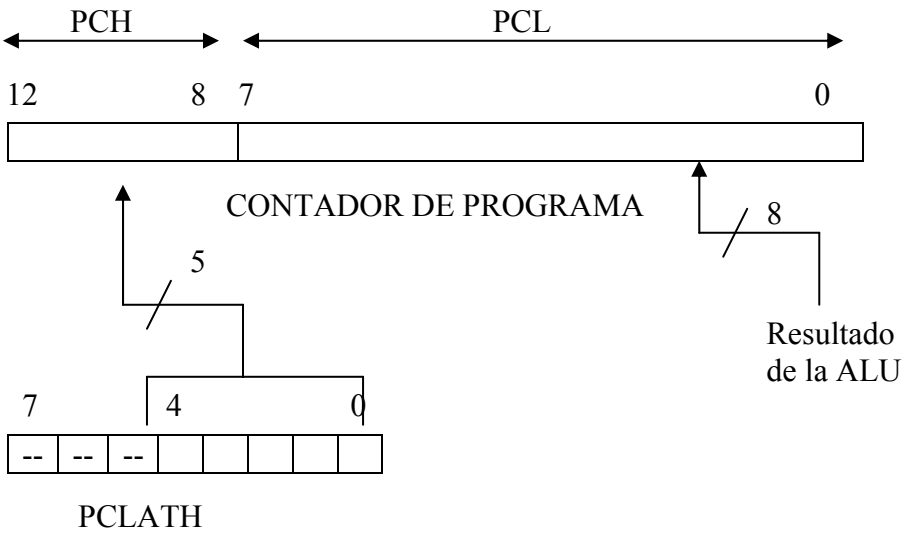
DETALLES DEL CONTADOR DE PROGRAMA Y LA PILA

Como ya sabemos, el “contador de programa” es un registro donde se carga la dirección de la memoria de programa, de la instrucción, próxima a ejecutarse.
La arquitectura de los PIC de la gama media, admite un mapa de memoria de programa, capaz de contener 8.192 instrucciones de 14 bits cada una. Este mapa se divide en páginas de 2.048 posiciones. Para direccionar 8 K posiciones, se necesitan 13 bits, que es la longitud que tiene el “contador de programa”. Sin embargo, el PIC 16X84 solo tiene implementadas 1024 (1K) posiciones, por lo que en el “contador de programa” (PC), se ignoran los tres bits de mayor peso, solo se tiene en cuenta los 10 bits de menor peso.
En la próxima figura vemos una representación de “la memoria de programa” y de “la pila” en el rango de direcciones que cubre el PIC16X84. Se observa que llega desde la dirección 0000 H hasta la 03FF H, o sea un total de 1024 posiciones, con palabras de 14 bits.

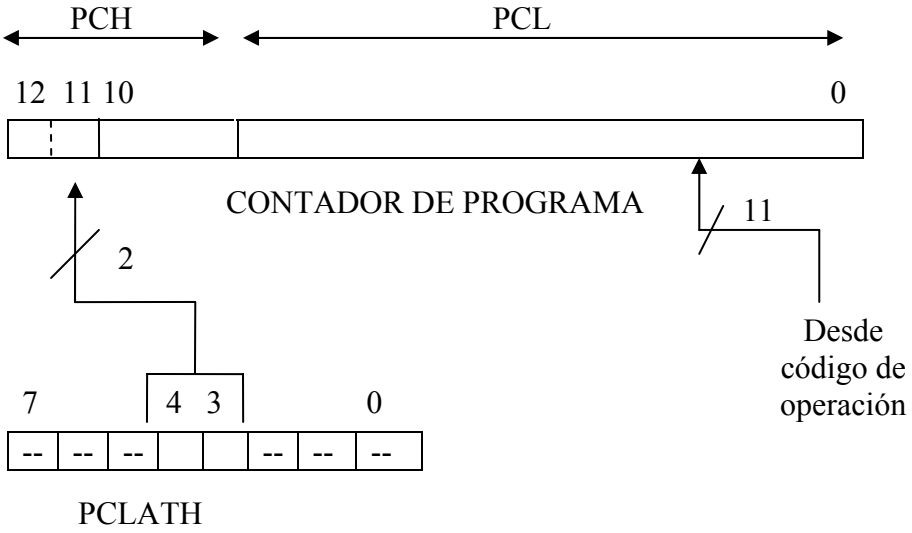


De la misma manera que todos los registros específicos que controlan la actividad del procesador, el contador de programa esta implementado sobre dos posiciones de la memoria RAM de datos. Cuando se escribe el contador de programa como resultado de una operación de la ALU, los 8 bits de menor peso del PC residen el registro PCL que ocupa repetido, la posición 2 de los dos bancos de la memoria RAM. Los bits de mas peso, PC<12 : 2>, residen en los 5 bits de menor peso del registro PCLATH, que ocupa la posición “0A” de los dos bancos de memoria RAM.
En las instrucciones de salto incondicional GOTO y CALL de la gama media, los 11 bits de menor peso del PC provienen del código de instrucción y los otros dos, de los bits PCLATH 3 y 4.
Con los 11 bits que se cargan en el PC desde el código de instrucción GOTO y CALL, se puede direccionar una página de 2 K de la memoria. Los bits restantes del PC el 11 y 12 tienen la misión de apuntar una de las cuatro paginas de memoria y , en los modelos PIC que alcanzan este tamaño, dichos bits proceden del PCLATH, bits 3 y 4.

La pila:
La pila es una zona aislada de las memorias de instrucciones y datos. Tiene una estructura LIFO, en la que el último guardado, es el primero en salir. Tiene 8 niveles de profundidad cada uno con 13 bits. Funciona como un "buffer" circular, de manera que el valor que se obtiene al hacer el "desempilado (pop) es igual al que se obtuvo en el primero.
La instrucción CALL y las "interrupciones" originan la carga del contenido del PC en el nivel superior o "cima "de la pila. El contenido del nivel superior se saca de la pila al ejecutarse las instrucciones RETURN, RETLW y RETFIE. El contenido del registro PCLATH, no es afectado por la entrada o salida, de la información de la pila.



La figura superior, se muestra como se carga el contador de programa cuando una instrucción deposita en él ,el resultado de una operación de la ALU.
En la figura inferior, se indica la carga del PC en las instrucciones GOTO y CALL



PROYECTO

En este proyecto, retomaremos la ejecución de un programa para resolver automatismos combinacionales, utilizando una variante respecto a las dos anteriores.
Para ello, resolveremos el mismo problema, referente al "control de motores", utilizando para confeccionar el programa de texto (en lenguaje Asembler), la siguiente técnica
El inicio del programa comenzara cargando en W, los valores lógicos de las "entradas".
Con la instrucción "call" pasaremos a una zona del programa, donde estará confeccionada una lista de instrucciones o "tabla de conversión de retorno", con instrucciones de retorno (retlw k),

una a continuación de la otra, con valores que carguen al registro de trabajo W con un literal que corresponda al valor de salida de la tabla de la verdad del automatismo a resolver. Para encontrar las direcciones de retorno, que correspondan con los valores de entrada, modificaremos el valor actual del contador de programa, que corresponde a la primera instrucción de la lista (después de ejecutada la instrucción call), sumándole el valor cargado en W, o sea el valor lógico de las entradas (PCL+W). En esa nueva dirección, se encontrara con la instrucción "retlw,k" que retornara a la dirección siguiente al de llamado de la rutina (con W cargado con el literal k), donde colocaremos una instrucción de transferencia al registro de salida.
Veamos el archivo de texto del programa "CONTROL DE MOTORES6.ASM

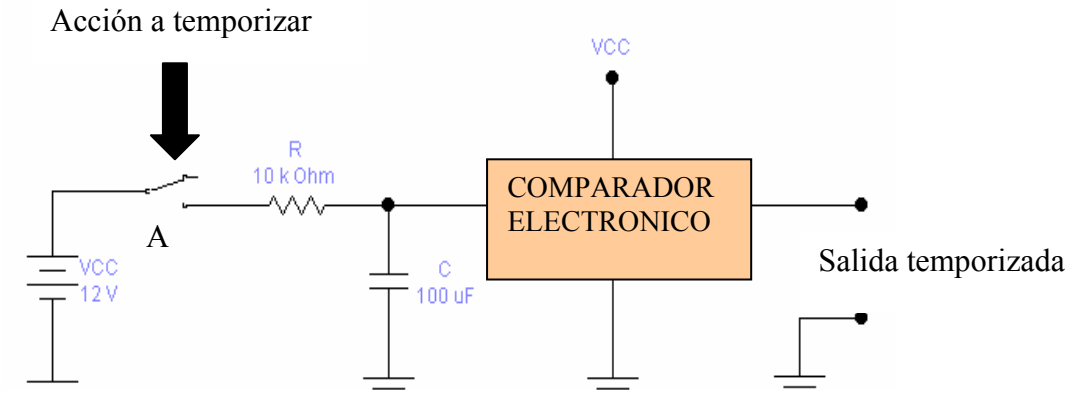
```
  ;                    MOT0R6.ASM
  ;Programa que permite controlar la cantidad de motores que se
conectan
  ;a una barra de alimentación de energía, que tiene limitaciones
  ;respecto a la máxima potencia entregada

  ;       B0=A0.A1.A2+A1.A3+A2.A3
  ;RESOLUCION POR TABLA DE CONVERSIÓN

            LIST    P=16C84
            RADIX   HEX



            ORG     0
            goto    INICIO
            ORG     5


   INICIO   bsf     0x03,5     ;selecciono el banco uno
            movlw   0xff       ;ff>w
            movwf   0x05       ;w>trisa A son entradas
            clrf    0x06       ;B son salidas
            bcf     0x03,5     ;selecciono el banco cero
            clrf    0x05
            clrf    0x06

   EXPLORA  movf    0x05,0     ;exploro las entradas y cargo en W
            andlw   0x0f       ;enmascaro con 0x0f
            call    TABLA      ;llamo a la rutina y retorno
            movwf   0x06       ;cargo salida con valor de W
            goto    EXPLORA
  ;comienza la tabla de conversión

   TABLA    addwf   0x02,1     ;modifico el contador de programa PCL
                               ;sumándole el valor de la entrada
            retlw   0x00       ;regreso de la rutina con el valor 00 H
                              ;cargado en W
            retlw   0x00
            retlw   0x00      ;idem al anterior
            retlw   0x00
            retlw   0x00
            retlw   0x00
            retlw   0x00
            retlw   0x01
```

```
retlw   0x00
retlw   0x00
retlw   0x01
retlw   0x00
retlw   0x01
retlw   0x01
retlw   0x01
retlw   0x01
end
```

CAPITULO 4: Control de tiempos, subrutinas de retardo y protección WDT

CONTROL DE TIEMPOS CON LOS MICROCONTROLADORES

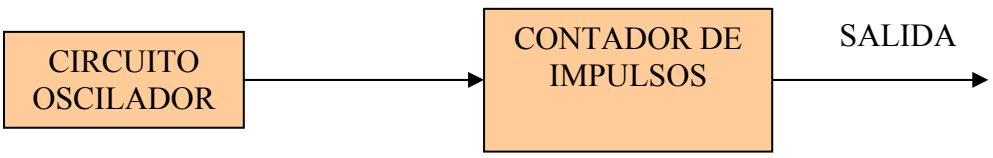
Una de las necesidades más habituales en los sistemas de control automático, es la de determinar “intervalos de tiempo”. Estos intervalos de tiempo, actúan generalmente, retrasando las variables de entrada o salida del automatismo. El dispositivo que produce esta acción, se denomina “temporizador”.
Los temporizadores se utilizan en procesos secuenciales, tales como dosificación de materiales, gobierno estrella –triangulo de contactores, gobierno de maquinas herramientas, cintas transportadoras, etc.
Los temporizadores desde el punto de vista de su construcción y tratamiento de las variables a temporizar, se pueden clasificar en “analógicos” y “digitales”. Los temporizadores analógicos, están basados en la carga o descarga de condensadores con una determinada constante de tiempo y un circuito comparador.



Cuando se cierra el contacto “A” el condensador comienza a cargarse eléctricamente con una constante de tiempo T = R x C. El comparador es un circuito electrónico que detecta un nivel fijo de tensión.
Cuando esto ocurre, la salida del comparador cambia bruscamente su valor de tensión (lógicamente pasa de 1 a 0 ó viceversa).
Para este caso, el tiempo que transcurre desde que se cierra el contacto “A” hasta que la salida cambia de estado, esta dado por la siguiente expresión:

$$T = R \times C \times \ln (VCC / VCC - VC)$$

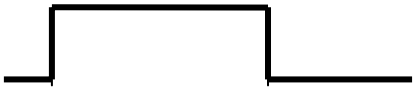
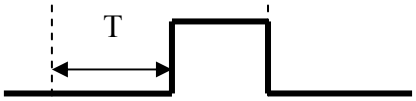
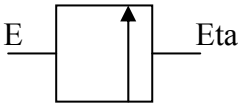
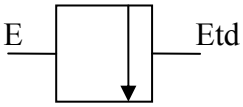
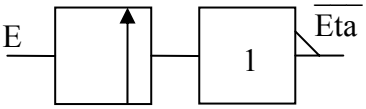
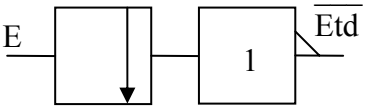
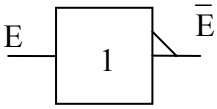
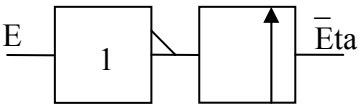
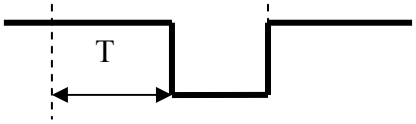
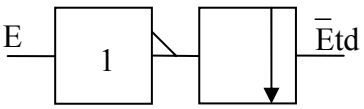
Siendo VC, la tensión de comparación.
El principio de los temporizadores digitales, esta basado en dos módulos importantes que son “el oscilador o generador de impulsos de periodo fijo” y “el contador de impulsos “cuya base constructiva, son los denominados FLIP FLOP.



La temporizacion, depende del periodo del oscilador y del numero de cuenta del contador: **T = periodo oscilador x cantidad de impulsos contados**

Diagramas de temporizaciones en los automatismos

A continuación, representaremos los seis tipos básicos de diagramas de temporizaciones, conjuntamente con el símbolo clásico y la función lógica correspondiente

FIG	DIAGRAMAS	FUNC.	SIMBOLO
A		E	
B		Eta	
C		Etd	
D		$\overline{Eta}$	
E		$\overline{Etd}$	
F		$\overline{E}$	
G		$\overline{Eta}$	
H		$\overline{Etd}$	

E: Entrada

Eta: Entrada temporizada a la activación

Etd: Entrada temporizada a la desactivación

$\overline{Eta}$: Entrada temporizada a la activación, complementada

$\overline{Etd}$: Entrada temporizada a la desactivación, complementada

$\overline{E}$: Entrada complementada.

$\overline{Eta}$: Entrada complementada, temporizada a la activación

$\overline{Etd}$: Entrada complementada, temporizada a la desactivación

Los diagramas B, C, D y E se obtienen a través de la señal directa de entrada. Los diagramas G y H a través de la señal inversa o complementada de la entrada.
Observando estos diagramas vemos que los diagramas D y H son iguales y asimismo los E y G. Por otra parte el diagrama E es el inverso del C y el diagrama D es el inverso del B. En consecuencia podemos admitir las siguientes igualdades:

$\overline{Eta} = \overline{Etd}$

$\overline{Etd} = \overline{Eta}$

$\equiv\equiv$

$Etd = Etd$

$\equiv\equiv$

$Eta = Eta$

Se puede observar que los seis diagramas se reducen a cuatro y a la vez tres de ellos se pueden expresar en función del otro; así se pueden establecer las siguientes equivalencias:

$\equiv\equiv$

$Etd = Etd$

$\overline{---}$

$Eta = \overline{Etd}$

$\equiv\equiv$

$\overline{Eta} = Eta$

Como conclusión, disponiendo de un solo tipo de modulo temporizador, podemos obtener las seis configuraciones básicas, con el agregado de funciones complementarias.
Además, combinando las temporizaciones a la activación y a la desactivación, logramos una señal temporizada, combinada con retraso , a la activación y a la desactivación, y sus derivaciones , con el agregado de complementaciones tanto en la entrada como en la salida.

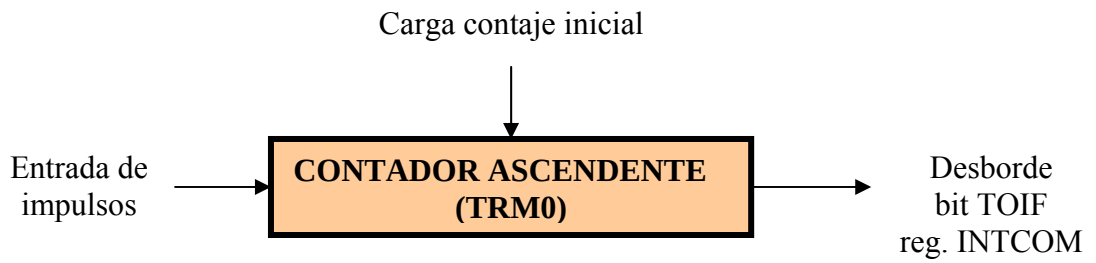
CONTROL DE TIEMPOS CON EL MICROCONTROLADOR PIC 16X84:

Como lo hemos dicho, una de las tareas más habituales en los programas de control automático de dispositivos, es la de determinar “intervalos de tiempo” a las variables de control. Para el caso de los microcontroladores, tenemos dos formas de realizarlo: Por programa (software) o utilizando módulos especiales (hardware) internos temporizables.
(timer).
Para la primer método, consiste básicamente en una subrutina de programa que contenga en su interior otras subrutinas que cargan un registro, del tipo SFR y aplicando instrucciones de salto condicional cuando se llega al tiempo establecido.
Para la segundo método, se disponen de temporizadores/ contadores de 8 bits, siendo uno solo para el PIC16X84, denominado TMR0, que puede actuar de dos maneras a saber:

a)- Como contador de impulsos aplicados a la entrada ”**RA4/TOCKI**”. Cuando el contador llega al valor FF H se desborda y con el siguiente impulso pasa a 00 H, advirtiendo esta circunstancia, con la activación a 1 del bit 2 “**TOIF**”, del registro **INTCOM**, ubicado en ambos

bancos de memoria, en la dirección 0x 0B. Este registro corresponde al de control de interrupciones.
El bit "**TOIF**", al pasar de 0 a 1 cuando se desborda el contador, se lo utiliza con una instrucción de salto condicionada. Para volverlo a cero, se lo hace con una instrucción que lo ordene (por software).

b)-Como temporizador, cargándolo con un valor inicial y luego incrementándolo con cada ciclo de instrucción con el oscilador de sincronismo del microcontrolador, con frecuencia Fosc/4. Cuando desborda o sea cuando pasa de 0xff a 0x00 avisa, colocando un "1" en el señalizador **TOIF** del registro **INTCOM**.
Veamos el esquema simplificado del contador / temporizador:



Previo al tratamiento del uso del TRM0, analizaremos el registro "OPTION" y su relación con los bloques internos involucrados con éste registro.

El registro "OPTION":

La misión principal de este registro es controlar al TMR0 y al divisor de frecuencias que comparten con el modulo temporizador denominado "perro guardián" (WDT).

RBP0#	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

Veamos la función de cada bit de este registro:
-PS2, PS1, y PS0 es el valor con el que actúa el divisor de frecuencia

PS2	PS1	PS0	División TMR0	División 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	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

-PSA: asignación del divisor de frecuencia
1 = El divisor se asigna al WDT
0 = El divisor se asigna al TMR0

-TOSE: Tipo de flanco en TOCKI
1 = Incremento de TMR0 con flanco descendente
0 = Incremento de TMR0 con flanco ascendente

-TOCS : Tipo de reloj para el TMR0
1 = Pulsos introducidos a través de TOCKI (uso como contador)
0 = Pulsos del reloj interno Fosc./4 (uso como temporizador)

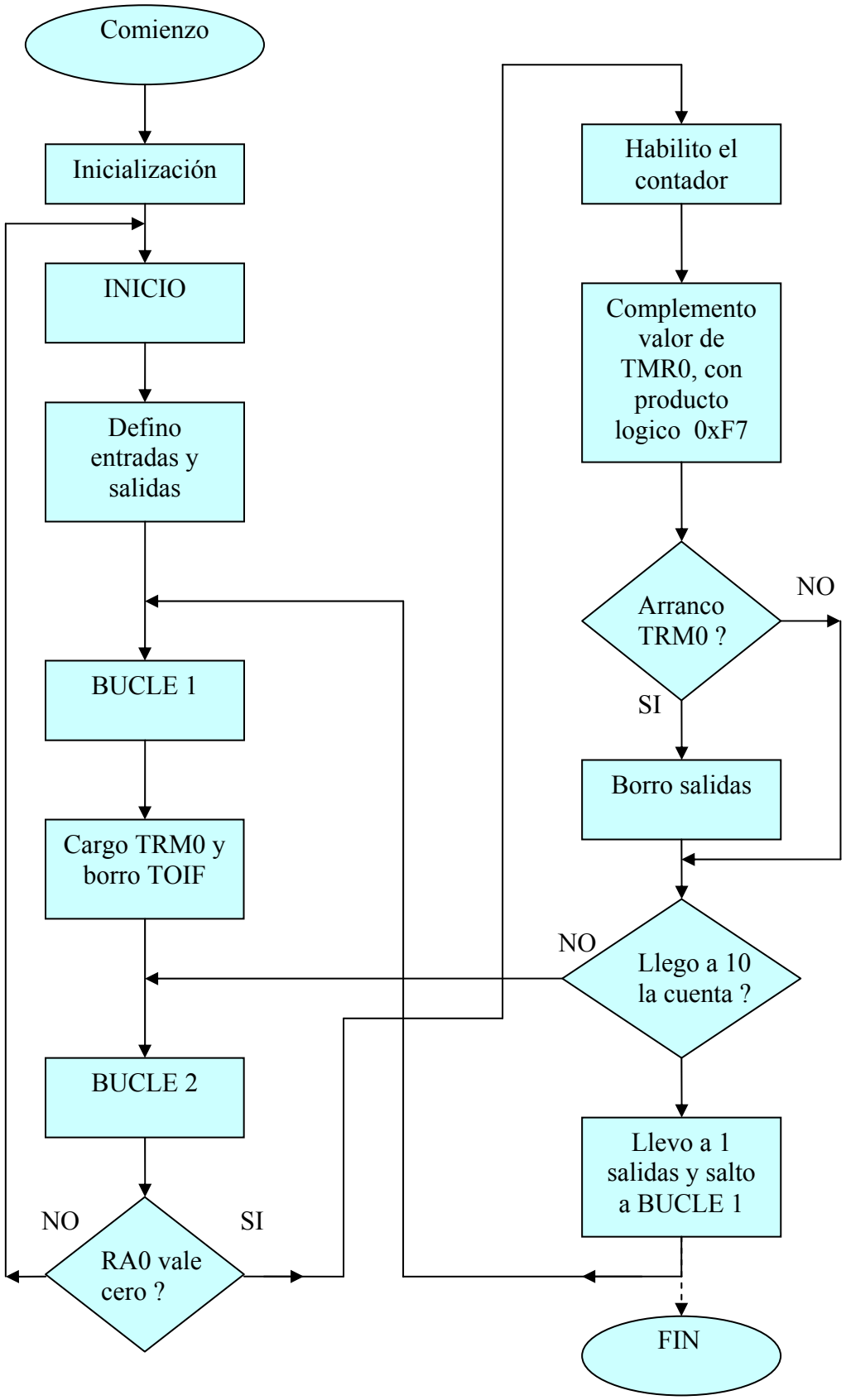
-INTEDG = Flanco activo para la interrupción externa
1 = Flanco ascendente
0 = Flanco descendente

-RBPO# = Resistencias Pull up puerta B
1 = desactivadas
0 = activadas

Uso del TMR0 como contador de impulsos:

Para utilizar el TRM0 como contador de impulsos externos provenientes del Terminal "RA4/TOCKI", debemos poner a "1" el bit "TOCS" que ocupa la dirección 5 del registro OPTION que se encuentra en la dirección 0x01 del banco uno. En la misma dirección pero en el banco cero, se encuentra el contador TRM0.
En el registro "OPTION" también podemos seleccionar el flanco activo para que actue el contador. En el bit TOSE (4) si es igual a 1, el flanco activo es el descendente. Si vale 0, el flanco activo es el ascendente.
A continuación y como ejemplo realizaremos un programa para usar al microcontrolador PIC16X84 como contador de impulsos externos hasta una cantidad de 10.Cuando llega a esta cuenta, todas las salidas RB0...RB7 se colocan a "uno"; el contador nuevamente queda en condiciones de contar nuevamente hasta 10. Cuando comienza a contar, las salidas pasan otra vez a cero. Además como parte del automatismo, se agrego la entrada RA0 que actúa como RESET (vuelve a cero la cuenta en cualquier estado) si alguna condición especial lo requiera.
Seguidamente, damos una aplicación práctica del TRM0 como contador de impulsos:

DIAGRAMA DE FLUJO CONTADOR1



A partir del diagrama de flujo, desarrollamos en archivo de texto, el programa "CONTADOR1.ASM"

```
;                   CONTADOR1.ASM
;                   =============
;el programa cuenta hasta 10 pulsos que ingresan por RA4/TOCKI
;en el flanco de subida. Cuando termina la cuenta, las salidas
;RB0...RB7,pasan al estado uno, para comenzar nuevamente a contar
; hasta 10.
;Durante la cuenta  a 10 ;es posible resetear con RA0=1 (F1)
; y volver a contar.


      LIST    P=16F84
        RADIX   HEX

        ORG     0
        goto    INICIO
        ORG     5



 INICIO clrf    0x05   ;borro r05,entradas
        clrf    0x06   ;borro r06,salidas

        bsf     0x03,5 ;selecciono el banco 1
        movlw   0xff   ; ff>w
        movwf   0x05   ; w>r05 trisA son entradas
        clrf    0x06   ; r06=0 trisB son salidas
        bcf     0x03,5 ;paso al banco cero
 BUCLE1 movlw   0xf6   ;cargo en w el literal f6
        movwf   0x01   ;cargo el contador con f6
        bcf     0x0b,2 ;borro el bit TOIF de desbordamiento
                       ;del contador (reg.INTCON)
 BUCLE2 btfsc   0x05,0 ;reviso el bit 0 de r05(RA0)(F1)salto
                       ;si vale cero
        goto    INICIO ;salto incondicional a etiqueta INICIO
        bsf     0x03,5 ;paso al banco uno
        movlw   0xe8   ;e8>w cargo a w con el literal e8
        movwf   0x01   ;w>r01 cargo con e8 r0(registro option)
        bcf     0x03,5 ;paso al banco cero
        comf    0x01,0 ;complemento valor TMR0,resultado en W
        andlw   0xf7   ;producto lógico de W con literal"f7"
        btfsc   0x03,2 ;reviso si la operación fue cero en el
                       ;indicador Z del registro de estado.si
                       ;lo fue , salto.
        clrf    0x06   ;borro las salidas
        btfss   0x0b,2 ;reviso el bit 2 (TOIF)de r0b(registro INTCOM)
                       ;salto si vale uno
        goto    BUCLE2 ;salto incondicional a etiqueta BUCLE2
        bcf     0x03,5 ;paso al banco cero (instrucc. erronea)
        movlw   0xff   ;cargo a w con el literal ff
        movwf   0x06   ;w>r06 cargo a r06(salidas) con ff
        goto    BUCLE1 ;salto incondicional a BUCLE
        end            ;fin del programa
```

Uso de TMR0 como temporizador:

Para que el TMR0 actue como temporizador, es decir se alimente de pulsos internos de frecuencia" Fosc/4 ", debemos colocar un cero (0) en el bit TOCS del registro OPTION. Si además necesitamos realizar temporizaciones largas, debemos hacer uso del divisor de frecuencias; para ello debemos colocar un "cero" en el bit PSA en el registro OPTION para que se le asigne el divisor de frecuencias que comparte con WDT. El valor de la división de frecuencias depende de los bits PS2, PS1 y PS0 según el grafico anterior.
El TMR0 se comporta como un registro de propósito especial (SFR) ubicado en la dirección 0x01 del banco 0. En igual dirección pero en el banco 1, se encuentra el registro OPTION.
El TMR0 puede ser leído y escrito en cualquier momento al estar conectado al bus de datos.
Funciona como un contador ascendente de 8 bits. Cuando funciona como temporizador,
Se puede detectar el tiempo de carga de TMR0, de dos formas a saber: La primera determinando el número de cuenta, por medio de lectura de uno de los bit del contador con una instrucción de salto condicionada. La segunda forma, conviene cargarle con el valor de los impulsos que se quiere temporizar, pero expresados como "complemento a 2". De esta manera al llegar al número de impulsos deseado, se desborda y al pasar por 00 H se activa el señalizador TOIF y /o se produce una interrupción.
El cálculo del tiempo que controla TMR0 lo calculamos con la siguiente expresión:

Temporización = 4. Tosc. . (Valor cargado en TMR0). (Rango del divisor)

Valor cargado en TMR0 = Temporización / 4 . Tosc . Rango del divisor.

La figura (pag 76) muestra el esquema de funcionamiento del TMR0. En ella, se puede observar que existe un bloque que retrasa dos ciclos el contaje para sincronizar el momento del incremento producido por la señal aplicada en TOCKI con el que se producen los impulsos internos de reloj. Cuando se escribe TMR0, se retrasan 2 ciclos su reincremento y se pone a 0 el divisor de frecuencia.

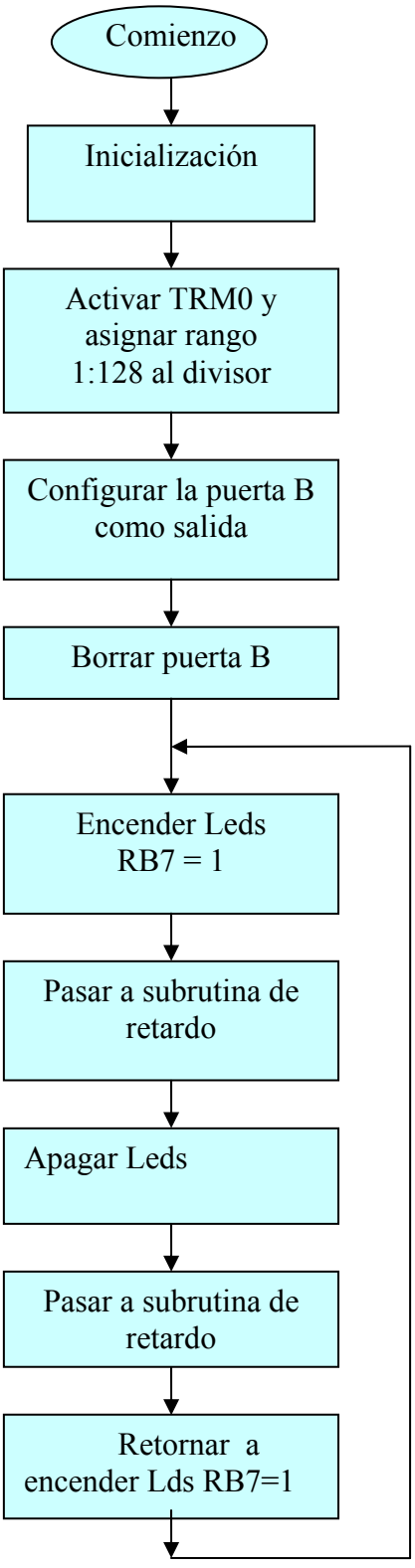
Subrutinas de retardo:

Cuando se necesita aplicar temporizaciones en automatismos, éstas suelen reiterarse durante el desarrollo del programa. Para simplificar y ahorrar espacio de la memoria de programa, conviene establecer subrutinas de retardo, de manera tal que puedan ser invocadas, en distintas partes del programa principal. Las subrutinas desvían el flujo de ejecución del programa "central", a otro punto, por medio de la instrucción "call k" que guarda la dirección actual en la pila y salta a la dirección "k", al modificar el contador de programa con esa dirección. Una vez ejecutada la subrutina se debe volver al programa principal, para ello mediante la instrucción **"return" y/o "retlw",** modifican al contador de programa con la dirección de retorno sacada de la pila. La pila del PIC16X84 tiene ocho niveles, lo que permite anidar hasta ocho subrutinas con las instrucciones mencionadas.
Como primera aplicación practica al uso de subrutinas de retardo, veremos un programa que produce el parpadeo de un diodo Leds. Tomaremos como tiempo de retardo 8,2 mS suficiente como para observarlo en el programa de simulación. En la practica deberemos usar tiempos mayores (aprox. 0,5 S) si queremos ver el parpadeo.

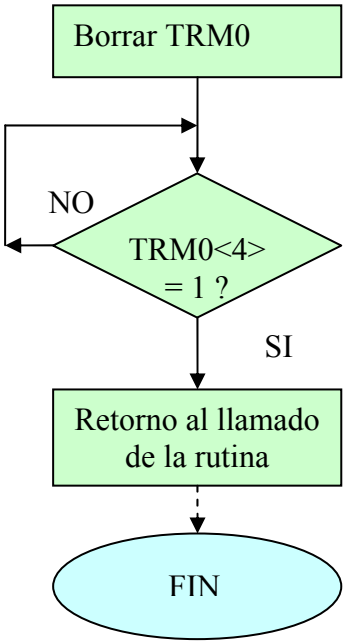
Tiempo de retardo : Si tenemos en cuenta que vamos a usar un oscilador de cristal de 1 MHZ, los 8,2 mS lo podemos obtener mediante una división de frecuencia de 128 y una cuenta en TMR0 de 16 pulsos.
Temporización = 4 . Tosc . valor cargado en TMR0 . Rango del divisor
Temporización = 4 . 1us . 16 . 128 = 8,2 ms

Diagrama de flujo del programa PARPADEO

Programa principal



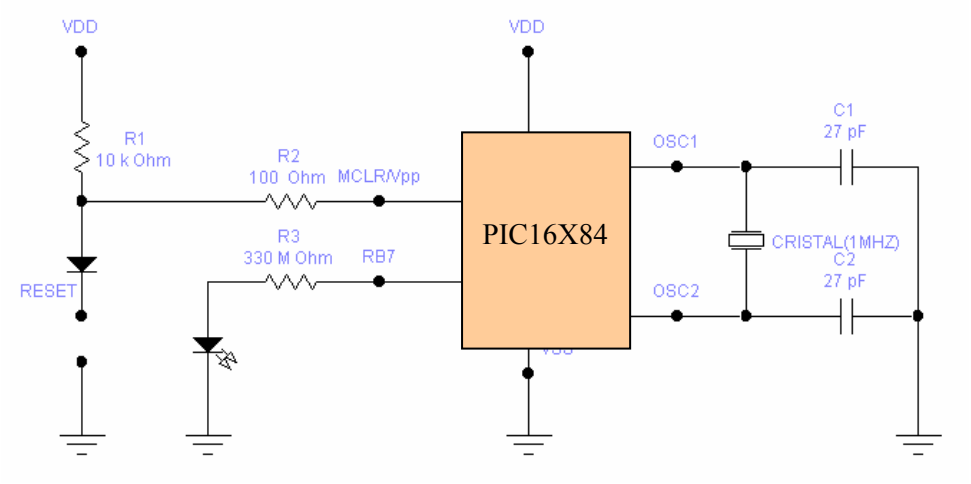
Subrutina de retorno



A continuación desarrollamos el programa con definición de etiquetas que llamaremos PARPADEO.ASM

```
;                   PARPADEO.ASM
; Programa que ilustra c¢mo realizar una temporización
; sin emplear interrupciones. Se realiza una temporización de 8,2 ms
; que se emplea para hacer parpadear un diodo led en RB7
; Reloj del PIC: 1Mhz


      LIST        P=16F84
      RADIX       HEX


; -----------------------------------------------------------------

        PUERTAB   EQU   0x06
        TMR0_OPT  EQU   0x01            ; TIMER0 en banco 0 y
                                        ; OPTION en banco 1
        ESTADO    EQU   0x03

; -----------------------------------------------------------------


       ORG        0           ; Inicio del programa en
                              ; dirección 0

; -----------------------------------------------------------------
      bsf         ESTADO,5    ; Banco 1
      movlw       b'11010110' ; Valor a cargar en
                              ; OPTION
      movwf       TMR0_OPT
      movlw       0x00
      movwf       PUERTAB     ; La Puerta B salida
      bcf         ESTADO,5    ; Banco 0
      clrf        PUERTAB     ; Las líneas de salida
                              ; de PB a 0
parpa bsf         PUERTAB,7   ; Enciende el led RB7 = 1
      call        retar       ; Llamada a subrutina de
                              ; RETARDO
      bcf         PUERTAB,7   ; Apaga el led, RB7 = 0
      call        retar
      goto        parpa

;--------------subrutina de retardo ------------------

retar clrf        TMR0_OPT    ; TMR0 = 0  y empieza su
                              ;  incremento
explora btfss     TMR0_OPT,4   ; TMR0<4> = 1?
      goto        explora      ; No ha llegado TMR0 a
                               ; 16d
      return                   ; Ha llegado TMR0 al
                               ; valor 16d  y retorna
                              ; al programa principal
      end
```

Conexionado del microcontrolador para el programa PARPADEO.ASM



En el programa simulador la temporización de 8,2 mS se puede observar con claridad, dado que la simulación, que no se hace en tiempo real, siempre es más lenta. En la práctica un parpadeo de ese valor prácticamente no se lo puede observar. Si usamos la máxima cuenta de TMR0 que es 256 con captación de su desbordamiento a través del bit TOIF del registro INTECON, y el divisor de frecuencia, e 256, solo lograríamos, para un oscilador de 1MHZ, un tiempo de:

T = 4 . 1 uS . 256 . 256 = 262 ms

Como podemos ver apenas llegamos a 1/4 seg. de temporización.
Para lograr temporizaciones de mayor tiempo, es necesario utilizar la temporización tantas veces que sea necesaria, hasta llegar al tiempo establecido. Esto lo podemos hacer con un contador auxiliar que registre el número de temporizaciones del TRM0, por medio de una subrutina dentro de la subrutina de temporización principal.
Para el caso nuestro, si temporizamos 8,2 ms 122 veces, T = 8,2 . 122 = 1 seg.
Vemos mas abajo, un trozo de programa que corresponde a una subrutina de temporización de 1segundo

```
;.....................................................................
retardo movlw           d'122'          ; 122 -> W
        movwf           CONTA           ;CONTA => 0x0D es el
                                         ;contador auxiliar
bucle   call            explora
        clrf            TMR0_OPT        ; TMR0 = 0 y empieza su
                                        ; su incremento
        decf            CONTA,0         ; CONTA - 1 -> W
        movwf           CONTA           ; Se actualiza FZ
        btfss           ESTADO,2        ; ¨FZ = 1?
        goto            bucle           ; Otra vez al bucle de
                                        ; exploracion
        return                          ; Se ha explorado 122 veces

explora btfss           TMR0_OPT,4      ; TMR0<4> = 1?
        goto            explora          ; No ha llegado TMR0 a 16d
      return                            ; Ha llegado TMR0 al
                                        ; valor 16d  y retorna
                                         ; al programa principal
      END
```

Modificaremos el programa “PARPADEO.ASM”, incorporándole la subrutina de 1segundo y le llamaremos “PARPADEO1.ASM”

```
;                   PARPADEO1.ASM
; Programa que ilustra c¢mo realizar una temporización
; sin emplear interrupciones. Se realiza una temporización de 1Seg.
; que se emplea para hacer parpadear un diodo led en RB7
; Reloj del PIC: 1Mhz


      LIST        P=16C84
      RADIX       HEX
; ------------------------------------------------------------------

        PUERTAB   EQU   0x06
        TMR0_OPT  EQU   0x01            ; TIMER0 en banco 0 y
                                        ; OPTION en banco 1
        CONTA     EQU   0X0D
        ESTADO    EQU   0x03
; ------------------------------------------------------------------


       ORG        0           ; Inicio del programa en
                              ; dirección 0
; ------------------------------------------------------------------

      bsf         ESTADO,5    ; Banco 1
      movlw       b'11010110' ; Valor a cargar en
                              ; OPTION
      movwf       TMR0_OPT
      movlw       0x00
      movwf       PUERTAB     ; La Puerta B salida
      bcf         ESTADO,5    ; Banco 0
      clrf        PUERTAB     ; Las líneas de salida
                              ; de PB a 0
parpa bsf         PUERTAB,7   ; Enciende el led RB7 = 1
      call        retardo           ; Llamada a subrutina de
                              ; RETARDO
      bcf         PUERTAB,7   ; Apaga el led, RB7 = 0
      call        retardo
      goto        parpa
;.................rutina de retardo.................
retardo movlw           d'122'          ; 122 -> W
        movwf           CONTA           ;CONTA => 0x0D es el
                                         ;contador auxiliar
bucle   call            explora
        clrf            TMR0_OPT        ; TMR0 = 0 y empieza su
                                        ; su incremento
        decf            CONTA,0         ; CONTA - 1 -> W
        movwf           CONTA           ; Se actualiza FZ
        btfss           ESTADO,2        ; ¨FZ = 1?
        goto            bucle           ; Otra vez al bucle de
                                        ; exploración
        return                          ; Se ha explorado 122 veces
;...............subrutina de retardo.....................
explora btfss           TMR0_OPT,4      ; TMR0<4> = 1?
        goto           explora          ; No ha llegado TMR0 a 16d
      return                            ; Ha llegado TMR0 al
                                        ; valor 16d  y retorna
                                         ; al programa principal

      END
```

A continuación, resolveremos el circuito combinacional “MOTORES”, con el agregado de una salida parpadeante, cuando se da la condición de exceso de potencia, en la barra de alimentación, de acuerdo al problema planteado en su inicio. Aprovecharemos además este programa para introducir “ayudas” que nos brinda el programa ensamblador, como las siguientes:

-INCLUDE <P16F84.INC> : El programa ensamblador nos suministra una librería que nos permite reemplazar la dirección hexadecimal y bits de los registros especiales por nombres nemotécnicos. Por ejemplo: bsf STATUS, RP0 ≡ bsf 0x 03, 5

-INCLUDE <RETARDO.ASM> : Este comando permite al programa ensamblador (MPASM) utilizar el programa editado RETARDO.ASM como subrutina en el programa nuevo que se va a editar.

-# define BORRAR clrf : Se le da definición nemotécnica a la instrucción “clrf” como “BORRAR”

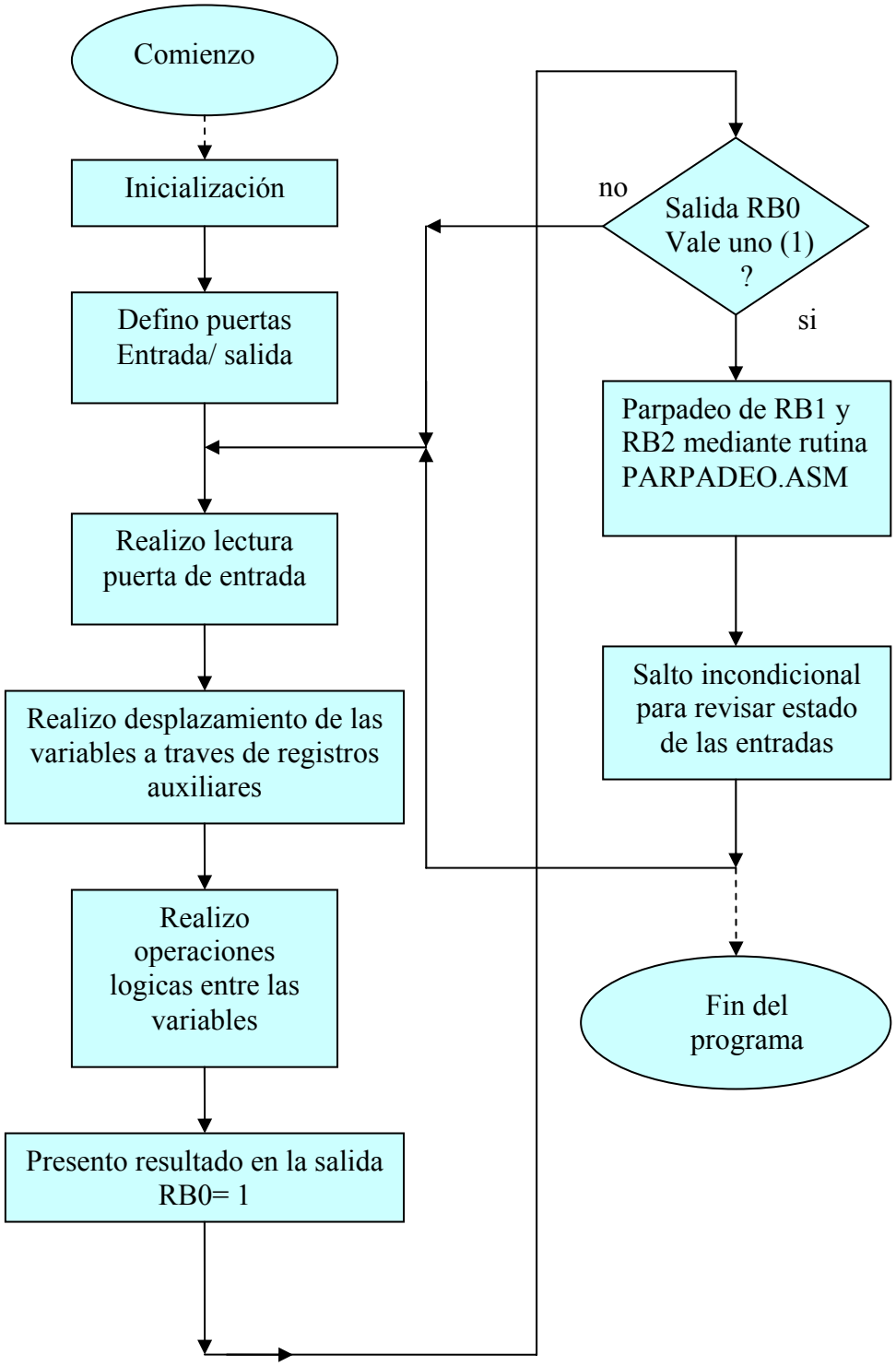
-# define BORRAR ENTRADA clrf 0x05 : Se da como definición nemotécnica “BORRAR ENTRADA” a la instrucción clrf 0x05

-A0 equ 0x 0C : Se define al registro 0x0C con el nombre “A0”

-La proxima sentencia sirve para denominar a un grupo de registros en forma consecutiva con nombres nemotécnicos:

```
Cblock    0x0c          ;inicio, se definen registros de propósito gral
          A0            ;=0x0C
          A1            ;=0x0D
          A2            ;=0x0E
          A3            ;=0x0F
          A2.A3         ;=0x10
          A1.A3         ;=0x11
  endc                  ;fin del bloque de registros
```

Diagrama de flujo de "MOTORES3.ASM"



PROGRAMA : MOTORES3.ASM

```
;                              MOTORES3.ASM
;Resolución de un automatismo combinacional
;Programa que controlar la cantidad de motores que se conectan
```

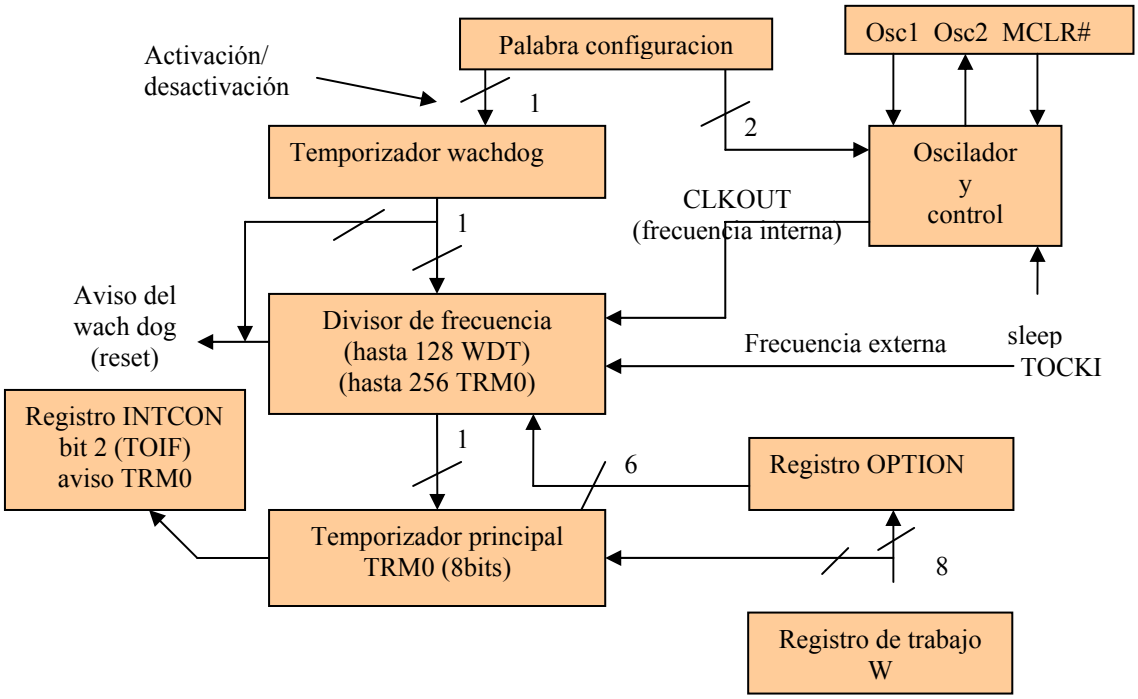
```
      ;a una barra de alimentación eléctrica, que tiene limitaciones
     ;respecto a la m xima potencia eléctrica entregada, con parpadeo
     ;en RB3 y RB4
  ;         RB0=RA0.RA1.RA2 + RA1.RA3 + RA2.RA3
  ;-----------------------------------------------------------------
          LIST    P=16F84     ;asigno el microc. Para el ensamblador
          RADIX   HEX         ;se editara con el sistema hexadecimal
       INCLUDE <P16F84.INC> ;incluyo libreria del PIC16F84
       #define BORRAR        clrf ;se define la instruc.con BORRAR
       #define BORRAR_ENTRADA  clrf 0x05  ;se define la instruc.y reg.
       cblock  0x0c     ;inicio,se definen registros de propósito gral
                    A0           ;=0x0C
                    A1           ;=0x0D
                    A2           ;=0x0E
                    A3           ;=0x0F
                    A2.A3        ;=0x10
                    A1.A3        ;=0x11
            endc                 ;fin del bloque de registros
;-----------------------------------------------------------------
         ORG     0            ;se indica la ubicación de
         goto    INICIO       ;1a instrucción en la direcc.0x00
         ORG     5            ;se salta el vector reset para ubicar la
                              ;próxima instrucción
 INICIO   BORRAR_ENTRADA      ;llevo a cero r05 (entradas)
          BORRAR  PORTB       ;llevo a cero r06 (salidas)
          bsf     STATUS,RP0  ;selecciono el banco uno
          movlw   0xff        ;ff>w
          movwf   TRISA       ;w>trisa A son entradas
          clrf    TRISB       ;B son salidas
          bcf     STATUS,RP0  ;selecciono el banco cero
 BUCLE1   movf    PORTA,0     ;entradas A>w
          movwf   A0          ;w>0C direcc. memoria datos.Entrada "Ao"
          movwf   A1          ;w>0D     "         "     "
          rrf     A1,1    ;desplazo A1 a columna A0 y lo cargo en 0x0D
          rrf     A1,0     ;desplazo A2 a la columnna A0 resultado>w
          movwf   A2          ;w>0E direcc.mem datos Entrada A2
          rrf     A2,0        ;desplazo A3 a la columna Ao resultado>w
          movwf   A3          ;w>0F direcc.mem datos Entrada A3
          andwf   A2,0        ;A2.A3>w
          movwf   A2.A3       ;w>10 direcc.mem datos producto "A2.A3"
          movf    A3,0        ;0F>w
          andwf   A1,0        ;A1.A3>w
          movwf   A1.A3       ;w>11 direcc.mem datos producto "A1.A3"
          movf    A0,0        ;0C>w
          andwf   A1,0        ;Ao.A1>w
          andwf   A2,0        ;Ao.A1.A2>w
          iorwf   A1.A3,0     ;Ao.A1.A2+A1.A3>w
          iorwf   A2.A3,0     ;Ao.A1.A2+A1.A3+A2.A3>w
          andlw   0x01        ;10 producto lógico con w resultado wo
          movwf   PORTB       ;w>06 puerta B salida
          btfss   PORTB,0     ;reviso salida RB0 y salto si vale uno
          goto    BUCLE1      ;si vale 1,activo RB4 y desactivo  RB3
          bsf     PORTB,4     ;activo RB4(1)
          bcf     PORTB,3     ;desactivo RB3(0)
          call    RETARDO     ;llamo subrutina archivo RETARDO:ASM
          bcf     PORTB,4     ;desactivo RB4 (0)
          bsf     PORTB,3     ;activo RB3 (1)
          call    RETARDO     ;llamo subrutina  archivo RETARDO.ASM
          goto    BUCLE1      ;reviso nuevamente las entradas
 INCLUDE <RETARDO.ASM>        ;incluye subrutina de retardo,ubicada
```

```
                              ;en el archivo RETARDO.ASM
        end                 ;fin del programa
```

El perro guardián (WDT):

El WDT, es un temporizador denominado "perro guardián." Se trata de un contador interno que origina un "Reset" cuando se desborda. Su control de tiempos es independiente del TMR0 y esta basado en una simple red RC. Su actuación puede ser opcional y puede bloquearse colocando un cero en el bit WDTE de l a palabra de Configuración. Cuando esta activado el WDT , tiene como misión, evitar que el programa quede "colgado" y para ello cada cierto tiempo (18 ms o mas si usa el divisor de frecuencia), comprueba si el programa se esta ejecutando normalmente. Caso contrario si el programa quedo detenido en un "bucle infinito" a la espera de un acontecimiento que no se produce, el WDT actua produciendo un Reset que reinicializa todo el sistema.
Para evitar que WDT desborde y actúe cuando el programa se desarrolla normalmente, es necesario "refrescarlo" que consiste en ponerlo a cero, mediante las instrucciones **"clrwdt" y "sleep".** Para este caso, el programador deberá analizar para colocar estas instrucciones en sitios estratégicos por lo que pase el flujo de control, antes que pase el tiempo asignado al WDT.La instrucción **"clrwdt"** borra al WDT y reinicia su cuenta. La instrucción **"sleep"** además de borrar WDT, detiene al sistema y lo conduce a un estado de "reposo" o de "bajo consumo". Para que el microcontrolador pase a este último estado sin que se desactive por la actuación de WDT, es necesario desactivar este último bloque.
En el registro de ESTADO existe un bit denominado TO# que pasa a valer cero después del desbordamiento de WDT.

Esquema simplificado de de la seccion del contador/temporizador TRM0 y WDT



CAPITULO 5: Interrupciones, escritura datos en EEPROM, reinicializacion. Estado de reposo

INTERRUPCIONES EN LOS MICROCONTROLADORES

Las interrupciones son desviaciones del desarrollo del programa principal provocadas asincrónicamente por diversos sucesos que no se hallan bajo la supervisión de las

instrucciones. Por ejemplo la instrucción “call” es una desviación sincrónica del programa principal, llamando a una subrutina. La interrupción, cuando se produce, pasa a ejecutar una subrutina, pero el acontecimiento se produce imprevistamente. En ambos casos, se guarda la dirección actual en la pila y se carga el contador de programa PC con una dirección, que en caso de la instrucción “call” ya viene en la propia instrucción. Para el caso de una interrupción, el contador de programa se carga con una dirección reservada de la memoria de instrucciones denominada “vector de interrupción”. Para los PIC16X8X, el vector interrupción se halla situado en la dirección 0004 H, donde comienza la rutina de servicio a la interrupción (RSI). En general en dicha dirección se coloca una instrucción de salto incondicional (goto), que traslada el flujo de control a la zona de la memoria de instrucciones, destinada a contener la rutina de atención a la interrupción.
La **RSI,** en general suele comenzar guardando en la memoria de datos algunos registros específicos que empleara y alterara durante el desarrollo de la rutina. Antes del retorno, al programa principal, es necesario recuperar estos valores para restaurar el estado previo a la interrupción. El retorno de una rutina de interrupción se produce con la instrucción **“rectifie”.**

Causas de interrupción:

Los PIX16X8X tienen cuatro fuentes posibles de interrupción:

1)-Activación de la entrada RB0 /INT de la puerta A

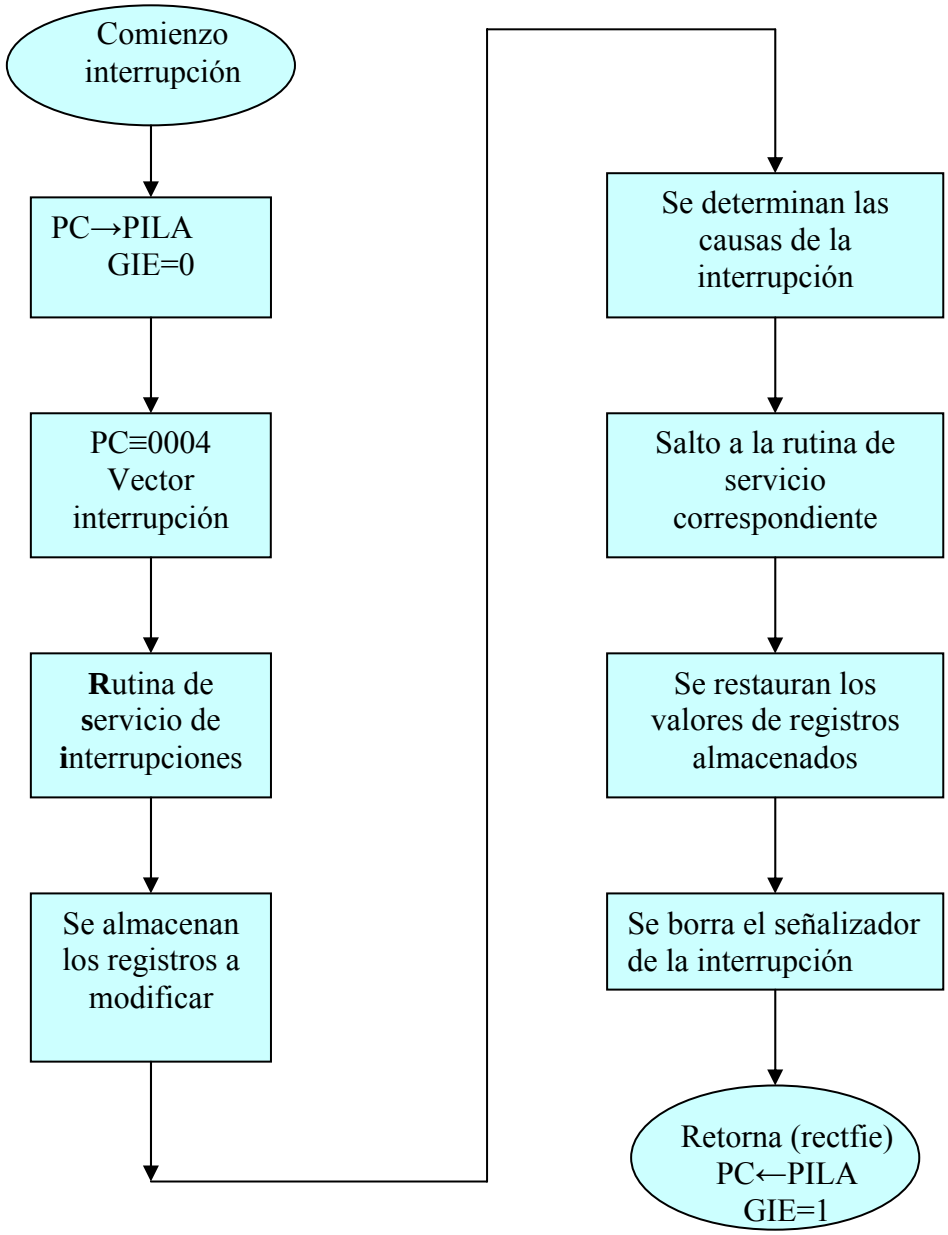
2)-Desbordamiento del temporizador TMR0.

3)-Cambio de estado en una de las cuatro entradas de mas peso (RB7...RB4) de la puerta B.

4)-Finalización de la escritura en la EEPROM de datos.

Para que se produzca la rutina de interrupción, el bit GIE (Global interrup Enable) del registro INTCON, debe valer uno (1); caso contrario, prohíbe todas las interrupciones.
Cuando se produce una interrupción, GIE pasa a valer cero (0), con el objeto de no atender nuevas interrupciones hasta que termine la que ha comenzado. GIE, pasa a valer uno automáticamente cuando se produce el retorno de la interrupción.
Cada fuente de interrupción, tiene un bit de permiso de interrupción como así también un bit de señalización. Estos bits de permiso y señalización, se encuentran en el registro INTCON, salvo el señalizador de fin de escritura de la EEPROM, denominado EEIF, que se encuentra en el bit 4 del registro EECON1.
Los señalizadores deben ponerse a cero por programa, antes del retorno de la interrupción y además son operativos aunque la interrupción este prohibida, por el bit de permiso correspondiente.

Organigrama de las operaciones principales de una interrupción



Veamos el registro de control de interrupciones INTCON:

GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

GIE : Permiso global de interrupciones

GIE= 1 Permite todas las interrupciones si los bits de permisos individuales lo permiten.
GIE =0 Prohíbe todas las interrupciones.

EEIE: Permiso de la interrupción por fin de la escritura en la EEPROM

EEIE=1 Permite la interrupción.
EEIE=0 Prohíbe la interrupción.

TOIE: Permiso de interrupción por desbordamiento de TMR0.

TOIE=1 Permite la interrupción.
TOIE=0 Prohíbe la interrupción.

INTE: Permiso de interrupción por activación de la entrada RB0 / INT.

INTE=1 Permite la interrupción.
INTE=0 Prohíbe la interrupción.

RBIE: Permiso de interrupción por cambio de estado en RB7...RB4.

RBIE=1 Permite la interrupción.
RBIE=0 Prohíbe la interrupción.

TOIF: Señalizador de desbordamiento del TMR0

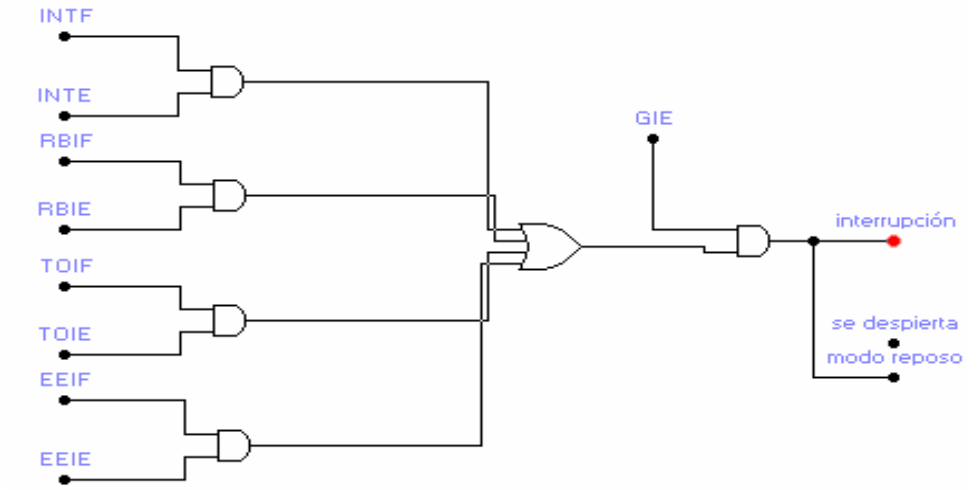
TOIF=1 Indica que sé a producido el desbordamiento
TOIF=0 Indica que no sé a producido el desbordamiento

INTF: Señalizador de la activación de la entrada RB0 / INT

INTF=1 Indica la activación de RBO / INT
INTF=0 Indica que no se a producido la activación de RB0 / INT

RBIF: Señalizador de cambio de estado en las entradas RB7...RB4

RBIF=1 Indica que se produjo el cambio de una de las cuatro entradas.
RBIF=0 Indica que no se a producido ningún cambio
Veamos la lógica de control para generar una interrupción:



Siempre que se produzca una interrupción por cualquier causa, GIE=0 y el PC se carga con el valor 0004 H, que es, el vector interrupción. Para conocer que causa provocó la interrupción, es necesario explorar los señalizadores. Éstos, posteriormente, deben ponerse a cero por programa, dado que son siempre operativos.
Existe un término, conocido como “latencia de la interrupción”, que se define como el tiempo que transcurre, desde que apareció la interrupción (el señalizador se pone a uno) hasta el momento en que la instrucción ubicada en la dirección 0004 H, comienza a ejecutarse. Para interrupciones sincrónicas, este tiempo es de tres ciclos de instrucción. Para interrupciones no sincrónicas como las externas, éste tiempo puede ser entre 3 y 3,75 ciclos de instrucción.

Describiremos a continuación, una porción de programa, que nos permite explorar todos los señalizadores de interrupciones y saltar a "la rutina de servicio de interrupciones" **(RSI**) correspondiente, al que se encuentra activado.

```
----------------------------------------------------------------------------------------------------
   int      btfss  intecon,intf ;explora el bit intf de intecon
                                ;y salta si vale uno
            goto   puertab      ;salto incondicional para explorar
                                ;activación interrupción RB7..RB4
            goto   inter1       ;RSI de la interrupción externa
                                ;por RB0/INT.
   puertab  btfss  intecon,rbif ;explora el bit rbif de intecon
                                ;y salta si vale uno
            goto   tmr0         ;salto incondicional para explorar
                                ; Activación interrupción de TMR0
            goto   rb7_rb4      ;RSI de la interrupción por activa
                                ;ción de las entradas RB7..RB4
   tmr0     btfss  intecon,toif ;explora el bit toif de intecon
                                ;y salta si vale uno
            goto   eeprom       ;salto incondicional para explorar
                                ;fin de escritura EEPROM
            goto   tempo        ;RSI de la interrupción por desbor
                                ;damiento de TMR0
   eeprom   btfss  eecon1,eeif  ;explora bit eeif de eecon1 y
                                ;salta si vale uno
            goto   int          ;salto incondicional  a int para
                                ;volver a explorar los señalizadores
            goto   escritura    ;RSI de la interrupción por fin de
                                ;escritura de la EEPROM
* continua el programa
```

Interrupción externa por RB0 /INT

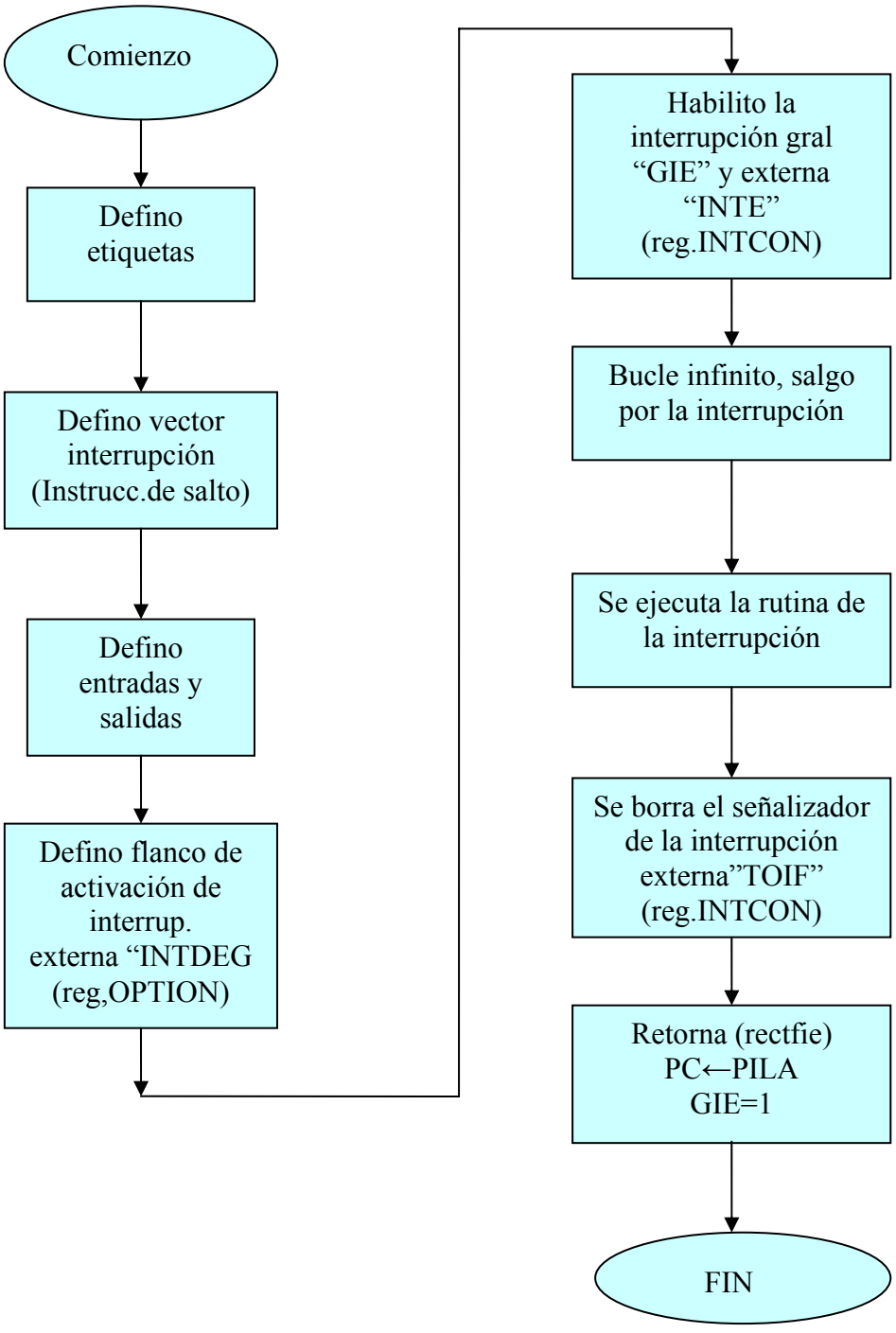
Este tipo de interrupción es muy importante cuando se debe atender acontecimientos externos en tiempo real. Cuando se activa la entrada RB0/INT, de forma automática, el bit INTF se hace igual a uno (1) y si el bit de permiso INTE es igual a uno (1), la interrupción se autoriza, desarrollándose los procesos que corresponden a la interrupción ya comentados.

Mediante el bit 6, llamado INTDEG, del registro OPTION, se puede seleccionar cual será el flanco activo en la entrada RB0/INT. Si deseamos que la activación se produzca con el flanco ascendente, entonces INTDEG= 1. Si deseamos que sea el descendente, entonces INTDEG =0. El tiempo de demora entre la activación del flanco hasta que se produce la interrupción, es de 3 o 4 ciclos reloj.

Como conclusión, con INTE=1 habilitamos la interrupción externa y, si se produce un flanco activo en la entrada RB0/INT, el señalizador INTF se pone a 1 automáticamente, solicitando la interrupción, que es aceptada sí GIE =1. Antes de regresar al programa principal es necesario borrar el bit INTF, puesto que en caso contrario al ejecutar la instrucción de retorno RETFIE, se volvería a desarrollar el mismo proceso de interrupción.

A modo didáctico, realizaremos un programa sencillo de aplicación de interrupción externa, consistente en tres salidas lógicas de un automatismo, que copian el valor lógico de tres entradas, solamente en el momento que una interrupción externa por RB0/INT lo autorice.

Diagrama de flujo

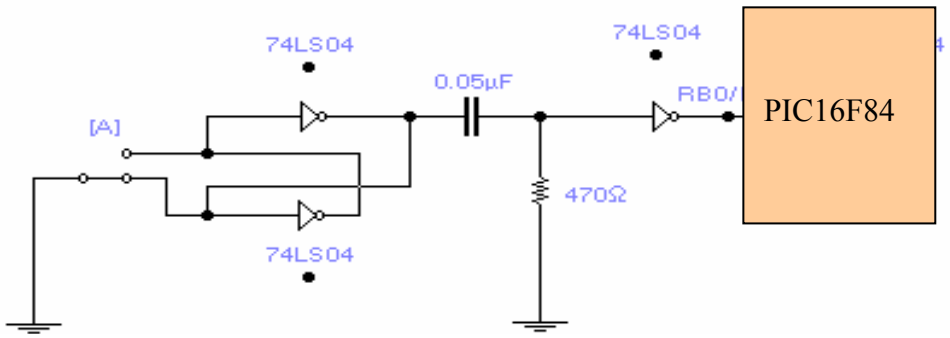


PROGRAMA

```
;EXTERNO.ASM
;programa por el cual las salidas RB1,RB2 y RB3 copian el valor
;lógico de las entradas RA1, RA2 y RA3, solamente en el momento
;que se autoriza a través de la entrada por interrupción externa
;RB0/INT (flanco descendente).

                LIST        p= 16F84
```

```
        RADIX       HEX

        ORG         0
        goto        INICIO ;inicio del programa
        ORG         4
        goto        INT    ;salto a la rutina interrup.


INICIO  bsf         0x03,5 ;selecciono banco 1
        movlw       0xff   ;cargo en W literal ff H
        movwf       0x05   ;RA0...RA4 son entradas
        movlw       0x01   ;cargo en W literal 01 H
        movwf       0x06   ;RB0 entrada,RB1..RB7 salidas
        bcf         0x01,6 ;flanco descendente int. externa
        movlw       0x90   ;cargo W con literal 90 H
        movwf       0x0b   ;cargo registro INTCON,habilito
                           ;interrupción GIE y externa INTE
                           ;desactivo resto interrupciones
                           ;y borro se¤alizador externo INTF
        bcf         0x03,5 ;paso al banco cero
        clrf        0x05   ;borro entradas
        clrf        0x06   ; borro salidas


BUCLE   goto        BUCLE  ;bucle infinito salgo por
                           ; interrupción
INT     movf        0x05,0 ;cargo entradas en W
        movwf       0x06   ;cargo en salidas contenido de W
        bcf         0x0b,1 ;borro se¤alizador interrup.externa
        retfie             ;retorno de la interrupcion
        end                ;fin del programa
```

En la figura mostramos un circuito práctico para generar pulsos de duración limitada (en el ejemplo 23,5 us) con un circuito antirrebote, para el pulsador, utilizando dos inversores del circuito integrado 74LS04 realimentados. La red RC y el último inversor, controlan la duración del impulso

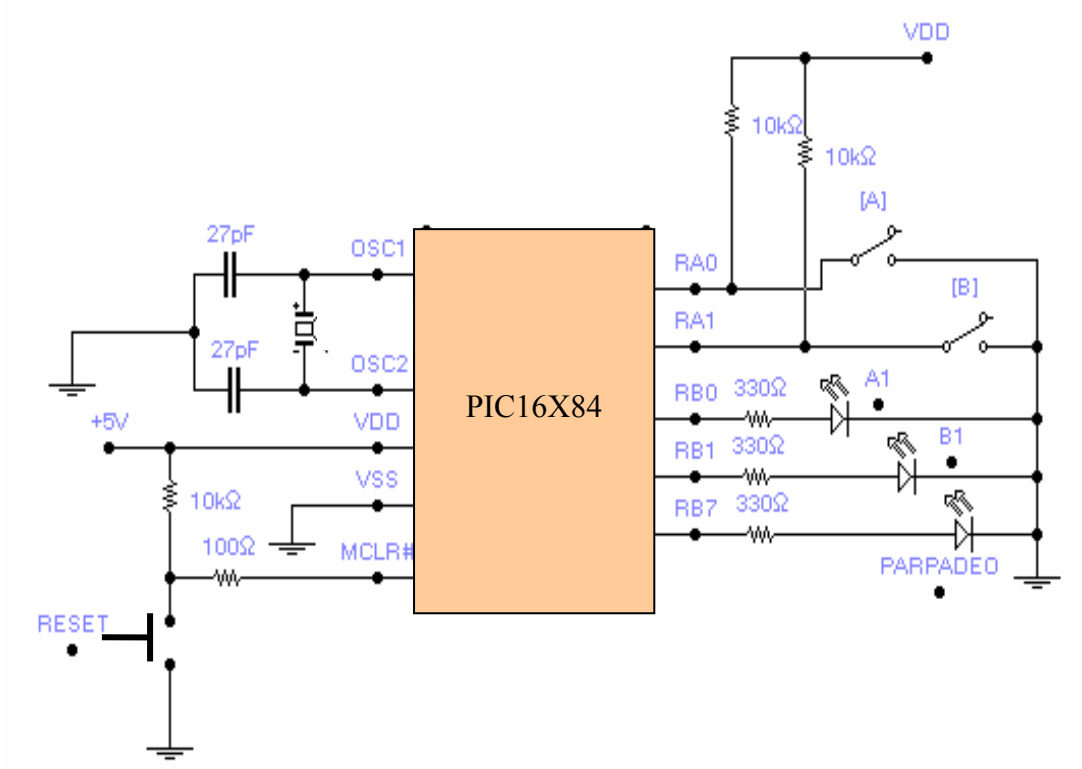


<u>Interrupción por desbordamiento del TMR0</u>:

una interrupción.
Si no se recarga el TMR0 cuando se desborda, sigue contando desde 00 H a FF H.
La lectura y escritura de este registro, se puede realizar en cualquier momento, pero cada vez que se escribe se pierden dos ciclos de reloj de sincronización.
Cuando se carga inicialmente TMRO con un valor determinado N, el desbordamiento se produce con 256 – N impulsos, por lo tanto el tiempo que tarda para llegar a esta situación vale: T = 4 . Tosc. .(256 – N) . Rango del divisor de frecuencia.

Como ejemplo practico de interrupciones por TMRO, tomaremos un ejercicio del libro Microntroladores PIC (Angulo, Usategui) con algunas modificaciones.
El programa consiste en utilizar un PIC16X84 , con un oscilador interno de frecuencia 4 MHZ. Conectado a las entradas RA0 y RA1 tenemos dos contactos A y B, los cuales deben ser explorados continuamente y reflejar su estado (abierto o cerrado) sobre dos diodos Leds A1 y B1, conectados a las líneas RB0 y RB1 de la puerta B.
Además tendremos otro diodo Leds, conectado a la salida RB7, que comenzara a parpadear cuando se activen los otros diodos, con un lapso de tiempo de 1 segundo entre los estados de encendido y apagado.

Esquema eléctrico



Para controlar el retardo de 1 segundo para realizar el parpadeo del led ,conectado a RB7, realizaremos una interrupción por desbordamiento de TMR0.
Cargaremos a TMRO con el valor decimal 12, con lo que el desbordamiento se producirá (si tomamos el rango del divisor de frecuencia en 256) , en un tiempo:
T= 4 . 250 ns .(256-12) . 256 = 62,4ms
Como no alcanzamos el retardo de 1 segundo, emplearemos un contador auxiliar CONTA (0x0c) que al cargarse con el valor 16 y decrementarse una unidad cada 62,4 ms, cuando llegue a cero , se conseguirá, aproximadamente el tiempo buscado
T = 62,4 ms . 16 = 1s
El programa utiliza las instrucciones de salto condicionado **btfss f,d y btfsc f,d** para inspeccionar las entradas y copiarlas en las salidas como así también verificar que el parpadeo solamente este presente , cuando se activen las salidas (se enciendan los diodos A1 y B1.
A los efectos que se pueda apreciar en la simulación el parpadeo del diodo, conviene utilizar el divisor de frecuencias por 2, cargar el TMRO con 0xf0 y el contador auxiliar con 0x03.
Desarrollamos a continuación el programa **EXTERNO2.ASM** :

```
;EXTERNO2.ASM Refleja el estado de dos interruptores,situados en RA0 y
;RA1 en RB0 y RB1 mientras hace parpadear un diodo en la línea RB7 sí
;RB0 ò RB1 valen uno (1)
;-----------------------------------------------------------------------
        LIST    P=16F84
```

```
        RADIX   HEX

;-----------------------------------------------------------------
        W       EQU     0
        F       EQU     1
        TMR_OPT EQU     0x01    ; TMRO en banco 0 OPTION en banco 1
        ESTADO  EQU     0x03
        PUERTAA EQU     0x05    ; PA en banco 0 TRISA en banco1
        PUERTAB EQU     0x06    ; PB en banco 0 TRISB en banco1
        INTCON  EQU     0x0B    ;
        CONTA   EQU     0x10    ; Contador auxiliar
;-----------------------------------------------------------------
        ORG     0               ; Vector de Reset
        goto    inicio

        ORG     4               ; Vector de Interrupción
        goto    inter           ; Salta a comienzo de rutina de
                                ; Interrupción
        ORG     5

inicio  bsf     ESTADO,5        ; Selección del banco 1
        clrf    PUERTAB         ; Configura PUERTA B como salida

        movlw   b'00000011'     ; Configura RA0, RA1 como entradas
        movwf   PUERTAA

        movlw   b'00000000'     ;asigno división de frecuencia
        movwf   TMR_OPT

        bcf     ESTADO,5        ; Banco 0
        clrf    PUERTAB         ; borro las salidas
        movlw   b'10100000'     ; Se permite interrupción del
        movwf   INTCON          ; TMR0 y la global (GIE)

        movlw   0x03
        movwf   CONTA           ; Se carga CONTA con 16 decimal

CARGA   movlw   0xf0
        movwf   TMR_OPT         ; Se carga TMR0 con 12 decimal

bucle   btfsc   PUERTAA,0       ; Explora RA0 y brinco si vale 0
        goto    ra0_1           ; salta a RA0_1
        bcf     PUERTAB,0       ; Si RA0 = 0 sa saca por RB0 un 0
        goto    ra1x            ; A explorar RA1

ra0_1   bsf     PUERTAB,0       ; Si RA0 = 1 se saca por RB0 un 1

ra1x    btfsc   PUERTAA,1       ; Examina ra1  y brinco si es 0
        goto    ra1_1           ; Salta  si RA1 = 1
        bcf     PUERTAB,1       ; Si RA1 = 0, RB1 = 0

        goto    bucle2

ra1_1   bsf     PUERTAB,1       ; Si RA1 = 1 , RB1 = 1

bucle2  movf    PUERTAB,1       ;traslado la salida sobre si mismo
        btfsc   ESTADO,2        ; para verificar que vale cero
        goto    CARGA
        goto    bucle           ; Bucle indefinido, se sale por la
                                ; interrupción
```

```
;-------------------------------------------------------------------

inter   decfsz  CONTA,1         ; RSI.Decrementa CONTA y brinco si
                                 ;vale 0

        goto    seguir

conta_0 movlw   0x03             ; Si CONTA = 0 se carga
        movwf   CONTA

        btfsc   PUERTAB,7        ; Si RB7 = 0, brinco
        goto    rb7_1
        bsf     PUERTAB,7        ; Si RB7 = 0, se invierte
        goto    seguir

rb7_1   bcf     PUERTAB,7        ; Si RB7 = 1 , se invierte

seguir  movlw   b'10100000'      ; Se restaura INTCON por desactivar
        movwf   INTCON           ; las interrupciones el procesador

        movlw   0xf0
        movwf   TMR_OPT          ; Se recarga TMR0 con 12
        retfie

        end
```

Interrupción por cambio de estado en líneas RB7...RB4:

Esta interrupción ha sido diseñada específicamente para detectar la pulsación de una tecla correspondiente a un teclado matricial que se explora con 4 líneas de E/S. para esta función se destinan las lineas RB7...RB4 de la puerta B, que cada vez que cambia el estado lógico de una de ellas, fuerza al señalizador RBIF a ponerse a 1, y si los bit de permiso RBIE= GIE = 1, entonces se autoriza la interrupción.
Como todavía no hemos analizado el teclado matricial, desarrollaremos un programa de control que involucre esta interrupción junto a la producida externamente por RB0/INT.
El siguiente programa controla una alarma conectada a las cuatro puertas de un coche que cuando se activa una de ellas se produce una interrupción que activan dos zumbadores conectados a las salidas RA0 y RA1. La desactivación de estos últimos, se produce por otra interrupción externa en la entrada RB0/INT.

PROGRAMA

```
;ALARMA.ASM: Un PIC 16F84 controla la alarma de un coche:
;Conectados a RB4 - RB7 hay 4 sensores que controlan cada uno
;una puerta del coche. Cuando una de las puertas del coche es abierta
;su sensor manda un 1 por su línea respectiva cambiando esta de estado
;con lo que se provoca una interrupción y comienzan a sonar 2 bocinas
;(buzzer) conectados a RA0 y RA1.
;Para detener la alarma,el usuario debe mandar una señal infrarroja a
;un sensor, que cuando la detecta, activa la patita RB0 con lo que se
;produce una interrupción, las bocinas se paran y el programa vuelve a
;su comienzo.
;-------------------------------------------------------------------
        LIST    P=16F84
        RADIX   HEX
;-------------------------------------------------------------------
        W       EQU     0
```

```
        F       EQU     1
        ESTADO  EQU     0x03
        PUERTAA EQU     0x05
        PUERTAB EQU     0x06
        INTCON  EQU     0x0B
;-----------------------------------------------------------------------
        ORG     0
        goto    inicio
        ORG     4
        goto    inter
        ORG     5

inicio  bsf     ESTADO,5        ; Banco 1
        movlw   b'00000000'
        movwf   PUERTAA         ; PUERTAA salidas
        movlw   b'11111111'
        movwf   PUERTAB         ; PUERTAB entradas
        bcf     ESTADO,5        ; Banco 0

        clrf    PUERTAA
        clrf    PUERTAB

        movlw   b'10011000'     ; Se activan GIE, Int externa
        movwf   INTCON          ; y Int por cambio de PB

bucle   goto    bucle
;-----------------------------------------------------------------------

inter   btfss   INTCON,0        ; Explora flag  int por cambio RB4 RB7
        goto    parar
        goto    alarma

alarma  clrf    PUERTAB
        movlw   b'10011000'     ; Se activan GIE, Int externa
        movwf   INTCON          ; y Int por cambio de PB
bocina  bsf     PUERTAA,0
        NOP
        bcf     PUERTAA,0
        bsf     PUERTAA,1
        NOP
        bcf     PUERTAA,1
        goto    bocina

parar   clrf    PUERTAA
        bcf     PUERTAB,0
        movlw   b'10011000'     ; Se activan GIE, Int externa
        movwf   INTCON          ; y Int por cambio de PB
        goto    bucle

        end
```

Interrupción por finalización de la escritura en la EEPROM de datos:

Dado que el tiempo típico para la operación de escritura de la EEPROM de datos de los PIC16X8X es de 10 ms, este valor resulta alto en comparación con la velocidad del procesador del microcontrolador. Para asegurarse entonces que la escritura se ha completado y se puede continuar con el flujo de control del programa, se aconseja utilizar la interrupción que se produce al finalizar la escritura, que pone automáticamente el señalizador EEIF a 1(del registro

EECON1) y se autoriza siempre que los bits EEIE= GIE= 1 (correspondientes al registro INTCON).
Previo a dar un ejemplo de este tipo de interrupción, veremos primero las características principales de la memoria de datos EEPROM

LA MEMORIA DE DATOS EEPROM:

Los PIC 16X8X tienen 64 bytes de memoria EEPROM de datos, donde se pueden guardar datos o variable que interesan que no se pierdan cuando se desconecta la fuente de alimentación al sistema. (Soportan 1.000.000 de ciclos de escritura/borrado y 40 años sin alterarse)
Para poder leerla y escribirla, durante el funcionamiento normal, se utilizan cuatro registros especiales ubicados en la memoria de datos RAM. Estos son:
_EEDATA
_EEADR
_EECON1
_EECON2

En el registro **EEADR**, ubicado en la dirección 0x09 del banco cero, se carga la dirección a acceder de la EEPROM , sea en lectura o escritura. Las direcciones son desde la 0x00 hasta la 0x3f (64 direcciones), por lo tanto los dos bit de mayor peso de **EEADR** son siempre cero.
En el registro **EEDATA**, ubicado en la dirección 0x08 del banco cero, se depositan los datos a leer o guardar.
El registro **EECON1**, que ocupa la dirección 0x08 del banco uno, tiene misiones de control de las operaciones en la EEPROM, siendo los siguiente

REGISTRO EECON1

------	------	------	EEIF	WRERR	WREN	WR	RD

RD: lectura
RD se pone a 1 cuando se va a realizar la operación de lectura. Luego se pone automáticamente a cero

WR: escritura
WR se pone a 1 cuando se inicia el proceso de escritura. Cuando se completa, se pone automáticamente a cero

WREN: permiso de escritura
1: permite la escritura de la EEPROM
0: prohíbe la escritura

WRERR: señalizador de error en escritura
1: se pone a cuando una operación de escritura ha terminado prematuramente
0: La operación de escritura se ha completado correctamente

EEIF: Señalizador de final de operación de escritura
1: indica que la operación de escritura se ha completado correctamente. Se pone a 0 por programa.
0: La operación de escritura no se ha completado.

El registro **EECON2** en realidad no esta implementado físicamente. Al leerlo, todos sus bit son cero. Se lo emplea como un dispositivo de seguridad durante el proceso de escritura de la EEPROM, para evitar las interferencias en el largo intervalo de tiempo que precisa su desarrollo.

Proceso de lectura:

El ciclo de lectura, se inicia colocando la dirección a acceder en el registro **EEADR** y se coloca el bit **RD=1** en el registro **EECON1**. El dato leído estará disponible en el registro **EEDATA** en el siguiente ciclo y permanecerá en él, hasta que se realice una nueva lectura o escritura en le EEPROM.
Veamos una porción de programa para leer la posición 0x01 de la memoria EEPROM de datos:

```
Bcf        0x03,5   ; paso al banco cero haciendo RP0=0 en el registro de ESTADO
Movlw      0x01     ; cargo en W el literal 0x01 (direcc.a acceder)
Movwf      0x09     ; cargo dirección 0x01 en registro EEADR
Bsf        0x03,5   ; paso al banco uno
Bsf        0x08,0   ; llevo a 1 RD del reg.EECON1 para realizar lectura de EEPROM
Bcf        0x03,5   ; paso al banco cero
Movf       0x09,0   ; llevo el dato de EEDATA al registro de trabajo W
```

Proceso de escritura:

El ciclo de escritura comienza cargando en EEADR la dirección de la posición a escribir y en el registro EEDATA el valor a grabar. luego se deben guardar en el registro no real EECON2 los valores 0x55 y 0xaa. Previo a esta escritura debe anularse la interrupción gral (GIE=0). Una vez escritos los valores en EECON2, se debe habilitar la interrupción gral (GIE=1) y la interrupción por escritura de la EEPROM , es decir el bit 6 (EEIE=1) del registro INTCON.

Veamos una porción de programa durante el proceso de escritura:

```
;comienzo con el proceso de guardar entrada en EEPROM-------------
                ;-----------------------------------------------
                bcf       0x03,5 ;paso al banco cero
                bcf       0x08,4 ;borro señalizador de INT escritura
                movf      0x0d,0 ;cargo direccion de EEPROM
                movwf     0x09   ;el registro EEADR
                movf      0x0c,0 ;cargo W con contenido de 0x0c
                movwf     0x08   ;cargo dato en EEDATA
                bsf       0x03,5 ;banco 1
                bcf       0x0b,7 ;anulo interrupcion gral GIE=0
                bsf       0x08,2 ;permiso de escritura
                                 ;inicio secuencia de escritura
                movlw     0x55   ;
                movwf     0x09   ;se escribe 55 H en EECON2
                movlw     0xaa
                movwf     0x09   ;se escribe AA H en EECON2
                bsf       0x08,1 ;comienza la escritura
                bsf       0x0b,7 ;habilito interrupción gral GIE=1
                bsf       0x0b,6 ;habilito interrupción por EEPROM
                bcf       0x03,5 ;banco cero
       bucle2   goto      bucle2 ;bucle infinito sale por
                                 ;interrupción
;.................................................................
```

Daremos a continuación como programa didáctico, un ejercicio para simulación, consistente en grabar en memoria EEPROM, 10 datos en direcciones consecutivas,
su lectura y observancia a través de las salidas del PIC.

```
;--------------------  EEP2.ASM  -------------------------------
;Programa que permite grabar y leer en memoria EEPROM,10 direcciones
;en forma secuencial, desde la 0x00 hasta la 0x09 inclusive.
;Procedimiento:1)se introduce el dato binario (4 bits)con las entradas
```

```
;RA0..RA3.
;               2)Pulsando RB4,se habilita la entrada del dato binario
;               3)Los datos se graban y luego se leen por RB0...RB3
;               4)La dirección leída aparece por RB4...RB7
;               5)El proceso se repite hasta grabar y leer la ultima
;dirección, donde finaliza el programa

                LIST      P= 16F84
                RADIX     HEX

                ORG       0
                goto      INICIO
                ORG       4
                goto      bucle3

                ORG       5
      INICIO    bsf       0x03,5 ;banco 1
                movlw     0xff   ;defino a RA como entradas
                movwf     0x05
                clrf      0x06   ;defino a RB como salidas
                bcf       0x03,5 ;banco 0
                clrf      0x06   ;borro salidas
                clrf      0x05   ;borro entradas
                clrf      0x0c   ;borro registro 0x0c
                movlw     0xff
                movwf     0x0d
      bucle1    btfss     0x05,4 ;espero que se pulse RA4
                goto      bucle1
                bcf       0x05,4 ;borro RA4
                movf      0x05,0   ;cargo entrada en W(RA0..RA3)
                movwf     0x0c   ;guardo entrada en 0x0c

  ;comienzo con el proceso de guardar entrada en EEPROM--------------
                incf      0x0d

                bcf       0x08,4 ;borro señalizador de int. escritura
                movf      0x0d,0 ;cargo dirección de EEPROM
                movwf     0x09   ;el registro EEADR
                movf      0x0c,0 ;cargo W con contenido de 0x0c
                movwf     0x08   ;cargo dato en EEDATA
                bsf       0x03,5 ;banco 1
                bcf       0x0b,7 ;anulo interrupción gral GIE=0
                bsf       0x08,2 ;permiso de escritura
                                 ;inicio secuencia de escritura
                movlw     0x55   ;
                movwf     0x09   ;se escribe 55 H en EECON2
                movlw     0xaa
                movwf     0x09   ;se escribe AA H en EECON2
                bsf       0x08,1 ;comienza la escritura
                bsf       0x0b,7 ;habilito interrupción gral GIE=1
                bsf       0x0b,6 ;habilito interrupción por EEPROM
                bcf       0x03,5 ;banco cero
      bucle2    goto      bucle2 ;bucle infinito sale por interrupció

      bucle3    bcf       0x0b,7 ;anulo interrupción gral GIE=0
                movf      0x0d,0 ;cargo dirección de EEPROM
                movwf     0x09   ;en el registro EEADR
                bsf       0x03,5 ;banco 1
                bsf       0x08,0 ;autorizo la lectura
                bcf       0x03,5 ;banco 0
```

```
movf      0x08,0 ;cargo dato en W
swapf     0x0d,1 ;intercambio nibles de 0x0d
iorwf     0x0d,0 ;0r entre W y 0x0d resultado en W
movwf     0x06   ;llevo dato a la salida
swapf     0x0d,1 ;normalizo 0x0d
comf      0x0d,0 ;complemento 0x0dresultado en W
andlw     0x09   ;and entre W y literal 0x09
btfss     0x03,2 ;reviso si la operación anterior
                 ;resulto cero(ultima dirección
                 ;grabada en la EEPROM)
goto      bucle1
end
```

REINICIALIZACION O RESET

En el caso particular de los PIX16X8X, tienen cinco causas que provocan la reinicialización del sistema, que consiste en cargar al contador de programa con el valor 000 H (vector reset) y poner el estado de los bits de los registros específicos (SFR) con un valor determinado. Se detallan a continuación las causas posibles de “reset”:

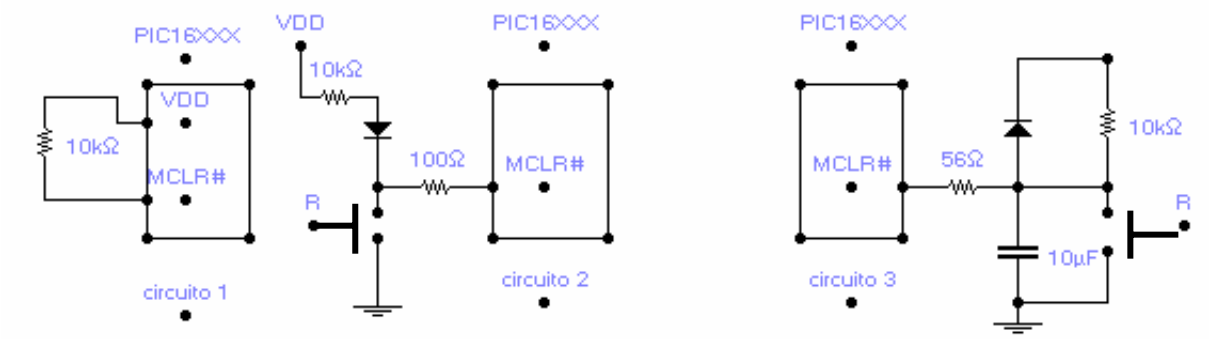
-Activación de la entrada ”MCLR# (Master Clear Reset) en funcionamiento normal
-Activación de la entrada “MCLR# en estado de reposo
-Conexión de la alimentación. POR (Power on Reset)
-Desbordamiento del “perro guardián” en funcionamiento normal
-Desbordamiento del “perro guardián “ en estado de reposo

Circuito de reinicialización (reset)

Colocando la entrada MCLR# a un nivel de tensión bajo, el microcontrolador entra en un estado de “reset” en el cual todas las salidas pasan a un estado bajo y el reloj se desactiva.
La entrada MCLR# dispone de un filtro para evitar que se active por señales de ruido.
Dentro del mapa de memoria de instrucciones, existe una dirección, denominada “vector reset”, siendo esta dirección donde el microcontrolador comienza la ejecución del programa. Para los PIC de la gama media y alta , corresponde a la dirección 0x0000.
Para los PIC de rango bajo, es la 0x01FF. Cada vez que el microcontrolador ingresa en un estado de reset, la CPU del mismo se dirige a esta posición de la memoria, para iniciar nuevamente la ejecución de todas las instrucciones del programa.

Reinicialización por conexión de la tensión de alimentación

Este estado de reset (Power On Reset: POR) se origina en el momento de detectarse un estado alto en la alimentación. Para aprovechar esta característica, se puede conectar una resistencia desde el pin MCLR# hasta el voltaje de alimentación (circ. 1) El circuito 2 muestra una aplicación típica para reinicialización POR y exterior por pulsador. El circuito 3 se utiliza cuando el sistema posee una fuente de alimentación de baja rampa de crecimiento; el diodo ayuda a descargar el capacitor cuando VDD se desactiva.



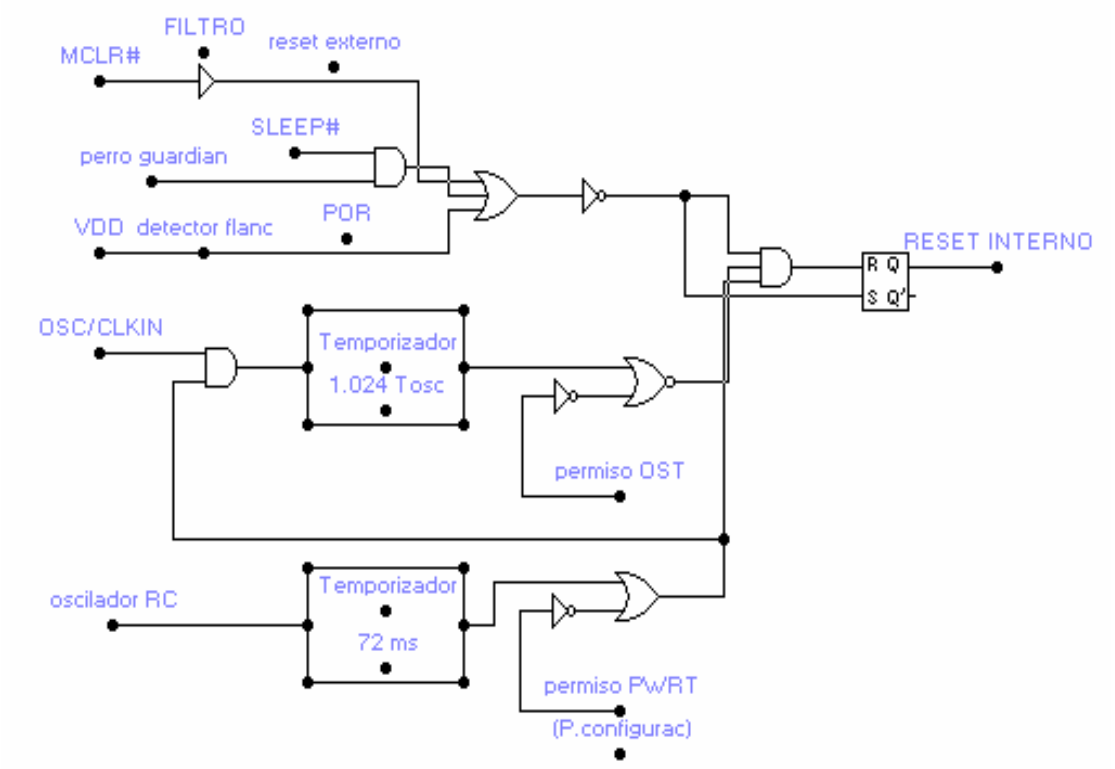
Cuando el dispositivo sale del estado de reset y comienza su operación normal, los parámetros de operación (tensión, frecuencia, temperatura, etc.) deben encontrarse dentro de sus valores normales de operación; de otra manera el microcontrolador no funcionara correctamente. El retardo provisto por el capacitor permite un retardo suficiente para la normalización.

Temporizador al encendido (PWRT)

El PWRT (Power Up Timer) es una característica que se puede incorporar en el momento de grabar un programa en la memoria del microcontrolador. Esto proporciona un retardo de 72 ms a la reinicialización POR o a la BOR. El PWRT se basa en un oscilador RC interno dedicado. El microcontrolador permanecerá en estado de reset mientras PWRT se encuentre activo. El retardo provisto por el PWRT le permite al voltaje de alimentación VDD alcanzar el nivel aceptable. La autorización del PWRT se logra en el bit PWRTE de la palabra de configuración.

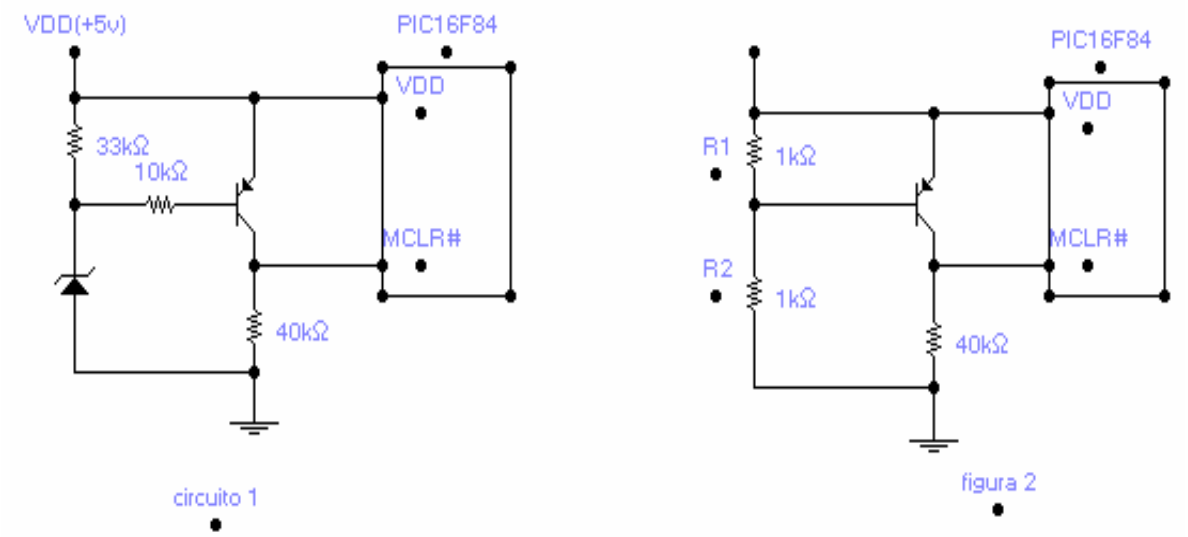
Temporizador de oscilación al encendido OST

El temporizador OST(Oscillator Start Timer) proporciona un retardo de 1.024 periodos del oscilador (aplicados al terminal OSC1/CLKIN) y sirve para asegurar que el cristal de cuarzo o resonador cerámico empleados en los osciladores tipo XT, LP, o HS este estabilizado y en marcha
La secuencia que sigue al alimentar el sistema es el siguiente: Primero se detecta el POR; si este esta habilitado, se invoca el PWRT. Después que termina el tiempo del PWRT, se activa el OST.
Se muestra en lo que sigue, el esquema electrónico para la generación del “reset”



Reinicialización por caída en la tensión de alimentación BOR

Algunos microcontroladores PIC disponen de un reset por caída en la tensión de alimentación. El reset BOR (Brown Out Reset) es muy utilizado en aplicaciones de líneas de CA o en aplicaciones donde se conmuten cargas y donde el voltaje puede caer temporalmente.
Para aquellos microcontroladores que no poseen este tipo de reset, se lo puede realizar con un circuito externo como los siguientes:



Para el circuito 1, cuando VDD desciende por debajo del valor VZ+ 0,7 volt, el transistor pnp se bloquea, lo que provoca un nivel bajo de tensión sobre el terminal MCLR#, provocando un reset por nivel bajo de tensión de alimentación (BOR).
En el circuito 2, el transistor pnp se bloquea y activa el reset al pasar a nivel bajo el terminal MCLR# cuando el valor VDD desciende por debajo de: VDD.R1/(R1+R2=0,7v.

Determinación del tipo de reset

Las condiciones que provocaron un reset en el microcontrolador se pueden determinar a través de los **bits TO# y PD#** del registro **de ESTADO** según la siguiente tabla y que puede ser consultados mediante una instrucción de lectura (movf 0x 03,0).

TO#	PD#	CONDICION DE RESET
1	1	POR (RESET POR CONEXIÓN VDD)
0	1	DESBORDAMIENTO WDT EN FUNCIONAMIENTO NORMAL
0	0	DESBORDAMIENTO WDT EN ESTADO DE REPOSO
1	1	ACTIVACION MCLR# EN FUNCIONAMIENTO NORMAL
1	0	ACTIVACION MCLR#EN REPOSO

MODO DE REPOSO O BAJO CONSUMO

El modo de reposo o bajo consumo, esta caracterizado por el reducido consumo de energía del microcontrolador, y esta recomendado en aquellas aplicaciones donde hay que esperar largos periodos de tiempo hasta que se produzca un suceso asincrónico, como por ejemplo una interrupción externa.
El consumo típico de un PIC de clase media es de 2ma aprox. ; cuando ingresa en el modo de bajo consumo pasa a menos de 10 µa.
Para pasar al modo de bajo consumo, es necesario ejecutar la instrucción “SLEEP”; Después de ejecutada, el microcontrolador pasa a un estado de bajo suministro de energía, manteniéndose en un estado sin actividad.
En este estado de reposo, el terminal TOCKI se conecta a VDD o a tierra, para eliminar la entrada de impulsos externos al TMR0. Por otra parte, como se detiene el oscilador principal que genera los impulsos Tosc. , También se para TMR0. Los terminales de Entrada / salida mantienen el estado anterior al de reposo y las que no se hallan conectados a periféricos y actúan como entradas de alta impedancia se aconseja conectarlas a VDD o a tierra, para evitar posibles fugas de corriente. El terminal MCLR# debe conectarse a nivel alto. Sin impulsos reloj,

el microcontrolador deja de ejecutar instrucciones hasta que se lo saque de ese estado (despierte).
Si el perro guardián continúa activo en el modo de reposo, al entrar en él, se borra, pero continúa funcionado. Los bits del registro de ESTADO TO# y PD# toman los valores 1 y 0 respectivamente.
Para salir del estado de "reposo" existen tres alternativas:

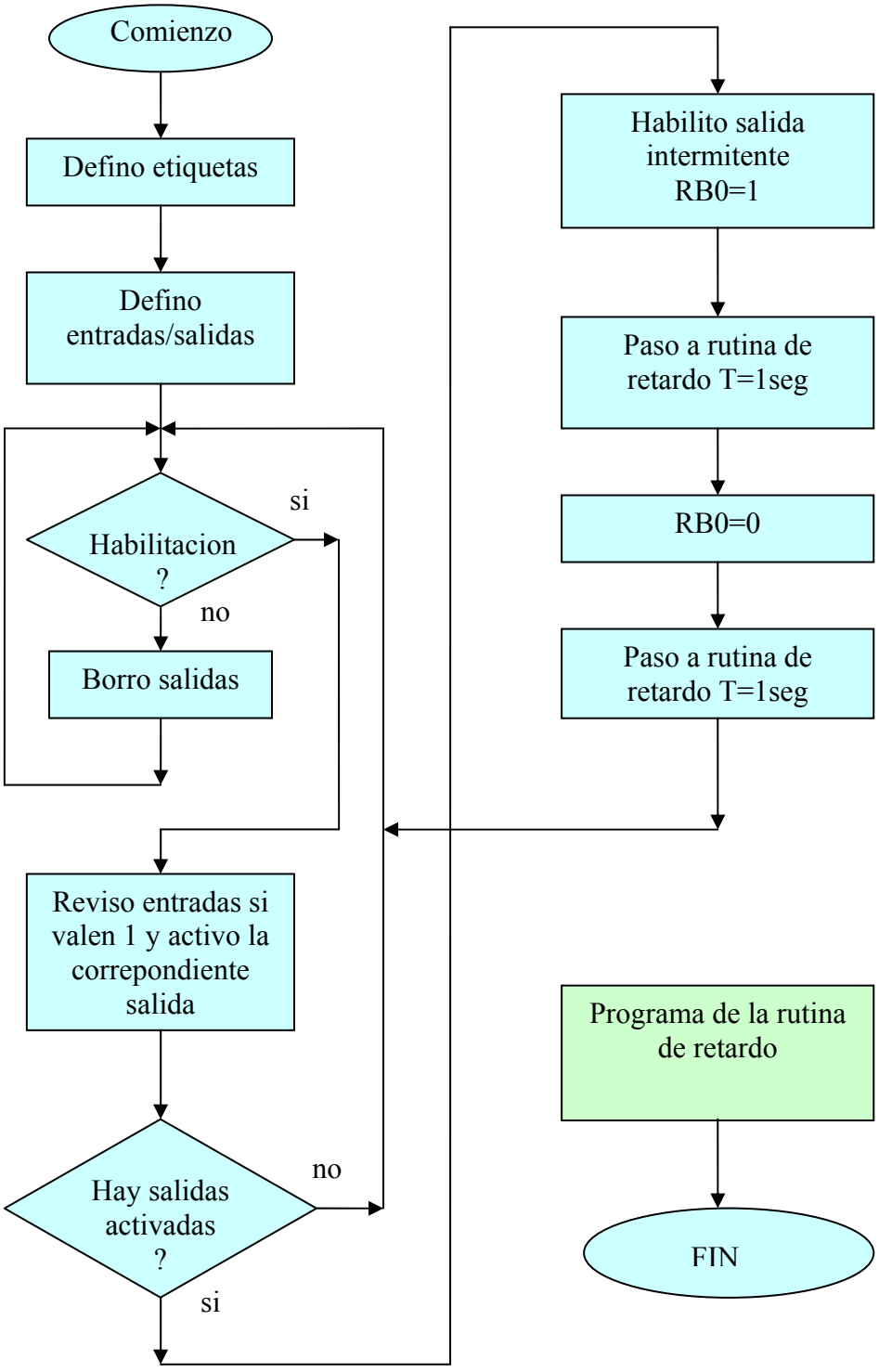
1) **Activación externa de MCLR# para provocar un "reset"**
2) **Desbordamiento del perro guardián si quedo operativo en el modo de reposo**
3) **Generación de una interrupción. (menos la de TMR0,dado su inactividad)**

Cuando el microcontrolador "despierta", desarrolla la secuencia del oscilador OST, que retarda 1.024 Tosc. Para estabilizar la frecuencia de trabajo, y luego pasa a ejecutar la instrucción siguiente a "sleep" a sea PC+1.
Los bits TO# y PD# se emplean para conocer la causa del reset que despierta al sistema. PD# pasa a 0 cuando se ejecuta la instrucción "sleep". TO# pasa a 0 cuando se desborda el perro guardián.

DIVERSOS PROYECTOS RESUELTOS DE AUTOMATISMOS
PROYECTO N°1

Realizaremos un programa de alarma (alarma1.asm), con tres sensores de entrada y uno de inhabilitación. Tendremos tres salidas, que identificaran las entradas activadas y quedaran fijas. Otra salida será intermitente y se activara con cualquiera de las salidas fijas. Todas las salidas pasaran a cero, cuando se active (1) la entrada de inhabilitación. El sistema volverá a activarse cuando la entrada de inhabilitación pase a cero.
El programa lo realizaremos inspeccionando en forma sucesiva el estado de cada una de las tres entradas conectadas a los sensores, que darán señales de 1 cuando se activen; utilizaremos entonces instrucciones de salto condicionado para revisar estas entradas.
Para la entrada de inhabilitación, también usaremos una instrucción de salto condicionado, para su revisión. Para la salida intermitente, usaremos como rutina de retardo, el contador TMRO, y una instrucción de salto condicionada para el final de la cuenta. El tiempo real seria de 1 segundo; nosotros le daremos menor tiempo para apreciarlo mejor en el programa de simulación.

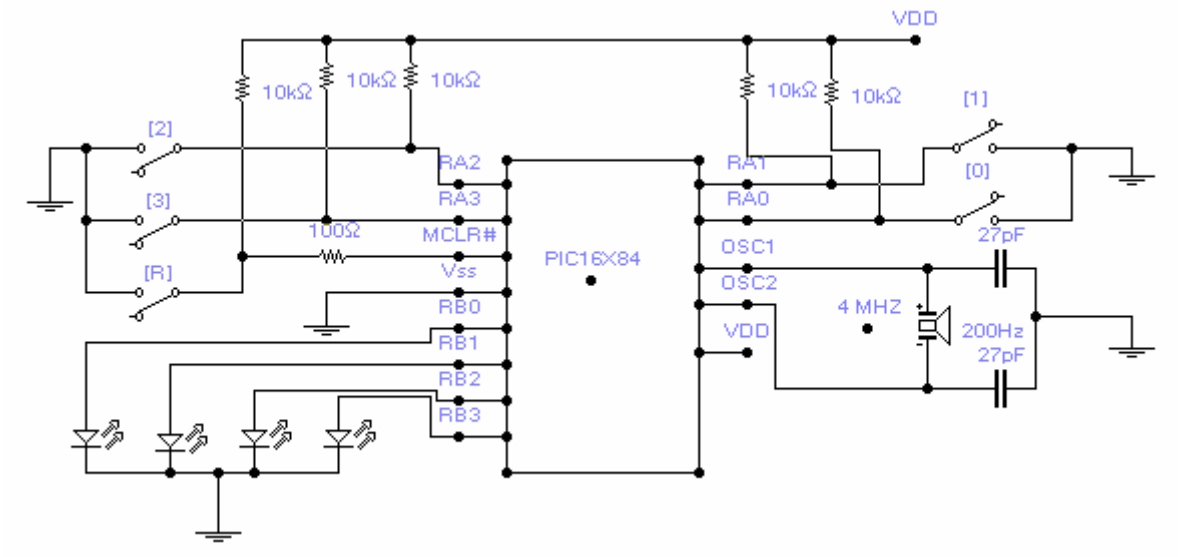
DIAGRAMA DE FLUJO ALARMA.ASM



Pasamos a mostrar a continuación el archivo de texto ALARMA.ASM que contiene el listado de instrucciones para ejecutar el automatismo propuesto

```
          ;ALARMA1.ASM
          ;RA0=0habilitacion. RA0=1 anulación
          ;RA1, RA2,RA3 entradas de alarma
          ;RB1, RB2,RB3 indicación de activación
          ;RB0 intermitencia

        LIST      P=16F84
        RADIX     HEX

        ORG       0
        goto      INICIO
        ORG       5
INICIO  bsf       0x03,5 ;banco 1
        movlw     0xff
        movwf     0x05   ;RA0..RA4 entradas
        clrf      0x06   ;RB0..RB7 salidas
        bcf       0x03,5 ;banco cero
        clrf      0x05
        clrf      0x06
BUCLE1  btfsc     0x05,0 ;verifico si la entrada RA0
        goto      BORR   ;esta habilitando o no la
        goto      REV1   ;alarma
BORR    clrf      0x06
        goto      BUCLE1

REV1    btfss     0x05,1 ;inspecciono la entrada RA1
        goto      REV2   ;salto a entrada RA2 si RA1=0
        bsf       0x06,1 ;coloco salida RB1=1si RA1=1
REV2    btfss     0x05,2 ;reviso estado entrada RA2
        goto      REV3   ;salto a entrada RA3 si RA2=0
        bsf       0x06,2 ;coloco salida RB2=1 si RA2=1
REV3    btfsc     0x05,3 ;reviso estado entrada RA3
        bsf       0x06,3 ;coloco salida RB3=1 si RA3=1
        movf      0x06,0 ;verifico estado de las salidas
        andlw     0x0E   ;determino el estado del
        btfsc     0x03,2 ;indicador de cero Z y salto
        goto      BUCLE1 ; si vale cero

INTER   bsf       0x06,0 ;coloco en 1 salida intermitente
        call      RETARDO;paso a la rutina de retardo
        bcf       0x06,0 ;coloco en 0 salida intermitente
        call      RETARDO;paso a rutina de retardo
        goto      BUCLE1 ;regreso a verificar la habilita-
                         ;cion y se¤alar en la salida las
                         ;entradas activadas

RETARDO bsf       0x03,5 ;banco 1
        movlw     0xd3   ;habilito el contador TMR0
        movwf     0x01
        bcf       0x03,5 ;banco 0
        clrf      0x01   ;llevo a cero al contador
EXPLORA btfss     0x01,4 ;exploro si llego a la cuenta
        goto      EXPLORA;si no llego, exploro de nuevo
        return           ;retorno a la direccion de lla-
                         ;mada de la rutina
        end
```

Circuito practico

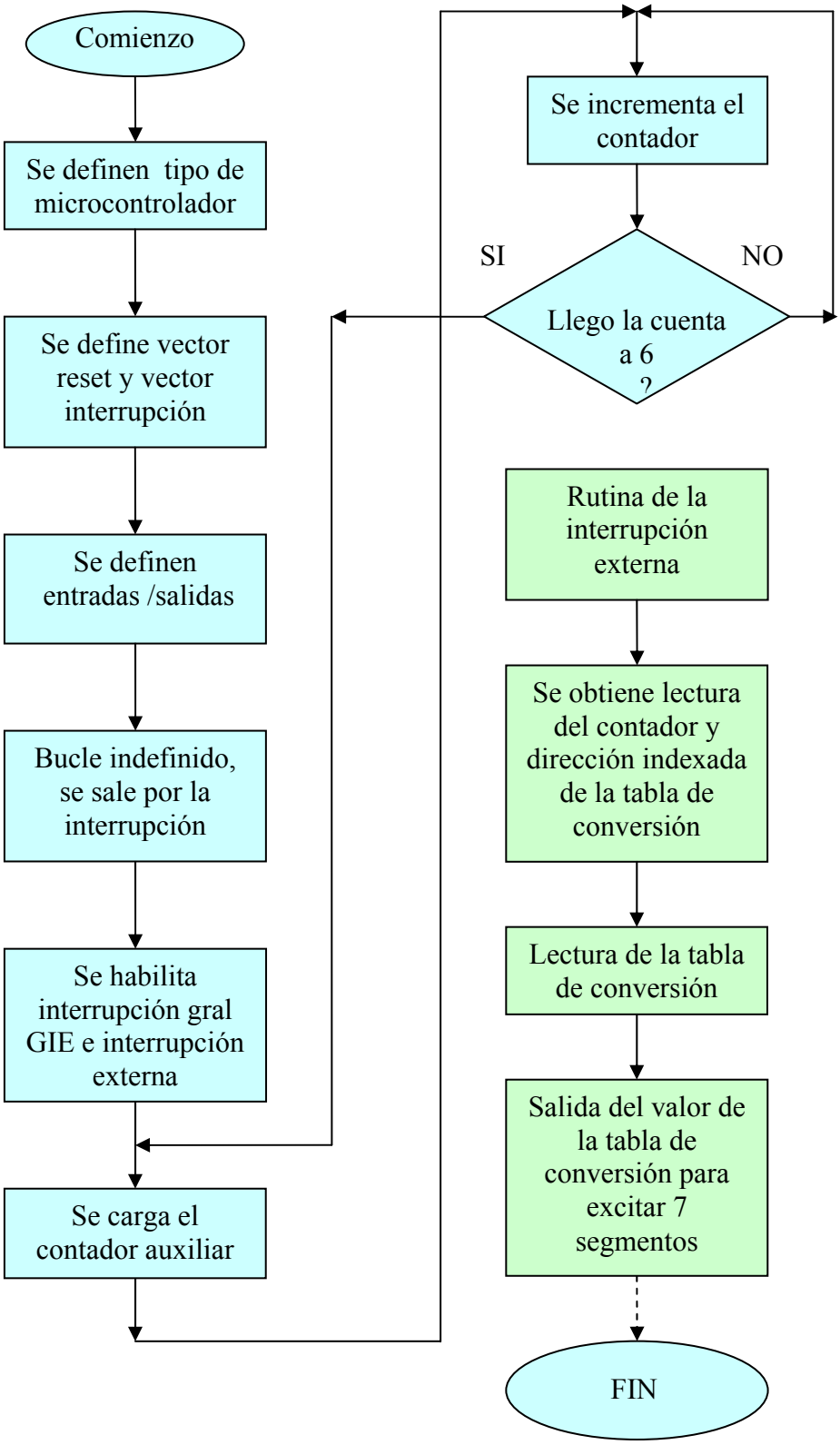


NOTA: El programa desarrollado, fue preparado para su simulación en el software “SIMUPIC”. Para que se pueda ejecutar en el circuito practico, es necesario realizar una pequeña modificación, consistente en cambiar la lógica de los valores de las entradas, es decir cambiar los” ceros” lógicos por” unos” lógicos. Además el tiempo de intermitencia en la salida RB0 no es simétrico dado que se agrega el tiempo de ejecución de las instrucciones. Para el circuito practico, con un retardo de 1seg, la intermitencia resulta prácticamente simétrica.

PROYECTO Nº 2

Realizaremos un programa para un juego como el caso del “Tiro del dado”, que consiste en sacar un numero arbitrario al azar, comprendido entre el nº1 y el nº 6 incluidos. La base de este programa, consiste en generar un contador,(usando un registro auxiliar) que cuente entre el nº1 y el nº6 en forma cíclica. Mediante una interrupción, se lee el estado actual del contador, y mediante una tabla de conversión, aplicando direccionamiento indirecto, se convierte el numero leído, en una salida para excitar un display de 7 segmentos.
A los efectos de que la probabilidad de lectura del contador sea la misma para cualquier número, se deberá prestar atención, para que el tiempo transitorio que permanece cada número, sea el mismo
El programa se desarrollara sin definir etiquetas.
Desarrollaremos primero el diagrama de flujo:

DIAGRAMA DE FLUJO "DADO.ASM"



PROGRAMA DADO.ASM

```
;DADO.ASM
  ;programa que simula un dado electronico con salida 7 segmentos
  ;(RB1...RB7)con numeros aleatorios de 1, 2, 3, 4, 5, 6. "El tiro del
dado"
  ;se realiza por un pulso en la entrada RB0/INT  que provoca una
  ; interrupcion externa (flanco descendente).

               LIST        p= 16F84
               RADIX       HEX

               ORG         0
               goto        INICIO ;inicio del programa
               ORG         4
               goto        INT    ;salto a la rutina interrup.


         INICIO bsf        0x03,5 ;selecciono banco 1
                movlw      0xff   ;cargo en W literal ff H
                movwf      0x05   ;RA0...RA4 son entradas
                movlw      0x01   ;cargo en W literal 01 H
                movwf      0x06   ;RB0 entrada,RB1..RB7 salidas
                bcf        0x01,6 ;flanco descendente int. externa
                bcf        0x03,5 ;paso al banco cero

          tabla movlw      0x0c   ;cargo la tabla de conversion
                movwf      0x0d   ;salida n§1
                movlw      0xb6
                movwf      0x0e   ;salida n§2
                movlw      0x9e
                movwf      0x0f   ;salida n§3
                movlw      0xcc
                movwf      0x10   ;salida n§4
                movlw      0xda
                movwf      0x11   ;salida n§5
                movlw      0xfa
                movwf      0x12   ;salida n§6

                movlw      0x90   ;cargo W con literal 90 H
                movwf      0x0b   ;cargo registro INTCON,habilito
                                  ;interrupcion GIE y externa INTE
                                  ;desactivo resto interrupciones
                                  ;y borro se¤alizador externo INTF

                clrf       0x05   ;borro entradas
                clrf       0x06   ;borro salidas

      bucle2    movlw      0x01   ; inicio la cuenta del contador
                movwf      0x0c   ; cargando registro 0c  con 01 Hex

                nop               ; tiempo de espera para igualar
                nop               ; los tiempos de permanencia del
                nop               ; valor de cuenta del contador para
                nop               ; y equilibrar  probabilidades de
                nop               ; salida de los numeros
                nop

      bucle1    incf       0x0c,1 ; incremento 0c una unidad
```

```
            movlw       0x06    ; cargo W con 0x06
            subwf       0x0c,0  ; producto logico W y 0c

            btfss       0x03,2  ; reviso Z y salto si vale 1
            goto        bucle1
            goto        bucle2


INT         movf        0x0c,0 ;cargo valor del contador en W
            addlw       0x0c   ;obtengo direcc. Indexada
            movwf       0x04   ;direcc. Indirecto
            movf        0x00,0 ; obtengo dato de la tabla
            movwf       0x06   ;presento dato en salida RB1..RB7
            bcf         0x0b,1 ;borro bit INF de interrupción.
            Retfie             ;vuelvo al programa principal
            end
```

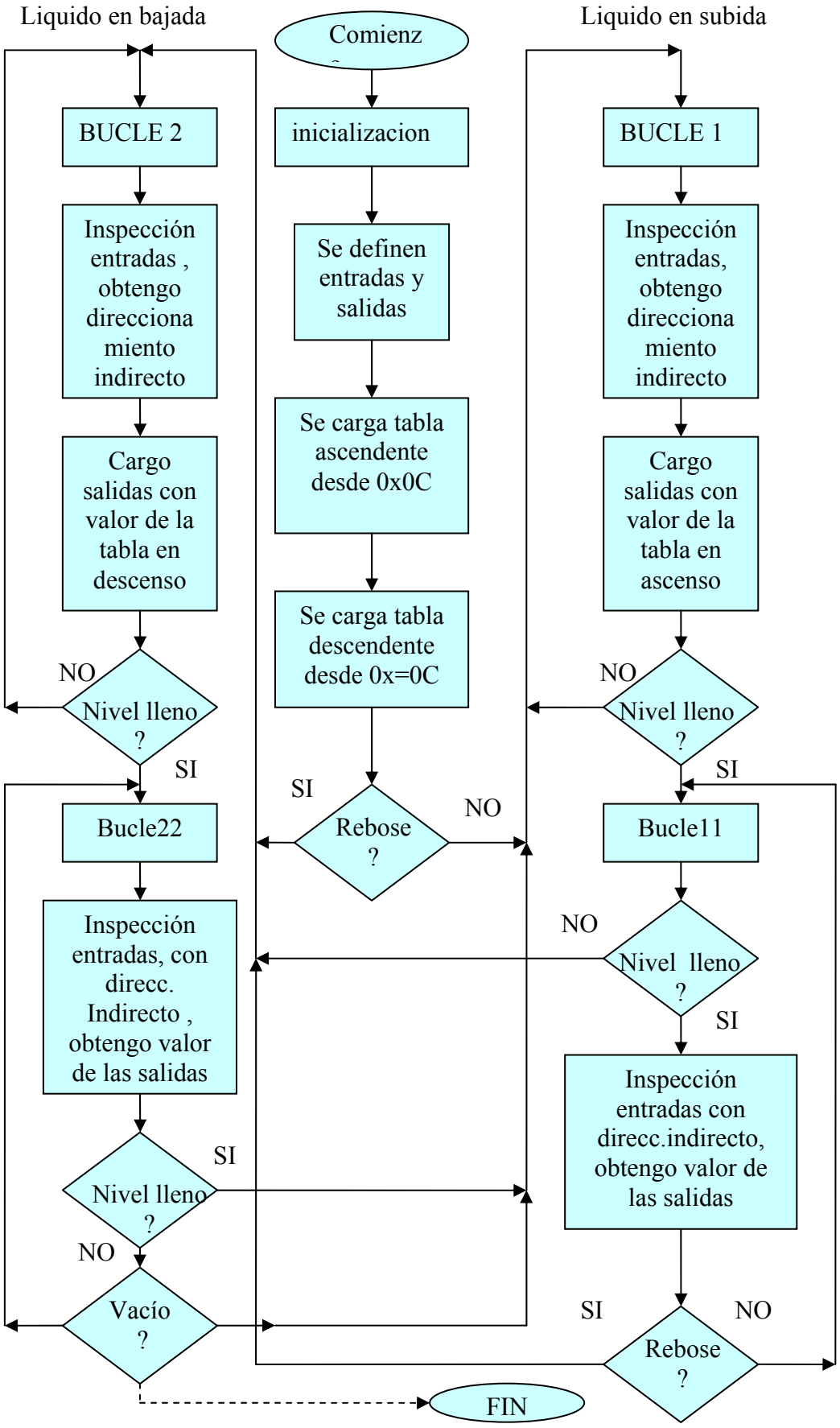
PROYECTO Nº4

Modificación automatismo "Nivel tanque.asm"

Este automatismo presentado en el temario nº3, adolece en la práctica, de un inconveniente no previsto en su desarrollo original. Este, consiste en la marcha y parada de una de las bombas, tanto con el nivel de líquido en subida como en bajada, durante las situaciones que se detecta el nivel de "lleno".
El programa que desarrollaremos a continuación, salva este inconveniente, presentando una especie de "histéresis" en el nivel mencionado.
Siguiendo la propuesta original, se desarrollo esta "histéresis" o secuencialidad, para evitar el continuo arranque y parada de las bombas en este nivel, con instrucciones de salto de direcciones del programa , condicionadas.
Esta modificación del "archivo fuente", la podemos utilizar como ejemplo de aplicación de un programa ensamblado y simulado correctamente, pero que en la practica, presenta los inconvenientes mencionados.

Desarrollamos a continuación el diagrama de flujo del programa modificado, llamándolo "NIVEL TANQUE1.ASM"

DIAGRAMA DE FLUJO "NIVEL TANQUE1.ASM"



PROGRAMA "NIVEL TANQUE1.ASM"

```
                    ; NIVEL TANQUE1.ASM

;Programa que controla el nivel de liquido de un tanque a través
;de tres sondas "A"(VACIO), "B"(LLENO), "C"(REBOSE),con acciona-
;miento de dos bombas electromecánicas B1 y B2.Además con indi-
;caciones de "alarma","rebose","lleno" y "vacío"
; A= RA2  B= RA1   C= RA0
; RB0= bomba B2  RB1=bomba B1   RB2= vacío  RB3= lleno
; RB4= rebose    RB5=alarma (discordancia)

                LIST       P=16F84

               RADIX      HEX

               ORG        0      ;coloco en el vector reset la
                                 ;instrucción que sigue abajo
               goto       INICIO ;salto incondicional a INICIO
               ORG        5      ;coloco en la dirección 0x05
                                 ;la próxima instrucción.

     INICIO    bsf        0x03,5 ;paso al banco 1
               movlw      0xff   ;cargo literal 0xff en W
               movwf      0x05   ;W=0x05.RA0..RA4 son entradas
               movlw      0xc0   ;cargo literal 0xc0 en W
               movwf      0x06   ;W=>0x06. RB0...RB5 son entradas
               bcf        0x03,5 ;paso al banco cero
               clrf       0x05   ;coloco en 0 entradas
               clrf       0x06   ;coloco en 0 salidas

             ; TABLA ASCENDENTE
             ; ----------------
               movlw      0x07   ;Guardo tabla Nø1 ascendente
               movwf      0x0c   ;a partir de la direcc. 0x0c
               movlw      0x20   ;hasta la direcc. 0x13
               movwf      0x0d
               movwf      0x0e
               movwf      0x0f
               movwf      0x11
               movlw      0x03
               movwf      0x10
               movlw      0x09
               movwf      0x12
               movlw      0x18
               movwf      0x13

             ;TABLA DESCENDENTE 1
             ;-------------------

               movlw      0x07   ;Guardo tabla Nø2 descendente
               movwf      0x14   ;a partir de la direcc. 0x14
               movlw      0x20   ;hasta la direcc. 0x1B
               movwf      0x15
               movwf      0x16
               movwf      0x17
               movwf      0x19
               movlw      0x01
               movwf      0x18
               movlw      0x08
```

```
            movwf       0x1a
            movlw       0x18
            movwf       0x1b



            btfss       0x06,4 ;Salto condicional revisando
                               ;si se produjo REBOSE en salida
                               ; despues de actuar el V.reset
            goto        BUCLE1 ;salto condicional a BUCLE1
            goto        BUCLE2 ;salto condicional a BUCLE2

 BUCLE1     movf        0x05,0 ; cargo entradas en W
            andlw       0x07   ; enmascaro W con literal 0x07
            addlw       0x0c   ;obtengo dirección indexada para
                               ;tabla ascendente. W+0x0c
            movwf       0x04   ;W=>FSR direccionamiento indirecto
            movf        0x00,0 ;obtengo dato tabla ascendente
                               ;a través del registro INDF
            movwf       0x06   ;W=>0x06 presento dato en salida
            btfsc       0x06,3
            goto        BUCLE11
            goto        BUCLE1

BUCLE11     btfss       0x06,3
            goto        BUCLE2
            movf        0x05,0 ; cargo entradas en W
            andlw       0x07   ; enmascaro W con literal 0x07
            addlw       0x0c   ;obtengo dirección indexada para
                               ;tabla ascendente. W+0x0c
            movwf       0x04   ;W=>FSR direccionamiento indirecto
            movf        0x00,0 ;obtengo dato tabla ascendente
                               ;a través del registro INDF
            movwf       0x06   ;W=>0x06 presento dato en salida
            btfss       0x06,4 ;salto condicional revisando si
                               ;se produjo alarma REBOSE en salida
            goto        BUCLE11 ;salto incondicional a BUCLE11
            goto        BUCLE2 ;salto incondicional a BUCLE2



 BUCLE2     movf        0x05,0 ;cargo valor de entradas en W
            andlw       0x07   ;enmascaro W con literal 0x07
            addlw       0x14   ;obtengo dirección indexada para
                               ;tabla descendente W+0x14
            movwf       0x04   ;W=>FSR direccionamiento indirecto
            movf        0x00,0 ;obtengo dato tabla descendente
                               ;a través del registro INDF
            movwf       0x06   ;presento dato en salida
            btfsc       0x06,3   ;reviso salida “lleno” salto si
            goto        BUCLE2 ;vale cero (tabla descendente)
            goto        BUCLE22

 BUCLE22    movf        0x05,0 ;cargo valor de entradas en W
            andlw       0x07   ;enmascaro W con literal 0x07
            addlw       0x14   ;obtengo dirección indexada para
            movwf       0x04   ;W=>FSR direccionamiento indirecto
            movf        0x00,0 ;obtengo dato tabla descendente
                               ;a través del registro INDF
```

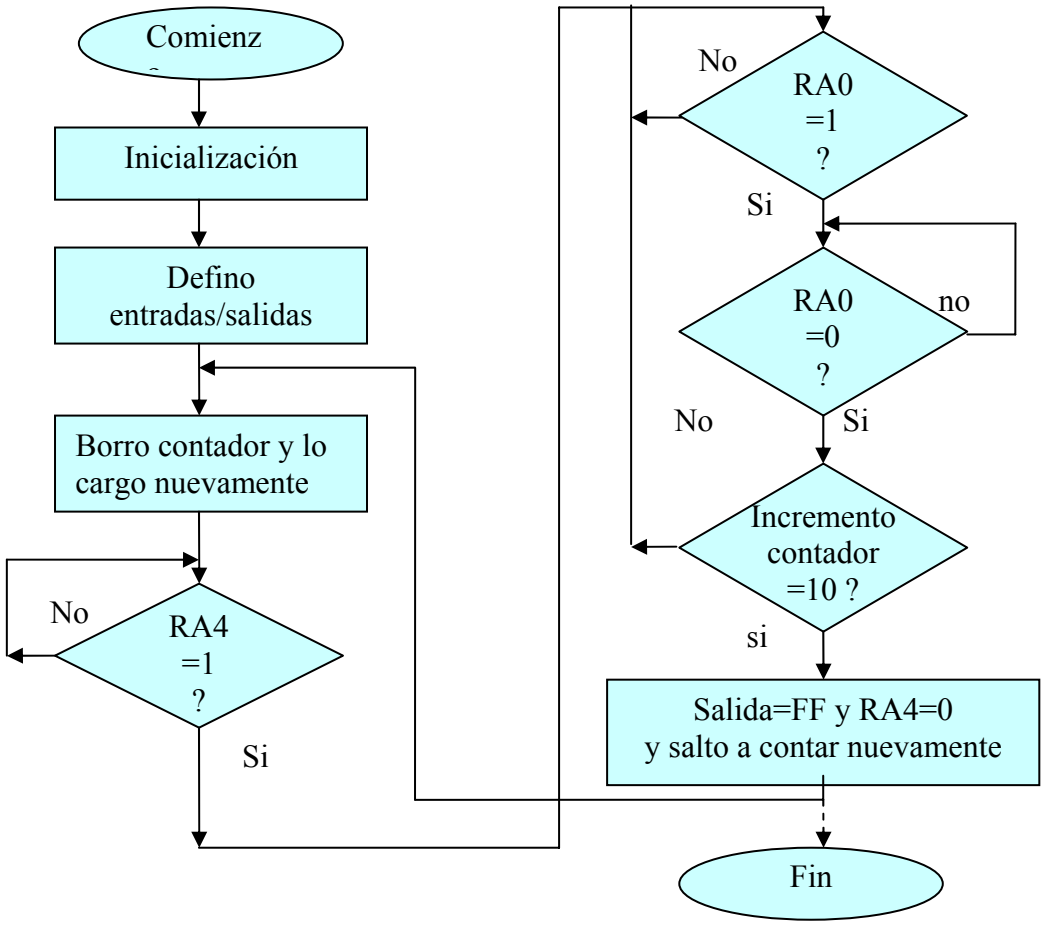
```
                    ;tabla descendente W+0x14
movwf       0x06    ;presento dato en salida
btfsc       0x06,3  ;reviso nuevamente salida "lleno"
goto        BUCLE1  ;cambio a tabla ascendente
btfsc       0x06,2  ;salto condicional revisando si
                    ;se produjo alarma VACIO en salida
goto        BUCLE1  ;salto incondicional a BUCLE1
goto        BUCLE22 ;salto incondicional a BUCLE2

end
```

PROYECTO Nº6: Saltos.asm

El siguiente programa es un ejemplo de aplicación de las instrucciones de salto condicionada btfsc , btfss y incfsz, consistente en contar 10 pulsos, contabilizados a partir de su flanco descendente, y previa habilitación mediante la entrada RA4=1. Al finalizar la cuenta, RA4=0, inhabilitándose el contador, hasta tanto RA4 pase a valer 1.

La detección del pulso a contar , en su flanco descendente, se logra con dos instrucciones de salto condicionado, btfss y btfsc, sobre el bit de entrada RA0. Para el contador, se utiliza un registro auxiliar, precargado con el complemento del numero binario natural, correspondiente al nueve; de esta manera al incrementar este registro con la instrucción de salto condicionada "incfsz", cuando se cuenta el décimo pulso, el registro queda con el valor hexadecimal "FF" , la instrucción "incfsz" lo incrementa pasa a 0x00) y genera un salto de dirección , ocasión que se aprovecha para presentar el valor "0xFF en la salida e inhabilitar la nueva cuenta a 10.

DIAGRAMA DE FLUJO :SALTOS.ASM



PROGRAMA : SALTOS.ASM

```
;                    saltos.asm
; programa que cuenta hasta 10 pulsos por la entrada RA0 (en
descenso),
; previa habilitacion con RA4 (1). Cuando se llega a la cuenta final,
;la salida (RB0....RB7)pasa a 1(FF, y se inhabilita la cuenta.


                    LIST  P=16F84

                     RADIX  HEX
          estado    equ    0x03  ;defino etiquetas de registros e
          entrada   equ    0x05  ;instrucciones
          salida    equ    0x06
          contador  equ    0x0c
          #define   borrar  clrf
                     ORG     0
                     GOTO     INICIO ;instrucción a cargar en 0x00
                     ORG     5

        INICIO      bsf    estado,5  ;instrucción a cargar en 0x05
                    movlw  0xff      ; defino entradas y salidas
                    movwf  entrada
                    borrar   salida
                    bcf    estado,5
                    borrar entrada
                    borrar salida
          conta     borrar contador
                    movlw  0xf6
                    movwf  contador    ; cargo contador con f6
        habilito    btfss  entrada,4   ;bucle infinito, se sale
                    goto   habilito    ;cuando RA4=1
                    borrar salida
          bucle1    btfss  entrada,0   ;bucle infinito se sale
                    goto   bucle1      ;cuando RA0=1
          bucle2    btfsc  entrada,0   ;Bucle infinito, se sale
                    goto   bucle2      ;cuando RA0=0
                    incfsz contador,1  ;se incrementa el registro
                                       ;contador
                    goto   bucle1      ; pero si vale FF, salta
                                       ;una instruccion
                    bcf    entrada,4   ;inhabilito el contador
                    movlw  0xff        ;cargo w con ff y deposito
                    movwf  salida      ;su valor en la salida
                    goto   conta       ;salto para volver a contar
                                       ; si lo habilito (RA4=1)
                    end
```