NOTAS DEL PIC16F84 PARA LA UEA

SISTEMAS DIGITALES II

José Ignacio Vega Luna
Gerardo Salgado Guzmán
Mario Alberto Lagos Acosta
Víctor Noé Tapia Vargas

Electrónica

UNIVERSIDAD AUTÓNOMA METROPOLITANA
Casa abierta al tiempo
Azcapotzalco

NOTAS DEL PIC16F84 PARA LA UEA

SISTEMAS DIGITALES II

NOTAS DEL PIC16F84
PARA LA UEA
SISTEMAS DIGITALES II

Este material fue dictaminado y aprobado por el Consejo Editorial de la División de Ciencias Básicas e Ingeniería, de la Unidad Azcapotzalco de la UAM, en su sesión del día 1 de octubre del 2003.

NOTAS DEL PIC16F84 PARA LA UEA

SISTEMAS DIGITALES II

José Ignacio Vega Luna
Gerardo Salgado Guzmán
Mario Alberto Lagos Acosta
Víctor Noé Tapia Vargas

UNIVERSIDAD AUTÓNOMA METROPOLITANA
Casa abierta al tiempo Azcapotzalco
División de Ciencias Básicas e Ingeniería
Departamento de Electrónica

UAM-AZCAPOTZALCO

RECTOR
Mtro. Víctor Manuel Sosa Godínez

SECRETARIO
Mtro. Cristian Eduardo Leriche Guzmán

COORDINADORA GENERAL DE DESARROLLO ACADÉMICO
Mtra. María Aguirre Tamez

COORDINADORA DE EXTENSIÓN UNIVERSITARIA
DCG Ma. Teresa Olalde Ramos

JEFA DE LA SECCIÓN DE PRODUCCIÓN Y DISTRIBUCIÓN EDITORIALES
DCG Silvia Guzmán Bofill

ISBN: 970-31-0259-X

José Ignacio Vega Luna
Gerardo Salgado Guzmán
Mario Alberto Lagos Acosta
Víctor Noé Tapia Vargas

Diseño de Portada:
Modesto Serrano Ramírez
Revisión de texto científico
Ma. Dolores Cárdenas Valdés

Universidad Autónoma Metropolitana
Unidad Azcapotzalco
Av. San Pablo 180
Col. Reynosa Tamaulipas
Delegación Azcapotzalco
C.P. 02200
México, D.F.

Sección de producción
y distribución editoriales
Tel. 5318-9222/9223
Fax. 5318-9222

Notas del PIC16F84 para la UEA
Sistemas digitales II,
1a. edición, 2004

Impreso en México.

NOTAS DEL PIC16F84 PARA LA UEA

SISTEMAS DIGITALES II

José Ignacio Vega Luna
Gerardo Salgado Guzmán
Mario Alberto Lagos Acosta
Víctor Noé Tapia Vargas

UNIVERSIDAD AUTONOMA METROPOLITANA
Casa abierta al tiempo Azcapotzalco
División de Ciencias Básicas e Ingeniería
Departamento de Electrónica

ISBN: 970-31-0259-X

José Ignacio Vega Luna
Gerardo Salgado Guzmán
Mario Alberto Lagos Acosta
Víctor Noé Tapia Vargas

Diseño de Portada:
Modesto Serrano Ramírez
Revisión de texto científico
Ma. Dolores Cárdenas Valdés

Universidad Autónoma Metropolitana
Unidad Azcapotzalco
Av. San Pablo 180
Col. Reynosa Tamaulipas
Delegación Azcapotzalco
C.P. 02200
México, D.F.

Sección de producción
y distribución editoriales
Tel. 5318-9222/9223
Fax 5318-9222

Notas del PIC16F84 para la UEA
Sistemas digitales II,
1a. edición, 2004

Impreso en México.

Este material fue realizado con el objetivo de apoyar a los profesores que impartan y a los alumnos que cursen las UEAs "Sistemas Digitales II" y "Laboratorio de Sistemas Digitales II" del Area de Concentración de Sistemas Digitales y Computadoras del Departamento de Electrónica de la Universidad Autónoma Metropolitana-Azcapotzalco en la Licenciatura de Ingeniería Electrónica.

En estas notas se cubren todos los temas que marcan los programas analíticos oficiales y aprobados en la UAM de las UEAs mencionadas anteriormente, en los que se indica que el profesor podrá usar cualquier microcontrolador para impartir dichas materias. En este caso el microcontrolador usado es el PIC16F84 que en la actualidad es uno de los mas ampliamente usados en aplicaciones de control de propósito general.

Este material es resultado de la recopilación de notas, ejercicios, cursos tomados y la experiencia misma de los autores. Se ha realizado sin fines de lucro y con objetivos puramente didácticos y académicos para el constante desarrollo y actualización de la carrera de Ingeniería Electrónica.

Deseamos agradecer la colaboración en la elaboración de las figuras y diagramas a los alumnos: Aquino Aquino Miguel y Arellano Mendoza Alejandro Santiago de Ingeniería Electrónica.

José Ignacio Vega Luna
Gerardo Salgado Guzmán
Mario A. Lagos Acosta
Víctor Noé Tapia Vargas

TABLA DE CONTENIDO

UNIDAD I

INTRODUCCION A LOS MICROCONTROLADORES

I.1. Evolución de los microcontroladores

El desarrollo cada vez más acelerado de los microcontroladores ha sido posible a las tecnologías recientes que incorporan cientos y cientos de transistores en un circuito integrado, que en un principio fueron microprocesadores para computadoras que fueron construidas adicionando periféricos externos como memoria, líneas de entrada-salida, temporizadores y otros dispositivos a esos microprocesadores. Con el tiempo, se diseñaron y construyeron circuitos que contuvieron tanto a los procesadores como a sus periféricos los cuales han sido llamados microcontroladores.

En el año de 1969 un equipo de ingenieros Japoneses llegó a Estados Unidos para solicitar a Intel la construcción de algunos circuitos para calculadoras. Por parte de Intel el responsable del proyecto fue Marcian Hoff quien tenía experiencia con una computadora PDP8. Hoff sugirió un sistema que requería más memoria de la que los Japoneses proponían surgiendo así el primer microprocesador del que en 1971 Intel obtuvo los derechos para comercializar como el 4004 que fue un microprocesador de 4 bits con una velocidad de 6000 operaciones por segundo.

Posteriormente, la compañía CTC solicitó a Intel y a Texas Instruments la fabricación de un microprocesador de 8 bits para usarlo en terminales y en Abril de 1972 apareció en el mercado el primer microprocesador de 8 bits llamado el 8008 que podía accesar 16 kb de memoria y con 45 instrucciones, operando a una velocidad de 300,000 operaciones por segundo. Intel mantuvo el desarrollo de microprocesadores y en Abril de 1974 apareció el procesador de 8 bits llamado el 8080 con 75 instrucciones y accesado 64 kb de memoria, con un precio inicial de $360 USD.

Por su parte, Motorola colocó en el mercado el microprocesador de 8 bits 6800 junto con otros periféricos como el 6820 y el 6850. Posteriormente MOS Technology fabricó en 1975 los microprocesadores de 8 bits 6501 y 6502 con 56 instrucciones y capacidad para direccionar 64 kb de memoria un costo de $25 USD, lo que lo hizo muy popular utilizándose en computadoras tales como: Apple, Comodore, Atari, Ultra y muchas más.

Federico Faggin de Intel fundó Zilog Inc. y en 1976 anunciaron el Z80. Tomando en cuenta que había ya muchos programas en el mercado que corrían en el 8080, el Z80 podía ejecutarlos sin problema, lo que lo hizo un microprocesador muy poderoso en su tiempo con 176 instrucciones, 64 kb de memoria, un gran número de registros y una opción integrada para refrescar memorias dinámicas, por lo que muchos sistemas se convirtieron de 8080 a Z80, el cual fue el corazón de muchas computadoras como Spectrum, Partner, TRS703 y Z-3.

En 1976 Intel apareció con una versión mejorada del microprocesador de 8 bits: el 8085, sin embargo el Z80 seguía siendo mejor y ya todo estaba decidido. Ya no hubo grandes mejoras por parte de los fabricantes y el Z80, el 6502 y el 6800 fueron los microprocesadores más representativos de 8 bits.

En los años 80´s aparecieron los primeros microcontroladores, lo cuales difieren de los microprocesadores en muchas formas. La primera y más importante es su funcionalidad. Para que un microprocesador se pueda usar se le deben agregar otros componentes como memoria y dispositivos para enviar y recibir datos. Un microcontrolador esta diseñado para ser todo en uno, ya que tiene incorporados todos los periféricos necesarios ahorrando tiempo y espacio al construir dispositivos electrónicos.

I.2. Familias de microcontroladores

Se han desarrollado muchos tipos y modelos de microcontroladores hasta la fecha, unos con más componentes integrados que otros pero todos con el mismo fin. En este curso nos enfocaremos en dos de ellos: el PIC16F84 y el 8051 de Intel. El primero de ellos por ser uno de los más exitosos de los últimos años y el 8051 por ser uno de los más básicos que aun con algunas ligeras modificaciones que su versión original se sigue usando en el diseño de algunos dispositivos electrónicos en la actualidad.

I.3. Comparación de microcontroladores frente a microprocesadores

CISC y RISC

El PIC16F84 tiene una arquitectura RISC y Harvard. La arquitectura Harvard es una arquitectura más nueva que la Von-Neumann. En la arquitectura Harvard el objetivo es acelerar la velocidad de trabajo de un microcontrolador, ya que los buses de datos y direcciones están separados, proporcionando un mayor flujo de datos entre la CPU y los bloques periféricos y memoria. El separar la memoria de datos y la memoria de programa hace posible además poder contar con instrucciones de palabras de más de 8 bits. El PIC16F84 usa 14 bits para las instrucciones, lo que permite que todas sus instrucciones sean de una palabra. También es típico de la arquitectura Harvard el contar con menos instrucciones que la Von-Neumann, las cuales se ejecutan en un ciclo de maquina.

Los microcontroladores con arquitectura Harvard también son llamados "microcontroladores RISC". RISC significa Reduced Instruction Set Computer y los microcontroladores con arquitectura Von-Neumann también son llamados "microcontroladores CISC". CISC significa Complex Instruction Set Computer.

Figura I.1. La arquitectura Harvard vs. la arquitectura Von-Neumann

El PIC16F84 tiene un conjunto reducido de instrucciones: 35 (los microcontroladores de Intel y Motorola tienen mas de 100 instrucciones). Todas esas instrucciones se ejecutan en un ciclo, excepto las instrucciones de salto. Usualmente alcanza resultados de 2:1 en compresión de código y 4:1 en velocidad con respecto a otros microcontroladores de 8 bits de su clase.

El PIC16F84 pertenece a una clase de microcontroladores de 8 bits de arquitectura RISC y cuenta con los siguientes bloques funcionales:

Memoria de programa (FLASH) – para almacenar un programa escrito. Esta memoria está construida con tecnología FLASH y puede programarse y borrarse más de una vez, lo que hace que el PIC16F84 sea un microcontrolador útil en el desarrollo de dispositivos.

EEPROM – memoria de datos que necesitan salvarse aun cuando el circuito no esté alimentado. Usualmente usada para almacenar datos importantes que no deben perderse si la fuente de alimentación es interrumpida abruptamente. Por ejemplo la temperatura asignada por un regulador de temperatura.

RAM – memoria de datos usada por un programa durante su ejecución. En la memoria RAM se almacenan todos los datos temporales durante la operación normal del sistema.

Puerto A y puerto B (PORTA, PORTB) – son conexiones físicas entre el microcontrolador y el mundo exterior. El puerto A tiene cinco líneas o pines y el puerto B tiene 8 líneas.

Temporizador (FREE-RUN TIMER) – es un registro de 8 bits del microcontrolador que trabaja independientemente del programa. Cada cuatro pulsos del reloj del oscilador incrementa su valor en uno hasta que alcanza su máximo (255), y a continuación inicia nuevamente su conteo desde cero. Ya que se conoce el tiempo exacto entre dos incrementos. El temporizador o timer se puede usar para medir el tiempo en algunos dispositivos o eventos.

CPU (CENTRAL PROCESSING UNIT) – tiene el papel de elemento de conexión entre los bloques del microcontrolador y coordina el trabajo de los mismos para ejecutar el programa del usuario.

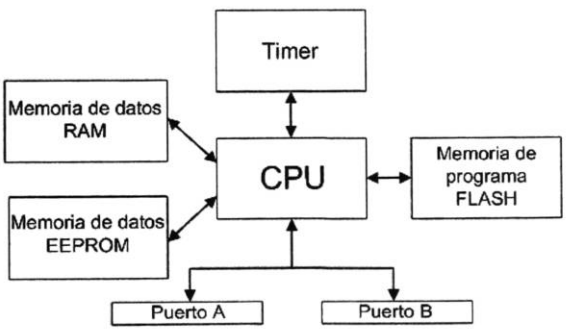


Figura I.2. Arquitectura interna general del PIC16F84

I.4. Ámbito de aplicación de los microcontroladores

El PIC16F84 se usa comúnmente en la industria automotriz, instrumentación industrial, aplicaciones de control doméstico, sensores remotos, automatización y dispositivos de seguridad y debido a su bajo consumo de energía es ideal para dispositivos alimentados por baterías.

La memoria EEPROM lo hace el indicado también para aplicaciones donde es necesario almacenar permanentemente varios parámetros como códigos de transmisores, velocidad de motores, frecuencia de recepción y otros.

Este chip cuenta con solo dos líneas para la transferencia de datos al programarlo, lo que lo convierte en un producto flexible para el desarrollo de prototipos y pasar a continuación a la producción en línea o bien puede ser usado para mejorar programas de productos terminados.

UNIDAD II

CARACTERISTICAS DE UN MICROCONTROLADOR

II.1. Diagrama a bloques

El reloj y el ciclo de instrucción

El reloj del microcontrolador es uno de los principales suministros, el cual es obtenido de un componente externo llamado el "oscilador". El reloj entra al microcontrolador por medio de la línea OSC1 donde un circuito interno del microcontrolador lo divide en cuatro ciclos iguales llamados Q1, Q2, Q3 y Q4. Estos cuatro ciclos constituyen un ciclo de instrucción (también llamado ciclo de máquina) durante el cual se ejecuta una instrucción.

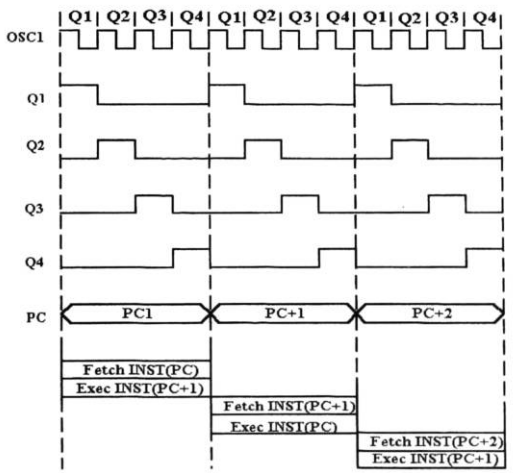


Figura II.1. Ciclo de Reloj/Instrucción del PIC16F84

Pipelining

La ejecución de una instrucción se inicia al llamar a la instrucción de la memoria de programa en cada ciclo Q1 y escribiéndola en el registro de instrucción (instruction register) durante Q4. La decodificación y ejecución de la instrucción se realizan entre los siguientes ciclos Q1 y Q4. En el diagrama siguiente se puede ver la relación entre el ciclo de instrucción y el reloj del oscilador (OSC1). El contador de programa (program counter-PC) contiene la dirección de la siguiente instrucción a ejecutar.

El ciclo de instrucción consiste de los ciclos Q1, Q2, Q3 y Q4, en donde gracias al pipeline cada instrucción se ejecuta efectivamente en un ciclo, pero si la instrucción causa un cambio en el program counter (el PC apunta a otra dirección), como por ejemplo saltos y llamadas a subrutinas, se necesitan dos ciclos para ejecutar la instrucción, ya que la instrucción debe ser procesada nuevamente.

Ejemplo:

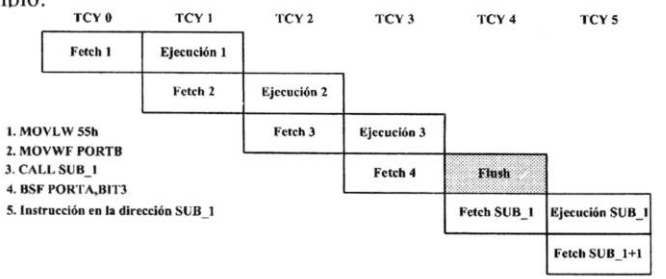


Figura II.2. Funcionamiento del pipeline de instrucciones

TCY0 lee la instrucción MOVLW 55h (no importando para nosotros en este momento que instrucción fue ejecutada).

TCY1 ejecuta la instrucción MOVLW 55h y lee MOVWF PORTB.
TCY2 ejecuta la instrucción MOVWF PORTB y lee CALL SUB_1.
TCY3 ejecuta la llamada a una subrutina CALL SUB_1 y lee la instrucción BSF PORTA,BIT3. Ya que esta instrucción no es la que se necesita o no es la primera instrucción de SUB_1, la instrucción debe ser leída nuevamente. Este es un claro ejemplo de una instrucción que necesita más de un ciclo para ejecutarse.

TCY4 se usa totalmente para leer la primera instrucción de la subrutina SUB_1.

TCY5 ejecuta la primera instrucción de SUB_1 y lee la siguiente instrucción.

Descripción de líneas

El PIC16F84 cuenta con 18 líneas. El encapsulado que se encuentra más frecuentemente en el mercado es del tipo DIP18 pero también existe como encapsulado SMD el cual es más pequeño que un DIP. DIP es una abreviación de Dual In Package. SMD es una abreviación de Surface Mount Devices.

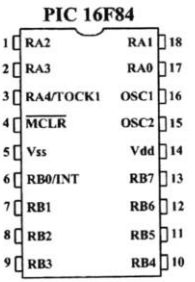


Figura II.3. Distribución de líneas del PIC16F84

Las líneas del PIC16F84 tienen el siguiente significado:

- Línea no. 1. **RA2** línea 2 del puerto A. No tiene función adicional.
- Línea no. 2. **RA3** línea 3 del puerto A. No tiene función adicional.

- Línea no. 3. **RA4** línea 4 del puerto A. Su función adicional es TOCK1 y es una línea de control del temporizador.
- Línea no. 4. **MCLR** entrada de reset y voltaje de programación Vpp del microcontrolador.
- Línea no. 5. **Vss** tierra.
- Línea no. 6. **RB0** línea 0 del puerto B. Funciona alternamente como entrada de interrupción.
- Línea no. 7. **RB1** línea 1 del puerto B. No tiene función adicional.
- Línea no. 8. **RB2** línea 2 del puerto B. No tiene función adicional.
- Línea no. 9. **RB3** línea 3 del puerto B. No tiene función adicional.
- Línea no. 10. **RB4** línea 4 del puerto B. No tiene función adicional.
- Línea no. 11. **RB5** línea 5 del puerto B. No tiene función adicional.
- Línea no. 12. **RB6** línea 6 del puerto B. Línea de reloj (clock) en el modo de programación.
- Línea no. 13. **RB7** línea 7 del puerto B. Línea de datos (data) en el modo de programación.
- Línea no. 14. **Vdd** alimentación.
- Línea no. 15. **OSC2** línea asignada para conectarlo con un oscilador.
- Línea no. 16. **OSC1** línea asignada para conectarlo con un oscilador.
- Línea no. 17. **RA0** línea 0 del puerto A. No tiene función adicional.
- Línea no. 18. **RA1** línea 1 del puerto A. No tiene función adicional.

Las líneas **RB4** a **RB7** pueden usarse también como entradas para detectar algún cambio de estado en las mismas.

II.2. Recursos internos

Generador de reloj – oscilador

Se usa un circuito oscilador para proporcionarle la señal de reloj al microcontrolador.

Tipos de osciladores

El PIC16F84 puede trabajar con cuatro diferentes configuraciones del oscilador. Las configuraciones hechas con un cristal y un circuito RC (resistor-capacitor) son las más frecuentemente usadas. La configuración con un cristal se designa XT y la segunda se denomina RC. Esto es importante porque, en ocasiones, cuando se compra o se configura un microcontrolador, es necesario especificar el tipo de oscilador.

Oscilador XT

El oscilador de cristal es un encapsulado metálico con dos líneas que tiene impresa la frecuencia a la cual oscila el cristal. Es necesario conectar un capacitor cerámico de 22 pF o 30 pF entre las terminales del cristal y tierra. En algunos casos se encuentran encapsulados los capacitores y el cristal en un circuito con tres líneas. Tal elemento se llama un resonador cerámico, las terminales del centro son la tierra mientras que las líneas de los extremos se conectan a las líneas OSC1 y OSC2 del microcontrolador. Es recomendable conectar el oscilador lo más cercano al microcontrolador para evitar interferencias en las líneas por las cuales el microcontrolador recibe la señal de reloj. La frecuencia del cristal es típicamente de 4 Mhz.

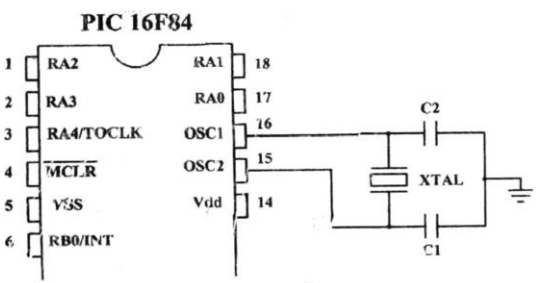


Figura II.4. Oscilador XT

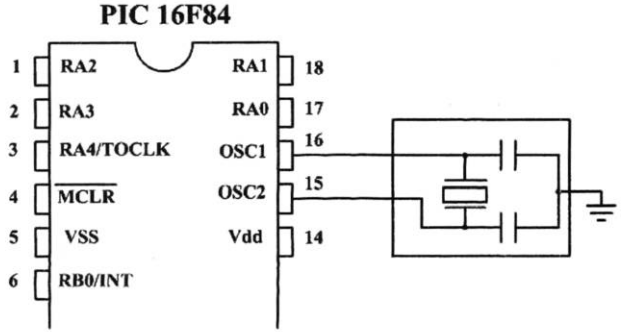


Figura II.5. Oscilador XT encapsulado

Oscilador RC

Se usa en aplicaciones donde no es necesaria una gran precisión de tiempo, este tipo de oscilador es más barato que el anterior y la frecuencia de resonancia del oscilador depende del valor de la resistencia R, el capacitor C, el valor del voltaje de alimentación, la temperatura de trabajo y la tolerancia de estos elementos.

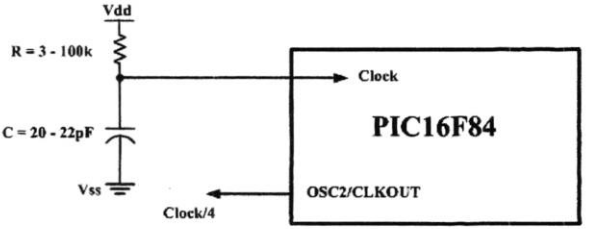


Figura II.6. Oscilador RC

En la figura anterior se muestra un oscilador RC conectado al PIC16F84. Con un valor menor a 2.2 k en la resistencia R, el oscilador puede volverse inestable y con valores muy altos, por ejemplo 1M, el oscilador

puede volverse muy sensible al ruido y la humedad, por lo que es recomendable usar resistencia de un valor entre 3 y 100 k.
Aunque el oscilador puede trabajar sin un capacitor externo (C=0 pF), se recomienda usar capacitores de 20 pF para el ruido y estabilidad.
No importando que tipo de oscilador que se use, el reloj es dividido entre 4 y puede obtenerse por medio de la línea OSC2/CLKOUT y emplearse para sincronizar otros circuitos externos.

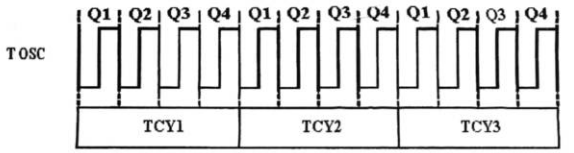


Figura II.7. Relación entre el reloj y ciclos de instrucción

Al alimentarse el microcontrolador el oscilador empieza a trabajar y al inicio la señal es inestable y con variaciones en su amplitud, pero con el tiempo se estabiliza.

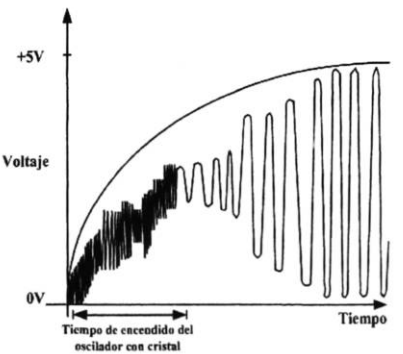


Figura II.8. Señal del oscilador al encender la fuente de alimentación.

Para evitar que la señal inestable del reloj tenga una mala influencia en el comportamiento del microcontrolador, es necesario mantener al microcontrolador en un estado de reset durante la estabilización del oscilador del reloj. La figura anterior muestra una señal de un oscilador de cuarzo al encender la fuente de alimentación.

El circuito de reset.

La señal de reset se usa para colocar al microcontrolador en un estado conocido. Bajo este estado, el contenido de los registros se lleva una posición de inicio. El reset no solo se usa cuando el microcontrolador se esta comportando de una forma no deseada, si no que también puede usarse para sacar al microcontrolador de un ciclo.
Para evitar colocar un cero lógico accidentalmente en la línea MCLR, esta línea debe conectarse vía un resistor al positivo de la fuente de alimentación. El valor del resistor debe ser entre 5 y 10 k. Este tipo de resistor es de los llamados pull up.

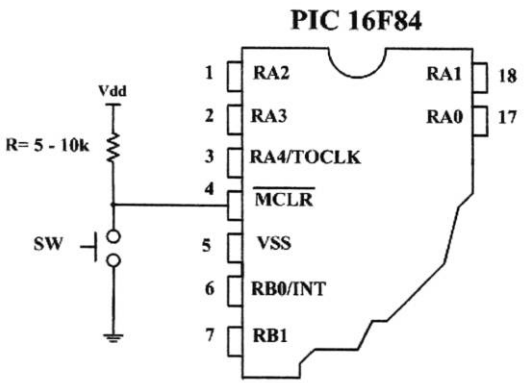


Figura II.9. Circuito de reset

El microcontrolador PIC16F84 conoce varias fuentes de reset:

1) Reset durante el encendido de la fuente de alimentación (también llamado Power On Reset o POR).
2) Reset durante trabajo regular llevando a cero lógico la línea MCLR.
3) Reset durante el estado SLEEP.
4) Reset al ocurrir un sobreflujo (overflow) en el watchdog timer (WDT).
5) Reset al ocurrir un sobreflujo (overflow) en el watchdog timer (WDT) encontrándose en el estado SLEEP.

Las fuentes más importantes son la primera y la segunda, la primera ocurre al encender la fuente de alimentación y sirve para llevar los registros internos a una posición o valor inicial y la segunda es más comúnmente usada durante el desarrollo de programas.

Durante un reset las localidades de la memoria RAM no son inicializadas. Al encender la fuente de alimentación esas localidades contienen un valor desconocido y al activar el reset su valor no es cambiado. Los registros del PIC16F84 conocidos como SFR (los cuales se verán posteriormente) si son inicializados a un valor conocido. El program counter es inicializado con 0000h.

Al encender la fuente de alimentación, y detectar un incremento en el voltaje de alimentación en Vdd (en el rango de 1.2V a 1.8V), el microcontrolador genera un pulso de reset. Ese pulso tiene una duración de 72ms, suficientes para que se estabilice el oscilador. Esos 72ms son proporcionados por un temporizador interno llamado PWRT el cual tiene su propio oscilador RC, el microcontrolador está en un estado de reset mientras el PWRT esté activo. Sin embargo, cuando el dispositivo ya se encuentra trabajando, puede surgir el problema de que la alimentación no cae a cero completamente, pero sí por abajo del límite que garantiza el correcto funcionamiento del microcontrolador. Este puede suceder en la práctica especialmente en ambientes industriales ruidosos y de

inestabilidad en la fuente de alimentación. Para resolver este problema es necesario asegurar que el microcontrolador se encuentre en un estado de reset cada vez que la fuente caiga por abajo del límite permitido.

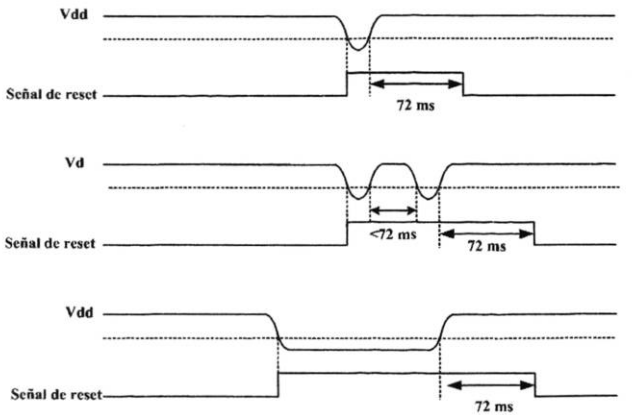


Figura II.10. Caída del voltaje de alimentación

Puede ser necesario usar componentes electrónicos especiales para generar la señal de reset y que aparezca un cero lógico en la línea MCLR hasta que el voltaje se encuentre en los límites correctos.

La unidad central de procesamiento (CPU)

La CPU es la encargada de buscar y decodificar las instrucciones que deben ser ejecutadas.

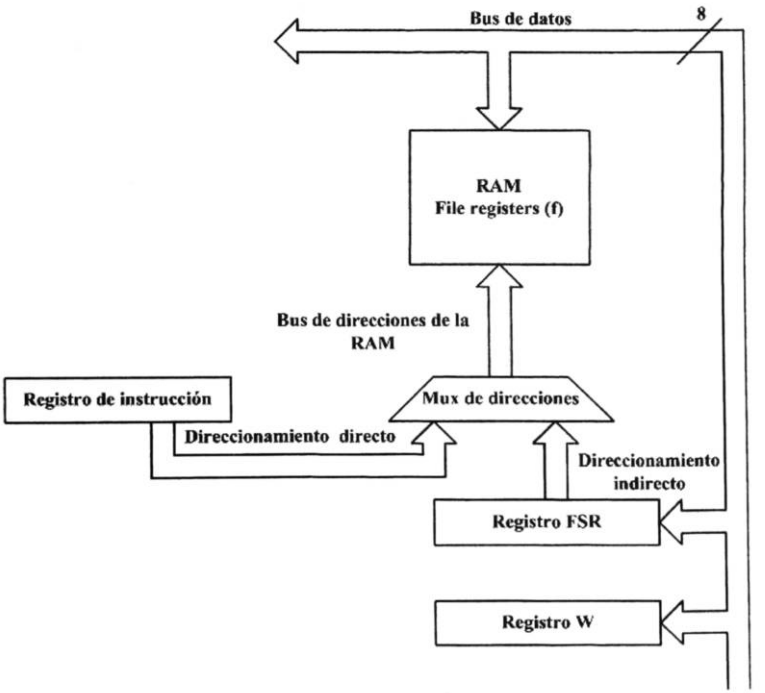


Figura II.11. Diagrama general de la CPU

La CPU conecta todas las partes del microcontrolador de manera centralizada y tal vez la función más importante es decodificar las instrucciones del programa. Esta conexión se lleva a cabo por medio de los buses de datos y direcciones.

La unidad aritmética y lógica (ALU)

La unidad aritmética y lógica es responsable de realizar las operaciones de adición, sustracción, movimientos (a la izquierda y derecha dentro de un registro) y operaciones lógicas. Los movimientos de datos dentro de un registro también se les conocen como corrimientos. El PIC16F84 tiene un ALU de 8 bits y registros de trabajo de 8 bits.

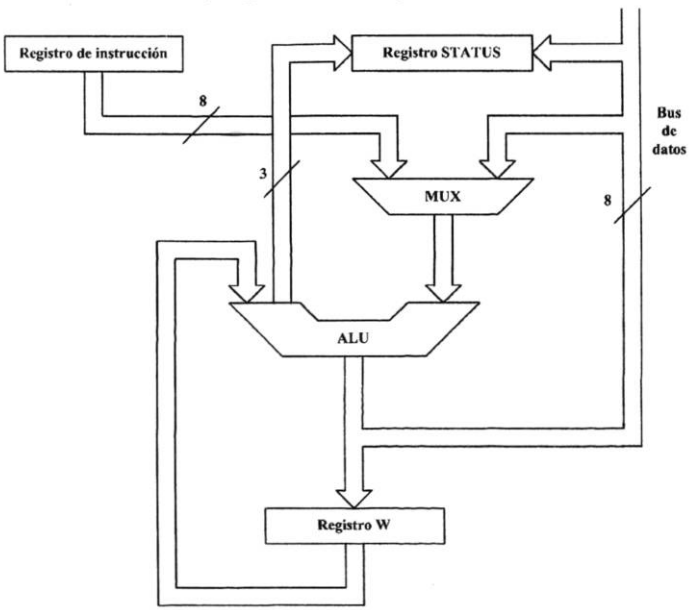


Figura II.12. Diagrama general de la ALU

En instrucciones con dos operandos, generalmente un operando está en el registro de trabajo (denominado registro W) y el otro es uno de los registros de propósito general o es una constante. Operando es el contenido sobre el cual se ejecuta una operación y un registro es cualquiera de los registros GPR o SFR, GPR es una abreviación de "General Purpose Register-Registro de Propósito General" y SFR es "Special Function Register-Registro de Función Especial".

En instrucciones de un solo operando, el operando es el contenido del registro W o uno de los registros de propósito general. El ALU también controla los bits de estado encontrados en el registro llamado STATUS. Dependiendo de la instrucción que se esté ejecutando, el ALU afecta los valores de los bits de acarreo-Carry (C), acarreo de dígito-Digit Carry (DC) y cero-Zero (Z) del registro de STATUS.

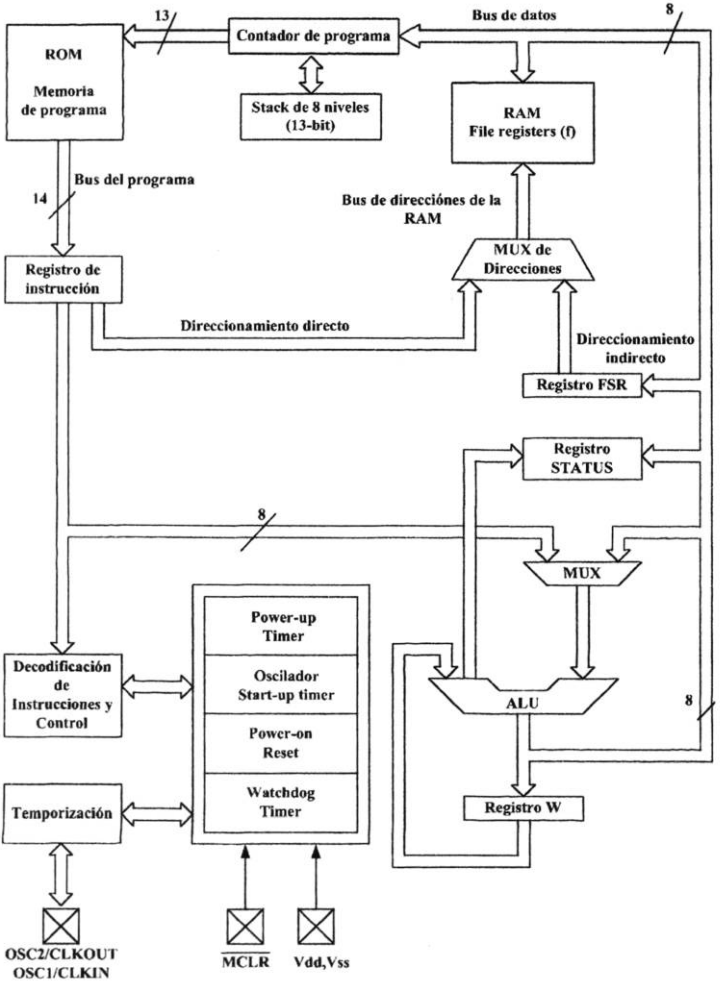


Figura II.13. Diagrama de bloques del PIC16F84

Registro STATUS

IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C

R = Bit que se puede leer. W = Bit que se puede escribir.
U = Bit no usado, se lee un 0.
-n = Valor inicial al encender la fuente de alimentación.

Bit 0-C (Acarreo-Carry)
Bit que es afectado por las operaciones de adición, sustracción y corrimiento.
1 = La transferencia del carry ocurrió del bit resultante de más alto orden.
0 = No ocurrió transferencia de carry.
El bit de carry es afectado por las instrucciones ADDWF, ADDLW, SUBLW y SUBWF.

Bit 1-DC (Acarreo de digito-Digit Carry)
Bit afectado por las operaciones de adición, sustracción y corrimiento. A diferencia del bit C, este bit representa la transferencia del bit 3 al bit 4. Es puesto a 1 en la adición cuando ocurre un acarreo del bit 3 al bit 4, o por la sustracción del bit 4 al bit 3 o por los corrimientos en ambas direcciones.
1 = La transferencia ocurrió en el cuarto bit de acuerdo al resultado.
0 = No ocurrió transferencia.
DC es afectado por las instrucciones ADDWF, ADDLW, SUBLW y SUBWF.

Bit 2-Z (Cero-Zero)
Este bit es puesto a 1 cuando el resultado de una operación aritmética o lógica es cero.
1 = El resultado es cero.
0 = El resultado es diferente de cero.

Bit 3-$\overline{\text{PD}}$ (Power-down)
Este bit es puesto a 1 lógico al suministrar energía o encender la fuente de alimentación, después de cada reset regular y después de ejecutar la instrucción CLRWDT. La instrucción SLEEP lo limpia cuando el microcontrolador pasa al modo de bajo consumo de energía. Colocarlo en uno también es posible al ocurrir una señal en la línea RB0/INT, al completarse una escritura en la EEPROM interna y por un watchdog también.
1 = Después de encender la fuente de alimentación.
0 = Ejecutando la instrucción SLEEP.

Bit 4-$\overline{\text{TO}}$ (Time-out, sobreflujo del watchdog timer)
Este bit es puesto a 1 lógico después de encender la fuente de alimentación y ejecutar las instrucciones CLRWDT y SLEEP. Es limpiado cuando el watchdog indica que algo no está correcto.
1 = No ocurrió sobreflujo.
0 = Ocurrió sobreflujo.

Bits 6 y 5-RP1:RP0 (Bits de selección banco de registros)
Estos dos bits son la parte superior de la dirección para el modo de direccionamiento directo. Ya que las instrucciones que direccionan la memoria directamente solo tienen siete bits, es necesario un bit mas para direccionar los 256 bytes del PIC16F84. RP1 no se usa pero está para expansiones posteriores de este microcontrolador.
01 = Banco uno.
00 = Banco cero.

Bit 7-IRP (Bit de selección banco de registros)
El papel de este bit es ser el octavo bit en el modo de direccionamiento indirecto de la memoria RAM interna.
1 = Banco 2 y 3.
0 = Banco 0 y 1 (de 00h a FFh).

UNIDAD III

ORGANIZACION DE LA MEMORIA

El PIC16F84 tiene dos bloques de memoria separados, uno de datos y el otro de programa. La memoria EEPROM y los registros GPR (registros de propósito general) localizados en la memoria RAM constituyen el bloque de datos, mientras que la memoria FLASH es el bloque de programa.

III.1. Memoria de programa

La memoria de programa está hecha con tecnología FLASH, lo cual permite programar al microcontrolador una gran cantidad de veces antes de instalarlo en un dispositivo y aún después de haberlo instalado si ocurren cambios eventuales en el programa o parámetros de procesos. El tamaño de la memoria de programa es de 1024 localidades de 14 bits cada una, donde las localidades cero y cuatro están reservadas para el reset y el vector de interrupción, respectivamente.

III.2. Memoria de datos

La memoria de datos está compuesta de la memoria EEPROM y la memoria RAM. La memoria EEPROM consiste de 64 localidades de 8 bits cada una cuyo contenido no se pierde al apagar la fuente de alimentación. La memoria EEPROM no es direccionable directamente sino que se accesa por medio de los registros EEADR y EEDATA. Ya que la memoria EEPROM sirve para almacenar parámetros importantes (por ejemplo, la temperatura establecida por un regulador), existe un

procedimiento estricto para escribir en la memoria EEPROM, el cual debe ser seguido para evitar escrituras erróneas o accidentales.

La memoria RAM ocupa espacio dentro del mapa de memoria de datos usando de la localidad 0x0C a la 0x4F (68 localidades). Las localidades de la memoria RAM se llaman también registros GPR lo cual es una abreviación de General Purpose Registers. Los registros GPR se pueden accesar sin importar el banco seleccionado en ese momento.

III.3. Registros

Registros SFR

Los registros SFR (Special Function Registers-registros de funciones especiales) ocupan las primeras 12 localidades en los bancos 0 y 1 de la memoria de datos.

III.4. Tipos de memoria

En la siguiente figura se muestra a bloques el mapa de memoria y los diferentes tipos de memoria con los que cuenta el PIC16F84.

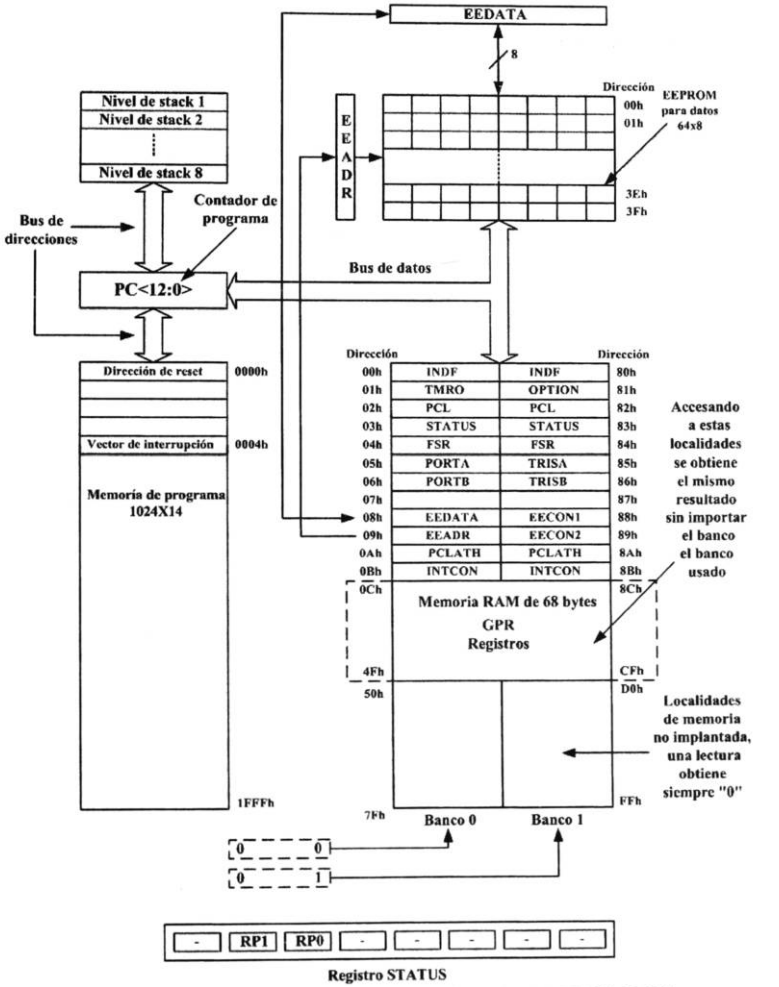


Figura III.1. Organización de memoria del PIC16F84

Bancos de memoria

En el espacio de memoria se observa una línea vertical y una línea horizontal en la memoria de datos. La línea horizontal divide los registros SFR de los registros GPR. La línea vertical separa dos áreas llamadas bancos. La selección de uno de los dos bancos se hace por medio de los bits RP0 y RP1 del registro STATUS.

Ejemplo:

```
bcf    STATUS,RP0
```

La instrucción bcf limpia el bit RP0 (RP0=0) del registro STATUS seleccionando el banco 0.

```
bsf    STATUS,RP0
```

La instrucción bsf establece a 1 lógico el bit RP0 (RP0=1) del registro STATUS seleccionando el banco 1.

Usualmente, se agrupan las instrucciones más comúnmente usadas en unidades llamadas macros y que fácilmente se pueden llamar o invocar en un programa, donde el nombre de esos grupos tiene un significado claro y relacionado con la función del grupo. Por ejemplo, se puede escribir un macro para seleccionar uno de los dos bancos y hacer el programa más legible.

```
BANK0    macro
         bcf   STATUS, RP0  ;Selecciona el banco 0
         endm

BANK1    macro
         bsf   STATUS, RP0  ;Selecciona el banco 1
         endm
```

Como ya se dijo anteriormente, las localidades 0Ch a la 4Fh son los registros de propósito general (GPR) los cuales pueden ser usados como memoria RAM. Cuando las localidades 8Ch a la CFh del banco 1 son accesadas, se estarán accesando exactamente las mismas localidades del banco 0. En otras palabras, si se desea accesar uno de los registros GPR, no importará el banco que esté seleccionado o en el cual se esté posicionado.

Program counter

El program counter (contador de programa-PC) es un registro de 13 bits que contiene la dirección de la siguiente instrucción a ejecutar. Este registro se incrementa o cambia (por ejemplo en caso de una instrucción de salto) cuando el microcontrolador ejecuta las instrucciones del programa paso-a-paso.

Stack

El PIC16F84 tiene un stack de 13 bits de 8 niveles, o en otras palabras, es un grupo de 8 localidades de memoria de 13 bits cada una. El papel del stack es almacenar el valor del program counter después de un salto en el programa principal a la dirección de un subprograma (por ejemplo con una instrucción CALL) y poder retornar después al punto donde se interrumpió el programa principal (por ejemplo ejecutando las instrucciones RETURN, RETLW o RETFIE). Las operaciones de colocar y tomar del stack el program counter se llaman PUSH y POP, respectivamente y como tal no están implantadas como instrucciones en el PIC16F84.

UNIDAD IV

PROGRAMACION

Programación del microcontrolador

Para grabar la memoria de programa el microcontrolador debe llevarse a un modo especial de trabajo suministrando 13.5V a la línea MCLR y el voltaje de la línea Vdd debe estabilizarse ente 4.5V y 5.5V. La memoria de programa se puede grabar de manera serial por medio de las líneas data y clock las cuales deben separarse previamente de las líneas del dispositivo para evitar errores durante la programación.

IV.1. Modos de direccionamiento

Las localidades de la memoria RAM se pueden accesar directamente o indirectamente.

Direccionamiento directo

El modo de direccionamiento directo se realiza por medio de direcciones de nueve bits. La dirección del operando se obtiene uniendo los siete bits de una dirección directa indicada en el código de la instrucción, con los bits RP0 y RP1 del registro STATUS, tal como se muestra en la siguiente figura. Cualquier acceso a los registros F (file registers) es un ejemplo de direccionamiento directo. Un file register es cualquier registro SFR o GPR.

```
bsf     STATUS,RP0 ;Selecciona el banco 1.
    movlw   0xFF        ;Carga en W un 0xFF.
    movwf   TRISA       ;La dirección del registro TRISA
                        ; es tomada del código de la
                        ;instrucción movwf.
```

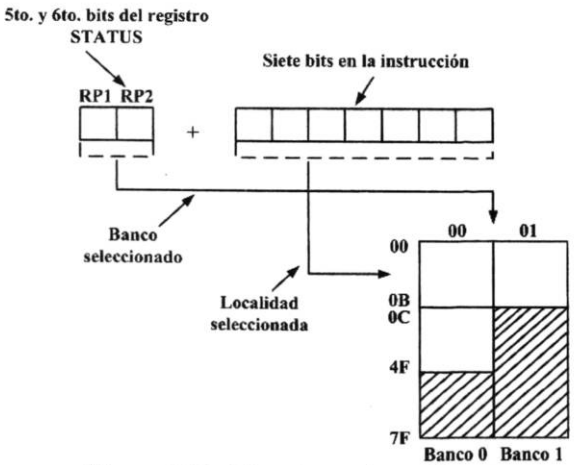


Figura IV.1. Direccionamiento directo

Direccionamiento indirecto

El direccionamiento indirecto, a diferencia del directo, no toma una dirección del código de una instrucción, sino que la construye con la ayuda del bit IRP del registro STATUS y del registro FSR. La localidad direccionada es accesada vía el registro INDF el cual contiene, en efecto, el dato de la dirección indicada por FSR. En otras palabras, cualquier instrucción que use al registro INDF como su registro en realidad estará accesando el dato de la localidad indicada por un registro FSR. Por ejemplo, si un registro de propósito general (GPR) en la dirección 0Fh contiene un valor de 20, al escribir un valor de 0Fh en el registro FSR y leer el registro INDF se obtendrá el valor de 20, lo cual indica que se estará leyendo el GPR sin accesarlo directamente (pero si vía FSR e INDF). Pareciera que este tipo de direccionamiento no tiene ventajas sobre el direccionamiento directo pero ciertas aplicaciones se resuelven más fácil y eficientemente usando direccionamiento indirecto. El registro INDF contiene el dato leído o escrito y el registro FSR la dirección de la localidad de memoria a leer o escribir.

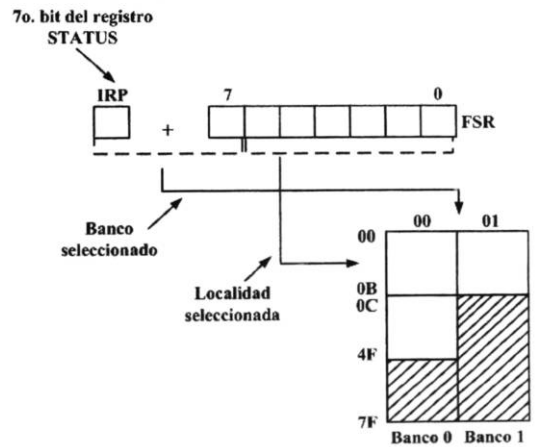


Figura IV.2. Direccionamiento indirecto

Un ejemplo de direccionamiento indirecto es el siguiente, donde se envía un grupo de datos por comunicación serie trabajando con buffer e índices, borrando una parte de la memoria RAM (16 localidades):

```
        movlw   0x0C    ;Establece dirección de inicio.
        movwf   FSR     ;FSR apunta a la dirección 0x0C.
LOOP    clrf  INDF      ;INDF=0.
        incf FSR        :dirección = dirección inicial + 1.
        btfss FSR,4     ;Están limpias todas las locs.?
        goto LOOP       ;No, regresa al ciclo.
        ....            ;Si, continua con el programa.
```

Una lectura del registro INDF cuando el contenido del registro FSR es cero regresa el valor de cero y una lectura del registro INDF cuando FSR es cero trae como resultado una operación NOP (no operation).

IV.2. Estructura general de un programa

Físicamente, un programa representa un archivo que se encuentra en el disco de la computadora (o en la memoria, si es leído de un microcontrolador) y se escribe de acuerdo a las reglas del ensamblador o algún otro lenguaje que consiste de signos alfabéticos y palabras. Al escribir el programa se deben seguir esas reglas para que un programa intérprete convierta cada instrucción como una serie de ceros y unos que tenga un significado para la lógica interna del microcontrolador.
La conversión se encuentra en un archivo ejecutable y en un archivo con la extensión .HEX, donde .HEX significa hexadecimal, el cual posteriormente se graba en el microcontrolador para su ejecución.
El programa fuente, en lenguaje ensamblador, se hace en un editor de texto y contiene los siguientes elementos básicos:

- Etiquetas.
- Instrucciones.
- Operandos.
- Directivas.
- Comentarios.

Etiquetas

Una etiqueta es una designación textual (generalmente una palabra fácil de leer) para una línea del programa, o sección de un programa a donde puede saltar el microcontrolador o bien el inicio de un conjunto de líneas de un programa. Una etiqueta inicia con una letra del alfabeto o con un caracter underline "_". La longitud de la etiqueta normalmente es de 32 caracteres máximo e inicia en la primera columna.

Instrucciones

Las instrucciones ya están definidas por el microcontrolador específico a usar, de manera tal que solo resta seguir su uso en el lenguaje ensamblador. La forma de escribir las instrucciones se le llama sintaxis de la instrucción. En el ejemplo siguiente se puede reconocer un error de escritura donde las instrucciones **movlp** y **gotto NO** existen para el microcontrolador PIC16F84.

Instrucciones escritas correctamente

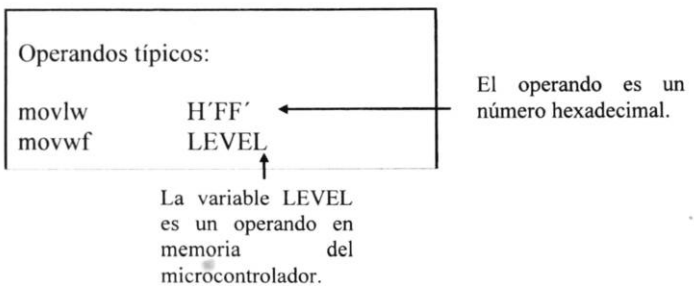
```
movlw   H'01FF'
goto    Start
```

Instrucciones escritas incorrectamente

```
movlp   H'01FF'
gotto   Start
```

Operandos

Los operandos son elementos de las instrucciones que necesitan éstas para poderse ejecutar. Usualmente los operandos son registros, variables o constantes.



Comentarios

Un comentario es una serie de palabras que escribe el programador para hacer más claro y legible el programa. Los comentarios comúnmente se colocan después de una instrucción e inician con un punto y coma ";".

Directivas

Una directiva es similar a una instrucción, pero a diferencia de una instrucción, la directiva es independiente del modelo del microcontrolador y representa una característica del lenguaje ensamblador mismo. Las directivas se usan para dar un significado poderoso a variables o registros. Por ejemplo, el nombre **NIVEL** se puede usar para designar una variable en la localidad de memoria RAM 0Dh. De esta forma es más fácil para el programador entender o recordar

que la localidad de memoria 0Dh contiene información acerca del **NIVEL**. Las directivas que a continuación se toman como ejemplo pertenecen al ensamblador MPASM de Microchip.

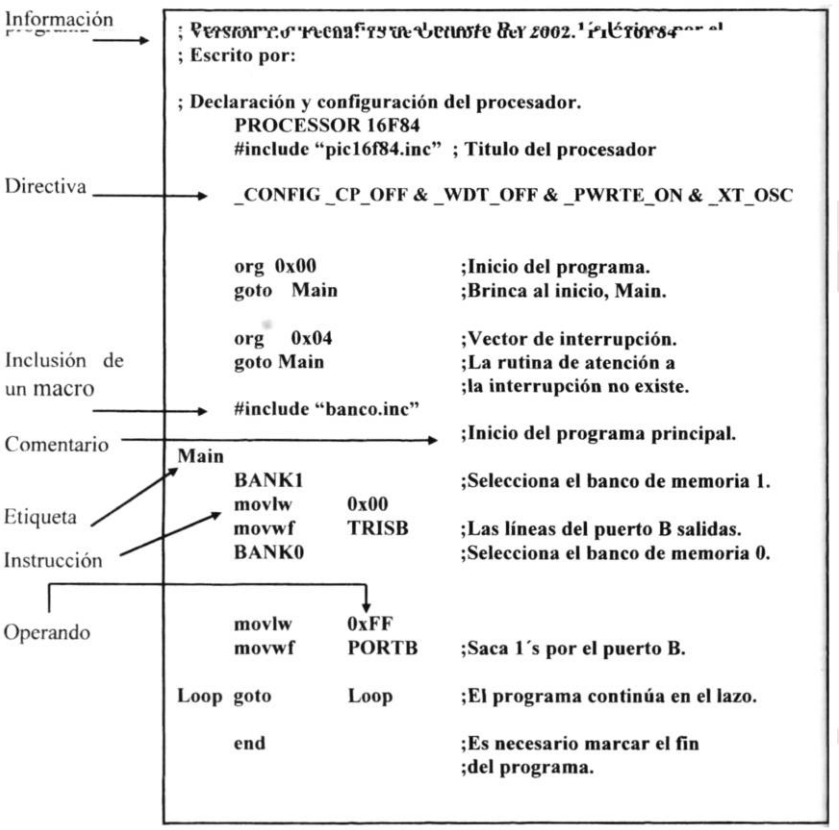
Algunas directivas más
frecuentemente usadas:

```
PROCESSOR 16F84
#include "p16f84.inc"

__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
```

Al escribir un programa, existen reglas obligatorias o estrictas y reglas que no son especificadas pero que es recomendable seguirlas. Algunas de estas reglas son las siguientes: escribir al inicio del mismo el nombre del programa, lo que hace el programa, la versión, la fecha cuando fue escrito, el tipo de microcontrolador a usar y el nombre del programador.

El ejemplo siguiente muestra un programa escrito en lenguaje ensamblador respetando las reglas básicas anteriores.



En el programa anterior se puede observar que después de los comentarios iniciales se definen varios parámetros importantes del microcontrolador como por ejemplo, el tipo del oscilador, el encendido/apagado del watchdog timer y la habilitación/deshabilitación del circuito interno de reloj, con la directiva siguiente:

__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

Cuando todos los elementos necesarios han sido definidos, se puede empezar a escribir un programa. Primero, es necesario determinar una dirección desde la cual inicia el microcontrolador al encender la fuente de alimentación (org 0x00). A continuación definir la dirección de inicio de la rutina de atención a la interrupción (org 0x04) y posteriormente iniciar el programa principal.

IV.3. Conjunto de instrucciones

El conjunto de instrucciones del PIC16F84 incluye 35 instrucciones ya que se trata de un microcontrolador RISC cuyas instrucciones han sido optimizadas considerando la velocidad de trabajo, arquitectura simple y código compacto.

Las instrucciones del PIC16F84 están clasificadas de la siguiente manera:

- Instrucciones de transferencia de datos.
- Instrucciones aritméticas y lógicas.
- Instrucciones de manejo de bits.
- Instrucciones de transferencia de control.
- Instrucciones especiales.

La transferencia de datos en el PIC16F84 se hace usando el registro de trabajo (W) y un registro f (file register) el cual representa cualquier localidad de la RAM interna (independientemente de que se trate de un SFR o un GPR).

f cualquier localidad de memoria del microcontrolador.
W el registro de trabajo W.
b posición de un bit en el registro "f".
d bit destino.

Mnemónico	Descripción	Operación	Banderas afectadas	Ciclos	Observaciones
MOVLW k	Mueve la constante k a W.	k → W		1	
MOVWF f	Mueve W a f.	W → f		1	
MOVF f,d	Mueve f.	f → d	Z	1	1,2
CLRW	Limpia W.	0 → W	Z	1	
CLRF f	Limpia f.	0 → f	Z	1	2
SWAPF f,d	Intercambia los nibbles de f.	f(7:4),(3:0) →f(3:0),(7:4)		1	1,2

Notas:

1-Si el operando fuente es un puerto, se lee el estado de las líneas del microcontrolador.

2-Si se ejecuta esta instrucción sobre el registro TMR0 y d=1, el prescalar asignado al timer automáticamente se limpia.

Las tres primeras instrucciones de la tabla anterior realizan las siguientes acciones: escribir una constante en el registro W (MOVLW significa MOVe Literal to W), copia un dato del registro W en la RAM y copia un dato de la RAM al registro W (o en la misma localidad de la RAM, en cuyo caso solo cambia el estado de la bandera Z). La instrucción CLRF escribe la constante 00h en el registro f, mientras que la instrucción

CLRW escribe la constante 00h en el registro W. La instrucción SWAPF intercambia de lugar los dos nibbles de 4 bits de un registro.

Instrucciones aritméticas y lógicas

De todas las operaciones matemáticas, el PIC16F84, como muchos microcontroladores, únicamente soporta la sustracción y la adición. Se afectan las banderas C, DC y Z de acuerdo a la operación realizada, con una sola excepción: ya que la sustracción se realiza como una adición de un valor negativo, la bandera C toma un valor inverso después de una sustracción. En otras palabras, esta bandera se activa si la operación es posible y se limpia cuando es sustraído un número grande de uno más pequeño.

Mnemónico	Descripción	Operación	Banderas afectadas	Ciclos	Observaciones
ADDLW k	Adiciona una constante y W.	W + k → W	C,DC,Z	1	
ADDWF f,d	Adiciona W y f.	W + f → d	C,DC,Z	1	1,2
SUBLW k	Sustrae W de una constante.	k - W → W	C,DC,Z	1	
SUBWF f,d	Sustrae W de f.	f - W → W	C,DC,Z	1	1,2
ANDLW k	AND de constante con W.	W AND k → W	Z	1	
ANDWF f,d	AND de W con f.	W AND f → d	Z	1	1,2
IORLW k	OR de constante con W.	W OR k → W	Z	1	
IORWF f,d	OR de W con f.	W OR f → d	Z	1	1,2
XORLW k	OR Exclusiva de constante con W.	W XOR k → W	Z	1	1,2
XORWF f,d	OR Exclusiva de W con f.	W XOR f → d	Z	1	
INCF f,d	Incrementa f.	f + 1 → f	Z	1	1,2
DECF f,d	Decrementa f.	f - 1 → f	Z	1	1,2
RLF f,d	Rota a la izquierda f a través del carry.		C	1	1,2
RRF f,d	Rota a la derecha f a través del carry.		C	1	1,2
COMF f,d	Complementa f.	f → d	Z	1	1,2

Notas:

1-Si el operando fuente es un puerto, se lee el estado de las líneas del microcontrolador.

2-Si se ejecuta esta instrucción sobre el registro TMR0 y d=1, el prescalar asignado al timer automáticamente se limpia.

La unidad lógica del PIC tiene la capacidad de realizar la operaciones de AND, OR, EXOR, complemento (COMF) y rotación (RLF y RRF).

Las instrucciones de rotación mueven los bits del registro por medio de la bandera de carry (C) un espacio a la izquierda o a la derecha. El bit que sale del registro se escribe en la bandera C y el valor de la bandera C se escribe en el bit d el lado opuesto del registro.

Instrucciones de manejo de bits

Las instrucciones BCF y BSF establecen a 1 lógico o a 0 lógico, respectivamente, un bit. Aunque esta es una operación muy simple, cuando la CPU la ejecuta primero lee el byte de la localidad de memoria, cambia el bit y después escribe el byte en el mismo lugar.

Mnemónico	Descripción	Operación	Banderas afectadas	Ciclos	Observaciones
BCF f,b	Limpia el bit b de f.	0 → f(b)		1	1.2
BSF f,b	Establece a 1 lógico el bit b de f.	1 → f(b)		1	1.2

Notas:

1-Si el operando fuente es un puerto, se lee el estado de las líneas del microcontrolador.

2-Si se ejecuta esta instrucción sobre el registro TMR0 y d=1, el prescalar asignado al timer automáticamente se limpia.

Las instrucciones GOTO, CALL y RETURN se ejecutan de manera similar que en otros microcontroladores, únicamente que el stack es independiente de la memoria RAM, fuera del alcance del programador y limitado a ocho niveles. La instrucción RETLW k es idéntica que la instrucción RETURN, excepto que antes de regresar de un subprograma se escribe en el registro W una constante definida por el operando de la instrucción. Esta instrucción permite diseñar fácilmente tablas de búsqueda (listas). Se usa más comúnmente para determinar la posición en una tabla adicionándole a la dirección de inicio de la tabla la constante definida por la instrucción y leer el dato de esa localidad (la cual se encuentra usualmente en memoria de programa).

La tabla se puede formar como un subprograma que consiste de una serie de instrucciones RETLW k, donde las constantes k son miembros de la tabla.

```
Main    movlw   2
                call    Lookup

        Lookup  addwf   PCL,f
                retlw   k
                retlw   k1
                retlw   k2
                :
                :
                retlw   kn
```

En el segmento anterior de un programa se debe escribir la posición de un miembro de la tabla en el registro W, y usando la instrucción CALL se llama a un subprograma que crea la tabla. La primera línea del subprograma, ADDWF PCL,f, suma la posición de un miembro,

almacenada en el registro W, a la dirección de inicio de la tabla encontrada en el registro PCL para encontrar la dirección real del dato localizado en memoria de programa. Al regresar del subprograma se tendrá en el registro W el contenido de un miembro de la tabla direccionada. En el ejemplo anterior, después de ejecutar la instrucción retlw se encontrará en el registro W la constante k2.

Mnemónico	Descripción	Operación	Banderas afectadas	Ciclos	Observaciones
BTFSC f,b	Prueba el bit b de f y salta si es 0 lógico.	Salta si f(b)=0		1(2)	3
BTFSS f,b	Prueba el bit b de f y salta si es 1 lógico.	Salta si f(b)=1		1(2)	3
DECFSZ f,d	Decrementa f y salta si es 0 lógico.	f - 1 → d, salta si Z=1		1(2)	1,2,3
INCFSZ f,d	Incrementa f y salta si es 0 lógico.	f + 1 → d, salta si Z=1		1(2)	1,2,3
GOTO k	Salta a la dirección o etiqueta k.			2	
CALL k	Llama a una subrutina k.			2	
RETURN	Regresa de una subrutina.			2	
RETLW k	Regresa con una constante en W.			2	
RETFIE	Regresa de una interrupción.			2	

Notas:

1-Si el operando fuente es un puerto, se lee el estado de las líneas del microcontrolador.

2-Si se ejecuta esta instrucción sobre el registro TMR0 y d=1, el prescalar asignado al timer automáticamente se limpia.

3-Si se modifica el PC o el resultado de la prueba es 1 lógico, la instrucción se ejecuta en dos ciclos.

La instrucción RETFIE (RETurn From Interrupt and Interrupt Enable) sirve para regresar de una rutina de una interrupción y difiere de la instrucción RETURN solo en que automáticamente establece a 1 lógico el bit GIE (Global Interrupt Enable). Cuando sucede una interrupción se limpia este bit y solo el valor del program counter se coloca en el tope del stack.

Los saltos condicionales se resumen en dos instrucciones: BTFSC y BTFSS. Dependiendo del bit de que se esté probando del registro f, se ejecutan o no las instrucciones siguientes al BTFSC o BTFSC.

Instrucciones especiales

Mnemónico	Descripción	Operación	Banderas afectadas	Ciclos	Observaciones
NOP	No operation.			1	
CLRWDT	Limpia el Watchdog Timer.	0 → WDT, 1 → TO, 1 → PD	TO, PD	1	
SLEEP	Pasa al modo standby.	0 → WDT, 1 → TO, 0 → PD	TO, PD	1	

IV.4. Periodo de ejecución de las instrucciones

Todas las instrucciones se ejecutan en un ciclo de máquina, excepto las instrucciones de salto condicional que se ejecutan en dos ciclos de máquina si la condición se cumple, o si el contenido del program counter es cambiado por alguna instrucción. En ese caso, la ejecución requiere dos ciclos y durante el segundo ciclo de instrucción se ejecuta una instrucción NOP (No Operation). Un ciclo de instrucción está compuesto de cuatro pulsos de la señal de reloj, por lo que si la frecuencia del oscilador para dicha señal es de 4Mhz, el tiempo para ejecutar una instrucción es de 1 μs, y en caso de saltos condicionales, el periodo de ejecución es de 2 microsegundos.

IV.5. Archivos creados al ensamblar un programa

Como resultado del proceso de ensamblado de un programa se obtienen los siguientes archivos:

- Archivo ejecutable en formato Intel (Nombre_del_programa.HEX).
- Archivo de errores del programa (Nombre_del_programa.ERR).
- Archivo de listado del programa (Nombre_del_programa.LST).

El primer archivo contiene el programa ensamblado y que será grabado en el microcontrolador. El segundo archivo contiene los posibles errores al escribir el programa y que fueron detectados por el ensamblador. Los errores se pueden ver también en el archivo de listado del programa, lo cual es muy útil en programas grandes.

El tercer archivo es el más útil para el programador, ya que contiene mucha información acerca de la ubicación de instrucciones y variables en memoria o señalización de errores, en este archivo se muestra, normalmente, en la parte superior de cada página el nombre del archivo, la fecha cuando fue ensamblado y el número de página. Este archivo se divide en varias columnas, de las cuales la primera indica la dirección en memoria de programa donde se almacena la instrucción de la línea correspondiente del programa. La segunda columna contiene el valor de las variables definidas por las directivas: SET, EQU, VARIABLE, CONSTANT o CBLOCK. La tercera columna está reservada para el ensamblador y la cuarta columna contiene las instrucciones y comentarios del programa. Los errores posibles aparecerán después de la línea donde ocurrió el error.

UNIDAD V

TEMPORIZADORES Y PUERTOS DE E/S

V.1. Temporizadores y contadores

Características generales

El free-run timer TMR0

Físicamente el timer es un registro cuyo valor se incrementa continuamente hasta 255 y a continuación vuelve a arrancar: 0, 1, 2, 3, 4...255...0, 1, 2, 3.....etcétera. .

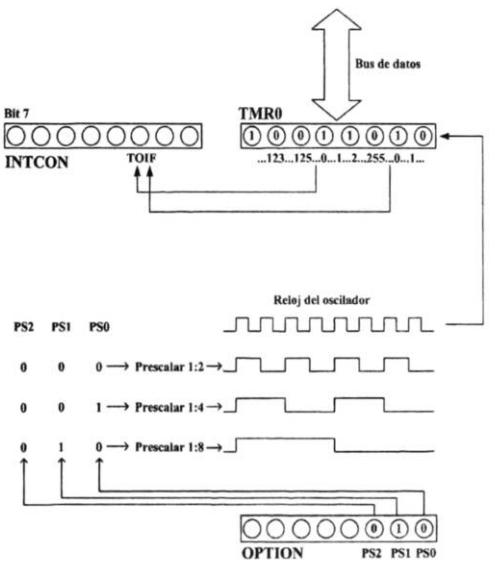


Figura V.1. Relación entre el timer TMR0 y el prescalar

Este incremento se hace en segundo plano (background) de todo lo que hace el microcontrolador. Una de las formas como el programador puede usar el timer es incrementando también una variable cada vez que ocurra un sobreflujo en el timer. Si se conoce cuanto tiempo necesita un timer para llegar de 0 a 255, entonces multiplicando el valor de una variable por ese tiempo se puede obtener el tiempo transcurrido.

Funcionamiento y modos de operación

El PIC16F84 tiene un timer de 8 bits. El número de bits determina hasta que valor contará el timer antes de iniciar el conteo desde cero nuevamente. En el caso de un timer de 8 bits, ese número es 255. En el diagrama anterior se muestra la relación existente entre un timer y un prescalar.
Prescalar es el nombre dado a la parte del microcontrolador que divide el reloj del oscilador antes de que pase a la lógica que incrementa el estado del timer. El número que divide al reloj se define por medio de los primeros tres bits del registro OPTION. El divisor más grande es 256. Esto significa que cada 256 pulsos de reloj se incrementa en uno el valor del timer. Esto es de gran ayuda en casos de necesitar medir periodos de tiempo grandes.

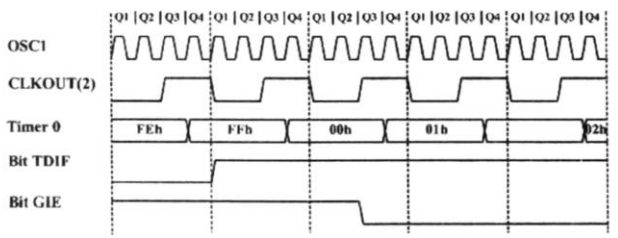


Figura V.2. Interrupción del timer TMR0

Después de cada conteo hasta 255 el timer reinicia su valor a cero y arranca un nuevo ciclo de conteo hasta 255. Durante cada transición de 255 a cero se activa el bit TOIF del registro INTCON. En al rutina de atención a la interrupción debe limpiarse el bit TOIF para que ocurra una

nueva interrupción o que se pueda detectar un nuevo sobreflujo. Además del reloj del oscilador interno, se puede usar un reloj externo por medio de la línea RA4/TOCKI para incrementar el estado del timer. Elegir una de estas dos opciones se hace por medio del bit T0CS del registro OPTION. Además, la opción del reloj externo, se puede definir el tipo de transición (subida o bajada) por medio de la cual el timer incrementará su valor.

V.2. Programación de los temporizadores y contadores

El registro OPTION

RBPU	INTEDG	T0CS	T0S	PSA	PS2	PS1	PS0

R = Bit que se puede leer. W = Bit que se puede escribir.
U = Bit no usado, se lee un 0.
-n = Valor inicial al encender la fuente de alimentación.

Bits 0, 1 y 2-PS0, PS1 y PS2 (Prescaler Rate Select Bit-Bits de selección del prescalar)
Estos tres bits definen el valor del prescalar del Timer 0 (TMR0) y del watchdog.

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

Bit 3-PSA (Prescaler Assignment Bit-Bit de asignación del prescalar)
Este bit asigna el prescalar al Timer 0-TMR0 o al watchdog.
1 = El prescalar es asignado al watchdog.
0 = El prescalar es asignado al free-run timer TMR0.

Bit 4-T0SE (TMR0 Source Edge Select Bit- Bit de selección del flanco o transición de la fuente de TMR0)
Se puede disparar TMR0 por medio de un pulso en la línea RA4/T0CKI, este bit determina si será con la transición de bajada o de subida.
1 = transición de bajada.
0 = transición de subida.

Bit 5-T0CS (TMR0 Clock Source Select Bit-Bit de selección del reloj de TMR0)
Este bit indica si el free-run timer incrementará su estado usando el oscilador interno cada 4 pulsos de la señal de reloj o por medio de pulsos externos en la línea RA4/T0CK1.
1 = Pulsos externos (contador).
0 = ¼ del reloj interno (temporizador).

Bit 6-INTEDG (Interrupt Edge Select Bit-Bit de selección del flanco o transición que dispara la interrupción)
Si la interrupción esta habilitada con este bit, es posible determinar el flanco o la transición con la que será activada la interrupción por medio de la línea RB0/INT.
1 = Transición de subida.
0 = Transición de bajada.

Bit 7-RBPU (PORTB Pull-up Enable Bit-Bit de habilitación de las resistencias de pull-up del puerto B)
Este bit activa o desactiva las resistencias pull-up internas del puerto B.
1 = Se desactivan las resistencias de pull-up.
0 = Se activan las resistencias de pull-up.

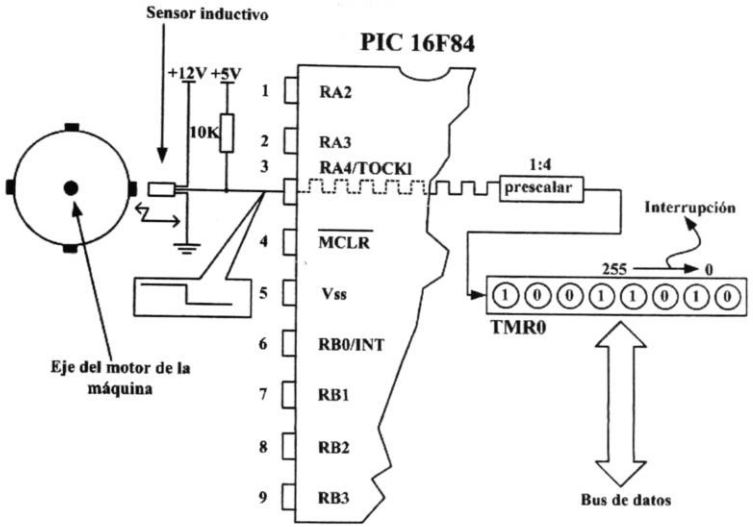


Figura V.3. Uso del timer para determinar el número de vueltas de un motor

En la figura anterior se muestra un ejemplo típico del uso del temporizador, donde se cuenta el número de vueltas completas que da un eje del motor de una máquina usando un reloj externo y el temporizador. Supóngase que se tiene un sensor inductivo a una distancia de 5 mm de la cabeza de los tornillos. El sensor inductivo generará una señal de bajada cada vez que la cabeza del tornillo esté paralela con la cabeza del sensor. Cada señal representará un cuarto de una vuelta completa y la suma de todas las vueltas completas se encontrará en el timer TMR0. El programa puede fácilmente leer este dato del temporizador por medio de un bus de datos.

A continuación se encuentra un programa que inicializa al temporizador para que se incremente con transiciones de bajada de la señal de reloj externa usando un prescalar de 1:4.

```
        clrf  TMR0          ;TMR0=0.
        clrf  INTCON        ;Interrupciones y TOIF=0
                            ;deshabilitadas.
        bsf   STATUS,RP0    ;Registro OPTION en banco 1.
        movlw B'00110001'   ;Prescalar 1:4, trans. de bajada,
                            ;reloj externo y pull up's del
                            ;puerto B activados.
        movwf OPTION_REG    ;OPTION_REG <- W.

T0_OVL
        btfss INTCON,TOIF   ;Prueba bit de sobreflujo.
        goto T0_OVL         ;No ocurrió interrup., espera.
;
; (Parte del programa que procesa datos de num. de vueltas).
;
        goto T0_OVL         ;Espera nuevo sobreflujo.
```

Este mismo ejemplo se puede realizar usando una interrupción de la manera siguiente:

```
push    macro
        movwf   W_Temp          ;W_Temp <- W.
        swapf   W_Temp,F        ;Intercámbialos.
        BANK1                   ;Cambio al banco 1.
        swapf   OPTION_REG,W    ;W <- OPTION_REG.
        movwf   Option_Temp     ;Option_Temp <- W.
        BANK0                   ;Cambio al banco 0.
        swapf   STATUS,W        ;W <- STATUS.
        movwf   Stat_Temp       ;Stat_Temp <- W.
        endm                    ;Fin del macro push.
```

```
pop     macro
        swapf   Stat_Temp,W     ;W <- Stat_Temp.
        movwf   STATUS          ;STATUS <- W.
        BANK1                   ;Cambio al banco 1.
        swapf   Option_Temp,W   ;W <- Option_Temp.
        movwf   OPTION_REG      ;OPTION_REG <- W.
        BANK0                   ;Cambio al banco 0.
        swapf   W_Temp,W        ;W <- W_Temp.
        endm                    ;Fin del macro pop.
```

El prescalar puede ser asignado al timer TMR0 o al watchdog. El watchdog es un mecanismo que usa el microcontrolador para poder salir por si mismo de programas mal realizados o cuando ocurre una falla eléctrica. Cuando esto sucede, el microcontrolador detiene su trabajo y permanece en ese estado hasta que alguien o algo lo reinicialice (reset).

Una forma de reinicializarlo es usando el watchdog después de un cierto periodo de tiempo. El watchdog trabaja usando un principio muy simple: si ocurre un sobreflujo en el timer, el microcontrolador se reinicializa y empieza a ejecutar su programa nuevamente. De esta forma, un reset ocurrirá en caso de funcionamiento correcto o incorrecto del microcontrolador, por lo que el siguiente paso es escribir un uno en el registro WDT (con la instrucción CLRWDT) cada vez que ocurra un sobreflujo para evitar un reset en caso de un correcto funcionamiento. De esta forma, se evitará un reset mientras el programa se esté ejecutando correctamente, y en casos de mal funcionamiento se escribirá un cero en WDT y el PIC16F84 se reinicializará.

El prescalar es asignado al timer TMR0 o al watchdog timer por medio del bit PSA del registro OPTION. Limpiando el bit PSA el prescalar será asignado al timer TMR0. Cuando el prescalar es asignado al timer TMR0, todas las instrucciones que escriban al registro de TMR0 (por ejemplo CLRF TMR0, MOVWF TMR0, BSF TMR0,...) limpiarán al

prescalar también. Cuando el prescalar es asignado al watchdog timer, solo la instrucción CLRWDT limpiará al prescalar y al watchdog timer al mismo tiempo. El cambio del prescalar está completamente bajo control del programador y puede cambiarse en cualquier momento mientras un programa se esté ejecutando.

Resumiendo entonces, existe un solo prescalar y un timer. Dependiendo de las necesidades, se asignan ya sea al timer TMR0 o al watchdog.

V.3. La memoria de datos EEPROM

El PIC16F84 tiene una EEPROM de 64 bytes que comprende las localidades 00h a la 63h. La característica más importante de esta memoria es que no pierde su contenido cuando se apaga su fuente de alimentación. Los datos pueden ser retenidos por la EEPROM aun sin fuente de alimentación hasta por 40 años (según el fabricante del PIC16F84) y se pueden ejecutar hasta 10 000 ciclos de escritura.

La memoria EEPROM sirve, en la práctica. Para almacenar datos importantes o parámetros de un proceso, como por ejemplo la temperatura o nivel que serán controlados o ajustados por un regulador o un cierto proceso.

La EEPROM está colocada en un espacio especial de la memoria y se puede accesar por medio de los siguientes registros especiales:

EEDATA localizado en la dirección 08h, contiene los datos leídos o escritos.

EEADR localizado en la dirección 09h, contiene la dirección de la localidad de EEPROM que será accesada.

EECON1 localizado en la dirección 88h, contiene bits de control.

EECON2 localizado en la dirección 89h. Este registro no existe físicamente y sirve para proteger a la EEPROM de escrituras accidentales.

El registro EECON1

Este registro es un registro de control localizado en la localidad de memoria 88h que contiene cinco bits funcionales. Los bits 5, 6 y 7 no se usan y leerlos dará como resultado cero.

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

R = Bit que se puede leer. W = Bit que se puede escribir.
U = Bit no usado, se lee un 0.
-n = Valor inicial al encender la fuente de alimentación.

Bit 0-RD (Read Control Bit-Bit de control de lectura)
Activando este bit se inicia la transferencia de datos desde la dirección definida en el registro EEADR al registro EEDATA.
1 = Inicia la lectura.
0 = No inicia la lectura.

Bit 1-WR (Write Control Bit-Bit de control de escritura)
Activando este bit se inicia la escritura de datos desde el registro EEDATA hacia la dirección especificada por el registro EEADR.
1 = Inicia la escritura.
0 = No inicia la escritura.

Bit 2-WREN (EEPROM Write Enable Bit-Bit de habilitación de escritura a EEPROM)
Si este bit no está activado, el microcontrolador no permite la escritura a EEPROM.
1 = Escritura permitida.
0 = Escritura no permitida.

Bit 3-WRERR (Write EEPROM Error Flag-Bandera de error de escritura a EEPROM)
Este bit se activa solo cuando una escritura a EEPROM ha sido interrumpida por una señal de reset o cuando ha expirado el watchdog timer (si es que se encuentra activado).
1 = Ocurrió error.
0 = No ocurrió error.

Bit 4-EEIF (EEPROM Write Operation Interrupt Flag Bit-Bandera de interrupción de escritura de la EEPROM)
Este bit se usa para informar que ha terminado una escritura a la EEPROM. El programador debe limpiar este bit para permitir o detectar la terminación de otras escrituras a EEPROM.
1 = Terminó una escritura.
0 = No ha terminado aún una escritura o no ha iniciado.

Lectura de la memoria EEPROM

Activando el bit RD se inicia una transferencia de datos de la localidad cuya dirección se encuentra en el registro EEADR al registro EEDATA. En las lecturas de datos no se necesita tanto tiempo como en las escrituras, de tal forma que el dato leído y que se encuentra en el registro EEDATA se puede usar en la instrucción siguiente.

A continuación se muestra el segmento de un programa usado para leer datos de la memoria EEPROM.

```
bcf     STATUS,RP0  ;EEADR esta en la 09h, banco 0.
movlw   0x00        ;Direcc. de la localidad a leer.
movwf   EEADR       :Direcc. transferida a EEADR.
bsf     STATUS,RP0  ;EECON1 esta en la 88h, banco1.
bsf     EECON1,RD   ;Lectura de la EEPROM.
bcf     STATUS,RP0  ;EEDATA esta en la 08h, banco 0.
movf    EEDATA,W    ;W ← EEDATA.
```

Después de la ultima instrucción, el contenido de la dirección cero de la EEPROM se encuentra en el registro de trabajo W.

Escritura a la memoria EEPROM

Para escribir datos a una localidad de la memoria EEPROM, el programador debe escribir primero la dirección en el registro EEADR y el dato en el registro EEDATA. A continuación debe activarse el bit WR para llevar a cabo la escritura. El bit WR será limpiado y el bit EEIF se activará al terminar la escritura y podrá generar así una interrupción. Los valores 55h y AAh son las dos claves que deshabilitan la característica del PIC16F84 que evita que ocurran escrituras accidentales a la EEPROM. Estos dos valores deben escribirse al registro EECON2 que sirve únicamente para este propósito. Las líneas marcadas en el siguiente programa como 1, 2, 3 y 4 deben ejecutarse en ese orden en intervalos iguales de tiempo, por lo que es de suma importancia deshabilitar las interrupciones ya que éstas podrían cambiar los intervalos de tiempo necesarios para ejecutar estas instrucciones. Después de estas instrucciones y escribir a la EEPROM se pueden habilitar nuevamente las interrupciones.

Es recomendable que el bit WREN esté apagado todo el tiempo excepto cuando se escriban datos a la EEPROM, de tal forma que la posibilidad de escribir accidentalmente a esta memoria sea mínima. Todas las escrituras a la EEPROM limpiarán automáticamente la localidad antes de escribirla.

A continuación se muestra el segmento de un programa que escribe el dato 0xEE a la primera localidad de la EEPROM.

```
    bcf     STATUS,RP0  ;EEADR esta en la 09h, banco 0.
    movlw   0x00        ;Direcc. de la localidad a escribir.
    movwf   EEADR       :Direcc. transferida a EEADR.
    movlw   0xEE        ;Escribe el dato 0xEE .
    movwf   EEDATA      :El dato pasa a EEDATA.
    bsf     STATUS,RP0  ;EECON1 esta en la 88h, banco1.
    bcf     INTCON,GIE  ;Deshabilita todas interrupciones.
    bsf     EECON1,WREN ;Habilita escritura a EEPROM.
    movlw   0x55
1)  movwf   EECON2      ;Primera clave 0x55 -> EECON2.
2)  movlw   0xAA
3)  movwf   EECON2      ;Segunda clave
                        ;  0xAA-> EECON2.
4)  bsf     EECON1,WR   ;Inicia escritura.
    bsf     INTCON,GIE  ;Habilita interrupciones.
```

V.4. Puertos de entrada/salida

Características generales

Como un puerto nos referiremos a un grupo de líneas o pines del microcontrolador que pueden ser accesados simultáneamente, o en los que se puede establecer una combinación deseada de unos y ceros o leer por medio de ellos un cierto estado. Físicamente un puerto es un registro dentro del microcontrolador el cual está conectado a las líneas del mismo. Un puerto representa una conexión física de la CPU con el mundo externo. El microcontrolador usa los puertos para monitorear o controlar otros componentes o dispositivos. Algunas líneas tienen dos funciones, como por el ejemplo la línea PA4/TOCKI, el cual simultáneamente es la línea 5 del puerto A y puede operar como una línea de entrada para el contador (free-run counter). La selección de una de las dos funciones se realiza en uno de los registros de configuración de los puertos de entrada/salida.

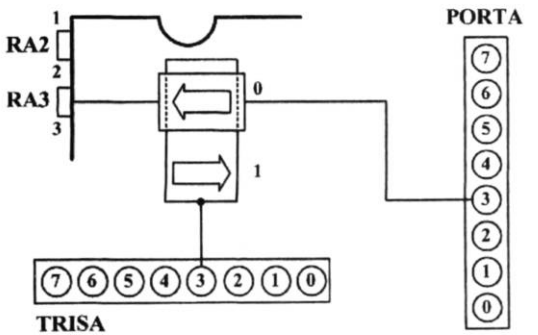


Figura V.4. Relación entre los registros TRISA y PORTA

Funcionamiento y modos de operación

Por medio de los registros TRIS del PIC16F84 se pueden definir las líneas de los puertos como entradas o como salidas, de acuerdo al dispositivo conectado a la línea correspondiente. Escribiendo un 1 lógico en el bit correspondiente del registro TRIS, la línea del puerto se configura como entrada, en caso contrario, un 0 lógico indicará que la línea es salida. Cada uno de los dos puertos del PIC16F84 tiene su correspondiente registro TRIS (TRISA y TRISB), localizados en las direcciones 85h y 86h, respectivamente.

V.5. Programación de los puertos

El puerto B

El puerto B del PIC16F84 tiene 8 líneas, su registro para configurar las líneas como entradas o salidas es el registro TRISB localizado en la dirección 86h. Cada línea del puerto B tiene una resistencia de pull-up interna (el resistor que define una línea a uno lógico), la cual puede activarse escribiendo un 0 lógico en séptimo bit (RBPU) del registro OPTION. Estas resistencias automáticamente se deshabilitan cuando la línea del puerto es configurada como salida. Cuando se alimenta el microcontrolador los pull-up´s se deshabilitan.

Las cuatro líneas más significativas del puerto B: RB4 a RB7, pueden generar una interrupción cuando su estado cambia de uno a cero lógico o de cero a uno lógico. Solo las líneas configuradas como entradas pueden generar la interrupción (si cualquiera de las líneas RB4-RB7 es configurada como salida, no se genera interrupción al cambiar su estado). Esta característica en conjunto con las resistencias de pull-up facilita la resolución de problemas prácticos como por ejemplo usar un teclado matricial. Si las filas del teclado se conectan a entradas del puerto B, al presionar una tecla se puede generar una interrupción. El

microcontrolador determina cual tecla se presionó al procesar la interrupción.

```
clrf    STATUS      ;Selección del banco 0.
clrf    PORTB       ;Puerto B=0.
bsf     STATUS,RP0  ;Selección del banco 1.
movlw   0x0F        ;Define las líneas de entrada y salida.
movwf   TRISB       ;Escribe al registro TRISB.
```

En el ejemplo anterior se muestra como las líneas 0, 1, 2 y 3 del puerto B se declaran como entradas y las líneas 4, 5, 6 y 7 como salidas.

El puerto A

El puerto A del PIC16F84 tiene cinco líneas, su registro para configurar las líneas como entradas o salidas es TRISA localizado en la dirección 85h. La quinta línea del puerto A tiene dos funciones. En esta línea está localizada también una entrada externa para el timer TMR0. Una de las dos funciones se selecciona con el bit T0CS (TMR0 Clock Source Selection bit) del registro OPTION. Este bit le indica al timer TMR0 que incremente su valor ya sea del oscilador interno o vía pulsos externos por medio de la línea RA4/T0CKI.

```
bcf     STATUS,RP0  ;Selección del banco 0.
clrf    PORTA       ;Puerto A=0.
bsf     STATUS,RP0  ;Selección del banco 1.
movlw   0x1F        ;Define las líneas de entrada y salida.
movwf   TRISA       ;Escribe al registro TRISA.
```

En el ejemplo anterior se muestra como las líneas 0, 1, 2, 3 y 4 del puerto A se declaran como entradas y las líneas 5, 6 y 7 como salidas.

Resumiendo, de lo que se ha visto hasta el momento se puede ya configurar el hardware para algunas líneas de los puertos del PIC16F84 funcionen como entradas y otras funcionen como salidas de la siguiente manera:

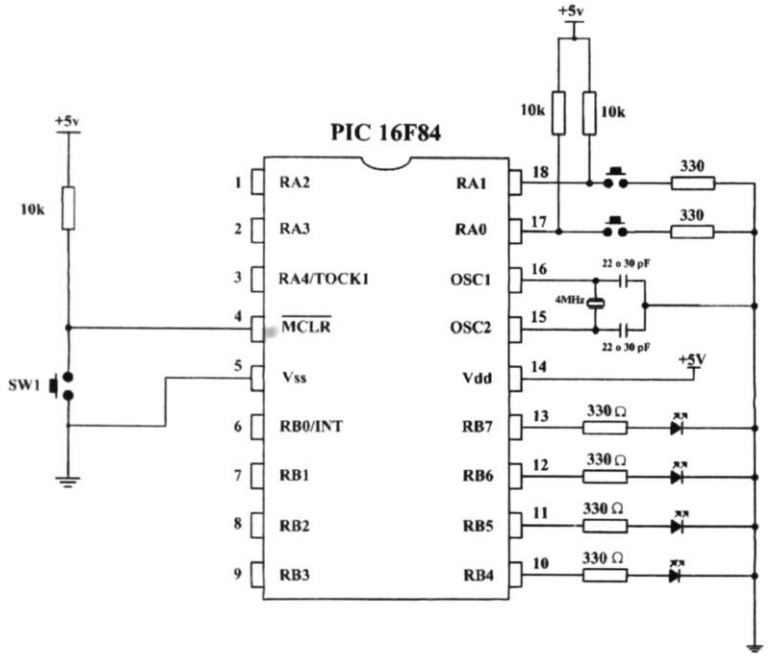


Figura V.5. Líneas de los puertos del PIC16F84 usadas como salidas y como entradas

A continuación se muestra el circuito de una fuente de alimentación para proporcionar una tensión estable de trabajo de +5 Vcc. a un sistema digital que use un PIC16F84.

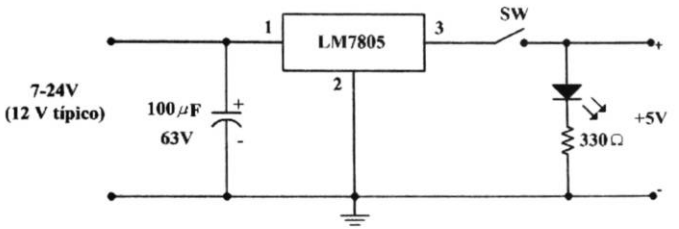


Figura V.6. Fuente de alimentación típica del PIC16F84

UNIDAD VI

MANEJO DE INTERRUPCIONES

VI.1. Tipos de interrupción

Las interrupciones son un mecanismo del microcontrolador que le permite responder a eventos asíncronos en el momento que ocurren, independientemente de lo que el microcontrolador este realizando. Esta es una parte muy importante del circuito porque proporciona una conexión entre un microcontrolador y el ambiente que lo rodea. Generalmente, cada interrupción cambia el flujo del programa, lo interrumpe y después de ejecutar una rutina de atención continúa en el mismo punto del programa.

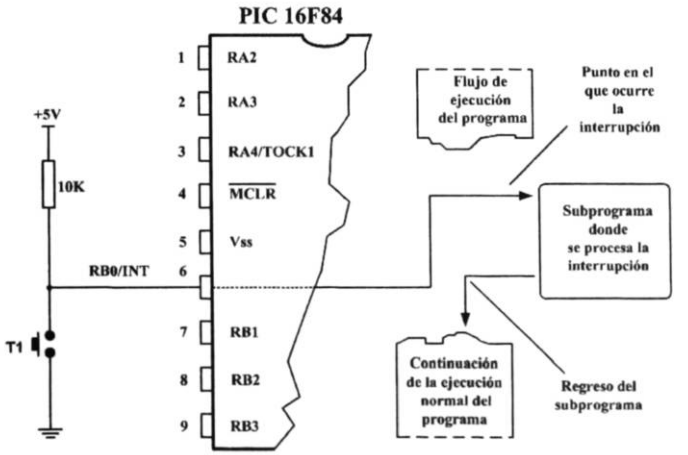


Figura VI.1. Relación entre una interrupción y el programa principal

VI.2. Modos de operación

La lógica de interrupciones del PIC16F84 tiene un registro de control llamado **INTCON** que se encuentra en la localidad de memoria RAM 0Bh. La función principal de este registro es habilitar o deshabilitar las interrupciones e indicar las solicitudes de interrupción recibidas por medio de cada uno de sus bits.

VI.3. Configuración de interrupciones

El registro INTCON

GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

R = Bit que se puede leer. W = Bit que se puede escribir.
U = Bit no usado, se lee un 0.
-n = Valor inicial al encender la fuente de alimentación.

Bit 0-RBIF (RB Port Change Interrupt Flag Bit-Bit de la bandera de la interrupción de un cambio en el puerto B)
Este bit informa si ha ocurrido un cambio en alguna de las líneas 4, 5, 6 y 7 del puerto B.
1 = al menos una línea ha cambiado su estado.
0 = no ha ocurrido cambio en alguna de las líneas.

Bit 1-INTF (INT External Interrupt Flag Bit-Bit de la bandera de la interrupción externa INT)
1 = ocurrió una interrupción de este tipo.
0 = no ocurrió una interrupción de este tipo.
Si un flanco de subida o bajada configurado se detectó en la línea RB0/INT, (el cual es seleccionado con el bit INTEDG del registro OPTION), se activa el bit INTF. Este bit debe limpiarse en la rutina de atención para detectar la siguiente interrupción.

Bit 2-TOIF (TMR0 Overflow Interrupt Flag Bit-Bit de la bandera de la interrupción de sobreflujo del TMR0)
1 = el contador cambio su cuenta de FFh a 00h.
0 = no ocurrió sobreflujo.
Este bit debe limpiarse en el programa para que pueda detectarse una siguiente interrupción de este tipo.

Bit 3-RBIE (RB Port Change Interrupt Enable Bit-Bit de habilitación de la interrupción de un cambio en el puerto B)
Habilita la interrupción de cambio de estado de las líneas 4, 5, 6 y 7 del puerto B.
1 = habilita que la interrupción ocurra con un cambio de estado.
0 = las interrupciones se deshabilitan cuando ocurra un cambio de estado.
Si RBIE y RBIF se habilitan simultáneamente, puede ocurrir una interrupción de este tipo.

Bit 4-INTE (INT External Interrupt Enable Bit-Bit de habilitación de la interrupción externa INT)
Este bit habilita la interrupción externa de la línea RB0/INT.
1 = interrupción externa habilitada.
0 = interrupción externa deshabilitada.

Bit 5-TOIE (TMR0 Overflow Interrupt Enable Bit-Bit de habilitación de la interrupción de sobreflujo del temporizador TMR0)
Este bit habilita la interrupción cuando ocurra un sobreflujo en el contador del temporizador TMR0.
1 = interrupción habilitada.
0 = interrupción deshabilitada.
Si TOIE y TOIF se habilitan simultáneamente, puede ocurrir una interrupción de este tipo.

Bit 6-EEIE (EEPROM Write Complete Interrupt Enable Bit-Bit de habilitación de la interrupción al completarse una escritura en la EEPROM)
Este bit habilita la interrupción que ocurre al terminar una rutina de escritura a la EEPROM.
1 = interrupción habilitada.
0 = interrupción deshabilitada.
Si EEIE y EEIF (que se encuentra en el registro EECON1) se habilitan simultáneamente, puede ocurrir una interrupción de este tipo.

Bit 7-GIE (Global Interrupt Enable Bit-Bit de habilitación global de interrupciones)
Este bit habilita o deshabilita todas las interrupciones.
1 = se habilitan todas las interrupciones.
0 = se deshabilitan todas las interrupciones.

Así pues, el PIC16F84 tiene cuatro fuentes de interrupción:

1) Al terminar una escritura de datos a la EEPROM.
2) La interrupción causada por el sobreflujo de TMR0.
3) La interrupción al cambiar de estado cualquiera de las líneas RB4, RB5, RB6 y RB7 del puerto B.
4) La interrupción externa proveniente de la línea RB0/INT del microcontrolador.

Generalmente hablando, cada interrupción tiene asociados dos bits. Uno habilita la interrupción y el otro indica cuándo la interrupción ha ocurrido. Existe un bit común llamado GIE el cual puede usarse para habilitar o deshabilitar todas las interrupciones simultáneamente. Este bit es muy útil al escribir un programa para deshabilitar todas las interrupciones por un periodo de tiempo tal que no pueda ser interrumpida alguna parte importante o crítica del programa.

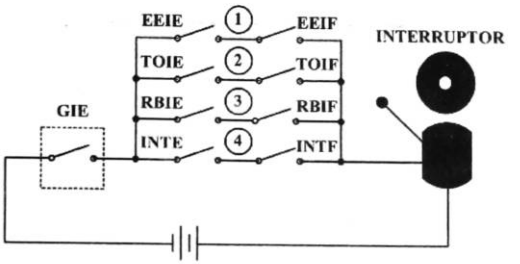


Figura VI.2. Lógica de interrupciones del PIC16F84

Cuando se atiende una interrupción se limpia automáticamente el bit GIE para deshabilitar interrupciones adicionales, la dirección de retorno del programa interrumpido se salva en el stack y se carga el program counter con la dirección 0004h y solo después de esto, la atención a la interrupción comienza. Después de procesarse la interrupción, el bit que la causó debe limpiarse o la rutina de la interrupción será procesada nuevamente al regresar al programa interrumpido.

Salvar el contenido de los registros importantes

Solamente se salva en el stack el valor de regreso del program counter durante una interrupción (la dirección de regreso del program counter es la dirección de la instrucción que debía ejecutarse pero no lo fue debido a que ocurrió la interrupción). Salvar el valor del program counter a menudo es suficiente, sin embargo, algunos registros que se están usando en el programa interrumpido se pueden usar en la rutina de la interrupción. Si esos registros no son salvados, el programa interrumpido podrá encontrar valores diferentes en esos registros al regresar de la interrupción, lo cual podrá causar errores.

En algunos microcontroladores el proceso de salvar el contenido de registros importantes antes de ir a una rutina de una interrupción se hace con instrucciones llamadas PUSH, y el proceso de restaurar su valor se hace con instrucciones llamadas POP. El PIC16F84 **NO** tiene instrucciones de este tipo y deben ser programadas en caso necesario.

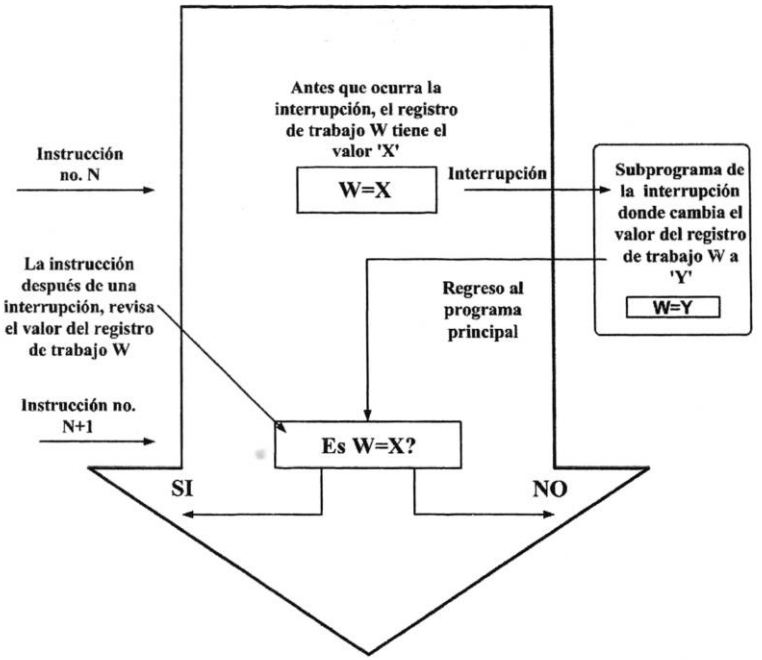


Figura VI.3. Problema que ocurre cuando no se salva el contexto antes de una rutina de interrupción

Debido a la frecuencia de uso y simplicidad de estos procesos, estas partes del programa pueden hacerse como macros. En el ejemplo siguiente, el contenido de los registros W y STATUS se salvan en las variables (o localidades de memoria) W_TEMP y STATUS_TEMP antes de la rutina de la interrupción. Para intercambiar los datos entre los registros STATUS y W se puede usar la instrucción SWAPF en lugar de la instrucción MOVF, ya que la primera no afecta el valor de los bits del registro STATUS.

La instrucción SWAPF STATUS,0 carga los bits 0-3 del registro STATUS en los bits o posiciones 4-7 del registro W y los bits 4-7 del registro STATUS en los bits 0-3 del registro W.

El ejemplo sigue los pasos indicados a continuación:

1) Almacena el registro W independientemente del banco actual.
2) Almacena el registro STATUS independientemente del banco actual.
3) Salva los registros deseados del banco 0.
4) Salva los registros deseados del banco 1.
5) Ejecuta la rutina de la interrupción (ISR).
6) Restaura los registros salvados del banco 0.
7) Restaura los registros salvados del banco 1.
8) Restaura el valor del registro STATUS.
9) Restaura el valor del registro W.

Si existen más variables o registros que necesitan salvarse, debe hacerse después de salvar el registro STATUS (pasos 3 y 4), y restaurarse antes del registro STATUS (pasos 6 y 7).

```
PUSH
     MOVWF     W_TEMP          ;Salva el registro W.
     SWAPF     STATUS,0        ;W ← STATUS.
     MOVWF     STATUS_TEMP     ;STATUS_TEMP ← W.
;
;***** Salva los registros del banco 0 *****
;
     BCF       STATUS,RP0      ;Cambia al banco 0.
     MOVF      TMR0,W          ;Salva el registro TMR0.
     MOVWF     TMR0_TEMP
     MOVF      PORTA,W         ;Salva el registro PORTA.
     MOVWF     PORTA_TEMP
;
;***** Salva los registros del banco 1 *****
:
     BSF       STATUS,RP0      ;Cambia al banco 1.
     MOVF      OPTION_REG,W    ;Salva el registro OPTION.
     MOVWF     OPTION_TEMP
     MOVF      TRISA,W         ;Salva el registro TRISA.
     MOVWF     TRISA_TEMP
                               ;Termina el PUSH.
ISR_CODE
     ;
     ;(Programa de la interrupción).
     ;
POP
;
;***** Restaura los registros del banco 0 *****
;
     BCF       STATUS,RP0      ;Cambia al banco 0.
     MOVF      TMR0_TEMP,W     ;Restaura reg. TMR0.
     MOVWF     TMR0
     MOVF      PORTA_TEMP,W    ;Restaura reg. PORTA.
     MOVWF     PORTA
;
```

```
;***** Restaura los registros del banco 1 *****
;
     BSF        STATUS,RP0      ;Cambia al banco 1.
     MOVF       OPTION_TEMP,W   ;Restaura reg. OPTION.
     MOVWF      OPTION_REG
     MOVF       TRISA_TEMP,W    ;Restaura reg. TRISA.
     MOVWF      TRISA
;
     SWAPF      STATUS_TEMP,0   ;W ← STATUS_TEMP.
     MOVWF      STATUS          ;STATUS ← W.
     MOVF       W_TEMP,W        ;Restaura contenido de W.
     RETFIE                     ;Termina el POP.
```

Este mismo ejemplo se puede realizar usando macros, obteniendo con esto un programa más legible. Los macros ya definidos pueden usarse para escribir nuevos macros. Los macros llamados BANK1 y BANK0, los cuales ya se explicaron en el tema de "Organización de memoria", se usan en los macros PUSH y POP.

```
PUSH    macro
        MOVWF       W_TEMP          ;Salva el registro W.
        SWAPF       STATUS,W        ;W ← STATUS.
        MOVWF       STATUS_TEMP     ;STATUS_TEMP ← W.
;
;***** Salva los registros del banco 0 *****
;
        BANK0                       ;Cambia al banco 0.
        MOVF        TMR0,W          ;Salva el registro TMR0.
        MOVWF       TMR0_TEMP
        MOVF        PORTA,W         ;Salva el registro PORTA.
        MOVWF       PORTA_TEMP
;
;***** Salva los registros del banco 1 *****
:
        BANK1                       ;Cambia al banco 1.
        MOVF        OPTION_REG,W    ;Salva el registro OPTION.
        MOVWF       OPTION_TEMP
        MOVF        TRISA,W         ;Salva el registro TRISA.
        MOVWF       TRISA_TEMP
        endm                        ;Termina el macro PUSH.
```

```
POP macro
;
;***** Restaura los registros del banco 0 *****
;
    BANK0                           ;Cambia al banco 0.
    MOVF        TMR0_TEMP,W         ;Restaura reg. TMR0.
    MOVWF       TMR0
    MOVF        PORTA_TEMP,W        ;Restaura reg. PORTA.
    MOVWF       PORTA
;
;***** Restaura los registros del banco 1 *****
;
    BANK1                           ;Cambia al banco 1.
    MOVF        OPTION_TEMP,W       ;Restaura reg. OPTION.
    MOVWF       OPTION_REG
    MOVF        TRISA_TEMP,W        ;Restaura reg. TRISA.
    MOVWF       TRISA
;
    SWAPF       STATUS_TEMP,W       ;W ← STATUS_TEMP.
    MOVWF       STATUS              ;STATUS ← W.
    MOVF        W_TEMP,W            ;Restaura contenido de W.
    endm                            ;Termina el macro POP.
```

Interrupción externa de la línea RB0/INT del microcontrolador

La interrupción externa de la línea RB0/INT es disparada por la transición de subida (si el bit INTEDG=1 en el registro OPTION<6>), o por la transición de bajada (si INTEDG=0). Cuando la señal correcta aparece en la línea INT se activa el bit INTF del registro INTCON. El bit INTF (INTCON<1>) debe limpiarse en la rutina de la interrupción para que no vuelva a ocurrir otra interrupción al regresar al programa principal. Esta es una parte importante que el programador no debe olvidar o el programa constantemente estará pasando a la rutina de la interrupción. La interrupción puede apagarse limpiando el bit INTE (INTCON<4>).

Interrupción de sobreflujo del contador de TMR0

Un sobreflujo del contador de TMR0 (cuando la cuenta pasa de FFh a 00h) activará el bit TOIF (INTCON<2>). Una de las aplicaciones de esta interrupción es para medir tiempo. Si se conoce cuanto tiempo necesita el contador para completar un ciclo de 00h a FFh, entonces el número de interrupciones multiplicado por esa cantidad de tiempo dará como resultado el tiempo total transcurrido. En la rutina de atención de la interrupción alguna variable podrá ser incrementada en la memoria RAM, el valor de esa variable multiplicado por la cantidad de tiempo que el contador necesita para contar un ciclo completo dará como resultado el tiempo transcurrido. La interrupción puede apagarse o encenderse limpiando o activando el bit TOIE (INTCON<5>).

Interrupción de cambio de estado de las líneas 4, 5, 6 y 7 del puerto B

El cambio de la señal de entrada de PORTB <7:4> activa el bit RBIF (INTCON<0>). Las cuatro líneas RB7, RB6, RB5 y RB4 del puerto B pueden disparar una interrupción la cual ocurre cuando el estado de ellas cambia de uno a cero lógico o viceversa. Para que las líneas sean sensitivas a este cambio, deben ser definidas como entradas. Si alguna de ellas es definida como salida, no se genera interrupción al ocurrir un cambio de estado en esa línea. Si son definidas como entradas, su estado actual se compara con el valor anterior que fue almacenado en la última lectura del puerto B. La interrupción puede apagarse o encenderse limpiando o activando el bit RBIE del registro INTCON.

Interrupción al terminar la subrutina de escritura a la EEPROM

Debido a que escribir a una localidad de la EEPROM toma aproximadamente 10 ms (el cual es un tiempo grande para el microcontrolador), muchas veces no tiene caso que el microcontrolador espere ocioso que termine la escritura. El PIC16F84 cuenta con un mecanismo de interrupción que le permite continuar ejecutando el programa principal mientras se realiza una escritura a la EEPROM en segundo plano. Al terminar la escritura, una interrupción informa al microcontrolador dicho evento. El bit EEIF, por medio del cual se informa el final de la escritura, se encuentra en el registro EECON1. La ocurrencia de la interrupción se puede deshabilitar limpiando el bit EEIE del registro INTCON.

Inicialización de una interrupción

Para usar el mecanismo de interrupciones del microcontrolador es necesario realizar algunas tareas preparatorias. Esas tareas comúnmente se les conoce como inicialización. La inicialización es indicar a cuales interrupciones responderá el microcontrolador y cuales ignorará. Si no se activa el bit que permite una cierta interrupción, el programa no ejecutará la rutina asociada a esa interrupción, pudiéndose tener así el control en la ocurrencia de interrupciones.

```
clrf    INTCON      ;Deshabilita todas las interrupciones.
movlw   B'00010000' ;Solo habilita interrupción externa.
movwf   INTCON
bsf     INTCON,GIE  ;Habilita ocurrencia de interrupciones.
```

El ejemplo anterior muestra la inicialización de la interrupción externa de la línea RB0.

El ejemplo siguiente muestra una forma típica para el manejo de interrupciones. El PIC16F84 tiene solo una localidad de memoria donde se almacena la dirección de una rutina de una interrupción. Esto implica que es necesario primero detectar cuál interrupción se ha generado (si más de una fuente de interrupción está activa) y poder ejecutar la parte del programa que se refiere a esa interrupción.

```
org    ISR_ADDR          ;ISR_ADDR es la direcc. de la rutina
                         ;de interrupción.
btfsc  INTCON,GIE        ;Esta apagado el bit GIE?
goto   ISR_ADDR          ;No, regresa al inicio.
PUSH                     ;Salva contenido de regs. importantes.
btfsc  INTCON,RBIF       ;Cambian líneas 4,5,6 y 7 de puerto B?
goto   ISR_PORTB         ;Si, salta a esa sección.
btfsc  INTCON,INTF       ;Ocurrió interrupción externa?
goto   ISR_RB0           ;Si, salta a esa parte.
btfsc  INTCON,TOIF       ;Overflow del timer TMR0?
goto   ISR_TMR0          ;Si, salta a esa sección.
BANK1                    ;EECON1 esta en el banco 1.
btfsc  EECON1,EEIF       ;Se completo escritura a EEPROM?
goto   ISR_EEPROM        ;Si, salta a esa sección.
BANK0                    ;Banco 0.
;
ISR_PORTB
 ;
                         ;Sección de código procesado
                         ;por la interrupción.
 ;
      goto END_ISR       ;Salta al final de la interrupción.
ISR_RB0
 ;
                         ;Sección de código procesado
                         ;por la interrupción.
 ;
```

```
        goto END_ISR        ;Salta al final de la interrupción.
ISR_TMR0
 ;
                            ;Sección de código procesado
                            ;por la interrupción.
 ;
        goto END_ISR        ;Salta al final de la interrupción.
ISR_EEPROM
 ;
                            ;Sección de código procesado
                            ;por la interrupción.
 ;
        goto END_ISR        ;Salta al final de la interrupción.
END_ISR
        POP                 ;Restaura el contenido de los
                            ;registros importantes.
        RETFIE              ;Regresa y activa bit GIE.
```

El regreso de una rutina de atención a una interrupción se puede realizar con las instrucciones RETURN, RETLW y RETFIE. Se recomienda usar la instrucción RETFIE, ya que esta instrucción es la única que automáticamente activa el bit GIE el cual permite que ocurran nuevas interrupciones.

UNIDAD VII

APLICACIONES

Programación del microcontrolador

Los ejemplos mostrados a continuación en esta unidad mostrarán cómo conectar el PIC16F84 con otros periféricos o dispositivos al desarrollar un sistema en particular.

VII.1. La fuente de alimentación del microcontrolador

Para un funcionamiento correcto del microcontrolador es necesario proporcionarle una fuente de alimentación estable, una señal de reset correcta al encenderlo y una señal de reloj (oscilador) también correcta. De acuerdo a las especificaciones técnicas del fabricante, la fuente de alimentación del PIC 16F84 debe ser de 2.0V a 6.0V. Comúnmente se usa un regulador de voltaje LM7805 para obtener una salida de +5V, tal y como se muestra en la siguiente figura.

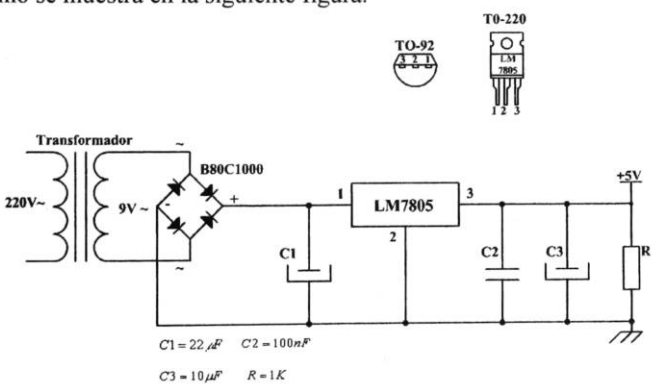


Figura VII.1. Fuente de alimentación típica

Para obtener una salida estable de 5V en la salida (línea 3) del regulador, el voltaje de entrada en la línea 1 del LM7805 debe estar entre 7V y 24V y dependiendo del consumo de corriente del circuito a alimentar se debe usar el regulador LM7805 apropiado. Existen diferentes versiones del LM7805 y para un consumo de corriente de hasta 1A se debe usar el LM7805 con encapsulado de tipo TO-220. Si la corriente total a consumir es de 50mA, se puede usar el 78L05 y para corrientes de hasta 100mA se puede usar el regulador con encapsulado pequeño del tipo TO-92.

Algunas acciones más comúnmente usadas en algunos programas son las rutinas de retardo y la rutina de impresión o desplegado por un puerto del PIC de un mensaje. Estas rutinas se verán a continuación en forma de Macros que podrán ser invocados en los ejemplos cubiertos en esta unidad.

VII.2. Los macros ESPERA y ESP_MAS

Estos dos macros usan para su funcionamiento el sobre flujo y el prescalar del timer **TMR0**, lo cual implica que se puede cambiar el intervalo del sobre flujo cambiando el prescalar.

Así pues, si se usa por ejemplo un oscilador de 4Mhz y valores para el prescalar de 0, 1 y 7 para dividir el reloj del oscilador, el intervalo del sobreflujo del timer **TMR0** será de 0.512, 1.02 y 65.536 mS.

Por ejemplo, si se usa un prescalar de 2, el intervalo del timer **TMR0** será:

Fosc = 4Mhz.

F_{TMR0} = [(4Mhz. / 4) / Prescalar] / 256 = [(4Mhz. / 4) / 2] / 256
$F_{TMR0} = (0.5 \times 10^{6}) / 256 = 1953.125$

$T_{TMR0} = 5.12 \text{ X } 10^{-4}$ s = 0.512 ms

Prescalar	Divisor	Intervalo
0000 0000	1:2	0.512 ms
0000 0001	1:4	1.024 ms
0000 0111	1:256	65.536 ms

En la tabla anterior se indica que el intervalo máximo que se puede lograr con el timer **TMR0** son 65.536 ms e invocando esa rutina de retardo con una constante de 8 bits (256 veces) se puede lograr un retardo de hasta 65.536 ms x 256 = 16.777 segundos.

A continuación se muestra un archivo que contiene dos Macros: **ESPERA** y **ESP_MAS,** el primero de ellos sirve para realizar un retardo pasándole como parámetro una constante de 8 bits que le indica cuantas veces esperará por el sobre flujo del timer **TMR0** usando un prescalar de 0000 0001.

El segundo Macro también sirve para realizar un retardo pasándole como parámetro una constante de 8 bits y además el valor para el prescalar del timer **TMR0**.

Ambos Macros se encontrarán en el archivo **"espera.inc"** para poder usarlos en otros programas posteriormente.

```
;***** Declaración de constantes *****
        CONSTANT PRESCALAR_1 = 1    ;Valor del prescalar por default.

;***** Macros *****

ESPERA  macro   constante_1
        movlw   constante_1         ;constante_1 es la cantidad de
                                    ;sobreflujos a esperar.
        call    RETARDO
        endm

ESP_MAS macro   constante_2,PRESCALAR_2
        movlw   constante_2         ;constante_2 es el numero de
                                    ;sobreflujos a esperar.
```

```
            movwf   CICLO           ;Contador de sobreflujos.
            movlw   PRESCALAR_2     ;PRESCALAR_2 es un prescalar para
                                    ;el TMR0 pasado como parámetro
                                    ;al macro.
            call    RETARDOX
            endm

;***** Cuerpo principal de rutinas *****

RETARDO
            movwf   CICLO           ;Contador de sobreflujos.
            movlw   PRESCALAR_1     ;Prescalar (fijo) asignado a TMR0.
RETARDOX
            clrf    TMR0
            BANK1
            movwf   OPTION_REG      ;Asigna el prescalar al timer TMR0.
            BANK0
WAIT1       bcf     INTCON,TOIF     ;Limpia bandera sobreflujo de TMR0.
WAIT2       btfss   INTCON,TOIF     ;Sobreflujo=1?

            goto    WAIT2           ;No, continua esperando sobreflujo.
            decfsz  CICLO,1         ;Si,decrementa contador de sobreflujos,
                                    ;contador de sobreflujos=0?
            goto    WAIT1           ;No, espera otro sobreflujo.

            RETURN                  ;Si, terminó retardo.
```

En los Macros anteriores el parámetro **constante_1** es un número de 0 a 255, de manera que si se invoca al Macro de la manera siguiente:

ESPERA 100

El retardo del Macro será de 100 x 1.02ms = 102ms.

Como ya se indicó anteriormente, el Macro **ESP_MAS** tiene un argumento adicional que le indica el prescalar usado, por ejemplo:

ESP_MAS 100,7

El retardo del Macro será de 100 x 65.536 ms = 6553.6ms. = 6.553 segundos.

VII.3. El macro PRINT

Este Macro envía una cadena de caracteres a un dispositivo de salida (un display de cristal líquido de cuarzo-LCD, el puerto serie RS232, una impresora, etcétera). Su funcionamiento se basa usando la directiva **dt** (define table) para definir una serie de datos en memoria de programa como un grupo de instrucciones **retwl** cuyo operando es un dato de la cadena. Este Macro se encontrará en el archivo **"print.inc"** para poder usarlo en otros programas.

```
PRINT   macro Tabla, Mensaje1, FinTabla, Dato, Salida

        Local   Otro        ;Etiquetas
        Local   Salida      ;     locales.

        movlw   Mensaje1    ;Direcc. primer elemento de la tabla.
        movwf   Dato
Otro
        movf    Dato,w      ;Apuntador → reg. W.
        call    Tabla       ;Dato a desplegar → reg. W.

        Out ....            ;Llamada a rutina para desplegar dato.

        movf    Dato,w
        xorlw   FinTabla-1
        btfsc   STATUS,Z    ;Es el final de la cadena?
        goto    Salida      ;Si, sale del loop.
        incf    Dato,f      ;No, incrementa apuntador.
        goto    Otro        ;Continua en el loop.
Salida
        endm
```

A continuación se muestra un segmento del programa que usa el Macro **PRINT**:

```
        org     0x00
        goto    Main

Tabla   movwf   PCL
Mensaje1 dt "Sistemas Digitales II"
Mensaje2 dt "Electrónica"

FinTabla
Main
        PRINT Tabla,Mensaje1,FinTabla,Dato,LCD
            ...
            ...
```

En el Macro **PRINT** se observa que se salta con una instrucción **call** a la etiqueta Tabla donde están definidos los mensajes a desplegar o imprimir y al inicio de esta tabla se le asigna al program counter (**PCL**) la posición de un miembro de alguno de los mensajes. A continuación, el program counter apuntará a localidad de memoria de una instrucción **retlw** y cuando se ejecuta esta instrucción, el miembro del mensaje correspondiente se carga en el registro **W**. La etiqueta **FinTabla** es una forma elegante de marcar el fin de la tabla de mensajes a desplegar.

El Macro **PRINT** tiene cinco argumentos:

```
PRINT Tabla,Mensaje1,FinTabla,Dato,LCD
```

Tabla es la dirección donde inician uno o más mensajes.
Mensaje1 es la dirección de inicio del primer mensaje.
FinTabla es la dirección donde terminan los mensajes.
Dato es la variable donde se almacena el apuntador a los miembros del mensaje.
LCD es la dirección del dispositivo de salida donde se despliegan los mensajes.

VII.4. Los diodos emisores de luz LED´s (Ligh-Emitting Diodes)

Los LED´s son algunos de los elementos más comúnmente usados en electrónica. LED es una abreviación de "Light Emitting Diode". Para seleccionar un LED se toman en cuenta varios parámetros como son el diámetro (típicamente 3 o 5 mm), corriente de trabajo (10mA, pero puede ser tan bajo como 2mA para LED´s de alta eficiencia) y el color.

El LED y su resistencia limitadora de corriente deben conectarse correctamente para evitar dañarlo o quemarlo. El positivo de la fuente debe conectarse al ánodo y el cátodo al negativo o tierra del circuito. El cátodo es comúnmente el de la pata más corta y es la parte plana del bulbo del LED. La unión PN debe polarizarse correctamente para que fluya corriente de ánodo a cátodo y el LED emita luz, por lo que debe determinarse el valor correcto de la resistencia limitadora de corriente.

El valor de la resistencia limitadora depende básicamente de la fuente de alimentación, de la cual se resta la caída de voltaje en el LED (Ud), cuyo valor es de 1.2v a 1.6v dependiendo del color del LED.

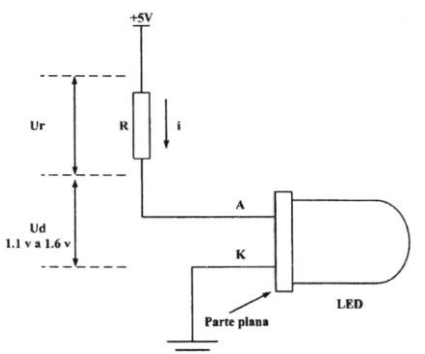


Figura VII.2. Polarización de un led

Usando la caída de voltaje en la resistencia limitadora (Ur) y el valor de la corriente a circular en el LED (0.002A a 0.01A) se puede determinar el valor de la resistencia R=Ur/I.

Ur = 5v – Ud = 5v – 1.5v = 3.5v.

Así, R = 3.5v / 0.01A = 350 ohms (330 ohms aproximadamente).

En los ejemplos a usar en esta unidad se conectarán los LED´s usando lógica positiva, es decir, para encender el LED se enviará por la línea a la cual esté conectado un uno lógico, tal y como lo muestra en el ejemplo siguiente en donde el programa inicializa al puerto B como salida y envía, en efecto, un uno lógico en cada línea del puerto B para encender los LED´s.

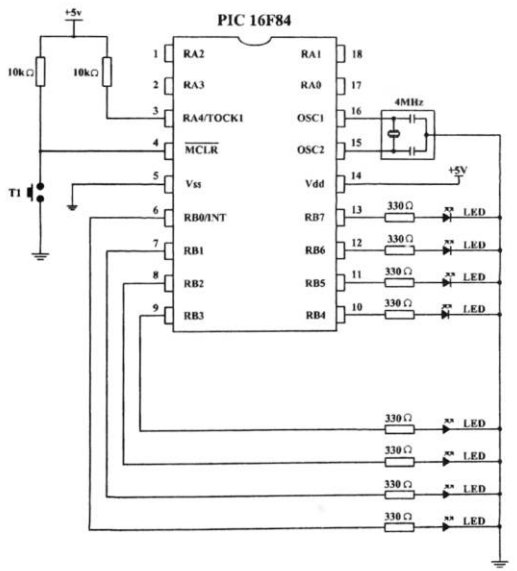


Figura VII.3. Uso de los puertos del PIC16F84 con leds

```
;***** Declaración y configuración del microcontrolador *****

        PROCESSOR 16F84
        #include "p16f84.inc"

        __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Estructura de la Memoria de Programa *****

        ORG     0x00        ;Vector de reset.
        goto    Main

        ORG     0x04        ;Vector de interrupción.
        goto    Main        ;No hay rutina de interrupción.

        #include "bancos.inc"

Main                        ;Inicio del programa principal.
        BANK1
        movlw   0xff        ;Inicializa puerto A.
        movwf   TRISA       ;TRISA ← 0xff, entradas.
        movlw   0x00        ;Inicializa puerto B.
        movwf   TRISB       ;TRISA ← 0x00, salidas.
        BANK0

        movlw   0xff
        movwf   PORTB       ;Enciende todos los leds.

Loop
        goto    Loop        ;Permanece en el loop.

        End                 ;Fin del programa.
```

VII.5. El teclado

Los teclados son dispositivos mecánicos usados para conectar dos puntos. Vienen en diferentes tipos y con diferentes propósitos. Las teclas que se usarán en este ejemplo son de los llamados "dip-switch" o "dip-keys" y tienen cuatro patas (dos para cada contacto) que les da estabilidad mecánica.

Cuando el switch se presiona se unen dos contactos, pero debido a la imperfección mecánica de los contactos existe un periodo muy corto de tiempo en el que puede ocurrir una vibración (oscilación) como resultado de esta desigualdad en los contactos o de la diferente velocidad al presionar una tecla. A este fenómeno se le denomina SWITCH DEBOUNCE, CONTACT DEBOUNCE o rebotes. Si esto se ignora al escribir un programa se pueden producir errores ya que se genera más de un pulso de entrada al presionar una tecla.

Para evitar rebotes se puede introducir un pequeño retardo al detectar el cierre del contacto de la tecla y asegurar que al presionar una tecla solo se produzca un pulso. El retardo se puede hacer conectando un capacitor en la tecla, resolviendo parcialmente el problema, por lo que es mejor usar un retardo por software cuya longitud dependerá de la tecla y que se puede ajustar hasta que sea eliminada completamente una detección falsa.

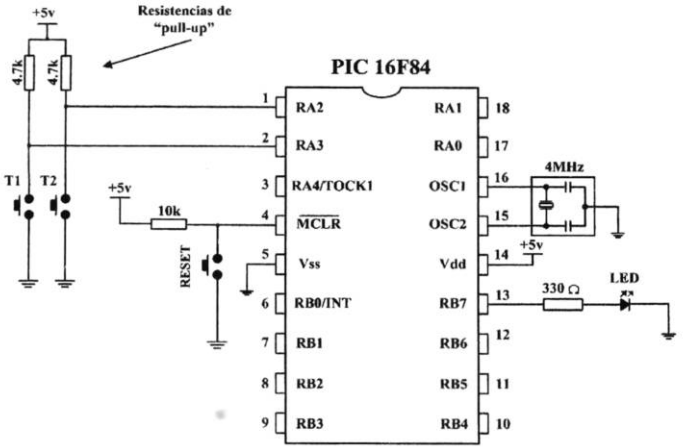


Figura VII.4. Líneas del PIC16F84 usadas como entradas con switch´s

La solución a los rebotes es tener entonces un programa que detecte el momento en el que se presiona una tecla y el momento en el que se libera la misma.

```
TECLA macro Nivel,Puerto,Bit,Retardo,Rutina

        Local   Salida              ;Etiquetas locales.
        Local   Loop
        if Nivel == 0               ;Nivel activo es 0?.
            btfsc Puerto,Bit        ;Línea de entrada activada?
                                    ;Si, salta a esperar retardo.
        else                        ;Nivel activo es 1?
            btfss Puerto,Bit        ;Línea de entrada activada?
                                    ;Si, salta a esperar retardo.
        endif
        goto Salida             ;No hay tecla presionada, sal del macro.

        ESPERA  Retardo             ;Retardo para el rebote.
Loop
        if Nivel == 0               ;Nivel activo es 0?.
            btfss Puerto,Bit        ;Línea de entrada desactivada
                                    ;(tecla liberada)?
                                    ;Si, salta a esperar retardo.
        else                        ;Nivel activo es 1.
            btfsc Puerto,Bit        ;Línea de entrada desactivada
                                    ;(tecla liberada)?
                                    ;Si, salta a esperar retardo.
        endif
        goto    Loop                ;No se ha liberado la tecla,
                                    ;permanece en el lazo.
        ESPERA Retardo              ;Retardo para el rebote.
        call    Rutina              ;Ejecuta la rutina de servicio.

Salida
        endm                        ;Fin del macro.
```

Este Macro se encontrará en el archivo **"tecla.inc"** para poder usarlo en otros programas y tiene los siguientes argumentos:

TECLA macro Nivel, Puerto ,Bit, Retardo, Rutina

Nivel Puede ser "0" o "1" y representa la transición de subida o de bajada a detectar para considerar que una tecla se ha presionado o liberado.

Puerto Es el puerto del microcontrolador al cual está conectado la tecla.

Bit Es la línea del puerto al cual está conectada la tecla.

Retardo Es un número de 0 a 255 usado para el tiempo necesario para eliminar el rebote. Es calculado como Tiempo = Retardo x 1.02ms.

Rutina Es la dirección donde salta el programa para atender la activación de la tecla.

TECLA 0,PUERTOA,0,100,Rutina1

La tecla 1 está conectada a **RA0** con un retardo de 100 microsegundos y una reacción a un cero lógico. La rutina que procesa la tecla se encuentra en la dirección de la etiqueta **Rutina1**.

En el programa siguiente se muestra el uso del Macro anterior y donde se enciende un LED conectado a la línea 7 del **puerto B** cuando se presiona la tecla 1 y se apaga al presionar la tecla 2.

```
;***** Declaración y configuración del microcontrolador *****
        PROCESSOR 16F84
        #include "p16f84.inc"

        __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
```

```
;***** Declaración de variables *****

        Cblock 0x0c             ;Inicio de la memoria RAM.
        CICLO                   ;Variables del Macro "ESPERA".
        PRESCALAR_2
        endc

;***** Estructura de la Memoria de Programa *****

        ORG     0x00    ;Vector de reset.
        goto    Main

        ORG     0x04    ;Vector de interrupción.
        goto    Main    ;No hay rutina de interrupción.

        #include "bancos.inc"
        #include "tecla.inc"
        #include "espera.inc"

Main                            ;Inicio del programa principal.
        BANK1
        movlw   0xff    ;Inicializa puerto A.
        movwf   TRISA   ;TRISA ← 0xff, entradas.
        movlw   0x00    ;Inicializa puerto B.
        movwf   TRISB   ;TRISB ← 0x00, salidas.
        BANK0

        clrf    PORTB   ;Puerto B ← 0.
Loop
        TECLA 0,PORTA,0,100,Enciende
        TECLA 0,PORTA,0,100,Apaga
        goto    Loop
Enciende
        bsf     PORTB,7 ;Enciende el LED
        return
Apaga
        bcf     PORTB,7 ;Apaga el LED
        return

        End                     ;Fin del programa.
```

VII.6. El optoacoplador

Un optoacoplador combina un LED y un foto-transistor en el mismo encapsulado. El propósito de un optoacoplador es separar dos partes de un circuito por las siguientes razones:

- **Interferencia.** Una parte del circuito se puede encontrar en un lugar donde exista mucha interferencia de motores eléctricos, de motores de combustible o de equipo de soldadura. Si la salida de este circuito se transfiere a otro circuito por medio de un optoacoplador, únicamente pasarán por el optoacopaldor las señales usadas, y las señales de las interferencias serán eliminadas para no activar el LED del optoacoplador, protegiendo así una sección de un dispositivo.

- **Amplificación de una señal.** Una señal de hasta 3v puede activar un optoacoplador y producir una salida de 5v que puede conectarse a una línea de entrada de un microcontrolador.

- **Separación de alto voltaje.** Los optoacopladores tienen características inherentes de separación de altos voltajes ya que el LED está separado completamente del foto-transistor pudiendo aislar voltajes de 3kv o más.

Un optoacoplador se puede usar como un dispositivo de entrada o como un dispositivo de salida. Pueden ser de encapsulados de uno o grupos de dos o más optoacopladores. También son llamados FOTO INTERRUPTORES ya que pueden contener una ranura entre el LED y el fototransistor para que cada vez que se interrumpa la luz el transistor produzca un pulso.

Cada optoacoplador necesita dos fuentes de alimentación para funcionar, aunque pueden usar una misma pero se pierde la característica de aislamiento de voltaje.

VII.7. El optoacoplador como línea de entrada

En esta configuración, cuando se le suministra al optoacoplador una señal, se enciende su LED interno y se activa la base de su foto-transistor, por lo que al encenderse dicho transistor, el voltaje entre el colector y el emisor cae a 0.5V o menos y el microcontrolador detecta esto como un cero lógico en la línea donde esta conectado el optoacoplador.

El ejemplo siguiente muestra un contador de artículos de una línea de producción. El sensor puede ser un micro-switch, de manera que cada vez que se cierra este micro-switch se ilumina el LED produciendo el optoacoplador un cero lógico en la entrada **RA4** del microcontrolador.

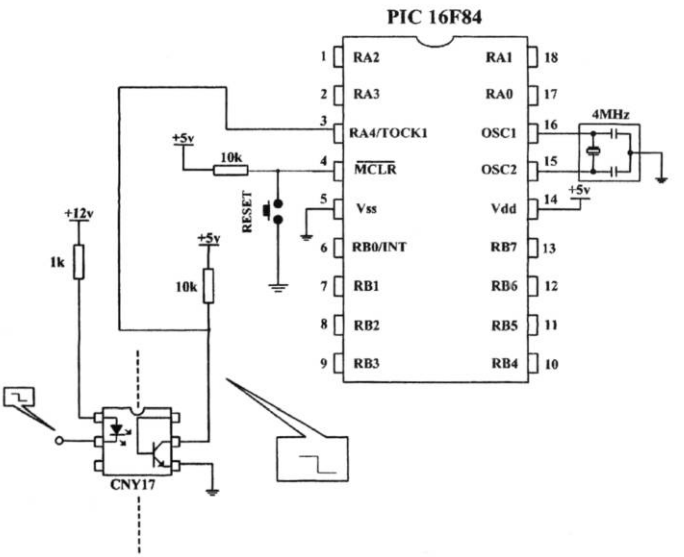


Figura VII.5. El optoacoplador como entrada

En el programa mostrado a continuación las salidas del microcontrolador desplegarán el estado del contador:

```
;***** Declaración y configuración del microcontrolador *****

        PROCESSOR 16F84
        #include "p16f84.inc"

        __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Estructura de la Memoria de Programa *****

        ORG     0x00        ;Vector de reset.
        goto    Main

        ORG     0x04        ;Vector de interrupción.
        goto    Main        ;No hay rutina de interrupción.

        #include "bancos.inc"

Main                        ;Inicio del programa principal.
        BANK1
        movlw   0xff        ;Inicializa puerto A.
        movwf   TRISA       ;TRISA ← 0xff, entradas.
        movlw   0x00        ;Inicializa puerto B.
        movwf   TRISB       ;TRISB ← 0x00, salidas.
        movlw   b'00110000'     ;RA4 → TMR0, Prescalar=1:2.
        movwf   OPTION_REG      ;Incrementa TMR0 con la bajada.
        BANK0

        clrf    PORTB       ;Puerto B ← 0.
        clrf    TMR0        ;TMR0 ← 0.

Loop    movf    TMR0,w      ;Copia el valor del timer al
        movwf   PORTB       ;al puerto B.
        goto    Loop        ;Repite el loop.

        End                 ;Fin del programa.
```

VII.8. El optoacoplador como línea de salida

Se puede usar un optoacoplador para separar la señal de salida de un microcontrolador de una etapa de alto voltaje o un como amplificador de corriente, ya que la salida de algunos microcontroladores está limitada a 25mA si trabaja como drenaje ("drain") o 20 mA si trabaja como fuente ("source").

El programa de este ejemplo deberá entregar un "1" lógico en la línea 3 del **Puerto A** para que se encienda el LED y se active el transistor del optoacoplador. La salida del transistor en este caso esta limitada aproximadamente a 250mA.

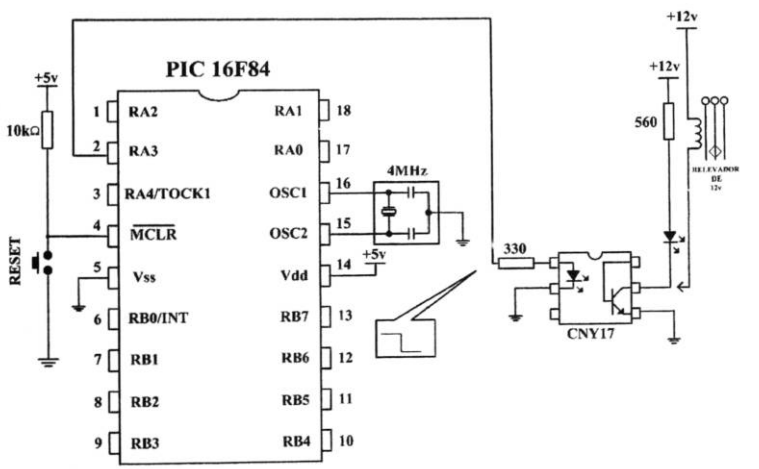


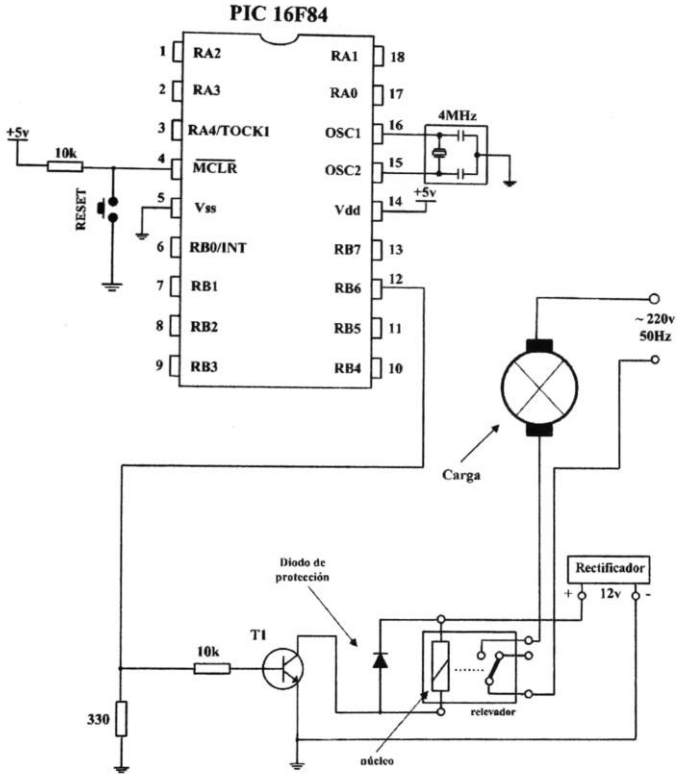
Figura VII.6. El optoacoplador como salida

VII.9. El relevador

El relevador es un dispositivo electromecánico que transforma una señal eléctrica en movimiento mecánico. Consiste de una bobina de alambre con material aislante, un núcleo de metal y una armadura metálica con uno o más contactos.

Conexión de un relevador al microcontrolador vía un transistor

Cuando se le suministra una fuente de voltaje a la bobina, circula una corriente y se produce un campo magnético que mueve la armadura para cerrar un conjunto de contactos y/o abrir otro conjunto. Cuando se remueve la alimentación del relevador, el flujo magnético de la bobina se colapsa y produce un alto voltaje en dirección opuesta. Este voltaje puede dañar el transistor que funciona como driver, por lo que se conecta un diodo polarizado inversamente en los extremos de la bobina para corto-circuitar el pico de voltaje cuando ocurra.



El transistor que funciona como driver se requiere porque muchos microcontroladores no pueden manejar directamente un relevador. Un nivel lógico alto en la base del transistor enciende y activa al relevador. El relevador se puede conectar a cualquier dispositivo eléctrico por medio de sus contactos.

La resistencia de 10K en la base del transistor sirve para limitar la corriente del microcontrolador a la requerida por el transistor. La resistencia de 10K entre la base del transistor y la tierra evita que el posible ruido existente en la base active al relevador, lo que permite que solo una señal clara y correcta del microcontrolador lo active.

Conexión de un relevador y un optoacoplador al microcontrolador

Se puede activar también un relevador por medio de un optoacoplador el cual amplifica la corriente de la salida del microcontrolador y proporciona un alto grado aislamiento. Los optoacopladores de corrientes altas típicamente contienen una salida con transistores "darlington" para proporcionar una corriente de salida alta.

La conexión vía un optoacoplador es recomendable especialmente para aplicaciones del microcontrolador donde se activan motores y el ruido producido al conmutarlos puede regresar al microcontrolador vía las líneas de alimentación. El optoacoplador maneja un relevador y el relevador activa un motor.

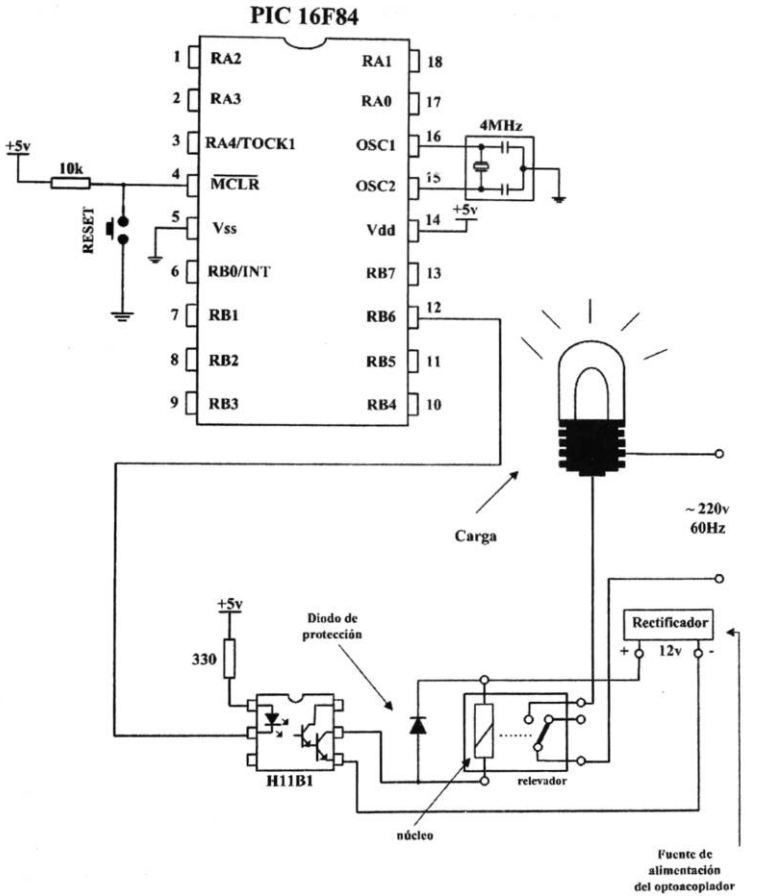


Figura VII.8. Conexión de un relevador y un optoacopaldor

El programa siguiente activa un relevador usando algunos de los Macros ya vistos en esta unidad.

```
;***** Declaración y configuración del microcontrolador *****

        PROCESSOR 16F84
        #include "p16f84.inc"

        __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Declaración de variables *****

        Cblock 0x0c             ;Inicio de la memoria RAM.
        CICLO                   ;Variables del Macro "ESPERA".
        PRESCALAR_2
        endc

;***** Declaración del puerto usado *****

        #define   RELE    PORTB,6     ;El relevador en la línea 6
                                      ;del puerto B.

;***** Estructura de la Memoria de Programa *****

        ORG       0x00        ;Vector de reset.
        goto      Main

        ORG       0x04        ;Vector de interrupción.
        goto      Main        ;No hay rutina de interrupción.

        #include "banking"
        #include "tecla.inc"
        #include "espera.inc"

Main                          ;Inicio del programa principal.
        BANK1
        movlw     0xff        ;Inicializa puerto A.
        movwf     TRISA       ;TRISA ← 0xff, entradas.
        movlw     0x00        ;Inicializa puerto B.
```

```
        movwf   TRISB       ;TRISA ← 0x00, salidas.
        BANK0

        clrf    PORTB       ;Puerto B ← 0.
Loop
        TECLA 0,PORTB,0,100,Enciende
        TECLA 0,PORTB,1,100,Apaga
        goto    Loop

Enciende
        bsf     RELE        ;Enciende el relevador.
        return

Apaga
        bcf     RELE        ;Apaga el relevador.
        return

        End                 ;Fin del programa.
```

VII.10. Los display´s de siete segmentos (multiplexaje)

Los segmentos en un display de 7 segmentos están distribuidos para formar un solo digito del 0 a la F.

Se puede desplegar un número de varios dígitos conectando varios display´s de este tipo y aunque los display´s de LCD son más flexibles, los display´s de 7 segmentos aún son un estándar en la industria debido a su robustez a la temperatura, visibilidad y amplio ángulo de vista. Los segmentos se llaman a, b, c, d, e, f, g y dp, donde dp es el punto decimal (decimal point).

Los 8 LED´s del display pueden estar configurados como cátodo o ánodo común. Con un display de cátodo común, el cátodo debe conectarse a 0V y los LED´s se encienden con uno lógico. En los de ánodo común el ánodo debe conectarse a +5V y los segmentos se encienden con un cero lógico.

El tamaño del display se indica por el tamaño del digito y no el tamaño del encapsulado, midiéndose en milímetros. Los display´s se encuentran disponibles con una altura o tamaño del digito de 7, 10, 13.5, 20 o 25 milímetros. Se encuentran también en diferentes colores. Existen algunos encapsulados que traen hasta 4 display´s con un driver integrado.

De manera alterna, un display lo puede manejar un microcontrolador y si es necesario manejar más de un display puede usarse un método llamado "multiplexaje". La diferencia entre manejar un solo display y un encapsulado de 4 display´s con un driver integrado es el número de líneas usado, ya que el encapsulado de 4 usa solo una línea de reloj y el driver accesa todos los segmentos e incrementa el numero mostrado por el display.

En el método del multiplexaje se usan 7 líneas para los segmentos y una para el punto decimal y por cada display adicional se requiere solo una línea extra. Las líneas de los segmentos y del punto se conectan en paralelo a todos los display´s. La línea común (el cátodo) se conecta separada para cada display y se activa con cero lógico por un corto periodo de tiempo para encender el display. Cada display se enciende a una velocidad de al menos 100 veces por segundo tal que parezca que todos los display´s están encendidos al mismo tiempo. Cuando se encienda cada display debe entregársele el carácter a desplegar.

Se pueden manejar hasta 6 display´s sin afectar la brillantez de cada uno y cada display estará encendido un sexto del tiempo de manera que el ojo humano piense que el display está encendido todo el tiempo. Todas las señales o temporización es producida por el programa de manera que la ventaja de este método es la flexibilidad.

A continuación se muestra un diagrama de cómo manejar dos display´s con el PIC16F84.

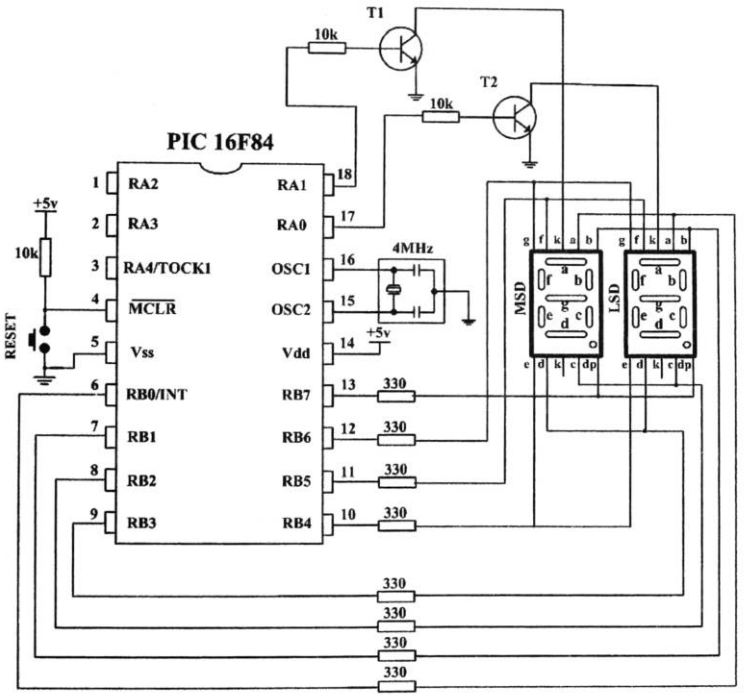


Figura VII.9. Multiplexaje de display´s

En el listado siguiente se encuentran dos Macros (**INIT_DISPLAY7** y **DISPLAY7**). El primero de ellos inicializa todo el ambiente y el hardware para manejar los display´s. En este Macro se definen datos como el periodo de refresco del display y las líneas del microcontrolador usadas para conectar los display´s. El segundo Macro se usa para desplegar los números del 0 al 99 en los dos display´s.

El Macro **DISPLAY7** tiene un argumento ("número") que le indica el número a desplegar en el digito más significativo (**MSD**) y en el menos significativo (**LSD**).

DISPLAY7 numero

Se debe entonces inicializar el timer con una constante de 96 ya que se desea activar cada display 100 veces por segundo, en otras palabras, se necesita encender el display cada 0.010 segundos. (10mS).

En el programa a realizar se encenderá primero el display del digito menos significativo (LSD), 5mS después encenderá el display del digito más significativo (MSD), 5 mS después volverá a encender el display del LSD y así sucesivamente de manera continua. Cada display es activado entonces cada 10mS.

Para obtener un retardo de T = 5mS se debe calcular la cantidad de incrementos del timer para que su cuenta pase de 255 a 0 (sobreflujo):

T = 1 / ([(4Mhz/4) / 32] / incrementos)

incrementos = 1 / ([(4Mhz/4) / 32] / T) = 1 / ([(4Mhz/4) / 32] / 5 mS)

incrementos = 160

Ya que el timer cuenta de manera ascendente, para que realice 160 incrementos se le debe cargar una constante de 256 – 160 = 96

Por otra parte, para determinar cuál de las salidas del Puerto A se debe encender (las salidas del Puerto A están conectadas a los cátodos de los display´s) se realiza una lectura del estado actual de las líneas del Puerto A para determinar cuál display está apagado y cual está encendido en ese momento y poder fijar el estado siguiente de los display´s, usando una tabla de estados del siguiente tipo:

Estado actual		**Estado siguiente**	
Display del digito más significativo (MSD)	Display del digito menos significativo (LSD)	Display del digito más significativo (MSD)	Display del digito menos significativo (LSD)
0	0	0	1
0	1	1	0
1	0	0	1
1	1	0	1

Un "1" lógico indica que el display correspondiente se encuentra encendido y un "0" que el display se encuentra apagado.

Por ejemplo, **DISPLAY7 0x21** desplegará el numero 21 en los display's como lo muestra el programa siguiente:

```
;***** Declaración y configuración del microcontrolador *****

        PROCESSOR 16F84
        #include "p16f84.inc"

        __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Declaración de variables *****

        Cblock 0x0c             ;Inicio de la memoria RAM.
        Encendido               ;Esta variable indica el display que se
                                ;encuentra encendido.
        Digito                  ;Variable que contiene el digito a desplegar.
        endc

;***** Estructura de la Memoria de Programa *****

        ORG     0x00    ;Vector de reset.
        goto    Main

        ORG     0x04    ;Vector de interrupción.
        goto    ISR     ;Salto a rutina de interrupción.

        #include "bank.inc"
        #include "display.inc"

Main
        INIT_DISPLAY
        DISPLAY7 0x21           ;Despliega en los dos display's de
                                ;7 segmentos.

loop    goto loop               ;Permanece en el loop.

        End                     ;Fin del programa
```

```
INIT_DISPLAY   macro
;***** Inicializa los Puertos *****
               BANK1
               clrf   TRISA          ;Puertos A y B como salidas.
               clrf   TRISB
               BANK0
               clrf   PORTA          ;Todas las salidas a "0".
               clrf   PORTB
;
;***** Inicializa el Timer *****
               BANK1
               movlw  B'10000100'    ;Mueve el prescalar (32) al
               movwf  OPTION_REG     ;TMR0.
               BANK0
               movlw  B'00100000'    ;Habilita interrupción del
               movwf  INTCON         ;TMR0, no globalmente.
               movlw  .96
               movwf  TMR0           ;Arranca el Timer.
               bsf    INTCON,GIE     ;Habilita globalmente interrups.
               endm


DISPLAY7 macro     Dato
         movlw     Dato
         movwf     Digito
         call      Despliega
         endm

;***** Rutina de Atención a la Interrupción del Timer *****
;
ISR      bcf       INTCON,GIE
         btfsc     INTCON,GIE
         goto      ISR
         movlw     .96
         movwf     TMR0
         bcf       INTCON,T0IF
         call      Despliega
         RETFIE
```

Despliega

```
        movf    PORTA,W         ;Estado actual de display encendido.
        clrf    PORTA           ;Apaga todos los display´s.
        andlw   0x0F            ;Enmascara parte baja del byte.
        movwf   Encendido

;***** Inicia proceso para determinar cual display se encenderá *****
;***** a continuación basándose en el estado actual            *****

        btfsc   Encendido,0     ;El display del LSD apagado?
        goto    Prueba_MSD      ;No, prueba display del MSD.
        bsf     Encendido,0     ;Si, enciende LSD,
        bcf     Encendido,1     ;apaga MSD y
        goto    Despliega_LSD   ;despliega LSD.

Prueba_MSD
        btfsc   Encendido,1     ;El display del MSD apagado?
        goto    Apaga_MSD       ;No, apaga MSD.
        bcf     Encendido,0     ;Si, Apaga LSD,
        bsf     Encendido,1     ;enciende MSD y
        goto    Despliega_MSD   ;despliega MSD.
Apaga_MSD
        bcf     Encendido,1     ;Apaga MSD.


Despliega_LSD
        movf    Digito,W        ;Mueve a W el numero a desplegar.
        andlw   0x0F            ;Separa el LSD.
        sublw   .9              ;Compara LSD con 9.
        btfss   STATUS,C        ;LSD mayor que 9?
        goto    Mayor_9_LSD     ;Si, LSD es numero invalido.
        movf    Digito,W        ;No, mueve a W solo el LSD.
        andlw   0x0F
        goto    DisplayOut      ;Obtiene código de 7-seg de LSD y
                                ;despliégalo.
Mayor_9_LSD
        movlw   .10             ;Si LSD>9, despliega con todos
        goto    DisplayOut      ;los segmentos apagados.
```

```
Despliega_MSD
        swapf   Digito,1        ;Obtiene solo MSD.
        andlw   0x0F
        sublw   .9              ;Compara MSD con 9.
        btfss   STATUS,C        ;MSD mayor que 9?
        goto    Mayor_9_MSD     ;Si, MSD es numero invalido.
        swapf   Digito,W        ;No, mueve a W solo el MSD.
        andlw   0x0F
        goto    DisplayOut      ;Obtiene código de 7-seg de MSD y
                                ;despliégalo.
Mayor_9_MSD
        movlw   .10             ;Si MSD>9, despliega con todos
                                ;los segmentos apagados.

DisplayOut
        call    Tabla_7Segs     ;Obtiene el código 7-seg del numero
        movwf   PORTB           ;a desplegar y envíalo al Puerto B.
        movf    Encendido,W     ;Por el Puerto A activa solo un
        movwf   PORTA           ;display.
        RETURN

; ***** Códigos de 7 segmentos de los números 0-9 *****
;
Tabla_7Segs
        addwf   PCL,1
        retlw   B'00111111'
        retlw   B'00000110'
        retlw   B'01011011'
        retlw   B'01001111'
        retlw   B'01100110'
        retlw   B'01101101'
        retlw   B'01111101'
        retlw   B'00000111'
        retlw   B'01111111'
        retlw   B'01101111'
        retlw   B'00000000'     ;Numero mayor a 9 se despliegan todos
                                ;los segmentos apagados.
```

NOTAS DEL PIC16F84
PARA LA UEA
SISTEMAS DIGITALES II
Se terminó de imprimir
en el mes de abril del año 2004
en los talleres de la Sección
de Impresión y Reproducción de la
Universidad Autónoma Metropolitana
Unidad Azcapotzalco

La edición estuvo
a cargo de la Sección
de Producción
y Distribución Editoriales

Se imprimieron
100 ejemplares más sobrantes
para reposición.

30 AÑOS

...transformando el diálogo por la razón

NOTAS DEL PIC16F84 PARA LA UEA SISTEMAS DIC
VEGA LUNA * SECCION DE IMPRESION
46462 $ 15.00
$
21-ANTOLOGIAS * 01-CBI

ISBN: 970-31-0259-X
978-97031-02594

UNIVERSIDAD AUTONOMA METROPOLITANA
Casa abierta al tiempo Azcapotzalco

División de Ciencias Básicas e Ingeniería
Departamento de Electrónica
Coordinación de Extensión Universitaria
Sección de Producción y Distribución Editoriales