2008

Microcontroladores

Leonel Pérez – Julio Alvarez D.

Universidad del Azuay

01/01/2008

Tema I: Diseño de Hardware basado en Microcontrolador.

Sumario:

1. Introducción a los Microcontroladores.
2. Características de las líneas de entrada / salida (puertos) en los Microcontroladores Microchip.
3. Conexión de periféricos fundamentales a un Microcontrolador.
4. Discusión del Kit de Entrenamiento.

Microcontroladores vs. Microprocesadores.

En 1971 aparece en el mercado el primer microprocesador (µP) que supuso un cambio decisivo en las técnicas de instrumentación y control. Un microprocesador es un chip programable, que integra pocos recursos de hardware; básicamente los relacionados con el procesamiento de la información (CPU[1]) y con el trabajo aritmético (ALU[2]). Para completar el desempeño de los microprocesadores aparecieron un conjunto de chips periféricos, tales como puertos de entrada / salida, memoria, temporizadores, entre otros. Tales periféricos formaron parte de una familia de chip discretos con los que el µP debía comunicarse empleando básicamente tres tipos de buses[3]: bus de datos, bus de direcciones y bus de control.

En el año 1971, gracias al aumento de la capacidad de integración aparece el primer Microcontrolador (µC). La diferencia fundamental entre un µC con un µP es que el Microcontrolador integra la mayor cantidad de recursos en un solo chip y se comunica con el exterior solamente a través de líneas de entrada / salida o líneas de puerto.

En la actualidad la solución de la mayoría de los proyectos electrónicos es pensada en primera instancia utilizando microcontroladores. Tal es el desarrollo alcanzado por las tecnologías de microcontroladores actuales que el papel de los µP ha sido relegado a la fabricación de PCs o a proyectos de gran escala.

Algunas razones que justifican la elección de un Microcontrolador son las siguientes:

- Bajo costo, puesto que integra muchos de los recursos que en un µP aparecen de forma discreta y se miniaturizan los diseños, lo que supone abaratar costos de fabricación.
- Fiabilidad. Un Microcontrolador integra la mayor parte de los recursos, por lo que se minimizan las interconexiones en la tarjeta de circuito impreso lográndose así un desempeño más fiable.
- Ahorro de tiempo en el desarrollo de los diseños.

[1] CPU son las siglas de: Central Processing Unit.
[2] ALU son las siglas de: Aritmetic Logic Unit.
[3] BUS: se asocia el término a un conjunto de líneas que permiten la comunicación en sistemas digitales.

Características de las líneas de entrada / salida en los Microcontroladores Microchip.

Como ya se había señalado, un Microcontrolador se comunica con el entorno solamente a través de líneas de entrada / salida o puertos que vienen integrados al chip. En este sentido, para realizar la conexión de cualquier dispositivo periférico es imprescindible conocer las características relevantes de los puertos de la tecnología que se trabaje.

Las características ventajosas de las líneas de puerto en los PICS, también han determinado en gran medida la enorme tasa de acogida entre los diseñadores electrónicos. Dentro de las características generales más sobresalientes pudieran citarse las siguientes:

- Pueden manejar hasta 25 mA de corriente, tanto como fuente o sumidero, esto hace que sean capaces de manejar LEDs sin necesidad de Buffers.
- Son configurables individualmente como salidas o entradas, mediante registros[4] denominados "TRIS". Existe un registro TRIS para cada puerto. Cada bit de cada registro está asociado al pin físico del puerto en cuestión. Al escribir un UNO en un bit de un TRIS queda programado el pin correspondiente como una entrada y al escribir un CERO queda programado como salida. Los pines vienen programados por defecto como entradas.
- Muchas de ellas están multiplexadas para realizar una de varias funciones: por ejemplo, una misma línea pudiera ser configurada como entrada ó salida digital ó como una entrada analógica.
- Al igual que otros periféricos, los puertos en los PICS tratan de mantener las mismas características constructivas al migrar de un dispositivo a otro, esto facilita el aprendizaje y manipulación de los chips más modernos.
- Todas las líneas tienen protección contra ESD (Electrostatic Discharge) y en el caso particular de las líneas de puerto tipo "B", existen resistencias de "PULL UP"[5] internas, habilitables o no por S.W. (Software), limpiando el bit 7 de un registro[6].

---

[4] Tales registros se encuentran definidos dentro de la Memoria de Datos (tipo RAM) del Microcontrolador.

[5] Las resistencias de "pull up" son resistores conectados desde un pin o terminal hacia la fuente. Su función es mantener un nivel estable de UNO lógico cuando la línea se programa como entrada y queda desconectada.

[6] Registro "OPTION". Este registro se detalla en la sección "Timers integrados a los PICs de la Familia Media.

Descripción de los Puertos Integrados a la serie 87X.

En la **figura 1.1** se puede apreciar el "pin out" (esquema de terminales) general de los microcontroladores de la serie 87X pertenecientes a la familia media de Microcontroladores Microchip.

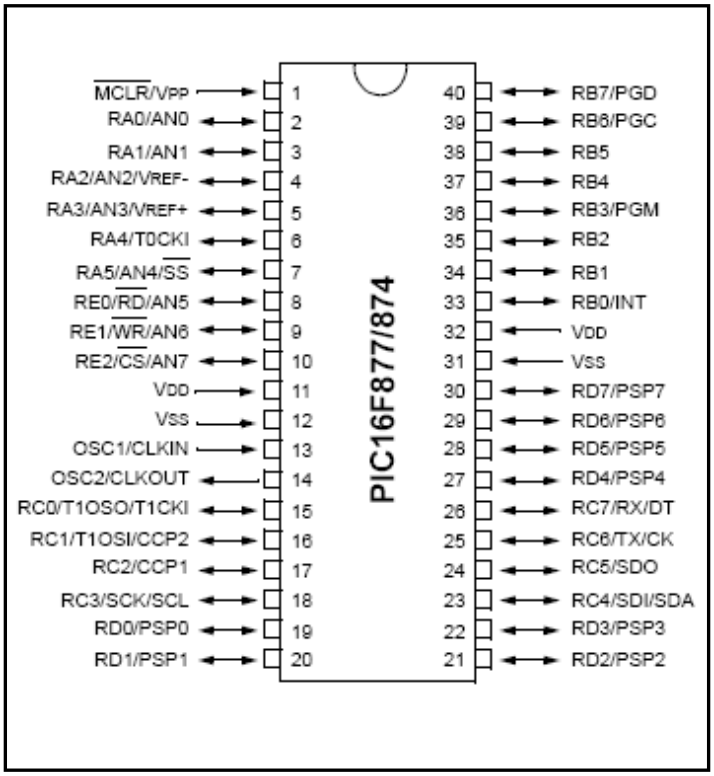


Figura 1.1. Pin out característico de los microcontroladores de la serie 87X.

Todas las líneas del chip son líneas de puerto de propósito general excepto las que se resumen en la **tabla 1.1**:

Nombre del pin	Número del pin	Función
(*) MCLR	1	Pin para generar un RESET externo[7]
OSC 1	13	Pin para la conexión del oscilador externo[8]
OSC 2	14	Pin para la conexión del oscilador externo[9]
VSS	12 ó 31	Pin de referencia. Generalmente conectado a la tierra del sistema
VDD	11 ó 32	Pin de alimentación. Generalmente conectado a + 5V

Tabla 1.1. Pines con funciones Especiales

[7] Refiérase a la sección "Reset"
[8] Refiérase a la sección "Osciladores"
[9] Refiérase a la sección "Osciladores"

Tipos de Puerto. Descripción Funcional.

Puerto A:

Es un puerto de entrada / salida de 6 pines (ra0……ra5) cuyas líneas pueden ser configuradas como entrada(s) / salida(s) digital(es) ó entrada(s)[10] analógica(s); excepto el pin ra4, que es la entrada digital de conteo del timer0[11] cuando este es programado como contador de eventos externos. Los pines del puerto A se numeran en el chip desde el pin 2 hasta el pin 6.

Puerto B:

Es un puerto de entrada / salida digital de 8 bits, con un pull up interno programable por S.W. Los "pull ups" se pueden activar (todos a la vez) limpiando el bit 7 del registro OPTION_REG. El registro OPTION_REG es la localización 0x81 del banco 1 de la Memoria RAM (Memoria de Datos).

Puerto C:

Es un puerto de entrada / salida digital de 8 bits que multiplexa algunas funciones para sus líneas. Algunas de las funciones multiplexadas para estas líneas son las siguientes:

- **Rc2:** puede ser un pin de entrada / salida digital o salida de una onda PWM generada a partir de un recurso de HW denominado modulo CCP.
- **Rc6 y Rc7:** pueden ser pines de entrada / salida digitales o los pines de comunicación para el USART Tx y Rx respectivamente. Como los niveles de salida en estos pines son CMOS (0 =>5V), es necesario conectarlos a un chip que convierta niveles TTL/CMOS a RS232 (ver plano del Kit de Entrenamiento).

Puerto D:

Es un puerto de entrada / salida digital de 8 bits, que multiplexa funciones con el periférico denominado Puerto Paralelo Esclavo (PSP). El PSP no es objetivo de este curso.

Puerto E:

Es un puerto de entrada / salida de 3 pines (re0, re1 y re2) cuyas líneas pueden ser configuradas como entrada(s) / salida(s) digital(es) ó entrada(s) analógica(s).

---

[10] Los pines del puerto A y E; en caso de ser programados como analógicos solo pueden configurarse como entradas.

[11] Refiérase a la sección "Timers"

Circuito Oscilador

La ejecución de un programa en un Microcontrolador es controlada por una referencia de tiempo (oscilador externo o interno en algunos casos). En la mayoría de los casos se emplean componentes externos que conjuntamente con alguna electrónica interna al Microcontrolador conforman un oscilador.

La configuración básica para lograr un circuito oscilador en los microcontroladores de la serie 87X consiste en conectar entre los pines OSC1 y OSC2 un cristal de la manera que se muestra en la **figura 1.2**. La conexión externa del cristal con dos condensadores conectados a tierra forma un oscilador con el inversor integrado al PIC y conectado entre los pines OSC1 y OSC2.

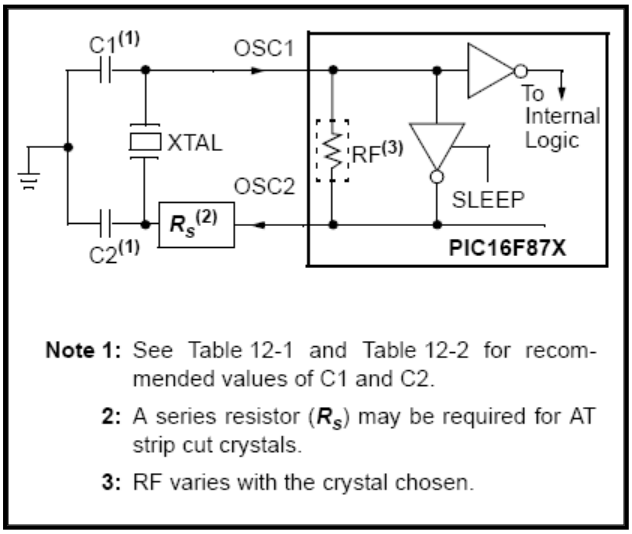


Figura 1.2. Configuración típica para el oscilador principal en los PICs de la familia media.

En general, los tipos de osciladores que puede utilizar esta familia se subdividen en las siguientes categorías:

- **HS:** Oscilador a cristal cuya frecuencia es típicamente mayor a 8 MHz.
- **XT:** Oscilador a cristal cuya frecuencia es típicamente menor a 8 MHz.
- **LP:** Oscilador basado en resonador cerámico o cuarzo de baja potencia, cuya frecuencia esta en el orden de los KHz. Esta modalidad no se utilizara en el curso.
- **RC:** Este tipo de oscilador permite conectar, en lugar del tradicional cristal, una resistencia y un condensador externo al pin OSC1, para formar junto a la circuitería integrada un oscilador de baja frecuencia muy económico pero también muy impreciso. Esta modalidad no se utilizara en el curso.

Una información ampliada puede consultarse en la sección 12 del Datasheet del PIC 16F877: “Special features of the CPU”, “Oscilador Configurations”.

Circuito de RESET externo.

Generar un RESET a un Microcontrolador significa que la ejecución del programa retorna a la primera instrucción (denominada también Vector del Reset). Para generar un RESET externo es necesario llevar a cero durante un determinado tiempo el pin 1 del chip (etiquetado como (*)MCLR). La **figura 1.3** muestra la circuitería necesaria para generar un pequeño pulso (activo en bajo) en el pin (*) MCLR.

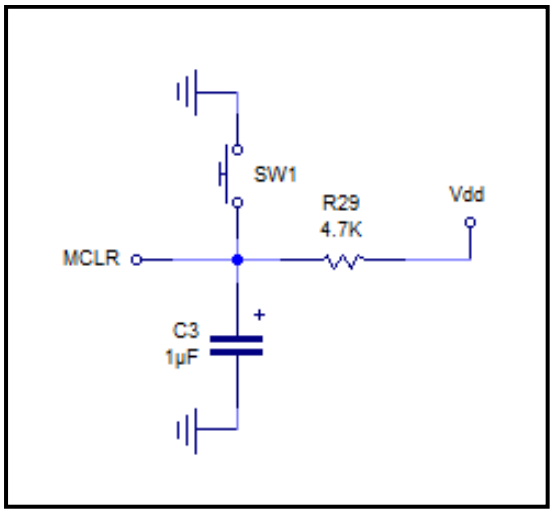


Figura 1.3. Circuito de RESET externo. El pin (*)MCLR se mantendrá en un nivel bajo durante el tiempo de carga del capacitor C3.

Existen otras fuentes de RESET internas en los PICs de esta familia. Para ampliar información puede consultarse el Datasheet del PIC 16F871/877.

Conexión de periféricos fundamentales a un Microcontrolador.

Leds:

Los leds pueden conectarse directamente utilizando una resistencia limitadora para la corriente. Para activar un led es necesario que el pin al cual se conecta sea programado como salida, escribiendo un cero en el bit del registro TRIS correspondiente. Existen 3 modalidades de conexión principales:

- El led se activa con 1 en el pin (**figura 1.4a**):

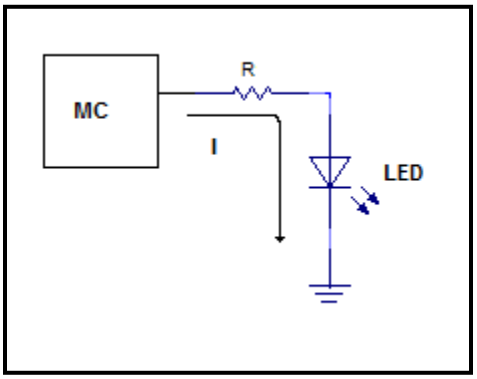


Figura 1.4a. Activación de un led con un "1" lógico.

- El led se activa con 0 en el pin (**figura 1.4b**):

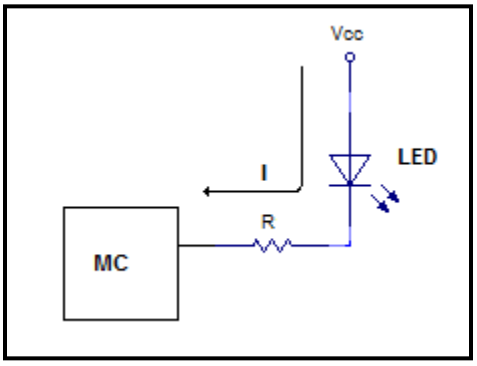


Figura 1.4b. Activación de un led con un "0" lógico.

- Conexión de dos leds a un mismo pin, uno activo con 0 y otro con 1 (**figura 1.4c**):

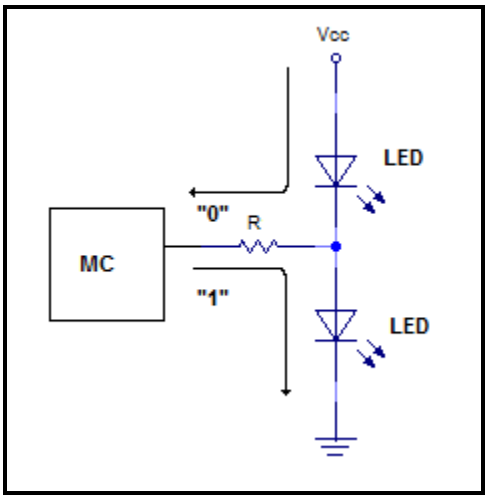


Figura 1.4c. Activación de dos leds con un mismo pin de puerto. El led 1 se activa con "0" y el led 2 se activa con "1".

En todos los casos anteriores, la resistencia R puede calcularse suponiendo una corriente de activación para el led (I) de alrededor de 10 mA y un voltaje de activación para el led de unos 2V. Bajo estas suposiciones:

$$R = \frac{Vcc - 2V}{10\ mA}$$

Si se aproxima el Voltaje de "1" al valor de la fuente (Vcc). Para Vcc = 5V se tiene que:

$$R = \frac{5V - 2V}{10\ mA} = 300\Omega$$

Displays de 7 segmentos:

Un display de 7 segmentos es un arreglo de leds dispuesto como se muestra en la **figura 1.5.** Los segmentos se codifican en sentido horario empleando las letras a, b, c hasta la g y PD (L y/o R) si los displays contienen punto decimal. Cada display posee una terminal común (COM) que sirve para activar o desactivar el display.

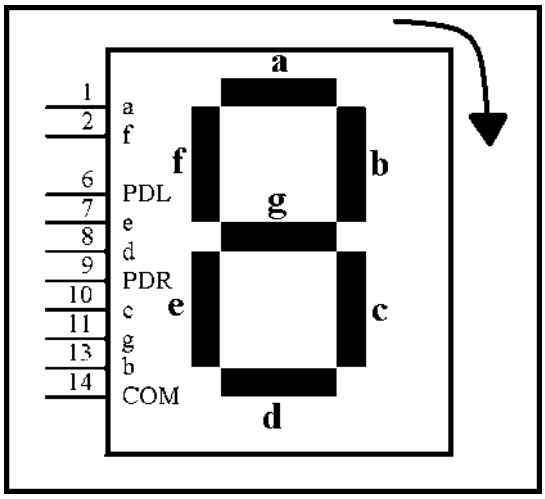


Figura 1.5. Esquema estándar para un Display de 7 segmentos.

Existen dos tecnologías de displays: cátodo común y ánodo común. En el primer caso todos los leds son activos con uno y el común es activo en cero. En el segundo los leds se activan con cero y el común con uno.

Dada la cantidad de corriente que circula por un común, es necesario bufferear las líneas de puerto que activan los comunes empleando un transistor (NPN o PNP) dependiendo de la tecnología tal y como se muestra en la **figura 1.6.**

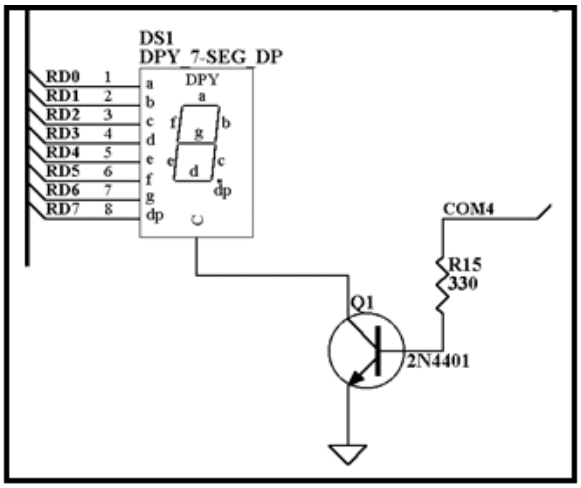


Figura 1.6. Esquema de activación para el común de un Display de 7 segmentos de tecnología cátodo común.

Tradicionalmente los Displays de 7 segmentos de leds eran tratados de forma independiente, utilizando 8 líneas de datos para encender cada led de cada uno. La conexión individual de displays resulta muy engorrosa dada la cantidad de líneas que se involucran.

El empleo de un µC permite implementar el manejo de displays de 7 segmentos multiplexado. El manejo multiplexado supone que se utiliza un único bus de datos (en nuestro caso el puerto D) para enviar el dato en código de 7 segmentos a los displays. Desde el punto de vista del hardware esto significa que los segmentos de los N displays involucrados se conectan en paralelo. Cada segmento (led) debe conectarse a través de una resistencia limitadora (de unos 220Ω). Adicionalmente se necesitan líneas para manejar de manera individual los comunes de cada display.

El método multiplexado enciende en cada momento el display cuyo dato ha sido enviado al bus (puerto D). La secuencia de encendido se realiza a una frecuencia lo suficientemente alta para provocar la sensación de persistencia de los datos. En la **figura 1.7** se esquematiza el método de refrescamiento multiplexado de displays.

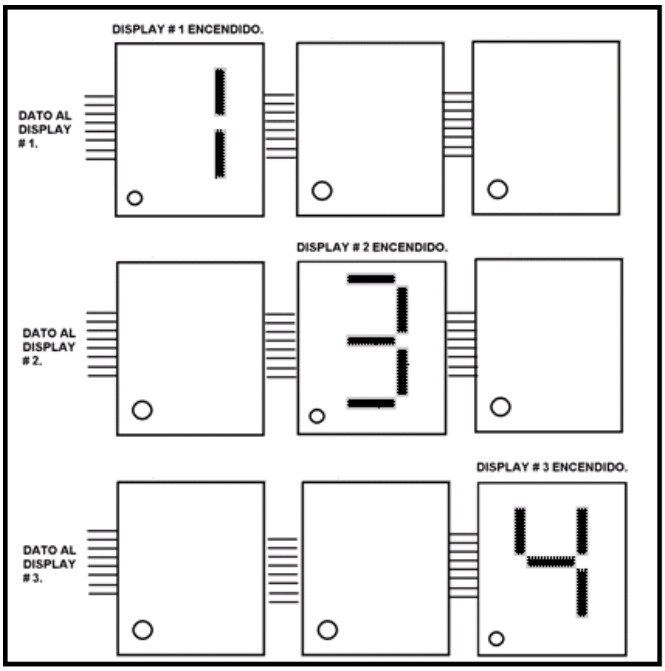


Figura 1.7. Esquema que ilustra el tratamiento multiplexado de displays de 7 segmentos.

Displays de Cristal Líquido (LCD).

Introducción

Los Módulos LCD (Liquid Crystal Display) son dispositivos para visualizar textos y/o gráficos. El empleo de los LCDs es conveniente en diseños que serán producidos a gran escala o cuando la información que se desea mostrar es demasiado compleja.

Los LCDs comerciales consisten del cristal de tinta liquida y un controlador electrónico inteligente, que se monta en una tarjeta PCB, al reverso de cristal. Luego, para realizar una interfaz con un µC es necesario comunicarse con el controlador, ya sea para enviar códigos que representan "ordenes" para en controlador o datos que se desean visualizar. Los códigos para los comandos u órdenes son especificados por el fabricante de cada modelo de LCD.

Existen dos grandes categorías de LCDs en el mercado, los LCDs matriciales y los LCDs gráficos. Es este apartado se ilustra el manejo de un LCD matricial de 2 líneas de 16 caracteres cada una. Cada caracter se forma con una matriz de puntos de 5x7 puntos (**figura 1.8**). Cada matriz, dentro de un LCD posee una dirección especifica, asociada a cada matriz existe una localización de memoria, que puede escribirse o leerse según se desee. Esta memoria se denomina DDRAM ("Display Data RAM"). Los LCDs también poseen una memoria ROM en su tarjeta controladora que almacena caracteres ASCII. La memoria ROM es directamente direccionable, escribiendo el caracter que se desea

visualizar. Antes de la escritura de caracter, debió ser escrita la dirección dentro de la DDRAM, donde se desea visualizar el caracter. Esta información se amplía en la sección "escritura de caracteres a un LCD", de este tema.

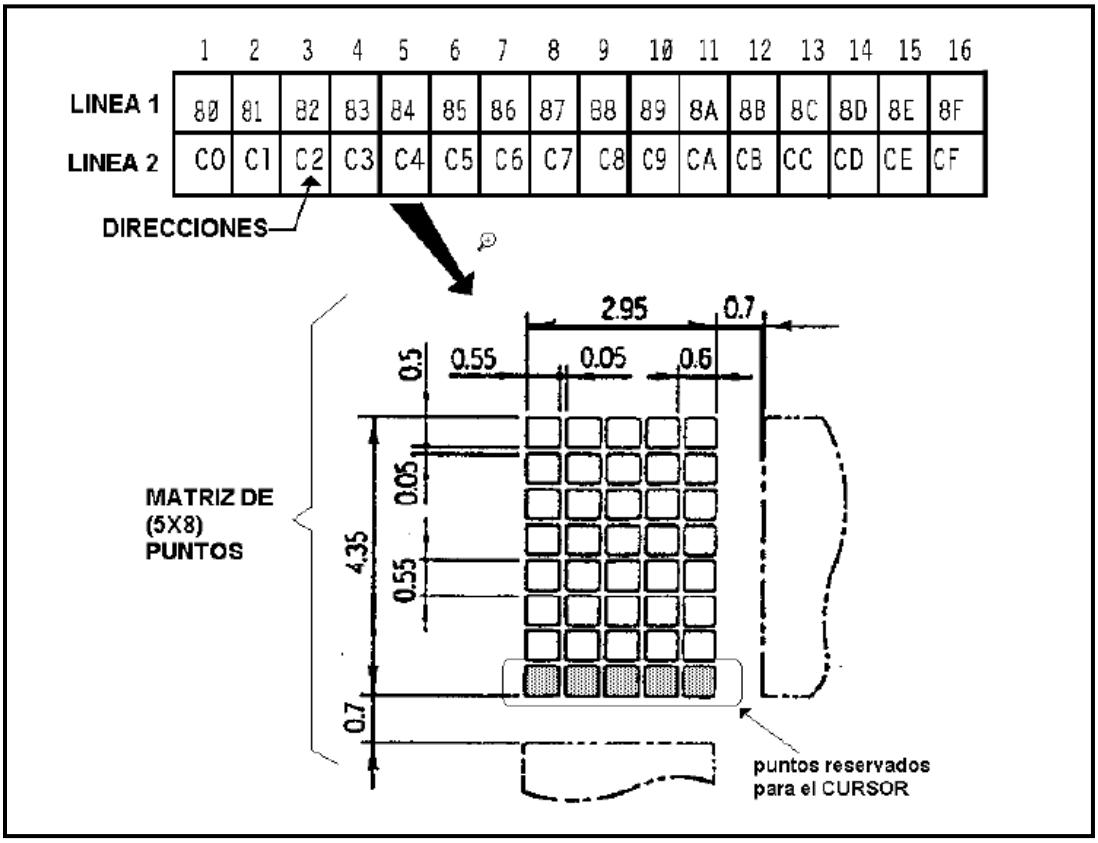


Figura 1.8. Mapa de direcciones de la RAM de Display de un LCD de 2x16 y estructura de matriz.

Modos de operación de un LCD. Interface con el Microcontrolador.

En general todos los LCDs pueden operarse en tres posibles modos (**figura 1.9**):

- Mediante un interface de 4 bits de datos y 3 líneas de control, con transferencia de los datos o instrucciones de 4 bits en 4 bits, empleando la parte alta del puerto conectado al LCD (port <7:4>), **figura 1.9a**.

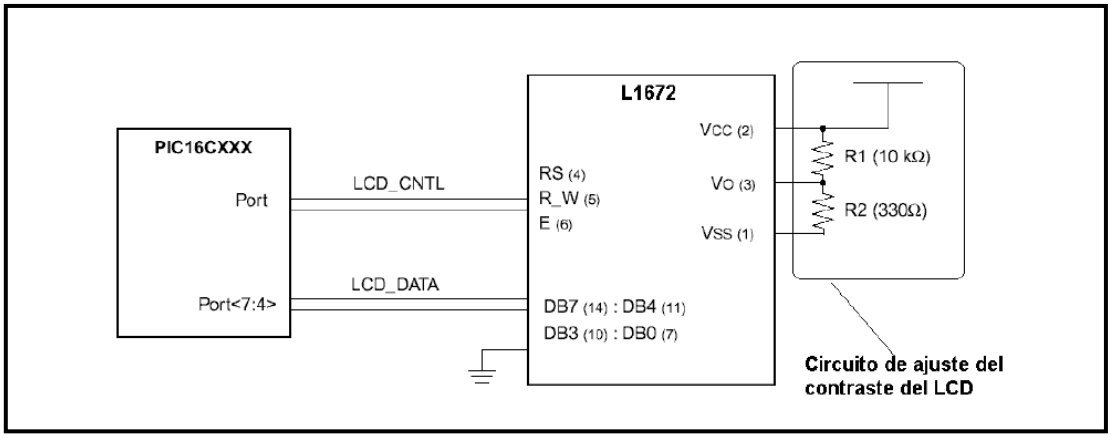


Figura 1.9a. Modo de operación en 4 bits utilizando la parte alta

- Mediante un interface de 4 bits de datos y 3 líneas de control, con transferencia de datos o instrucciones de 4 bits en 4 bits, empleando la parte baja de puerto conectado al LCD (port <3:0>), **figura 1.9b**.

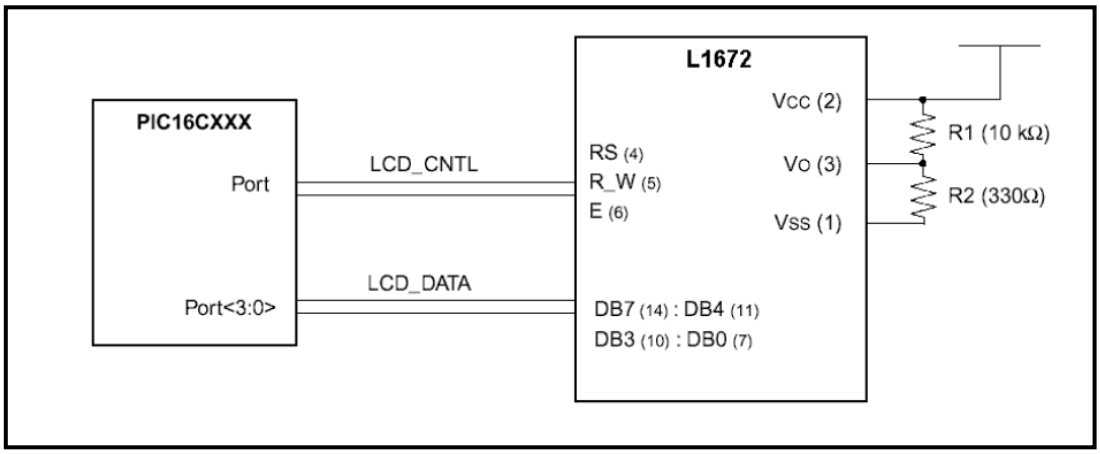


Figura 1.9b. Modo de operación en 4 bits, utilizando la parte baja del puerto.

- Mediante un interface de 8 bits de datos y 3 líneas de control. En este caso tanto los datos como las palabras de instrucción se envían en 8 bits, se emplea todo el puerto, **figura 1.9c**.

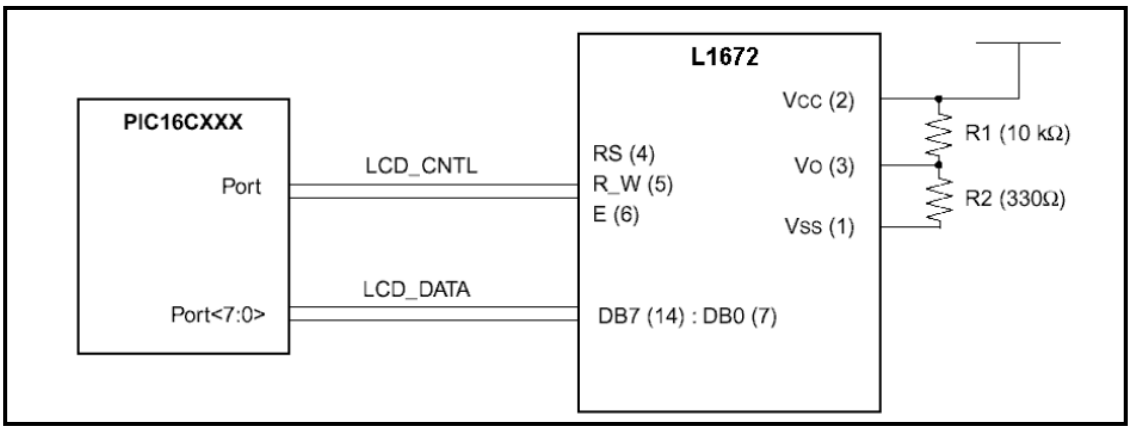


Figura 1.9c. Modo de operación en 8 bits, utilizando completamente el puerto.

Líneas de Control.

La mayoría de LCDs presentes en el mercado se controlan mediante tres líneas: "E", "RS" y "R/*W". La **tabla 1.2** resume las funciones de cada línea de control.

Señal de control	Función
E (Enable)	• Con el frente de subida de esta señal se almacena el estado de los pines RS ("instrucción o dato) y R/W ("lectura o escritura). • Con el frente de caída de esta señal se almacena el dato o la instrucción que se haya colocado en el Bus del LCD. • Para operar el LCD, primero poner el dato (o la instrucción) en el puerto y después dar un pulso al LCD por "E". La duración de este pulso depende de cada cristal. Típicamente se exige como mínimo 250 ns en el alto. En la práctica, 500 ns es suficiente.
RS	"RS" sirve para diferenciar un dato de un comando. • Si RS = 1 significa que la palabra colocada en el Bus del LCD es un dato. • Si RS = 0 significa que la palabra colocada en el Bus del LCD es un comando.
R/*W	R/*W permite realizar lecturas o escrituras al LCD, ya sea de datos o comandos. • R/*W = 0 se va a escribir el LCD • R/*W = 1 se va a leer el LCD

Tabla 1.2. Funciones de las líneas de control de un LCD Matricial.

Teclas (“Push Buttom”).

Las líneas que se utilicen para conectar las teclas deben programarse como entradas, puesto que las teclas son periféricos de entrada al Microcontrolador.

Para conectar teclas resulta excelente el puerto B puesto que posee resistencias de Pull Ups internas como ya se había señalado. En la **figura 1.10** se muestra la conexión directa de pulsantes a través de una resistencia de 220Ω. Las teclas entregan en este caso un nivel bajo cuando son pulsadas y un nivel alto cuando están en estado abierto.

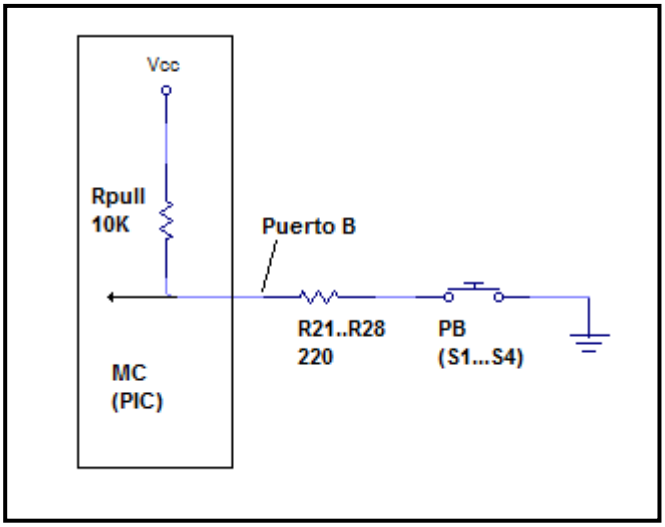


Figura 1.10. Esquema general de conexión de un pulsante tipo “Push Buttom”.

La resistencia de 220 Ω se coloca para proteger la línea de puerto contra cortocircuitos a tierra ante una situación indeseada como la siguiente:

Suponga que por error Ud. programa los pines del puerto B como salidas (registro TRISB = 00000000) y además escribe un “1” en la línea a la cual conecta una tecla. Si la resistencia no estuviera presente, al pulsar la tecla se conectaría a tierra la línea de puerto que en ese momento está a un voltaje alto, por lo que pudiera circular una corriente por ella lo suficientemente alta como para producir daños.

A pesar de la existencia de la resistencia limitadora de 220 Ω, el nivel de “0” no se afecta. Consulte: cálculo del nivel de “0” para tecla activa.

Cargas Optoacopladas.

Se utiliza optoacoplamiento para aislar cargas que pudieran introducir interferencias en el funcionamiento del Microcontrolador. Tal es el caso de los motores, debido al ruido inductivo, o de conmutación en el caso del motor de escobillas.

En otros casos se implementa optoaislamiento cuando no se desea compartir la misma referencia para la tarjeta de control y algún elemento externo por razones de seguridad eléctrica. Tal es el caso de muchos equipos destinados a uso médico o a zonas con alto riesgo de explosión. Seguidamente se ofrecen algunas alternativas de conexión para diferentes tipos de carga:

Cargas en régimen de Corriente Continua.

En la **figura 1.11a y 1.11b** respectivamente se muestra un esquema alternativo para la conexión de una salida y una entrada digital.

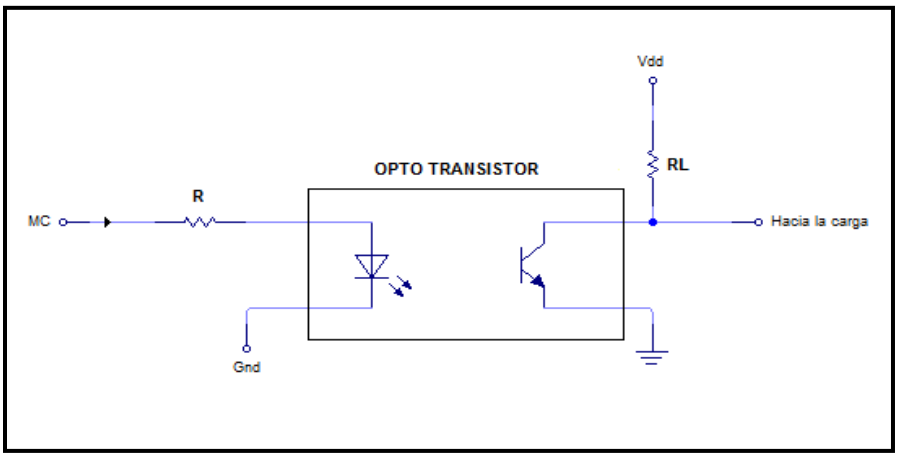


Figura 1.11a. Conexión aislada de un pin de salida que maneja una carga que trabaja en régimen DC.

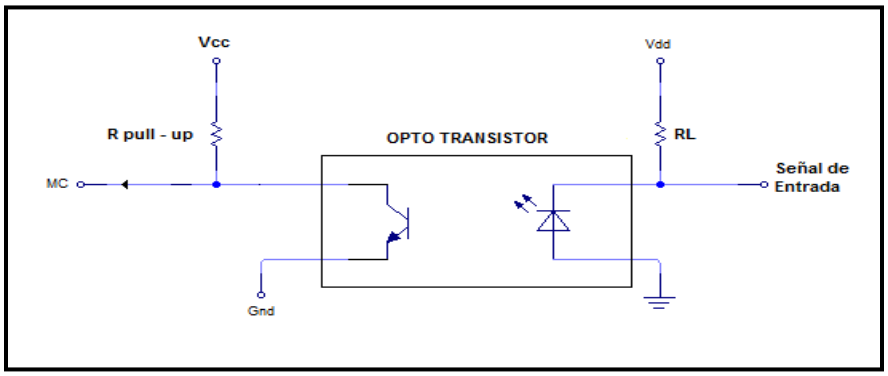


Figura 1.11b. Conexión aislada de una señal a un pin de entrada digital.

En el caso de la salida digital el fototransistor es excitado por la señal de luz que se genera en el led conectado al Microcontrolador. Observe que el "lado del led" comparte la referencia del Microcontrolador y el "lado del fototransistor" comparte la referencia de la carga, por lo tanto este es un esquema de conexión aislada.

Por otra parte la entrada digital es conmutada por el fototransistor, cada vez que es activado por los pulsos de luz que se generan en el led a partir de la señal de entrada. Observe que en este caso el "lado del led" comparte la referencia de la señal de entrada y el "lado del fototransistor" comparte la referencia del Microcontrolador, por lo tanto también este es un esquema de conexión aislada.

Cargas en régimen de C.A.

En la **figura 1.12** se representa un esquema de acople elemental de una salida digital con una carga que trabaja en régimen CA. Es probable que para motores de gran potencia, se necesite un interfaz adicional entre el optotriac y el motor.

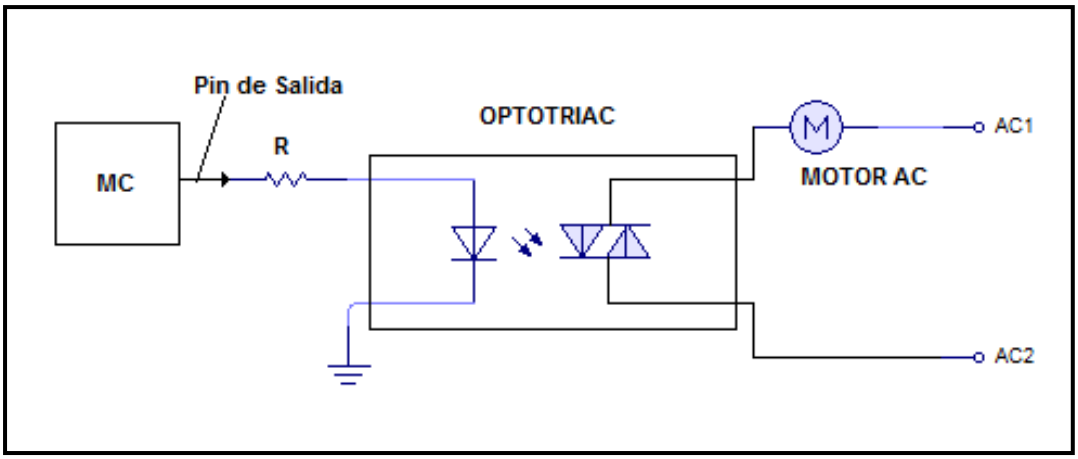


Figura 1.12. Acople con una carga que trabaja en régimen de CA.

Conexión de Relés.

Usualmente la bobina de la mayoría de los relevadores posee baja impedancia y trabaja a un nivel de voltaje superior al de alimentación del Microcontrolador. Estas dos razones hacen necesaria la conexión de relés utilizando transistores, que sirven de buffers y acopladores de nivel. En la **figura 1.13** se muestra el acople de un relé a una salida digital empleando un transistor NPN. La activación del relé se produce en este caso con un "1" lógico.

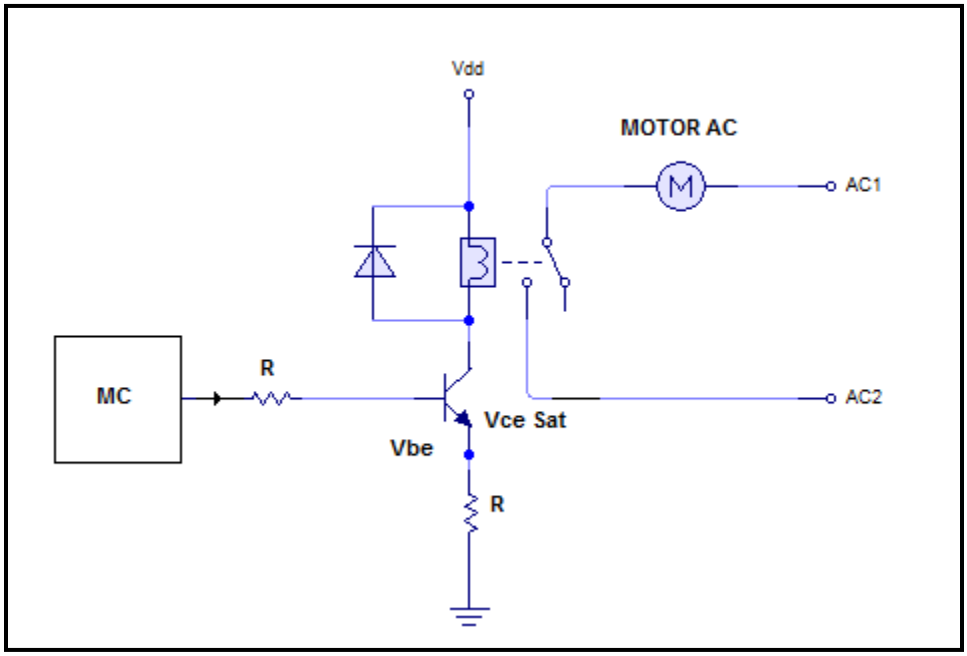


Figura 1.13. Esquema de conexión para relé.

Discusión del Kit de Entrenamiento.

Esquema del Kit de entrenamiento.

A continuación se detalla el esquema (**figura 1.14**) y el listado de recursos del Kit de Entrenamiento (**tabla 1.3**).

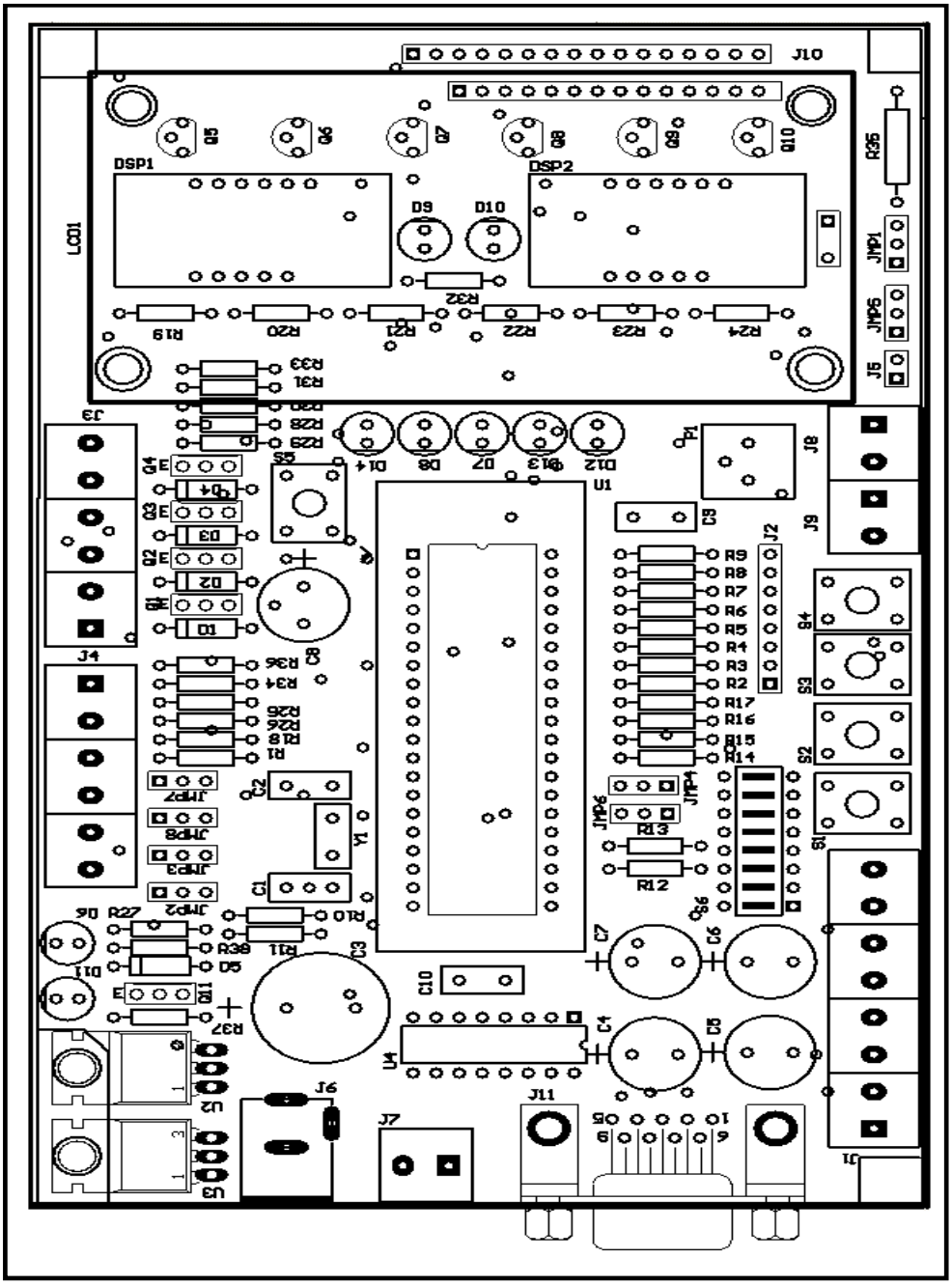


Figura 1.14. Esquema de posición de recursos del Kit de Entrenamiento.

Listado de Recursos del Kit de Entrenamiento.

Valor	Nombre en el Plano	Descripción
0.1UF	C10	Capacitor Cerámico
0.1UF	C9	Capacitor Cerámico
18PF	C1	Capacitor Cerámico
18PF	C2	Capacitor Cerámico
1UF	C8	Capacitor Electrolítico
470UF/35V	C3	Capacitor Electrolítico
10UF/16V	C7	Capacitor Electrolítico
10UF/16V	C6	Capacitor Electrolítico
10UF/16V	C4	Capacitor Electrolítico
10UF/16V	C5	Capacitor Electrolítico
1N4004	D5	Diodo Rectificador
1N4007	D2	Diodo Rectificador
1N4007	D1	Diodo Rectificador
1N4007	D4	Diodo Rectificador
1N4007	D3	Diodo Rectificador
2N3904	Q7	Transistor NPN de Pequeña señal
2N3904	Q8	Transistor NPN de Pequeña señal
2N3904	Q10	Transistor NPN de Pequeña señal
2N3904	Q9	Transistor NPN de Pequeña señal
2N3904	Q6	Transistor NPN de Pequeña señal
2N3904	Q5	Transistor NPN de Pequeña señal
4.7K	R36	Resistor 1/4W
10K	R1	Resistor 1/4W
10K	P1	Potentenciometro tipo trimmer
16.000MHZ	Y1	Cristal Piezoeléctrico
36 Ohmios / 1/4 W	R35	Resistor 1/4W
220	R10	Resistor 1/4W
220	R11	Resistor 1/4W
220	R14	Resistor 1/4W
220	R2	Resistor 1/4W
220	R17	Resistor 1/4W
220	R16	Resistor 1/4W
220	R12	Resistor 1/4W
220	R15	Resistor 1/4W
220	R8	Resistor 1/4W
220	R9	Resistor 1/4W
220	R6	Resistor 1/4W
220	R7	Resistor 1/4W
220	R4	Resistor 1/4W
220	R3	Resistor 1/4W
220	R5	Resistor 1/4W
220	R13	Resistor 1/4W
330	R25	Resistor 1/4W
330	R22	Resistor 1/4W
330	R23	Resistor 1/4W

330	R24	Resistor 1/4W
330	R21	Resistor 1/4W
330	R34	Resistor 1/4W
330	R19	Resistor 1/4W
330	R20	Resistor 1/4W
330	R27	Resistor 1/4W
330	R30	Resistor 1/4W
330	R32	Resistor 1/4W
330	R29	Resistor 1/4W
330	R31	Resistor 1/4W
330	R28	Resistor 1/4W
330	R26	Resistor 1/4W
330	R33	Resistor 1/4W
330	R18	Resistor 1/4W
470	R38	Resistor 1/4W
560	R37	Resistor 1/4W
LED amarillo	D13	Diodo LED
BD135	Q11	Transistor NPN de Media Potencia
BD135	Q2	Transistor NPN de Media Potencia
BD135	Q1	Transistor NPN de Media Potencia
BD135	Q4	Transistor NPN de Media Potencia
BD135	Q3	Transistor NPN de Media Potencia
CON2	J9	UNA Bornera de 2 pines
CON2	J7	UNA Bornera de 2 pines
CON2	J8	UNA Bornera de 2 pines
CON2	J5	Peineta de DOS pines (cortar de una tira)
CON2	J6	Jack para Conexión de Adaptador (AC/DC)
CON6	J3	TRES Borneras de 2 pines
CON6	J4	TRES Borneras de 2 pines
CON8	J1	CUATRO Borneras de 2 pines
CON8	J2	UNA Bornera de 2 pines
CON16	J10	Peineta Hembra o mitad de sócalo plástico DIP
DB9	J11	Conector DB9 para PCB hembra
BT-M405RE-002	DSP1	Display de 7 Segmentos de 3 Dígitos Multiplexado
BT-M405RE-002	DSP2	Display de 7 Segmentos de 3 Dígitos Multiplexado
JUMPER	JMP6	PEINETA (DOS PINES)
JUMPER	JMP7	PEINETA (DOS PINES)
JUMPER	JMP5	PEINETA (DOS PINES)
JUMPER	JMP4	PEINETA (DOS PINES)
JUMPER	JMP1	PEINETA (DOS PINES)
JUMPER	JMP2	PEINETA (DOS PINES)
JUMPER	JMP3	PEINETA (DOS PINES)
LCD	LCD1	LCD Matricial de 2 X 16 Líneas (Backlight Amarillo o Rojo)
LED rojo	D14	Diodo LED
MAX232ACSE(16)	U4	Transceiver TTL - RS232
PIC16F871	U1	MICROCONTROLADOR FLASH DE 8 BITS
PULS PCB	S3	Pulsante PCB

PULS PCB	S4	Pulsante PCB
PULS PCB	S1	Pulsante PCB
PULS PCB	S2	Pulsante PCB
PULS PCB	S5	Pulsante PCB
LED Rojo	D10	Diodo LED
LED Rojo	D9	Diodo LED
LED Rojo	D7	Diodo LED
LED Rojo	D6	Diodo LED
LED Rojo	D8	Diodo LED
SW DIP-8	S6	DIP Switch de 16 pines
UA7812	U2	Regulador Monolítico Integrado (Encapsulado TO-220)
Led Verde	D11	Diodo LED
Led Verde	D12	Diodo LED
uA7805	U3	Regulador Monolítico Integrado (Encapsulado TO-220)
Sócalo ZIP	******	Sócalo ZIP de Palanca 40 pines (opcional)
Sócalo DIP	******	Sócalo DIP (ANCHO) Plástico 40 pines
Tec. Mat.	******	Teclado Matricial de 16 teclas
Adapt. AC/DC	******	Adaptador (fuente) AC/DC 16V Salida Mínimo.

Tabla 1.3. Listado de recursos del Kit de Entrenamiento.

Distribución de los recursos en el Kit.

Las siguientes tablas explican la distribución de los recursos en el PCB y su correspondiente relación con los pines del Microcontrolador.

Puerto	Denominación del pin.	Función	Conector Asociado.	Pin #
A	Ra0	Entrada Analógica o E/S Digital Externa.	J4 (1)	2
	Ra1	Entrada Analógica o E/S Digital Externa.	J4 (1)	3
	Ra2	Entrada Analógica o E/S Digital Externa.	J4 (1)	4
	Ra3	Entrada Analógica o E/S Digital Externa.	J4 (1)	5
	Ra4	Control de LCD (RS)	J4 (1)	6
	Ra5	Entrada Analógica o E/S Digital Externa.	J4 (1)	7
B	Rb0	Puerto para conexión de pulsantes, Dip Switch, Teclado Matricial (externo) ó E/S Digital Externa con pull – ups internos.	J1 (1)	33
	Rb1	Puerto para conexión de pulsantes, Dip Switch, Teclado Matricial (externo) ó E/S Digital Externa con pull – ups internos.	J1 (2)	34
	Rb2	Puerto para conexión de pulsantes, Dip Switch, Teclado Matricial (externo) ó E/S Digital Externa con pull – ups internos.	J1 (3)	35
	Rb3	Puerto para conexión de pulsantes, Dip Switch, Teclado Matricial (externo) ó E/S Digital Externa con pull – ups internos.	J1 (4)	36
	Rb4	Puerto para conexión de pulsantes, Dip Switch, Teclado Matricial (externo) ó E/S Digital Externa con pull – ups internos.	J1 (5)	37

	Rb5	Puerto para conexión de pulsantes, Dip Switch, Teclado Matricial (externo) ó E/S Digital Externa con pull – ups internos.	J1 (6)	38
	Rb6	Puerto para conexión de pulsantes, Dip Switch, Teclado Matricial (externo) ó E/S Digital Externa con pull – ups internos.	J1 (7)	39
	Rb7	Puerto para conexión de pulsantes, Dip Switch, Teclado Matricial (externo) ó E/S Digital Externa con pull – ups internos.	J1 (8)	40
C	Rc0	Pin de Control LCD (E)	N.A	15
	Rc1	E/S de Módulo CCP2	J6 (1)	16
	Rc2	E/S de Módulo CCP1	J7 (1)	17
	Rc3	Control de Común Display de 7 Segmentos L1.	N.A	18
	Rc4	Control de Común Display de 7 Segmentos L2.	N.A	23
	Rc5	Control de Común Display de 7 Segmentos L3.	N.A	24
	Rc6	Pin Tx USART ó CK Puerto Sincrónico.	J8 (1)	25
	Rc7	Pin Rx USART ó CK Puerto Sincrónico.	J8 (2)	26
D	Rd0	D0 Bus de Datos LCD / Display de 7 Segmentos / Control de Transistor de Potencia Q1 (jumper)	JMP1 (1)	19
	Rd1	D1 Bus de Datos LCD / Display de 7 Segmentos / Control de Transistor de Potencia Q2 (jumper)	JMP3 (1)	20
	Rd2	D2 Bus de Datos LCD / Display de 7 Segmentos / Control de Transistor de Potencia Q3 (jumper)	JMP2 (1)	21
	Rd3	D3 Bus de Datos LCD / Display de 7 Segmentos / Control de Transistor de Potencia Q1 (jumper)	JMP5 (1)	22
	Rd4	D4 Bus de Datos LCD / Display de 7 Segmentos	N.A	27
	Rd5	D5 Bus de Datos LCD / Display de 7 Segmentos	N.A	28
	Rd6	D6 Bus de Datos LCD / Display de 7 Segmentos	N.A	29
	Rd7	D7 Bus de Datos LCD / Display de 7 Segmentos	N.A	30
E	Re0	Control de Común Display de 7 Segmentos L4	N.A	8
	Re1	Control de Común Display de 7 Segmentos L5	N.A	9
	Re2	Control de Común Display de 7 Segmentos L6	N.A	10

Tabla 1.4 Distribución de Recursos en la Tarjeta de Entrenamiento.

Pines con Funciones Especiales		
Denominación y número	**Función**	**Componentes Asociados**
1 (MCLR)	Reset Externo	C7, R36, S6
12, 31 (VSS)	Vss (conectado a tierra)	N.A
11, 32 (VCC)	Vcc (conectado a + 5V)	U3 (MC7805K)

Tabla 1.5 Pines con Funciones Especiales.

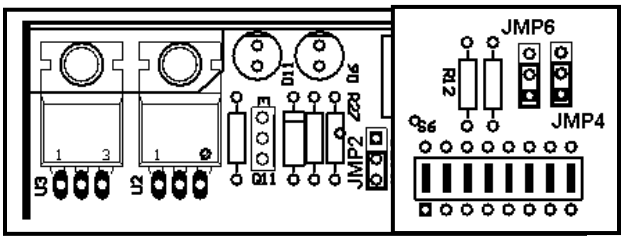
Prueba funcional del Kit de Entrenamiento.

Para probar el Kit de Entrenamiento a partir de la ejecución de un Programa de Pruebas; siga los siguientes pasos:

1. **Grabar el Microcontrolador** con el código objeto derivado del archivo "testkit7.asm".

2. Garantizar las siguientes **condiciones iniciales para el hardware**:

   - Colocar los Jumpers JMP2, JMP3, JMP6 y JMP4, como se indica en las **figuras 1.15 y 1.16**. Esto garantizará que queden conectados inicialmente los leds conectados a RD0, RD1, RD2 y RD3.



Figuras 1.15 y 1.16 Posicionado de Jumpers para leds conectados al PUERTO D.

   - Colocar los DIP Switches hacia la posición de OFF, tal y como se indica en la **figura 1.17**. Esto garantizará la realización de prueba de pulsantes, conectados a los pines RB0 … RB3.

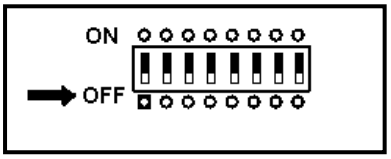


Figura 1.17. Posición inicial (OFF) de los DIP Switches.

- **Colocar el Jumper JMP5 hacia la posición de VCC (derecha)** si se han montado Displays de 7 Segmentos de Tecnología Ánodo Común o hacia la posición izquierda si se han montado Displays de 7 Segmentos de Tecnología Cátodo Común. JMP5 actúa como selector de alimentación para los Displays de 7 Segmentos; lo que permite montar sobre la tarjeta displays de tecnologías diferentes; siempre y cuando la distribución de pines de los mismos sea compatible con la huella en la Tarjeta PCB. En la **figura 1.18** se muestra el Jumper **JMP5** configurado para **Ánodo Común**.

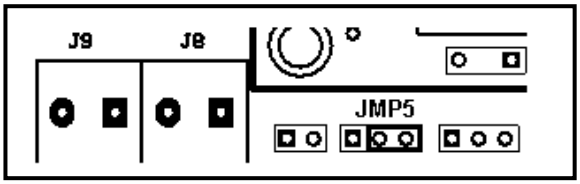


Figura 1.18. Posición del Jumper JMP5 para tecnología de Displays de 7 Segmentos de Ánodo Común.

- **Desconectar totalmente el jumper JMP1**. El jumper JMP1 (a la derecha de JMP5) sirve solamente para conectar o desconectar la tierra del LCD. De esta forma se conecta o no la tierra al LCD.

3. **Energizar la tarjeta**. Al energizar la tarjeta el Programa de Pruebas hará lo siguiente:

    - Rotará los LEDS conectados al puerto D y encenderá los LEDS conectados al puerto C. **La rotación termina con el encendido del LED D12; durante un tiempo que debe ser aprovechado para cambiar la posición de los jumpers JMP2, JMP3, JMP6 y JMP4 a la posición contraria a la indicada en las figuras 1.15 y 1.16**.

    - **Se encenderán en forma rotatoria cada uno de los segmentos del Displays de 7 Segmentos.** Observar que todos estén en buen estado.

    - El encendido rotatorio terminará con la iluminación de todos los segmentos G de los displays indefinidamente en espera de que sean presionados los pulsantes conectados al puerto B.

    - Presionar en orden los pulsantes conectados al puerto B. Cada vez que sea presionado un pulsante se visualizará un cartel señalizando que el pulsante está en buen estado. El proceso se realizará para cada pulsante, luego de lo que se visualizará temporalmente un mensaje de **"Pulsantes OK".** El test de los pulsantes terminará de manera similar al test de los segmentos.

- **Test de los Switches.** Ir activando cada uno de los switches (desde rb0 .. hasta rb7) sin presionar los pulsantes. En el display se visualizará un mensaje para cada switch hasta llegar a 7, luego de lo cual se visualizará un mensaje de "SW OK" y se pagarán las lámparas.

- **Test del LCD.** Desconecte totalmente el jumper JMP5. De esta manera quedarán sin alimentación los Displays de 7 Segmentos.

- Conecte el LCD vigilando la correcta posición de pines.

- Active el jumper **JMP1** (tierra del LCD). **Abra todos switches que Ud. había cerrado para dar paso al Test del LCD.** En principio al conectar el LCD pudieran aparecer cuadros negros en la primera línea. Las causas pueden ser: no se ha iniciado el test del LCD; JMP1 desconectado; demasiado contraste **u olvido abrir el switch**. Las soluciones son:

  - ❑ Esperar.
  - ❑ Conectar JMP1.
  - ❑ Mover el potenciómetro de contraste.
  - ❑ **Abrir los Switches**.
  - ❑ Cambiar LCD (caso muy raro).

  El test terminará visualizando: "PIC Trainer UDA" … "End Test" … "Conecte Teclado:".

- **Test del Teclado**. Conecte el teclado matricial a las borneras del conector J1 (conector externo del puerto B). El teclado matricial debió haber sido cableado correctamente; de lo contrario aparecerán cambiados los caracteres en el display.

- Después de realizar el Test del Teclado aparecerá el mensaje **"End Test" y "Shutting Down".** El LCD finalmente debe quedar en blanco y con el cursor en la esquina superior izquierda.

Tema II: Introducción a la Arquitectura Interna de la Familia Media de Microcontroladores Microchip™.

Sumario:

1. Arquitectura Interna de los Microcontroladores Microchip™ (serie 87X).
2. Memoria de Datos.
3. Memoria de Programas.

Arquitectura Interna de los Microcontroladores Microchip™ (serie 87X).

Introducción.

En el mundo existen más de 50 fabricantes de microcontroladores. Evidentemente, en estas condiciones es muy difícil elegir "el mejor". Realizando un análisis justo, "el mejor", en este tipo de tecnología no existe, porque cada aplicación generará necesidades especificas. Ahora bien Microchip, es la empresa que ha logrado reunir en sus microcontroladores la mayor cantidad de características ventajosas para satisfacer numerosas necesidades en diferentes tipos de aplicaciones. Algunas de las razones por las que los PICs han sido aceptados ampliamente, son las siguientes:

- Poseen una relación costo-desempeño relativamente baja, comparada con la de sus competidores.
- Trabajan a velocidades grandes: hasta 20 MHz la serie 87X y hasta 40 MHz la última serie (18FXXX).
- Hasta la gama media, poseen Arquitectura RICS, o sea manejan un Set Reducido de Instrucciones. Posteriormente se analizará por qué es posible esto y cuáles son sus ventajas.
- Amplia variedad de CHIPS. Microchip ha optimizado la variedad de periféricos que incluye en sus microcontroladores, con el objetivo de que el diseñador encuentre el óptimo para su aplicación y no carezca o derroche recursos electrónicos en el CHIP. Microchip oferta pequeños microcontroladores de 8 pines muy poderosos. Existen series muy baratas y con gran desempeño, como la de los PICs 16F87X y series ultra poderosas, que pueden inclusive emular el modo de trabajo de microprocesadores como la serie 18XXX.
- Otra gran ventaja de Microchip, es la migrabilidad del código, entre diferentes tipos de CHIPS y familias, esto permite que un programa escrito para correr en un PIC 12C508 de 8 pines sea fácilmente reelaborado para correr en un PIC 16C57 (por ejemplo).
- Tecnología fácil de estudiar. Ud. puede estudiar Microchip de manera modular. Para que se comprenda lo anterior se ilustra a través de un ejemplo: un PIC 16F84, posee los módulos básicos: puerto A, puerto B, TMR0 y EEPROM de datos y es capaz de manejar 4 fuentes de interrupciones. Un PIC 16F870 posee los mismos módulos y maneja también estas 4 fuentes de interrupción, pero adicionalmente posee dos timers, un modulo CCP, un puerto serie, un conversor A/D de 10 bits y maneja 7 fuentes de interrupciones adicionales, lo que quiere decir que si se

asimiló el trabajo con los módulos incluidos en un F84, solo se deberá estudiar en el 870, los módulos adicionales que aparecen. Adicionalmente, se facilita el diseño de firmware porque ambos micros manejan el mismo "Set" de instrucciones.

- Existe gran variedad de Herramientas de Desarrollo, proporcionadas por Microchip o empresas alternativas. Gran cantidad de herramientas de SW para el desarrollo las proporciona Microchip de manera gratuita, además de una gran cantidad de Notas de Aplicación, que merecen una profunda consulta.

Características generales de la Arquitectura Interna de los Microcontroladores PICs.

Las mejoras introducidas en el desempeño de los PICs se deben principalmente a la estructura RISC de su Arquitectura. La Arquitectura RISC (Reduced Instruction Set Computer), hace posible el manejo de un Set de Instrucciones Reducido, a diferencia de los procesadores CISC (Complex Instruction Set Computer). Es necesario entender, que los PICs pueden manejar un Set corto de instrucciones gracias a su arquitectura, no de manera arbitraria.

En los procesadores de tipo RISC, encontramos comúnmente las siguientes características:

- Arquitectura tipo HARDVARD.
- Palabras de instrucción largas.
- Instrucciones codificadas en una sola palabra.
- Posibilidad de multitarea, porque pueden realizar la canalización de las instrucciones.
- Arquitectura "registro-fichero" y Simetría de Instrucciones.

"Arquitectura tipo HARDVARD", "palabras de instrucción largas" y "canalización de instrucciones".

Este tipo de arquitectura supone la separación de los Buses que comunican la Memoria de Programas y la Memoria de Datos, a diferencia de la tradicional arquitectura tipo Von Neuman (**figura 2.1**). La separación de buses posee estas importantes ventajas:

- Acceder a la Memoria de Datos (MD) y la Memoria de Programas (MP) al mismo tiempo, esto permite realizar multitarea, esto es: a la misma vez que se busca el código de una instrucción es posible ejecutar la que ya se decodificó. A este proceso se le denomina: canalización ("pipeling") de instrucciones.
- Como los buses son separados, es viable construir un bus para la Memoria de Programas más extenso, esto hace posible que el código de las instrucciones

pueda ser grabado en la Memoria de Programas en una sola palabra de longitud mayor a 8 bits (longitud tradicional), lo que es ventajoso porque de esta manera es posible decodificar toda las instrucción en un solo acceso a la MP, pues la palabra del código se lee de una vez. Esto es codificación en una sola palabra.

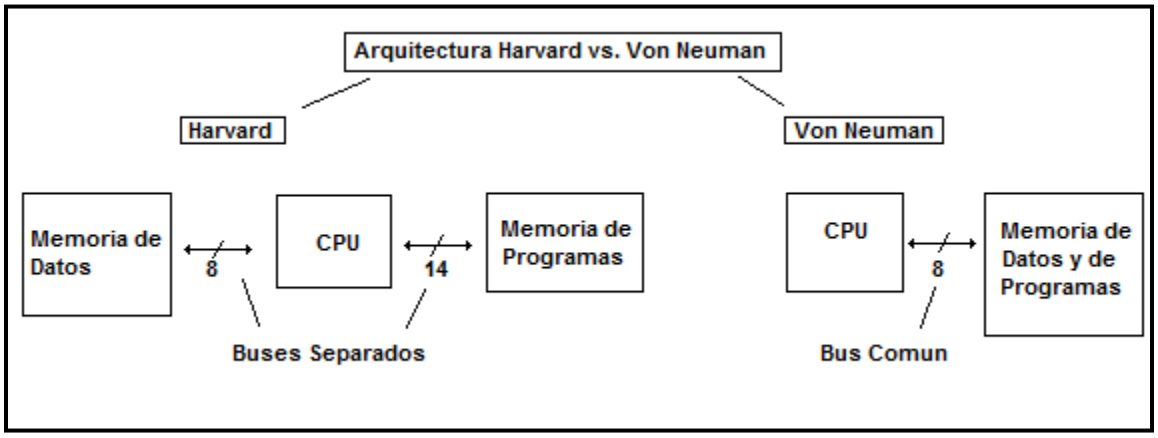


Figura 2.1. Arquitectura tipo Hardvard vs. Arquitectura tradicional "Von Neuman".

Simetría (Ortogonalidad de las Instrucciones).

Un set de instrucciones es simétrico cuando con una misma instrucción es posible direccionar y actuar, ya sea, sobre una localización de RAM donde se guardan datos o sobre un registro de propósito específico. Los registros en la Memoria de Datos, se dividen en dos categorías: "de Propósito General" (GPRs) y "de Propósito Especifico" (SFRs). Los primeros se utilizan para almacenar datos durante la corrida del programa y los segundos almacenan información que compromete el trabajo de los periféricos integrados al Microcontrolador.

La simetría u ortogonalidad es indispensable para lograr un "Set" de Instrucciones tipo RISC. La simetría es posible gracias a un tipo de arquitectura denominada "registro – fichero", que permite direccionar de igual manera un registro de propósito especifico y uno de propósito general. En temas posteriores se complementa la información aquí descrita.

Ciclo de Instrucción y Ciclo de Máquina en Microprocesadores PICs.

Un "Ciclo de Instrucción" puede entenderse como el tiempo que transcurre desde que un comando del programa es leído por el CPU, decodificado y ejecutado. Un Ciclo de Instrucción puede estar forma por uno o más Ciclos de Máquina. Un Ciclo de Máquina se refiere al tiempo que dura la ejecución de cualquier proceso que forma parte de un Ciclo de Instrucción.

En los PICs el reloj de la entrada (el que se conecta a los pines 13 y 14, en la serie 87X y 18F4XX) se divide en 4 para generar 4 relojes de cuadratura denominados Q1, Q2, Q3 y Q4. En el intervalo de tiempo de Q1 hasta Q4 se busca la instrucción N y se almacena su código en el registro de instrucción (dentro del CPU para decodificarla), mientras que a la misma vez se ejecuta la instrucción N-1, o sea la que se había buscado en el espacio de tiempo Q1-Q4 anterior (**figura 2.2**).

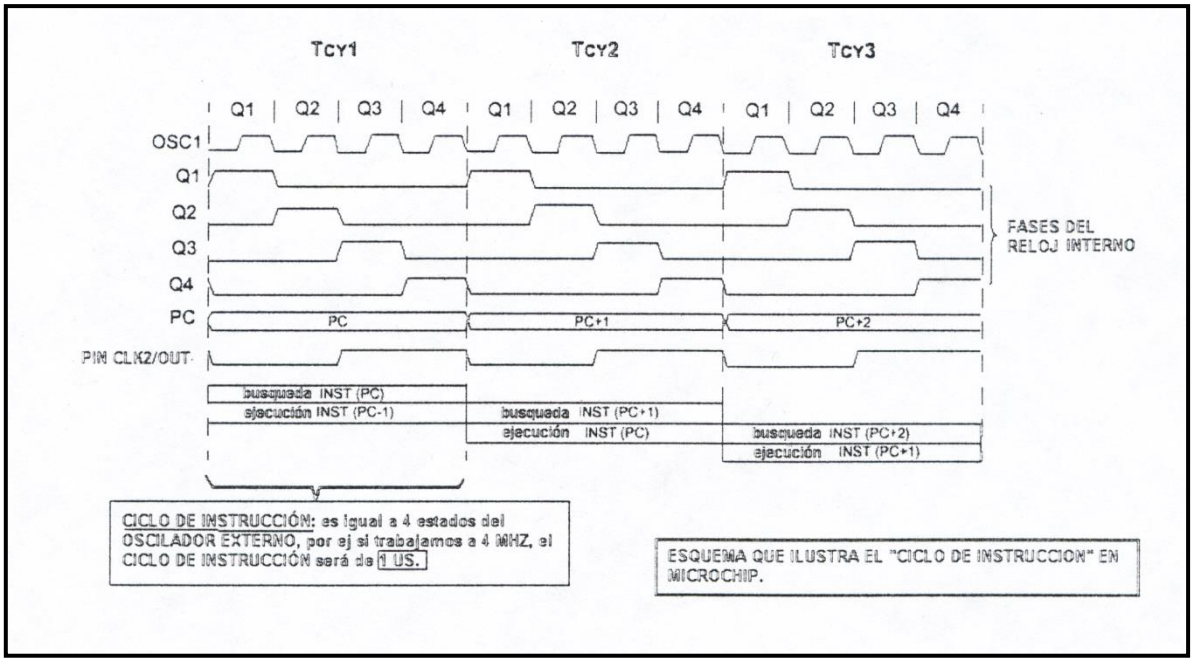


Figura 2.2. Ciclo de Instrucción en Microchip.

Duración del Ciclo de Máquina y Canalización "Pipelining" en Microchip.

En los PICs el Ciclo de Instrucción es igual a un único Ciclo de Máquina, que consiste en 4 estados "Q". Físicamente esto significa que la mayoría de las instrucciones se ejecutaran en un tiempo igual a 4 veces el periodo del oscilador externo: por ejemplo, si se trabajara a 4 MHz (T = 0.25µS), se obtuviera un Ciclo de Instrucción igual a 1 µS, en cambio, si el cristal se lo cambia a 12 MHz se obtuviera un Ciclo de Instrucción de 0.33333 µS. Este proceso, evidentemente hace más eficiente el desempeño del procesador y se realiza gracias a la canalización "pipelining" de las instrucciones.

Aunque la búsqueda de la instrucción se produce en el Ciclo N-1 y su ejecución en el Ciclo N, físicamente la ejecución solo queda limitada a un solo Ciclo pues éste es un proceso encadenado, que interesa a la instrucción N-1 y a la instrucción N. Veamos cómo se realiza a través de un ejemplo (**figura 2.3**):

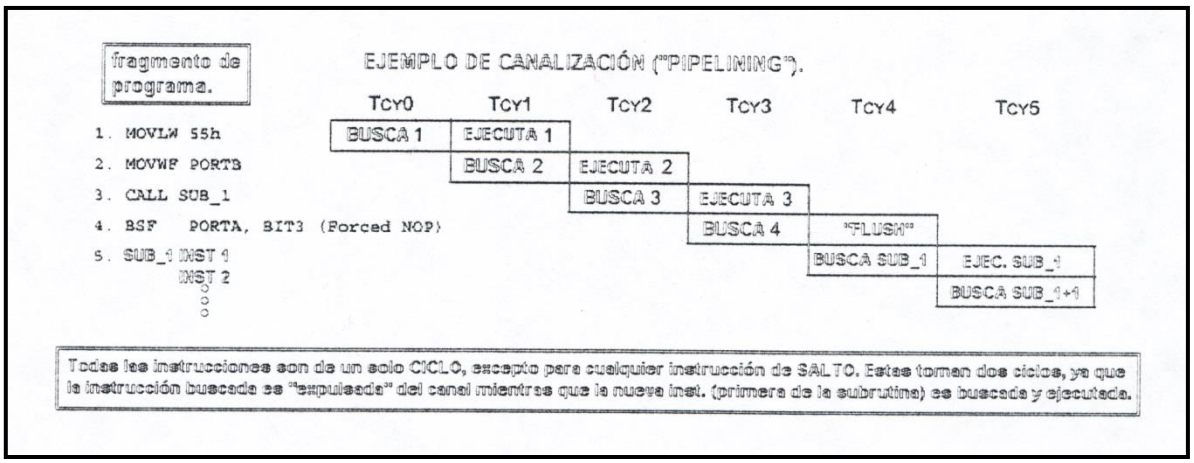


Figura 2.3. Proceso de Canalización de Instrucciones.

En la **figura 2.3,** los intervalos "Tcyx" representan el N enésimo Ciclo de Máquina, igual a 4 estados "Q". Observe que en Tcy0 se busca el código de la instrucción "MOVLW 55h" (mover el literal 55h al acumulador), en Tcy1 se ejecuta "MOVLW 55h" y se busca el código de "MOVWF portb" (mover el contenido del acumulador al puerto B). El proceso continúa hasta la instrucción número 3, que es un "CALL" a la dirección SUB_1. Observe que esta instrucción "CALL" cambiara el contenido del PC (contador de programas) y ahora en lugar de ejecutarse la instrucción "BSF porta,bit3" deberá ejecutarse la primera instrucción de la subrutina SUB_1, luego ahora en el Tcy4 no podrá ejecutarse el "BSF" cuyo código se había buscado en Tcy3 mientras se ejecutaba el "CALL" (inst. num. 3). En este caso el procesador realiza lo que se denomina "flush", lo que equivale más o menos a la "expulsión" de la instrucción "BSF" de la tubería, entonces en el Ciclo Tcy4 solo busca el código de la primera instrucción de la subrutina SUB_1 y no realiza nada sobre la instrucción que le sigue al "CALL". Esta situación hace que físicamente el "CALL" dure dos Ciclos de Máquina.

Finalmente luego de buscar el código de la inst 1 de la subrutina, en Tcy5 se ejecuta mientras se busca el código de la segunda y así sucesivamente hasta encontrar al final de la subrutina la instrucción RETURN, que también dura 2 CICLOS, porque realiza un proceso inverso al CALL.

Arquitectura de la Memoria de Programas en los PICs de la Serie 87X.

La Memoria de Programas (MP) de los microcontroladores de esta familia está organizada en páginas de “2K palabras cada una”. Se dice “2K palabras cada una”, porque los códigos de instrucciones en la MP en esta serie tienen una longitud de 14 bits.

Los PICs de esta serie poseen como máximo 4 páginas de “2K palabras”, lo que hace un total de 8192 palabras de 14 bits cada una. Para direccionar esta cantidad de memoria son necesarios 13 bits, por eso el Contador de Programas en los PICs de esta familia es un registro de 13 bits. (**Figura 2.4**).

El salto entre páginas se realiza escribiendo los bits de un registro denominado PCLATH. Este registro no es más que la parte alta del Contador de Programas (PC).

En los PICs el PC no puede ser escrito todo de una vez (debido a su longitud), por eso existen dos registros en la Zona de Registros Específicos de la RAM que permiten leer o escribir el PC separado en dos partes. Estos registros son el PCLATH y el PCL (parte baja del PC). Más adelante serán tratados en detalle estos registros.

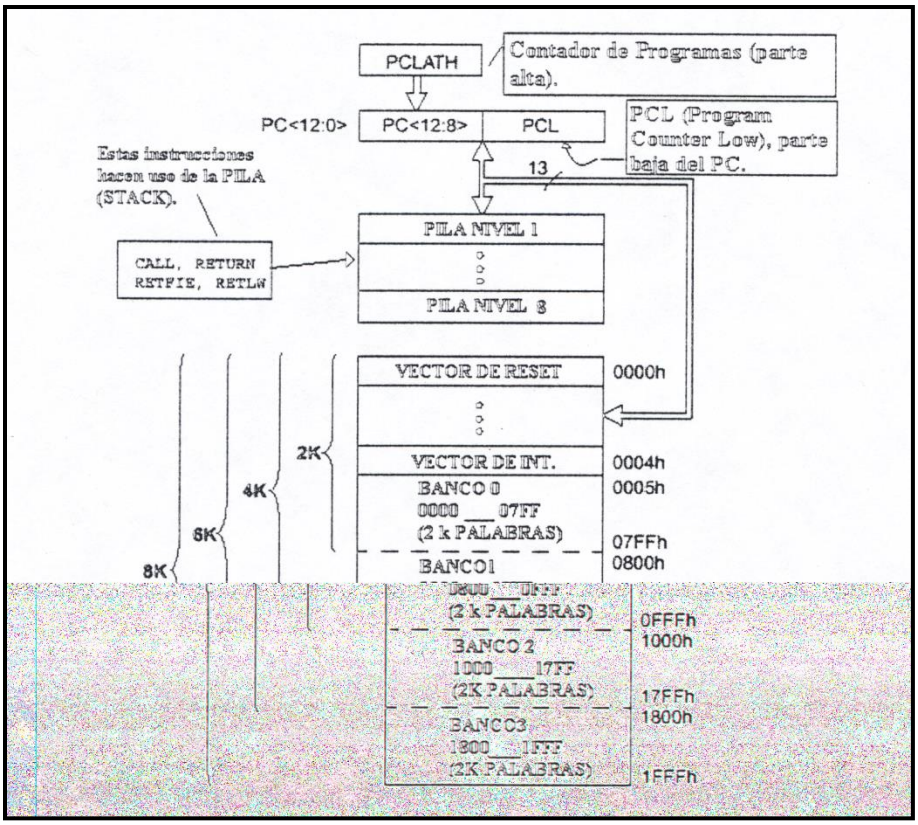


Figura 2.4. Esquema general de la Memoria de Programas en los MC PICs de la familia media.

Lo que se presenta en la **figura 2.4** es un esquema general de la MP. No todos los procesadores de la familia poseen 8K palabras de memoria.

Aspectos relevantes de la Memoria de Programas.

Vector de Reset y Vector de Interrupción.

El Vector de Reset es la localización 0000h de la MP, en ella se graba la primera instrucción que debería ejecutar el MC después de una operación de RESET.

El Vector de Interrupción es la localización 0004h de la MP (para la serie 87X). Si se trabaja con interrupciones en la aplicación que se está desarrollando, en ella se graba una instrucción de salto incondicional "GOTO XXXXX", a la dirección de comienzo de la subrutina de atención a interrupción: "XXXXX". En la serie 18FXXX existen dos Vectores de Interrupción porque las interrupciones se encuentran en esta tecnología categorizadas de Alta y Baja Prioridad. En el tópico correspondiente se ampliará al respecto.

Contador de Programas. Direccionado de la Memoria de Programas.

Como en todo sistema Microcontrolador o Microprocesador, el Contador de Programas (PC), especifica la dirección dentro de la MP donde se encuentra la instrucción cuyo código se debe buscar en todo momento. Particularmente en los PICs de esta familia el PC es un registro de 13 bits, que puede se escrito o leído por el programador por medio de dos registros auxiliares que se encuentran en la RAM del MC, estos registros son el PCLATH (últimos 5 bits del PC) y el PCL (parte baja del PC), o sea los 7 primeros bits, lo que hace un total de 13 bits. El PC puede cambiar de manera normal durante la ejecución del programa o puede ser forzado a un valor cambiando el contenido de los registros PCL y PCLATH.

A diferencia de los microcontroladores de la serie 18FXXX; en los PICs de la serie 87X no existe un puntero para direccionar la MP (por ejemplo para acceder a tablas). El direccionado de la MP se realiza cambiando el contenido del propio PC.

Una dirección dentro de la MP se debe dividir en 11 bits que conforman el tamaño máximo posible de una página (2K) y dos bits adicionales para cambiar de página, estos bits son los bits 3 y 4 del registro PCLATH. Así las instrucciones de Salto en un Microchip tienen un alcance de 2048 (7FF), direcciones posibles, o sea un "CALL" o un "GOTO" solo permitirán saltos de dirección dentro de la misma página, si se quiere lograr un salto entre paginas es obligatorio especificarlo con alguna combinación de los bits 3 y 4 del registro PCLATH.

Direccionado de la MP. Diferentes situaciones en las que se modifica el contenido del PC.

En las **figuras 2.5a, b, c, d** se observan diferentes casos, en los que se modifica el contenido del PC provocándose un salto dentro de la Memoria de Programas.

Caso 1: Se modifica el contenido del registro PCL.

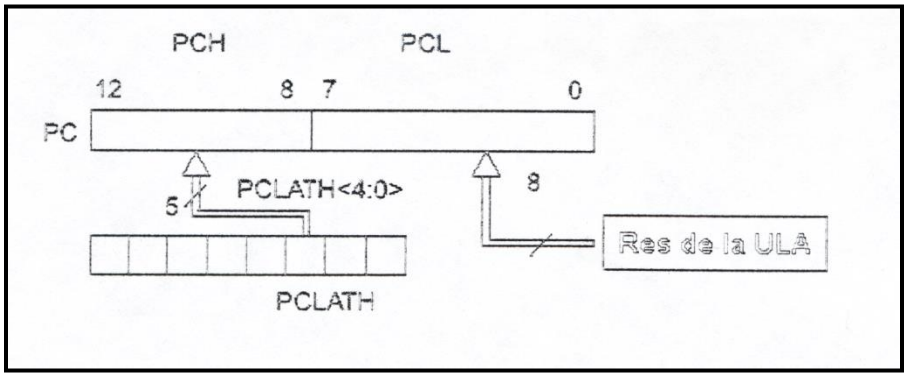


Figura 2.5a. Modificación directa del Contador de Programas.

En este caso se ha realizado una operación aritmética o lógica cuyo resultado se almacenará en los 8 primeros bits del PC, lo que es equivalente a escribir el PCL. El resultado que se genera en la Unidad Lógica Aritmética, se graba en el PCL. Modificar el contenido del PCL causa que automáticamente el contenido de PCLATH se grabe en los últimos 5 bits del PC para con esto completar los 13 bits de la dirección de salto. Luego, es importante, antes de modificar el contenido del PCL cargar el PCLATH con el valor correcto. Antes de realizar este tipo de operaciones, en PICs donde exista más de una página de direcciones en la MP es necesario que el contenido del PCLATH, garantice estar en la página deseada. NUNCA OLVIDE QUE EL "PC" ES EL PUNTERO DE LA MP.

Operaciones de este tipo son por ejemplo:

ADDWF PCL,1	; adiciona el contenido del acumulador al registro PCL y el resultado queda en PCL.
INCF PCL	; incrementa el contenido del PCL y el resultado (PCL + 1) se graba en PCL.

Caso 2: El contenido del PC se modifica por acción de una instrucción GOTO.

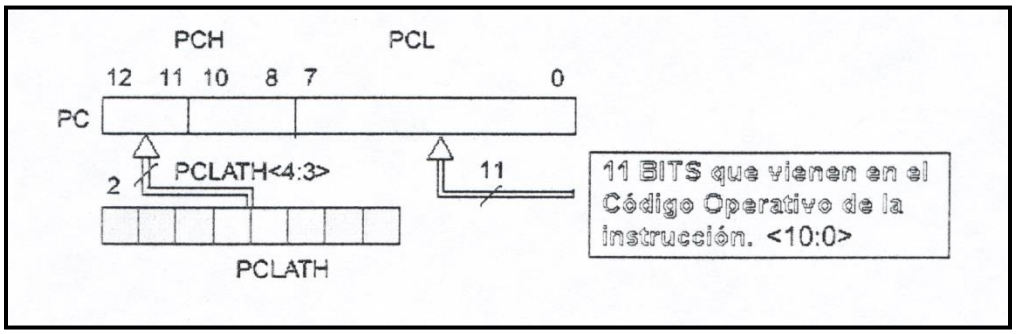


Figura 2.5b. El PC es modificado por la instrucción de salto "GOTO".

En este caso el contenido del PC es cargado por la ejecución de una instrucción GOTO. Se carga un número de 11 bits que viene del código operativo de la instrucción. Este número es la dirección de salto dentro de la página de la MP. Los 8 primeros bits de este número cargan el PCL los 3 restantes cargan los bits 8, 9 y 10 del PC directamente. Para formar la dirección completa dentro de los 8K faltarían los bits 11 y 12 del PC. Estos bits son llenados automáticamente con el contenido de los bits 3 y 4 del registro PCLATH.

Luego, si se trabaja con PICs que posean más de una página de MP es necesario cargar los bits 4 y 3 con el valor correcto, antes de ejecutar un salto, si la dirección de este se encuentra en una página diferente a la página donde esta ensamblada la instrucción de salto.

Por último tenga en cuenta que el GOTO es un salto de "no retorno", o sea al saltar se pierde la dirección de origen y NO es posible regresar. En otras palabras no se guarda en la PILA la dirección de retorno.

Caso 3: El contenido del PC se modifica por acción de una instrucción CALL.

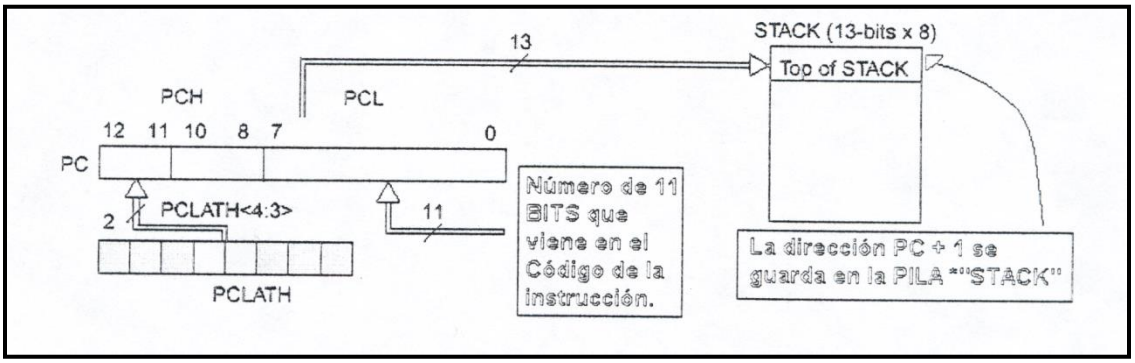


Figura 2.5c. El PC se modifica por la ejecución de la instrucción "CALL".

En este caso funciona de manera similar al anterior pero con la diferencia de que la dirección de retorno (que sería PC + 1), se guarda en la PILA o STACK del sistema (dirección siguiente); o sea un CALL, produce un salto luego del cual se puede retornar a la dirección siguiente a donde esta ensamblado el CALL.

Caso 4: El contenido del PC es devuelto desde la PILA por acción de instrucciones de retorno como: RETURN, RETFIE O RETLW.

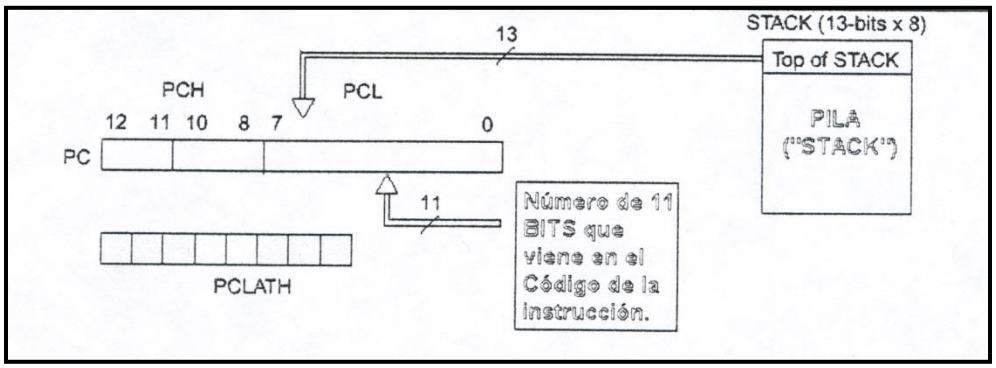


Figura 2.5d. El contenido del PC es restituido después de un retorno de una subrutina o de atención a IT.

Este caso es similar al CALL, pero se realiza la operación contraria: recupera el contenido del PC respaldado en la PILA (PC + 1) al registro PC para conducir la ejecución del programa a esa dirección. Importante es señalar que este caso no requiere de la carga de los bits 3 y 4 de PCLATH, puesto que los 13 bits necesarios son transferidos de la PILA al PC de manera íntegra, de esta manera la corrida del programa regresa a la pila adecuada.

La PILA o "STACK".

En los MC PICs la PILA es una memoria especial diseñada según un sistema LIFO (Last In First Out). En esta familia la pila tiene 8 niveles de profundidad, lo que quiere decir que se pueden respaldar hasta 8 direcciones de retorno diferentes.

Cada una de las 8 localizaciones de la PILA tiene 13 bits de longitud pues en ella se respaldan hasta 8 contenidos del PC, que es un registro de 13 bits.

El espacio de la PILA no es parte de la Memoria de Programas ni de Datos, es un espacio independiente de memoria que posee un puntero que NO se puede LEER ni ESCRIBIR; esto significa que en ensamblador Microchip no existen las instrucciones POP y PUSH, el PUSH se realiza de manera automática ante una instrucción CALL o una Interrupción y el PUSH también se realiza de manera automática ante una instrucción RETURN, RETFIE o RETLW.

Otro detalle a considerar es que los PICs de la serie 87X no existe ningún tipo de aviso acerca del desbordamiento de la PILA, solo se pueden anidar hasta 8 saltos. Si este número se excede se sobrescribirán las direcciones del PC guardadas (**figura 2.6**) y se perderá el curso del programa.

Por otra parte, la PILA en la serie 18F4XXX es mucho más compleja y si puede ser manipulada desde el programa, lo que ofrece grandes ventajas que serán tratas en los temas correspondientes a esta tecnología.

Después de un análisis, nunca olvide que solo se puede anidar hasta 8 subrutinas en un programa escrito para PICs de la serie 87X. Tal cantidad parece poca, pero es más que suficiente para la mayoría de las aplicaciones.

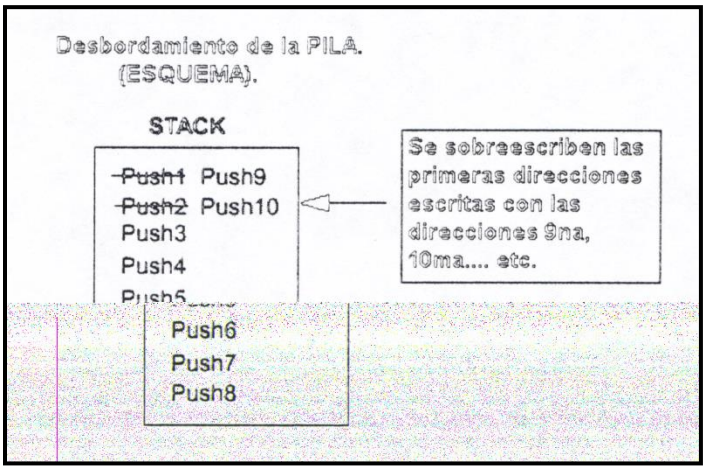


Figura 2.6. Desbordamiento de la PILA.

Arquitectura de la Memoria de Datos en los PICs de la Serie 87X.

Al igual que la MP, la Memoria de Datos (MD) en los PICs esta divida en páginas. Para referirnos específicamente a las páginas de la MD las llamaremos BANCOS. Los PICs de esta familia poseen hasta 512 Bytes (note que en este caso si se habla de Bytes) de MD. Estos 512 Bytes están agrupados en hasta 4 bancos de 128 Bytes cada uno. La afirmación: “hasta 4 bancos” quiere decir que no todos los dispositivos tienen implementados todo el espacio de MD, de manera similar a lo que ocurre en la MP.

Direccionado de la Memoria de Datos.

Para direccionar la MD es necesario establecer la dirección dentro del banco (cada banco posee 7F localizaciones) en el cual, de los 4 bancos posibles, se está trabajando. Para seleccionar el Banco a direccionar existen dos bits (RP1 y RP0), estos bits se encuentran en un registro especial: el STATUS. El registro STATUS se encuentra a su vez en la dirección 03h de los 4 bancos, o sea es la tercera localización de cada uno de los bancos (**figuras 2.7 y 2.8**).

Cuando el registro que se esté trabajando posea una imagen en otro banco (además del banco 0) no es necesario cambiar de banco para direccionarlo, en este caso se dice que todas la direcciones repetidas están "mapeadas al banco 0".

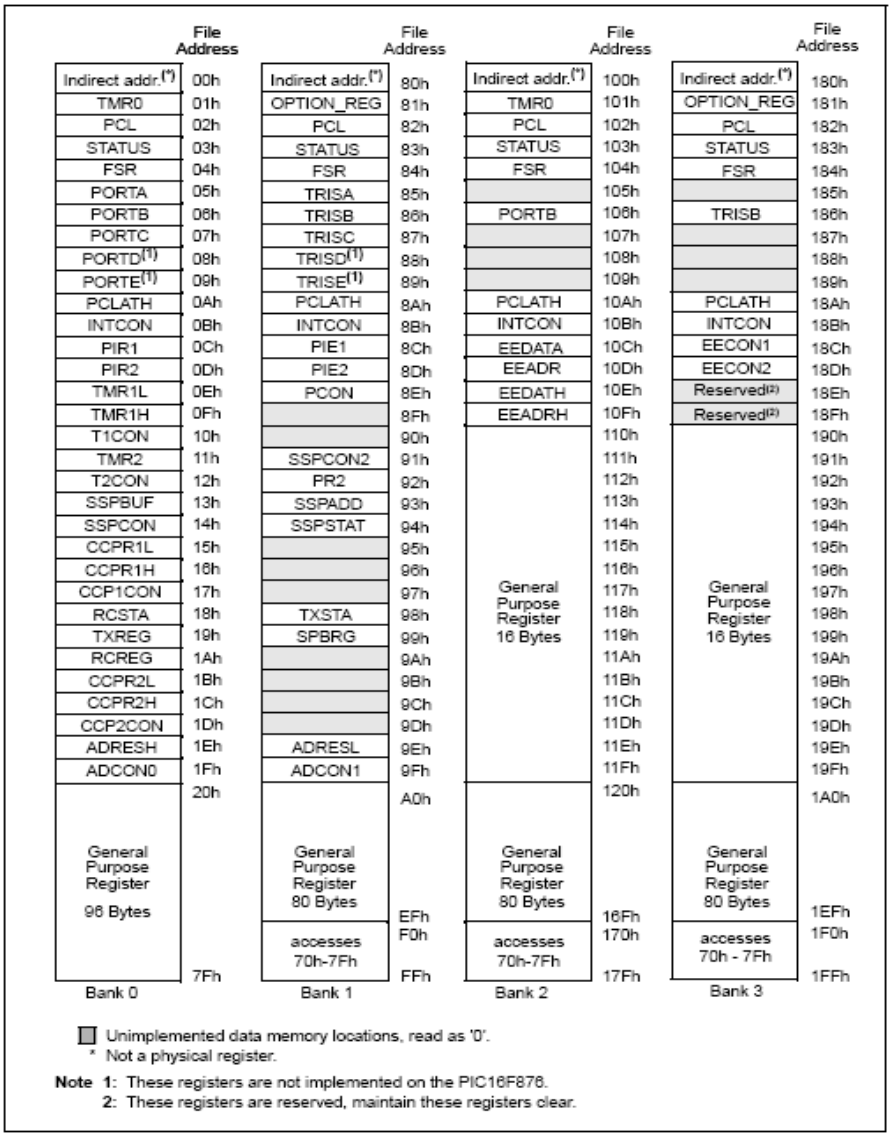


Figura 2.7. Mapa de la Memoria de Datos (PIC 16F877/876).

BITS PARA EL DIRECCIONADO DIRECTO E INDIRECTO DE LA RAM

BANCO ACCESADO.	DIRECTO (RP1:RP0)	INDIRECTO. (IRP)
0	0 0	0
1	0 1	
2	1 0	1
3	1 1	

Figura 2.8. Bits para direccionado de la RAM.

Modo de Direccionado de la Memoria de Datos.

Las direcciones de la MD podrán tener como máximo 9 bits pues 512 es un número de 9 bits (111111111). Basado en esto, existen dos modos de direccionado para formar la dirección completa de localización de la MD: el Directo y el Indirecto.

Modo Directo

En el Modo Directo (**figura 2.9, izquierda**) 7 bits de la dirección vienen en el Código de la Instrucción. Estos 7 bits son la dirección dentro de uno de los 4 Bancos posibles (cada Banco posee como máximo 128 (7F) Bytes). Para formar la dirección completa dentro de los 512 Bytes que pueden direccionarse, faltarían dos bits; estos dos bits son los bits 5 y 6 del registro STATUS (RP1 y RP0).

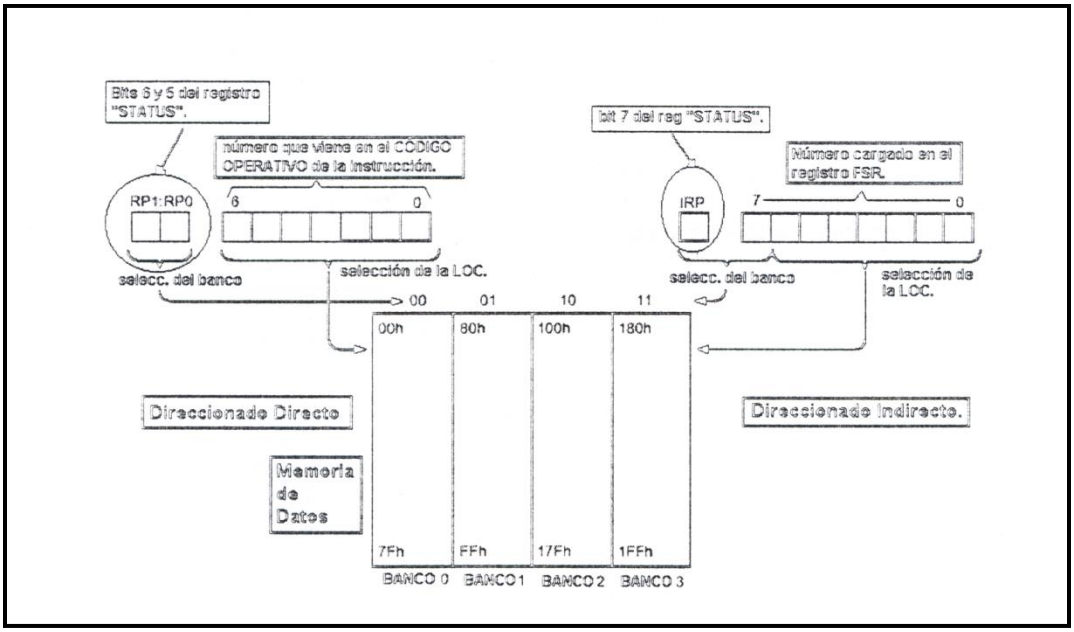


Figura 2.9. Modos de Direccionado de la MD en los MC PICS.

Modo Indirecto.

En el Modo Indirecto (**figura 2.9, derecha**), la localización de memoria se direcciona mediante un puntero; el registro FSR y un registro auxiliar, el INDF. El proceso de direccionado ocurre de la manera siguiente:

En el FSR (puntero de la MD) se cargan los 8 bits de la dirección base. Como la dirección cargada es de 8 bits, esta puede referirse a un localización que se encuentre ubicada entre la DIR 00 y la FF o entre la dirección 100 y 1FF, o sea es necesario también definir a que bloque nos referimos, si al Banco 0 y 1 ó al Banco 2 y 3. Para formar entonces la dirección completa, es necesario entonces es necesario definir el estado del bit "IRP", que es el bit 7 del registro "STATUS".

Por ejemplo: si cargamos en el registro FSR la dirección 0Ch, podremos referirnos a la localización 0Ch del Banco 0 si el bit IRP está en 0 o la localización 10Ch (Banco 2) si el bit IRP está en 1.

Registro auxiliar INDF.

El registro INDF no es un registro físico. Simplemente actuar sobre la dirección denominada INDF (primera localización de cada Banco), significa "actuar" sobre la localización de la MD a la que apunta el registro FSR. Por ejemplo, si FSR = 20h, y hacemos "CLRF INDF", estamos limpiando la localización 20h de la RAM.

El direccionado indirecto de la MD es muy útil para tratar bloques de localizaciones consecutivas de MD, con pocas instrucciones.

Registro de Estado (STATUS).

El registro "STATUS" se encuentra en las direcciones 03h, 83h, 103h, 183h. es un registro que contiene bits de control para el cambio de Banco en la MD, los bits que funciona como banderas aritméticas y dos bits indicadores de Modo Power Down (**figura 2.10**).

REGISTER 2-1: STATUS REGISTER (ADDRESS 03h, 83h, 103h, 183h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

bit 7 **IRP:** Register Bank Select bit (used for indirect addressing)
1 = Bank 2, 3 (100h - 1FFh)
0 = Bank 0, 1 (00h - FFh)

bit 6-5 **RP1:RP0**: Register Bank Select bits (used for direct addressing)
11 = Bank 3 (180h - 1FFh)
10 = Bank 2 (100h - 17Fh)
01 = Bank 1 (80h - FFh)
00 = Bank 0 (00h - 7Fh)
Each bank is 128 bytes

bit 4 $\overline{TO}$: Time-out bit
1 = After power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred

bit 3 $\overline{PD}$: Power-down bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 2 **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1 **DC**: Digit carry/$\overline{\text{borrow}}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
(for $\overline{\text{borrow}}$, the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0 **C**: Carry/$\overline{\text{borrow}}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

Note: For $\overline{\text{borrow}}$, the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high, or low order bit of the source register.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Figura 2.10. Registro de Estado "STATUS".

Ejercicios Correspondientes al Tema

1. ¿Por qué en los Microcontroladores Microchip™ es posible codificar las instrucciones en la Memoria de Programas utilizando una sola palabra binaria (de 14 bits en la serie 87X)?
2. ¿Qué significa que un Set de Instrucciones sea "Ortogonal"?
3. ¿A qué se denomina "Ciclo de Máquina" y "Ciclo de Instrucción"?
4. ¿Cuánto dura (en general) el Ciclo de Máquina en los Microcontroladores Microchip?
5. ¿A qué se denomina "canalización (pipelining) de instrucciones?
6. Describa estructuralmente la Memoria de Datos (RAM) integrada a los microcontroladores de la serie 87X.
7. ¿Cuántos tipos de registros se definen dentro del espacio de la Memoria de Datos de un Microcontrolador?
8. Elabore un esquema que represente la ubicación física de los registros definidos en la pregunta anterior dentro cualquier banco de la Memoria de Datos.
9. ¿Qué funciones se asocian a los bits del registro "STATUS"?
10. ¿Qué es el registro acumulador "W" y en qué procesos participa dentro de la ejecución de un programa en un Microcontrolador?
11. Diseñe un segmento de programas que permita programar las 8 líneas del Puerto D (PORTD) como salidas digitales.
12. Dado el siguiente "segmento de hardware basado en Microcontrolador" que se muestra en la figura 1; defina el valor que debe cargarse al o los registros "TRIS" correspondientes.
13. Explique cómo se forma una dirección cualesquiera dentro de la Memoria de Datos, utilizando ya sea mecanismos de direccionado directo e indirecto.
14. Seleccione la alternativa correcta y fundamente su respuesta en cuanto a si en el siguiente Segmento de Programa se emplea(n) la(s) técnica(s) de direccionado de la Memoria de Datos del tipo:

- Indirecto.
- Directo.
- Ambas técnicas.
- Ninguna.

Segmento de Programa:

```
again bsf 3,5
clrf 6
bcf 3,5
;------------
movlw 0x20
movwf 4
;------------
```

```
clean clrf 0
movf 4,0
sublw 0x2F
btfsc 3,2
goto again
incf 4
goto clean
```

15. Diseñe dos segmentos de programa que permitan incrementar el valor de los contenidos de las localizaciones de RAM cuyas direcciones van desde la 0x20 hasta la 0x2F; uno utilizando direccionado directo y el otro utilizando direccionado directo.

16. ¿Qué ventajas operativas tendría el empleo de la técnica de Direccionado Indirecto en ciertos casos de programación?

17. Seleccione la(s) alternativa(s) correctas:

"En un microcontrolador, el Contador de Programas es":

- Un Contador de Eventos Externos.
- El Puntero de la Memoria de Datos (RAM).
- Un registro que almacena la dirección que posee en la Memoria de Programas la instrucción (comando) que se está ejecutando en cada momento durante la corrida del programa.
- Un registro que almacena el contenido de la primera localización de la Pila ("Stack").

18. Seleccione la(s) alternativa(s) correcta(s) y justifíquela(s).

"En un Microcontrolador (del la serie 87X), la PILA o STACK es":

- Un segmento especial de memoria en el que se respalda el contenido del registro acumulador (W) y el estatus (STATUS).
- Un segmento especial de memoria de 31 bits x 21 localizaciones en el que se respalda el contenido del Contador de Programas.
- Un segmento especial de memoria de 13 bits x 8 localizaciones en el que se respalda el contenido del Contador de Programas (PC), ante la ejecución de una instrucción tipo "call".
- Un Registro de Funciones Especiales que sirve para conmutar los Bancos de la Memoria de Datos.

19. Diseñe el esquema para conectar 3 Displays de 7 segmentos de Leds de tecnología cátodo común a un Microcontrolador PIC16F877, utilizando el método de multiplexación del bus.

20. Diseñe un algoritmo que permita “refrescar” la información que se muestra en tres Displays de 7 segmentos de Leds con bus de datos multiplexado y comunes independientes.

Tema III: Introducción a la programación de los MC PICS de la Familia Media.

Sumario:

1. Herramientas de Desarrollo.
2. Introducción al Set de Instrucciones básico que manejan los microcontroladores de la Serie 87X.
3. Control del tiempo de ejecución de un programa.
4. Métodos para implementar Tablas en la Memoria de Programas.

Herramientas de desarrollo

Estructura básica de un programa escrito en Ensamblador.

La edición del código, involucra conocer al menos la estructura elemental de un programa escrito en ensamblador Microchip. Se analizarán algunos aspectos necesarios para comenzar a trabajar.

- En el encabezamiento del programa normalmente se definen la siguientes directivas: `list p=<tipo de procesador>` e `include <nombre del fichero include, relacionado con el procesador que se utiliza>`, por ejemplo: `list p=16f877A` y `#include "p16f877A.inc"`.
- Usualmente también se definen los Vectores de Reset e Interrupción, utilizando la directiva "org XXXXX". La directiva "org XXXXX" (origin); establece una dirección dentro de la Memoria de Programas a partir de la cual se ensambla una instrucción o un conjunto de ellas. El Vector de Reset (primera instrucción que se ejecutará) se ensambla en la dirección 0000 y en muchos estilos de programación contiene una instrucción de salto a la dirección a partir de la cual se escribirá el Programa Principal de la Aplicación (por ejemplo; "goto inicio"); el que normalmente comienza con una serie de instrucciones que conforman el bloque de configuración del hardware; algo así como la función que realiza el BIOS en una PC. Por otra parte el Vector de Interrupciones (único en la Serie 87X no así en la serie 18FXXXX) se ensambla en la dirección 0004 y puede consistir de un salto a la dirección donde comienza la Subrutina de Atención a la Interrupción o en muchos casos es la primera instrucción de la mencionada subrutina; lo que significa que se ha optado por escribir la Subrutina de Atención a partir de la dirección 0004.
- La estructura básica de un programa escrito en ensamblador Microchip se esboza en la **figura 3.1.**

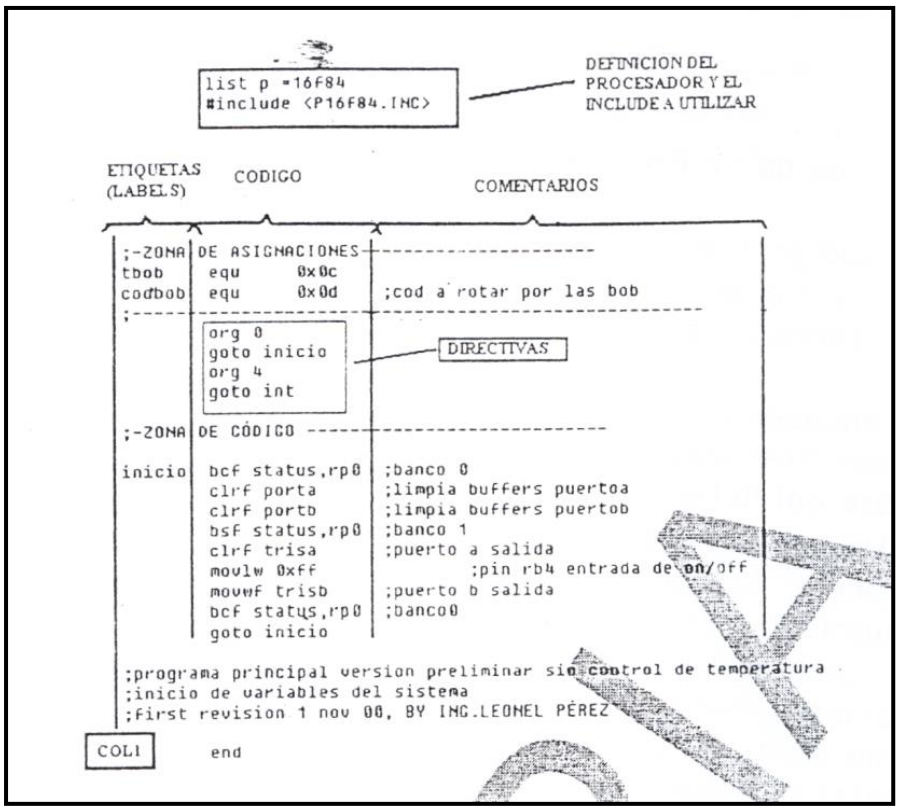


Figura 3.1. Estructura básica de un programa escrito en Ensamblador.

Otros detalles de interés sobre el esquema anterior:

- Una directiva es una orden dentro del texto del programa ensamblador que indica un proceso que deberá ejecutar el programa compilador, pero que no genera código de máquina (no se graban en la Memoria de Programas del PIC). Ejemplos de directivas son: EQU, LIST P16F870, INCLUDE, ORG, etc. Para ampliar el tema "Directive Languaje" puede referirse al capítulo 5 del "MPASM USERS GUIDE".
- El código será escrito de tal manera que los "labels" o etiquetas comiencen a editarse desde la columna 1 del editor. Si el ensamblador encuentra un label después de la columna 1 emitirá un "warning". Lo mismo ocurrirá si se encuentra una "directiva" editada a partir de la columna 1.
- Todos los comentarios comenzarán con el caracter ";", de lo contrario será emitido un error de sintaxis durante la compilación.
- Todo programa escrito en ensamblador Microchip terminara con la directiva "end".
- Se recomienda realizar una edición en minúsculas, organizada e incluir los comentarios correspondientes, para garantizar la legibilidad de los programas.

- Se recomienda al inicio de cada subrutina comentar acerca de su funcionamiento, variables de entrada y salida y en general cualquier otro aspecto de interés relacionado con ella.
- Es posible realizar también la edición y compilación del código para una aplicación en lenguaje C, siempre que se tengan instaladas sobre MPLAB las correspondientes herramientas. Esta opción NO es aconsejable para procesadores que posean una memoria de programas muy limitada, puesto que de la compilación de un segmento de programa en leguaje C, resultan segmentos de código Microchip bastante extensos, que pudieran copar la capacidad de la Memoria de Programas.

Compilación y Simulación de Códigos.

Compilación.

Compilar es una acción automatizada para verificar si el código escrito es correcto desde el punto de vista de la "sintaxis". No necesariamente un programa que compila correctamente se ejecuta correctamente; sin embargo en sentido contrario la afirmación si se cumple.

Para compilar un proyecto se puede utilizar los "shortcuts" F10 ("Make") o CTRL + F10 ("Build All"). Otra alternativa sería utilizar las opciones del menú "Project".

Los resultados de una compilación se muestran en una ventana titulada "output", pueden aparecer tres tipos de errores:

- **Message:** un "message" no es un error, simplemente constituye una advertencia de orden primario que en algunos casos pudiera ser causa de un no funcionamiento correcto; por lo tanto se sugiere en las primeras sesiones tomarlos en cuenta.
- **Warning:** un "warning", como su nombre lo indica, es una advertencia o mensaje con connotación más severa que en algunos casos pudiera ser la causa de un no correcto funcionamiento; por lo tanto se sugiere en las primeras sesiones tomarlos en cuenta.
- **Error:** explícitamente es un error de tipo severo que no permite una compilación exitosa (no así en los casos anteriores). En caso de error este debe ser corregido o de lo contrario no puede pasarse a las etapas siguientes del desarrollo del programa.

Formato de exposición de errores. Ejemplos.

El formato general será explicado a través de un error tipo “message”.

Ejemplo 1. Message <>.

- <Message[305] D:\PROYECTOS\CONDUCIMETRO\CODIGO\COND12 09 2005.ASM 563 : Using default destination of 1 (file)>. El formato en este ejemplo se presenta de la siguiente forma: [305] es el código del tipo de error, D:\PROYECTOS\CONDUCIMETRO\CODIGO\COND12 09 2005.ASM es el “path” del archivo donde se comete el error y su nombre (en este caso COND 12 09 2005.ASM). El número 563 es el número de la línea en donde se produce el “error”. En general se puede acceder directamente a la línea en donde se produce un error dando doble click en la ventana “output” encima del error.

Ejemplo 2. Error <>.

- <Error[113] D:\PROYECTOS\CONDUCIMETRO\CODIGO\COND12 09 2005.ASM 414 : Symbol not previously defined (ACON0)>. Para este ejemplo es válido todo lo descrito para el “message”, en cuanto a formato. Este tipo de error “Symbol not previously defined” se produce cuando se ha confundido el nombre de algún Registro de Propósito Especifico o cuando se incluyen en el código símbolos no declarados o que no se encuentran en un “include”.

Recomendaciones para una compilación rápida y exitosa.

- Conocer el Set de Instrucciones
- Trabajar con el fichero auxiliar “include” si se utilizan simbolos para nombrar a los SFRs (por ejemplo PORTB, STATUS, etc).
- Escribir el cuerpo de las instrucciones en minúsculas y el nombre de los registros en mayúsculas.
- Comentar los segmentos necesarios del programa para ubicar errores rápidamente.
- Leer e interpretar el o los errores presentes en la ventana “output”.
- Establecer una edición organizada y cuidadosa.

Simulación.

Para una simulación (“debbugging”) del código primero es necesario: haber logrado una compilación exitosa y habilitar la herramienta simuladora; tal y como se muestra en la **figura 3.2**.

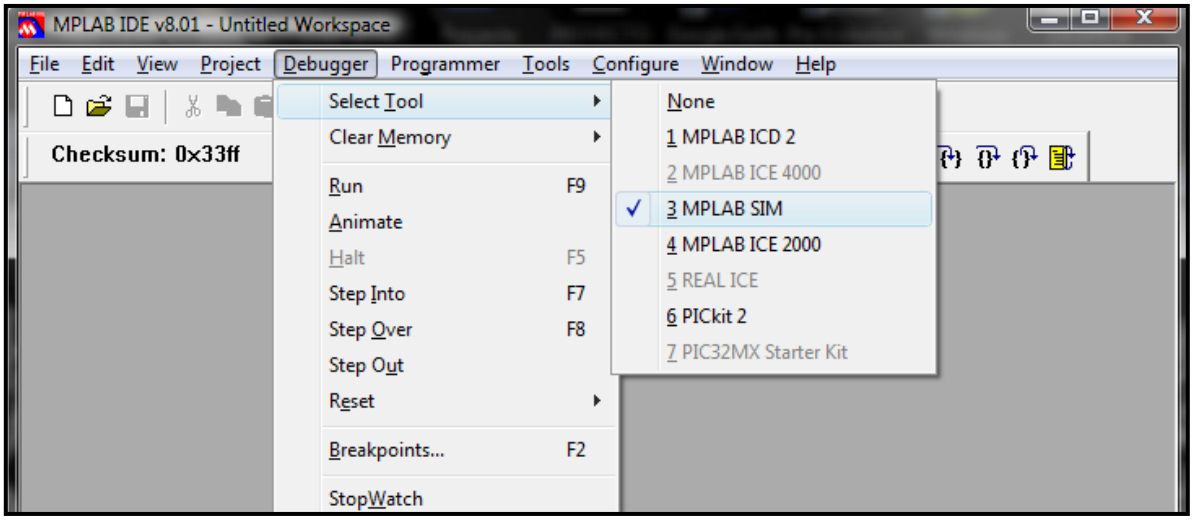


Figura 3.2. Habilitación del Simulador.

Crear una ventana para visualizar los Registros de Control y de Propósito General necesarios. Para crear ventanas utilice el camino de menús que se muestra en la **figura 3.3**.

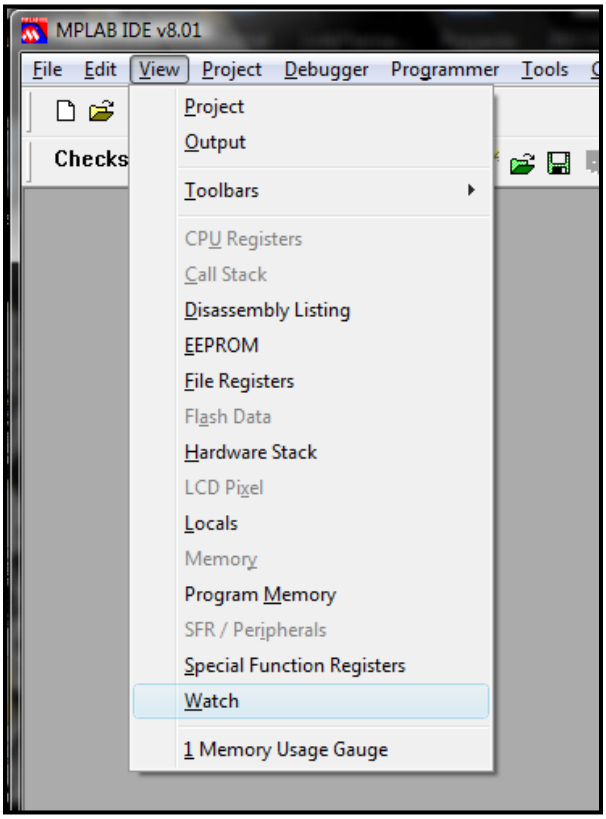


Figura 3.3. Creación de una nueva ventana.

Estructura de una ventana.

La **figura 3.4** muestra la estructura de una ventana.

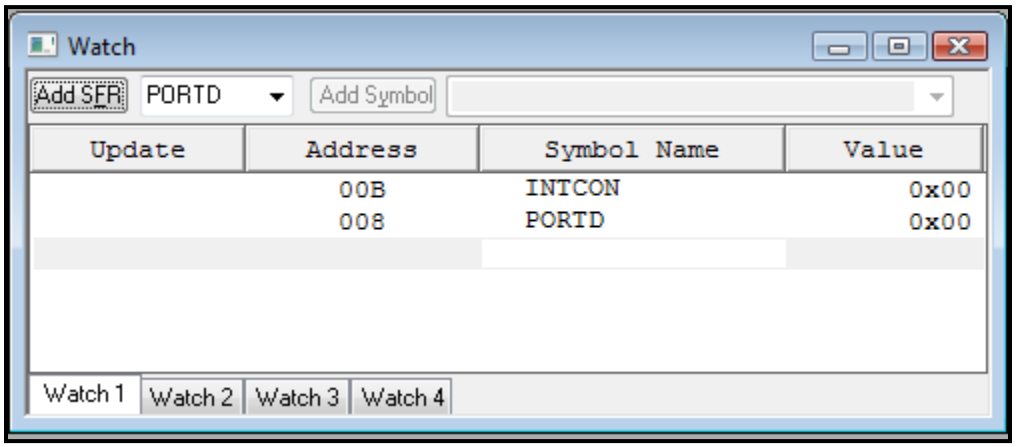


Figura 3.4 ejemplo de ventana. Note que en una misma ventana se pueden crear hasta 4 (watch 1 …..watch4).

La ventana permite visualizar tanto registros tipo SFR como GPRs. Para añadir SFRs accese la lista izquierda, al lado del botón Add SFR. Una vez buscado el SFR que se desea añadir presione el botón correspondiente. Para añadir GPRs o símbolos accese la lista derecha, al lado del botón Add Symbol. Una vez buscando el GPR que se desea añadir presione el botón correspondiente.

Debe considerarse siempre en la simulación el formato en que se desea ver el registro (SFR o GPR). Para cambiar el formato dar click derecho encima del registro (en la ventana) y aparecerá la Caja de Diálogo correspondiente (se muestra en la **figura 3.5**).

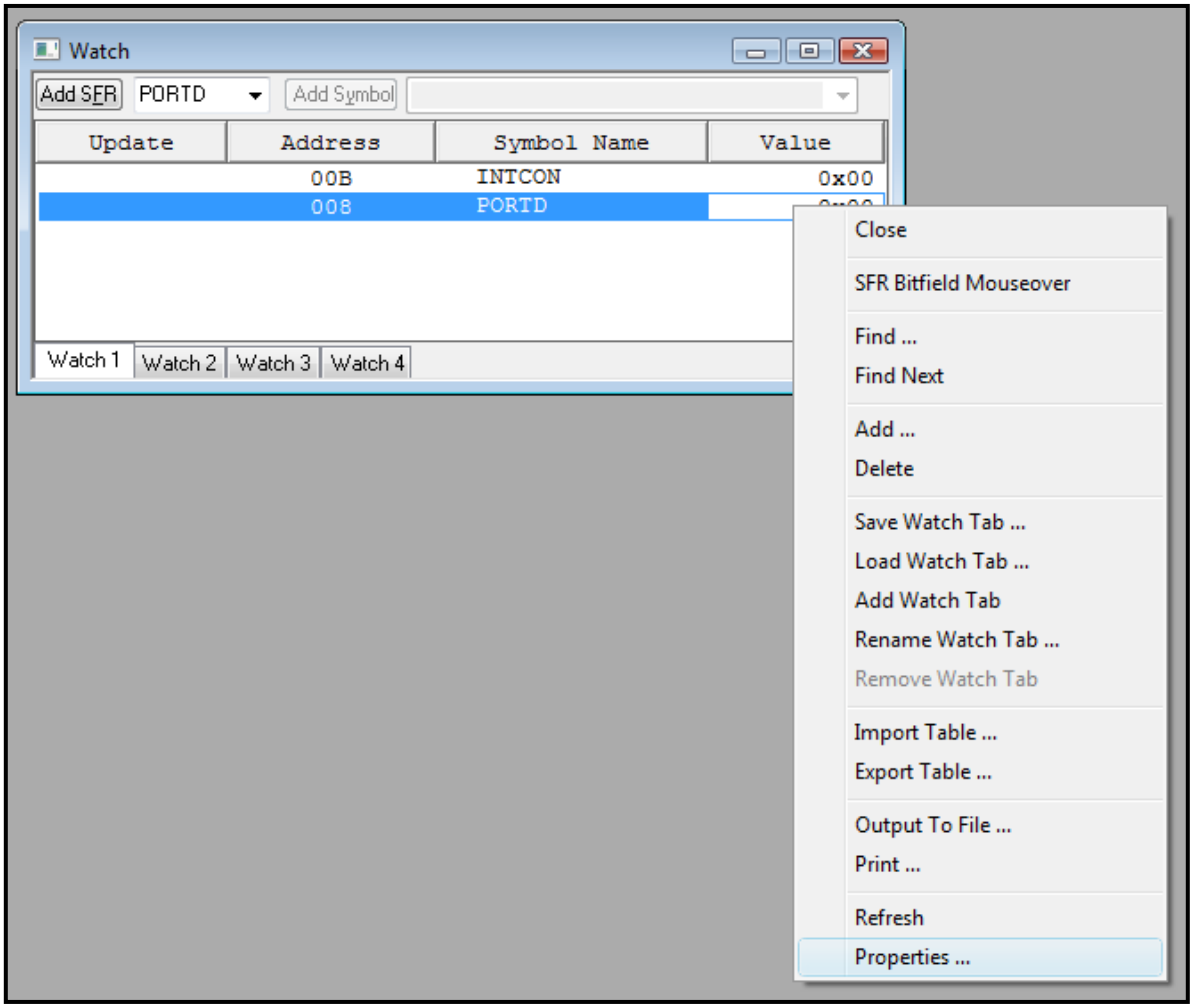


Figura 3.5. Acceso a las propiedades de un Registro Visualizado en una ventana.

Definir Frecuencia de Trabajo del sistema.

Este punto permite definir la Frecuencia de Trabajo del procesador; con el objetivo de poder simular eventos donde el tiempo es importante (por ejemplo: demoras).

Para definir la frecuencia (que debe ser la misma que la utilizada en la Tarjeta), utilice el siguiente procedimiento: debbuger/settings (**figura 3.6**).

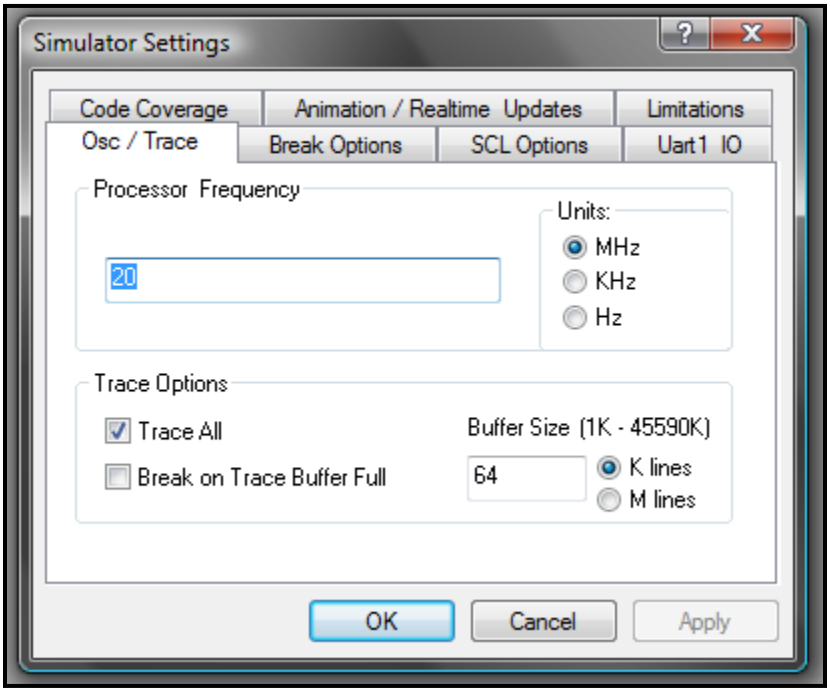


Figura 3.6. Ventana que permite definir la frecuencia de trabajo.

Simulando el tiempo.

El simulador permite observar el tiempo que transcurre durante la ejecución mediante una ventana especial denominada "Stop Watch". Para acceder a dicha ventana utilice: debbuger/stop watch (**figura 3.7**).

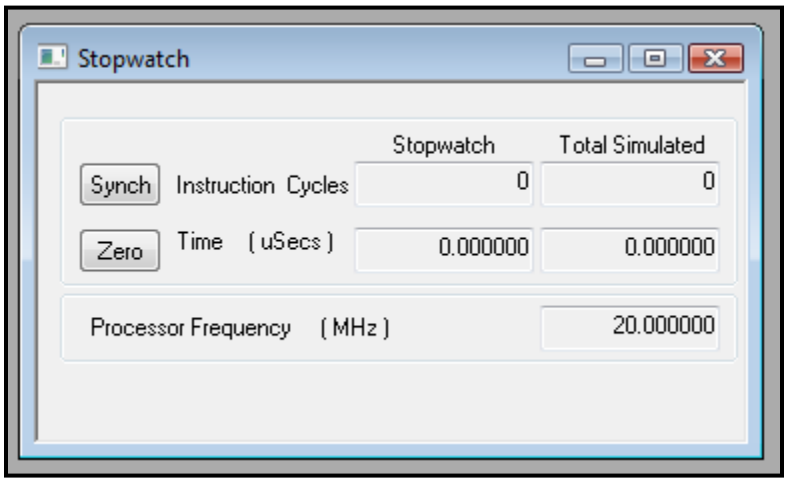


Figura 3.7. Ventana "Stop Watch".

Shortcuts Principales para simular.

- **F7:** Corrida paso a paso.
- **F9:** Corrida continua.
- **F6:** RESET (similar al dado en el pin MCLR).

Recomendaciones para una rápida y cómoda simulación.

- Visualizar los registros necesarios y en el formato necesario en una ventana. Si es necesario separar los registros por categorías en varias ventanas.
- Definir correctamente la frecuencia.
- Acomodar adecuadamente las ventanas en el área del monitor.

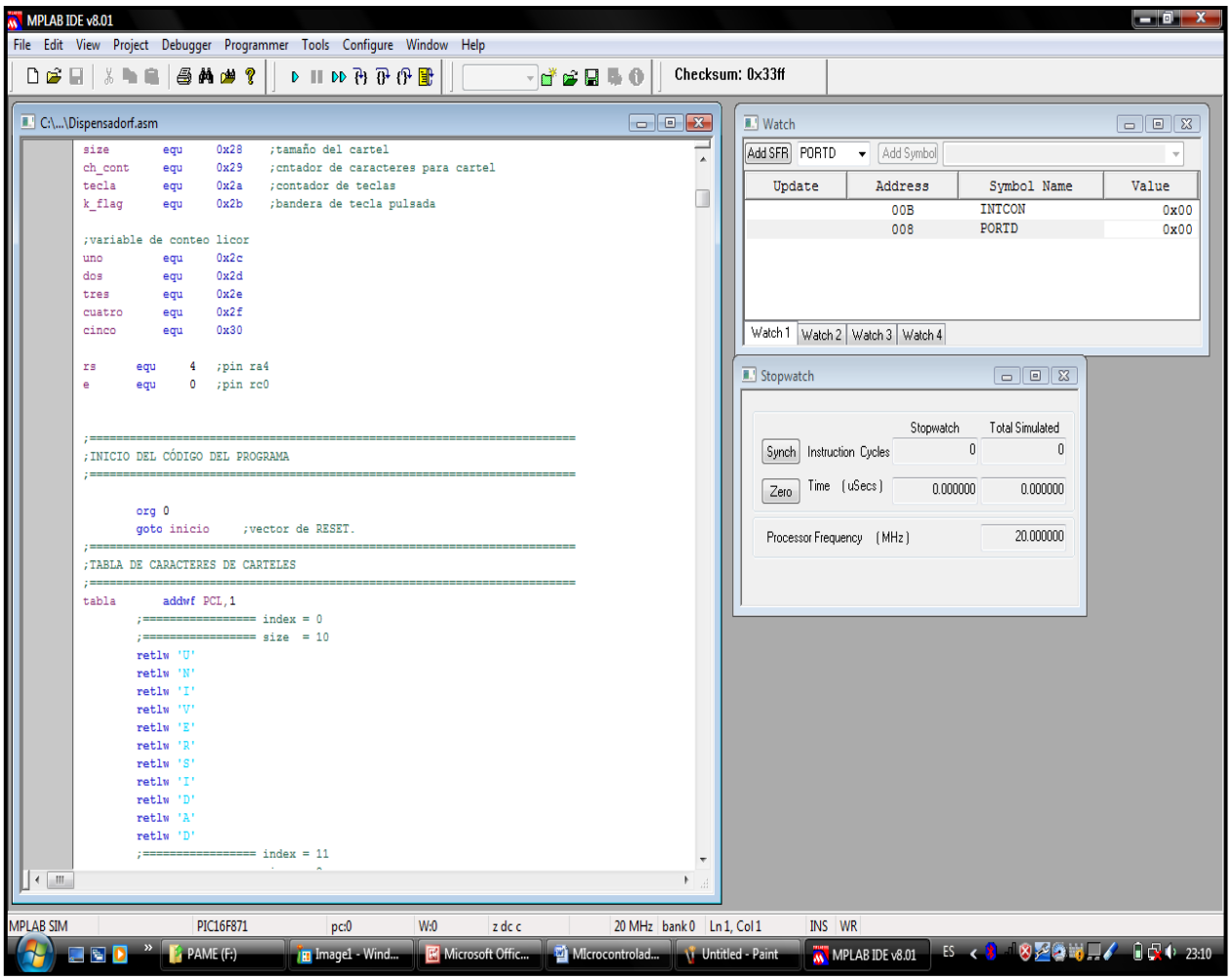


Figura 3.8. Estructura cómoda sugerida para simular.

- Observar paso a paso la evolución de la corrida del programa (F7), al menos de una vez.
- No malgastar el tiempo en pruebas en tarjeta sin corroborar que la simulación es correcta.

Ejercicio: editar, compilar y practicar la simulación del código dado. Adecuar el código para que trabaje con LEDS conectados al puerto D (rd0, rd1 y rd2).

Introducción al Set de Instrucciones básico que manejan los microcontroladores de la Serie 87X.

Introducción

Como ya se había señalado, los PICs son microcontroladores basados en Arquitectura Harvard y manejan un Juego Reducido de Instrucciones.

Gracias la Arquitectura Harvard es posible la canalización (pipelining). La canalización permite realizar la decodificación de instrucciones y la búsqueda de datos de forma paralela.

Por otra parte, a pesar de manejar un Juego de Instrucciones Reducido; este tiene casi las mismas prestaciones, en términos de compacidad de código que un juego de instrucciones complejo.

Otras características ventajosas relativas al Juego de Instrucciones que manejan los MC microchip son las siguientes:

- **Instrucciones Asimétricas:** las instrucciones de este tipo hacen posible llevar a cabo cualquier operación sobre cualquier registro (de propósito general o específico) de la Memoria de Datos utilizando cualquier modo de direccionamiento.
- **Instrucciones codificadas en una sola palabra:** como el Bus de Instrucciones es separado del Bus de Datos, este posee la cantidad suficiente de líneas como para realizarse la transferencia de los códigos de instrucciones de una sola vez hacia la Unidad de Procesamiento Central; lo que resulta en un ahorro de tiempo de ejecución del programa.

Tipos (categorías) de Instrucciones Microchip.

Las instrucciones de la familia media se agrupan en tres categorías:

- Instrucciones orientadas a Byte.
- Instrucciones orientadas a Bits.
- Instrucciones para operaciones con literales y de control.

Instrucciones orientadas a Bytes.

En estas instrucciones un operando "f" representa un registro de la Memoria de Datos y "d" representa el destino que tendrá el resultado de la operación que se realiza. Recuerde que como el Juego de Instrucciones es simétrico, "f" en general es un número de 7 bits que puede representar un registro de propósito específico o cualquier localización de RAM. "d", solo puede tener dos valores (0,1), si "d = 0" significa que el resultado de la operación realizada será depositado en el Registro Acumulador (W)[12] y si d = 1 significa que el resultado quedara grabado en el propio registro "f" especificado en la instrucción.

Ejemplos de instrucciones orientadas a Byte:

MOVWF	**Move W to f**
Syntax:	[*label*] MOVWF f
Operands:	0 ≤ f ≤ 127
Operation:	(W) → (f)
Status Affected: None	
Description:	Move data from W register to register 'f'.

La instrucción "movwf" copia el contenido del registro "W" en "f". No afecta las banderas del "Status".

MOVF	**Move f**
Syntax:	[*label*] MOVF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) → (destination)
Status Affected:	Z
Description:	The contents of register f are moved to a destination dependant upon the status of d. If d = 0, destination is W register. If d = 1, the destination is file register f itself. d = 1 is useful to test a file register, since status flag Z is affected.

[12] El acumulador se refiere como "Working Register (W)" y constituye un registro especial que ocupa un espacio de memoria independiente a la Memoria de Datos y de Programas.

La instrucción “movf” copia el contenido del registro “f” en el propio “f” si d = 1 ó copia el contenido de “f” en “W” si d = 0. Se utiliza típicamente para descargar datos desde la Memoria de Datos al acumulador. Afecta la bandera “Z” del “Status”.

ADDWF	**Add W and f**
Syntax:	[*label*] ADDWF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(W) + (f) → (destination)
Status Affected:	C, DC, Z
Description:	Add the contents of the W register with register ’f’. If ’d’ is 0, the result is stored in the W register. If ’d’ is 1, the result is stored back in register ’f’.

La instrucción “addwf f,d” es un ejemplo de Instrucción Aritmética Orientada a Byte, que suma el contenido de “W” con el del registro “f” y el resultado lo copia en “W” si d = 0 ó en “f” si d = 1. Afecta las banderas “C”, “DC” y “Z” del “Status”.

DECFSZ	**Decrement f, Skip if 0**
Syntax:	[*label*] DECFSZ f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) - 1 → (destination); skip if result = 0
Status Affected:	None
Description:	The contents of register ’f’ are decremented. If ’d’ is 0, the result is placed in the W register. If ’d’ is 1, the result is placed back in register ’f’. If the result is 1, the next instruction is executed. If the result is 0, then a NOP is executed instead making it a 2TCY instruction.

INCFSZ	**Increment f, Skip if 0**
Syntax:	[*label*] INCFSZ f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) + 1 → (destination), skip if result = 0
Status Affected:	None
Description:	The contents of register ’f’ are incremented. If ’d’ is 0, the result is placed in the W register. If ’d’ is 1, the result is placed back in register ’f’. If the result is 1, the next instruction is executed. If the result is 0, a NOP is executed instead, making it a 2TCY instruction.

Las instrucciones “decfsz f” e “incfsz f” son instrucciones que incrementan o decrementan el contenido del registro “f” (según el caso). Si f = 00h, no ejecutan la instrucción inmediata siguiente y en caso de que f sea <> 00h la ejecutan. En cierto sentido son análogas a una estructura “while” de nivel simple; empleada en los lenguajes de programación de alto nivel. No afectan las banderas del “Status”.

Instrucciones orientadas a Bits.

Esta categoría especialmente dedicada al manejo independiente de bits, siguiendo el mismo principio de simetría; con estas instrucciones se pueden manejar por separado bits de cualquier registro, ya sea un registro de propósito especifico o uno de propósito general.

En estas instrucciones un operando “b” representa el número del bit dentro del registro “f”, que será afectado por la operación, por lo tanto, como todos los registros son solo de 8 bits, “b” es un número decimal que va desde 0 hasta 7. “f” es un operando similar al empleado en la categoría anterior, o sea un número entre 0 y 127 (decimal).

Ejemplos de instrucciones orientadas a Bits.

BSF	**Bit Set f**
Syntax:	[*label*] BSF f,b
Operands:	0 ≤ f ≤ 127
	0 ≤ b ≤ 7
Operation:	1 → (f<b>)
Status Affected: None	
Description:	Bit 'b' in register 'f' is set.

La instrucción “bsf f,b” setea el bit “b” dentro del registro “f”. No afecta el “Status”.

BTFSC	**Bit Test, Skip if Clear**
Syntax:	[*label*] BTFSC f,b
Operands:	0 ≤ f ≤ 127
	0 ≤ b ≤ 7
Operation:	skip if (f<b>) = 0
Status Affected: None	
Description:	If bit 'b' in register 'f' is '1', the next instruction is executed. If bit 'b', in register 'f', is '0', the next instruction is discarded, and a NOP is executed instead, making this a 2TCY instruction.

La instrucción “btfsc f,b” testea el bit “b” dentro del registro “f”. Si b = 0, no se ejecuta la instrucción inmediata siguiente y en caso contrario se ejecuta. En cierto sentido esta instrucción es análoga a una estructura “if” de nivel simple; empleada en los lenguajes de programación de alto nivel. Es ideal para el testeo de periféricos externos (pulsantes, por ejemplo) a través de pines de puerto del Microcontrolador. No afecta las banderas del “Status”.

Instrucciones para operaciones con literales y de control.

Estas instrucciones, están diseñadas para realizar operaciones aritméticas y lógicas sobre literales (constantes) y para cambiar el contenido del contador de programas de manera incondicional, con el objetivo de direccionar la Memoria de Programas. Emplean el operador "k" que es un valor de 8 ó 9 bits según sea el caso.

Ejemplos de instrucciones para el trabajo con literales:

En este caso el operando "k" representa un literal (número), de 8 bits. Los números pueden escribirse utilizando los siguientes formatos:

- 0x##, en hexadecimal
- '010110101'b, en binario
- '###'d, en decimal

Ejemplos de estas instrucciones son las siguientes:

ANDLW	**AND Literal with W**
Syntax:	[*label*] ANDLW k
Operands:	$0 \leq k \leq 255$
Operation:	(W) .AND. (k) → (W)
Status Affected:	Z
Description:	The contents of W register are AND'ed with the eight bit literal 'k'. The result is placed in the W register.

La instrucción "andlw k" es un ejemplo de Instrucción Aritmética para operaciones con literales, que realiza el "and" del contenido "W" con el literal "k". El resultado en estos casos siempre va a "W". Note la diferencia con "andwf f,d".

MOVLW	**Move Literal to W**
Syntax:	[*label*] MOVLW k
Operands:	$0 \leq k \leq 255$
Operation:	k → (W)
Status Affected:	None
Description:	The eight bit literal 'k' is loaded into W register. The don't cares will assemble as 0's.

La instrucción "movlw k" copia el literal de 8 bits "k" en el registro "W".

Ejemplos de instrucciones "de Control":

Tales instrucciones también son denominadas "de Control", porque modifican el contenido del PC provocando un salto a una dirección determinada dentro de la Memoria de Programas.

El operando “k” en este caso constituye una dirección dentro de la Memoria de Programas, dentro de una página. “k” es un número de 11 bits (2K). Dentro de un programa “k” se suele sustituir por una cadena de caracteres (etiqueta), que el programador puede manejar con facilidad.

Se recomienda fuertemente consultar alrededor de este tema los siguientes documentos:

- Aspectos generales sobre arquitectura interna(…). Tópico: “Direccionado de la MP. Diferentes situaciones en las que se modifica el contenido del PC”.
- Nota de aplicación AN526 (www.microchip.com).

Ejemplos de instrucciones de control son los siguientes:

CALL	**Call Subroutine**
Syntax:	[*label*] CALL k
Operands:	0 ≤ k ≤ 2047
Operation:	(PC)+ 1→ TOS, k → PC<10:0>, (PCLATH<4:3>) → PC<12:11>
Status Affected:	None
Description:	Call Subroutine. First, return address (PC+1) is pushed onto the stack. The eleven-bit immediate address is loaded into PC bits <10:0>. The upper bits of the PC are loaded from PCLATH. CALL is a two-cycle instruction.

La instrucción “call k” es una llamada a la subrutina ensamblada a partir de la dirección representada por “k”. Antes de que se ejecute un “call” el CPU suma uno al PC, lo guarda en la Pila. Luego, los 11 bits menos significativos del PC se cargan con el valor de la dirección “k” y los dos que restan se cargan desde los dos bits más significativos del PCLATH. Así se genera la dirección completa del salto[13].

GOTO	**Unconditional Branch**
Syntax:	[*label*] GOTO k
Operands:	0 ≤ k ≤ 2047
Operation:	k → PC<10:0> PCLATH<4:3> → PC<12:11>
Status Affected:	None
Description:	GOTO is an unconditional branch. The eleven-bit immediate value is loaded into PC bits <10:0>. The upper bits of PC are loaded from PCLATH<4:3>. GOTO is a twocycle instruction.

[13] Para detalles adicionales consulte documentación recomendada.

La instrucción “goto k” es un salto incondicional a la dirección representada por “k”. Los 11 bits menos significativos de PC se cargan con el valor de la dirección “k” y los dos que restan se cargan desde los dos bits más significativos del PCLATH. Asi se genera la dirección completa del salto. Note que la dirección actual del PC se pierde en este caso.

Formato general de Codificación de las Instrucciones.

Las instrucciones Microchip se codifican en una sola palabra, en el caso de la Familia Media la longitud de esta palabra es de 14 bits. Dentro de la palabra de 14 bits, según la categoría o tipo de instrucción.

En la **figura 3.9** se muestra el formato para cada una de las tres categorías, analizadas en el epígrafe anterior:

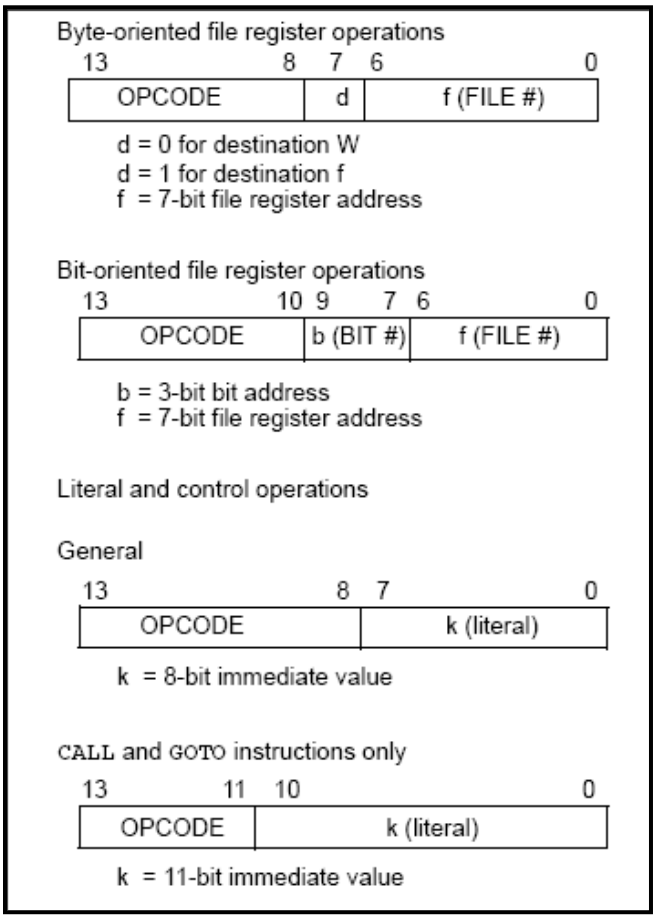


Figura 3.9. Formato general de codificación de las instrucciones de la Familia Media para las categorías descritas. El campo “OPCODE” se refiere al código operativo de la instrucción (la acción que realiza) y el resto de los campos se refieren a los operandos.

Duración de las instrucciones

La mayoría de las instrucciones Microchip se ejecutan en un solo Ciclo de Máquina, que es igual a 4 veces el período del Reloj Externo. Dentro del Juego de Instrucciones existen las siguientes excepciones, que se ejecutan en uno o dos ciclos:

Se ejecutarán siempre en DOS CICLOS DE MAQUINA las siguientes instrucciones:

- "call k". Llamada a una subrutina.
- "goto k". Salto incondicional a la dirección "k".
- "retfie". Retorno de una subrutina de atención a interrupción.
- "return". Retorno de un "call".
- "retlw k". Retorno de un "call" pero grabando el numero "k" en el acumulador.

Se ejecutarán en UNO O DOS CICLOS de instrucción:

- "decfsz f,d". Decrementa el registro "f" y salta la instrucción siguiente si "f" llegó a cero (en este caso dura dos ciclos).
- "incfsz f,d". Similar, pero incrementando "f", o sea se salta la instrucción siguiente cuando "f" sufre un "overflow". En este caso la instrucción demora dos ciclos.
- "btfsc f,b". Hace el test del bit "b" dentro del registro "f" y salta la próxima instrucción cuando este está en cero. En este caso la instrucción dura dos ciclos.
- "btfss f,b". Hace el test del bit "b" dentro del registro "f" y salta la próxima instrucción cuanto este está en uno. En este caso la instrucción dura dos ciclos.

Control del tiempo de ejecución de un programa.

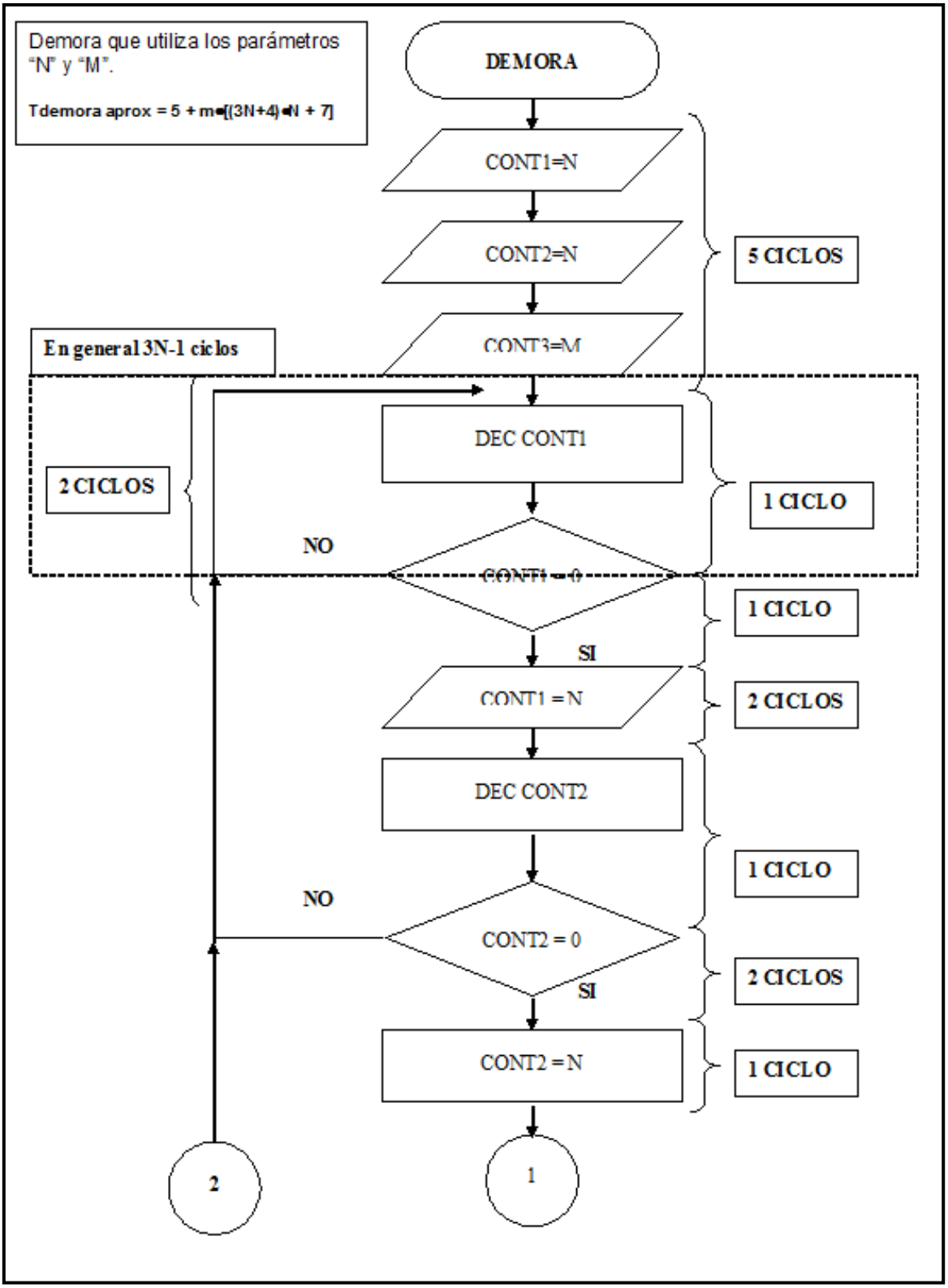
El diseño de demoras por software se basa sencillamente en la generación de lazos de espera cuya duración se calibra en múltiplos del Ciclo de Máquina del Microcontrolador. Como en las aplicaciones prácticas se necesita por lo general gran variedad de tiempos de espera resulta conveniente diseñar demoras de tiempo paramétricas. Una demora paramétrica es aquella cuya duración puede ser definida mediante parámetros de entrada. La subrutina que a continuación se expone pertenece a este tipo de demoras y su duración puede calibrarse fijando el valor de dos parámetros **M** y **N** que deben ser cargados con un valor antes de utilizar la subrutina.

Ejemplo de una variante de demora paramétrica:

La siguiente subrutina genera una demora paramétrica anidando conteos mediante las variables “cont1, cont2, cont3”, que constituyen 3 contadores auxiliares y que deben ser definidas en conjunto con M y N en la sección “definición de símbolos y variables del código”.

```
demora      ;subrutina de DEMORA
            movf N,w
            movwf cont1
            movwf cont2
            movf M,w
            movwf cont3
loop        decfsz cont1
            goto loop
            movf N,w
            movwf cont1
            decfsz cont2
            goto loop
            movf N,w
            movwf cont2
            decfsz cont3
            goto loop
            RETURN
```

Para utilizar la subrutina los parámetros “N” y “M” deben cargarse antes con el valor deseado. En el **figura 3.10** se muestra un Diagrama de Flujo que ilustra el funcionamiento de la subrutina.


Demora que utiliza los parámetros "N" y "M".
Tdemora aprox = 5 + m•[(3N+4)•N + 7]
DEMORA
CONT1=N
CONT2=N
CONT3=M
5 CICLOS
En general 3N-1 ciclos
DEC CONT1
2 CICLOS
1 CICLO
NO
CONT1 = 0
1 CICLO
SI
CONT1 = N
2 CICLOS
DEC CONT2
1 CICLO
NO
CONT2 = 0
2 CICLOS
SI
CONT2 = N
1 CICLO
2
1

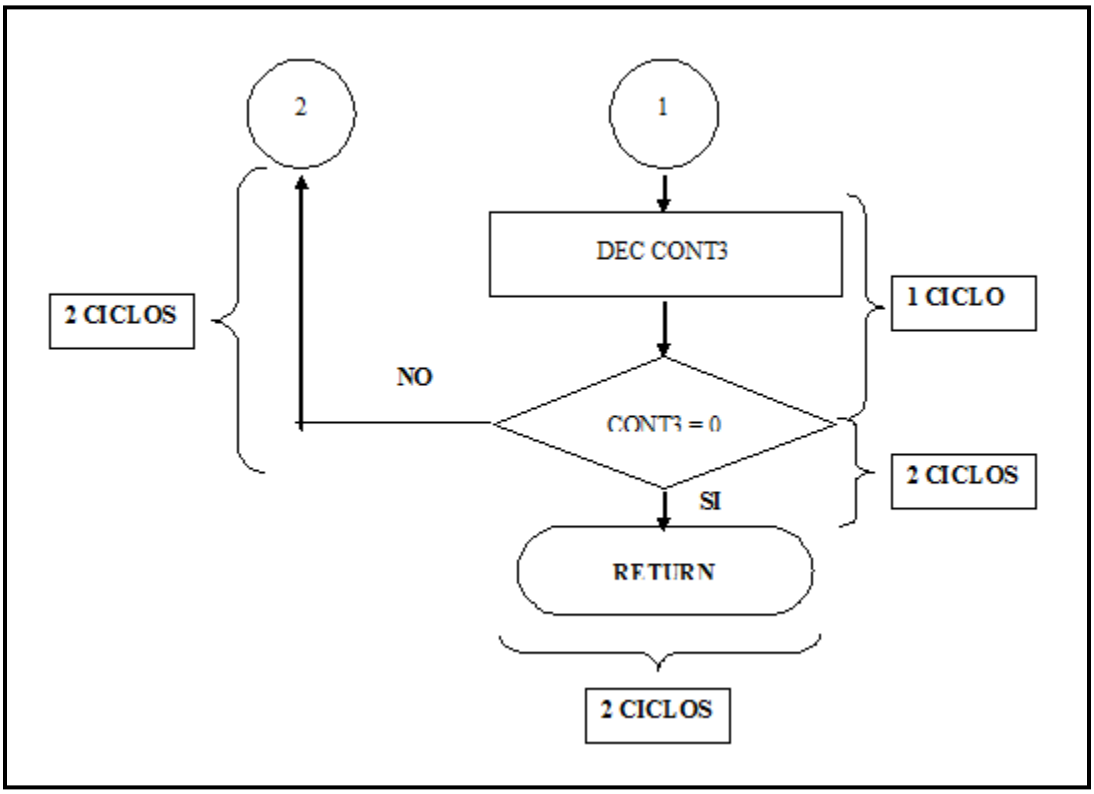


Figura 3.10. Algoritmo funcional de una demora por parámetros.

Métodos para implementar Tablas en la Memoria de Programas.

Estructura de una tabla en la Memoria de Programas.

Una tabla es una subrutina cuya función es guardar valores de 8 ó 16 bits de manera permanente en la Memoria de Programas. La serie 87X utiliza el manejo clásico de tablas denominado "computed GOTO" que permite almacenar valores de hasta 8 bits a pesar de que la extensión de las localizaciones de la memoria de programas es de 14 bits. Por otra parte la serie 18FXXXX incluye el manejo clásico (similar a la serie 87X) y además un esquema mejorado que permite aprovechar los 16 bits de extensión de las localizaciones de su Memoria de Programas para guardar valores cuya extensión pudiera ser de hasta 16 bits.

Estructura de una tabla que utiliza el método "computed GOTO".

Todas las tablas definidas según el método "computed GOTO" comienzan con la instrucción "addwf PCL,1". La instrucción "addwf PCL,1" suma el contenido del registro "W" al registro PCL (Program Counter Low) y coloca el resultado de la suma en el propio registro PCL. El hecho de que el resultado de la suma se coloque en el propio registro PCL provoca un salto dentro de la tabla definida debido a que el valor cargado al registro PCL modificará automáticamente los 8 primeros bits del PC (Program Counter). El salto provocado será igual a: $PCL_{actual} = PCL_{inicial} + W$. El $PCL_{inicial}$ es la parte baja de la dirección dentro de la Memoria de Programas donde está ensamblada la instrucción addwf PCL,1 que es marcada siempre con una etiqueta que da nombre a la tabla. Según lo anterior puede inferirse que el valor que transporta el acumulador "W" hacia la tabla en este caso funciona como el "índice de cada valor de la tabla". El efecto final de la suma "addwf PCL,1" es un salto hacia el valor que debe extraerse de la tabla. El valor que se extrae de la tabla es cargado en "W" por efecto de la instrucción "retlw k"[14]. El siguiente segmento de código ilustra la estructura e una tabla que utiliza el método "computed GOTO".

```
;*************************************************************

;Tabla que sirve para decodificar valores de entrada en BCD a
;códigos "7 ;segmento" para activar segmentos conectados en
;"ánodo común"

cod_ss     addwf PCL,1

           retlw 0xc0            ;"0"

           retlw 0xf9            ;"1"

           retlw 0xa4            ;"2"

           retlw 0xb0            ;"3"

           retlw 0x99            ;"4"

           retlw 0x92            ;"5"

           retlw 0x82            ;"6"

           retlw 0xb8            ;"7"

           retlw 0x80            ;"8"

           retlw 0x98            ;"9"
```

[14] Consulte la sección "Instruction Set Summary" de la Hoja de Datos.

Una manera más compacta para definir una tabla puede lograrse mediante la utilización de la directiva **"DT" (Define Table)**. La directiva "DT" permite omitir la escritura de sucesivas instrucciones **"retlw k"**. La omisión de las instrucciones sólo se produce en el momento de la edición del programa; al compilar serán insertadas las instrucciones **"retlw k"** necesarias.

Una vez construida la tabla, para que pueda ser utilizada correctamente se debe **cargar en "W" el índice del valor a sacar**; llamar a la subrutina tabla y finalmente recoger el valor en el propio registro **"W"**.

En la **figura 3.11** se muestra de manera esquemática el proceso operativo que se efectúa sobre una tabla.

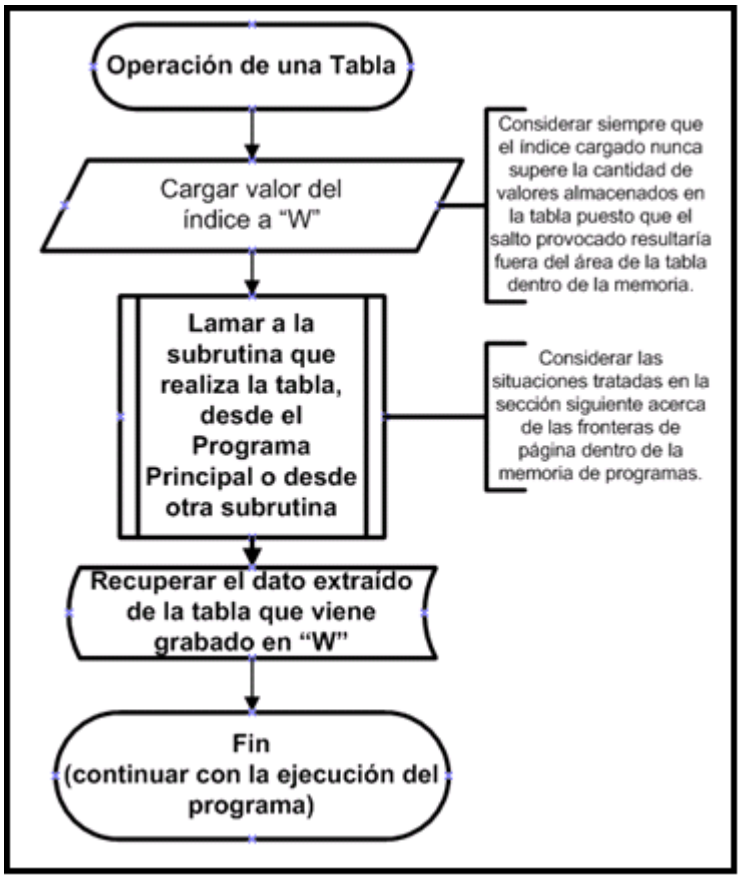


Figura 3.11. Proceso de utilización de una tabla.

Ubicación de las tablas dentro de la Memoria de Programas.

Teóricamente una tabla puede ser ubicada en cualquier lugar de la Memoria de Programas, ahora bien; los "problemas" de acceso a una tabla aparecen en las siguientes situaciones que son totalmente controlables:

- Cuando la tabla se ubica en una página superior a la página cero de la Memoria de Programas.
- Cuando el número de valores de la tabla supera la cantidad de 255.
- Cuando algunos valores de la tabla se encuentren por debajo de la frontera de un bloque de 256 direcciones dentro de una página de memoria y el resto por encima.

Para resolver los problemas anteriores se debe considerar lo siguiente:

La primera situación es la más común por lo que si el micro controlador que se trabaja sólo posee 2K de memoria, no es interesante considerarla dado que su Memoria de Programas sólo posee una página. Si es necesario considerar las situaciones 2 y 3.

Para evitar las situaciones 2 y 3 si la tabla es grande en cantidad de valores es aconsejable ubicarla a partir de una dirección definida dentro de la Memoria de Programas cuyos 8 bits menos significativos comiencen en 00 (PCL i_{nicial} = 00). Para ubicar el inicio de la tabla se puede utilizar la directiva "org XXXX", donde "XXXX" es la dirección dentro de la Memoria de Programas a partir de donde será ubicada la tabla. El siguiente fragmento de código ilustra lo anterior:

```
;=========================================================
             org 0x900; dirección de comienzo de la tabla


;TABLA DE CALIBRACIÓN TEMPERATURA UBICADA A PARTIR DE
;LA DIRECCIÓN 900 DE LA MEMORIA DE PROGRAMAS. LA TABLA
;SE EXTIENDE DESDE LA DIRECCIÓN 900 A LA 9FF, CONTENIENDO
;256 VALORES DE CALIBRACIÓN. EL VALOR DEL PCL VARÍA DESDE
;00h hasta FFh.
;=========================================================
tab_temp     addwf PCL,1
             retlw d'46'         ;valor 0 de la tabla
             retlw d'15'
             retlw d'15'
```

```
retlw d'15'
retlw d'15'
retlw d'15'
retlw d'15'
retlw d'15'
retlw d'15'
retlw d'15'
retlw d'16'
retlw d'16'
retlw d'16'
        •
        •
        •
        •
        •
retlw d'45'
retlw d'45'
retlw d'45'
retlw d'45'
retlw d'45'
retlw d'45'
retlw d'45'
retlw d'45'
retlw d'45'
retlw d'46'
retlw d'46'
retlw d'46'
retlw d'46'
retlw d'46'         ;valor 255 de la tabla
```

Acceso a la tabla.

Como la tabla anterior fue ubicada a partir de la dirección 0x900 de la memoria de programas, en la página 1 dado que los dos bits más significativos de la dirección son: 01 (0x900 hex = 01001 0000 0000). Luego antes de llamar a la tabla es necesario cargar el PCLATH con el valor adecuado: PCLATH = 01001 = 9. La carga del PCLATH se ejemplifica en el siguiente segmento de código:

```
;---------------------------------------
movlw 9
movwf PCLATH ;carga valor del PCLATH
;---------------------------------------
;carga índice en "W"
;---------------------------------------
movf indice,w
call tab_temp
movwf tmp           ;recupera valor de temperatura
clrf PCLATH
```

En general, cuando se trabajan programas largos en micro controladores con memoria superior a 2K (1 página) se recomienda cargar el PCLATH con el valor adecuado antes de llamar a cualquier subrutina (no necesariamente una tabla). Una carga más compacta del PCLATH que funciona para subrutinas que no son tablas se ejemplifica en el siguiente segmento de código:

```
;-----------------------------------
;convierte dato
pagesel convierte
call convierte
```

La directiva **"PAGESEL <nombre de la subrutina que será llamada>"** añade al código las dos instrucciones de carga del PCLATH utilizadas en el caso de la tabla (ejemplo anterior) pero con la ventaja de que el compilador detecta automáticamente al compilar el valor que debe ser cargado en el PCLATH independientemente de la dirección de inicio de la subrutina que será llamada (en este caso **"convierte"**). Así se recomienda siempre en programas que rebasan las 2K instrucciones antes de llamar a cualquier subrutina seleccionar la página.

Tema IV: Periféricos Básicos Integrados a los Microcontroladores de la serie 87X y sus aplicaciones.

Sumario:

1. Temporizadores integrados en los Microcontroladores de la Serie 87X de la Familia Media.
2. Trabajo con Interrupciones.
3. Técnicas de Programación para el control multiplexado de visualizadores de Leds de 7 segmentos por interrupción de un temporizador.
4. Técnicas de Programación para lectura de Teclados Lineales y Matriciales.
5. Técnicas de Programación para manejo de Visualizadores de Tinta Liquida (LCD).
6. Diseño de aplicaciones utilizando temporización por hardware e interrupciones. Diseño de contadores utilizando Visualizadores de Leds de 7 segmentos y Visualizadores de Tinta Liquida (LCD). Diseño de aplicaciones utilizando Teclados Matriciales. Técnicas de decodificación aplicadas al diseño de Sistemas de Seguridad.
7. Modulo de EEPROM de datos y Tratamiento de Memoria Flash. Estructura, Registros de Control y Programación de Aplicaciones.
8. Modulo CCP (Compara / Captura /PWM). Estructura, Registros de Control y Programación de Aplicaciones. Diseño de Aplicaciones con módulo CCP.
9. Diseño de Aplicaciones Integradas utilizando Visualizadores de Tinta Liquida (LCD).

Temporizadores integrados en los Microcontroladores de la Serie 87X de la Familia Media.

En los microcontroladores Microchip de la Serie 87X de la familia media existen tres temporizadores “timers” denominados TMR0, TMR1, TMR2 cada uno de ellos tiene características particulares, a continuación se describen los temporizadores integrados en el PIC de la serie 87X.

Módulo Timer0.

El temporizador/contador Timer0 tiene las siguientes características:

- Contador/temporizador de 8 bits.
- Se puede leer y escribir.
- Posee un prescaler de 8 bits programable por software.
- Selección interna o externa del reloj.
- Interrupción en desborde “overflow” desde FFh a 00h.
- Selección del límite para el reloj externo.

El modo temporizador es seleccionado mediante la limpieza del bit T0CS (OPTION_REG<5>). En el modo temporizador, el modulo Timer0 se incrementará cada ciclo de instrucción (sin prescaler). Si el registro TMR0 es escrito, el incremento es inhibido por los siguientes dos ciclos de instrucción.

El modo contador es seleccionado seteando el bit T0CS (OPTION_REG<5>). En el modo contador, el Timer0 se incrementará tanto cuando exista una subida o bajada del límite existente en el pin RA4/T0CKI. El límite de incremento es determinado por el bit de selección de fuente límite del Timer0, T0SE (OPTION_REG<4>). Limpiando el bit T0SE se selecciona la subida del límite.

El prescaler es mutuamente compartido entre el Timer0 y el Watch Dog Timer. El prescaler no se puede leer o escribir.

Interrupción del Timer0.

La interrupción del TMR0 es generada cuando el registro TMR0 se desborda desde FFh a 00h. Este desborde setea el bit T0IF (INTCON<2>). La interrupción puede ser enmascarada limpiando el bit T0IE (INTCON<5>). El bit T0IF debe ser limpiado en el programa por la rutina de servicio de interrupción del modulo Timer0 antes de rehabilitar esta interrupción. La interrupción del TMR0 no puede despertar al procesador desde la condición SLEEP, esta se debe a que el Timer está apagado durante el SLEEP.

Usando el Timer0 con un reloj externo.

Cuando el prescaler no está siendo usado, la entrada del reloj interno es la misma que la salida del prescaler. La sincronización de T0CKI con la fase interna del reloj se logra mediante el muestreo de la salida del prescaler en los ciclos Q2 y Q4 de la fase del reloj interno. Sin embargo, es necesario que el T0CKI se mantenga en alto por lo menos durante 2 Tosc y que también se mantenga en bajo al menos durante 2 Tosc.

Prescaler

Existe un único prescaler disponible, el cual es compartido entre el Timer0 y el Watch Dog Timer pero es mutuamente excluyente, es decir si una función utiliza el prescaler del modulo del Timer0 significa que no existirá un prescaler para el Watch Dog Timer y viceversa. Este prescaler no se puede leer o escribir (**figura 4.1**).

El PSA y los bits PS2:PS0 (OPTION_REG<3:0>) determinan la asignación del prescaler y la relación del prescaler.

Cuando ha sido asignado al Timer0, todas las instrucciones que escriben al registro TMR0 limpiaran al prescaler. Cuando el prescaler se asigna al WDT, una instrucción CLRWDT limpiara el prescaler junto con el WDT.

Recuerde que cuando se escribe en el TMR0 (y se encuentra asignado el prescaler a éste) se limpiara el conteo del prescaler pero no cambiara la asignación del prescaler.

REGISTER 5-1: OPTION_REG REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 $\overline{\text{RBPU}}$

bit 6 **INTEDG**

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
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

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note: To avoid an unintended device RESET, the instruction sequence shown in the PICmicro™ Mid-Range MCU Family Reference Manual (DS33023) must be executed when changing the prescaler assignment from Timer0 to the WDT. This sequence must be followed even if the WDT is disabled.

Figura 4.1. Registro Option_Reg. Fuente "PIC 16F87X Data Sheet"

Módulo Timer1.

El modulo Timer1 es un contador/temporizador que consiste de dos registros de 8 bits (TMR1H y TMR1L), el cual se puede leer o escribir. El par de registros del TMR1 (TMR1H:TMR1L) se incrementa desde 0000h hasta FFFFh y retorna a 0000h. La interrupción del TMR1, si está habilitada, es generada por desbordamiento "overflow", la cual es reflejada en el bit de la bandera de interrupción TMR1IF (PIR<0>). Esta interrupción puede ser habilitada/deshabilitada mediante seteo/limpieza del bit de habilitación de interrupción del TMR1IE (PIE1<0>).

El Timer1 puede operar en uno de los dos siguientes modos:

- Como un temporizador
- Como un contador

El modo de operación es determinado por el bit de selección del reloj, TMR1CS (T1CON<1>).

En el modo temporizador, el Timer1 se incrementa cada ciclo de instrucción. En modo contador, se incrementa en cada subida del límite del reloj externo.

El Timer1 puede ser habilitado/deshabilitado mediante el seteo/limpieza del bit de control TMR1ON (T1CON<0>).

El Timer1 también tiene una entrada de RESET interna. Este RESET puede ser generado por cualquiera de los dos módulos CCP. El registro de control del Timer1 se muestra en la **figura 4.2**.

Cuando el oscilador del Timer1 es habilitado (T1OSCEN es seteado), los pines RC1 y RC0 se convierten en entradas. Es decir, los valores del TRISC son ignorados y estos pines se leen como ceros.

REGISTER 6-1: T1CON: TIMER1 CONTROL REGISTER (ADDRESS 10h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **T1CKPS1:T1CKPS0**: Timer1 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **T1OSCEN**: Timer1 Oscillator Enable Control bit
1 = Oscillator is enabled
0 = Oscillator is shut-off (the oscillator inverter is turned off to eliminate power drain)

bit 2 **$\overline{\text{T1SYNC}}$**: Timer1 External Clock Input Synchronization Control bit
When TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS**: Timer1 Clock Source Select bit
1 = External clock from pin RC0/T1OSO/T1CKI (on the rising edge)
0 = Internal clock (Fosc/4)

bit 0 **TMR1ON**: Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Figura 4.2. Registro de control del Timer1. Fuente "PIC 16F87X Data Sheet".

Operación del Timer1 en modo temporizador.

El modo temporizador es seleccionado mediante la limpieza del bit TMR1CS (T1CON<1>), en este modo, la entrada del reloj al temporizador es Fosc/4. El bit de control de sincronización T1SYNC (T1CON<2>) no tiene ningún efecto, esto es debido a que el reloj interno esta siempre en sincronía.

Operación del Timer1 en modo contador.

El Timer1 puede operar tanto en modo sincrónico o asincrónico, dependiendo del estado del bit TMR1CS.

Cuando el Timer1 está siendo incrementado mediante una fuente externa, los incrementos ocurren en un incremento del flanco. Después de que el Timer1 sea habilitado en modo contador, el módulo debe tener primero un flanco de bajada antes de que el contador empiece con el incremento.

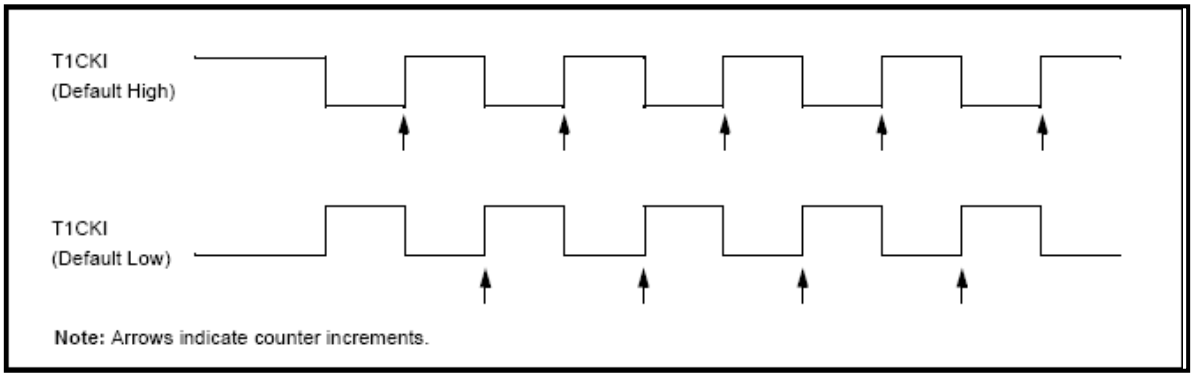


Figura 4.3. Flancos de Incremento del Timer1. Fuente "PIC 16F87X Data Sheet".

Operación del Timer1 en modo de contador sincronizado.

El modo contador puede ser seleccionado seteando el bit TMR1CS, en este modo el Timer se incrementa en cada subida de flanco del reloj en el pin RC1, cuando el bit T1OSCEN es seteado, ó en el pin RC0 cuando el bit T1OSCEN es limpiado.

Si T1SYNC es limpiado, la entrada del reloj externo es sincronizada con la fase del reloj interno. La sincronización esta después de la etapa del prescaler. La etapa del prescaler es un contador de ripple asincrónico.

En esta configuración, durante el modo SLEEP, el Timer1 no se incrementará incluso si el reloj externo está presente, debido a que la sincronización del circuito está apagada. El prescaler, sin embargo, continuará con el incremento.

Operación del Timer1 en modo de contador asincrónico.

Si el bit de control T1SYNC (T1CON<2>) se encuentra en “uno”, la entrada externa del reloj no está sincronizada. El Timer continúa incrementando asincrónicamente la fase del reloj interno. El Timer continuara “corriendo” incluso durante el “SLEEP” y puede generar una interrupción de desbordamiento, la cual despertará al procesador. Sin embargo se debe tener ciertas precauciones en el programa para leer o escribir el Timer.

En el modo de contador asincrónico, el Timer1 no puede ser usado como una base de tiempo para operaciones de captura o comparación (CCP).

Lectura/escritura del Timer1 en modo de contador asincrónico.

Leer TMR1H y TMR1L mientras el Timer está operando desde un reloj asincrónico garantizará una lectura valida. Sin embargo, se debe tener en cuenta que la lectura de un Timer de 16 bits, en dos valores de 8 bits, genera ciertos problemas debido a que el Timer puede desbordarse entre las lecturas.

Para la escritura, es recomendado que se detenga al Timer y se escriba los valores deseados. Una “disputa” de escritura puede ocurrir, por la escritura de los registros del Timer, mientras el registro se está incrementando. Esto puede producir un valor impredecible en el registro del Timer.

Reseteando el Timer1 usando una entrada de disparo “trigger” CCP

Si los módulos CCP1 o CCP2 están configurados en modo “Compara Captura” para generar un “disparo por evento especial” (CCP1M3:CCP1M0 = 1011), esta señal reseteará el Timer1.

Tome en cuenta que el disparo por evento especial desde los módulos CCP1 y CCP2 no habilitará la bandera de interrupción TMR1IF (PIR <0>).

El Timer1 deberá estar configurado ya sea en modo temporizador o modo contador sincrónico para poder utilizar esta característica. Si el Timer1 está funcionando en modo contador asincrónico esta operación de RESET no funcionará.

Si sucede un evento de escritura en el Timer1 que coincide con un disparo de evento especial desde CCP1 o CCP2, la escritura se realizara primero y luego se procederá a dar atención al disparo por evento especial.

Reseteando el par de registros (TMR1H, TMR1L) del Timer1.

Los registros TMR1H y TMR1L no son resetados hacia 00h o en POR o cualquier otro RESET, excepto por los disparos de evento especial de CCP1 y CCP2.

T1CON es reseteado hacia 00h en un RESET de encendido “Power-on RESET” o en un “Brown-out RESET”, los cuales apagan el Timer y dejan un prescale de 1:1. En todos los demás RESETS, el registro no se ve afectado.

Prescaler del Timer1.

El contador del prescaler es limpiado cuando existe escritura en los registros TMR1H o TMR2L.

Módulo Timer2.

Timer2 es un temporizador de 8 bits con un prescaler y un postscaler. Puede ser usado como la base de tiempo PWM para el modo PWM del modulo CCP. El registro TMR2 es editable (se puede leer y escribir), y es limpiado con cualquier RESET del equipo.

La entrada del reloj (Fosc/4) tiene opciones de prescale de 1:1, 1:4 o 1:16, las cuales son seleccionadas por los bits de control T2CKPS1:T2CKPS0 (T2CON<1:0>) en la **figura 4.4** se puede apreciar el registro T2CON.

REGISTER 7-1: T2CON: TIMER2 CONTROL REGISTER (ADDRESS 12h)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6-3 **TOUTPS3:TOUTPS0**: Timer2 Output Postscale Select bits
0000 = 1:1 Postscale
0001 = 1:2 Postscale
0010 = 1:3 Postscale
•
•
•
1111 = 1:16 Postscale

bit 2 **TMR2ON**: Timer2 On bit
1 = Timer2 is on
0 = Timer2 is off

bit 1-0 **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits
00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Figura 4.4. Registro de control T2CON del Timer2. Fuente “PIC16F87X Data Sheet”.

El Timer2 tiene un registro de periodo de 8 bits, PR2. El Timer2 se incrementará desde 00h hasta que coincida con PR2 y entonces retornara a 00h en el siguiente incremento del ciclo. PR2 es un registro que se puede leer y escribir.

La salida del TMR2 va a través de un postscaler de 4 bits (el cual da una escala de 1:1 hasta 1:16) para generar una interrupción del TMR2 (la cual activa la bandera de interrupción TMR2IF, (PIR<1>)). Ver **figura 4.5**.

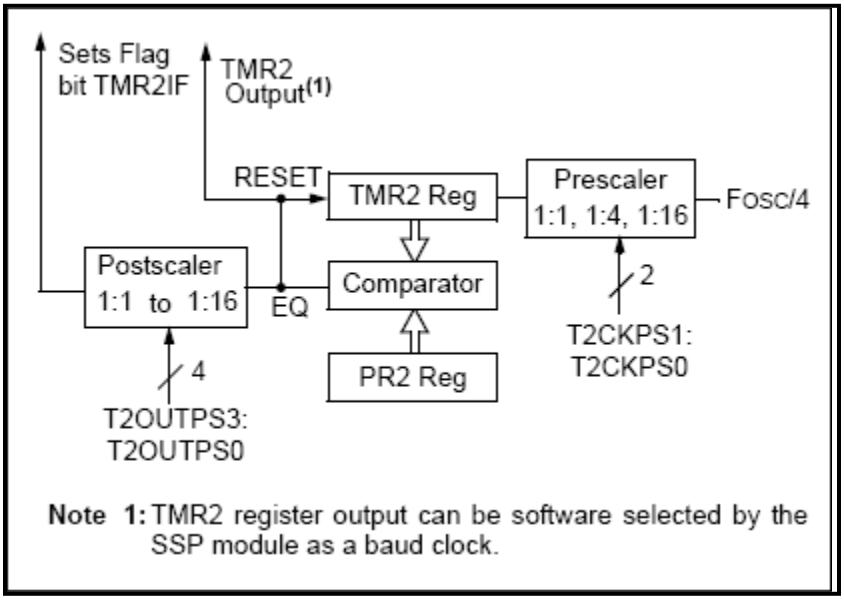


Figura 4.5. Diagrama de bloques del Timer2. Fuente "PIC 16F87X Data Sheets"

El Timer2 puede ser apagado limpiando el bit de control TMR2ON (T2CON<2>), para minimizar el consumo de energía.

Prescaler y postscaler del Timer2.

Los contadores del prescaler y postscaler son limpiados cuando alguno de los siguientes eventos ocurren:

- Una escritura en el registro TMR2.
- Una escritura en el registro T2CON.
- Cualquier RESET del equipo (POR, MCLR RESET, WDT RESET o BOR).

El TMR2 no es limpiado cuando T2CON es escrito.

Salida del TMR2

La salida del TMR2 (antes del postscaler) esta alimentando al modulo SSP, el cual opcionalmente usa la salida para generar un cambio de reloj.

Trabajo con Interrupciones.

El trabajo con interrupciones se caracteriza por ser un servicio directo entre periféricos y el CPU, siempre que esta “desee establecer el dialogo con el periférico”. En términos concretos, esto significa que la interrupción del periférico había sido habilitada. La característica fundamental y ventaja principal del servicio de interrupciones es la inmediatez; lo que significa que pueden eliminarse total o parcialmente los ciclos de consulta. Otra ventaja es que puede inhibirse la interrupción del periférico dado cuando se considera que esta es “inoportuna” y por tanto perjudica en alguna medida la marcha del proceso principal, o sea la corrida del resto del programa. Esta forma de trabajo es indispensable en el trabajo del control de procesos en tiempo real.

Formas de iniciar una Interrupción.

Por un periférico o entidad externa ajena al Microcontrolador a través de los pines específicos del mismo. En estas condiciones la interrupción es externa. Dentro del propio MC, entonces la interrupción es interna.

Fuentes de Interrupción en los PICs de la Serie 87X.

Los PICs pueden manejar diversas fuentes de interrupción (IT). Por lo general con cada periférico integrado al Microcontrolador hay asociada una o más fuentes de interrupción.

A continuación se resumen todas las fuentes de IT posibles en los microcontroladores de esta serie. Los tipos de fuente de IT se subdividen en dos categorías: Básicas y Adicionales.

Fuentes Básicas.

- Interrupción por el pin externo (pin RB0/INT). El pin RB0 es un pin de puerto que está multiplexado con la función de INT externa.
- Por desbordamiento del TMR0.
- Por cambio en los pines de puerto RB7 : RB4.
- Cuando se completa un ciclo de escritura en la EEPROM de Datos. La EEPROM de Datos es otro periférico muy común en los microcontroladores de esta familia. Es una memoria tipo EEPROM y por lo tanto no volátil, que puede escribirse y leerse desde el programa de aplicación.

Fuentes adicionales.

- INT por desbordamiento del TMR1.
- INT por desbordamiento del TMR2.
- Por cambio en el Módulo Comparador (este se refiere a un comparador analógico que puede estar incluido en algunos Microcontroladores de esta familia).
- INT del Puerto Paralelo Esclavo.
- Interrupciones varias del USART (Universal Sincronous Asincronous Reciver Tx).
- INT por completamiento de Ciclo de Cconversion en los conversores A/D.
- INT del Módulo LCD.
- INT por el Módulo CCP.
- INT por el Módulo SSP (Serial Sincronous Port).

Ciclos de Interrupción en los PICs de la Serie 87X.

En los Microcontroladores PICs cuando algún periférico (interno o externo) solicita INT, ocurre la siguiente secuencia de eventos:

1. Se completa la ejecución de la instrucción que se está ejecutando en ese momento.
2. Se incrementa el PC y se guarda en la PILA (STACK).
3. El PC se carga con la dirección 0004 de la Memoria de Programas (PC = 0004), que es donde se encuentra ensamblado el "Vector de Interrupción". El Vector de Interrupción puede ejecutar una instrucción de salto[15] (goto INT, por ejemplo) a partir de la cual se ensamblo la subrutina de atención a interrupción o la primera instrucción de la subrutina de atención a interrupción (también denominada de "Servicio de Interrupción").
4. Una vez dentro de la subrutina de atención a interrupción se procede a identificar la fuente de interrupción (encuestando las banderas correspondientes) y a "atender" el periférico que en ese momento solicita la interrupción. Para ello todos los PICs poseen registros donde se encuentran las banderas que sirven de aviso sobre el periférico que generó el pedido de interrupción.
5. Finalmente antes de abandonar la subrutina de atención a interrupción, deben limpiarse las banderas del o los periféricos que fueron atendidos en ella para evitar una interrupción reentrante. De la subrutina se sale con la instrucción "retfie", que es la encargada de cargar el PC con el valor de retorno que había sido guardado en la PILA.

[15] El salto a una IT demora 3 ciclos de Máquina.

Registros para el Manejo de Interrupciones en los PICs.

Para el manejo de IT en los PICs existen dos tipos de registros especiales: los Registros de Control de Interrupción y los Registros Banderas de Interrupción. Los primeros permiten habilitar las distintas fuentes de IT y los segundos son necesarios para averiguar cual periférico generó la IT. La manera más fácil de estudiar estos registros es relacionarlos con el periférico del Microcontrolador que se esté empleando.

Registro de Control de Interrupciones Básicas (INTCON).

El registro INTCON (**figura 4.6**), aparece desde los procesadores más primitivos de la familia media (PIC 16F84, por ejemplo). En este registro se incluyen los bits para habilitar las fuentes básicas de interrupción, así como los bits que sirven de banderas para cada uno de las fuentes individuales de interrupción.

REGISTER 2-3: INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6 **PEIE**: Peripheral Interrupt Enable bit
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts

bit 5 **T0IE**: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4 **INTE**: RB0/INT External Interrupt Enable bit
1 = Enables the RB0/INT external interrupt
0 = Disables the RB0/INT external interrupt

bit 3 **RBIE**: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **T0IF**: TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INTF**: RB0/INT External Interrupt Flag bit
1 = The RB0/INT external interrupt occurred (must be cleared in software)
0 = The RB0/INT external interrupt did not occur

bit 0 **RBIF**: RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state; a mismatch condition will continue to set the bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared (must be cleared in software).
0 = None of the RB7:RB4 pins have changed state

Figura 4.6. Registro INTCON. Fuente: Datasheet Serie 87X.

El registro INTCON se encuentra en las localizaciones: 0Bh, 8Bh, 10Bh y 18Bh de los bancos 0, 1, 2 y 3 de la Memoria de Datos; por lo tanto para escribir o leer su contenido

NO es necesario cambiar el banco modificando los bits correspondientes del registro STATUS.

Registros para el Control y Banderas de Interrupciones Adicionales.

La denominación de los registros para el control de interrupciones adicionales se codifica de manera general con las letras PIE (Peripheral Interrupt Enable Register), más un número (por ejemplo PIE1, PIE2, etc). En un mismo PIC pueden existir varios de estos registros en función de la cantidad de periféricos integrados al MC que sean capaces de generar interrupción.

Por otra parte, las banderas que indican los eventos de interrupción adicionales, se agrupan en registros tipo PIR (Peripheral Interrupt Registers). De manera similar a lo que ocurre con los registros tipo PIE, pueden existir varios registros PIR (por ejemplo, PIR1, PIR2, etc).

Las **figuras 4.7 a** y **b** ilustran dos tipos de registros: uno de control (PIE1) y otro de banderas (PIR1). Note que los bits de habilitación y bandera se encuentran en posiciones similares dentro de ambos registros; por ejemplo: el bit 0 del registro PIE1, sirve para habilitar la interrupción del timer1 y el bit 0 del registro PIR1 sirve de bandera de interrupción por desbordamiento del timer1.

REGISTER 2-4: PIE1 REGISTER (ADDRESS 8Ch)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

bit 7 **PSPIE(1):** Parallel Slave Port Read/Write Interrupt Enable bit
1 = Enables the PSP read/write interrupt
0 = Disables the PSP read/write interrupt

bit 6 **ADIE**: A/D Converter Interrupt Enable bit
1 = Enables the A/D converter interrupt
0 = Disables the A/D converter interrupt

bit 5 **RCIE**: USART Receive Interrupt Enable bit
1 = Enables the USART receive interrupt
0 = Disables the USART receive interrupt

bit 4 **TXIE**: USART Transmit Interrupt Enable bit
1 = Enables the USART transmit interrupt
0 = Disables the USART transmit interrupt

bit 3 **SSPIE**: Synchronous Serial Port Interrupt Enable bit
1 = Enables the SSP interrupt
0 = Disables the SSP interrupt

bit 2 **CCP1IE**: CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1 **TMR2IE**: TMR2 to PR2 Match Interrupt Enable bit
1 = Enables the TMR2 to PR2 match interrupt
0 = Disables the TMR2 to PR2 match interrupt

bit 0 **TMR1IE**: TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

Note 1: PSPIE is reserved on PIC16F873/876 devices; always maintain this bit clear.

Figura 4.7a. Registro de habilitación de interrupciones PIE1. Fuente: Datasheet Serie 87X.

REGISTER 2-5: PIR1 REGISTER (ADDRESS 0Ch)

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

bit 7 **PSPIF(1):** Parallel Slave Port Read/Write Interrupt Flag bit
1 = A read or a write operation has taken place (must be cleared in software)
0 = No read or write has occurred

bit 6 **ADIF**: A/D Converter Interrupt Flag bit
1 = An A/D conversion completed
0 = The A/D conversion is not complete

bit 5 **RCIF**: USART Receive Interrupt Flag bit
1 = The USART receive buffer is full
0 = The USART receive buffer is empty

bit 4 **TXIF**: USART Transmit Interrupt Flag bit
1 = The USART transmit buffer is empty
0 = The USART transmit buffer is full

bit 3 **SSPIF**: Synchronous Serial Port (SSP) Interrupt Flag
1 = The SSP interrupt condition has occurred, and must be cleared in software before returning from the Interrupt Service Routine. The conditions that will set this bit are:
- SPI
  - A transmission/reception has taken place.
- I^2C Slave
  - A transmission/reception has taken place.
- I^2C Master
  - A transmission/reception has taken place.
  - The initiated START condition was completed by the SSP module.
  - The initiated STOP condition was completed by the SSP module.
  - The initiated Restart condition was completed by the SSP module.
  - The initiated Acknowledge condition was completed by the SSP module.
  - A START condition occurred while the SSP module was idle (Multi-Master system).
  - A STOP condition occurred while the SSP module was idle (Multi-Master system).

0 = No SSP interrupt condition has occurred.

bit 2 **CCP1IF**: CCP1 Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused in this mode

bit 1 **TMR2IF**: TMR2 to PR2 Match Interrupt Flag bit
1 = TMR2 to PR2 match occurred (must be cleared in software)
0 = No TMR2 to PR2 match occurred

bit 0 **TMR1IF**: TMR1 Overflow Interrupt Flag bit
1 = TMR1 register overflowed (must be cleared in software)
0 = TMR1 register did not overflow

Note 1: PSPIF is reserved on PIC16F873/876 devices; always maintain this bit clear.

Figura 4.7b. Registro de banderas de interrupciones PIR1. Fuente: Datasheet Serie 87X.

Técnicas de Programación para el control multiplexado de visualizadores de Leds de 7 segmentos por interrupción de un temporizador.

Estructura y tecnologías de los Displays de 7 Segmentos de Leds.

Un Display de 7 segmentos de Leds es un arreglo de Leds dispuesto como se muestra en la **figura 4.8.** Los segmentos se codifican en sentido horario empleando las letras desde la a hasta la g y PD si el display contiene punto decimal. Cada display posee un terminal común (COM) que sirve para activar o desactivar el display.

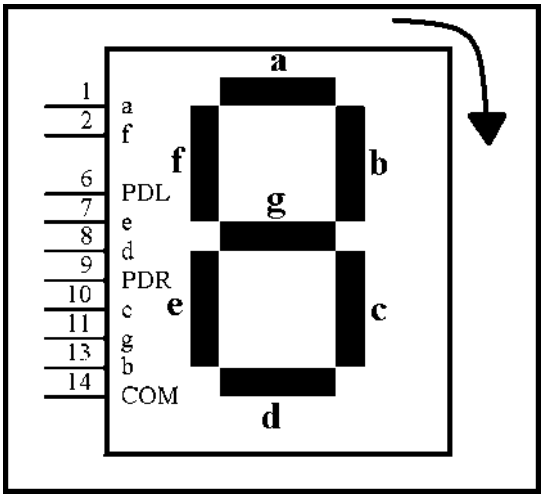


Figura 4.8. Esquema estándar para un Display de 7 segmentos.

Existen dos tecnologías de displays: cátodo común y ánodo común. En el primer caso todos los leds son activos con uno lógico y el común es activo con cero. En el segundo los leds se activan con cero y el común con un uno lógico.

Dada la cantidad de corriente que circula por un común es necesario colocar un “buffer” a las líneas de puerto que se empleen para activar los comunes. Por lo general se emplea un transistor como “buffer” PNP o NPN dependiendo de la tecnología tal y como se muestra en la **figura 4.9**.

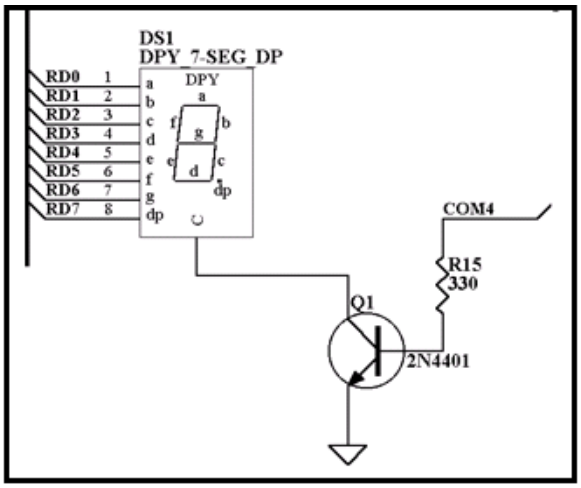


Figura 4.9. Esquema de activación para el común de un display de 7 segmentos de tecnología cátodo común.

Tratamiento Multiplexado de Displays de 7 Segmentos.

Tradicionalmente los displays de 7 segmentos de leds eran tratados de forma independiente, utilizando 8 líneas de datos para encender cada segmento. La conexión individual de displays resulta muy engorrosa dada la cantidad de líneas que se involucran en ella.

El empleo de un μC permite implementar el manejo de displays de 7 segmentos multiplexado. El manejo multiplexado supone que se utiliza un único bus de datos (en la Tarjeta de Entrenamiento el puerto D) para enviar el dato en código 7 segmento a los segmentos. Desde el punto de vista del hardware esto significa que los segmentos de los N displays involucrados se conectan en paralelo. Cada segmento (led) debe conectarse a través de una resistencia limitadora (de unos 220 Ω). Adicionalmente se necesitan líneas para manejar de manera individual los comunes de cada display.

El firmware para tratar displays multiplexados enciende en cada momento el display cuyo dato ha sido enviado al bus (puerto D). La secuencia de encendido se realiza a una frecuencia lo suficientemente alta como para provocar la sensación de persistencia de los datos en la retina. En la **figura 4.10** se esquematiza el Método de Refrescamiento Multiplexado de displays.

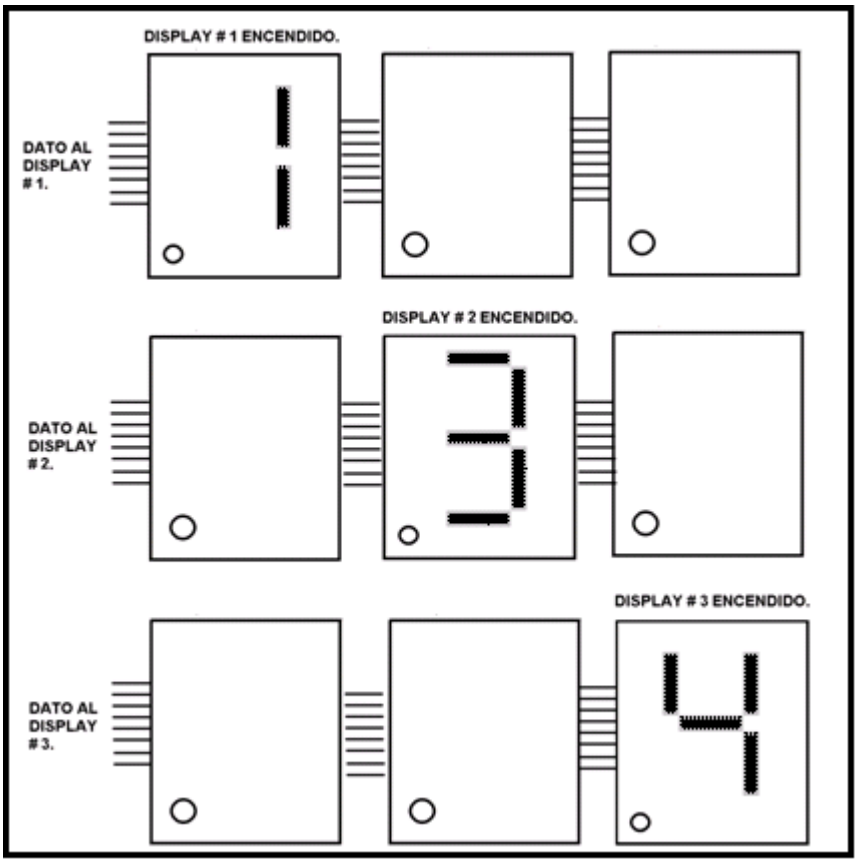


Figura 4.10. Esquema que ilustra el tratamiento multiplexado de displays de 7 segmentos.

El refrescamiento de la información en los displays normalmente se realiza mediante la interrupción de un temporizador que ha sido programado con una base de tiempo determinada. El empleo de interrupción del temporizador se justifica con el hecho de que el refrescamiento es un proceso rutinario al cual el lazo principal del programa no debe estar dedicado puesto que se produciría pérdida de atención a otros procesos.

La subrutina que se discute a continuación normalmente se ejecuta cada vez que el temporizador designado para marcar el tiempo de refrescamiento genera una interrupción.

Ejemplo de subrutina de refrescamiento.

En líneas generales la subrutina basa su funcionamiento en las siguientes técnicas:

- La variable “nlamp” es un contador que almacena el número del display que en cada momento se está refrescando en nuestro caso nlamp$\in$ [0,5], dado que existen seis displays. Cada vez que se genera una interrupción se refresca la

información de un display. El Ciclo de Refrescamiento se muestra en la **figura 4.11** La información en los displays se refresca de uno en uno a intervalos de tiempo "T" (dado por el temporizador).

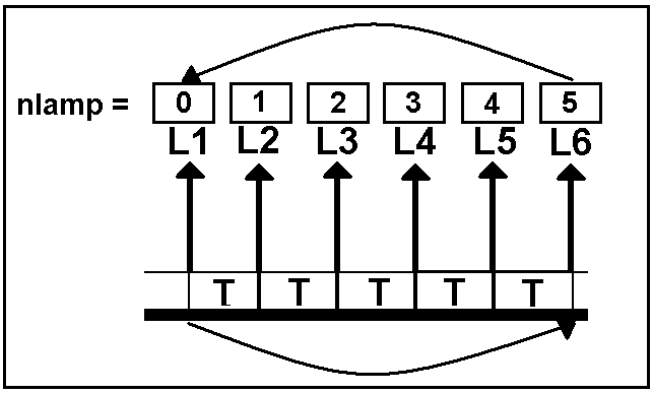


Figura 4.11. Esquema del Ciclo de Refrescamiento de Displays.

Básicamente el algoritmo de funcionamiento de la subrutina se concentra en cinco procesos fundamentales:

1. Apagar todos los displays al inicio. Se hace con el objetivo de que los comunes estén en un estado estable antes de encender el display que corresponde refrescar en cada momento para evitar corrientes transitorias de alto valor que puedan provocar fallas en los transistores que manejan los comunes. Iniciar el FSR en la dirección base de la RAM de Display (en este caso FSR = 0x20). El siguiente fragmento de código ilustra el proceso.

```
            ;apaga comunes del puerto C
refresh     bsf PORTC,3
            bsf PORTC,4
            bsf PORTC,5
            ;apaga comunes del puerto E
            movlw 0xff
            movwf 9
            ;--------------------------------
            ;inicia el FSR
            movlw 0x20
            movwf FSR
```

2. Activar el común del display que debe visualizar el dato colocado en el BUS. Como los displays en la Tarjeta no están colocados en líneas del mismo puerto, se recomienda utilizar otra tabla para almacenar el patrón de bits que debe ser cargado al puerto que maneja el común del display refrescado (PUERTO C ó PUERTO E). Para accesar la tabla se utiliza la misma variable "nlamp" como índice

de la tabla de códigos de activación de comunes. Una vez que el código de activación se extrae de la tabla el software debe decidir a qué puerto debe enviarse, dependiendo de si se está refrescando uno de los displays de los grupos L1, L2 ó L3 cuyos comunes se conectan a RC3, RC4 y RC5 ó uno de los displays de los grupos L4, L5 ó L6 cuyos comunes se conectan a RE0, RE1 y RE2.

```
;chequea puerto de comunes (C ó E), en función de nlamp
;----------------------------------------
;nlamp >= 2?
movf nlamp,0
sublw 2
btfsc 3,2
goto menor        ;nlamp = 2
btfss 3,0         ;pregunta por carry, es >< 2
goto mayor_2

;carry = 1 ==> nlamp = < 2

menor       movf nlamp,0
call cod_act
andlw 7           ;máscara para eliminar puerto E
movwf temp
bcf 3,0           ;limpia carry antes de rotar
rlf temp
rlf temp
rlf temp          ;rota izq


;trucaje para no afectar dato
movf temp,0       ;leo cod act
andwf 7,1         ;sumo para no afectar
goto dato

;carry = 1 nlamp > 2

mayor_2     movf nlamp,0
call cod_act
clrf temp
movwf temp        ;respalda valor
bcf 3,0           ;limpia carry antes de rotar
rrf temp
rrf temp
rrf temp
movf temp,w       ;cod act al acum.
movwf 9           ;cod. de act al puerto E
```

3. Extraer el dato desde la RAM de Display para decodificarlo mediante una tabla. Para extraer el dato se utiliza la Técnica de Direccionado Indirecto de la RAM para facilitar la programación. El siguiente fragmento de código ilustra el proceso.

```
;sacar el dato para la lámpara nlamp
dato        movf nlamp,w
addwf FSR,1
movf INDF,0
call cod_ss ;decodificar dato
```

4. Colocar el dato en código 7 segmentos en el BUS de Datos (en la Tarjeta de Entrenamiento el Puerto D). Pudiera ser necesario complementar el dato cargado al BUS dependiendo de la tecnología de displays utilizada (cátodo o ánodo común).

```
movwf 8     ;coloca dato en el PUERTO D
```

5. Finalmente, chequear el valor de “nlamp”. Si “nlamp” = 5 significa que se han refrescado todos los displays. En este caso debe limpiarse “nlamp” (“nlamp” = 0). Si “nlamp” < 5 significa que NO se han refrescado todos los displays por lo que debe darse paso al próximo en la próxima interrupción del temporizador. En este caso debe incrementarse “nlamp” (“nlamp” = “nlamp” + 1). Para que el proceso sea sincronizado no debe olvidarse limpiar “nlamp” en el bloque de condiciones iniciales del sistema.

```
            ;prox lampara
            movf nlamp,w
            sublw 5
            btfss 3,2
            goto menor_5

            ;nlamp = 0
            clrf nlamp
            goto back_t0

            ;nlamp < 5
menor_5     incf nlamp
```

El fragmento de código siguiente representa toda la subrutina de Atención a interrupción del temporizador 0. El código de refrescamiento comienza concretamente a partir de la etiqueta “refresh”. También se anexa debajo la subrutina tabla que almacena los códigos de activación.

```
;*****************************************************************************
;subrutina de Atención a Interrupción. Este caso incluye la decodificación de
;BCD a 7 segmentos
;Atención al timer0, cada vez que se desborda; para refrescar los displays ;cada
1ms.
;*****************************************************************************
IT
;-----------------------------------------------------------------------------
            ;respaldo de registros.
            movwf wtemp         ;respaldo ac.
            swapf STATUS,w
            movwf stat_temp
            movf FSR,0
            movwf fsr_temp      ;respaldo el FSR
            ;-------------------------------------------------------------
            ;encuesta de banderas.
            btfss INTCON,2      ;T0IF =? 0
            goto back_t0 ;refresca el display
            ;INT del timer 0: Rfrescar display.
            ;apaga lámparas al inicio.
            ;-------------------------------------------------------------
            ;apaga comunes del puerto C
refresh     bsf PORTC,3
            bsf PORTC,4
            bsf PORTC,5
            ;apaga comunes del puerto E
            movlw 0xff
            movwf 9
            ;-------------------------------------------------------------
            ;inicia el FSR
            movlw 0x20
            movwf FSR
            ;chequea puerto de comunes (C ó E), en función de nlamp
            ;-------------------------------------------------------------
            ;nlamp >= 2?
            movf nlamp,0
            sublw 2
            btfsc 3,2
            goto menor          ;nlamp = 2
            btfss 3,0           ;pregunta por carry, es >< 2
            goto mayor_2

            ;carry = 1 ==> nlamp = < 2
menor       movf nlamp,0
            call cod_act
            andlw 7             ;máscara para eliminar puerto E
            movwf temp
            bcf 3,0             ;limpia carry antes de rotar
            rlf temp
            rlf temp
            rlf temp            ;rota izq

            ;trucaje para no afectar dato
            movf temp,0         ;leo cod act
            andwf 7,1           ;sumo para no afectar
```

```
            goto dato

            ;carry = 1 nlamp > 2
mayor_2     movf nlamp,0
            call cod_act
            clrf temp
            movwf temp          ;respalda valor
            bcf 3,0             ;limpia carry antes de rotar
            rrf temp
            rrf temp
            rrf temp
            movf temp,w         ;cod act al acum.
            movwf 9             ;cod. de act al puerto E

            ;sacar el dato para la lámpara nlamp
dato        movf nlamp,w
            addwf FSR,1
            movf INDF,0
            ;call cod_ss
            movwf 8

            ;prox lampara
            movf nlamp,w
            sublw 5
            btfss 3,2
            goto menor_5

            ;nlamp = 0
            clrf nlamp
            goto back_t0

            ;nlamp < 5
menor_5     incf nlamp

back_t0     bcf INTCON,2        ;limpia la bandera IT del timer0

back        movf fsr_temp,w
            movwf FSR           ;restituye FSR
            swapf stat_temp,w
            movwf STATUS        ;restituye STATUS
            swapf wtemp,f
            swapf wtemp,w       ;restituye W sin afectar las banderas del
                                ;STATUS.
            ;la instrucción "swapf   f ", no afecta el contenido del STATUS

            retfie              ;retorno de la subrutina.
;*****************************************************************************
;tabla de códigos de activación (ánodo común)
cod_act     addwf PCL,1         ;suma el offset al acum.
            retlw b'11111110'
            retlw b'11111101'
            retlw b'11111011'
            retlw b'11110111'
            retlw b'11101111'
            retlw b'11011111'
```

Técnicas de Programación para lectura de Teclados Lineales y Matriciales.

Teclados.

Existen dos modalidades para conectar pulsantes o interruptores a un micro controlador: conexión lineal y conexión matricial. La conexión lineal se prefiere cuando el número de teclas a conectar es pequeño y se dispone de la suficiente cantidad de líneas de puerto. Por otra parte la conexión matricial se prefiere cuando el número de teclas a conectar es numeroso (por lo general superior a 4) y no se dispone de un número suficiente de líneas de puerto.

Conexión Lineal.

En la **figura 4.12** se muestra un esquema de conexión lineal directa al puerto de un micro controlador suponiendo que este último posee "pull ups" internos tal como el puerto B en los micro controladores Microchip.

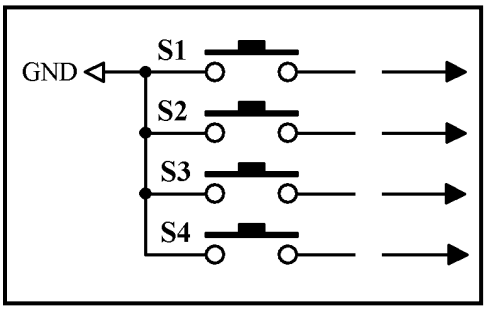


Figura 4.12. Conexión lineal de teclas considerando que el puerto de conexión posee "pull ups" internos.

Algoritmo para escanear teclados lineales.

El algoritmo para escanear un teclado lineal es relativamente simple y se basa en una encuesta secuencial del conjunto de teclas. En la **figura 4.13** se muestra un Diagrama de Flujo que ilustra el mencionado algoritmo.

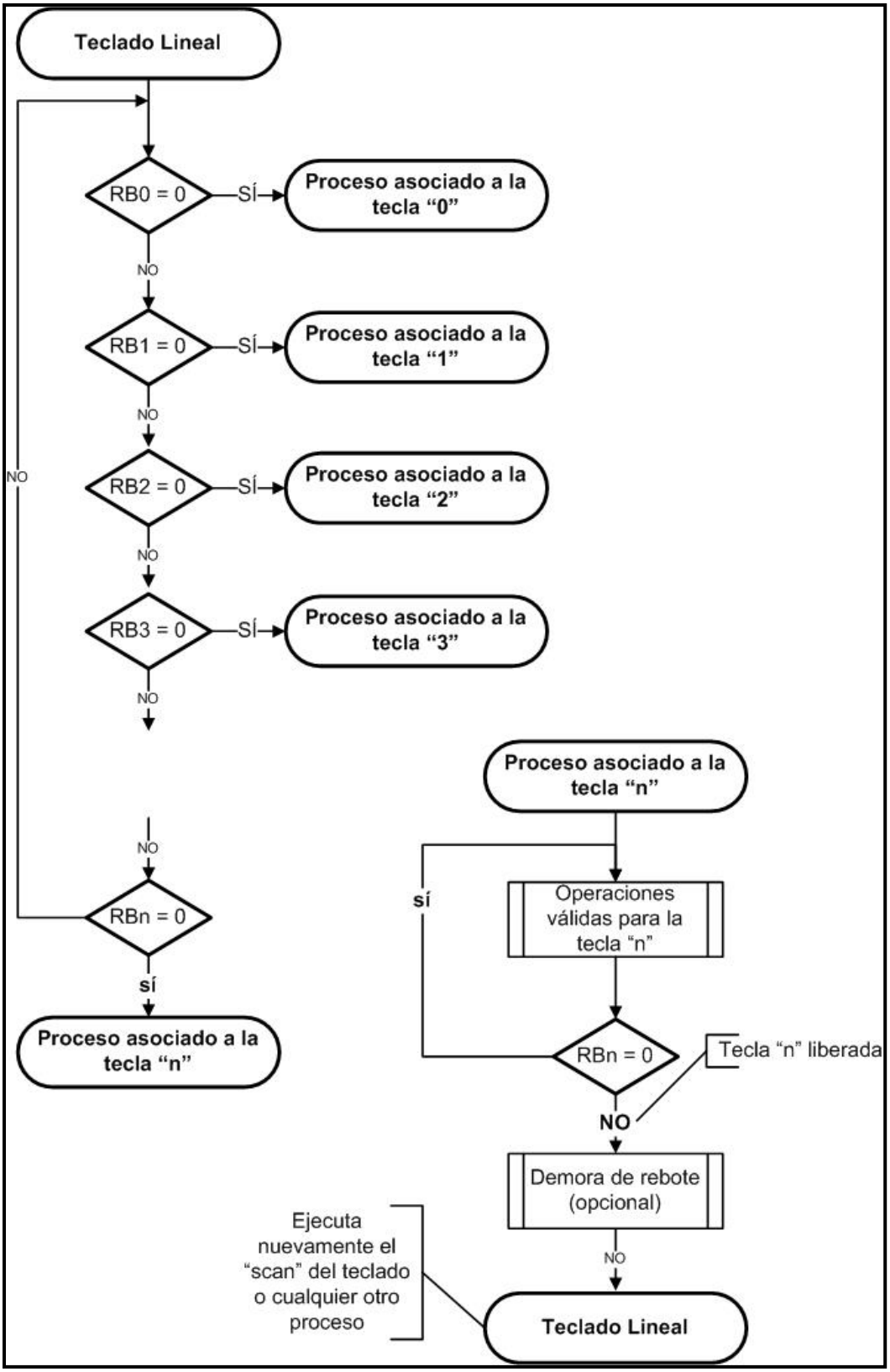


Figura 4.13. Diagrama de flujo para el escaneo de un teclado lineal

Conexión Matricial.

La conexión matricial se prefiere cuando el número de teclas es grande y por ende escasean las líneas de puerto del micro controlador para asumir una conexión lineal.

Para definir una conexión matricial se necesita definir un grupo de líneas del teclado como filas y otras como columnas con el objetivo de formar una matriz. Las filas serán pines de salida del micro controlador y las columnas entradas con “pull – up”. En la **figura 4.14** se muestra un esquema típico de un teclado matricial de 4 x 4 teclas (16 teclas) en el que se han definido cuatro filas y cuatro columnas. En caso de utilizar un puerto del micro controlador que posea “pull ups” internos se puede prescindir de los “pull ups” externos tal y como sucede con el puerto B de los micro controladores Microchip.

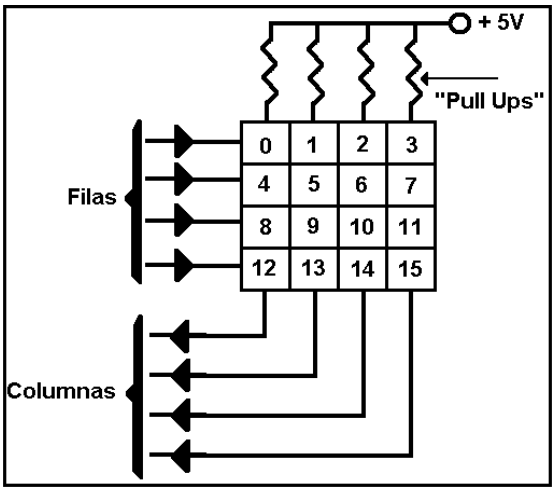


Figura 4.14. Conexión matricial de 16 teclas

Para un micro controlador un teclado matricial es simplemente un conjunto de teclas preferiblemente enumeradas desde cero en adelante independientemente de la codificación gráfica (símbolo) que muestra al usuario cada una de ellas. En la mayoría de los casos se necesita imprimir el caracter físico en un visualizador. La impresión física se realiza mediante un proceso denominado “decodificación del teclado”, que consiste en asociar en el visualizador el símbolo de la tecla al número que esta posee dentro del teclado. Generalmente para construir la decodificación se utiliza una tabla cuyo índice es el número de la tecla y los valores de salida son los símbolos físicos asociados a cada tecla. En la **figura 4.15** se ilustra el proceso de decodificación.

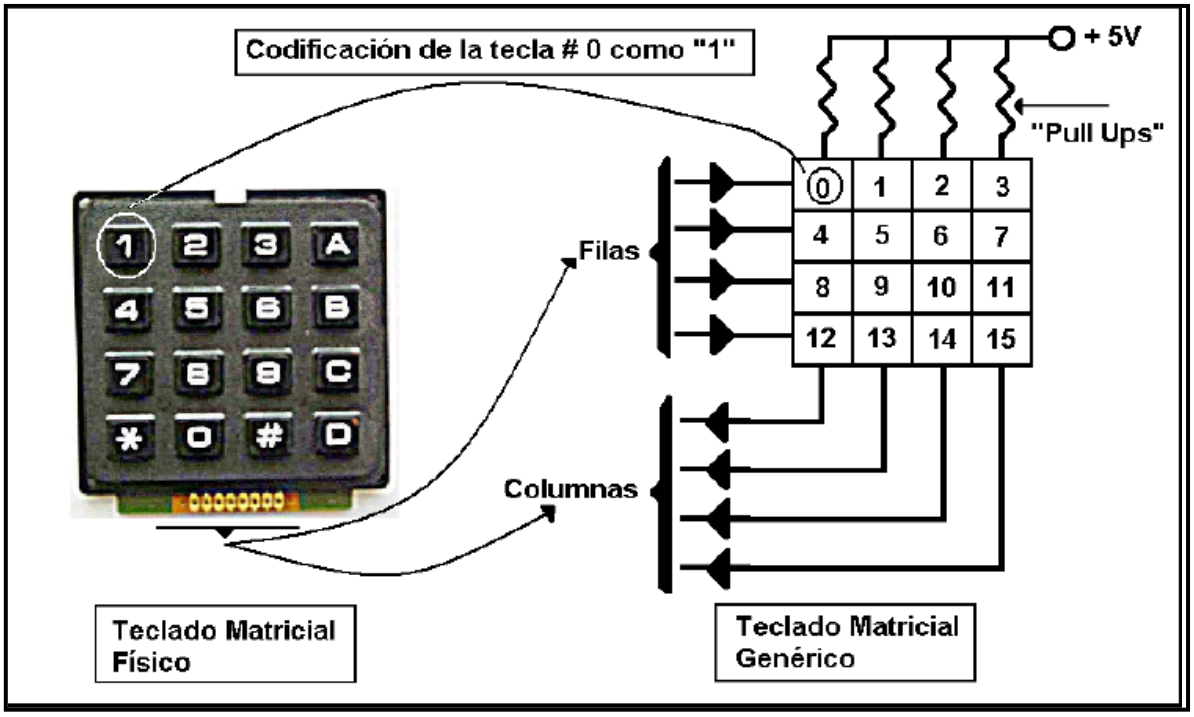


Figura 4.15. Decodificación de teclas.

Un ejemplo de tabla de decodificación para el teclado de la **figura 4.15** se muestra en el segmento de programa siguiente:

```
;tabla de decodificación a 7 segmentos para un teclado matricial de
;16 teclas muy popular en el mercado.
;Códigos para displays de Anodo Común.

cod_ss      addwf PCL,1
            retlw 0xf9              ;tecla # 0 => "1"
            retlw 0xa4              ;tecla # 1 => "2"
            retlw 0xb0              ;tecla # 2 => "3"
            retlw 0x88              ;tecla # 3 => "A"
            retlw 0x99              ;tecla # 4 => "4"
            retlw 0x92              ;tecla # 5 => "5"
            retlw 0x82              ;tecla # 6 => "6"
            retlw 0x83              ;tecla # 7 => "B"
            retlw 0xb8              ;tecla # 8 => "7"
            retlw 0x80              ;tecla # 9 => "8"
            retlw 0x98              ;tecla # 10 => "9"
            retlw 0xc6              ;tecla # 11 => "C"
            retlw 0x9c              ;tecla # 12 => "*"
            retlw 0xc0              ;tecla # 13 => "0"
            retlw 0xa3              ;tecla # 14 => "#"
            retlw 0xa1              ;tecla # 15 => "D"
            retlw 0xbf              ;tecla # 16=> "-" (no tecla)
```

Algoritmo para escanear teclados matriciales.

El algoritmo para escanear un teclado matricial de manera general se puede resumir en los siguientes pasos:

- Al inicio de la subrutina que realiza el escaneo se limpia un contador de las teclas que irán siendo escaneadas ("tecla" en el ejemplo de código) y una bandera ("flag" en el ejemplo de código) que servirá de aviso al programa principal en caso de que se detecte alguna tecla pulsada en la matriz. Se coloca además un patrón de bits con un cero en la primera fila a escanear y el resto en unos.
- Una vez listas las condiciones iniciales se procede a escanear las columnas incrementando "tecla" en caso de no detectar cero en la columna que fue escaneada.
- Al terminar de escanear las columnas se pregunta por el valor de "tecla" para verificar si ya se han escaneado todas las teclas del teclado. Si ya se han escaneado todas (tecla = 16d en este ejemplo), significa que no ha sido pulsada ninguna tecla lo que se indica con flag = 0. Por el contrario si se detecta que tecla < 16 d se procede a cambiar de posición el cero hacia la fila siguiente, lo que se consigue rotando el valor binario de las filas a través del acarreo (rrf PORTB, en el ejemplo de código). Para lograr ir ingresando valores de uno a las filas precedentes a través del acarreo es necesario ponerlo en uno siempre antes de rotar (bsf 3,0, en el ejemplo de código).
- Finalmente, si se detectara una tecla pulsada (fila = 0 y columna =0), la subrutina finaliza indicando que fue pulsada una tecla con flag = 0 y el número de la tecla pulsada queda almacenado automáticamente en el contador "tecla", cuyo valor será utilizado para decodificar la tecla.

Ejemplo de subrutina de escaneo matricial.

El siguiente segmento de programa muestra un ejemplo de subrutina para realizar el escaneo de un teclado matricial de 16 teclas (4 filas y 4 columnas):

Las columnas se conectan a los pines rb3, rb2, rb1 y rb0. Tales pines deberán ser programados como entradas con "pull-up" activo. Por otra parte, las filas se conectan a los pines rb7, rb6, rb5 y rb4. Tales pines deberán configurarse como salidas.

La subrutina devuelve el valor de la tecla (0 ... 15) en la variable **"tecla"** y a su vez indica si hubo tecla o no mediante el bit 0 de la bandera **"flag".**

```
;*****************************************************************
;subrutina de atención a teclado matricial**
;*****************************************************************
teclado     clrf tecla
            bcf flag,0        ;indica "no tecla"
            movlw b'01111111'
            movwf  6
            ;testeo de columnas
nextcol     btfss 6,3
            goto sitecla
            incf tecla
            btfss 6,2
            goto sitecla
            incf tecla
            btfss 6,1
            goto sitecla
            incf tecla
            btfss 6,0
            goto sitecla
            incf tecla
            movf tecla,w
            sublw 0x10
            btfss 3,2
            goto menor

            ;igual a 17d      (no hubo tecla), se indica con el bit 0 de
flag en 1
            bcf flag,0  ;indica "no tecla"
            return

menor       ;seguir escaneando, todavía no es la última tecla
            bsf 3,0           ;carry = 1, preparo para rotar
            rrf 6,1           ;rota puerto b a la derecha
            goto nextcol

sitecla     bsf flag,0        ;hubo tecla, y su valor está en "tecla"
            return
```

Para utilizar correctamente la subrutina de escaneo matricial debe tenerse en cuenta lo siguiente:

- Cada vez que se ejecute la subrutina se escanea todo el teclado.
- El valor de la tecla pulsada si la hubiere se devuelve en la variable contador "tecla", comenzando por tecla = 0. En caso de no haber sido pulsada ninguna tecla el contador "tecla" tomará el valor del número máximo de teclas del teclado más uno (por ejemplo "tecla" = 16 en un teclado de 4 x 4 teclas).

- Es necesario siempre después de ejecutar la subrutina, preguntar por el valor de “flag” para conocer si se pulsó una tecla o no. Según el valor de “flag” se procederá a leer el valor de “tecla” o no.

Técnicas de Programación para manejo de Visualizadores de Tinta Liquida (LCD).

Instrucciones para el manejo de LCDs.

La mayoría de los LCDs, admiten comandos que permiten realizar, entre otras las siguientes operaciones:

- Limpiar el display.
- Mover la posición del cursor al primer caracter de la primera línea.
- Establecer la dirección del movimiento del cursor (izquierda o derecha), elegir parpadeo para el cursor o cursor fijo, desplazamiento de un dato en el Display, etc..
- Apagar o encender el LCD.
- Direccionar la RAM de Display (DDRAM) o la RAM del Generador de Caracteres (CGRAM).
- Leer el estado del controlador del Display (BUSY flag) y la dirección actual del contador de direcciones de la RAM.
- Escribir un Dato a la DDRAM o a la CGRAM, según la Memoria que haya sido direccionada.

Los controladores de LCD son dispositivos lentos en comparación con la velocidad de operación de la mayoría de los MCUs, luego es necesario respetar las debidas demoras de ejecución para cada operación realizada sobre el LCD. Cada operación realizada sobre el LCD seguirá la siguiente secuencia (**figura 4.16**):

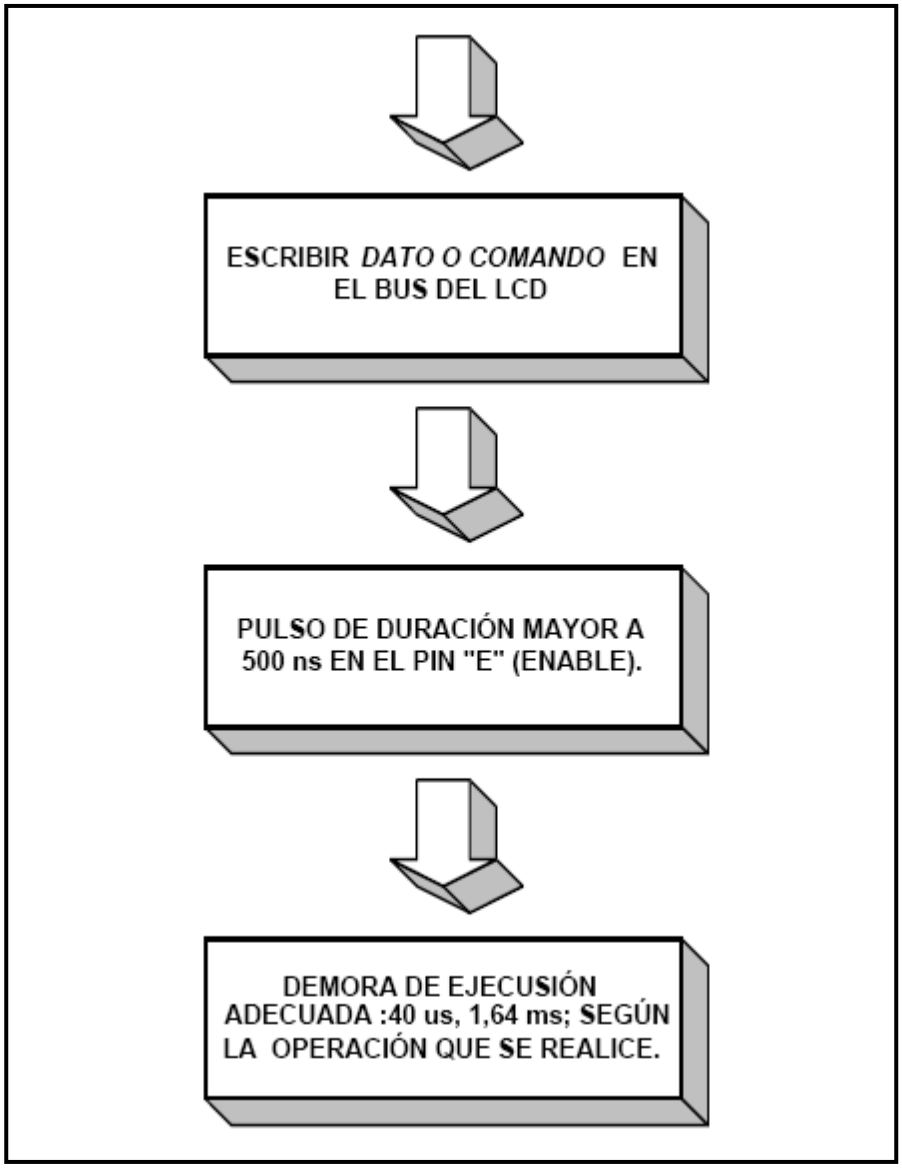


Figura 4.16. Secuencia básica de operación de un LCD.

Principales instrucciones para el manejo de LCDs.

En el ***anexo 1*** a este artículo se resumen las principales instrucciones necesarias para el manejo de los LCD de las series **L1672, L1682, y L1692 de "SEIKO INSTRUMENTS™".** Este Set de Instrucciones es válido para todos los LCDs que utilicen el chip controlador **KS0066**.

Iniciación del LCD para el trabajo.

Toda aplicación que incluya el trabajo con LCDs debe prever un proceso de iniciación del LCD, previo a la realización de escrituras o lecturas en el mismo. El proceso de iniciación de un LCD incluye las siguientes acciones:

- Establecer el modo "comando" (RS =0) .
- Establecer escritura (R/*W = 0).
- Realizar la **"secuencia de Reset**" del LCD.
- Definir el tipo de interfaz (4 u 8 bits), el # de líneas del LCD y el área de la matriz de los caracteres; mediante una instrucción **"Function Set"** (anexo 1).
- Definir la dirección del movimiento del cursor y especificar si se va a desplazar o no el dato que se escriba en el LCD, mediante una instrucción **"Entry Mode Set"** (anexo 1).
- Encender el LCD (en la condición de OFF la memoria no se borra), habilitar o no el cursor (el cursor puede apagarse) y la modalidad del cursor (fijo o parpadeante); mediante una instrucción **"Display ON/OFF Control"** (anexo 1).
- Dar un clear al LCD (opcional), mediante la instrucción **"Clear Display"** (anexo 1).

Después de realizar estos procesos, es necesario antes de escribir o leer definir la dirección inicial (refiérase al epígrafe II.3.3.1).

La siguiente subrutina escrita en ensamblador Microchip™ ilustra la iniciación de un LCD de 2x16 líneas para el trabajo.

```
;**************************************************
;SUBRUTINA DE INICIACIÓN DEL LCD*
;**************************************************
;inicia el LCD para el trabajo en modo de transferencia de datos de 4
bits
;el bus de datos del LCD se maneja con el puerto "D" de un MCU de 40
pines.
;las líneas de control RS, R/*W y E, se conectan a los pines Ra0, Ra1 y
Ra2
;del puerto "A" respectivamente.
inicia      bcf 5,2 ;E = 0 (LCD inhabilitado E = 0)
            bcf 5,0 ;RS = 0 (Modo de Instrucciones)
            bcf 5,1 ;R/*W = 0 (habilitación de escritura al LCD)

;INICIO DE LA SECUENCIA DE RESET DEL CRISTAL
            call dem_15ms ;demora de power UP
            movlw 0x30
            movwf 8
            call pulse ;enable
            call dem_5ms
            movlw 0x30
```

```
            movwf 8
            call pulse ;enable
            call dem_100us
            movlw 0x30
            movwf 8
            call pulse ;enable
            movlw 0x30
            movwf 8
            call pulse
            movlw 0x20
            movwf 8
            call pulse ;enable
            call dem_40us
;FIN DE SECUENCIA DE RESET DEL CRISTAL
  ;*************************

;FUNCTION SET INST
;*************************
;Modo de transferencia de Datos: 4 bits
;Display en 2 líneas
;Matriz de 5x7 puntos
            movlw b'00100000' ;0100
            movwf 8
            call pulse ;pulso de 1us en "E"
            movlw b'10000000' ;1000
            movwf 8
            call pulse
            call dem_40us ;demora de 40 us
;****************************
;ENTRY MODE SET INST
;****************************
;La posición del cursor se incrementa (direcciones de la
;DD RAM)
;No desplazar el Dato
            movlw b'00000000' ;0000
            movwf 8
            call pulse ;pulso de 1us en "E"
            movlw b'01100000' ;0110
            movwf 8
            call pulsepulso de 1us en "E"
            ; call dem_40us
;****************************
;DISPLAY ON/OFF CONTROL
;****************************
;display ON
;cursor ON
;cursor parpadea
            movlw b'00000000' ;0000
            movwf 8
            call pulse ;pulso de 1us en "E"
            movlw b'11100000' ;1110
            movwf 8
            call pulse ;pulso de 1us en "E"
            call dem_40us
```

```
;****************************
;DISPLAY CLEAR
;****************************
;limpia la RAM de display y pone el cursor en la posición cero
            movlw b'00000000' ;0000
            movwf 8
            call pulse ;pulso de 1us en "E"
            movlw b'00010000' ;1110
            movwf 8
            call pulse ;pulso de 1us en "E"
            call dem_1640us
;fin del proceso
; esta subrutina NO posiciona el cursor en la dirección 80h
            Return
```

Comentarios sobre esta subrutina: observe que para el manejo de LCDs es necesario utilizar varias demoras. Es aconsejable diseñar una única demora que permita generar los diferentes tiempos que se necesitan, modificando algún parámetro de entrada a la subrutina, para de esta manera ahorrar líneas de código.

La subrutina “pulse” simplemente conmuta la línea de habilitación “E”:

```
;*******************************************************
;SUBRUTINA QUE GENERA UN PULSO EN "E"*
;*******************************************************
; fosc = 16 Mhz.  Ciclo de Máquina = 0,25 µs
            pulse bsf 5,E ;E = 1
            nop ;0,25 µs
            nop ;0,25 µs
            nop ;0,25 µs
            nop ;0,25 µs
            bcf 5,E ;E = 0

            return
```

Lecturas y escrituras del LCD.

Escribiendo directamente la Ram de Display.

- Antes de escribir un dato, es necesario direccionar previamente la localización de la RAM que se quiere escribir. La mayoría de los LCDs soportan el modo incremental. Este modo, garantiza que después de una escritura o lectura de la Ram de Display (DDRAM) o la Ram del Generador de Caracteres (CGRAM) se incrementa o decrementa automáticamente la dirección de escritura o lectura. Solo es necesario tener en cuenta la dirección inicial: por ejemplo, si se desea comenzar a escribir un cartel en la segunda fila, desde la extrema izquierda; es

necesario establecer la dirección inicial C0h (primer caracter de la 2da fila). Para direccionar la DDRAM es necesario poner RS = 0, R/*W = 0, DB7 = 0 y poner la dirección en los bits D6:D0. si el interfaz es en 4 bits primero se envían los 4 bits más significativos (D7:D4) y después los 4 menos significativos.

- Una vez direccionada la RAM, los caracteres a escribir pueden enviarse directamente en ASCII. El controlador es capaz de interpretar directamente códigos ASCII.

- Si Ud. necesita algún caracter especial no codificado, entonces el procedimiento será "crear" el carácter programando el caracter en la RAM del Generador de Caracteres.

El siguiente fragmento de código en ensamblador Microchip™ ilustra el proceso de direccionado, específicamente el cambio de fila y la escritura del caracter 'A':

```
;*************************************************************
;Ra0 = "RS"
;índice = 14(después de 14 caracteres es necesario cambiar de fila)
;cambio de fila
;modo de transferencia en 4 bits
bcf 5,0                 ;RS = 0 ( modo comandos)
movlw 0xc0              ;parte alta de la primera dirección de la
                        ;segunda  fila "C"
movwf 8
call pulse              ;pulso de habilitación de 1µS en "E"
call dem_40us           ;demora de 40 us, tiempo que demora la
                        ;ejecución de esta inst.
movlw 0                 ;parte baja de la primera dirección de la
                        ;segunda fila "0"
movwf 8
call pulse
call dem_40us           ;demora de 40 us, tiempo que demora la
                        ;ejecución de esta inst.
bsf 5,0 ;RS = 1 ( modo datos )
                        ;escribe caracter
movlw 'A'               ;caracter a escribir.
movwf 8
call pulse              ;pulso de habilitación de 1µS en "E"
call dem_40us           ;demora de 40 us, tiempo que demora la
                        ;ejecución de esta inst.
```

Leyendo la posición del cursor.

La lectura de la posición del cursor en determinado momento es muy útil para realizar menús interactivos. Esto es, se le presenta al usuario un conjunto de caracteres de los cuales el deberá seleccionar un conjunto. Se permite incrementar la posición del cursor, mediante teclas de selección “UP”, “DOWN” y validar la selección mediante una tecla “ENTER”. En el momento de la validación se lee la posición del cursor, para asociar la posición con la acción deseada. Para mover el cursor dentro del Display sin modificar el contenida de la DD RAM utilice la instrucción “Cursor or Display Shift” (anexo 1). Para leer la posición del cursor, utilice la instrucción “Read Busy Flag and Address” (anexo 1).

Escribiendo caracteres “custom”. Programando la CGRAM.

La RAM del Generador de Caracteres (CGRAM) permite al usuario crear hasta 8 caracteres “custom” de 5x8 puntos. Una vez que estos caracteres se programen pueden ser accesados como si estuvieran en la ROM del controlador (la ROM almacena los caracteres incorporados), la única dificultad es que la CGRAM es volátil y cuando se desconecta la alimentación se borra la información grabada.

Los módulos de RAM de un Cristal se dividen en dos partes: la RAM de Datos del Display (DDRAM) y la RAM del Generador de Caracteres (CGRAM); ambas zonas de memoria son independientes y por ende ocupan espacios de direcciones diferentes dentro de la memoria del controlador. La DDRAM ocupa las direcciones 80h al 8Fh para la primera fila de caracteres y las direcciones C0h al CFh para la segunda fila de caracteres. La CGRAM comienza en la dirección 40h hasta la dirección 7Fh. Cada caracter dentro de la CGRAM ocupa 8 direcciones; así el primer carácter “custom” ocupará desde la 40h hasta la 47h, el segundo desde la 48h hasta la 4Fh y así sucesivamente hasta agotar las 64 localizaciones de esta memoria. Si durante la iniciación del LCD Ud. lo programó para que ocurriera un incremento automático del contador de direcciones, solo será necesario enviar la dirección inicial de la CGRAM (40h), mediante la instrucción “Set CG RAM Address” (**figura 4.17**).

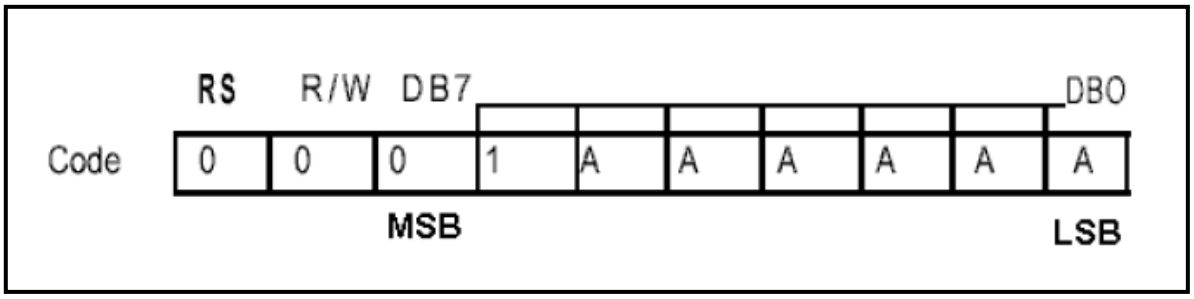


Fig. 4.17. Instrucción para el direccionado de la CG RAM. En la figura “A” significa el bit de la dirección.

Después de direccionar la localización inicial solo será necesario realizar 8 escrituras consecutivas por cada carácter. Si Ud. desea programar los 8 caracteres necesitará de 64 escrituras (trabajando en la modalidad de 8 bits). Cada localización de la CGRAM tiene una extensión de 8 bits, aunque para el carácter solo se utilizan 5 (5 puntos). En el dato que se envié a la CGRAM un “1” significa que el píxel estará en ON y un “0” en OFF (**figura 4.18**).

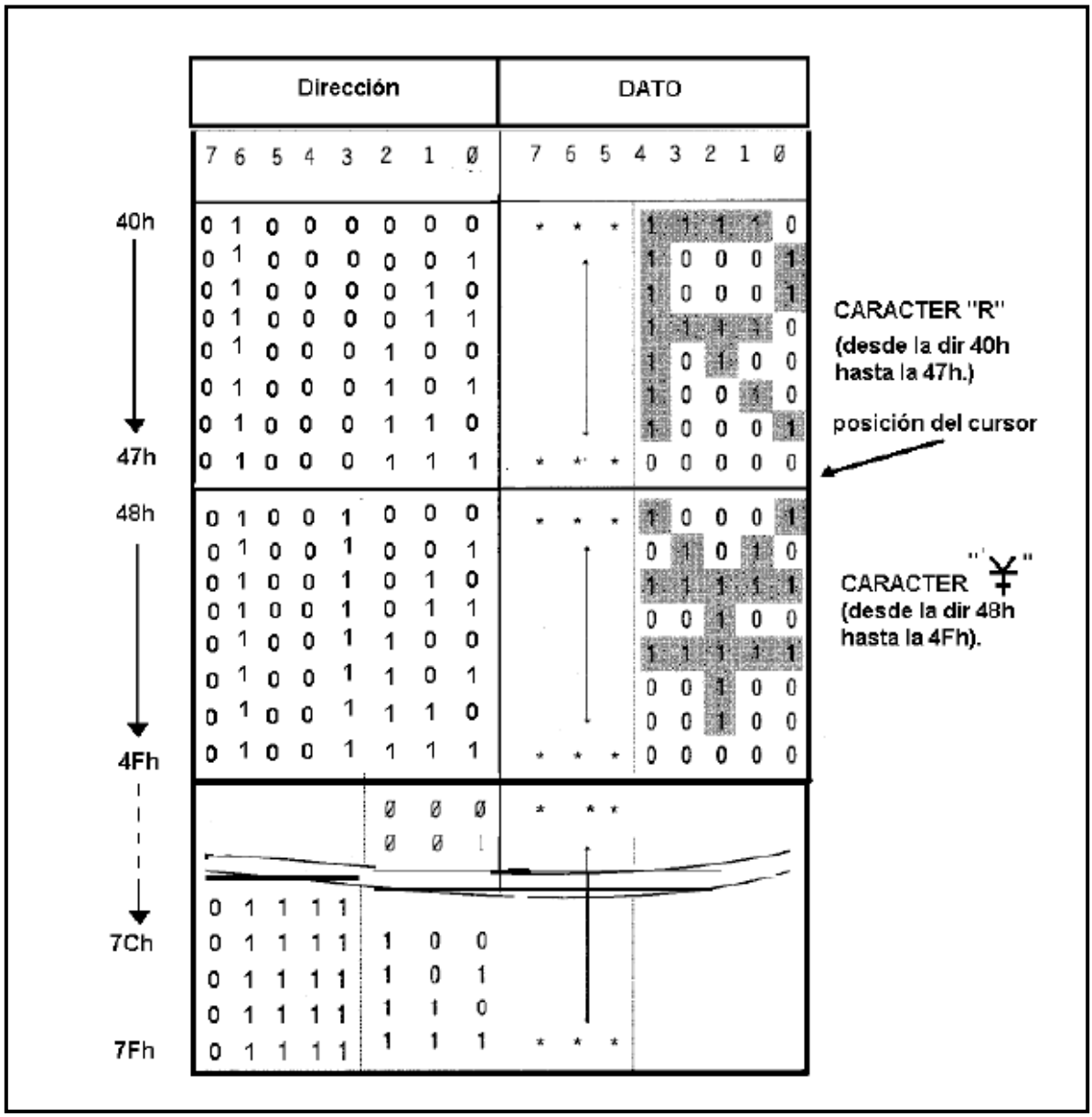


Fig. 4.18. Representación de caracteres “custom” en la CGRAM.

Ejemplo de técnica de acceso a datos a visualizar en LCD.

Introducción.

Esta segunda parte completa un conjunto de técnicas útiles para el tratamiento rápido de pantallas líquidas matriciales.

La escritura de varios carteles en pantallas líquidas **pudiera** ser tratada de la siguiente forma: todos los carteles se escriben en una única tabla. La técnica general consiste en indexar la tabla y extraer los caracteres que correspondan al cartel que en ese momento se desea visualizar. El siguiente ejemplo propone la escritura de un cartel siguiendo los pasos siguientes:

1. **Cambiar a la dirección de escritura deseada dentro del LCD**[1], esta función la realiza la subrutina **"chdir"**:

```
;dirección inicial de escritura: 0x83
movlw 0x83 call chdir ;del primer cartel.
```

2. **Indexar la tabla utilizando una variable auxiliar (en este caso *index*)**. "indexar" significa cargar el índice inicial, que dentro de la tabla tendrá la primera letra del cartel (en este caso *index = 0*)

```
clrf index ;apunta al primer cartel (index = 0)
```

3. **Definir el tamaño que tendrá el cartel** utilizando una variable auxiliar (en este caso "*size*"). La búsqueda de caracteres se realizará desde el valor inicial de *index* hasta *index = index + size*.

```
movlw 8
movwf size ;define tamaño del cartel
```

4. **Finalmente escribir el cartel** tomando en cuenta los valores de *index* y *size*. Esta función la realiza la subrutina **"write"**:

```
write clrf ch_cont            ;contador de caracteres del
;cartel que se está sacando
next_char movf index,w        ;indexar
call tabla                    ;tabla de caracteres del cartel
movwf 8                       ;saco caracter
call pulse                    ;pulso enable
call dem_40us                 ;demora de ejecución
movf ch_cont,w
subwf size,w                  ;resta ch_cont del size
btfsc 3,2
return                        ;RETORNA, se escribió completo
                              ;próximo caracter del cartel
incf index
incf ch_cont
goto next_char
```

Note que la subrutina write utiliza una variable auxiliar de conteo “ch_cont” , una subrutina para habilitar el LCD (“pulse”) y otra donde se guardan los caracteres de los carteles (“tabla”).

La estructura general del código del ejemplo sería la siguiente:

En el programa principal:

```
                          ;poner CARTEL 1 LÍNEA DE ARRIBA

movlw 0x83                ;dirección inicial de escritura: 0x83
call chdir                ;del primer cartel.
clrf index                ;apunta al primer cartel (index = 0)
movlw 8
                          ;define tamaño del cartel
movwf size call write     ;Escribe el cartel
                          ;poner CARTEL 1 LÍNEA DE ABAJO
movlw 0xc1
                          ;dirección inicial de escritura: 0XC1
movwf dir call chdir
movlw 9
movwf index
movlw d'12'               ;define tamaño de cartel
movwf
size call write           ;Escribe cartel
call dem_1s               ;demora de exhibición cartel1
call dem_1s
;-----------------------------------------------------------------------
;SPOT INICIAL DE LA APLICACIÓN (CARTEL 2)
;-----------------------------------------------------------------------
```

Y así continúa la exhibición de carteles.

Bloque de subrutinas:

```
write        clrf ch_cont              ;contador de caracteres del
                                       ;cartel que se está sacando
next_        char movf index,w         ;indexo
             call tabla                ;tabla de caracteres del cartel
             movwf 8                   ;saco caracter
             call pulse                ;pulso enable
             call dem_40us             ;demora de ejecución
             movf ch_cont,w
             subwf size,w              ;resta ch_cont del size
             btfsc 3,2
             return                    ;RETORNA, se escribió completo
                                       ;próximo caracter del cartel
             incf index
             incf ch_cont
             goto next_char
```

Para la tabla de caracteres se sugiere la siguiente estructura:

```
;==========================================================================

;TABLA DE CARACTERES ****************************************************
;==========================================================================
tabla addwf PCL,1
             ;================= index = 0
             ;================= size = 8 ESPOL
             retlw 'E'
             retlw '.'
             retlw 'S'
             retlw '.'
             retlw 'P'
             retlw '.'
             retlw 'O'
             retlw '.'
             retlw 'L'
             ;================= index = 9
             ;================= size = 12 Ing. Eléctrica.
             retlw 'I'
             retlw 'n'
             retlw 'g'
             retlw '.'
             retlw 'E'
             retlw 'l'
             retlw 'e'
             retlw 'c'
             retlw 't'
             retlw 'r'
             retlw 'i'
             retlw 'c'
             retlw 'a'
             ;================= index = 22
             ;================= size = 16 Ctrl. de Velocidad.
             retlw 'C'
             retlw 't'
             retlw 'r'
             retlw 'l'
             retlw '.'
             retlw 'd'
             retlw 'e'
             retlw ' '
             retlw 'V'
             retlw 'e'
             retlw 'l'
             retlw ' '
             retlw 'C'
             retlw '.'
             retlw 'C'
             retlw ' '
             retlw ' '
```

Diseño de aplicaciones utilizando temporización por hardware e interrupciones. Diseño de contadores utilizando Visualizadores de Leds de 7 segmentos y Visualizadores de Tinta Liquida (LCD). Diseño de aplicaciones utilizando Teclados Matriciales. Técnicas de decodificación aplicadas al diseño de Sistemas de Seguridad.

Diseño de Contadores.

Los contadores BCD, binarios, "de anillo" (tipo Jhonson) o arbitrarios se aplican extensamente dentro del área de proyectos. En sí mismo un contador puede ser:

- Un contador de eventos u objetos.
- Un contador de opciones de un menú.
- Un contador de parámetros en Paneles Frontales de Equipos Electrónicos.
- Un cronómetro o reloj.
- Un frecuencímetro.

Entre otras aplicaciones.

Según su complejidad un contador puede ser ascendente, ascendente / descendente o pre cargable (se carga la base inicial de conteo).

Este apartado pretende ilustrar el algoritmo de conteo ascendente BCD para que sea utilizado como base para el diseño de contadores más complejos.

La siguiente subrutina utiliza como elementos de conteo la propia RAM de display definida por las variables L1 … L6. En este caso cada localización de la RAM de Display contendrá un número BCD (0 … 9), que llevará el conteo individual por dígito de un número de seis. El caso discutido a continuación se refiere entonces como un contador BCD ascendente de seis dígitos o sea con una base máxima de conteo de 999 999.

Subrutina de Conteo Ascendente.

El siguiente fragmento de código ilustra una subrutina de conteo BCD ascendente de 6 dígitos.

```
;*****************************************************************
subrutina de conteo ascendente (6 dígitos)

cuenta_up    incf l6
             movf l6,w
             sublw d'10'
             btfss 3,2
             return       ;no incrementar dígito adyacente
             ;llego a 10
             clrf l6
             ;-------------------
             ;procesa l5
             incf l5
             movf l5,w
             sublw d'10'
             btfss 3,2
             return       ;no incrementar dígito adyacente
             ;llego a 10
             clrf l5
             ;-------------------
             ;procesa l4
             incf l4
             movf l4,w
             sublw d'10'
             btfss 3,2
             return       ;no incrementar dígito adyacente
             ;llego a 10
             clrf l4
             ;-------------------
             ;procesa l3
             incf l3
             movf l3,w
             sublw d'10'
             btfss 3,2
             return       ;no incrementar dígito adyacente
             ;llego a 10
             clrf l3
             ;-------------------
             ;procesa l2
             incf l2
             movf l2,w
             sublw d'10'
             btfss 3,2
             return       ;no incrementar dígito adyacente
             ;llego a 10
             clrf l2
             ;-------------------
             ;procesa l1
             incf l1
             movf l1,w
             sublw d'10'
             btfss 3,2
             return       ;no incrementar dígito adyacente
             ;llego a 10
             clrf l1
             return
```

La subrutina anterior se basa en un **algoritmo** muy sencillo **que es aplicado a cada dígito de conteo (Lx)** y que se puede resumir como sigue:

1. Se incrementa Lx (donde x es el número del dígito representado en la RAM).
2. Se chequea si el dígito Lx se desbordó (sobrepasó el valor de 9).
3. Si el dígito Lx no sobrepasa 9 significa que el resto de los dígitos deben mantener su valor anterior por lo que se retorna al Programa Principal.
4. Si el dígito Lx sobrepasa 9 significa que debe llevarse a cero y continuar desde el paso 1 para el siguiente dígito (L_{x+1}) hasta que se encuentre un dígito que no sobrepase el 9 para retornar al Programa Principal.

Utilización de la subrutina.

La subrutina de conteo se utilizará cada vez que exista la necesidad de conteo. Es necesario aclarar que esta subrutina sólo actualiza el valor de conteo en la RAM, no en el Display. Dependiendo del tipo de visualizador que se utilice será necesario otro elemento de software para visualizar la información del conteo. Por ejemplo si el conteo se visualiza en Displays de 7 segmentos será necesario emplear una Técnica de Refrescamiento como la expuesta en el apartado correspondiente.

Ejemplos de algunas subrutinas aplicadas al manejo de LCD.

Introducción.

Las subrutinas que a continuación se ejemplifican sirven de base para el diseño de aplicaciones basadas en LCD. Para su empleo deben tenerse en cuenta los siguientes aspectos:

Pudieran existir ligeras variaciones para distintos tipos de LCD. Estas subrutinas están basadas en uno de los Controladores de LCD más populares.

Iniciación del LCD. Esta subrutina debe ser ejecutada **una sola vez** al inicio de cualquier aplicación, que emplee un LCD. En ella se definen determinados parámetros que regulan el funcionamiento del LCD, tales como:

- **Reset al controlador del LCD:** enviando un 38 hex, en modo comando. La demora de “power up”, es necesaria para garantizar que la fuente de la tarjeta se estabilice. Generalmente el Reset se ejecuta una sola vez, en la subrutina de iniciación.

- **Seteo de Funciones ("Function Set").** Esta instrucción permite establecer, el modo de tranferencia de datos (4 u 8 bits), el número de líneas de visualización (1, 2, ó 4) y el tamaño de la matriz de puntos (generalmente 5x7). La octava fila de la matriz se reserva para el cursor. Generalmente el Seteo de Funciones se ejecuta una sola vez, en la subrutina de iniciación.
- **Control del encendido ("Display ON/OFF Control").** Permite apagar el LCD y/o el cursor. Debe aclararse que el apagado no significa que la información enviada al controlador se pierda. El Control de Encendido puede ejecutarse si se desea en cualquier momento durante la ejecución de la aplicación; para ahorrar energía.
- **Modo de Trabajo ("Entry Mode Set").** Permite definir si el cursor del LCD se desplazará durante la escritura (lo más común es que se desplace) y permite desplazar o no el dato que se vaya escribiendo. El Modo de Trabajo puede cambiarse en cualquier momento de la ejecución si se necesita.
- **Limpiar Display ("Display Clear").** Permite limpiar la pantalla. Al ejecutar un "Clear Display", se pierde la información en pantalla y el cursor se posiciona automáticamente en la dirección 80 hex (esquina superior izquierda de la pantalla). El Display puede limpiarse en cualquier momento de la ejecución si se necesita.

```
;=========================================================================
;SUBRUTINA DE INICIACIÓN DEL LCD (Modo 8 Bits).
;=========================================================================
ini_lcd     bcf 9,e           ;E = 0 (LCD ihabilitado E = 0)
            bcf 9,rs          ;RS = 0 (Modo de Instrucciones)
            bcf 9,r_w         ;R/*W = 0 (habilitación de escritura al LCD)
            ;INICIO DE LA SECUENCIA DE RESET DEL CRISTAL
            call dem_15ms     ;demora de power UP
            movlw 0x38

            movwf 8
            call pulse        ;enable
            call dem_40us
            ;FIN DE SECUENCIA DE RESET DEL CRISTAL
            ;*******************
            ;FUNCTION SET INST
            ;*******************
            ;Modo de transferencia de Datos: 8 bits
            ;Display en 2 líneas
            ;Matriz de 5x7 puntos
            movlw 0x38        ;funtion set = 38h
            movwf 8
            call pulse        ;pulso de 1us en "E"
            call dem_40us     ;demora de 40 us
            ;****************************
            ;DISPLAY ON/OFF CONTROL
            ;****************************
            ;display ON
```

```
;cursor ON
;cursor parpadea
movlw 0x0e        ;display ON/OFF control = 0ch
movwf 8
call pulse        ;pulso de 1us en "E"
call dem_40us
;****************************
;ENTRY MODE SET INST
;****************************
;La posición del cursor se incrementa (direcciones de la DD
RAM)
;No desplazar el Dato
movlw 6           ;entry mode set = 6
movwf 8
call pulse        ;pulso de 1us en "E"
call dem_40us
;****************************
;DISPLAY CLEAR
;****************************
;limpia la RAM de display y pone el cursor en la posición
cero
movlw 1           ;display clear = 1
movwf 8
call pulse        ;pulso de 1us en "E"
call dem_1640us
;****************************
;RAM A LA 80H
;****************************
;limpia la RAM de display y pone el cursor en la posición
cero
movlw 0x80        ;display clear = 1
movwf 8
call pulse        ;pulso de 1us en "E"
call dem_40us
bsf 9,rs          ;modo datos

return
```

Habilitación del LCD. El proceso de habilitación consiste de un pulso positivo en el pin "enable" del LCD de duración **mayor a 250 ns**.

```
;=========================================================================

;SUBRUTINA DE HABILITACIÓN DEL LCD (Pulso en Enable).
;=========================================================================
pulse       bcf 9,e
      bsf 9,e     ;E = 1
      nop
      nop
      nop
      nop         ;demora de 1 us
      bcf 9,e     ;E = 0
      return
```

Demoras. Típicamente para el manejo de LCDs se necesita diseñar varias demoras, por lo que se sugiere diseñar una demora general que funcione a partir de ciertos parámetros de entrada (en este caso las variables “M” y “N”). Algunos de los tiempos necesarios son los siguientes:

- Demora de “Power ON” (5 a 15 ms).
- Demora de Instrucción o Dato (mayor a 40µS).
- Demora de “Clear” (mayor a 1.64 ms).

```
;==========================================================================
;Subrutinas de DEMORA (Valores de N y M calculados para Fosc = 16.000
MHz.)
;==========================================================================
;-------------------------------------------------
;DEMORA DE 5.32 ms
;-------------------------------------------------
dem_5ms     movlw d'37'
            movwf N           ;parámetro a cargar en contadores
            movlw d'5'
            movwf M           ;parámetro a cargar en contadores
            call demora       ;demora de 5 ms.
            return
;-------------------------------------------------
;DEMORA DE 1.64MS (1640 US)
;-------------------------------------------------
dem_1640us  movlw d'46'
            movwf N           ;parámetro a cargar en contadores
            movlw d'1'
            movwf M           ;parámetro a cargar en contadores
            call demora       ;demora de 1640 US
            return
;-------------------------------------------------
;DEMORA DE 47US*
;-------------------------------------------------
dem_40us    movlw d'6'
            movwf N           ;parámetro a cargar en contadores
            movlw d'1'
            movwf M           ;parámetro a cargar en contadores
            call demora       ;demora de 47us
            return
;-------------------------------------------------
;DEMORA DE 100US*

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

dem_100us   movlw d'6'
            movwf N           ;parámetro a cargar en contadores
            movlw d'2'
            movwf M           ;parámetro a cargar en contadores
            call demora       ;demora de 100us
            return
```

```
;-----------------------------------------------
;DEMORA PARAMÉTRICA DE TIEMPOS (GENERAL)
;-----------------------------------------------
Demora      movf N,w
            movwf cont1
            movwf cont2
            movf M,w
            movwf cont3
            loop decfsz cont1
            goto loop
            movf N,w
            movwf cont1
            decfsz cont2
            goto loop
            movf N,w
            movwf cont2
            decfsz cont3
            goto loop
            ;fin del proceso, inicia contadores
            return
;=====================================================================
;CHDIR; SUBRUTINA QUE ESTABLECE UNA DIRECCIÓN INICIAL DE ESCRITURA
;=====================================================================
chdir       bcf 9,rs    ;modo comandos
            movwf 8     ;la dirección viene en el acumulador
            call pulse
            call dem_40us
            bsf 9,rs
            return
;=====================================================================
;CURSOR; ENCIENDE EL CURSOR DEL LCD.
;=====================================================================
cur_on      bcf 9,rs    ;modo comando
            bsf 5,3     ;cur_on
            movlw 0x0f  ;encemder cursor, cursor parapadea
            movwf 8
            call pulse
            call dem_40us
            bsf 9,rs
            return
;=====================================================================
;CUR_OFF; APAGA EL CURSOR DEL LCD.
;=====================================================================
cur_off     bcf 9,rs    ;modo comando
            bcf 5,3     ;cur off
            movlw 0x0c  ;apaga cursor, cursor parapadea
            movwf 8
            call pulse
            call dem_40us
            bsf 9,rs
            return
;=====================================================================
;WRITE; SUBRUTINA QUE ESCRIBE CUALQUIER CARTEL
;=====================================================================
write clrf  ch_cont           ;contador de caracteres del

                              ;cartel que se está sacando
```

```
next_char   movf index,w      ;indexo
            call tabla        ;tabla de caracteres del cartel
            movwf 8           ;saco caracter
            call pulse        ;pulso enable
            call dem_40us     ;demora de ejecución
            movf ch_cont,w
            subwf size,w      ;resta ch_cont del size
                              ;tamaño del cartel).
            btfsc 3,2
            return            ;RETORNA, se escribió completo
                              ;próximo caracter del cartel
            incf index
            incf ch_cont
            goto next_char
;=====================================================================
;CLRDISP; LIMPIA LA PANTALLA LÍQUIDA
;=====================================================================
clrdisp     bcf 9,rs
            movlw 1           ;display clear = 1
            movwf 8
            call pulse        ;pulso de 1us en "E"
            call dem_1640us
            bsf 9,rs          ;modo datos
            return
```

Modulo de EEPROM de datos y Tratamiento de Memoria Flash. Estructura, Registros de Control y Programación de Aplicaciones.

Introducción.

La EEPROM de Datos es una Memoria Permanente (no volátil) de 64 hasta 128 Bytes en la Serie 87X.

Su principal aplicación es la de almacenar información no volátil de propósito general.

Estructura de la memoria.

En la **figura 4.19** se muestra un esquema estructural de la Memoria EEPROM de Datos. Consiste en un arreglo de registros que puede direccionarse a través de un registro de control desde el programa (EEADR); con el objetivo de realizar operaciones de lectura y/o escritura.

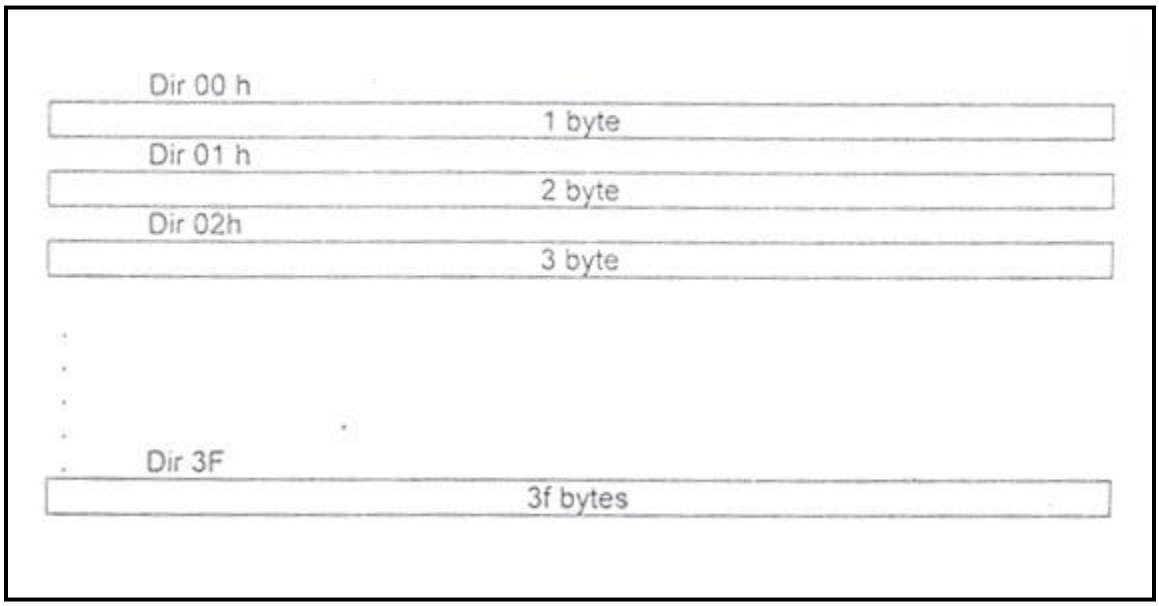


Figura 4.19. Esquema estructural de la Memoria EEPROM de Datos.

Operaciones sobre la EEPROM de Datos

Registros de Control. Funciones.

- **EEDATA:** Aquí se carga el dato que se desea grabar o se recoge el dato leído.
  Ejemplo: Grabacion de la variable "centH".

```
movf centH,w                ;carga la variable "centH" al acumulador
movwf dato_w                ;mueve "W" a la var "dato_w"
call w_eeprom               ;subrutina que escribe en la EEPROM el
                            ;el dato que entra en "dato_w"
```

  Ejemplo: Lectura del valor de la variable "dol_c" que está almacenado en la EEPROM.

```
;lee centenas de dólares ("dol_c")
movlw 0x20        ;el dato a leer esta almacenado en la dir 0x20
movwf dir         ;variable auxiliar "dir" que utiliza la
                  ;subrutina "r_eeprom"
call r_eeprom     ;subrutina que lee la dirección "dir" de le EEPROM
movf dato_r,w     ;se obtiene el dato leído desde la var "dato_r"
movwf dol_c       ;mueve el dato leído a la variable de destino en
                  ;RAM ("dol_c")
```

- **EEADR:** En este registro se carga la dirección a la cual se desea grabar el dato. Por lo general la dir se carga en una variable previa antes de llamar a la subrutina que escribe cualquier dato (en este caso "w_eeprom").

- **EECON1:** Registro que controla los procesos sobre la EEPROM. En la **figura 4.20** se muestra un resumen de las funciones de los bits que contiene la EECON1.

REGISTER 4-1: EECON1 REGISTER (ADDRESS 18Ch)

R/W-x	U-0	U-0	U-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	—	—	—	WRERR	WREN	WR	RD
bit 7							bit 0

bit 7 **EEPGD**: Program/Data EEPROM Select bit
1 = Accesses program memory
0 = Accesses data memory
(This bit cannot be changed while a read or write operation is in progress)

bit 6-4 **Unimplemented:** Read as '0'

bit 3 **WRERR**: EEPROM Error Flag bit
1 = A write operation is prematurely terminated
(any $\overline{\text{MCLR}}$ Reset or any WDT Reset during normal operation)
0 = The write operation completed

bit 2 **WREN**: EEPROM Write Enable bit
1 = Allows write cycles
0 = Inhibits write to the EEPROM

bit 1 **WR**: Write Control bit
1 = Initiates a write cycle. (The bit is cleared by hardware once write is complete. The WR bit can only be set (not cleared) in software.)
0 = Write cycle to the EEPROM is complete

bit 0 **RD**: Read Control bit
1 = Initiates an EEPROM read. (RD is cleared in hardware. The RD bit can only be set (not cleared) in software.)
0 = Does not initiate an EEPROM read

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Figura 4.20. Registro EECON1. Fuente "PIC 16F87X Datasheet".

Descripción de las funciones de los bits.

- EEPGD: Permite seleccionar la memoria que se quiere grabar EEPROM o la propia memoria FLASH (de programas).
    - EEPGD = 0 (EEPROM de Datos)
    - EEPGD = 1 (Memoria de Programas)
- WRERR: Es una bandera que se pone en uno cuando la escritura se termina antes de tiempo ya sea por un MCLR (RESET) o por desborde del "Watch Dog Timer".
- WREN: Habilita con uno la escritura. Es necesario señalar que este bit solo habilita la escritura, no la inicia.
- WR: Inicia el ciclo de escritura con un uno. Una vez que se termina la escritura, este bit debe ir a cero automáticamente. Por lo tanto es imprescindible chequear que este bit este en cero antes de comenzar una escritura o después de realizarla previo a salir de la subrutina.
- RD: Inicia la lectura. En general la lectura es menos compleja; necesita de menos controles.
- EECON2: Registro de control auxiliar para sincronizar la escritura (dado por el fabricante).

Escribir datos hacia la EEPROM.

Se parte del análisis de la subrutina "w_eeprom". La subrutina "w_eeprom" escribe cualquier dato a la EEPROM siempre que se especifique, antes de llamarla, la dirección (en la variable "dir") y por supuesto el dato a escribir (en la variable "dato_w").

Subrutina

```
;Escribe en la EEPROM de datos
w_eeprom    bsf 3,6
            bsf 3,5     ;banco 3
            ;verifica si no hay escritura en curso
wait        btfsc EECON1, WR
            goto wait
            ;comienza el proceso de una nueva escritura
            ;banco 0
            ;rescata la dirección en donde escribirá
            bcf 3,5
            bcf 3,6
            movf dir,w
            ;banco 2
            ;carga la dirección a EEADR
            bcf 3,5
            bsf 3,6
            movwf EEADR       ;EEADR = dir
```

```
;banco 0
;rescata dato a escribir
bcf 3,5
bcf 3,6
movf dato_w,w
;carga dato en EEDATA
;banco 2
bcf 3,5
bsf 3,6
movwf EEDATA      ;EEDATA = dato_w
;banco 3
;selecciona la memoria (en este caso EEPROM)
;habilita la escritura
bsf 3,5
bsf 3,6
bcf EECON1,EEPGD        ;EEPGD = 0 (EEPROM)
bcf EECON1,WREN         ;WREN = 1 (Escritura habilitada)
;segmento recomendado por el fabricante
;inhabilita ints
bcf INTCON,7
;Secuencia necesaria para sincronía
movlw 0x55
movwf EECON2
movlw 0xAA
movwf EECON2
;Inicia la escritura
bsf EECON1,WR           ;WR = 1 (Inicia la escritura)
;habilita ints
bsf INTCON,7            ;habilita las ints nuevamente
;inhabilita la escritura
bcf EECON1,WREN         ;WREN = 0 (Escritura Inhabilitada)
;regresa al banco 0 !!!!
bcf 3,5
bcf 3,6
return
```

Ejemplo de utilización 1. Graba el número hexadecimal 0x43 en la localización número 0x20 de la EEPROM.

```
movlw 0x20
movwf dir         ;carga la dirección (en este caso 0x20)
movlw 0x43
movwf dato_w
call w_eeprom
;(0x20) = 0x43
```

Ejemplo de utilización 2. Limpia (graba 0x00) desde la dirección 0x00 hasta la dir 0x25.

```
clear_mem   clrf dato_w       ;dato a escribir = 00
            clrf dir          ;dirección inicial = 00
            ;comienza el ciclo
clean       call w_eeprom
            movf dir,w
            sublw 0x25        ;comparo con la dir max a borrar
            btfsc 3,2
            return            ;fin del proceso
            ;seguir borrando
            incf dir          ;paso a la próxima direccion
            goto clean
```

Ejemplo de utilización 3. Limpia la EEPROM desde la dirección "dir_ini" hasta la dirección "dir_final".

```
clear_mem   clrf dato_w       ;dato a escribir = 00
            movf dir_ini,w
            movwf dir         ;comienza a borrar desde "dir_ini"
            ;comienza el ciclo
clean       call w_eeprom
            movf dir,w        ;w = dir (dirección actual de borrado)
            subwf dir_final,w ;dir – dir_final = w
            btfsc 3,2
            return            ;fin del proceso (dir = dir_final)
            ;seguir borrando
            incf dir          ;paso a la próxima dirección
            goto clean
```

- Primero cargar antes de llamarla la dirección inicial del borrado (dir_ini) y la dirección final del borrado (dir_final).
- Definir los simbolos "dir_ini" y "dir_final".

Ejemplo de utilización 4. Graba un grupo de datos que están en la RAM hacia la EEPROM, utilizando la técnica de Direccionado Indirecto de la RAM.

```
graba_dato  movf fsr_ini,w
            movwf 4           ;carga FSR inicial
prox_dato   movf 0,0
            movwf dato_w      ;descarga dato a escribir desde la RAM
            call w_eeprom     ;manda a escribir
            ;chequea si es el ultimo
            movf 4,w
            subwf fsr_max,w
            btfsc 3,2
            return
            ;no es el ultimo incrementar el FSR
```

```
incf 4
incf dir
goto prox_dato
```

- Antes de llamar a la subrutina definir la dirección inicial donde comenzará a grabarse el bloque en la EEPROM (variable dir).
- Antes de llamar a la subrutina definir la dirección inicial de la RAM (donde comienzan los datos que deben grabarse: “fsr_ini”) y la dirección final de los datos en RAM (“fsr_max”), por ejemplo:

```
movf 7
movwf dir          ;dirección inicial en la EEPROM
movlw 0x27
movwf fsr_ini      ;dirección inicial en la RAM (0x27)
movlw 0x2e
movwf fsr_max      ;dirección final en la RAM (0x2e)
call graba_dato
```

Leer datos desde la EEPROM.

Se parte del análisis de la subrutina “r_eeprom”. La subrutina “r_eeprom” lee cualquier dato desde la EEPROM siempre que se especifique, antes de llamarla, la dirección (“dir”) del dato que se desea leer. El dato leído es devuelto por la subrutina en la variable “dato_r”.

Subrutina

```
;Lee la EEPROM de Datos
r_eeprom    ;banco 0
            ;rescata la dirección que debe leerse
            bcf 3,5
            bcf 3,6
            movf dir,w        ;w = dir
            ;banco 2
            ;se carga el registro EEADR
            bcf 3,5
            bsf 3,6
            movwf EEADR       ;EEADR = dir
            ;banco 3
            ;selecciona la EEPROM
            ;iniciar la lectura
            bsf 3,5
            bcf EECON1,EEPGD  ;selecciona EEPROM de Datos
            bsf EECON1,RD     ;inicia lectura
            nop               ;ciclo de espera (opcional)
            ;banco 2
```

```
            ;rescata el dato desde "EEDATA"
            bcf 3,5
            bsf 3,6
            movf EEDATA,w
            ;banco 0
            ;cargar el dato leído en la variable de salida "dato_r"
            bcf 3,5
            bcf 3,6
            movwf dato_r
            return            ;retorna con el dato leído en "dato_r"
```

Ejemplo de utilización de la subrutina "r_eeprom".

```
;lee reloj de tiempo real (fragmentos)
Lee_reloj   ;-----------------------------------------------------
            ;lee horas parte alta
            movlw 0x25
            movwf dir         ;dir = 0x25 (decimas de hora)
            call r_eeprom     ;mando a leer
            movf dato_r,w     ;descargo dato leido
            movwf H_h         ;lo muevo a la RAM
```

Recomendaciones generales para el trabajo con la EEPROM de Datos.

- Al simular la EEPROM es posible introducir el dato inicial que se leerá desde ella después de un RESET; siempre que se haga una lectura después del RESET dentro del programa.
- También es posible al grabar el PIC, grabar la EEPROM, si es conveniente. Para grabar la EEPROM al grabar el PIC se debe actualizar el contenido de la misma a través de la ventana "EEPROM" del menú VIEW.
- Se recomienda inhabilitar ints (ver subrutina), si la subrutina de escritura (w_eeprom), se utiliza fuera de la subrutina de atención a INT.

Modulo CCP (Compara / Captura /PWM). Estructura, Registros de Control y Programación de Aplicaciones. Diseño de Aplicaciones con módulo CCP.

Introducción.

Los aspectos tratados a continuación son válidos para cualquier PIC que posea uno o más módulos CCP. En función de esto aparecerán uno o más registros de control asociados a él o los módulos presentes; es por esto que genéricamente la denominación de los registros terminará en una “x” minúscula.

El Módulo C/C/P es un elemento de hardware adicional que poseen la mayoría los microcontroladores Microchip modernos que puede trabajar en tres modos diferentes y mutuamente excluyentes:

- **Modo Compara.** El módulo funciona como comparador digital, generando una señal ante la coincidencia de valores entre el temporizador 1 (“timer 1”) y un par de registros en los que se ha guardado un número a comparar (CCPRxH y CCPRxL).

- **Modo Captura.** Funciona como un capturador de eventos que se producen en el pin CCPx del puerto C. Cuando ocurre un evento en el pin CCPx, se graba el valor del TMR1, en el par de registros CCPRxH y CCPRxL.

- **Modo PWM.** Funciona como un generador digital de una onda de P.W.M (**“Pulse Width Modulation”**). La resolución de la onda de P.W.M puede alcanzar los 10 bits como máximo.

El **pin CCPx** normalmente es rc2 en los PICs que poseen un solo módulo CCP. En los que poseen más de un módulo, aparecen los pines rc1 y rc3 como pines adicionales CCPx.

Modos de Operación.

El modo de operación para el Módulo CCP se puede seleccionar a partir de los bits CCPxM3 => CCPxM0 del registro que lo controla (CCPxCON). En la **figura 4.21** se muestra el registro de control para el Módulo CCP.

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **CCPxX:CCPxY**: PWM Least Significant bits

Capture mode:
Unused

Compare mode:
Unused

PWM mode:
These bits are the two LSbs of the PWM duty cycle. The eight MSbs are found in CCPRxL.

bit 3-0 **CCPxM3:CCPxM0**: CCPx Mode Select bits

0000 = Capture/Compare/PWM disabled (resets CCPx module)
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode, set output on match (CCPxIF bit is set)
1001 = Compare mode, clear output on match (CCPxIF bit is set)
1010 = Compare mode, generate software interrupt on match (CCPxIF bit is set, CCPx pin is unaffected)
1011 = Compare mode, trigger special event (CCPxIF bit is set, CCPx pin is unaffected); CCP1 resets TMR1; CCP2 resets TMR1 and starts an A/D conversion (if A/D module is enabled)
11xx = PWM mode

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Figura 4.21. Registro de control del módulo CCP. Fuente "PIC 16F87X Data Sheet".

Los bits 3 => 0 sirven para seleccionar el modo y/o la opción de trabajo del Módulo CCP. Los bits 5 y 4 sólo operan en Modo PWM y sirven para almacenar los dos bits menos significativos del valor del Ciclo Útil de la onda de PWM.

Modo "Compara". Funcionamiento.

En este modo, se compara constantemente un valor cargado en los registros CCPRH y CCPRL contra el valor de los registros de temporización del timer 1 (TMR1H y TMR1L). Cuando ambos valores coinciden, se pueden realizar cualquiera de estos 3 procesos sobre el pin CCPx:

- Llevarlo a 1 (combinación "1000").
- Llevarlo a 0 (combinación "1001").
- Que permanezca invariable y se genere una interrupción interna (combinación "1010").

- Que se genere un “disparo de algún evento especial” (combinación “1011”). Esta modalidad se trata en el tópico correspondiente más abajo.

En la **figura 4.22** se muestra un Diagrama de Operaciones Lógico del Modo Compara.

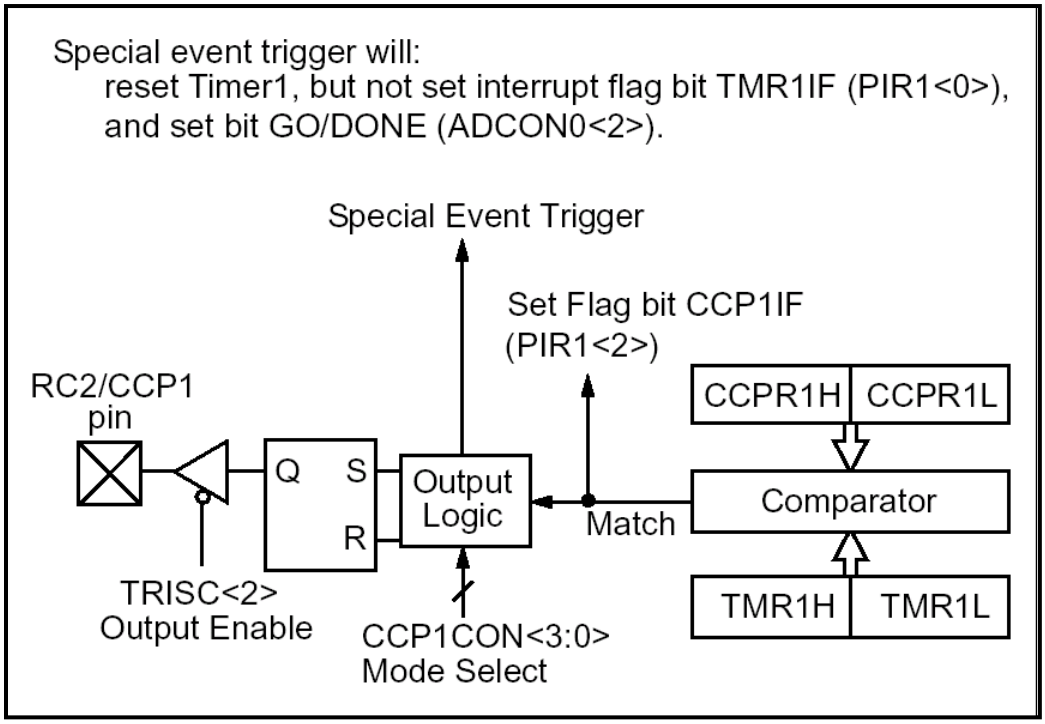


Figura 4.22. Diagrama de Bloques que ilustra el funcionamiento del Modo Compara.

La acción que se realice sobre el pin CCPx está basada en el valor que posean los bits de control CCPxM3:CCPxM0 del registro CCPxCON, tal y como se describe arriba.

El Módulo CCP en modo compara funciona conjuntamente con el timer 1. El timer 1 en este caso sólo aporta el número que debe compararse en todo momento con el par de registros CCPRxH y CCPRxL, lo que no impide que también pueda ser utilizado al mismo tiempo como temporizador si la aplicación lo requiriera. La única excepción para el caso está en que para que el Módulo CCP funcione en el Modo Compara el timer 1 debe ser configurado en Modo Temporizador o en Modo Contador Sincrónico. En Modo Contador Asincrónico no funciona.

Operación del pin CCpx en Modo Compara.

Para operar en Modo Compara se debe setear el pin CCPx como salida limpiando el correspondiente bit del registro TRISC. Al seleccionar el Modo Compara el pin CCP es forzado a un nivel contrario al que tomará cuando coincidan los valores de los registros CCPRx y TMR1x. Por ejemplo si se selecciona Modo Compara para forzar la salida a cero al

coincidir los valores, entonces al seleccionar el Modo Compara la salida será puesta en uno.

Modo de Interrupción por SW (Software Interrupt Mode).

Si se selecciona esta opción dentro del Modo Compara, al producirse la coincidencia se producirá una interrupción si esta se ha habilitado y el estado del pin CCPx no cambiará.

Modo de Disparo por Evento Especial (Special Event Trigger).

En esta opción se genera un pulso de disparo interno, cuando se produce la coincidencia, que puede aprovecharse para iniciar alguna acción, por ejemplo, iniciar una conversión analógico – digital.

Modo "Captura". Funcionamiento.

Funcionamiento.

En este modo cuando ocurre un evento en el pin CCPx, se graba el valor de los registros de temporización del timer 1 (TMR1H y TMR1L), en el par de registros CCPRxH y CCPRxL. Los eventos que pueden ocurrir en el pin CCPx pueden ser:

- Cada frente de caída de la señal **(combinación 0100)**.
- Cada frente de subida de la señal **(combinación 0101)**.
- Cada 4**to** frente de subida de la señal **(combinación 0110)**.
- Cada 16**avo** frente de subida de la señal **(combinación 0111)**.

Las diferentes opciones de eventos a detectar se seleccionan mediante los 4 primeros bits del registro CCPxCON (**figura 4.21**).

En la **figura 4.23** se muestra Diagrama Lógico de Operación en Modo Captura.

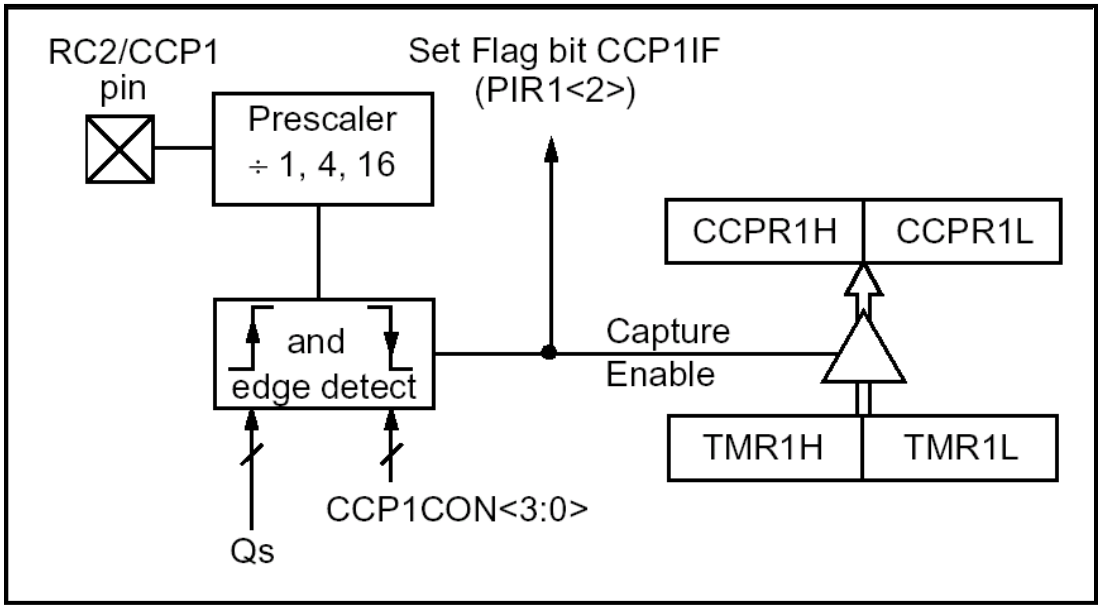


Figura 4.23. Diagrama de Bloques que ilustra el funcionamiento del Modo Captura. Cada vez que se produce un evento de Captura, se produce una INT (CCPxIF = 1).

Operación del pin CCpx en Modo Captura.

El pin CCPx debe ser programado como una entrada, seteando el correspondiente bit del registro TRISC.

Modo "PWM". Funcionamiento.

En modo PWM el Módulo CCP funciona como generador digital de PWM produciendo una salida de PWM de hasta 10 bits de resolución en el pin CCPx. Debido a que la salida de PWM se multiplexa con la línea de puerto correspondiente del puerto C, este debe ser configurado como salida limpiando el bit correspondiente del registro TRISC.

Nota: si se realiza un CLRF sobre el registro CCPxCON el pin CCPx va a cero, (valor por default del modo PWM. Este hecho no tiene nada que ver con la línea de puerto con la que está multiplexada la salida de PWM. La generación de PWM utiliza el temporizador 2 del PIC para establecer la frecuencia de la onda de PWM.

Modulación por Ancho de Pulso (P.W.M).

PWM es un tipo de modulación que se basa en la variación del Ciclo Útil de un tren de pulsos. En la **figura 4.24** se muestra un tren de pulsos y sus parámetros.

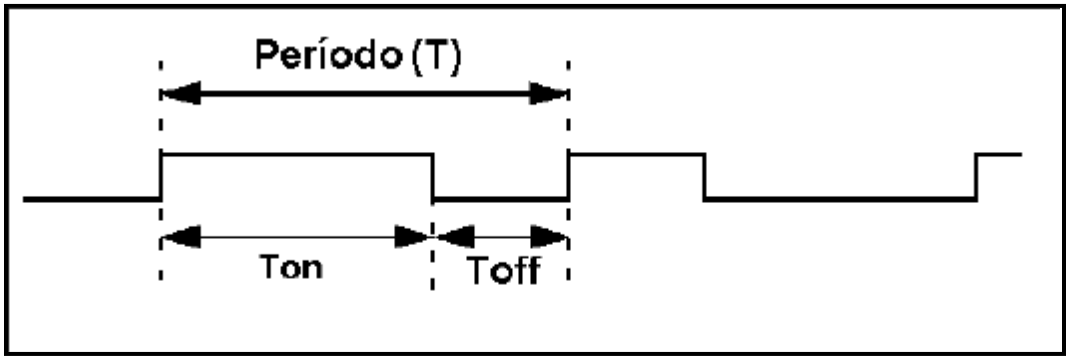


Figura 4.24 Tren de Pulsos y sus parámetros.

Dentro de una onda de PWM son importantes los siguientes parámetros:

Período y Frecuencia de la Onda de PWM.

Dado que el generador de PWM incluido en los PICs es digital, el período de la onda PWM (T) se especifica escribiendo el registro PR2 del microcontrolador. Se debe cargar un número entre 00 y FF hex. En el registro PR2 para garantizar la frecuencia de PWM deseada. El número a cargar en PR2 para una frecuencia de PWM deseada puede calcularse utilizando la siguiente fórmula:

 # a cargar en PR2 = TPWM /(4 * TOSC * Valor del Prescaler del Timer2)

La Frecuencia de PWM.

Se define como el inverso del período de PWM **[1/ TPWM]**. De la expresión anterior se puede inferir que la Frecuencia de PWM que se desea es uno de 256 valores posibles y su valor concreto depende de la frecuencia de Oscilador Externo que se utilice, por lo que su elección no es arbitraria.

Cuando el registro de temporización del timer 2 (TMR2) es igual al registro PR2 (se completa un ciclo de la onda de PWM), ocurren los tres eventos siguientes durante el siguiente ciclo de instrucción:

- Se limpia el registro TMR2 (se inicia el timer 2).
- El pin CCPx se pone en uno excepto en el caso de ciclo útil de PWM = 0%.
- El Ciclo Útil de PWM es copiado del registro CCPRxL al registro CCPRxH.

Nota: El postscaler del Timer 2 no se utiliza en la determinación de la frecuencia de PWM. El postscaler pudiera utilizarse para realizar una actualización periódica servo controlada a una frecuencia diferente a la salida de PWM.

Ciclo Útil de PWM.

En general el Ciclo Útil de un tren de pulsos se define como:

CU % = [Ton / T] * 100; expresado en %

En un Generador de PWM Analógico, teóricamente la onda de PWM tendría infinitos valores de Ciclo Útil, por lo que se dice que su resolución es infinita no así en un Generador Digital. El Ciclo Útil de la onda de PWM en el Generador de PWM en PICs se especifica con un número de 10 bits. En el registro CCPRxL se escriben los 8 bits más significativos y en los bits DCxB1:DCxB0 (bits 4 y 5 del registro CCPxCON) se escriben los dos menos significativos restantes. Según lo anterior es posible lograr una resolución de hasta 10 bits dado que el registro CCPRxL contiene los 8 bits más significativos y los bits DCxB1 y DCxB0 contienen los dos bits menos significativos del número de la resolución. Una resolución de 10 bits implica que el número máximo de valores de Ciclo Útil en que se podrá subdividir un Período de PWM es igual a 1024 (210)

El valor del registro CCPRxL y de los bits DCxB1:DCxB0 se pueden modificar en cualquier momento de la ejecución del programa para cambiar el Ciclo Útil (CU).

Resolución de la onda de PWM.

Como se había señalado la resolución de la onda de PWM está dada en bits y su valor no es arbitrario (puede alcanzar hasta 10 bits), lo que significa que si se fija la Frecuencia de PWM y la Frecuencia del Oscilador del Sistema la resolución de la onda quedará determinada a través de la siguiente expresión:

$$\textbf{Resolución} = \frac{\log\left(\frac{F_{OSC}}{F_{PWM}}\right)}{\log(2)} \text{ [bits]}$$

Ejemplo 1. Calcular si es posible garantizar una resolución de 10 bits para el generador de PWM si se desea trabajar con una Frecuencia de PWM de 1000 Hz y una Frecuencia de Oscilador igual a 16 MHz.

Primero se calcula el # a cargar en el registro PR2 para constatar si es posible lograr la Frecuencia de PWM deseada dado el oscilador que se está utilizando (16 MHz).

 # a cargar en PR2 = TPWM / (4 * TOSC * Valor del Prescaler del Timer2)

 # a cargar en PR2 = (1/1000 Hz) /(4 * 1/16 Mhz * 16)

 # a cargar en PR2 = 1 *** 10^{-3} * 0,25 **** 10^{6}

 # a cargar en PR2 = 250 (en decimal)

Como el número a cargar resultó menor a 255 es adecuada la elección de la Frecuencia de PWM.

Segundo calcular la resolución:

$$\textbf{Resolución} = \frac{\log\left(\frac{F_{OSC}}{F_{PWM}}\right)}{\log(2)} \text{ [bits]}$$

Resolución = [log (16 MHz / 1000 Hz)] / log 2

Resolución = 13,96 bits

Como el valor encontrado es superior a 10 bits, significa que es posible lograr una resolución de 10 bits por lo que es adecuada la elección de ambas frecuencias.

Ejemplo 2. Encontrar la máxima resolución del CU que puede alcanzarse con una frecuencia de 78.125 KHZ y un Oscilador Externo de 20 MHz.

Si se desea una frecuencia de PWM de 78.125 Khz. y se está trabajando con un oscilador de 20 Mhz para un valor de prescaler del Timer2 = 1 se tiene que:

1/78.125 Khz. = [(PR2) + 1] · 4 · 1/20 MHz · 1

12.8 mS = [(PR2) + 1] · 4 · 50 ns · 1

PR2 = 63, este es el valor decimal que se debe cargar en el registro PR2.

Calculando igual que en el **ejemplo 1** la resolución se tiene que:

Resolución de la onda de PWM = 8.0

Este resultado significa que como máximo se puede lograr una resolución de 8 bits en el Ciclo Útil si se genera una Frecuencia de PWM de 78.125 Khz. y se utiliza un oscilador de 20 Mhz, que es la frecuencia máxima de operación para la serie 87X. Significa también que el valor cargado en los bits DCxB9 : DCxB0 será un valor entre 0 y 255. Cualquier valor mayor a 255 resultará en un Ciclo Útil del 100%.

Conclusión. Para lograr una mayor resolución en el CU, o sea que el CU pueda variarse en un mayor espectro de valores, se debe trabajar a una frecuencia de PWM menor. Así mismo para lograr una mayor frecuencia de PWM, se debe decrementar la resolución para una Frecuencia de Oscilador Externo fija.

Pasos para configurar el Módulo CCP en Modo PWM.

1. Establecer el período de PWM escribiendo el registro PR2.
2. Establecer el CU escribiendo los bits DCxB9 : DcxB0.
3. Programar el pin CCPx como salida poniendo en cero el bit correspondiente del registro TRISC.
4. Establecer el valor del Pscaler para el Tmr2 y habilitar el Tmr2 escribiendo el registro T2CON.
5. Configurar el Módulo CCP para operación en PWM.

Ejercicios Propuestos.

1. ¿Cuáles temporizadores pueden utilizarse para garantizar el funcionamiento de los modos “compara y captura” ?

2. ¿Cuáles temporizadores pueden utilizarse en los modos PWM ?

3. ¿Puedo utilizar un módulo C/C/P para realizar captura (o comparación) y PWM al mismo tiempo, ya que ellos emplean como referencia temporizadores diferentes?.

4. ¿Cómo afecta el RESET al módulo C/C/P ?.

5. Estoy configurando el registro CCP1CON para “ modo compara y disparo mediante evento especial (1011) ”, condición que lleva al RESET al timer1. Cuando el timer 1 es igual al número cargado en los registros CCPR1L y CCPR1H¿tendré dos interrupciones pendientes, la del tmr1 y la del módulo CCP, o sea las banderas TMR1IF y CCP1IF están en uno.?

Tema V: Adquisición de Datos

Sumario:

1. Introducción a la Conversión A/D. Muestreo y cuantificación de señales. Estructura de un Conversor A/D. Tipos de Conversores A/D.
2. Parámetros de un conversor A/D.Módulo Conversor Análogo/Digital. Registros de Control para el módulo ADC. Requisitos para la operación del Conversor.
3. Modos de Operación del Conversor A/D. Conversión A/D multicanal.Diseño de Aplicaciones.

Introducción a la Conversión A/D. Muestreo y cuantificación de señales. Estructura de un Conversor A/D. Tipos de Conversores A/D.

Módulo Conversor Análogo/Digital.

Especificaciones del Conversor A/D.

Dentro de las especificaciones más importantes del Conversor A/D integrado a los Microcontroladores de la Familia Media se pueden resumir las siguientes:

- Existe un solo conversor dentro del microcontrolador por lo que se multiplexan los canales analógicos de entrada que pudieran existir. En los PICs de 28 pines se incluyen hasta 5 canales analógicos multiplexados y en los de 40 pines hasta 8 canales (ra0, ra1, ra2, ra3, ra5, re0, re1 y re2).
- Como ocurre en todos los conversores A/D la entrada analógica carga el capacitor de un circuito **"Sample and Hold".** La salida del capacitor del "sample and hold" es la entrada al cuantificador. Para cada muestra de la señal analógica almacenada el cuantificador genera entonces un código digital de 10 bits mediante el método de aproximaciones sucesivas.
- El A/D maneja también un **Voltaje de Referencia** superior e inferior, que se puede seleccionar entre cualesquiera de estas tres posibilidades:
    - **Vdd.**
    - **Vss.**
    - **Un voltaje conectado al pin ra2.**
    - **Un voltaje conectado al pin ra3.**
- El A/D tiene la posibilidad exclusiva de operar en "Modo Sleep" o de bajo consumo**.** Para operar en modo "sleep", el reloj del ADC debe ser derivado del oscilador RC interno.
- El Conversor A/D se controla a través de dos registros:
    - A/D Control Register 0 (ADCON0).
    - A/D Control Register 1 (ADCON1).
- El resultado de la conversión se almacena en un par de registros:
    - A/D Result High Register. (ADRESH).
    - A/D Result Low Register. (ADRESL).

Parámetros de un conversor A/D.Módulo Conversor Análogo/Digital. Registros de Control para el módulo ADC. Requisitos para la operación del Conversor.

Registros de Control del Conversor A/D.

A/D Control Register 0 (ADCON0).

Se ubica en la dirección 1Fh del banco 0 de la RAM. La **figura 5.1** muestra la distribución de bits para el registro.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit7							bit0

bit 7-6: **ADCS1:ADCS0: Bits de selección del reloj de conversión.**
00 = FOSC/2
01 = FOSC/8
10 = FOSC/32
11 = FRC **Reloj derivado del Osc. RC**

bit 5-3: **CHS2:CHS0: Bit de selección del canal.**
000 = **Canal** 0, (RA0/AN0)
001 = **Canal** 1, (RA1/AN1)
010 = **Canal** 2, (RA2/AN2)
011 = **Canal** 3, (RA3/AN3)
100 = **Canal** 4, (RA5/AN4)
101 = **Canal** 5, (RE0/AN5)(1)
110 = **Canal** 6, (RE1/AN6)(1)
111 = **Canal** 7, (RE2/AN7)(1)
No disponibles en el PIC16F870

bit 2: **GO/$\overline{\text{DONE}}$: Bit de Status de Conversión. (Este BIT está condicionado a que el BIT ADON esté en 1).**
1 = Conversión en progreso.
0 = Fin de Conversión

bit 1: **No implementado, se lee como cero.**

bit 0: **ADON Bit de apagado y encendido del A/D**
1 = **A/D encendido.**
0 = **A/D apagado.**

Figura 5.1. Registro ADCON0. Fuente "PIC 16F87X Data Sheet".

El registro **ADCON 0**, permite controlar las siguientes funciones:

- Seleccionar el **reloj de conversión**. El **reloj de conversión** es el período de la señal de reloj interna al conversor que controla el trabajo del cuantificador. Por cada período de la señal de reloj se construye un bit del código de la salida. Según especifica el fabricante, la combinación de los bits 7 y 6 debe escogerse de forma

tal, que el valor obtenido sea mayor a 1,6µS. consulte sección**: "Seleccionando el Reloj de Conversión"**.

- Los **bits 5 ... 3** permiten seleccionar el canal que se desea convertir.
- El **bit 2** sirve para iniciar la conversión **(GO)** o para verificar si se ha finalizado **(*DONE)**.
- El **bit 0** es un interruptor que permite apagar o encender el conversor.

A/D Control Register 1 (ADCON1).

Se ubica en la dirección 9Fh del banco 1 de la RAM. La **figura 5.2** muestra la distribución de bits para el registro.

U-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit7							bit0

bit 7: **ADFM: Selección del Formato del resultado del ADC.**

1 = **Justificado a la derecha: los 6 bits más significativos del ADRESH se leen como ceros.**

0 = **Justificado a la izquierda los 6 bits menos significativos del ADERESL se leen como ceros.**

bit 6-4: **No implementados, se leen como ceros.**

bit 3-0: **PCFG3:PCFG0: Configuración de las líneas de los puertos A y E**

PCFG3: PCFG0	AN7(1) RE2	AN6(1) RE1	AN5(1) RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	VREF+	VREF-	CHAN / Refs(2)
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	RA3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	RA3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	RA3	VSS	2/1
011x	D	D	D	D	D	D	D	D	VDD	VSS	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	RA3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1/2

A = **Entrada analógica.**

D = **Entrada digital**

Figura 5.2. Registro ADCON1. Estructura de control para PICs de 40 pines dentro de la serie 87X. Fuente: PIC16F87X Datasheet (www.microchip.com).

El registro **ADCON 1**, permite controlar las siguientes funciones:

- Especificar en qué formato se quiere obtener el resultado de la conversión (**figura 5.3**).
- Especificar si las líneas de los puertos A y E que están multiplexadas con las entradas analógicas del conversor son digitales o analógicas. (**figura 5.2**).

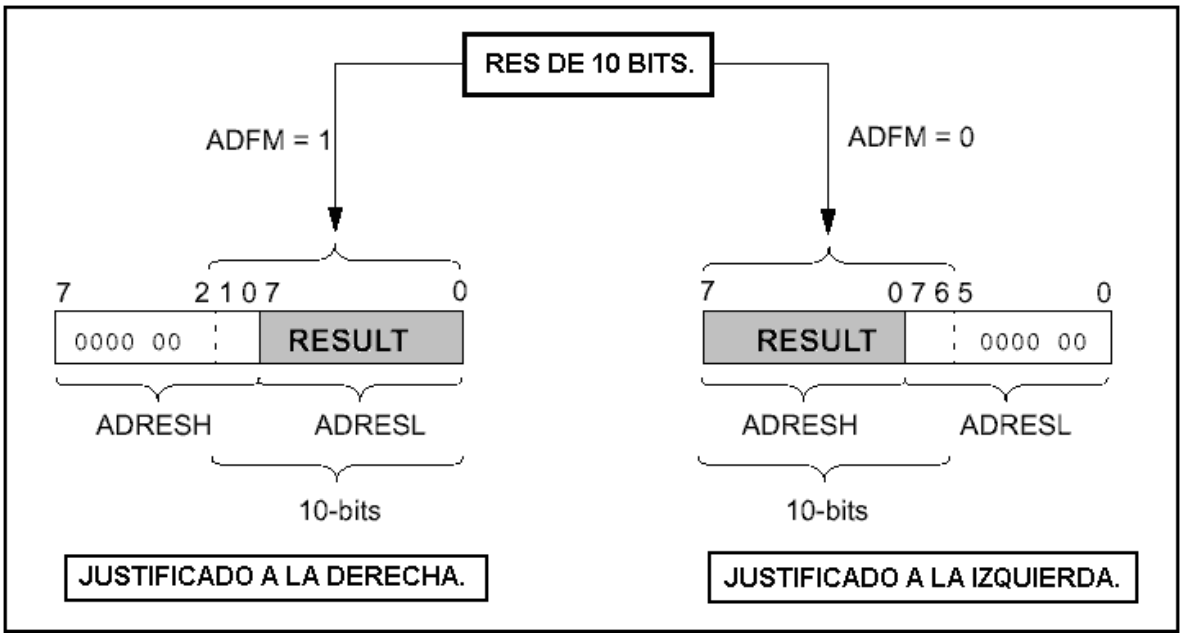


Figura 3. Formatos del resultado de la conversión.

Registros almacenadores del resultado de la conversión.

Los registros almacenadores del resultado son: A/D Result High Register. (ADRESH), dirección 1Eh del banco 0 de la RAM y A/D Result Low Register. (ADRESL), dirección 9Eh del banco 1 de la RAM. El valor de estos registros se actualiza cada vez que se finaliza una conversión completa señalizada con un cero en el bit "GO" del registro ADCON0. El resultado de la conversión se puede leer en el formato que se haya especificado, según el bit ADFM del registro ADCON1.

Operación del conversor. Requisitos para el tiempo mínimo de adquisición. Secuencia de Operación.

1. Configurar el módulo A/D.
   - **Configurar los pines** analógicos, referencia de voltajes y E/S digitales utilizando los correspondientes bits de control del registro ADCON1.

- Seleccionar el **canal analógico** utilizando los correspondientes bits de control del registro ADCON0.
- Seleccionar el **reloj de conversión** utilizando los correspondientes bits de control del registro ADCON0.
- Encender el módulo A/D utilizando el bit de control "ADON" del registro ADCON0. El encendido también puede realizarse si se desea antes de iniciar la conversión del dato.

2. **Configurar la interrupción del A/D si se desea atender el fin de conversión por interrupción.** La ventaja de atender el fin de la conversión por interrupción es el ahorro del tiempo de cuantificación. Tiempo durante el cual el microcontrolador pudiera realizar otras operaciones. Configurar la interrupción del A/D necesita de lo siguiente:
   - Limpiar la bandera ADIF.
   - Setear el BIT ADIE.
   - Setear el BIT GIE.
3. **Esperar el tiempo de adquisición requerido.** El tiempo de adquisición es el tiempo de carga del capacitor del "sample and hold (S/H)".
4. **Iniciar la conversión**
   - Setear el BIT GO/DONE* (reg. ADCON0).
5. **Chequear que se complete la conversión** del dato, ya sea:
   - Encuestando el bit GO/DONE* hasta que se ponga en cero.
   - Cuando ocurra la interrupción del A/D si había sido habilitada.
6. **Leer el resultado** en el par de registros ADRESL y ADRESH y limpiar el BIT ADIF, en el caso en que el fin de la conversión haya sido atendido por interrupción.
7. Para realizar una **nueva conversión ir a los pasos 1 y 2** según se requiera.

Requisitos para la adquisición.

Para que la conversión sea precisa, se debe lograr la carga completa del capacitor del S/H. El nivel de carga, que este capacitor logre estará en función de la resistencia de la fuente analógica y de las características de la línea de entrada analógica. En la **figura 5.4** se presenta un modelo para todos los pines analógicos en los MC Microchip de esta familia.

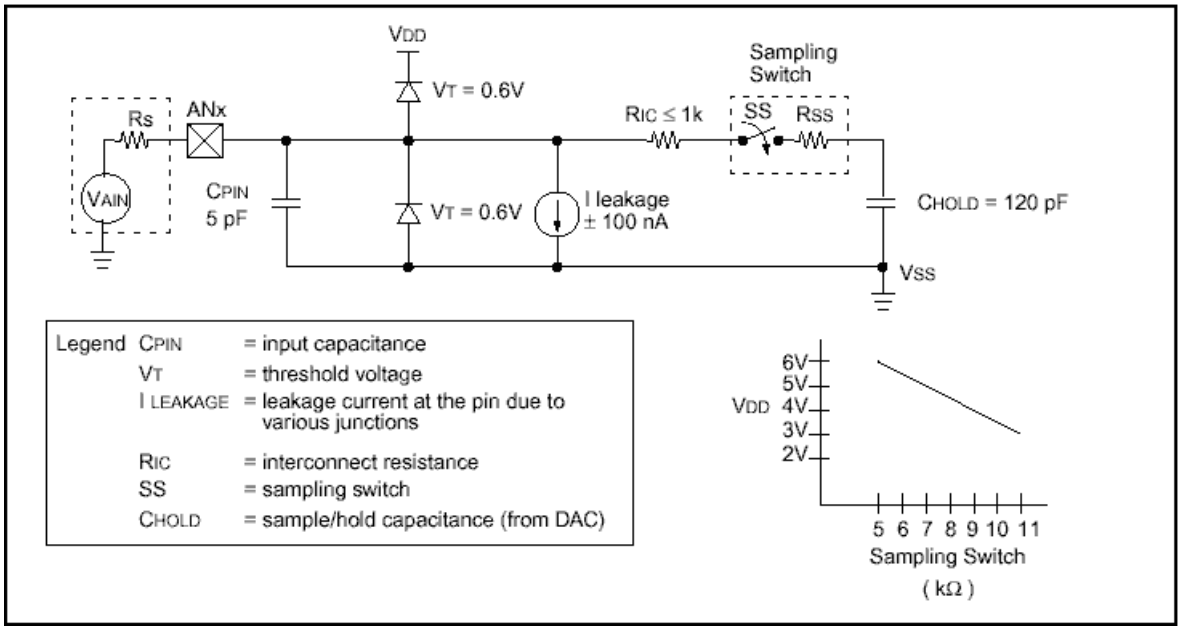


Figura 5.4. Modelo Eléctrico para cualquier pin de entrada analógico.

Tiempo de adquisición.

Es el tiempo mínimo que debe ser esperado hasta que se cargue el capacitor del S/H del conversor, antes de ordenar el inicio de una conversión. Este tiempo puede calcularse empleando la siguiente expresión:

$$T_{ACQ} = T_{AMP} + T_C + T_{COFF}$$

$$T_{ACQ} = \boxed{2\ \mu s} + \boxed{(120\ pF)(1\ k\Omega + R_{SS} + R_S)\ \ln(1/2047)} + \boxed{[(Temp - 25°C)(0.05\ \mu s/°C)]}$$

En la expresión:

- **Tamp:** es el tiempo de respuesta de la salida del amplificador (o el dispositivo) que hay conectado a la entrada analógica. 2µS es un valor típico.
- **Tc:** es el tiempo de carga del capacitor del S/H.
- **RSS:** resistencia de ON del switch, típicamente de 7 K.
- **RS:** resistencia interna de la fuente de señal.
- **Nota:** el valor de 120 pF corresponde al valor típico del capacitor del S/H (**figura 5.4**).
- **Tcoff:** es el coeficiente de temperatura (la respuesta del S/H varia con la temperatura del chip).
- **Temp:** temperatura de trabajo del chip.

Ejemplo:

Supongamos que se trabaja a una temperatura de 50 °C, con una fuente analógica que tiene una resistencia interna de 10K, una fuente de alimentación de 5V y una resistencia interna del switch de 7K.

$$T_{ACQ} = T_{AMP} + T_C + T_{COFF}$$

$$T_{ACQ} = 2\ \mu s + T_c + [(Temp - 25°C)(0.05\ \mu s/°C)]$$

$$\begin{aligned} T_C = & -C_{HOLD}\,(R_{IC} + R_{SS} + R_S)\ \ln(1/2047) \\ & -120\ pF\ (1\ k\Omega + 7\ k\Omega + 10\ k\Omega)\ \ln(0.0004885) \\ & -120\ pF\ (18\ k\Omega)\ \ln(0.0004885) \\ & -2.16\ \mu s\ (-7.6241) \\ & 16.47\ \mu s \end{aligned}$$

$$\begin{aligned} T_{ACQ} = & 2\ \mu s + 16.47\ \mu s + [(50°C - 25°C)(0.05\ \mu s/°C)] \\ & 18.47\ \mu s + 1.25\ \mu s \\ & 19.72\ \mu s \end{aligned}$$

Nota: el coeficiente de temp. (TCOFF) solo es necesario para temp > 25oC.

Estos resultados implican que hay que esperar unos 20 µS antes de ordenar el inicio de conversión.

Ciclo de Conversión total.

Un ciclo de conversión total es igual al tiempo de adquisición mas el tiempo de cuantificación (**figura 5.5**):

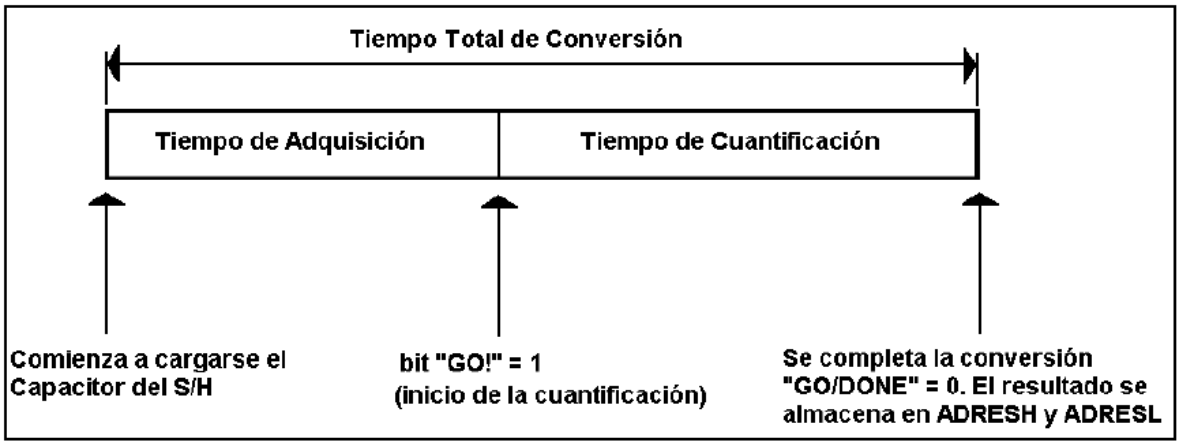


Figura 5.5. Tiempo Total de Conversión de un Dato (esquema).

El tiempo de conversión es igual a 12 veces el Periodo del Reloj de Conversión (Tad). La suma de este tiempo más el tiempo de adquisición es igual al Tiempo Total de Conversión.

Seleccionando el Período del Reloj de Conversión (Tad) adecuado.

El “Tad” es el período de la señal del reloj de cuantificación interno al A/D. Un período de esta señal es igual al tiempo que demora el A/D en construir un bit de los 10 posibles del resultado.

La fuente del reloj para realizar la conversión se selecciona con los bits ADCS1 y ADCS0 del registro ADCON0.

Para lograr una conversión correcta, el reloj de conversión del A/D debe ser mayor o igual a 1,6µS. Este parámetro será inviolable.

En la práctica, la selección de la combinación correcta para los bits ADCS1 y ADCS0 está relacionada con la frecuencia del Oscilador Externo que se utiliza para el microcontrolador. Por ejemplo si en la aplicación se utiliza una frecuencia de 16 MHz la combinación adecuada será “10”, que corresponde a la restricción: 32 Tosc:

Si f = 16 Mhz; el Tosc = 1/16Mhz = 0,0625µS

Entonces:

32 * 0,0625µS = 2 µS que supera los 1,6 µS que propone el fabricante.

Modos de Operación del Conversor A/D. Conversión A/D multicanal.Diseño de Aplicaciones.

Empleo practico del conversor.

Para emplear el conversor correctamente es necesario tener claros los principios básicos que trata la “Teoría de Cuantificación y Muestreo”; lo que significa valorar dos cosas:

- Escoger un tiempo de muestreo adecuado de modo que básicamente cumpla con el Teorema de Nyquist.
- Valorar si la resolución máxima del conversor (10bits) es adecuada para garantizar un mínimo de Ruido de Cuantificación; dentro de la señal digital; de lo contrario no se podrá emplear el conversor integrado y deberá conectarse uno externo de mayor resolución.

Tiempo de Muestreo.

Es el tiempo que transcurre entre la toma de muestras sucesivas de una señal. Puede ser constante o no dependiendo de las necesidades y objetivos que se persigan dentro de la aplicación. Normalmente es constante y su valor es mucho mayor que el Tiempo Total de Conversión tratado en la sección anterior. La **figura 5.6** muestra un grafico a escala de todos los tiempos involucrados en la adquisición de una señal.

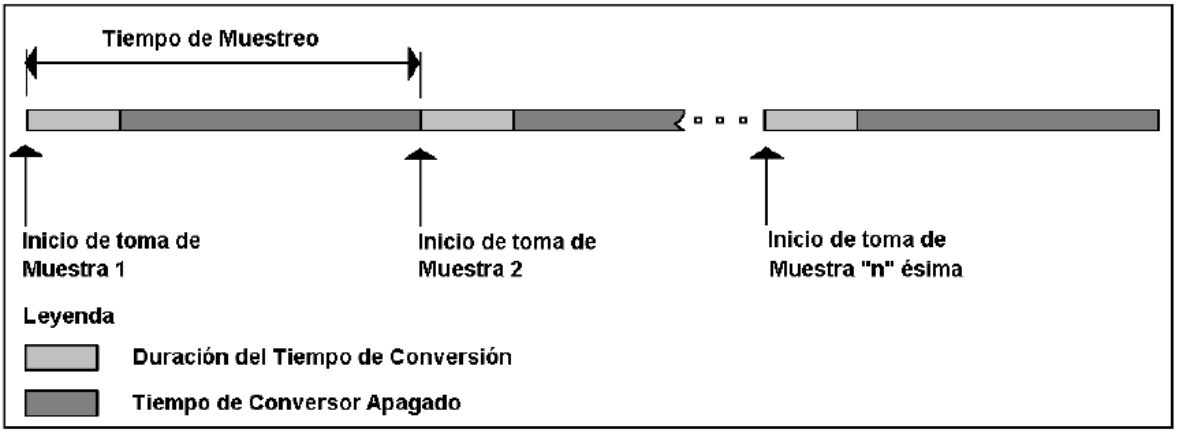


Figura 5.6. Tiempo de muestreo.

El tiempo de muestreo normalmente es generado por un temporizador o una demora fabricada por software. En cualquier caso una vez que se finaliza un intervalo de muestreo debe adquirirse una muestra de la señal. Existen básicamente dos técnicas para tender la conversión de una muestra:

- Encuestando el fin de conversión por software.
- Permitiendo que el conversor genere una interrupción una vez que finalice la conversión.

Fin de Conversión por encuesta.

Primero se configura el conversor. El ejemplo siguiente supone que se desea convertir el canal 0 (pin ra0) con dato justificado a la izquierda, utilizando una frecuencia de oscilador externo de 16 MHz.

Ejemplo 1. Configuración del conversor para fin de conversión por encuesta.

```
;En el Bloque de Configuración de la Aplicación se debe
;establecer lo siguiente.
;-----------------------------------------------------------------
;Configuración del Registro ADCON1
      bsf 3,5 ;banco 1 de la RAM
      movlw b'00001110' ;todos los canales son digitales excepto ra0
      ;y resultado justificado a la izquierda.
```

```
      movwf ADCON1
;-----------------------------------------------------------------
      □
      □
      □
      □
      □
      bcf 3,5 ;banco 0 de la RAM


;-----------------------------------------------------------------
;Configuración del Registro ADCON0:
;reloj de conversión mayor a 1,6µS (combinación "10").
;canal cero seleccionado
;conversor apagado al inicio
;GO/DONE = 0 (no hay conversión en curso)
      movlw b'10000000'
      movwf ADCON0
      □
      □
      □
      □
      □
```

Segundo convertir el dato. Como se señaló anteriormente, el tiempo de conversión se puede generar a partir de la interrupción de un temporizador o una demora por software. En cualquier caso pudiera utilizarse la siguiente subrutina, que en este caso ha sido denominada "convierte". La subrutina "convierte" sigue el procedimiento adecuado para logra una conversión exitosa y se puede utilizar siempre que el conversor esté configurado y el canal a convertir seleccionado.

```
;subrutina de conversión para cualquier canal (previamente
;seleccionando por encuesta

convierte   bsf ADCON0,0 ;enciende el conversor A/D
            call dem_tacq ;demora del Tiempo de Adquisición
            bsf ADCON0,2 ;GO! (inicia la cuantificación)

;encuesta del bit GO/DONE hasta que finalice la conversión
;en este caso se pierde un tiempo de ejecución = 12 □  32 □
;0,0625µS, que es lo que dura la cuantificación.
wait        btfsc ADCON0,2
            goto wait

;fin de la conversión, resultado listo en los
;registros ADRESH y ADRESL
            bcf ADCON0,0 ;apaga conversor A/D
            return
```

Fin de Conversión por Interrupción del conversor A/D.

Este caso exige además de la configuración realizada en el **ejemplo 1**, también la habilitación del sistema de interrupciones del microcontrolador y particularmente del conversor A/D. Se puede además utilizar la subrutina convierte pero suprimiendo la espera por el fin de conversión, ya que en este caso el aviso de fin de conversión es automático (aviso de interrupción del A/D). El ejemplo 2 muestra las modificaciones para el caso.

Ejemplo 2. Configuración del conversor para fin de conversión por interrupción.

```
;En el Bloque de Configuración de la Aplicación se debe
;establecer lo siguiente.
;-----------------------------------------------------------------
;Configuración del Registro ADCON1
bsf 3,5 ;banco 1 de la RAM

movlw b'00001110' ;todos los canales son digitales excepto ra0
                  ;y resultado justificado a la izquierda.
movwf ADCON1
;-----------------------------------------------------------------
□
□
□
□
□  ;-----------------------------------------------------------------
;habilitación de la interrupción del Conversor A/D
bsf PIE1,ADIE
bcf 3,5 ;banco 0 de la RAM
;-----------------------------------------------------------------
;Configuración del Registro ADCON0:
;reloj de conversión mayor a 1,6µS (combinación "10").
;canal cero seleccionado
;conversor apagado al inicio
;GO/DONE = 0 (no hay conversión en curso)

movlw b'10000000'
movwf ADCON0
;-----------------------------------------------------------------
;habilitación GLOBAL de las interrupciónes
;(GIE = 1, PEIE = 1).
bsf INTCON,7
bsf INTCON,5
□
□
□
□
□
```

Por otra parte la subrutina “convierte” se modificaría en este caso como sigue:

```
;subrutina de conversión para cualquier canal (previamente
;seleccionando por encuesta

convierte   bsf ADCON0,0 ;enciende el conversor A/D
            call dem_tacq ;demora del Tiempo de Adquisición
            bsf ADCON0,2 ;GO! (inicia la cuantificación)
            ;el fin de cuantificación genererá una interrupción.
            return
```

En la interrupción se tendría en cuenta lo siguiente:

```
;subrutina de Atención a Interrupción. Este caso incluye la
;recogida del dato convertido

IT    ;-----------------------------------------------------
      ;respaldo de registros.
      movwf wtemp       ;respaldo acumulador.
      swapf STATUS,w
      movwf stat_temp   ;respaldo del STATUS
      movf FSR,0
      movwf fsr_temp    ;respaldo del FSR
      ;-----------------------------------------------------
      ;encuesta de bandera de interrupción del A/D.

      btfss PIR1,ADIF ;ADIF =? 1
      goto back ;regresa de int (error)
      ;INT del A/D, recoger resultado de la conversión
      ;-----------------------------------------------------
      □
      □
      □
      □
      ;apaga conversor A/D
      bcf ADCON0,0
      ;regresa de interrupción

Back  bcf PIR1,ADIF     ;limpia la bandera del Conversor
                        ;restituye registros
      movf fsr_temp,w
      movwf FSR         ;restituye FSR
      swapf stat_temp,w
      movwf STATUS      ;restituye STATUS
      swapf wtemp,f
      swapf wtemp,w     ;restituye W sin afectar
                        ;las banderas del STATUS.
                        ;la instrucción "swapf f "
                        ;no afecta el contenido del STATUS
      retfie            ;retorno de la subrutina.
```

Tema VI: Comunicación Serial

Sumario:

1. Introducción a la Comunicación Serial RS-232C utilizando Microcontroladores.
2. Módulo USART. Registros de Control. Operación del Módulo en Modo Asincrónico; Sincrónico Máster y Sincrónico Esclavo.
3. Módulo MSSP.
4. Diseño de Aplicaciones.

El USART (Universal Synchronous Asyncronous Receiver Transmitter) integrado a los microcontroladores PICs de la Familia Media.

Introducción.

El USART ("Universal Synchronous Asynchronous Receiver Transmitter") es uno de los dos módulos de comunicación serial que poseen los PICS de la familia media. El USART es también conocido como "Serial Interface" o "SCI". El USART se puede configurar como un sistema asincrónico "full duplex" que puede comunicarse con periféricos como pantallas CRT, o PC´s o puede configurarse como un sistema sincrónico "half duplex" que puede comunicarse con circuitos integrados tales como A/D's o D/A's, Memorias Seriales EEPROMs, entre otros dispositivos.

Modos de trabajo. Introducción.

El **USART** puede configurarse en 3 modos de trabajo diferentes:

- Asincrónico ("full duplex").
- Sincrónico Master ("half duplex").
- Sincrónico Esclavo ("half duplex").

Modo Asíncrono.

Este modo se utiliza para comunicación con periféricos asíncronos (Computadoras Personales, por ejemplo) en modo **full duplex.** Su funcionamiento no necesita de enlace de reloj. Cada dispositivo involucrado refiere la comunicación a un Reloj Interno previamente definido para ambos.

En la **figura 6.1** se muestra la trama utilizada para el protocolo asíncrono.

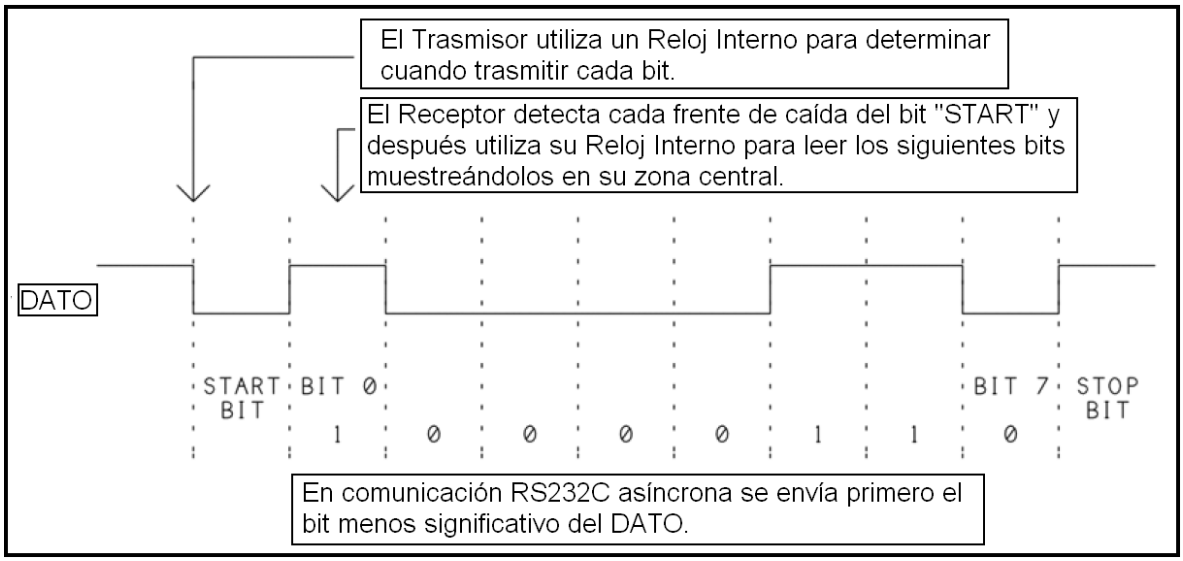


Figura 6.1. Trama asincrónica.

Modo síncrono.

Este modo se utiliza para comunicación con dispositivos síncronos (Memorias Seriales Externas, por ejemplo). La comunicación en este modo es half dúplex y dependiente del enlace físico del reloj.

En la **figura 6.2** se muestra la trama utilizada para el protocolo síncrono.

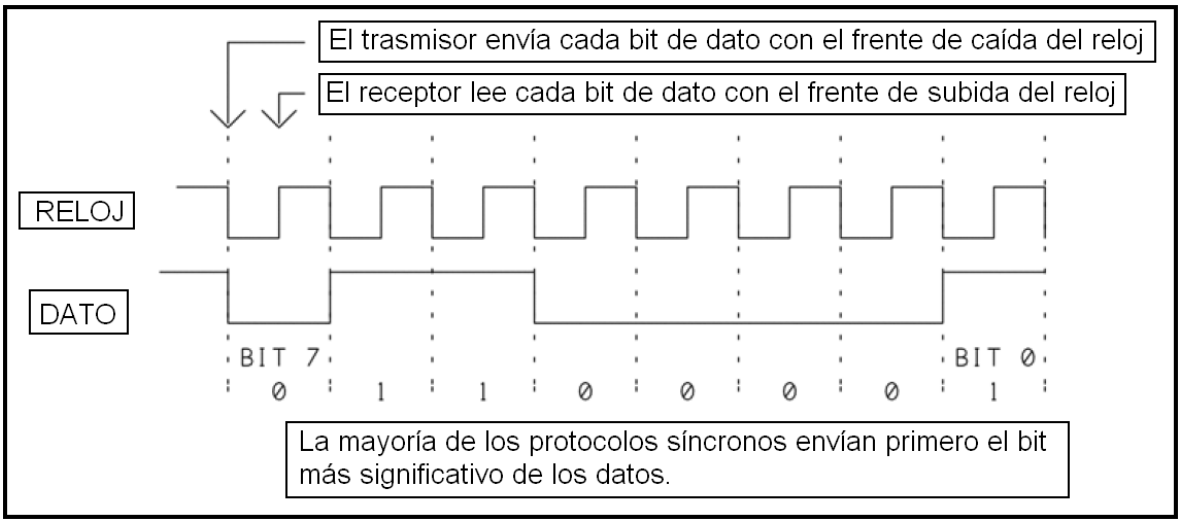


Figura 6.2. Trama síncrona.

Registros para controlar el USART.

Registro de Control y Estado de la Trasmisión (TXSTA).

En la **figura 6.3** se muestra la distribución de bits del Registro TXSTA.

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

Figura 3. Registro TXSTA.

- **Bit 7 (CSRC).** Bit de selección de la fuente de reloj para la trasmisión. Este bit es irrelevante en modo asincrónico puesto que la fuente de reloj siempre es interna (derivada del generador de baudios). En el Modo Sincrónico, si este bit es igual a uno el USART trabaja en Modo Master por lo que el reloj es generado internamente mediante el Generador de Baudios. El Modo Master supone que el puerto que ha sido considerado como tal pone la señal de reloj en el pin CK (pin RC6). Por otra parte para que el puerto trabaje en el Modo Sincrónico Esclavo este bit debe llevarse a cero. En este último caso, el reloj debe ser generado mediante una fuente externa. Este modo permite que el puerto pueda trabajar en modo sleep.
- **Bit 6 (TX9).** Habilita / inhabilita la trasmisión del 9no bit dentro de la palabra de datos.
- **Bit 5 (TXEN).** Habilita / inhabilita la trasmisión. Nota: en el modo sincrónico el bit TXEN y los bits SREN/CREN no tienen ninguna relación.
- **Bit 4 (SYNC).** Selecciona Modo Asincrónico o Sincrónico. Con uno se selecciona Modo Sincrónico.
- **Bit 3 (No se utiliza).**
- **Bit 2 (BRGH).** Este modo selecciona el modo de Tx en alta velocidad para el modo asincrónico. En el modo sincrónico este bit no se utiliza porque en trasmisión sincrónica solamente se trabaja en alta velocidad. Ver la sección: “Generador de Baudios”.
- **Bit 1 (TRMT).** Es una bandera que indica con un uno que el Registro de Desplazamiento del Trasmisor (TSR) está vacío y con un cero que está lleno.
- **Bit 0 (TX9D).** Es el noveno bit que se desea trasmitir en caso de habilitar la trasmisión en nueve bits.

Control de la Velocidad de Trasmisión (Generador de Baudios en el microcontrolador).

El Generador de Baudios de los PICs de la Familia Media es un generador autónomo de 8 bits que posee un timer asociado de 8 bits, cuyo período se controla mediante el registro SPBRG (Serial Port Baud Rate Generator). El Generador funciona en ambos modos de trabajo del USART. En el Modo Asincrónico el bit BRGH (bit 2 del registro TXSTA) permite seleccionar dos rangos de velocidades de trabajo: alta y baja velocidad. En el Modo Sincrónico este bit se ignora porque el USART trabaja siempre en modo de alta velocidad.

Cálculo de la Velocidad en baudios.

Para calcular la velocidad en baudios aplique las siguientes expresiones según el modo de trabajo:

Modo Asincrónico de baja velocidad (BRGH = 0, SYNC = 0):

$$\text{Baud Rate} = Fosc/(64(X+1))$$

Modo Asincrónico de alta velocidad (BRGH = 1, SYNC = 0):

$$\text{Baud Rate} = Fosc/(16(X+1))$$

Modo sincrónico (BRGH = irrelevante, SYNC = 1):

$$\text{Baud Rate} = Fosc/(4(X+1))$$

Nota: en las expresiones **X** representa el valor que debe cargarse en el registro **SPBRG** [0 ... 255].

Registro de Control y Estado de la Recepción. (dir 18h).

En la **figura 6.4** se muestra el esquema de bits para el registro RCSTA.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

Figura 4. Registro RCSTA.

El registro RCSTA permite realizar las siguientes funciones sobre el puerto serial:

- **bit 7 (SPEN). Habilitar el Puerto Serial.** Cuando se habilita el Puerto Serial los pines RC7/RX/DT y RC6/TX/CK se configuran como pines para la recepción y trasmisión en el modo asincrónico (RX y TX) y como pines de datos y reloj (DT y CK) en el modo sincrónico.
- **bit 6 (RX9). Seleccionar modo de recepción en 9 u 8 bits.** La palabra de datos, si se desea puede incluir un noveno bit (Tx9 en el trasmisor y Rx9 en el receptor). El noveno bit puede utilizarse para:
    - Generar el bit para el chequeo de paridad en el receptor. Es necesario aclarar que el USART, no soporta chequeo de paridad por HW; esto significa que el chequeo de paridad en cada byte que se recibe debe realizarse por S. W . Así mismo la trasmisión de un noveno bit como bit de paridad también debe implementarse por S.W.
    - Direccionar el USART dentro de un sistema en el que exista más de un puerto serial. El proceso de direccionado mediante el noveno bit es el siguiente:
        - Si se coloca el bit ADDEN (Address Detect Enable) en uno, entonces si se recibe un uno en el noveno bit, es porque lo que se recibió en los 8 bits restantes es el valor de la dirección de uno de los dispositivos de comunicación del sistema. El programador debe verificar si este valor de dirección coincide con el asignado al dispositivo para identificar si los datos que serán enviados deben ser recibidos por él.
- **Bit 5 (SREN).** Bit de habilitación de recepción única. Este bit sólo se utiliza en el Modo Sincrónico Máster. Es irrelevante en el Modo Asincrónico y en el Modo Sincrónico Esclavo. Cuando se comunica un puerto sincrónico máster con un esclavo la comunicación es “half duplex”, o sea solo se produce en el sentido del máster al esclavo.
- **Bit 4 (CREN).** Bit de habilitación / inhabilitación de la recepción. Un cero en este bit detiene el proceso de entrada de datos al registro de desplazamiento del receptor (RSR), en esencia inhibe la recepción.
- **Bit 3 (ADDEN).** Un uno en este bit provoca que el noveno bit recibido sea considerado como el bit de dirección.
- **Bit 2 (FERR).** Bit que señala que se ha producido un Error de Trama (“Framing Error”). Este bit se pone en uno cuando se detecta un bit de stop como cero. Este bit está doblemente bufereado al igual que los datos.
- **Bit 1 (OERR).** Bit que señala que se ha producido un “error de sobrescritura”. Este USART posee una FIFO de dos niveles, por lo que es posible almacenar solamente

dos bytes recibidos. El bit de overrun se pone en uno cuando la FIFO se ha llenado y se detecta el bit de STOP del tercer byte recibido. El tercer byte que se encuentra en el RSR se perderá, produciéndose el aviso a través del bit OERR.

- **Bit 0 (RX9D).** Es el noveno bit de la palabra de datos, que puede ser utilizado para el chequeo de paridad o como bit de dirección. Relea los apartados correspondientes al bit 6 y 3 de este registro.

Modo de operación Asincrónico.

Generalidades.

En este modo el USART utiliza un formato NRZ (Non Return to Zero). La trama de este formato posee un bit de START, 8 ó 9 bits de datos y un BIT de parada (comúnmente se utilizan 8 bits para datos). La frecuencia de trasmisión está determinada por el Generador de Baudios. El USART trasmite o recibe primero el bit menos significativo de la palabra de la trama. El trasmisor y el receptor del USART son funcionalmente independientes pero utilizan el mismo formato de datos y la misma velocidad en baudios. El chequeo del bit de paridad no lo soporta el HW pero puede implementarse por software utilizando como bit de paridad el 9 bit de la palabra de datos.

El Modo Asincrónico se selecciona limpiando el bit SYNC del registro TXSTA.

Funcionamiento del Trasmisor Asincrónico.

El elemento principal del trasmisor es el registro de desplazamiento TSR (Transmitter Serial Shift Register). Este registro es el encargado de serializar el dato a trasmitir. El dato a trasmitir se carga en el buffer de trasmisión TXREG. El TXREG puede ser accesado por el usuario. Por otra parte el registro TSR no se carga hasta que no ha sido trasmitido el bit de STOP de la palabra anterior. Tan pronto como el bit de STOP de la palabra anterior se trasmite el TSR se carga con la nueva palabra a trasmitir si está disponible en el buffer TXREG. La transferencia del dato entre el TXREG y el TSR ocurre en un Ciclo de Máquina. Una vez concluida la transferencia se activa la bandera de interrupción del trasmisor (TXIF, PIR <4>), ocurriendo la interrupción si ha sido habilitada seteando el bit 4 del registro PIE1 (TXIE). La bandera TXIF va a uno independientemente de si la interrupción ha sido habilitada o no y no puede limpiarse por S. W. La bandera TXIF irá a cero únicamente cuando se cargue un nuevo dato en el buffer de trasmisión (TXREG).

El estado del registro de desplazamiento TSR (vacío o lleno), puede averiguarse mediante el bit TRMT. El TMRT sólo puede leerse mediante un proceso de encuesta. No existe ninguna interrupción ligada a este bit.

Resumen del Proceso de Trasmisión Asincrónica.

1. Cargar el registro SPBRG con el valor adecuado para generar la velocidad que se desea. Si se desea modo de alta velocidad, poner en uno el bit BRGH.
2. Habilitar el **Modo Asincrónico** limpiando el bit SYNC y seteando el bit SPEN para habilitar el Puerto Serial.
3. Si se desea interrupción después de trasmitir una palabra, poner en uno el bit TXIE.
4. Si desea trasmitir el noveno bit, entonces setear el bit TX9.
5. Habilitar la trasmisión seteando el bit TXEN. Esta acción llevará a uno el bit TXIF.
6. Si seleccionó trasmisión de noveno bit, cargue el valor en el bit TX9D.
7. Cargar el dato a trasmitir en el registro TXREG. Este evento inicia la trasmisión de la palabra.

Notas importantes:

- El registro TSR no está mapeado en ninguna dirección de la RAM por lo tanto no está disponible para el usuario.
- Cuando el bit TXEN se pone en uno, la bandera TXIF también va a 1 ya que el buffer TX (registro TXREG), todavía no ha sido cargado con el dato nuevo. Una vez que se cargue el registro con el nuevo dato, la bandera irá a cero.
- La trasmisión se habilita poniendo en 1 el bit TXEN (bit 5 del reg. TXSTA). La trasmisión real NO ocurrirá hasta que se cargue el registro TXREG con el dato a TX y el Generador de Baudios (BRG) produzca el pulso de desplazamiento.
- Otra forma de iniciar la TX es cargando primero el registro TXREG y después seteando el bit TXEN.
- Si durante la TX Ud. limpia el bit TXEN, la trasmisión será abortada, el Módulo TX sufrirá un RESET y el pin TX/CK pasará a estado de alta impedancia.
- Para seleccionar un modo de TX de 9 bits (raras veces), UD. debe poner en 1 el bit TX9 (bit 6 del registro TXSTA) y el noveno bit a trasmitir debe escribirse en el bit TX9D (bit 0 del registro TXSTA). El noveno bit debe escribirse antes de cargar en el registro TXREG los 8 restantes de la palabra de datos, porque una carga del registro TXREG puede resultar en una transferencia inmediata de datos al registro TSR. En tal caso pudiera suceder que el 9no bit trasmitido contenga un dato falso.

Funcionamiento del Receptor Asincrónico.

El dato se recibe en el pin RC7/RX/DT. El Modo de Recepción Asincrónica se selecciona seteando el bit CREN (RCSTA <4>).

De manera similar al TX el receptor posee un registro de desplazamiento serie para la recepción (RSR). Después del muestreo que el receptor realiza al pin RC7/RX/DT en busca del bit de STOP, el dato recibido en el registro RSR se transfiere al Buffer de Recepción (registro RCREG). Cuando la transferencia se completa la bandera de interrupción RCIF se pone en 1 y la interrupción del receptor ocurrirá si ha sido habilitada. Para habilitar la recepción se debe setear el bit RCIE (PIE <5>). La bandera RCIF es un bit de solo lectura que el programador no puede limpiar. Solo el hardware limpia esta bandera. La bandera RCIF se limpia siempre que se lea el registro RCREG y este esté vacío.

Buffer de recepción.

Es una FIFO (First In First Out) de dos niveles. Lo que significa que se pueden almacenar hasta dos bytes recibidos. En caso de que se acumulen dos bytes, para extraerlos se debe leer dos veces el registro RCREG. Si no se leen los dos bytes almacenados y se termina de detectar el bit de STOP del tercero se perderá el último byte recibido (que se encuentra aún en el RSR) y se señalizará error de sobrescritura mediante el bit OERR (RCSTA <1>).

Tratamiento de errores en la recepción.

"Overrun error" (error de sobrescritura).

El registro RCREG es un registro doblemente bufereado, o sea que se pueden almacenar 2 bytes recibidos mientras se desplaza el tercero en el registro de desplazamiento RSR. Si cuando se detecta el **bit de parada** del tercer byte el registro RCREG está aún lleno se pondrá en uno el bit OERR ("overrun error"), bit 1 del registro RCSTA, como señal de aviso y la palabra que se estaba procesando en el registro RSR se pierde. Para volver a la recepción normal se debe leer el registro RCREG para sacar los dos datos ya almacenados y se debe limpiar el bit de error por SW. Para limpiar el bit OERR se puede poner el bit CREN en 0 y después en 1. Es esencial limpiar el bit OERR mientras este bit permanezca en 1, se inhiben las transferencias de datos entre el registro RSR y el buffer de recepción (registro RCREG).

"Framing error" (error de trama).

Este error se produce cuando un bit de STOP se detecta como un nivel bajo. Este error se avisa con el bit FERR ("framing error"), bit 2 del registro RCSTA.

Importante: cada vez que se lea el buffer de Rx (RCREG) el bit FERR se cargará con un nuevo valor, luego para conocer si hubo error en el dato recibido, es necesario primero leer el registro RCSTA, antes de leer el RCREG.

Proceso para realizar la recepción asincrónica con chequeo de errores.

1. Cargar el registro SPBRG con el valor adecuado para generar la velocidad que se desea. Si se desea modo de alta velocidad, poner en uno el bit BRGH.
2. Habilite el puerto serie asincrónico poniendo el bit SYNC = 0 y el bit SPEN =1.
3. Si se desea ser interrumpido cada vez que se reciba un dato ponga en uno los bits RCIE (PIE1 <5>), GIE y PEIE del INTCON.
4. Si desea modo de recepción en 9 bits ponga en uno el bit RX9.
5. Habilite la recepción poniendo en uno el BIT CREN
6. La bandera RCIF irá a uno cuando la recepción se complete y se generará una interrupción si esta fue habilitada.
7. Primero lea el registro **RCSTA** para chequear si hubo errores y/o para obtener el 9no BIT de la trama si fue habilitado el modo de recepción en 9 bits.
8. Lea el dato recibido en el registro **RCREG**.
9. Si ocurrió algún error, limpie el bit CREN y póngalo en uno después y espere nuevamente por la interrupción o chequee por encuesta la bandera RCIF.

Proceso para realizar la recepción asincrónica en nueve bits con detección de dirección.

1. Cargar el registro SPBRG con el valor adecuado para generar la velocidad que se desea. Si se desea modo de alta velocidad, poner en uno el bit BRGH.
2. Habilite el puerto serie asincrónico poniendo el bit SYNC = 0 y el bit SPEN =1.
3. Si se desea ser interrumpido cada vez que se reciba un dato ponga en uno los bits RCIE (PIE1 <5>), GIE y PEIE del INTCON.
4. Ponga en uno el bit RX9, para habilitar la recepción en 9 bits.
5. Ponga en uno el bit ADDEN, para habilitar la detección de dirección.
6. Habilite la recepción poniendo en uno el bit CREN
7. La bandera RCIF irá a uno cuando la recepción se complete y se generará una interrupción si esta fue habilitada.
8. Primero lea el registro **RCSTA** para chequear si hubo errores y para obtener el 9no BIT de la trama.

9. Lea el dato recibido en el registro **RCREG** para determinar si el puerto está siendo direccionado.
10. Si ocurrió algún error, limpie el bit CREN.
11. Si el puerto está siendo direccionado, limpie el bit ADDEN para que puedan leerse sucesivos bytes de datos o dirección.

Tema VII: Tecnología Microchip: Familia 18FXXX

Sumario:

1. Arquitectura Interna de la Familia 18FXXX.
2. Programación de los Microcontroladores de la familia 18FXXX.
3. Periféricos Integrados a la Familia 18FXXX. Enfoque comparativo con la serie 16F87X. Introducción al PIC18F4550. Módulo USB integrado a los micro controladores 18F2/45X0.

Arquitectura Interna de la Familia 18FXXX.

Especificaciones Generales (Serie 18FXX2).

La serie 18FXX2 garantiza plena compatibilidad de pines con la serie 87X, aunque engloba microcontroladores de 8 bits mucho más poderosos que la serie 87X. Las especificaciones generales más relevantes se resumen a continuación:

Memorias.

- Memoria de Programas[16] con capacidad de almacenamiento de hasta 32K.
- Memoria de Datos con capacidad de almacenamiento de hasta 1,5K.

Frecuencia de Operación.

- DC hasta 40MHz con oscilador estándar.
- 4MHz hasta 10 MHz con P.L.L activo.

Instrucciones.

- Instrucciones codificadas en 16 bits, a diferencia de la serie 87X que codifica en 14 bits.
- **76 Instrucciones**, a diferencia de la serie 87X que **maneja 33**.
- Consta además de un multiplicador por H.W de 8 bits en un solo ciclo de máquina.

Interrupciones.

- A diferencia de la serie 87X la serie 18FXX2 maneja niveles de prioridad para las interrupciones.
- A diferencia de la serie 87X la serie 18FXX2 consta de **tres pines de**

Interrupción externa.

- A diferencia de la serie 87X la serie 18FXX2 **maneja dos vectores de interrupción uno para las interrupciones de alta prioridad y otro para las de baja prioridad** ubicados en las direcciones 000008 h y 0000018 h de la Memoria de Programas respectivamente.

[16] En general la Familia 18FXXX puede direccionar hasta 2Mb de Memoria de Programas interna.

Periféricos incluidos:

Timers.

- La serie 18XX2 consta de **4 timers:** timer 0, 1, 2 y 3 (la serie 87X solo maneja 3 timers). El timer0 es configurable como timer o contador de 8 ó 16 bits, a diferencia del timer0 incluido en la serie 87X (contador temporizador de 8 bits).
- Los timers 1 y 2 conservan la misma estructura, salvo algunos detalles (consulte sección timers[17]).
- El timer 32 es un contador / temporizador de 16 bits con prescaler y relacionado con el Módulo CCP en los modos campara o captura.

Módulo(s) CCP.

- La serie 18XX2 consta de dos módulos CCP que conservan básicamente la misma estructura de los incluidos en la serie 87X.

Hardware para comunicaciones.

- Un puerto **MSSP** (Master Synchronous Serial Port) configurable en dos modos:
  - 3 – wire SPI®.
  - I2C Master y Esclavo[18].
- Un puerto **USART** direccionable (RS 232 / RS485) que básicamente permanece invariante.

Hardware Analógico.

- **Conversor A/D de hasta 8 canales** multiplexados con **resolución de 10 bits**, similar al que se incluye en la serie 87X.
- **PLVD[19] (Programmable Low Voltage Detection).** Este módulo no aparece en la serie 87X. El módulo PLDV es un comparador de ventana que permite definir un valor de la fuente de alimentación del sistema por debajo del cual se puede generar un "shut down" por software del sistema.
- **BOR (Brown Out Reset Programable),** similar al que se incluye en la serie 87X.

---

[17] Sección "Timers" de este documento ó sección 11 del DataSheet para ampliar información.

[18] 3 3 – wire SPI® e I2C Master y Esclavo son dos protocolos de comunicación sincrónica.

[19] 4 Consulte sección 18: "PLDV Module" en el DataSheet o el tópico correspondiente en este documento.

Otras posibilidades.

- La serie 18FXX2 al igual que la serie 87X incluye un Power-up Timer (PWRT), Power On Reset (POR) y un Start-up Timer (OST).
- Modo Sleep.
- Protección de Código programable.
- 8 modos para el oscilador del sistema (4 modos adicionales a los que posee la serie 87X).
- ICSP® mediante dos pines.
- ICD mediante dos pines.

Oscilador Externo.

En la serie 18FXXXX el Oscilador Externo puede funcionar en **8 modos diferentes**:

1. LP Low Power Crystal.
2. XT Crystal/Resonator.
3. HS High Speed Crystal/Resonator.
4. HS + PLL High Speed Crystal/Resonator with P.L.L enabled.
5. RC External Resistor Capacitor.
6. RCIO External Resistor Capacitor with I/O pin enabled.
7. EC External Clock.
8. ECIO External Clock with I/O pin enabled.

Modos LP, XT, HS y HS + PLL.

Los modos LP, XT, HS y RC se heredan de la serie 87X. **El modo HS + PLL es especial.** En los modos **LP, XT, HS y HS + PLL** se debe conectar un cristal o resonador cerámico a los pines OSC1 y OSC2 (**figura 2.1**). La serie 18FXX2 requiere de un cristal con corte paralelo[20].

[20] Los cristales pueden cortarse en modo longitudinal ("strip") o paralelo ("parallel").

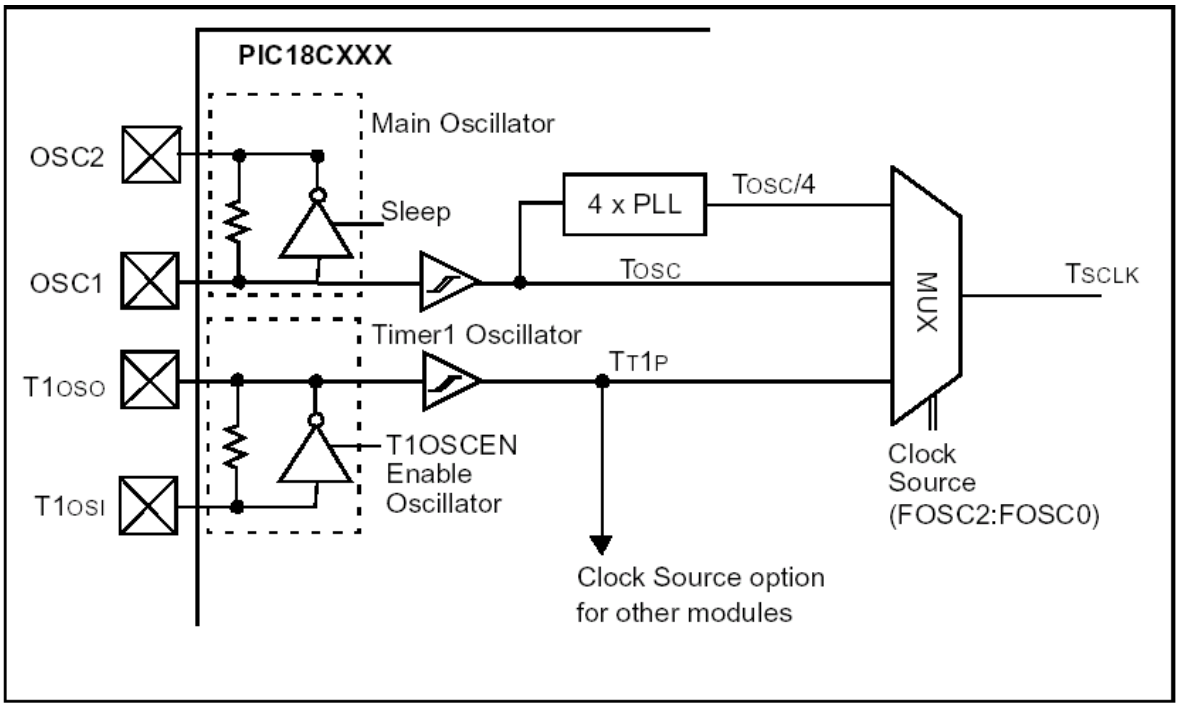


Figura 7.1. Lógica y pines de conexión para el oscilador externo (serie 18FXXXX). Fuente: PIC18FXX2 Data Sheet; High Performance Enhanced FLASH Microcontrollers with 10-Bit A/D.

En la **tabla 7.1** se muestran los valores de condensadores[21] para los modos XT y HS para resonadores cerámicos:

Ranges Tested:			
Mode	**Freq**	**C1**	**C2**
XT	455 kHz 2.0 MHz 4.0 MHz	68 - 100 pF 15 - 68 pF 15 - 68 pF	68 - 100 pF 15 - 68 pF 15 - 68 pF
HS	8.0 MHz 16.0 MHz	10 - 68 pF 10 - 22 pF	10 - 68 pF 10 - 22 pF
These values are for design guidance only. See notes following this table.			
Resonators Used:			
455 kHz	Panasonic EFO-A455K04B	± 0.3%	
2.0 MHz	Murata Erie CSA2.00MG	± 0.5%	
4.0 MHz	Murata Erie CSA4.00MG	± 0.5%	
8.0 MHz	Murata Erie CSA8.00MT	± 0.5%	
16.0 MHz	Murata Erie CSA16.00MX	± 0.5%	
All resonators used did not have built-in capacitors.			

[21] Estos valores son solo una referencia.

Tabla2.1. Valores de capacitancia recomendados para la serie 18FXX2, en caso de utilizar resonadores cerámicos. Fuente: PIC18FXX2 Data Sheet; High Performance Enhanced FLASH Microcontrollers with 10-Bit A/D.

En la **tabla 7.2** se muestran los valores de condensadores[22] para los modos LP, XT y HS para cristales:

Ranges Tested:			
Mode	**Freq**	**C1**	**C2**
LP	32.0 kHz	33 pF	33 pF
	200 kHz	15 pF	15 pF
XT	200 kHz	22-68 pF	22-68 pF
	1.0 MHz	15 pF	15 pF
	4.0 MHz	15 pF	15 pF
HS	4.0 MHz	15 pF	15 pF
	8.0 MHz	15-33 pF	15-33 pF
	20.0 MHz	15-33 pF	15-33 pF
	25.0 MHz	15-33 pF	15-33 pF
These values are for design guidance only. See notes following this table.			
Crystals Used			
32.0 kHz	Epson C-001R32.768K-A		± 20 PPM
200 kHz	STD XTL 200.000KHz		± 20 PPM
1.0 MHz	ECS ECS-10-13-1		± 50 PPM
4.0 MHz	ECS ECS-40-20-1		± 50 PPM
8.0 MHz	Epson CA-301 8.000M-C		± 30 PPM
20.0 MHz	Epson CA-301 20.000M-C		± 30 PPM

Tabla 7.2. Valores de capacitancia recomendados para la serie 18FXX2, en caso de utilizar cristales. Fuente: PIC18FXX2 Data Sheet; High Performance Enhanced FLASH Microcontrollers with 10-Bit A/D.

Modo especial HS + PLL (Phase Locked Loop).

Este modo incluye un PLL[23] (**figura 7.2**) que permite multiplicar la frecuencia del cristal conectado por 4. Por ejemplo, para un cristal de 10 MHz externo el chip trabajará a 40MHz internos. El modo HS + PLL es útil en proyectos donde se quiere reducir la emisión de interferencias. El PLL sólo se puede habilitar si se trabaja en modo HS.

[22] Estos valores son solo una referencia.

[23] Consulte sobre PLLs en material adjunto a esta conferencia.

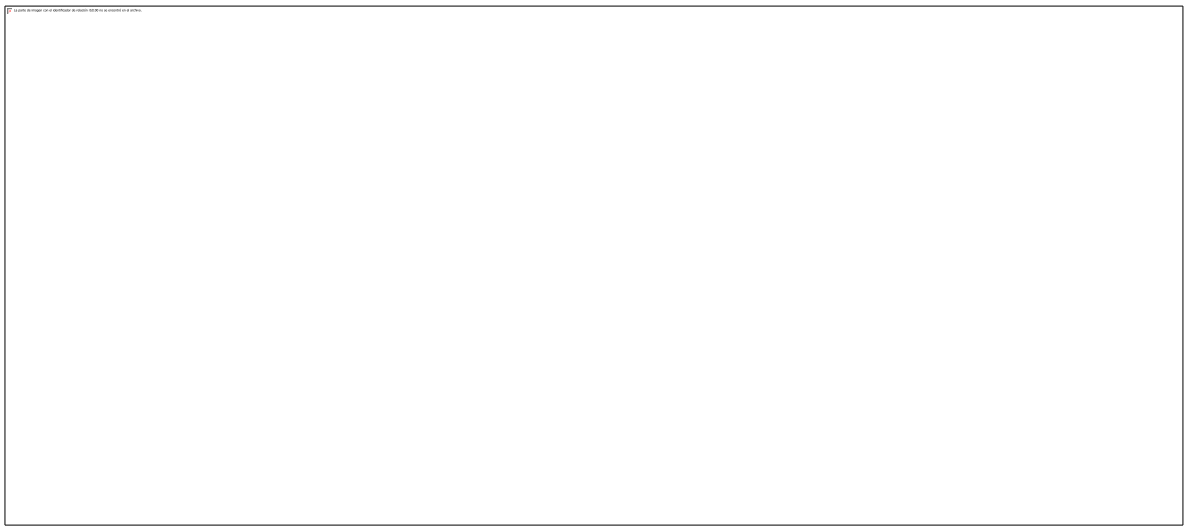


Figura 7.2. Lógica y pines de conexión para el oscilador externo en Modo PLL. Fuente: PIC18FXX2 Data Sheet; High Performance Enhanced FLASH Microcontrollers with 10-Bit A/D.

Modos RC.

Trabajan con un resistor y capacitor externo. El Modo RC clásico se hereda de la serie 87X. El Modo RCIO (**figura 7.3**) es una extensión del modo RC que aparece en la serie 18FXXXX. En el Modo RCIO el pin OSC2 se convierte en un pin de E/S de propósito general; al igual que el modo ECIO que se trata en la sección siguiente.

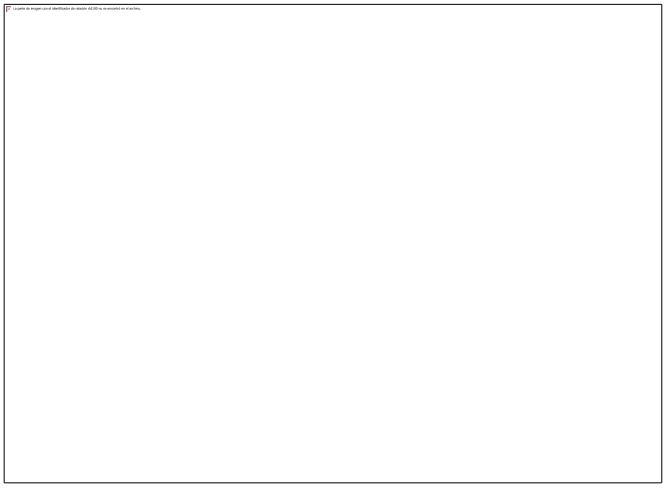


Figura 7.3. Lógica y pines de conexión para el oscilador externo en Modo RC/RCIO. Fuente: PIC18FXX2 Data Sheet; High Performance Enhanced FLASH Microcontrollers with 10-Bit A/D.

Modos EC / ECIO.

Los modos EC y ECIO requieren de la conexión externa de un oscilador al pin OSC1 (**figuras 7.4 y 7.5**). Si se selecciona modo **EC (figura 7.4)**; en el pin OSC2 se devuelve la señal conectada a OSC1 dividida por 4. La señal de OSC2 puede utilizarse para realizar pruebas (calibración) o para sincronizar otra lógica. Por otra parte si se selecciona **ECIO (figura 7.5)**, el pin OSC2 se convierte en un pin de E/S de propósito general; concretamente pasa a ser el pin bit **ra6 del puerto A**.

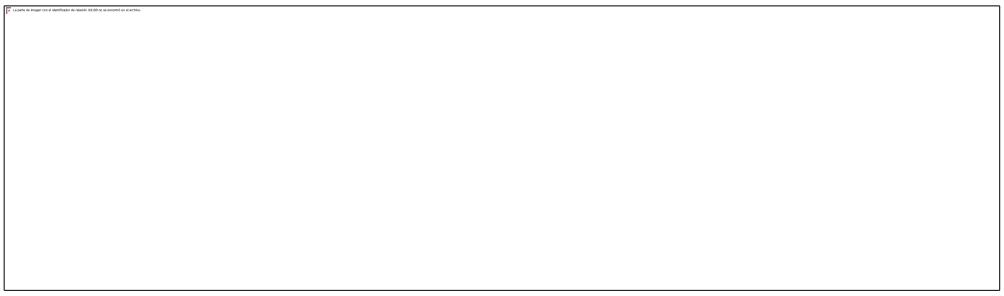


Figura 7.4. Lógica y pines de conexión para el oscilador externo en Modo EC. Fuente: PIC18FXX2 Data Sheet; High Performance Enhanced FLASH Microcontrollers with 10-Bit A/D.

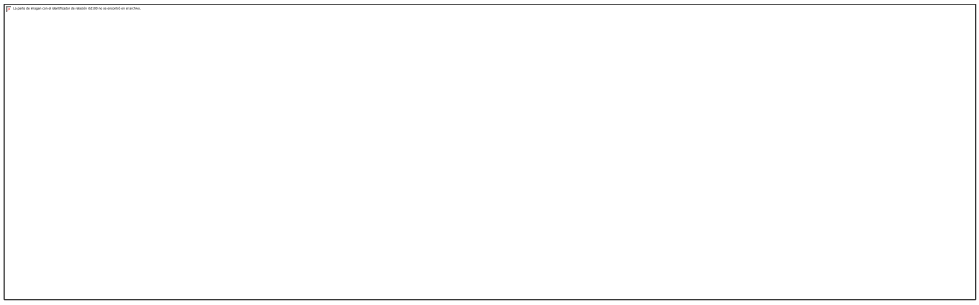


Figura 7.5. Lógica y pines de conexión para el oscilador externo en Modo ECIO. Fuente: PIC18FXX2 Data Sheet; High Performance Enhanced FLASH Microcontrollers with 10-Bit A/D.

“Resets”.

La **familia 18FXXX** puede responder a **8 tipos de RESET:**

1. Power On Reset.
2. Reset por MCLR durante la operación normal.
3. Reset por MCLR en modo Sleep.
4. Reset por el “Watch Dog” durante la operación normal.
5. Reset Programable por Brown Out.
6. Reset cuando se ejecuta la instrucción RESET.
7. Reset por “overflow” de la pila.
8. Reset por “underflow” de la pila.

Los tipos de RESET 1,2,3,4 y 5 se heredan de la serie 87X; los tres restantes (6,7,8) son propios de la familia 18FXXX. Se describirán los tipos de RESET más relevantes. Los PICs de la familia 18FXXX poseen un filtro conectado al terminal de RESET externo que detecta y elimina pequeños pulsos de rebote; aunque se aconseja seguir eliminando rebote externo mediante circuito RC.

Power On Reset (POR) y el “PWRT”.

Consiste en un pulso de RESET que se genera internamente cuando se detecta que la fuente sube. El POR permite conectar directamente el MCLR a VDD a través de una resistencia de pull – up. Si la fuente sube con lentitud entonces se aconseja utilizar el clásico circuito RC. El POR trabaja en conjunto con el Power Up Timer (PWRT). El PWRT es un timer que mantiene un RESET durante 70 ms hasta que la fuente se estabilice. Es aconsejable utilizar el POR y el PWRT a la vez.

Adicionalmente se incluyen un Oscillator Start – Up Timer (OST) y una demora para el PLL (en modo HS + PLL). El OST añade una demora adicional de 1024 períodos del oscilador (conectado a OSC1) después que termina la demora del PWRT para garantizar que el oscilador se encuentre en estado estable. El OST se activa solamente para LP, XT y HS si se había habilitado el PWRT. La demora para el PLL (PLL “Lock Time Out”), dura típicamente 2ms y comienza después que termina el OST. Durante 2 ms se bloquea el PLL hasta que la frecuencia del oscilador principal se estabilice.

Brown Out RESET (BOR).

El BOR es un RESET que se produce cuando la fuente de alimentación cae por debajo de determinado valor. El chip se mantendrá en RESET hasta que VDD suba nuevamente. Si se había habilitado el PWRT, este se activará después que el circuito de Brown Out deje de actuar; lo que sucede cuando VDD supera un valor umbral.

Resumen de secuencia de demoras que se pueden activar después del encendido del microcontrolador.

Después que el chip se energiza se pueden activar los siguientes eventos de demora si se encuentran habilitados a través de la Palabra de Configuración del Microcontrolador durante su programación física:

Primero: se inicia el PoWeR up Timer (PWRT) después que termina el pulso de Power On. El tiempo de PWRT normalmente dura 72 ms.

Segundo: después que termina la demora del PWRT se activa la demora del OST [Oscillator Start-Up Time out] que dura normalmente 1024 períodos del oscilador externo. Si se habilita el modo PLL entonces debe añadirse una demora de 2ms.

Programación de los Microcontroladores de la familia 18FXXX.

Arquitectura de la Memoria de Programas.

Introducción.

Al igual que en la serie 87X en la familia 18FXXX existen tres bloques de memoria:

- Memoria de Programas.
- Memoria de Datos.
- Memoria EEPROM de datos.

Estos dispositivos también soportan arquitectura Harvard lo que significa que la Memoria de Programas y de Datos se comunican con Buses separados con el CPU.

Organización de la Memoria de Programas.

Especificaciones Generales de la Memoria de Programas:

- A diferencia de la serie 87X (87XA) la Memoria de Programas en la familia 18FXXX pudiera extenderse hasta 2 Mb.
- El Contador de Programas es de 21 bits, para poder direccionar esta cantidad de de Memoria de Programas (2 Mbytes). El Contador de Programas en la serie 87X en cambio es de 13 bits. Como el PC en la familia 18FXXXX es de 21 bits puede ser accesado a través de 3 Registros de Propósito Específico incluidos en la RAM: PCLATU ("PC Latch Upper"), PCLATH ("PC Latch High") y PCL. Los tres registros antes mencionados almacenan la parte Superior, Alta y Baja del PC respectivamente.
- La capacidades de memoria de los dispositivos de la serie 18FXX2 son las siguientes:

- El vector de RESET se ubica en la dirección 0000 h, al igual que en la serie 87X.
- Los vectores de INTERRUPCIÓN están situados en las direcciones 0008h (interrupciones de alta prioridad) y la 0018h (interrupciones de baja prioridad). La serie XX2 maneja dos vectores a diferencia de la 87X, que maneja uno.
- El **registro acumulador (Working Register [WREG]),** al igual que en los microcontroladores de la serie 87X ocupa un espacio especial, independiente del resto de las memorias (Datos, Programas y EEPROM).

En la **figura 7.6** se muestra el mapa de la Memoria de Programas para los PICs integrantes de la serie 18FXX2.

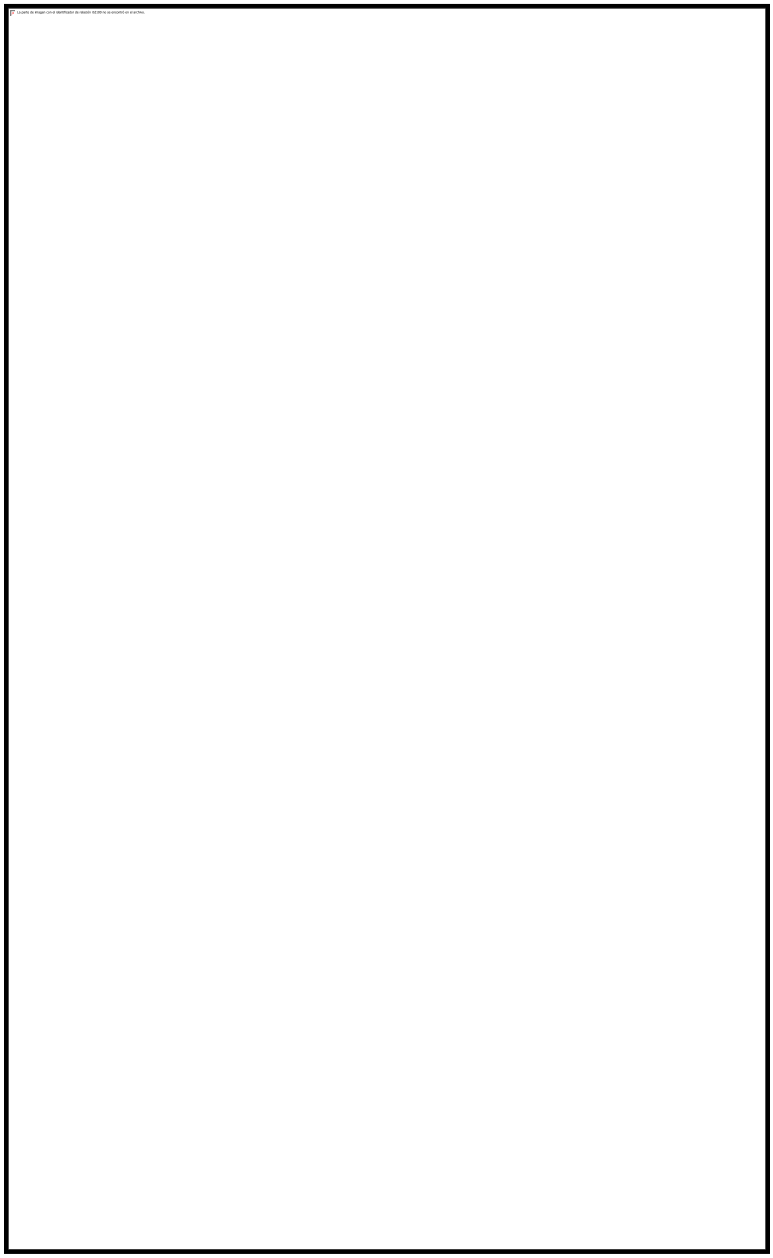


Figura 7.6. Esquema general de la Memoria de Programas para los Microcontroladores de la Familia 18FXXXX. Fuente: PIC18FXX2 Data Sheet; High Performance Enhanced FLASH Microcontrollers with 10-Bit A/D.

Saltos dentro de la Memoria de Programas.

Los saltos dentro de la MP se generan por acciones desde el programa que necesariamente modifican el contenido del PC. Para modificar el contenido del PC se pueden utilizar las siguientes metodologías:

1. **“Computed GOTO”.** Esta metodología, también utilizada en la serie 16F87X consiste en adicionar un valor al registro PCL mediante la instrucción “addwf PCL”. Los saltos “Computed GOTO” tienen un alcance limitado a 256 direcciones, porque sólo afectan al PCL.
2. **Mediante “CALLS” y “GOTO”.** Los GOTO funcionan de manera similar a la serie 87X. Existen dos tipos de “CALL”:
    a. **Los que se ejecutan mediante la instrucción “CALL” (Subroutine Call).** El CALL en este caso permite realizar saltos dentro de todo el espacio de memoria posible (2MB).

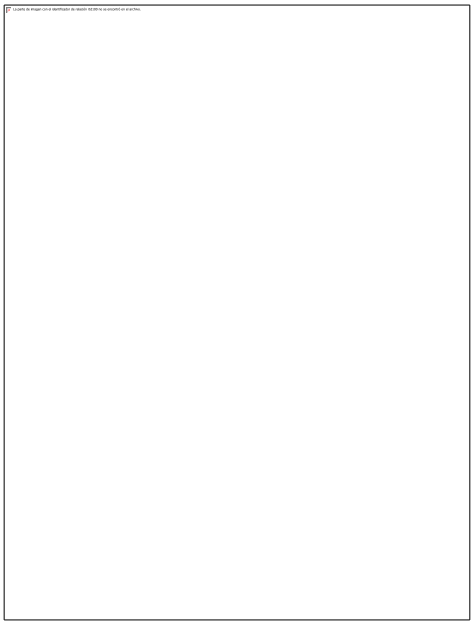


Figura 7.7. Instrucción CALL en la Familia 18FXXXX. . Fuente: PIC18FXX2 Data Sheet; High Performance Enhanced FLASH Microcontrollers with 10-Bit A/D (sección 20).

Después de la instrucción CALL suceden los siguientes eventos:

- Se suma 4 al PC y se guarda en la PILA.
- Se carga el PC con la nueva dirección de salto (“k”); que generalmente es marcada por un “label”.
- Si el operando “s” es = 1 los registros W y STATUS se guardan en los respectivos registros “SHADOW”: WS y STATUSS. Los registros “Shadow” son una mejora que incluyen los 18FXXXX para respaldar el STATUS y W después de un salto.

b. Los “relativos”, que se ejecutan mediante la instrucción “RCALL”. La instrucción RCALL **(figura 7.8)** funciona de manera idéntica en cambio la

RCALL permite realizar saltos con una longitud de hasta 1024 direcciones (similar al CALL tradicional en la serie 87X).

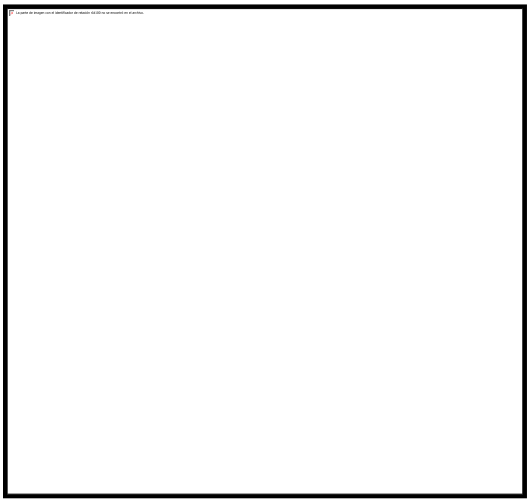


Figura 7.8. Instrucción RCALL en la Familia 18FXXXX. . Fuente: PIC18FXX2 Data Sheet; High Performance Enhanced FLASH Microcontrollers with 10-Bit A/D (sección 20).

3. **Mediante instrucciones de rama (Branch Instructions).** Son instrucciones que modifican el valor del PC si se cumple determinada condición aritmética. En la serie 87X se describen instrucciones similares tales como: “BTFSS f,b / BTFSC f,b ; INCFSZ f,d / DECFSZ f,d”. Además de estas instrucciones el Set de la Familia 18FXXXX ofrece las instrucciones tipo “B” (Branch); que son siete en total. Considérese el siguiente ejemplo como representativo de la familia “B”.

Ejemplo de Instrucción de Rama (nueva en la familia 18FXXXX):

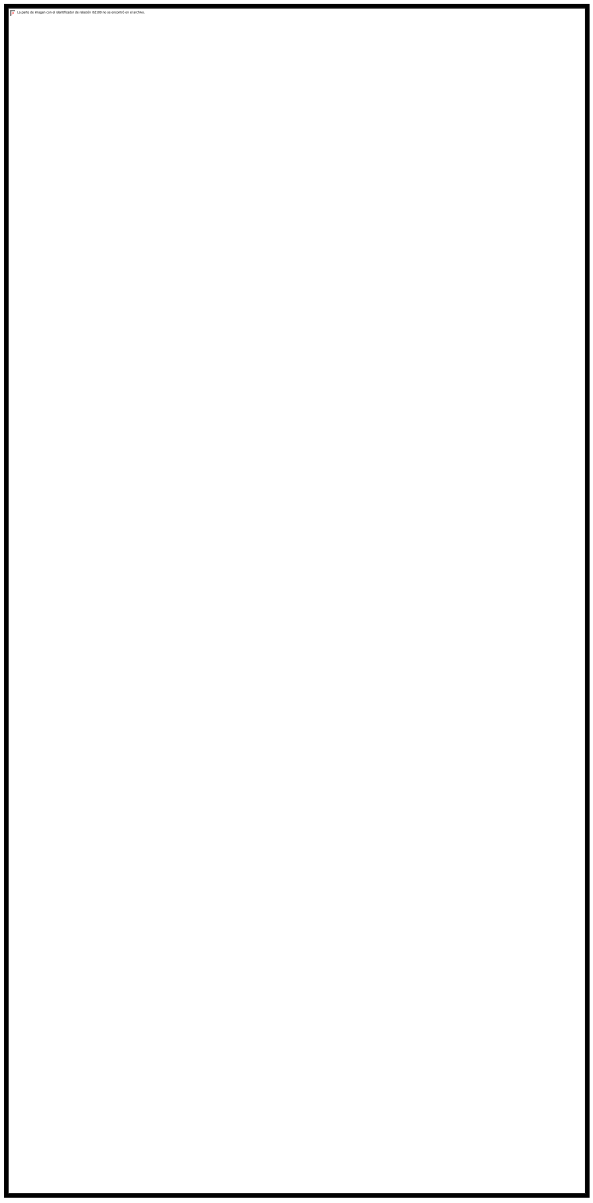


Figura 7.9. Instrucción de Rama (BC) en la Familia 18FXXXX. . Fuente: PIC18FXX2 Data Sheet; High Performance Enhanced FLASH Microcontrollers with 10-Bit A/D (sección 20).

Funcionamiento: Salta la siguiente instrucción solamente **(PC = PC +2)** si **carry = 0** en la operación aritmética anterior. Salta la siguiente instrucción y suma al PC el valor "n" multiplicado por dos (2*n) **[PC = PC + 2 + 2*n]** si **carry = 1** en la operación aritmética anterior.

Estructura del “STACK” (Pila) en la Familia 18XXX.

El diseño de la Pila en la familia 18XXX es muy revolucionario comparado con el de los microcontroladores de la serie 87X:

La serie 87X posee un stack simple de 8 niveles X 13 bits que no puede manipular el programador, lo que significa que no se pueden respaldar registros en ella y tampoco es posible conocer su estatus (si se produjo un “overflow” o “underflow”). Por otra parte el “stack” en los micros de la familia18FXX2 incluye las siguientes ventajas a considerar:

- El “stack” opera como una Memoria RAM de 31 niveles x 21 bits que permiten anidar hasta 31 subrutinas o 31 interrupciones. En este sentido supera considerablemente a la pila de la serie 87X.
- El “stack” posee un Registro Puntero y Control de la Pila (STKPTR). El STKPTR es un Registro de Propósito Específico de 8 bits tipo r/w de la RAM, que se inicia en 00000000b después de cualquiera de las 8 fuentes de RESET descritas anteriormente y sirve para direccionar la pila y verificar su desbordamiento (“over” y “under” “flow”). En la **figura 7.10** se muestra el registro STKPTR.

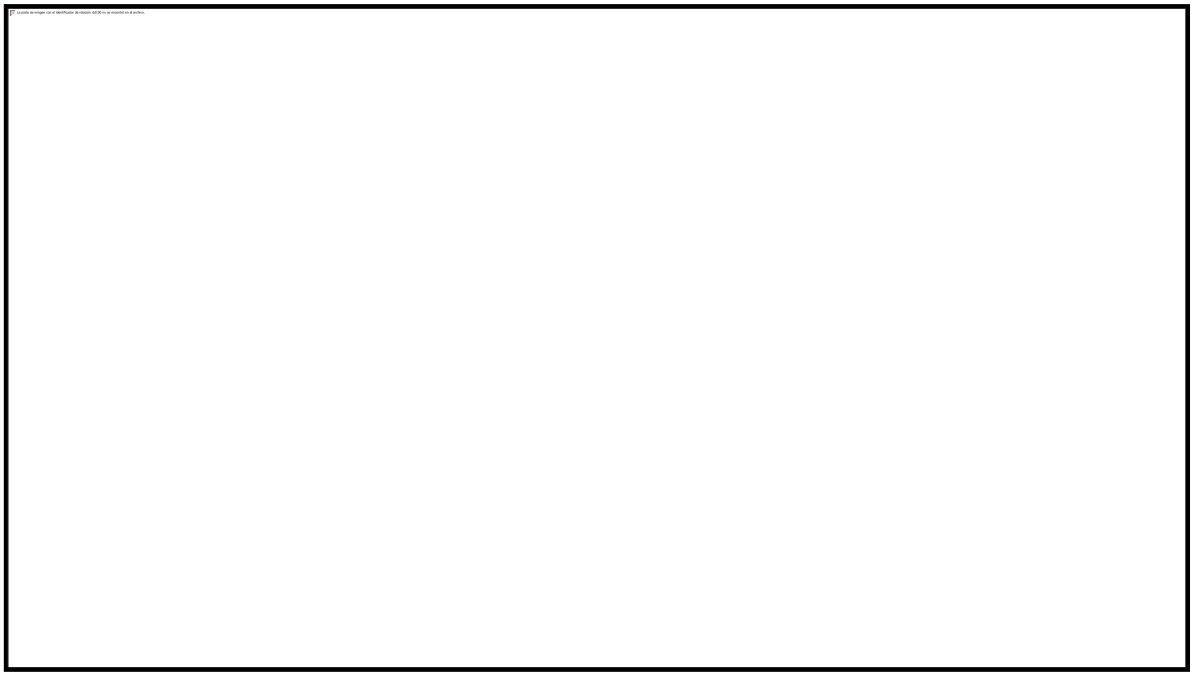


Figura 7.10. Registro Puntero de la PILA. Fuente: PIC18FXX2 Data Sheet; High Performance Enhanced FLASH Microcontrollers with 10-Bit A/D (sección 4).

- Aspectos asociados al **“Stack Pointer: STKPTR”:**
    - El contenido del “STKPTR” normalmente se incrementa anteuna instrucción “CALL” para escribir en la pila el contenidodel PC (Program Counter).
    - El contenido del “STKPTR” normalmente se decrementa ante una instrucción “RETURN” para leer desde la pila el contenido del PC (Program Counter) de la dirección de retorno.
    - La dirección dentro de la pila se puede cambiar con los bits: SP4 … SP0.
    - El “overflow” (desbordamiento por exceso) o “underflow” (desbordamiento por defecto) se puede chequear encuestando los bits TKOVF y STKUNF respectivamente.
    - El “techo” de la pila se puede leer o escribir desde el programa.

Recuerde que en la serie 87X el programador no podía realizar ninguna operación sobre la pila. Existen tres registros que permiten realizar operaciones de lectura escritura sobre la pila: TOSU, TOSH y TOSL .Donde U, H y L significan Upper, High and Low; respectivamente. Se necesita escribir y leer la PILA para implementar (si es necesario) un respaldo de registros en ella; similar a la metodología que se aconsejaba seguir en la Subrutina de Servicio de Interrupción en los Microcontroladores de la Serie 87X. Durante la lectura o escritura de la PILA se recomienda inhabilitar las interrupciones para evitar accesos inesperados a la PILA. Luego de lograr el respaldo o la lectura habilite nuevamente las interrupciones.

Respaldos en la PILA.

En la PILA de la familia 18FXXXX, a diferencia de la PILA de la serie 87X; se pueden guardar los siguientes tipos de información:

1. **Contenidos del Contador de Programas.** La grabación y recuperación del PC son automáticas.
2. **Respaldos de Registros de Propósito Específico o Generales**; utilizando el mecanismo de direccionado de la PILA mediante el **STKPRT** y los registros **TOSU, TOSH y TOSL.**
3. **Opción de respaldo “rápido”** de los registros **WREG, BSR y STATUS**; ante la ocurrencia de una interrupción.

Manejo de Tablas en la Memoria de Programas (serie 18FXXXX).

Existen dos métodos para generar y direccionar tablas en la Memoria de Programas de los UC de la Familia 18FXXXX:

"Computed GOTO". Es el método clásico normalmente utilizado en la serie 87X. Tiene la desventaja que permite sólo almacenar valores de 8 bits en la M.P.

Mediante escritura/lectura directa de tablas. Es completamente ventajoso, dado que no utiliza el PC para el direccionado y permite almacenar valores de hasta 16 bits en cada localización de memoria.

Escritura/lectura directa de tablas.

La escritura/lectura directa utiliza dos instrucciones especiales: TBLRD (para lectura) y TBLWT (para escritura).

Además de las dos instrucciones la Familia 18FXXXX ha implementado un registro para el intercambio de datos (8 bits) denominado TABLAT y un puntero TBLPTR de 21 bits accesible a través de tres registros de control: TBLPTRU, TBLPTRH y TBLPTRL.

Registro TABLAT.

Es un almacenador de 8 bits mapeado en el espacio del FSR de la RAM. Se utiliza para la transferencia de datos entre la RAM y la Memoria de Programas.

Puntero para Tablas.

Permite direccionar una localización de la Memoria de Programas, posee 22 bits por lo que puede accesarse mediante tres SFRs (TBLPTRU, TBLPTRH, TBLPTRL). Los tres registros se unen para formar el puntero. Los primeros 21 bits permiten direccionar la localización. El bit número 22 permite escribir la Palabra de Configuración del dispositivo. Siempre que el bit 22 se mantenga en cero, el valor del resto de los bits corresponderá a la dirección dentro de la memoria. El puntero de tablas se utiliza con las instrucciones **TBLRD** y **TBLWR**. Tales instrucciones pueden actualizar el puntero de la tabla según los 4 modos siguientes que se muestran en la **tabla 7.3:**

Tabla 7.3.. Modos de Actualización del puntero para tablas. **Fuente: PIC18FXX2 Data Sheet; High Performance Enhanced FLASH Microcontrollers with 10-Bit A/D (sección 5).**

Las **figuras 7.11 y 7.12** representan esquemáticamente las operaciones de lectura y escritura de tablas en la Memoria de Programas (Familia 18FXXXX).

Figura 7.11. Lectura de Tablas. Esquematización Fuente: PIC18FXX2 Data Sheet; High Performance Enhanced FLASH Microcontrollers with 10-Bit A/D (sección 5).

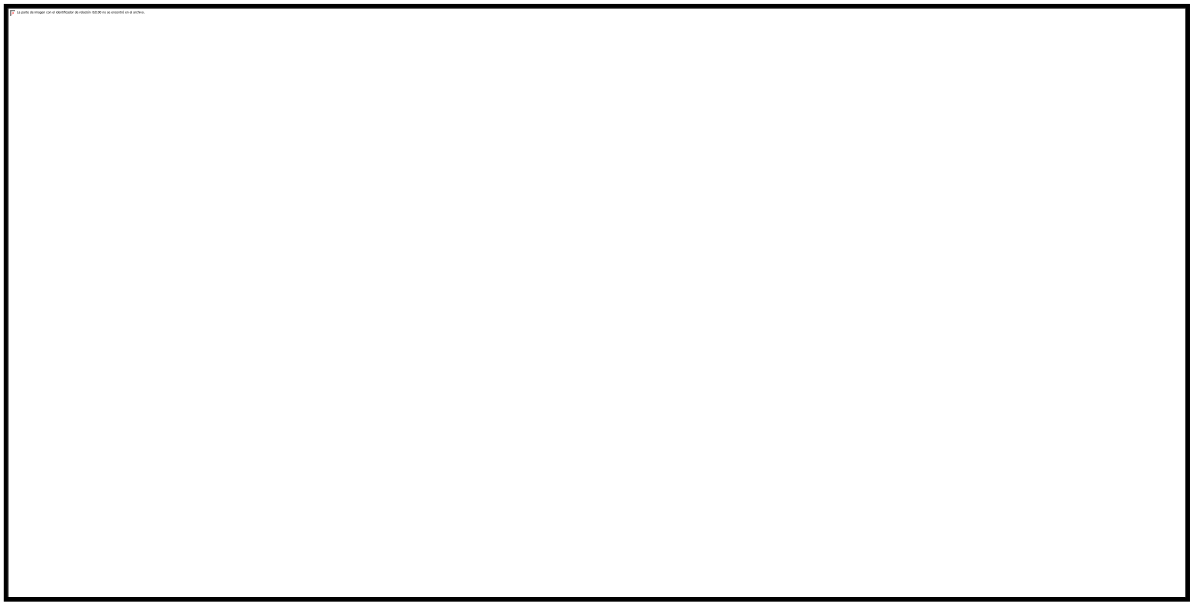


Figura 7.12. Escritura de Tablas. Esquematización Fuente: PIC18FXX2 Data Sheet; High Performance Enhanced FLASH Microcontrollers with 10-Bit A/D (sección 5).

Especificaciones generales del Set de Instrucciones.

Número de Instrucciones.

Los Microcontroladores 18FXXXX manejan un **Set de 75 Instrucciones**.

Codificación.

La mayoría de las instrucciones se codifican en una sola palabra de 16 bits, dado que las localizaciones de la Memoria de Programas miden 16 bits de ancho. Sólo 4 instrucciones se codifican en dos palabras (su código ocupa 2 localizaciones de memoria).

Categorías (clasificación de las instrucciones)

Las instrucciones se agrupan en **cuatro categorías**:

- Instrucciones para operaciones con bytes.
- Instrucciones para operaciones con bits.
- Instrucciones para operaciones con literales (números).
- Instrucciones para operaciones de control.

Operandos según categoría.

Los operandos con que funcionan las instrucciones se pueden dividir según la categoría, como sigue:

- Operandos en las instrucciones para **operaciones con bytes**.
  - File register "f". Posee igual significado que en la serie 87X.
  - Destino del resultado "d". Posee igual significado que en la serie 87X.
  - Segmento de memoria accesada "a".
- Operandos en las Instrucciones para **operaciones con bits.**
  - File register "f". Posee igual significado que en la serie 87X.
  - Número del bit en el registro "b". Posee igual significado que en la serie 87X.
  - Segmento de memoria accesada "a".
- Operandos en las instrucciones **para operaciones con literales** (números).
  - Valor del literal que se desea cargar al registro "k". Posee igual significado que en la serie 87X.
  - El registro FSR seleccionado para cargar el valor del literal en el registro especificado por "f".
  - Operando no requerido " ".
- Operandos en las instrucciones **para operaciones de control**.
  - Una dirección de la Memoria de Programas (especificada por "n").
  - Si se desea o no respaldar los registros WREG, STATUS y BSR (Bank Select Register) en los registros "Shadow" asociados "s".
  - El modo de operación de las instrucciones TBLRD y TBLWR (especificado por "m").
  - Operando no requerido " ".

Cambios Introducidos en el funcionamiento del direccionado de la Memoria de Datos. Memoria de Datos. Especificaciones generales.

- Tipo: RAM estática.
- Capacidad: hasta 16 bancos de 256 bytes cada uno.
- Direccionado: posibilidades de direccionado directo e indirecto. El direccionado directo puede realizarse a través de los 4 bits menos significativos del registro BSR (Bank Select Register) o utilizando un operando "a" que se incluye en la instrucción (referido arriba). El direccionado indirecto se realiza vía punteros FSRs y registro INDF.

Existen ligeras modificaciones en el direccionado que se tratan en el siguiente subtópico. Una dirección dentro de la RAM ocupa 12 bits (se pudieran direccionar hasta 4096 bytes).

- Mapa de Registros (Mapa de la RAM): Los Registros de Propósito Específico (SFR) se encuentran ubicados en el banco 15 a partir desde la dirección (0xF80 hasta la 0xFFF). Los Registros de Propósito General comienzan en el banco 0 (dirección 0x000), teóricamente hasta la dirección 0xEFF; dependiendo del microcontrolador que se esté utilizando[24] El Mapa de registros se muestra en la **figura 7.13** (sección siguiente).

Direccionado de la Memoria de Datos. Direccionado Directo de la RAM. Funcionamiento y utilización del operador "a".

El operador "a" sirve para direccionar directamente dos zonas especiales de la RAM definidas como: Access RAM Low y Access RAM High. La parte baja ("Low") está mapeada desde la dirección 0x000 hasta la 0x07F y corresponde a 128 bytes de GPRs. La parte alta ("High") está mapeada desde la dirección 0xF80 hasta la 0xFFF y corresponde a 128 bytes de SFRs.

Si el operador a = 0 se ignora el valor contenido en los 4 bits menos significativos del BSR (**figura 7.13**).

[24] Por ejemplo, para los PICs 18F242/442 los GPRs se extienden desde la 0x000 hasta la 0x2FF. En el 18F4550 debe tenerse en cuenta el espacio para definir los "endpoits" si se utiliza el Módulo USB; etc.

Figura 7.13. Mapa de la Memoria de Datos (familia 18FXXXX); especificando el Modo de direccionado indirecto tipo “Access Bank”.

Si el operador a = 1 se utiliza el valor cargado en el Registro BSR para formar la dirección (**figura 7.14**).

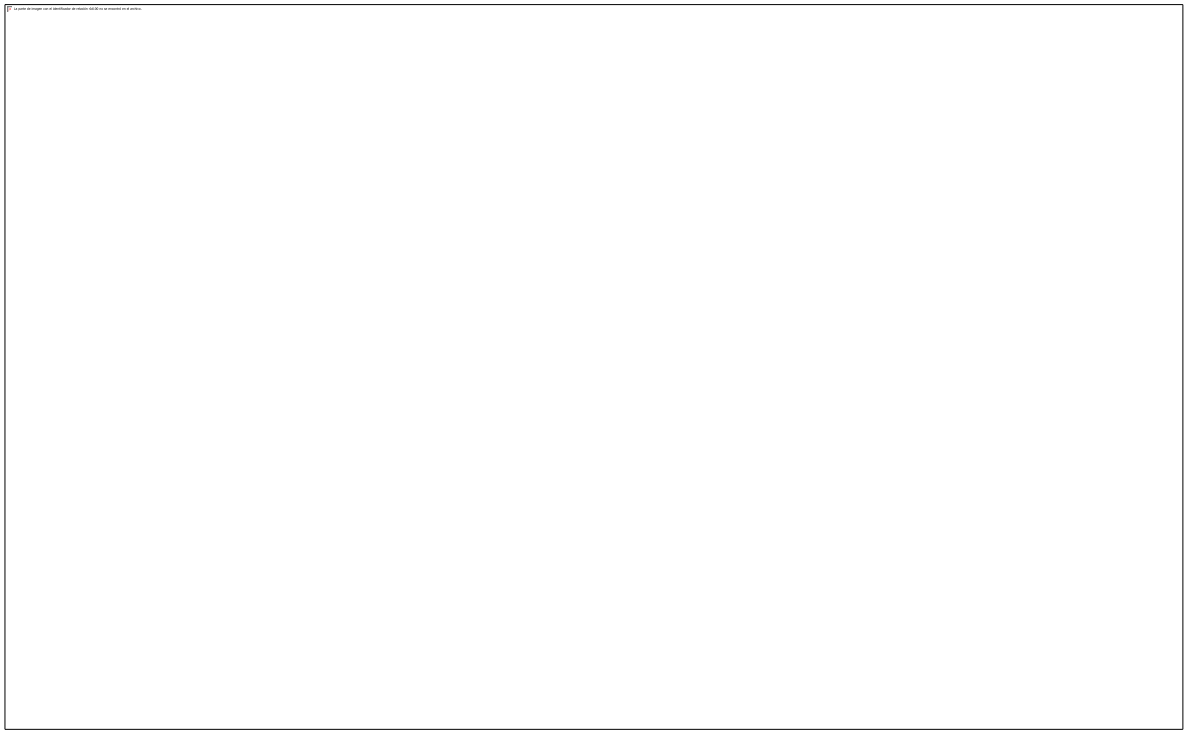


Figura 7.14. Esquema de Direccionado directo utilizando el registro BSR (a = 1).

Casos especiales relativos al direccionado directo.

- La instrucción **"movlwb k"** permite cargar directamente el registro BSR con el literal K.
- La instrucción **"movff fs,fd"** ignora el valor cargado en el BSR puesto que las direcciones de 12 bits tanto para el registro fuente (fs) como para el de destino (fd), se incluyen en el código de la instrucción. Más información puede encontrarse en la página 236 del "DataSheet".

Direccionado de la Memoria de Datos. Direccionado Indirecto de la RAM.

Se utiliza básicamente el mismo principio que en la serie 87X sólo que en este caso existen tres punteros FSR con un ancho de 12 bits cada uno (una dirección de RAM ocupa 12 bits). Cada FSR es accesado mediante un par de registros (FSRHx: FSRLx), donde x ∈[0,1,2].

Adicionalmente existen 3 "registros" INDF (INDF0, INDF1 e INDF2). Recuerde que el INDF en esencia no es un registro físico y que la acción sobre él recae sobre el contenido

de la dirección a la que apunta el FSR. Cada INDF tendrá asociado entonces un registro FSR.

Casos especiales para el tratamiento del FSR.

Las operaciones que se realicen sobre el INDF pudieran o no configurarse en la propia instrucción para que el FSR sufra cambios antes o después de realizar la operación. Tales casos son los siguientes:

- **Instrucción INDFx**. No se realizan cambios en el FSR (Modo clásico, compatible con 87X).

  Ejemplo: CLRF **INDF0**

- **Instrucción POSTDECx**. Se decrementa el FSR después de realizar la acción sobre el INDF.

  Ejemplo: CLRF **POSTDEC0**

- **Instrucción POSTINCx**. Se incrementa el FSR después de realizar la acción sobre el INDF.

  Ejemplo: CLRF **POSTINC0**

- **Instrucción PREINCx**. Se incrementa el FSR antes de realizar la acción sobre el INDF.

  Ejemplo: CLRF **PREINC0**

- **Instrucción PLUSWx**. Utiliza el valor almacenado en el registro W como offset (FSR + W) para el FSR. No modifica los valores del FSR ni el W después que se realiza la acción sobre el INDF.

  Ejemplo: CLRF **PLUSW0**

Periféricos Integrados a la Familia 18FXXX. Enfoque comparativo con la serie 16F87X. Introducción al PIC18F4550. Módulo USB integrado a los micro controladores 18F2/45X0.

Migración del Módulo CCP hacia la Serie 18F2455/2550/4455/4550.

Generalidades.

Los siguientes características resumen los cambios introducidos al Módulo CCP (Compara / Captura / PWM) en los micros de la serie 18F2455/2550/4555/4550.

- Cada micro controlador integrante de la serie 18F2455/2550/4555/4550 posee dos módulos CCP. En los PICs de 40 pines de la serie antes descrita el Módulo CCP1 pueden operar en modo "enhanced" (ECCP), el modo ECCP se discute en la sección 16.
- Operativamente los Módulos CCP funcionan igual que en los microcontroladores de la serie 87X.
- La operación de los Módulos CCP en modo Compara ó Captura comparten en la serie 18FXXXX los recursos TIMER1 ó TIMER3; no así en la serie 87X (sólo compartían la operación con TIMER1).
- La asignación del pin físico asociado al Módulo CCP2 puede variar según el dispositivo. En la serie 87X este pin siempre se asigna al pin RC1. En la serie 18FXXXX, el pin asociado a CCP2 puede multiplexarse hacia el pin RB3 modificando el valor del bit de configuración CCP2MX. Por defecto CCP2MX = 1, lo que asigna a RC1 como pin asociado a CCP2 por defecto. CCP2MX es un bit de la palabra de configuración del chip.

Operación del Módulo CCP en modo enhanced.

Los módulos **CCP1** en los microcontroladores de 40 pines pueden ser operados en modo estándar o "enhanced". El modo "enhanced" incluye las siguientes prestaciones:

- Permite seleccionar entre **2 ó 4 canales de salida (modo "half bridge" y "full bridge"). Las salidas de control** pueden configurarse en nivel alto o bajo.
- Permite añadir a la Onda de PWM un "tiempo muerto" ("dead band").
- Permite apagado y reinicio automático.

En el **modo “enhanced”** el registro de control CCP1CON difiere del CCP estándar en dos factores:

- Se utilizan los dos bits más significativos para configurar el modo de operación “enhanced”.
- Las combinaciones: 1100 => 1111 de los bits CCP1M3 : CCP1M0 se utilizan para definir las combinaciones de activación de las salidas P1A, P1B, P1C y P1D.

En la **figura 7.15** se muestra el nuevo esquema de bits para el registro **CCP1CON** en los microcontroladores **18FXXXX** de **40 pines**.

Figura 7.15. Esquema de bits para el registro CCP1CON de los microcontroladores de 40 pines (serie 18FXXXX). Fuente 18F2455/2550/4555/4550 data sheet (www.microchip.com).

El funcionamiento de los nuevos bits se conjuga con el valor que tengan los bits CCP1M3 / CCP1M2:

- Si los bits CCP1M3 / CCP1M2 poseen cualesquiera de las siguientes combinaciones: 00, 01 ó 10 los dos bits más significativos (P1M1 y P1M0) no son relevantes.
- Si los bits CCP1M3 / CCP1M2 poseen la combinación: 11 los dos bits más significativos (P1M1 y P1M0) funcionan según la lógica que se resume en la **tabla 7.4**.

Tabla 7.4. Combinaciones operativas para los bits P1M1 / P1M0 (enhanced mode).

Descripción detallada de los Modos de Operación.

Single Output.

Este modo opera de manera similar al modo estándar con único canal PWM asociado al pin RC2.

"Half bridge output".

Este modo utiliza dos pines para manejar cargas push – pull. Por el pin P1A emerge la Onda de PWM y por P1B la misma onda invertida. En la **figura 7.16 a** se muestran las formas de onda para este modo y en la **figura 7.16 b** un ejemplo de interface.

Figura 7.16 a. Formas de Onda de PWM en los pines P1A y P1B. Las ondas P1A y P1B están desfasadas en un valor "td" (Dead Band Delay) para evitar la conmutación simultánea de los dos transistores MOSFETS (figura 2 b).

Figura 7.16 b. Ejemplo de circuito "push pull" conectado a los pines P1A y P1B.

"Full bridge output forward".

Este modo utiliza los 4 pines como salidas: P1D genera la Onda de PWM. P1A = 1, P1B = P1C = 0 (niveles permanentes). En la **figura 7.17 a** se muestran las formas de onda de los pines P1A, P1B, P1C y P1D. En la **figura 7.17 b** se muestra el sentido de circulación de la corriente por el puente "H".

Figura 7.17 a. Señales en los pines P1A, P1B, P1C y P1D para modo “forward”.

Figura 7.17 b. Puente “H” conectado en modo “forward” a los pines P1A, P1B, P1C y P1D.

“Full bridge output reverse”.

El modo “reverse” funciona igual que el anterior sólo que la corriente circula en sentido contrario por el puente “H”. El modo “reverse” utiliza los 4 pines como salidas: **P1B** genera la Onda de PWM. **P1C = 1, P1D = P1A = 0 (niveles permanentes).** En la **figura 7.18 a** se muestran las formas de onda de los pines P1A, P1B, P1C y P1D. En la **figura 7.18 b** se muestra el sentido de circulación de la corriente por el puente “H”.

La parte de imagen con el identificador de relación rID26 no se encontró en el archivo.

Figura 7.18 a. Señales en los pines P1A, P1B, P1C y P1D para modo “reverse”.

La parte de imagen con el identificador de relación rID26 no se encontró en el archivo.

Figura 7.18 b. Puente “H” conectado en modo “reverse” a los pines P1A, P1B, P1C y P1D.

Migración del Conversor A/D hacia la Serie 18F2455/2550/4455/4550.

Generalidades.

Las variaciones entendidas como mejoras introducidas al Conversor A/D de la serie 18F2455/2550/4455/4550 se resumen a continuación:

En los microcontroladores de 28 pines se tienen hasta 10 canales analógicos. En los microcontroladores de 40 pines se tienen hasta 13. En la **figura 7.19** se muestra el Diagrama de Bloques del Conversor A/D en la serie 18F2455/2550/4455/4550.

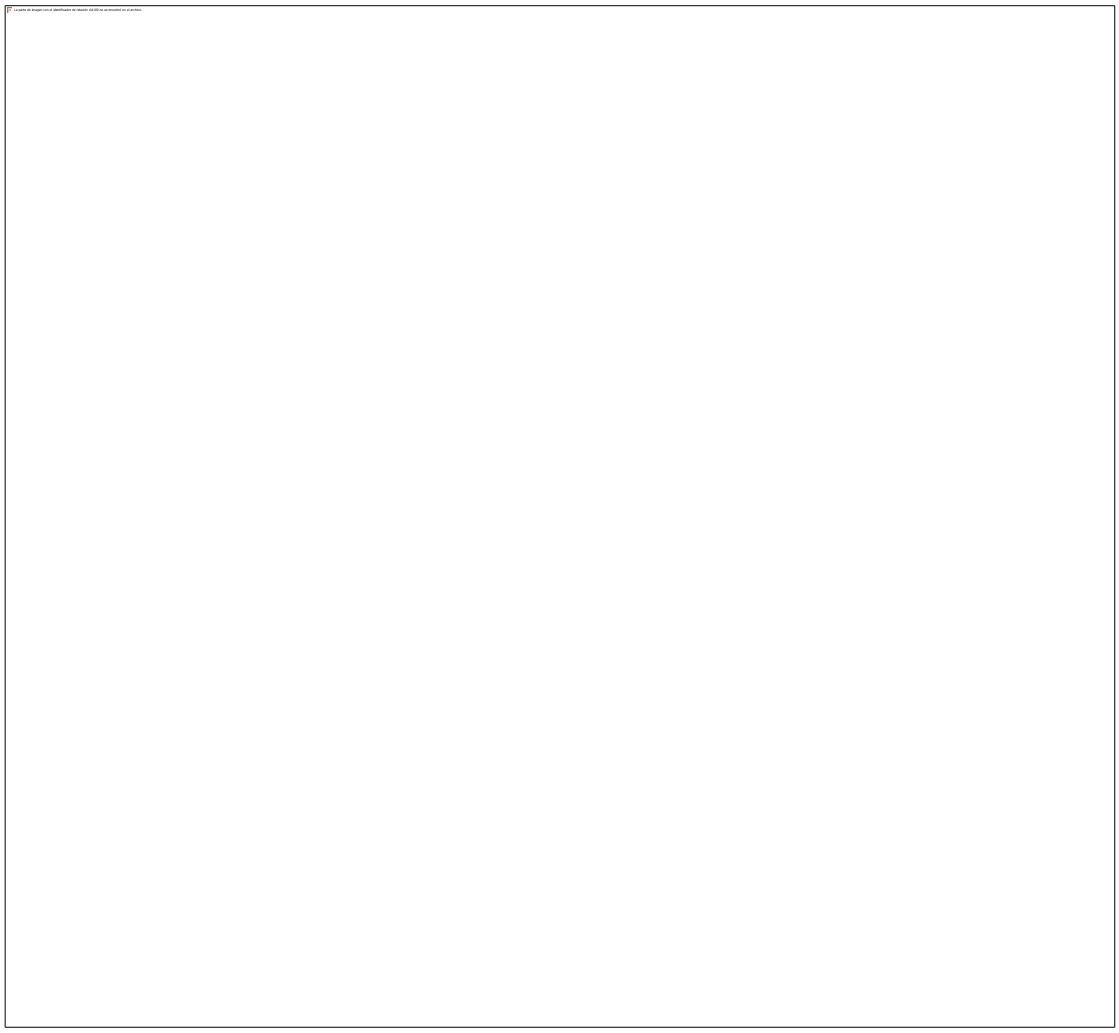


Figura 7.19. Diagrama de Bloques del conversor A/D (serie 18F2455/2550/4455/4550)

Para controlar el conversor se utilizan ahora tres registros: ADCON0, ADCON1 y ADCON2:

El ADCON0 (**figura 7.20**) permite seleccionar el canal a convertir (ahora con cuatro bits: <CHS3:CHS0>), iniciar la conversión o chequear fin de conversión (GO/*DONE) y apagar o encender el A/D (ADON).

Figura 7.20. Registro ADCON0 serie 18F2455/2550/4455/4550)

El ADCON1 (**figuras 7.21 a y b**) permite configurar la referencia de voltaje para el conversor (mediante dos bits: <VCFG1:VCFG0>) y seleccionar las combinaciones de canales analógicos o digitales de una forma cómoda que en la serie 87X (bits: <PCFG3:PCFG0>).

Figura 7.21 a. Esquema de bits, registro ADCON1 conversor A/D (serie 18F2455/2550/4455/4550).

Figura 7.21 b. Bits para el control de la referencia de voltaje del conversor A/D registro ADCON1 conversor A/D (serie 18F2455/2550/4455/4550).

El ADCON2 (**figura 7.22**) permite configurar el formato del resultado (bit 7: <ADFM>) de manera similar a la serie 87X. Permite además definir un múltiplo del Tiempo de Adquisición (bits 5:3 <ACQT2:ACQT0>) y Tiempo de Conversión (bits 1:0 <ADCS2:ADCS0>) mediante tres bits. En la serie 87X el Tiempo de Adquisición era generado por software generalmente, ahora se puede configurar mediante este registro. El Tiempo de Conversión (Reloj de Conversión del Cuantificador) ahora admite hasta 8 opciones, en la serie 87X admite tres opciones.

Figura 7.22. Registro ADCON2 serie 18F2455/2550/4455/4550).

Preguntas Teóricas de la Tecnología 18FXXXX

1. Describa el funcionamiento de los modos de trabajo del oscilador en la tecnología 18FXXXX: RCIO, ECIO y HS + PLL.
2. ¿Qué ventajas ofrece el hecho de que los micro controladores de tecnología 18FXXXX codifiquen sus instrucciones utilizando 16 bits?.
3. ¿Qué capacidad máxima de direccionado de Memoria de Programas manejan los micros 18FXXXX?
4. ¿Qué diferencias operativas existen entre las instrucciones “CALL” y “RCALL” en lo micros 18FXXXX?
5. Defina el funcionamiento de las siguientes instrucciones de rama, incluidas en el “Set de Instrucciones” de la tecnología 18FXXXX:
   - BN
   - BNC
   - BNZ
   - BRA
6. Resuma las ventajas que presenta la PILA (STACK) en los micros 18FXXXX con respecto a la PILA en los microcontroladores 87X.
7. Describa el mecanismo para manejo de tablas en la Memoria de Programas implementado en los micros de tecnología 18FXXXX. Dentro de la descripción considere el funcionamiento de las instrucciones TBL (RD y WT).
8. Resuma las variaciones introducidas al Conversor Analógico Digital integrado a los microcontroladores de tecnología 18FXXXX.
9. Describa el funcionamiento en modo “enhanced” del Módulo CCP1 para los microcontroladores de 40 pines de tecnología 18FXXXX.

Anexos

Anexo 1

Set de instrucciones para el manejo de LCD

La partie de droite contient identifiants des réseaux 42.202 ne se trouvant ne d'archive.

La partie de image car et identificateur de classe 320.00 ne se trouvent ce et ailleurs.

Anexo 2

Introducción a la programación en C de los microcontroladores de tecnología 18FXXXX.

Herramientas de Programación en C.

Existen varios compiladores ANSI C que permiten escribir los programas para micro controladores en este lenguaje. Al programar en C se consigue compactar el código en programas extensos mejorando la legibilidad y capacidad de manipulación de los programas pero al mismo tiempo se pierde un poco el "control total" del código que se consigue programando en ensamblador. Dos ejemplos compiladores muy utilizados son el "MPLAB C18™ Compiler" de Microchip™ y el "Hi-Tech Tools – Picc 18™".

Compilador MPLAB C18™.

El compilador C18 optimiza el compilador estándar de "C" para programar los microcontroladores PIC18. MPLAB C18 se aparta del estándar ANSI "C" en los casos en los que este último genera conflictos para el soporte eficiente del micro controlador. Dentro de las especificaciones del compilador MPLAB C18 se pueden citar las siguientes:

- Compatibilidad con el estándar ANSI C X3.159-1989.
- Integrado al ambiente de desarrollo MPLAB IDE.
- Asimila objetos de código escritos en ensamblador y en C, por lo que es posible programar en ambos lenguajes.
- Posee una extensa biblioteca de funciones para manipulación de periféricos, de cadenas y problemas matemáticos.

Utilización del compilador MPLAB C18 std_ed_version.

Instalación.

El proceso de instalación puede consultarse en el documento MPLABC18_Getting_Started.pdf[25]. Resulta importante considerar los elementos componentes del compilador MPLAB C18. Los componentes fundamentales se muestran en la **tabla a2.1**.

[25] www.microchip.com

Tabla a2.1. Elementos componentes principales del MPLAB C18 compiler.

Utilizando MPLAB C18 desde MPLAB IDE.

Los siguientes pasos resumen de manera simplificada el proceso para hacer funcionar el compilador MPLAB C18 sobre MPLAB IDE.

1. Crear un nuevo proyecto.

Crear un nuevo proyecto utilizando el menú **Project => New**. Se recomienda crear una carpeta que contenga todos los archivos del proyecto. Los archivos de "proyecto" tienen una extensión **"*.mcp"**. Como resultado de la creación del proyecto aparecerá una ventana denominada "project TREE" . "Project TREE es un explorador que contiene

información sobre todos los archivos del proyecto. En las **figuras a2.1 a, b y c** se ilustra el proceso descrito en este primer paso.

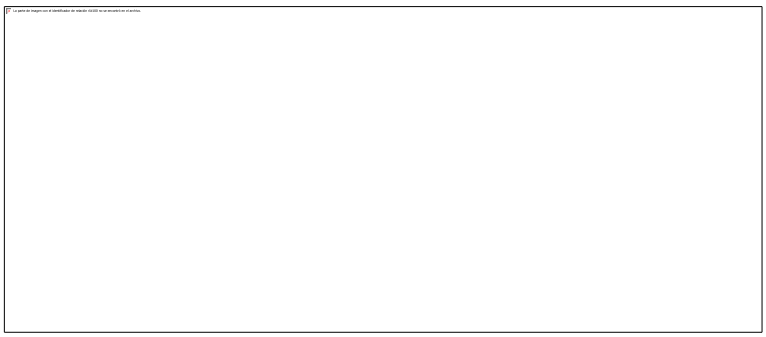


Figura a2.1 a. Seleccionar creación de un nuevo proyecto.

Figura a2.1 b. Caja de diálogo para el nuevo proyecto. Debe ubicarse mediante el "Browser" el archivo del proyecto.

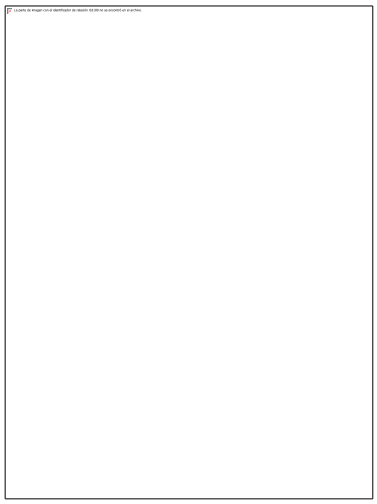


Figura a2.1 c. Caja de diálogo para el árbol del nuevo proyecto. En esta ventana se muestran todos los archivos componentes de un proyecto. Si la ventana no se visualiza utilice los menús View => Project. Información de los archivos componentes de un proyecto se encuentran en la tabla a2.1.

2. Seleccionar el tipo de procesador.

Para seleccionar el procesador utilice los menús: Configure => Select Device. En la **figura a2.2** se muestra la caja de diálogo para seleccionar el dispositivo.

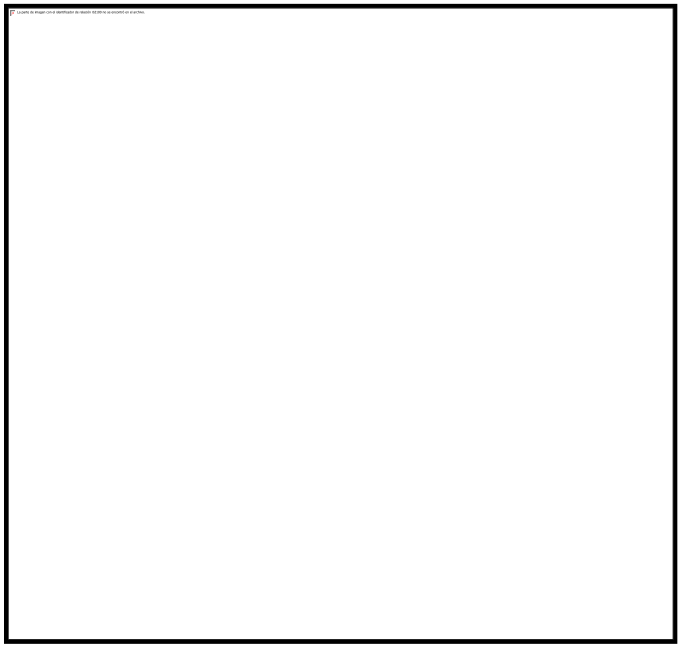


Figura a2.2. Caja de diálogo para seleccionar el dispositivo.

El compilador MPLAB C18 trata de manera genérica a los microcontroladores 18FXXXX, aunque es posible definir un procesador específico para garantizar los bits de configuración adecuados a la hora de programar físicamente el chip.

3. Establecer los parámetros del proyecto que fue creado.

En este punto se configuran los siguientes “items”:

- Se selecciona y configura el “tool suite” que se va a trabajar. Por ejemplo: “Microchip C18 Tool suite™”. “Tool suite” es el conjunto de herramientas con las que se desarrollarán las operaciones sobre el proyecto.
- Se configuran las opciones de compilación para el proyecto (“Build Options”).
- Se configuran el Compilador (C18 en este caso) y el “Linker”.

Selección del “tool suite”.

Para seleccionar el “tool suite” utilice la combinación de menús: Project => Set Languaje Toolsuite … (**figura a2.3 a**).

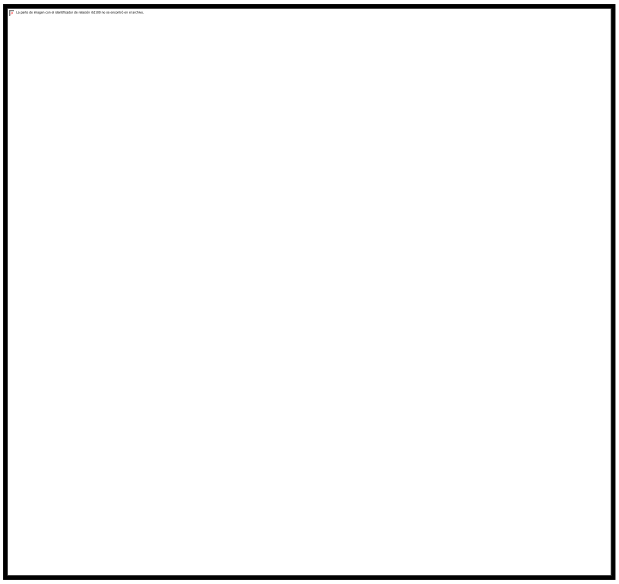


Figura a2.3 a. Menús de selección “tool suite”.

Aparecerá entonces la siguiente caja de diálogos (**figura a2.3 b**).

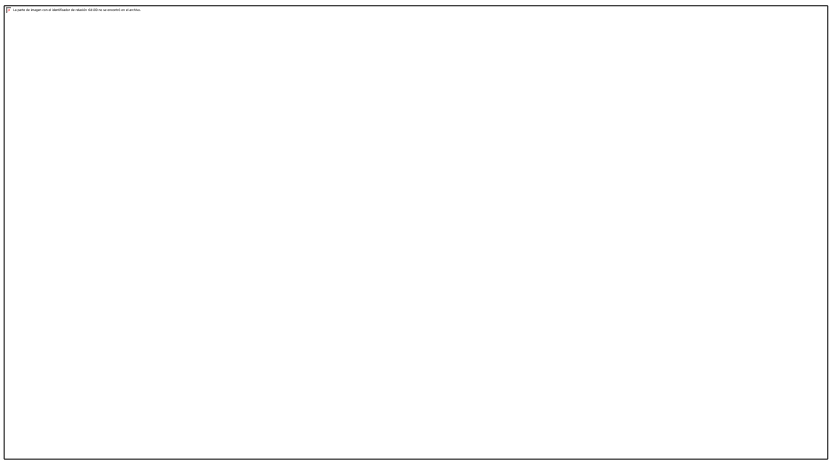


Figura a2.3 b. Caja de diálogo para seleccionar el “tool suite”. Se debe seleccionar la herramienta y el “path” del ejecutable (*.exe) en caso de que no aparezca en el campo “Location”.

Configurar el proyecto (general), el compilador y el "linker".

Configuración general del proyecto.

Para entrar a configurar el proyecto, el compilador o el linker utilice la combinación de menús: **Project => Build Options … => Project (figura a2.4).**

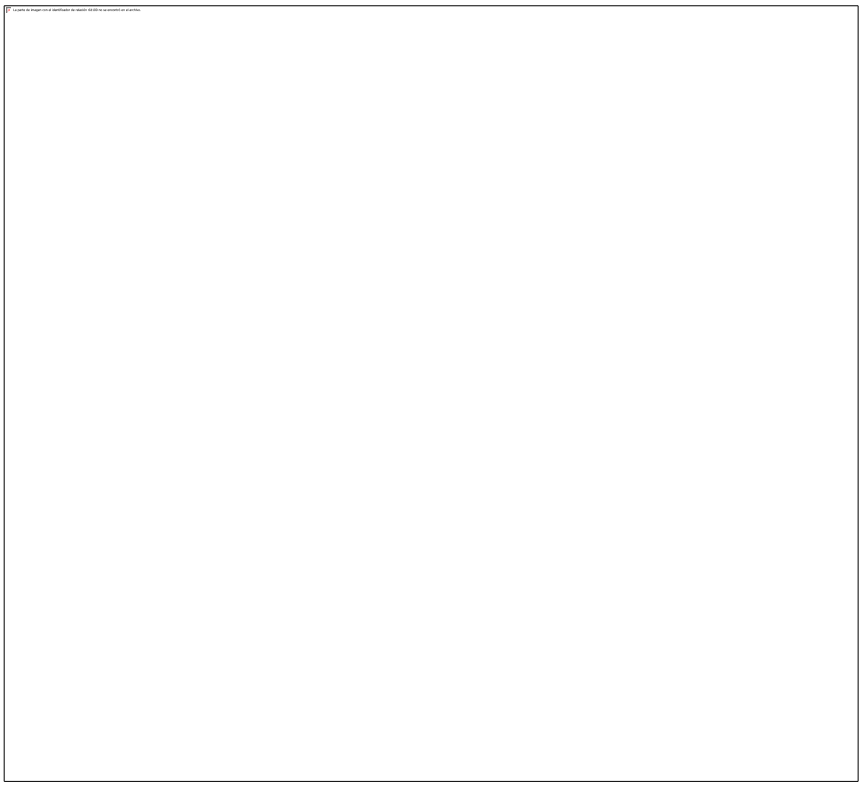


Figura a2.4. Juego de menús para configurar el compilador y el "linker" para el proyecto actual.

En el **"Tab" "Directories"** se puede establecer la ubicación ("Path") para los archivos relacionados con el proyecto **(figura a2.5).**

Figura a2.5. “Tab” “Directories” de la caja de configuración para el proyecto.

Las carpetas que pueden seleccionarse en el campo señalado en la **figura a2.5** son:

- **“Output Directory”** es el destino final de los ficheros resultantes de la compilación completa del proyecto: *.cod, *.cof y *.hex. Si el “Path” se deja en blanco todos los archivos de salida se grabarán en la carpeta donde ha sido creado el proyecto.
- **“Intermediates Directory”** es el lugar donde el compilador graba lo archivos objeto (productos de la compilación). Igualmente, si el “Path” se deja en blanco todos los archivos de salida se grabarán en la carpeta donde ha sido creado el proyecto.
- **“Include search path”.** Define el “Path” donde se han grabado los ficheros Include (*.h). El path sería: C:\MCC18\h\.
- **“Library search path”.** Define el “Path” donde se han grabado los ficheros LIB (*.lib). El path sería: C:\MCC18\lib\. C:\MCC18\lkr
- **“Linker – Script Search Path”.** Define el “Path” donde se han grabado los ficheros LKR (*.lkr). El path sería: C:\MCC18\lkr\.

Configurar el compilador y el "linker".

Para configurar el compilador o el linker desde el MPLAB, una vez dentro del menú señalado en la figura 4; seleccione el "Tab" correspondiente. La **figura a2.6** muestra las opciones "por defecto" para configurar el compilador.

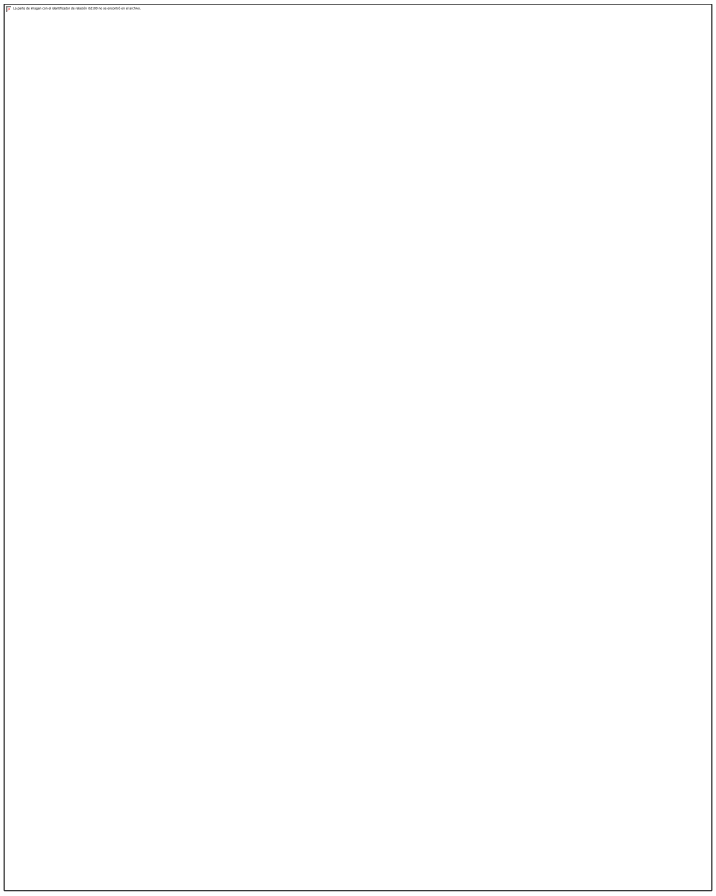


Figura a2.6. Opciones de configuración por defecto para el compilador.

Para configurar el "linker" seleccione el "Tab": **MPLINK Linker** (figura 7). Las opciones "por defecto" son completamente válidas.

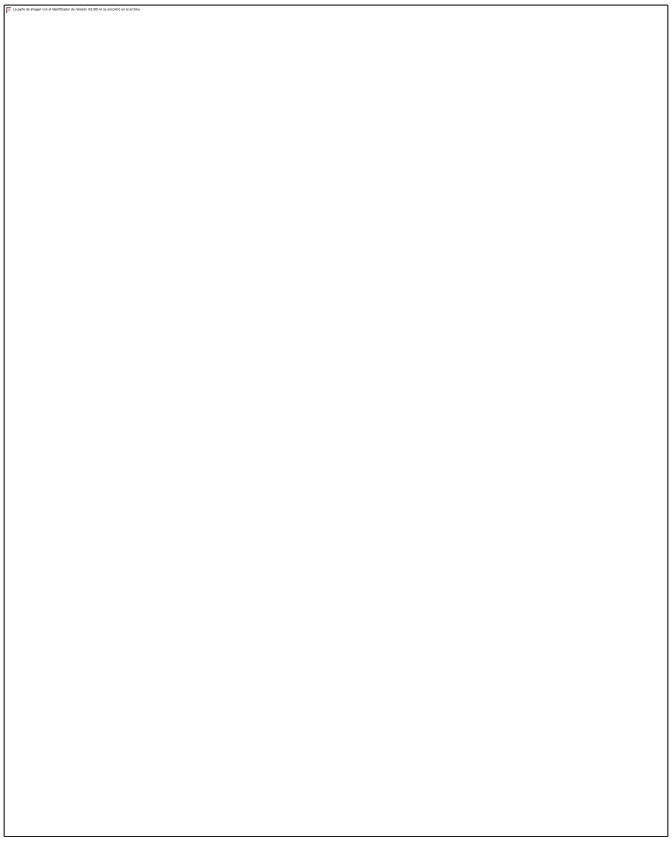


Figura a2.7. Opciones de configuración por defecto para el Linker.

4. Crear y editar un nuevo archivo *.C o añadirlo al proyecto.

Para añadir cualquier archivo fuente (*.c) al proyecto desde el "Project TREE" (**figura a2.1c**): "click" derecho encima de la carpeta **"Source Files"** y seleccionar **"Add Files …"**.

Tips y ejemplos (programación).

Este tópico pretende ejemplificar el empleo de la sintaxis para un programa C para micro controladores que compile sobre MPLAB C18 en diferentes situaciones.

1. Referirse a un registro específico.

En los correspondientes (*.h) se definen los registros como variables tipo char: por ejemplo **"extern volatile near unsigned char PORTA;".** Esto significa que el puerto A en el código puede ser tratado como **"PORTA".** Lo anterior es válido para cualquier registro. Por ejemplo la sentencia siguiente asigna el valor cero al registro TRISB:

TRISB = 0;

Esta otra sentencia asigna el valor de la variable **“led_display” (tipo char)** al **PORTB:**

PORTB = led_display;

2. Referirse a un bit específico de cualquier registro.

Para referirse a un bit específico emplear la sintáxis:

<Nombre del Puerto><bits><.><Nombre del Bit>

Por ejemplo, la sentencia siguiente se refiere al bit T0IF del registro INTCON:

INTCONbits.TMR0IF = 0;

3. Utilizar las funciones integradas a las bibliotecas para manejo de periféricos o de propósito general.

Para utilizar estas funciones es necesario conocer a fondo las posibilidades de las bibliotecas C, preparadas por Microchip™ u otro proveedor alternativo. En este segmento sólo se ilustran algunos ejemplos. La documentación relativa a las bibliotecas Microchip™ se discuten en el documento: **MPLAB-C18-Libraries_51297f.pdf**. Las bibliotecas de funciones se dividen en varias categorías.

A continuación se mencionan las funciones componentes de la categoría “Hardware”:

- A/D Converter (Section 2.2 “A/D Converter Functions”)
- Input Capture (Section 2.3 “Input Capture Functions”)
- I2C™ (Section 2.4 “I2C™ Functions”)
- I/O Ports (Section 2.5 “I/O Port Functions”)
- Microwire (Section 2.6 “Microwire Functions”)
- Pulse-Width Modulation (PWM) (Section 2.7 “Pulse-Width Modulation Functions”)
- SPI™ (Section 2.8 “SPI™ Functions”)
- Timer (Section 2.9 “Timer Functions”)
- USART (Section 2.10 “USART Functions”)

La categoría “Hardware” agrupa las funciones que sirven para configurar diferentes elementos de “hardware” integrados al microcontrolador.

Ejemplo 1. Configuración del módulo PWM.

- **OpenPWM1 (**char *Period***).** Esta función configura el Módulo CCP1 en modo PWM para un período *"Period".* El período debe ser un número en tre 0x00 y oxFF.

Ejemplo 2. Setea un valor de Ciclo Útil para el canal de PWM (en este caso CCP1).

- **SetDCPWM1 (**unsigned int ***dutycycle*****).** Esta función escribe el valor del Ciclo Útil en los correspondientes registros de control; ***dutycycle*** puede ser un número de 10 bits entre cero y 1023.

Ejemplo 3. Función que pregunta si el Conversor A/D tiene conversión en curso (uso del A/D por encuesta).

- **BusyADC ().** Esta función retorna el valor uno si el A/D está realizando una conversión y cero cuando finaliza la conversión.

Ejemplo 4. Función que retorna el resultado de la conversión (entero de 16 bits).

- **ReadADC().** Retorna el resultado de la conversión.

La categoría "**Software Peripheral Library"** incluye numerosas funciones para el tratamiento de periféricos internos o externos estándares.

A continuación se mencionan las funciones componentes de la **categoría** "**Software Peripheral Library":**

- **External Matrix LCD** Functions (**Section 3.2 "External LCD Functions"**)
- **External CAN2510** Functions (**Section 3.3 "External CAN2510 Functions"**).
- **Software I2C™ Functions** (**Section 3.4 "Software I2C Functions"**).
- **Software SPI™** Functions (**Section 3.5 "Software SPI™ Functions"**).
- **Software UART** Functions (**Section 3.6 "Software UART Functions"**).

Ejemplo 5. Para LCD. Configura el tipo de interface a utilizar con el LCD matricial (4 u 8 bits de datos) y la cantidad de puntos de la matriz.

- **OpenXLCD**(unsigned char ***lcdtype***). ***"lcdtype"*** se forma con el AND de dos constantes que definen el tipo de interface y los parámetros de la matriz:

Tipo de Interface de datos:

- **FOUR_BIT**: 4-bit Data Interface mode
- **EIGHT_BIT**: 8-bit Data Interface mode

Configuración de la matriz:

- **LINE_5X7**: 5x7 characters, single line display
- **LINE_5X10**: 5x10 characters display
- **LINES_5X7**: 5x7 characters, multiple line display

Código ejemplo: **OpenXLCD(EIGHT_BIT & LINES_5X7);**

La categoría "**General Software Library"** incluye numerosas funciones de propósito general.

A continuación se mencionan las funciones componentes de la **categoría** "**General Software Library":**

- Character Classification Functions
- Data Conversion Functions
- Memory and String Manipulation Functions
- Delay Functions
- Reset Functions
- Character Output Functions

La categoría "**Math Libraries"** incluye funciones matemáticas divididas en dos categorías:

- 32-Bit Floating Point Math Library.
- The C Standard Library Math Functions.

4. Análisis de ejemplos de código.

Ejemplo 1. Saca el valor de un contador por el puerto B. No es práctico pues no maneja demoras. Fuente: **MPLABC18_Getting_Started.pdf**

```
#include <p18cxxx.h> /* Include genérico para todos los PICS 18FXXXX */
int counter;
/* Programa Principal */
void main (void)
{
counter = 1;
TRISB = 0; /* Configura el puerto B como salida */
/*Conteo*/
while (counter <= 15)
{
PORTB = counter; /* Saca valor de conteo por puerto B */
counter++;
}
}
```

Ejemplo 2. Lee caracteres recibidos por el USART y los pone en un LCD Matricial (controlador HD44780 de Hitachi™) Fuente: MPLAB C18 examples.

```
#include <p18C452.h>

#include <xlcd.h>
#include <delays.h>
#include <usart.h>
/* FUNCIÓN demora de 18TCY */
/*Utiliza el *.h "delays.h"
void DelayFor18TCY( void )
{
Nop();
Nop();
Nop();
Nop();
Nop();
Nop();
Nop();
Nop();
Nop();
Nop();
Nop();
Nop();
}
/* FUNCIÓN Implentación de una demora de 15 ms*/
/*Utiliza el *.h "delays.h"
void DelayPORXLCD (void)
{
Delay1KTCYx(60); // demora de 5ms (60 * 1000 * Valor de 1 Ciclo de
Máquina
// Ciclos = (Tiempo de demora * Fosc) / 4
// Ciclos = (5ms * 16MHz) / 4
// Ciclos = 60000
return;
}
/*FUNCIÓN Implentación de una demora de 5 ms (Power ON para el LCD)*/
/*Utiliza el *.h "delays.h"
void DelayXLCD (void)
{
Delay1KTCYx(20); // demora de 5ms (20 * 1000 * Valor de 1 Ciclo de
Máquina
// Ciclos = (Tiempo de demora * Fosc) / 4
// Ciclos = (5ms * 16MHz) / 4
// Ciclos = 20000
return;
}
/*PROGRAMA PRINCIPAL*/
void main( void )
{
char data;
// Inicia el LCD para 8 bits y 5 x 7 puntos
OpenXLCD( EIGHT_BIT & LINES_5X7 );
// configura el USART
OpenUSART( USART_TX_INT_OFF & USART_RX_INT_OFF & USART_ASYNCH_MODE &
USART_EIGHT_BIT &
```

```
USART_CONT_RX,
25); /*Configura el USART*/
while(1)/*Recepción del USART por encuesta*/
{
while(!DataRdyUSART()); //espera por dato listo
data = ReadUSART(); //lee el dato
WriteDataXLCD(data); //Escribe el dato al LCD (se supone en ASCII)
if(data=='Q')
break;
}
CloseUSART(); // Inhabilita el USART
}
```