CURSO BÁSICO DE PIC16F877

Raúl Peralta Meza
Carlos Quiñones Quispe

Generalidades

Módulo 1: Manejo de Puertos Digitales

Módulo 2: Manejo de Temporizadores

Módulo 3 : Convertidor Análogo Digital

Módulo 4: Comunicación Serie Asíncrona

Módulo 5 : Manejo de interrupciones

Módulo 6: Memoria EEPROM

Módulo 7 : Manejo de Páginas de Memoria y Watch Dog

Anexos

Generalidades

Historia e importancia de los microcontroladores

Hasta antes de la aparición de los microprocesadores (1971), la mayor parte de las aplicaciones digitales en electrónica se basaban en la llamada lógica cableada, es decir, si existía un problema este era analizado y se sintetizaba una función en base a la lógica de boole que era la solución al problema planteado.

Con la aparición de los microprocesadores, se varió el esquema de diseño de tal forma que un problema era descompuesto en una serie de tareas mas simples, el microprocesador ejecutaba una serie de pasos o instrucciones para llevar a efecto cada una de las tareas, en ocasiones no era necesario volver a armar un circuito para solucionar otro problema sino que se cambiaba las instrucciones (programa) para obtener otra aplicación

Desde luego el microprocesador es como el cerebro que ejecuta operaciones de índole aritméticas y lógicas por tanto no manejaba líneas externas (periféricos) más aún tampoco tenia un lugar donde almacenar el programa y los datos que necesitaba el programa para encontrar la respuesta al problema. El microprocesador buscaba una instrucción y la ejecutaba; al conjunto de circuitos (hardware) que daban el soporte necesario al microprocesador se le llamo sistema mínimo.
Con el pasar de los años el sistema mínimo se convirtió en un estándar, por otro lado la escala de integración mejoro y posibilito (1976) sintetizar en un solo chip un sistema mínimo, al cual se le llamo SISTEMA A que no era otra cosa que el primer microcontrolador.
En consecuencia definimos así a un microcontrolador; como un procesador con su sistema mínimo en un chip (incluye memoria para programa y datos, periféricos de entrada / salida, conversores de AD y DA, módulos especializados en la transmisión y recepción de datos). Desde luego que hay diferencias sustanciales como la arquitectura cerrada de un microcontrolador, en cambio en un microprocesador es abierta, podemos sumar nuevos dispositivos en hardware en función a las necesidades que la aplicación demande.
Otra diferencia entre los microcontroladores y los microprocesadores es que los primeros cuentan con un **set de instrucciones reducido** en cambio la mayoría de los microprocesadores tienen mayor cantidad de instrucciones. Por otro lado la mayoría de los microcontroladores posee una arquitectura Harvard frente a una arquitectura Von Neuman de los microprocesadores.
Los microcontroladores se especializan en aplicaciones industriales para resolver problemas planteados específicos por ejemplo: los encontramos en los teclados o mouse de las computadoras, son el cerebro de electrodomésticos, también los encontramos en las industria automotriz, en el procesamiento de imagen y video.
Cabe señalar que los el aumento progresivo de la escala de integración y las técnicas de fabricación hacen que cada vez aparezcan microcontroladores mas poderosos y rápidos.

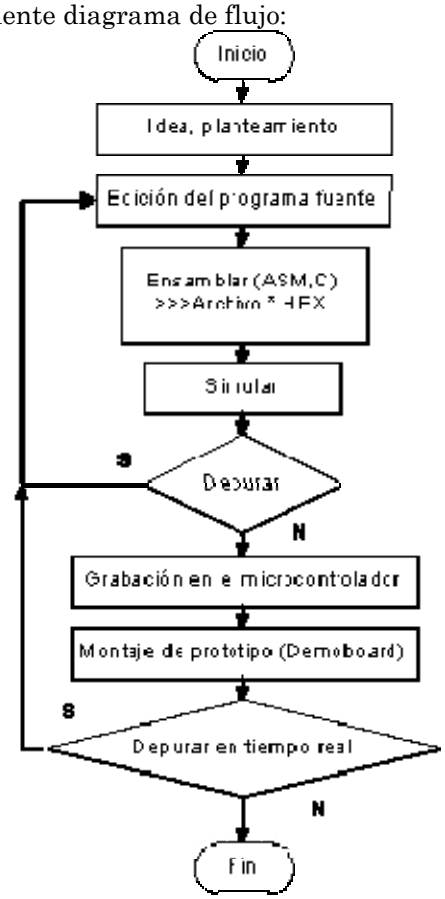
Metodología de estudio

En el presente curso pretendemos aprender a usar microcontroladores. Aprender significa:

- Entender como funciona la arquitectura.
- Comprender y aplicar las instrucciones que tiene el dispositivo.
- Plantear soluciones a problemas.
- Aprender a usar la herramientas de programación y depuración. (ICD

- Detectar y corregir los errores sintácticos y lógicos.
- Optimizar el programa final.

En el desarrollo de texto implementaremos circuitos simples pero demostrativos de las bondades y posibilidades de los microcontroladores. El proceso de diseño puede ser expresado a través del siguiente diagrama de flujo:



Como podemos apreciar todo principia en una idea la cual se ha de plasmar en diagramas de flujo o automatas o alguna otra metodologia que ayude al modelamiento, una vez superado este punto procedemos a usar un editor de texto para codificar el diagrama de flujo a través de las instrucciones con que cuenta el microcontrolador.

A continuación presentamos ese archivo a un programa ensamblador (si es que usamos el lenguaje ensamblador) o un compilador (si usamos otro lenguaje como el C o Basic) aqui se depuran los errores sintácticos que son errores en la estructura del lenguaje de programación.

Una vez que superamos esa etapa procedemos a usar un software, para simular el programa verificando que la solución es válida. En caso que la simulación indique errores procedemos a replantear la solución retomando el punto inicial. Si la solución es la que deseamos procedemos a grabar el programa (debidamente compilado) en el microcontrolador haciendo uso de una herramienta (grabador de microcontroladores). En este punto es posible aplicar un emulador o un ICD a fin de verificar que los resultados elaborados por el simulador son apropiados, el emulador o ICD a diferencia del simulador trabaja sobre hardware real. En caso que falle en este punto retomamos el diseño original.

Superada ambas fases procedemos a colocarlo sobre el hardware final que ha de operar.

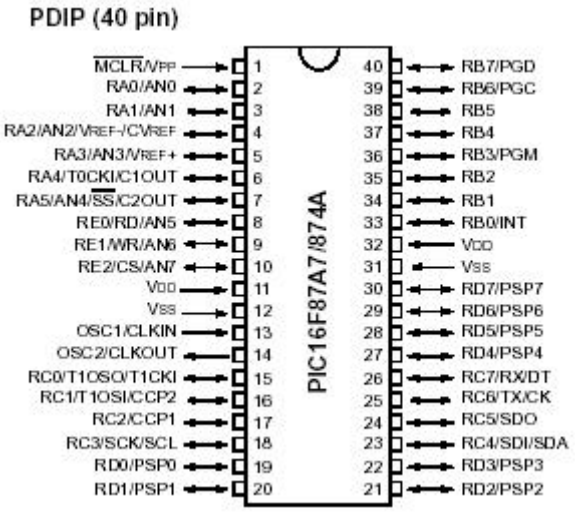
Microcontroladores Microchip 16F87X

Casi todos los fabricantes de microprocesadores lo son también de microcontroladores, en el mercado existen una serie de marcas bastante conocidas y reconocidas como es el caso de Microchip, Motorola, Hitachi, etc. Hemos seleccionado a Microchip y en particular la serie 16F87X, motivos para usar este dispositivo sobran, el principal de ellos es la abundante información y herramientas de diseño existente en el mercado (tanto local como internacional). También salta a la vista el hecho que es sencillo en el manejo y contiene un buen promedio elevado en los parámetros (velocidad, consumo, tamaño, alimentación).

Las principales características con que cuenta el 16F87X son:

- Procesador de arquitectura RISC avanzada
- Juego de 35 instrucciones con 14 bits de longitud. Todas ellas se ejecutan en un ciclo de instrucción menos las de salto que tardan 2.
- Frecuencia de 20 Mhz
- Hasta 8K palabras de 14 bits para la memoria de codigo, tipo flash.
- Hasta 368 bytes de memoria de datos RAM
- Hasta 256 bytes de memoria de datos EEPROM
- Hasta 14 fuentes de interrupción internas y externas
- Pila con 8 niveles
- Modos de direccionamiento directo, indirecto y relativo
- Perro guardian (WDT)
- Código de protección programable
- Modo Sleep de bajo consumo
- Programación serie en circuito con 2 patitas
- Voltaje de alimentación comprendido entre 2 y 5.5 voltios
- Bajo consumo (menos de 2 mA a 5 V y 5 Mhz)

El siguiente diagrama da cuenta de los pines del PIC16F87X:



Donde:

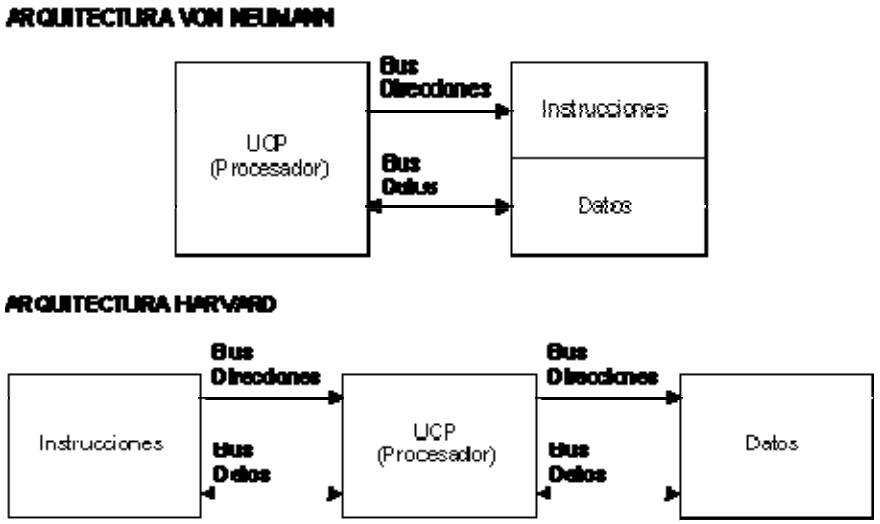
PIN	DESCRIPCION
OSC1/CLKIN(9)	Entrada para el oscilador o cristal externo.
OSC2/CLKOUT (10)	Salida del oscilador. Este pin debe conectarse al cristal o resonador. En caso de usar una red RC este pin se puede usar como tren de pulsos o reloj cuya frecuencia es 1/4 de OSC1

PIN	DESCRIPCION
MCLR/VPP/ THV(1)	Este pin es el reset del microcontrolador, también se usa como entrada o pulso de grabación al momento de programar el dispositivo.
RA0/AN0(2)	Puede actuar como línea digital de E/S o como entrada analógica del conversor AD (canal 0)
RA1/AN1(3)	Similar a RA0/AN0
RA2/AN2/VREF-(4)	Puede actuar como línea dígital de E/S o como entrada analógica del conversor AD (canal 2) o entrada negativa de voltaje de referencia
RA3/AN3/VREF+(5)	Puede actuar como línea digital de E/S o como entrada analógica del conversor AD (canal 3) o entrada positiva de voltaje de referencia
RA4/T0CKI (6)	Línea digital de E/S o entrada del reloj del timer 0. Salida con colector abierto
RA5/SS#/AN4(7)	Línea digital de E/S, entrada analógica o selección como esclavo de la puerta serie síncrona.
RB0/INT(21)	Puerto B pin 0, bidireccional. Este pin puede ser la entrada para solicitar una interrupción.
RB1(22)	Puerto B pin 1, bidireccional.
RB2(23)	Puerto B pin 2, bidireccional.
RB3/PGM(24)	Puerto B pin 3, bidireccional o entrada del voltaje bajo para programación
RB4(25)	Puerto B pin 4, bidireccional. Puede programarse como petición de interrupción cuando el pin cambia de estado.
RB5(26)	Puerto B pin 5, bidireccional. Puede programarse como petición de interrupción cuando el pin cambia de estado.
RB6/PGC(27)	Puerto B pin 6, bidireccional. Puede programarse como petición de interrupción cuando el pin cambia de estado. En la programación serie recibe las señales de reloj.
RB7/PGD(28)	Puerto B pin 7, bidireccional. Puede programarse como petición de interrupción cuando el pin cambia de estado. En la programación serie actua como entrada de datos
RC0/T1OSO/ T1CKI(11)	Línea digital de E/S o salida del oscilador del timer 1 o como entrada de reloj del timer 1
RC1/T1OSI/ CCP2(12)	Línea digital de E/S o entrada al oscilador del timer 1 o entrada al módulo captura 2/salida comparación 2/ salida del PWM 2
RC2/CCP1(13)	E/S digital. También puede actuar como entrada captura 1,/salida comparación 1/ salida de PWM 1
RC3/SCK/SCL (14)	E/S digital o entrada de reloj serie síncrona /salida de los módulos SP1 e I2C.
RC4/SDI/SDA (15)	E/S digital o entrada de datos en modo SPI o I/O datos en modo I2C
RC5/SDO(16)	E/S digital o salida digital en modo SPI
RC6/TX/CK(17)	E/S digital o patita de transmisión de USART asíncrono o como reloj del síncrono
RC7/RX/DT(18)	E/S digital o receptor del USART asíncrono o como datos en el síncrono
RD0/PSP0- RD7/PSP7 (19-22, 27-30)	Las ocho paptitas de esta puerta pueden actuar como E/S digitales o como líneas para la transferencia de información en la comunicación de la puerta paralela esclava. Solo están disponibles en los PIC 16F874/7.
RE0/RD#/AN5 (8)	E/S digital o señal de lectura para la puerta paralela esclava o entrada analógica canal 5.
RE1/WR#/AN6 (9)	E/S digital o señal de escritura para la puerta paralela esclava o entrada analógica canal 6.
RE2/CS#/AN7	E/S digital o señal de activación/desactivacion de la puerta paralela esclava o entrada analógica canal 7.

PIN	DESCRIPCION
VSS(8,19)	Tierra.
VDD(20,32)	Fuente (5V).

Arquitectura 16F87X

Los PIC16F87X de Microchip pertenecen al tipo de procesador RICS que es un procesador de instrucciones reducidas, se caracteriza por que el número de instrucciones es pequeño y ademas casi todas se realiza en la misma cantidad de tiempo, por otro lado posee unidades que trabajan en paralelo conectadas por pipes o tuberias. Este tipo de procesador emplea una arquitectura Harvard lo que significa que trabaja las zonas de memoria de programa y datos en forma separada. En el siguiente diagrama se muestra la arquitectura Von Neuman frente a la Harvard:



En ambas arquitecturas observamos bloques de memoria, cada bloque tiene posiciones y cada posición un valor. Para recoger o dejar un valor en una determinada posición es necesario primero indicar cual es la dirección a leer o escribir de la memoria, en consecuencia hay un grupo de líneas que nos permiten hacer esa función conocida como el bus de direcciones, también existe un bus de datos que son líneas paralelas por donde discurren los valores de cada dirección.

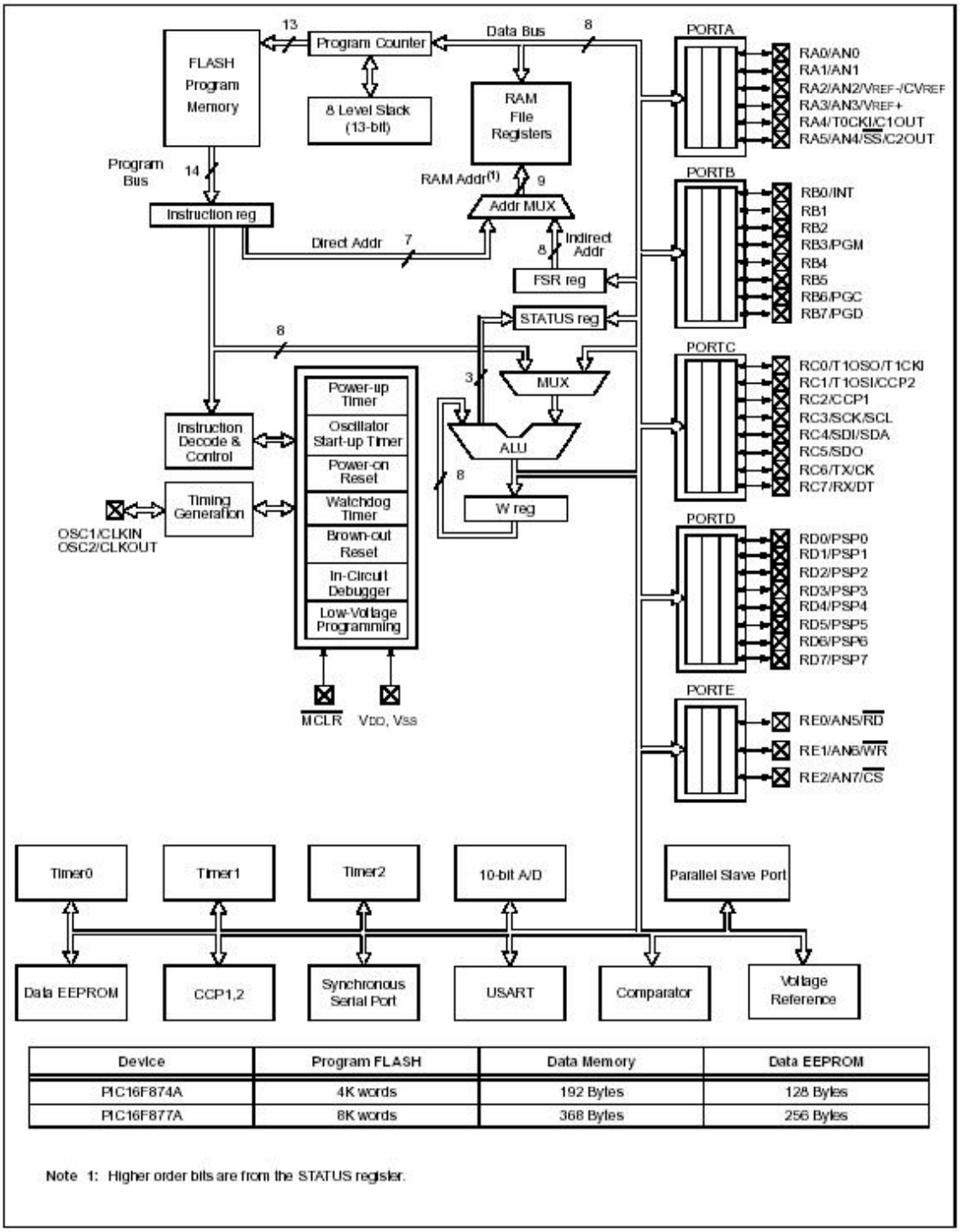
En el caso de la arquitectura Von Neuman podemos apreciar que existe un único bus de direcciones y de datos. Podemos apreciar como cada posición de memoria tiene una dirección, a su vez la memoria se divide en memoria de programa (conocida como ROM) y memoria de datos (conocida como RAM).

En el caso de la arquitectura Harvard existen dos bloques de memoria separados. Un bloque para instrucciones y otro para datos. Note como hay dos buses independientes de direcciones y el bus de instrucciones solo tiene una dirección, a diferencia del bus de datos que es de naturaleza bidireccional. Todo esto sugiere que puede existir una dirección por ejemplo la 0. Entonces tenemos una instrucción en la posición 0 y también un dato en la 0. En el caso de la arquitectura Von Neumann esa dirección es de programa o de instrucción pero no de ambas.

La arquitectura Harvard mejora el ancho de banda por que el bus de datos es de 14 bits frente a los de 8 de un bus tradicional Von Neumann por tanto en una sola lectura puede llevar mayor cantidad de datos.

Arquitectura interna 16F87x

Hemos señalado que el microcontrolador posee varios elementos en forma interna: el procesador, memoria de programa, memoria de datos, periféricos, contadores. Observemos el siguiente diagrama de bloques del PIC16F87X:



Device	Program FLASH	Data Memory	Data EEPROM
PIC16F874A	4K words	192 Bytes	128 Bytes
PIC16F877A	8K words	368 Bytes	256 Bytes

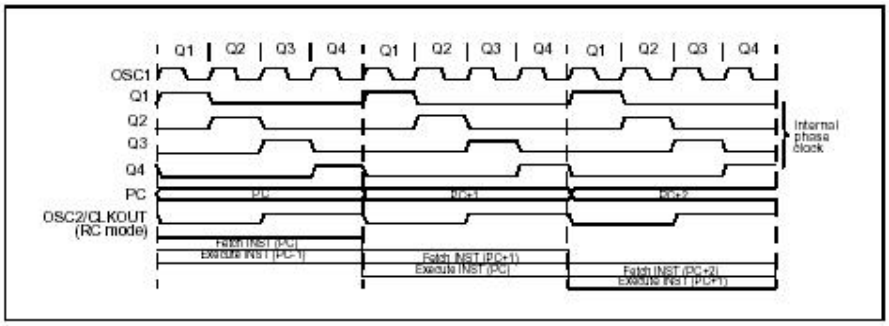
Note 1: Higher order bits are from the STATUS register.

En el diagrama podemos identificar la memoria del Programa en la parte superior izquierda con 8K posiciones por 14 bits, también esta presenta la memoria de datos (RAM) de 368 posiciones por 8 bits. La memoria EEPROM 256 posiciones x 8 bits. El procesador propiamente dicho esta formado por la ALU (unidad aritmetica lógica) el registro de trabajo W. Tenemos los periféricos I/O Port A, B, C, D, E el TMR0 (temporizador contador de eventos), TMR1 y TMR2 entre otros módulos. También contamos con un registro de instrucción que se carga cada vez que la ALU solicita una nueva instrucción a procesar. En la parte intermedia encontramos algunos bloques como son el Status Reg. que es el registro de estado encargado de anotar el estado actual del sistema, cada vez que se ejecuta una instrucción se llevan a cabo cambios dentro del microcontrolador como desborde, acarreo, etc. Cada uno de esos eventos esta asociado a un bit de este registro. Existe un registro de vital importancia que se llama el Program Counter o contador de programa este registro indica la dirección de la instrucción a ejecutar. El registro en cuestión no es necesariamente secuencial, esto es no se incrementa necesariamente de uno en uno ya que puede darse el caso en el que salte dependiendo si hay una instrucción de bifurcación de por medio o puede haber alguna instrucción de llamada a función y/o procedimiento. También observamos el bloque de la pila, la función de la pila es ser un buffer temporal en el que se guarda el contador de programa cada vez que se suscita una llamada a un procedimiento y/o función (incluyendo interrupciones). Por tanto el nivel de anidamineto es de hasta 8 llamadas. También esta presente el FSR reg. que es el registro que cumple una función similar a la del contador de programa direccionando en este caso la RAM, el FSR es un puntero a una dirección de la RAM. La aparición de mutiplexores se debe a que los datos pueden tener diferentes fuentes. Mas adelante explicamos este punto.

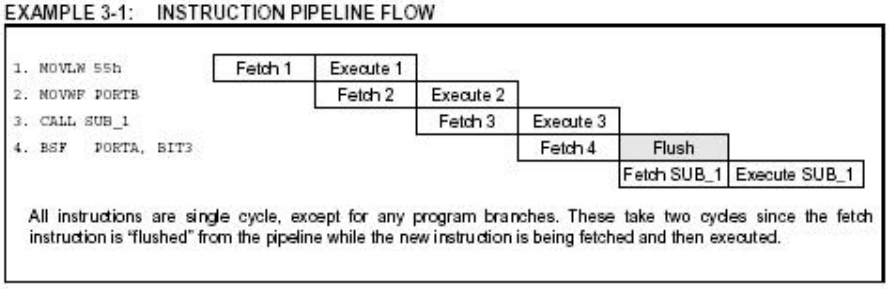
Cuando programamos el microcontrolador debemos siempre tener en mente que es lo que el hace. Cuando lo prendemos asume un valor por defecto, el contador de programa asume la posición cero por tanto el microcontrolador toma la instrucción que se encuentra en esa posición en la memoria de programa y la ejecuta. Al momento de ejecutarla procede a informar si se ha llevado a cabo alguna operación en particular registrándola en el registro de estado (STATUS). Si la instrucción es de salto o bifurcación evaluará las condiciones para saber si continua o no con la siguiente instrucción, en caso que no sea así saltará a otra posición de memoria. En caso el programa haga un llamado a a una función guardará en la pila el valor del contador de programa ejecutará la rutina y al momento que termina restituirá el valor correspondiente para seguir con la siguiente instrucción.

Ciclo de instrucción

Observemos el siguiente diagrama de tiempos:



EXAMPLE 3-1: INSTRUCTION PIPELINE FLOW



All instructions are single cycle, except for any program branches. These take two cycles since the fetch instruction is "flushed" from the pipeline while the new instruction is being fetched and then executed.

El oscilador o cristal genera pulsos, cada uno de los cuales se les llama cuadratura Q1, Q2, Q3 y Q4. Toda instrucción tiene asociada dos procesos uno de búsqueda y otro de ejecución, cada uno de ellos ocupa un ciclo de instrucción. La primera tarea que se lleva a cabo es la búsqueda, internamente el contador de programa se incrementa en Q1. La busqueda de la siguiente instrucción se lleva a cabo en Q4. La instrucción es ejecutada en el siguiente ciclo de instrucción como lo muestra el diagrama.

También debemos notar que los procedimientos de búsqueda y ejecución de una misma instrucción son secuenciales mas son paralelos para diferentes instrucciones, es decir que mientras se ejecuta una instrucción la ALU ya esta buscando la siguiente por tanto se asume que el tiempo de ejecución de una instrucción es un ciclo de instrucción salvo que sea una instrucción de salto o bifurcación en cuyo caso el tiempo es de dos ciclos de instrucción dado que el procesador no sabe hacia donde va a saltar lo normal es introducir un ciclo muerto a fin que evalue la condición necesaria y recién proceda a saltar.

Organización de la memoria

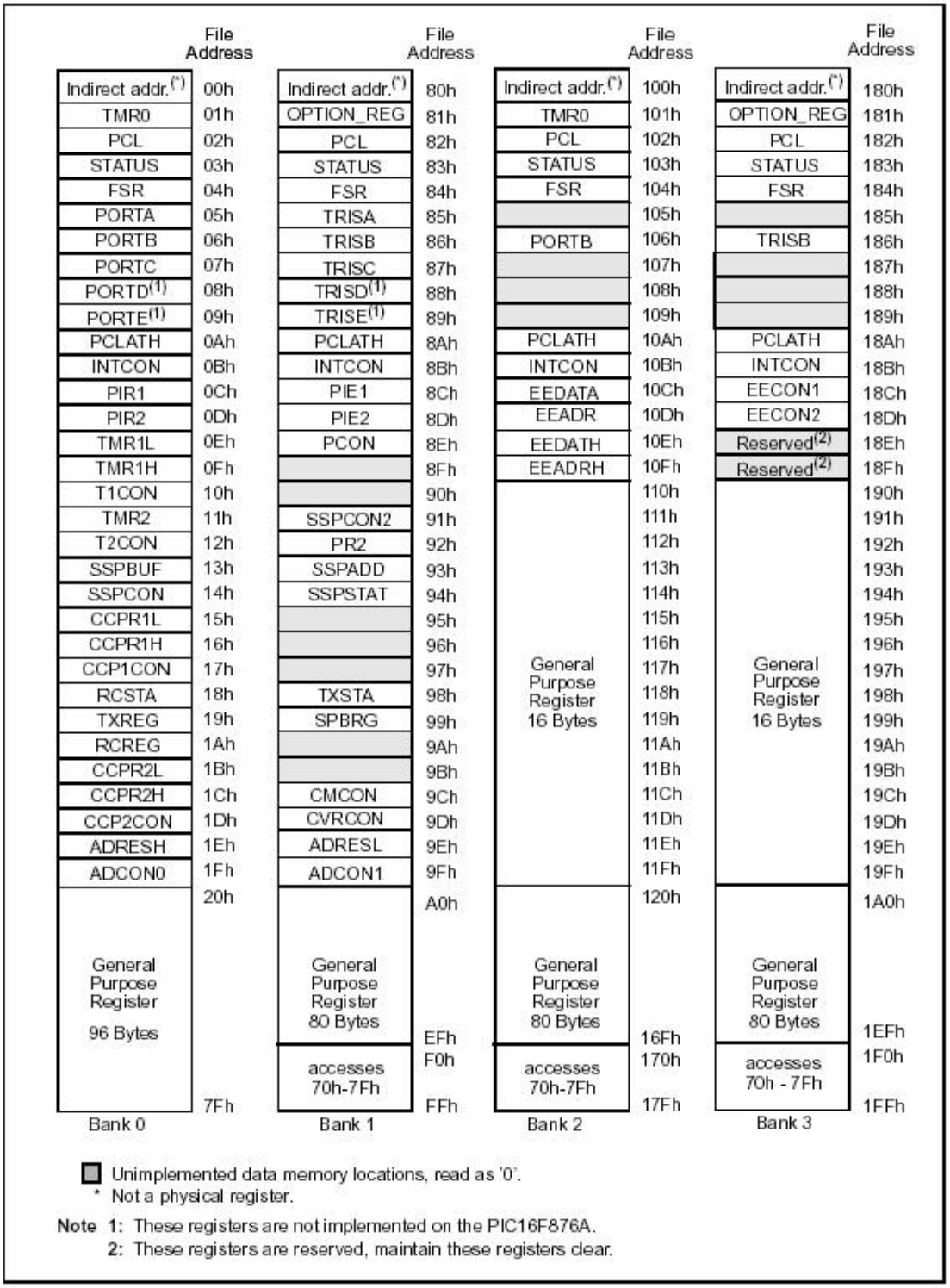
La memoria se divide en memoria de datos y programa. La de datos a su vez se divide en:

- SFR (Special Function Register) Registros de propósito especial, son registros que ayudan a configurar el hardware interno asi como sirven para escribir o leer valores de los diferentes componente que constituyen el microcontrolador. Por ejemplo el registro "trisa" que nos permite configurar el modo de trabajo de las líneas del puerto A

- GFR (General Function register) Registros de propósito general, son posiciones de memoria que podemos usar para almacenar valores que emplean nuestros programa

Debe señalarse también que la memoria de datos se divide en cuatro bancos (esto para el caso específico del 16F87X). Las posiciones bajas siempre estan reservadas para los SFR en tanto que las altas para los GFR.

También tenemos una memoria EEPROM, con 256 posiciones, para acceder a la memoria no podemos leer o escribir directamente es decir colocar la dirección y obtener o dejer el valor. Para trabajarla debemos apoyarnos en registros adicionales de tal forma que la usamos indirectamente



El contador de programa tiene 13 bits con los cuales se pueden direcionar 8K posiciones. Cuando levantamos el microcontrolador el contador de programa siempre apunta a una dirección conocida como el VECTOR DE RESET, la dirección es la posición de memoria

0000h. También existe otro vector llamado de VECTOR DE INTERRUPCIONES que ocupa la posición 0004h. Cuando se lleva a cabo una petición de interrupción el contador de programa va automáticamente a esa posición en busca de las instrucción que atiendan la petición de interrupción. Como se ha mencionado la pila trabaja con el contador de programa cada vez que hay una instrucción de llamada a procedimiento o función (call) el contador de programa se almacena allí y va en busca de la rutina, cuando acaba la rutina (con la ejecución de una instrucción return retfie o retlw) se restituye el valor del contador de programa, la capacidad de la pila es de 8 posiciones en caso tengamos un desborde (ej 9 llamadas anidadas) la pila se dice que se desborda y vuelve a 0. Por tanto hemos de pensar que la pila tambien cuenta con un contador que indica cual es la siguiente dirección vacia.

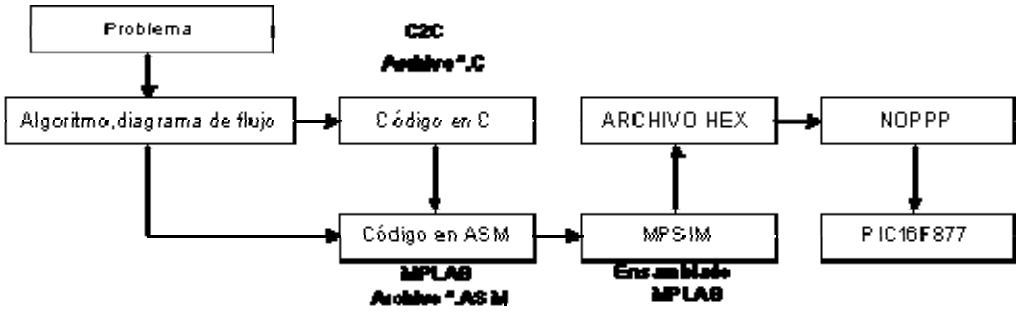
Herramientas

Para programar es necesario contar con herramientas en hardware y software, en el mercado existen muchas herramientas que van de ensambladores a simuladores, emuladores o debuger físicos.

- MPLAB
  El MPLAB es un entorno de desarrollo es decir es un recipiente que incluye varias herramientas:
  contiene un editor de textos que no permite ingresar el programa expresado en códigos nemónico (o simplemente llamado ensamblador), normalmente este se guarda en una archivo con extensión ASM. Una vez que hemos ingresado el programa dentro de un archivo creamos un proyecto dentro del MPLAB el proyecto puede contener a su vez varios archivos ASM que se relacionen a través de llamadas a rutinas o compartan y/o variables, adicionalmente el proyecto tiene un grupo de variables que debemos configurar como es el tipo de microcontrolador que vamos a usar ya que el MPLAB soporta todas las familias de microcontroladores Microchip (MPLAB es producido por Microchip). A continuación procedemos a llamar al programa ensamblador que lleva el MPLAB capaz de transformar los código nemónicos (instrucciones) a los correspondientes valores binarios que a su vez grabaremos en el microcontrolador. El ensamblador (MPASMWIN) también genera otros archivos de salida que ayudan en el diseño de aplicaciones. Dentro del MPLAB encontramos también al MPSIM que es un potente simulador que nos permitirá observar el comportamento del programa antes de proceder grabar el programa. El archivo .HEX es el que contine los códigos binarios a grabar en el microcontrolador. (El MPLAB es un software de libre distribución que se encuentra disponible en el homepage de microchip www.microchip.com)
- ICD
  El ICD (In circuit debugger) es una herramienta que tiene componentes en hardware y en software. El software viene incluido en el MPLAB es de facil uso y configuración. ICD es una característica incluída en algunos microcontroladores de Microchip. Se habilita mediante un bit al momento de grabar el microcontrolador de tal forma que el microcontrolador ejecuta el programa hasta una determinada instrucción, en ese momento el microcontrolador se detiene y procese a transmitir via puerta serial todos los datos que tiene en los bancos de memoria (SFR y los GFR). De esta forma podemos ver en la pantalla del ordenador lo que pasa internamente en el microcontrolador cuando ejecutamos un programa. El hardware es otro componente del ICD consiste en una serie de circuitos que ejecutan la grabación (transistores que trabajan como interruptores en corte y saturacion) asi como un microcontrolador (PIC16F876) que recibe los datos y los transmite a la computadora. Finalmente el ICD se conecta a una tarjeta básica pero efectiva llamada DEMOBOARD. El ICD es una herramienta potente en el sentido que permite corregir rápidamente los errores lógicos que siempre se presentan en la programación.
- GRABADORES

Los grabadores de microcontroladores, toman como entrada un archivo HEX para grabarlo en un microcontrolador. Generalmente los grabadores son herramientas que trabaja con un circuito conectado al puerto paralelo, la idea es tomar el contenido del archivo HEX y depositarlo en la memoria de programa del PIC. Uno de los mas populares es el NOPPP, en internet esta disponible el programa ejecutable, el programa fuente y el circuito. Es un excelente punto de partida para entender como se lleva a cabo el proceso de grabacion de un PIC.

Las herramientas descritas anteriormente trabajan intimamente, si graficamos la forma como se relacionan podriamos expresar la idea a través del siguiente diagrama:



Una vez identificado el problema planteamos la solución a través de un algoritmo de allí en un diagrama de flujo, el diagrama de flujo se puede codificar en cualquier lenguaje de programación de allí que presentamos dos caminos o en ensamblador o en C, si elejimos el C es necesario compilar el código para obtener el equivalente en ASM. Una vez en ASM procedemos a ensamblar y simular. Si todo esta sin problemas procedemos a grabar el programa. El MPLAB genera un archivo HEX que puede ser leído por el NOPPP o el ICD que es el que a su vez graba el programa dentro del microcontrolador. Con el ICD verificamos el correcto funcionamiento del programa si encontramos algún problema procedemos a depurar el error

Módulo 1: Manejo de Puertos Digitales

Registros para el manejo de puertos digitales

El PIC16F877 contine 5 puertos que pueden ser configurados como entrada o salida digitales (A, B, C, D, E). El puerto A contiene 6 bists (RA0-5). El puerto B (RB0-7), el puerto C (RC0-7) y el puerto D (RD0-7) tiene cada uno 8 líneas. El puerto E solo cuenta con 3 líneas (RE0-2)

La operación de configuración de los puertos en general implica la siguiente secuencia:

- Ingresar al banco 1
- Configurar los puertos (registros TRISA, TRISB, TRISC, TRISD y TRISE)
- Regresar al banco 0
- Escribir o leer datos desde los puertos. (registros PORTA, PORTB, PORTC, PORTD y PORTE)

Hemos indicado que la memoria de datos del PIC16F877 se divide en cuatro bancos: 0, 1, 2 y 3. En las posiciones inferiores de ambos bancos se encuentran los registros especiales de función (SFR). En la posición 0x05, 0x06, 0x07, 0x08 y 0x09 respectivamente se encuentran los registros PORTA, PORTB, PORTC, PORTD y PORTE que se usan para leer o escribir datos en tanto que en las posiciones 0x85, 0x86, 087, 0x88 y 0x89 se encuentran los registros TRISA, TRISB, TRISC, TRISD y TRISE respectivamente, es allí donde se configuran los puertos. Cabe señalar que el PORTB también aparece en el banco 2 en la posición de memoria 0x106 y el TRISB en la posición de memoria 0x186.

Posmem	**Banco 0**	**Banco 1**	**Posmem**
	.....	.....	
0x05	PORTA	TRISA	0x85
0x06	PORTB	TRISB	0x86
0x07	PORTC	TRISC	0x87
0x08	PORTD	TRISD	0x88
0x09	PORTE	TRISE	0x89
	.....	.....	
		ADCON1	0x1F

Cada una de las líneas de los puertos puede ser configurado como entrada o como salida. En el registros TRIS determinamos la configuración de los puertos. Los registros son una suerte de mascara. Por ejemplo si escribimos un 0 en el bit 0 del TRISA la línea RA0 se comportará como una línea de salida. Si colocamos a 1 el bit 0 del TRISA a la línea RA0 se comportará como entrada.

A través de los valores que escribamos en los registros TRIS determinamos el comportamiento de los puertos.

La escritura y lectura de valores desde los puertos se hace a través de los registros PORT que se encuentran en el Banco 0 (y banco 2 para el puerto B). Desde luego si configuramos un puerto como entrada (lectura) los valores que escribamos en el no tendrán efecto porque fue configurado como entrada y no como salida. A través de las instrucciones MOV podemos leer o escribir valores.

NOTA.- El puerto A es un puerto multifunción que se puede configurar como digital o como analógico este modo de funcionamiento dependerá del registro ADCON1 (banco 1 posicion 0x1F). Por el momento no profundizaremos en el tema sino cuando llegemos al módulo ADC. Solo nos bastará saber que el debemos configurar los bits de la siguiente manera:

Registro ADCON1 (Banco 1 posición 0x1F)

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
X	X	X	X	0	1	1	X

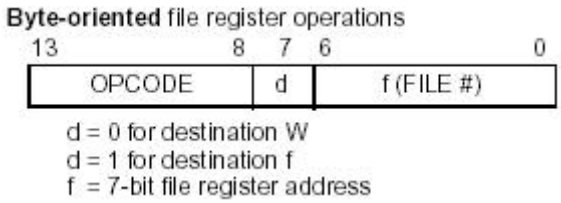
Descripción general de las instrucciones

Los programas estan compuestos por instrucciones. El PIC16F877 cuenta con 35 instrucciones que iremos desarrollando conforme avancemos en el curso. Cada instrucción esta representada por 14 bits. Los 14 bits a su vez se dividen en:

- Código de operación (OPCODE), que especifica cual es la instrucción a la que hacemos referencia, por ende cada instrucción tiene un código en particular.
- Operadores, cada instrucción es aplicada sobre determinados operadores, parte de los 14 bits están destinados a especificar quienes son los registros o valores que se veran afectados como resultado de la aplicación de la instrucción.

Las instrucciones estan divididas en tres clases:

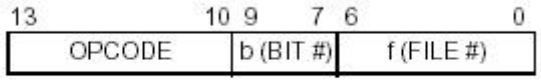
Orientadas a byte Instrucciones cuya representacion es:



Las instrucciones orientadas a byte reservan los 7 bits de menor peso para indicar la dirección del registro que será operado. Una vez que se lleva a efecto la operación usamos el bit d para indicar donde será almacenado el resultado. Si d es 0 el resultado se almacena en el registro de trabajo W, si d es 1 el resultado será guardado en el mismo registro (o file) que se opero.

Orientadas a bit Representada por:

Bit-oriented file register operations

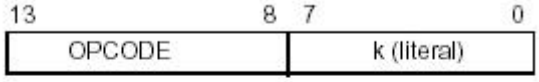


b = 3-bit bit address
f = 7-bit file register address

Las operaciones orientas a bit buscan escribir o leer una posición (bit) dentro de un file o registro. Una vez mas los 7 bits inferiores son destinados para indicar la dirección de registro o file que vamos a trabajar y los siguientes tres bit especifican el bit dentro del registro.

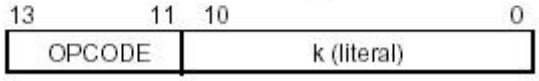
Literales o de control Con formato:

General



k = 8-bit immediate value

CALL and GOTO instructions only



k = 11-bit immediate value

Las instrucciones de control son las que ayudan a formar bucles dentro de los programas asi como sirven para llamar a rutinas o procedimientos (instrucciones CALL o GOTO). En este caso en particular se emplea los 11 bits inferiores para enviar la dirección a la cual el contador de programa (PC) saltará. Los bits superiores de la instrucción sirven para identificar a la instrucción.

En ocasiones es necesario cargar constantes a los registros del microcontrolador, las instrucciones literales nos sirven para mover las constantes a un registro en particular, en este caso empleamos los 8 bits inferiores para definir la constante que deseamos almacenar, en tanto que los bits restantes sirven para identificar la instrucción.

El siguiente es un cuadro resumen de las instrucciones clasificadas en función a las categorías que hemos descrito:

Mnemonic, Operands		**Description**	**Cycles**	**14-Bit Opcode**				**Status Affected**	**Notes**
				MSb			**LSb**		
		BYTE-ORIENTED FILE REGISTER OPERATIONS							
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2

MOVWF	f	Move W to f	1	00	0000	lfff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
		BIT-ORIENTED FILE REGISTER OPERATIONS							
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
		LITERAL AND CONTROL OPERATIONS							
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	TO,PD	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into Standby mode	1	00	0000	0110	0011	TO,PD	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

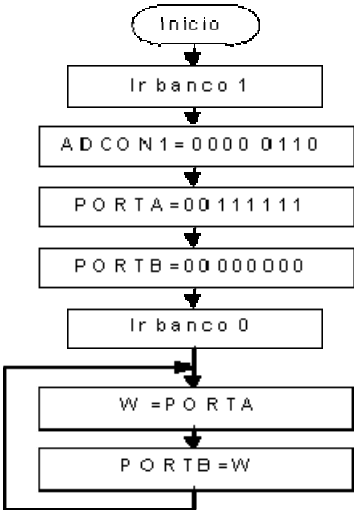
Lectura y escritura en puertos

a) Desarrollar un programa que configure las líneas del puerto A como entrada y las líneas del puerto B como salida. Y que muestre en forma permamente la entrada del puerto A en el puerto B.

Si desarrollamos el algoritmo se reduce a :

1. Configurar PA como entrada y PB salida
2. W= PA
3. PortB=W
4. Ir paso 2

El diagrama de flujo se resume



Partes de un programa en ASM

DIRECTIVAS

Antes de codificar el diagrama anterior es conveniente revisar algunos conceptos breves de las partes que componen un programa en ensamblador para el PIC16F877 usando el MPLAB. Ademas de las instrucciones que necesitamos es necesario revisar las directivas de compilación que son comandos que permiten mejorar la programación.

- **Directiva ORG**

[<etiqueta>] ORG <exp>

Sirve para indicar la direccion de memoria en la cual será colocada el código generado a continuación . Si el ORG no es indicado se empieza en la dirección 0. Ejemplo

ORG 0x04
nop

Indica que el siguiente "nop" se colocará en la dirección 0x04 de la dirección de programa.

- **DIRECTIVA EQU**

<identificador> EQU <expresion>

Permite asignar el valor de expresión al identificador. El general el identificador es un nombre que le es mas familiar al programador. Ejemplo

CONF_ADCON1 EQU b'00000110'

Crea el identificador CONF_ADCON1 con valor 0x06

- **DIRECTIVA END**

END

Es de uso obligatorio y siempre se coloca al final del programa sirve para marcar el final del programa. El MPLAB solo reconoce las líneas que esten escritas previas a la aparición de la direcitiva END.

- **DIRECTIVA LIST**

Sirve para indicarle al MPLAB cual es el formato del archivo *.list dentro de los parámetros esta el tipo de procesador que se va a emplear. Ejemplo:

list p=16F877

- **DIRECTIVA INCLUDE**

include <file>

Sirve para incluir en el ensamblado el archivo indicado por el parametro “file”. Es como si el “file” fuera parte del archivo, como si se hubiera situado en la posición en la cual la directiva aparece. El parámetro “file” puede incluir el path o camino en el cual se encuentra el fichero a incluir. En caso se omita asumirá los directorios del MPLAB y del archivo fuente. Ejemplo

include <p16f877.inc>

Incluye el archivo “p16F877.inc” que contiene las etiquetas genéricas del PIC16F877

- **PRIMERAS INSTRUCCIONES**

BSF **Bit Set f**
Sintaxis: [*label*] f,b
BSF
Operandos: 0 □ f □ 127
0 □ b □ 7
Operación: 1 □ (f<b>)
Afecta Status: No
Descripción: El bit 'b' del registro 'f' es puesto a 1-lógico.

BCF **Bit Clear f**
Sintaxis: [*label*] f,b
BCF
Operandos: 0 □ f □ 127
0 □ b □ 7
Operación: 0 □ (f<b>)
Afecta Status: No
Descripción: El bit 'b' del registro 'f' es puesto a o-lógico.

CLRF Clear f
Sintaxis: [*label*] CLRF f
Operandos: 0 □ f □ 127
Operación: 00h □ (f)
1 □ Z
Afecta Status : Z
Descripción: El contenido del registro ’f’ es puesto a 0-lógicos y el bit Z del STATUS es puesto a 1-lógico.

GOTO Unconditional Branch
Sintaxis: [*label*] GOTO k
Operandos: 0 □ k □ 2047
Operación: k □ PC<10:0>
PCLATH<4:3> □ PC<12:11> Status Afecta Status: No
Descripción: GOTO es un salto incondicional.
Los once primeros bits son cargados en el registro PC bits <10:0>. The bits superiores de PC son cargados de PCLATH<4:3>. GOTO es una instrucción que demora dos ciclos de instrucción.

MOVLW Move Literal to W
Sintaxis: [*label*] MOVLW k
Operandos: 0 □ k □ 255
Operación: k □ (W)
Afecta Status: No
Description: Los ocho bits literales de 'k' son cargados dentro del registro W.

MOVWF Move W to f

Sintaxis: [*label*] MOVWF f
Operandos: 0 □ f □ 127
Operación: (W) □ (f)
Afecta Status: No
Descripción: Mueve el dato del registro W al registro 'f'.

Con las directivas y las instrucciones mostradas procedemos a elaborar el código del primer programa.

Primer programa en ensamblador:

```
    list p=16F877
    include "p16f877.inc"

    CONF_ADCON1 equ b'00000110' ; Configuracion PORTA E/S digital

org 0x000 ; Origen del codigo
    nop ; No operacion
    nop ; No operacion

    bsf STATUS,RP0 ; Ir banco 1
    bcf STATUS,RP1

    movlw CONF_ADCON1 ; Configurar el PORTA como digital
    movwf ADCON1

    movlw b'00111111' ; PORTA como entrada
    movwf TRISA
    clrf TRISB ; PORTB como salida

    bcf STATUS,RP0 ; Ir banco 0
    bcf STATUS,RP1

BUCLE
    movf PORTA,W ; W=PORTA
    movwf PORTB ; PORTB=W
    goto BUCLE ; Ir bucle

END ; Fin del programa
```

Uso del MPLAB

a) Nociones previas

MPLAB. Es un entorno de desarrollo que incluye varias herramientas que ayudan a desarrollar aplicaciones en torno a microcontroladores de la familia Microchip. Incluye un editor, un ensamblador (MPASMWIN), un simulador (MPSIM), el software del PICSTART (programador), software para el ICD (in circuit debugger).

PROYECTO. Es un recipiente que contiene los archivos con el código de la aplicación. Un proyecto puede contener uno o mas archivo de código.

SIMULACIÓN. Acción por la cual podemos observar en pantalla el desarrollo de programa como si estuvieramos dentro del microcontrolador. También podemos manipular las señales de entrada. Es necesario indicar que todo lo que vemos se realiza en la memoria del computador, a diferencia de los emuladores (ICD) que nos presentan los datos que hay dentro del dispositivo.

c) Pasos a ejecutar en MPLAB

Cada vez que usamos el MPLAB para programar apliaciones debemos ejecutar la siguiente secuencia de pasos:

1. **Ingresar al MPLAB:**
   Inicio-> Programas-> Microchip MPLAB-> MPLAB

o haga doble click sobre el icono del programa en el escritorio:

2. Cree un proyecto:

Primero con ayuda del explorador de WINDOWS cree una carpeta de trabajo:

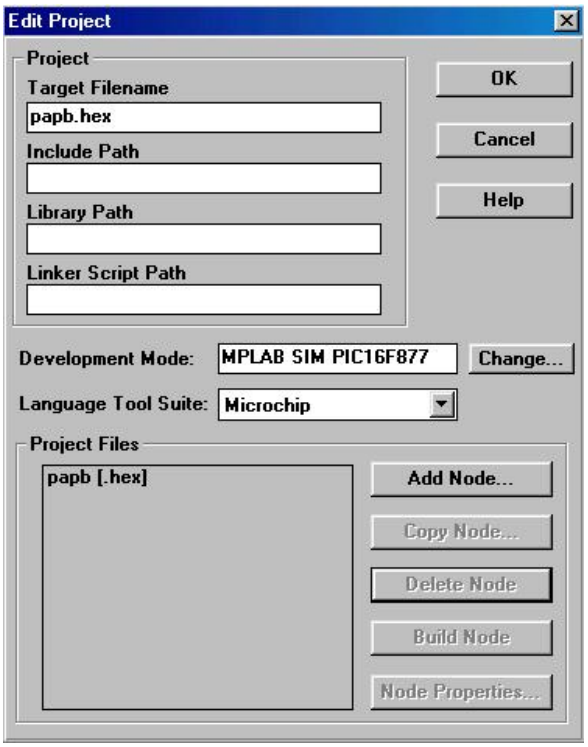
c:\Archivos de Programa\MPLAB\CURSO

La idea es contar con una carpeta en la cual coloquemos nuestros trabajos. El MPLAB cuenta, como toda aplicación en WINDOWS, con una barra de menus, una barra de iconos y una de estado en la parte inferior.

Abra el menu **PROJECT**y elija la opción **NEW**.

Asigne un nombre al projecto (por ejemplo **papb**) y asegurece que el proyecto sea creado en la carpeta CURSO ademas el campo **DEVELOPMENT MODE** debe estar con la opción **MPLAB-SIM 16F877**. Como lo muestra la siguiente figura:

Presione el boton **OK**.



3. Edite el programa

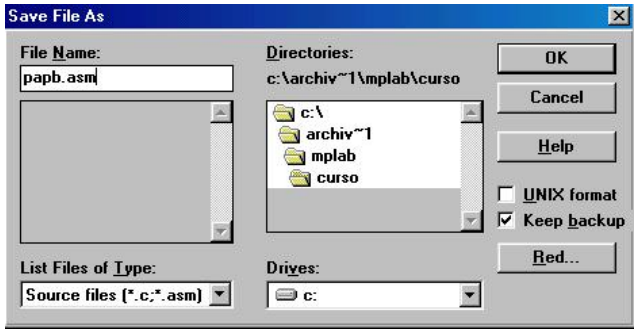
Ahora que contamos con el recipiente el siguiente paso consiste en adicionar el código; para eso nos apoyaremos en el editor. Abra un nuevo archivo:

Ingrese al menu **EDIT** y elija la opcion **NEW**.

Notara como se muestra un documento nuevo. Ingrese el codigo en ensamblador que necesita como lo muestra la siguiente figura:

```
        list p=16F877
        include "p16f877.inc"

CONF_ADCON1 equ b'00000110'

        org     0x000
        nop
        nop

        bsf     STATUS,RP0
        bcf     STATUS,RP1

        movlw   CONF_ADCON1
        movwf   ADCON1
```

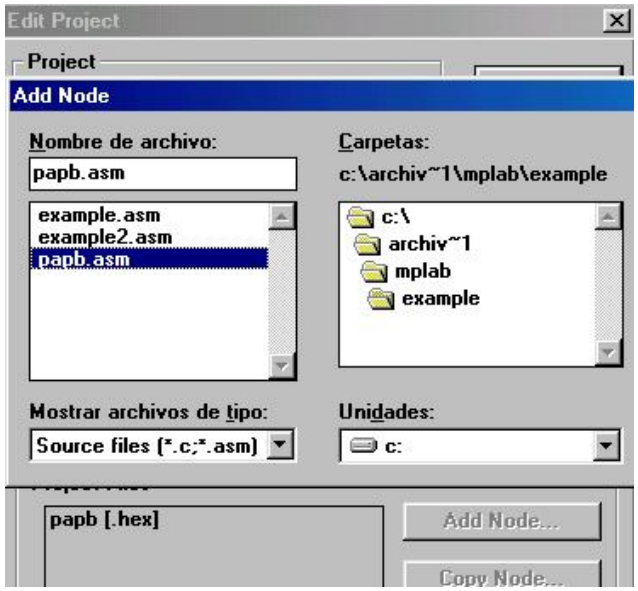
Ahora guarde el archivo. Asegúrece que se cree en la misma carpeta donde esta el proyecto (CURSO) coloque un nombre (**PAPB.ASM**) con extensión asm:



Ahora incluya el archivo "papb.asm" como parte del proyecto .

Menú **PROJECT** y elija la opción **EDIT PROJECT**

Haga click en el boton **ADD NODE ,** use la ventana para seleccionar el archivo "papb.asm"



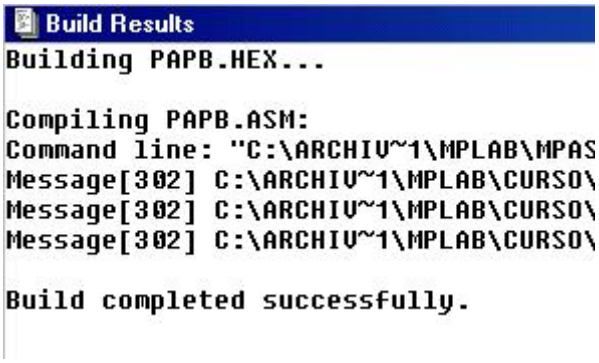
4. Ensamble el programa

Una vez que el programa esta listo llamamos al ensamblador (MPASMWIN):

Menú **PROJECT** y elija la opción **BUILD ALL**.

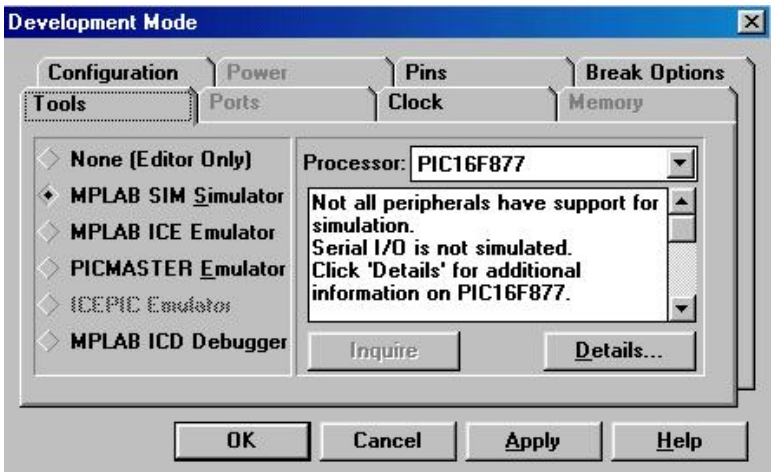
Si el código esta libre de errores aparecerá una ventana similar a la siguiente.

Si ha cometido algún error de sintaxis el MPASMWIN le indicara en una ventana la línea y el error a fin que lo solucione. Corrijalo y vuelva a compilar hasta que no haya problemas.

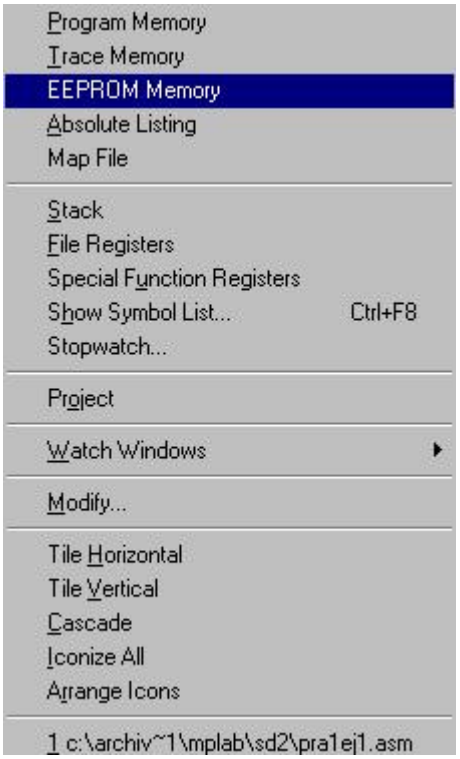


5. Mostrar los datos relevantes

Antes de entrar al modo de simulación debe asegurarse que el proyecto tiene habilitado el simulador (MPSIM). Ingrese al menú OPTIONS elija DEVELOPMENT MODE



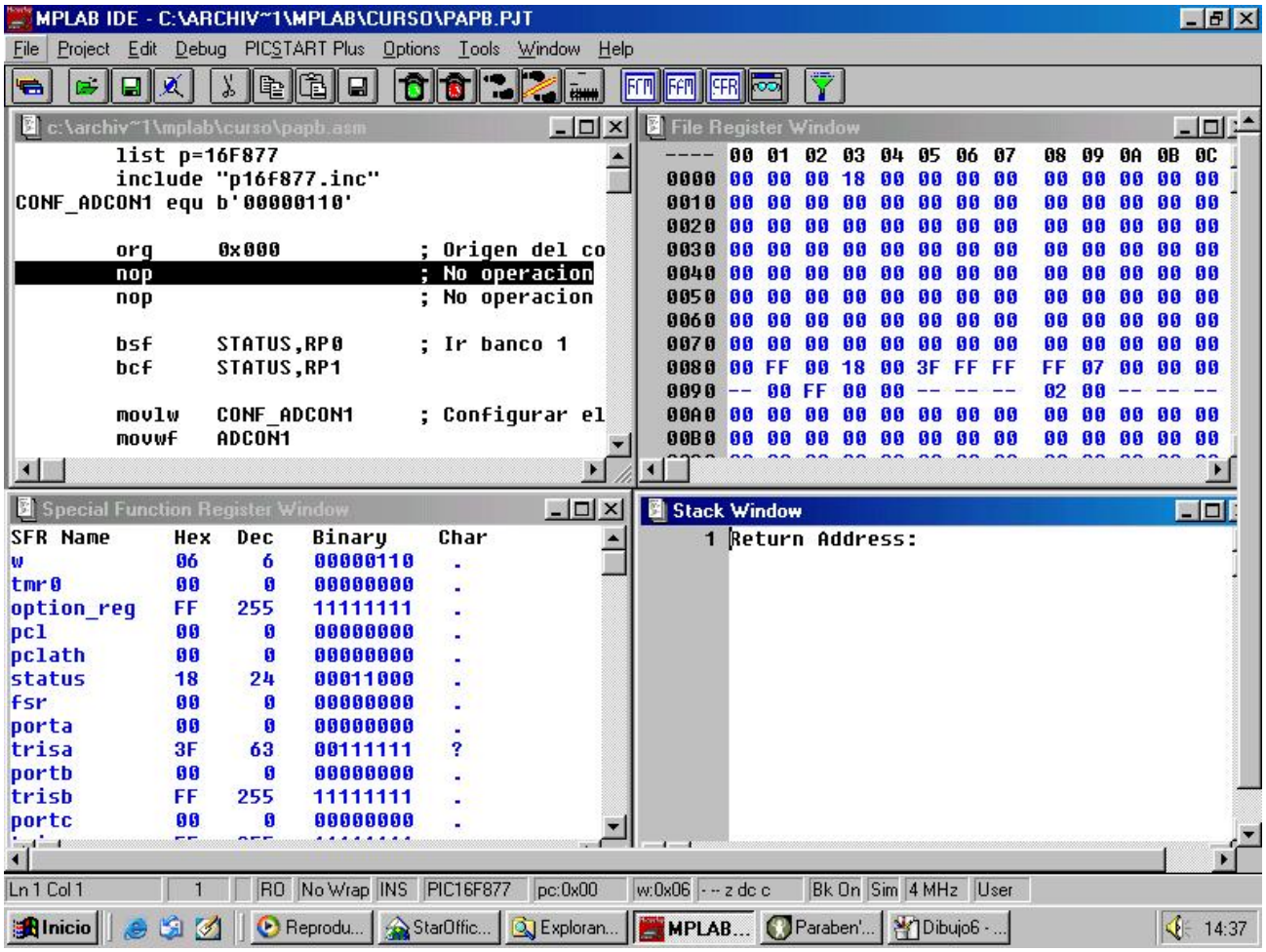
Para observar lo que va a suceder en el microcontrolador debemos abrir las ventanas que nos muestran los datos relevantes para ello ingrese al menú **WINDOWS** observará las siguiente figura:



Por el momento solo habilitaremos las siguientes ventanas:

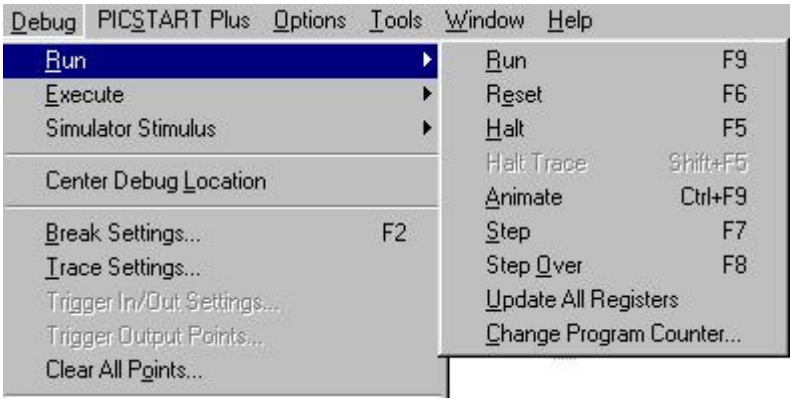
- Special Function Register: Nos muestra los registros de configuración del microcontrolador
- File Register. Nos muestra la zona de memoria de datos (GFR+SFR)
- Stopwatch: Muestra un clock para la evolución del programa paso a paso
- Stack:Muestra la pila

Una vez abiertas las ventanas ordenelas en la pantalla (workspace) a fin que se vean a la vez:



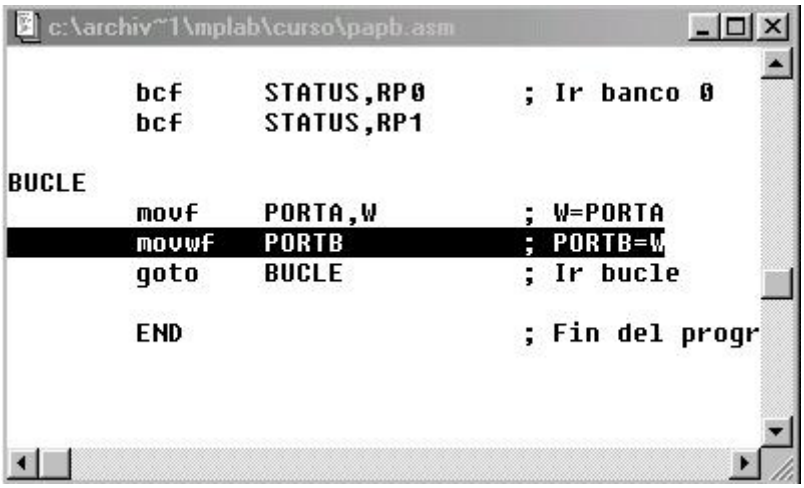
6.Simulación

La simulación propiamente dicha se hace através del menú **DEBUG**:



Recuerde que el microcontrolador tiene un registro llamado **PC** (contador de programa) que le indica que instrucción debe ejecutar (puede ver parte del valor en la ventana **Special Function Register - pcl**). Cada uno de los item dentro del submenu **RUN** le indican al simulacion que debe hacer con el contador de programa.

Si presiona la opción **RESET (F6)** el programa se detiene y el contador de programa se va a 0 (vector de reset). Vamos a ejecutar el programa instrucción por instrucción. Para ello ejecute la opción **STEP (F7)** note como es que aparece un cursor en la ventana de editor y algunos valores de las demas ventanas se han modificado (los valores que se han modificado siempre aparecen en color rojo). Para continuar con la simulación paso a paso vuelva a ejecutar la opción **STEP (F7)** y observe que pasa en la pantalla.



Ahora resetee el programa **(**opción **RESET -F6).**

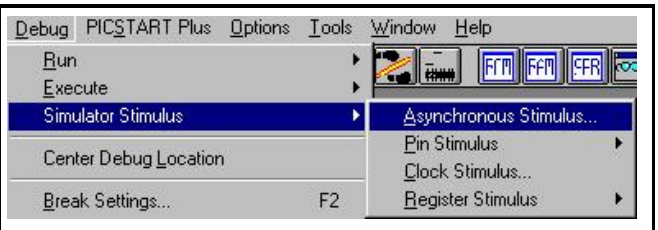
Otra forma de ver como es la evolución del programa sin necesidad de apretar la secuencia paso a paso es através de la opción ANIMATE. Ejecutela y observe que es lo que pasa.

Para salir del estado de animación es necesario usar la opción **HALT (F5)** del submenú run o presione el icono del semáforo rojo.

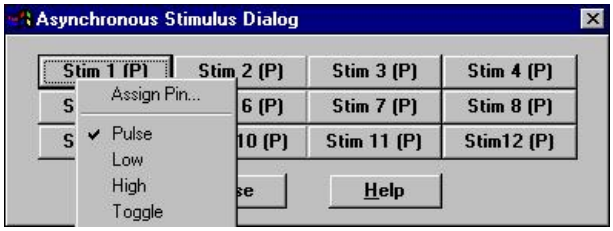
También contamos con una opción que hace que el microcontrolador corra el programa libremente esa es la opción **RUN (F9)** o presione el icono del semáforo verde, cuando ejecute esta acción note como la barra de estado (parte baja de la pantalla) cambia de color a amarillo. Para salir elija nuevamente la opción HALT.

7. Modificando las entradas

El MPLAB cuenta con opciones que nos permiten variar las entradas durante la simulación a fin de observar el comportamiento del programa. Ingrese al menú **DEBUG** y seleccione el submenú **SIMULATOR STIMULUS**, observará que presenta cuatro opciones. Por el momento trabajaremos con la primera. Elija **ASYNCHRONOUS STIMULUS**.



Observará la siguiente ventana:

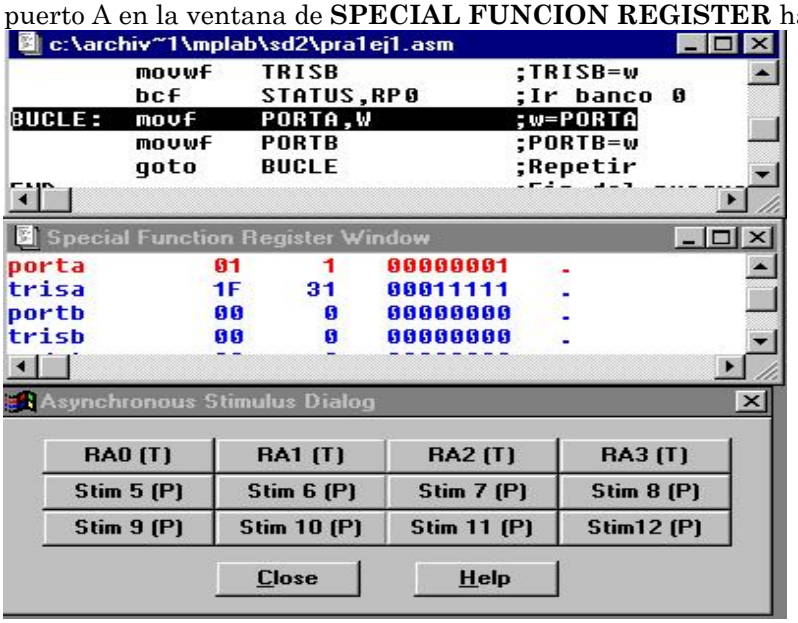


Podemos asignar a cada boton (de los 12 disponibles) uno de los pines del PIC. Haga click con el boton derecho del mouse sobre alguno de los botones (menú de contexto) podemos asignar el botón a un pin y además definirá el tipo de estímulo.

- **Pulse,** equivale a ingresar un pulso en el pin
- **Low**, equivale a colocar 0 en el pin
- **High**, equivale a colocar 1 en el pin
- **Toggle**, es un interruptor que oscila entre 1 y 0. Si lo presiona una vez ira a 0 si lo presiona de nuevo irá a 1 y asi sucesivamente.

Tome 4 botones y asignelos a RA0, RA1, RA2 y RA3 asegúrece que sean del tipo Toggle, como lo muestra la figura adjunta. Note como la ventana queda flotando no la cierre ubiquela en alguna zona de la pantalla que no estorbe la visibilidad de la pantalla.

Para probar que funcioná ingrese al menú **DEBUG** elija **STEP** y continue hasta que el programa entre en el bucle de lecto escritura. Haga un click sobre uno de los botones recien creados y vuelva a avanzar en la simulación (presione F7). Notará como es que el valor del puerto A en la ventana de **SPECIAL FUNCION REGISTER** ha variado:



Modifique el estado de cada uno de los botones y simule el programa para ver que sucede. También es posible combinar el **ASINCHONUS STIMULUS** con la opción **ANIMATE** del submenú **DEBUG**. Para ello resetee el programa, a continuación active la opción **ANIMATE** (mantega abierta la ventana de **ASINCHRNOUS STIMULUS**). Ahora haga click sobre los botones asignados a RA0-4.
NOTA.- Las demás opciones de simulación las iremos usando en las siguentes prácticas.

Observaciones

Las instrucciones de escritura en puertos siempre son precedidas de una operación de lectura. Es decir ponemos el valor del registro PORTA o PORTB en el registro de trabajo W. Allí modificamos su valor (podemos variar uno o más bits) y luego llevamos el valor de W al registro del puerto. Esto se hace por ejemplo en la ejecución de las instrucciones BCF y BSF.

Por tanto debemos tener cuidado con aquellos puertos cuyas líneas son configuradas como entrada y salida a la vez. Por ejemplo: una instrucción bsf PORTB,5 hace que W =PORTB luego en bit 5 de W se pone a 1 y finalmente W es llevado a PORTB. Si otro bit del PORTB es definido al inicio del programa como entrada y en el desarrollo del programa es reconfiguardo como salida, la línea no necesariamente coincidirá con el valor que habia antes, por tanto se pierde.

Si observamos el diagrama de tiempos de las instrucciones de escritura en puerto podemos observar que se llevan a cabo al final del ciclo de instrucción, en tanto que la lectura se lleva a cabo al inicio del ciclo de instrucción. Por otra parte los dispositivos físicos que conforman los puertos son semiconductores con un tiempo de respuesta pequeño exite un periodo de tiempo en el cual la señal está pasando de 1 a 0 y viceversa (transitorio). Por tanto hemos de ser cuidadosos en la aplicacion de instrucciones sucesivas de escritura en el puerto por que lo que hacemos es aplicar una lectura luego una escritura (primera instrucción) inmediatamente volvemos a leer el puerto y aplicamos una escritura (segunda instrucción). Note como la escritura de la primera instrucción esta seguida inmediatamente de la lectura. Si ese periodo no es suficientemente grande podriamos estar leyendo valores erroneos en la segunda instrucción debido a que el transitorio aun no ha finalizado. Para evitar el problema es recomendable colocar instrucciones nop (no operación) de por medio.

Ejemplo:

bcf PORTB,7 bcf PORTB,6 bsf STATUS,RP0 bcf TRISB,7 bcf TRISB,6	bcf PORTB,7 nop bcf PORTB,6 nop bsf STATUS,RP0 nop bcf TRISB,7 nop bcf TRISB,6

La introducción de las instrucciones de “no operación” (nop) no hacen mas que crear un espacio de tiempo para leer el dato después que el transitorio ha finalizado. Si la frecuencia de funcionamiento del clock es muy alta conviene ubicar mas instrucciones nop a fin de evitar la pérdida de data.

Principales registros del PIC16F877 (STATUS)

Hemos indicado que la memoria del datos del microcontrolador se divide en bancos de memoria, las posiciones inferiores estan destinadas a los registros especiales de función (SPECIAL FUNCTION REGISTER). En esta sección profundizaremos un poco mas acerca de los principales registros y observaremos el uso que se les puede dar en el desarrollo del programa.

El registro de Estado (STATUS)

El STATUS es un archivo o registro que ocupa la posición la posición 0x03 de los bancos de memoria:

Banco0		**Banco 1**		**Banco 2**		**Banco 3**	
Status	0x03	Status	0x83	Status	0x103	Status	0x183

El STATUS es un registro del microcontrolador que almacena información relacionada con:

- La última operación aritmética lógica realizada en la ALU
- El estado de reset del microcontrolador
- El banco de memoria que actualmente se tiene en uso

El STATUS contine los siguientes bits:

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
IRP	RP1	RP0	-T0	-PD	Z	DC	C

Bit 7: IRP Register Bank Select Bit (es un bit que se usa para las operaciones de direccionamiento indirecto

0 = Si se trabaja sobre el banco 0 ó 1 (posiciones de memoria que van desde 00h hasta FFh)
1 = Si trabajamos con el banco 2 ó 3 (posiciones de memoria que van desde 100h hasta 1FFh)

Bit 6-5: RP1, RP0 Register Bank Select (bits usando en el direccionamiento directo)

00 = Banco 0 , (posiciones de memoria 00-7Fh)
01 = Banco 1, (posiciones de memoria 80-FFh)
10 = Banco 2, (posiciones de memoria 100-17Fh)
11 = Banco 3, (posiciones de memoria 180-1FFh)

Los banco pueden contener hasta 128 posiciones.

Bit 4: -T0 Time out bit

1 = Asume el valor de 1 después de encenderse el PIC o por la aplicación de la instrucción CLRWDT o por la aplicación de la instrucción SLEEP
0 = Cuando se ha vencido el periodo programado en el Watchdog

Bit 3 : -PD Power down bit

1 = Después de encender el microcontrolador o por la aplicación de una instrucción CLRWDT
0 = Cuando se ejecuta la instrucción SLEEP

Bit 2: Z Zero Bit

1 = Cuando el resultado de una instrucción aritmética lógica da por resultado 0.
0 = Si el resultado de la operación aritmética o lógica da por resultado un valor distinto de cero

Bit 1: DC Digit carry/borrow bit usado como acarreo en las instrucciones de suma (ej: ADDWF y ADDWL) en caso se lleve a cabo una operación de resta se procede a tomarlo como bit de préstamo). Este bit trabaja con los 4 bits inferiores o nible bajo.

1 = Si se ha producido el acarreo en el nible bajo
0 = No se ha producido acarreo en el nible bajo

Bit 0: C: Carry/borrow bit Similar al anterior con la diferencia que toma el acarreo de todo el registro es decir trabaja en 8 bits.

1 = Si se ha producido el acarreo en el nible bajo
0 = No se ha producido acarreo en el nible bajo

Los tres bits que se encuentran en la parte inferior son bits que reportan el estado de la ALU tras la ejecución de una instrucción. Estos bits son de lectura. Los bits pueden ser de escritura si y solo si la instrucción no afecta el estado de los bits Z, DC y C. Las instrucciones que no afetan al STATUS son BCF, BSF, SWAP y MOVWF. Si quisieramos colocar todos los bits del STATUS a 0 lógico la aplicación de la instrucción CLRF STATUS fallaría por que la instrucción afecta a los bis Z, DC y C. De hecho los bits C y DC conservarían el valor previo a la aplicación de la instrucción CRLF STATUS en tanto que el bit Z se colocaría a 1 por que el resultado de la última operación fue un 0.

Los bits que se encuentran en medio (-T0 y -PD) son usados para registrar si el microcontrolador esta trabajando y cual es el modo de operación. El microcontrolador puede estar encendido y a su vez tiene dos modos de operación: activo y reposo (o bajo consumo). Los bits también reportan el estado del WATCHDOG, cuando el periodo de tiempo del WATCHDOG se ha vencido el bit -T0 se coloca a 1. Es necesario indicar que ambos bits son solo de lectura y se modifican en función al modo de operación de microcontrolador (visto desde el punto de vista de la alimentación o consumo de energía). En consecuencia la aplicación de la instrucción CLRF STATUS fallaría también por que no es posible modificar el estado de los bits.

Los bits localizados en la parte superior del registro STATUS son relacionados al uso de los bancos la combinación de los mismo determina cual es el banco actual con el que estamos trabajando. Las instrucciones recomendables para manipular los bits son BSF y BCF. Los bits superiores son de lectura y escritura.

Observaciones

Cuando se aplican las instrucciones de rotación de bits a la izquierda o derecha (RLF ó RRF) el bit de C es el valor que se emplea para llenar el agujero producido por la instrucción, motivo por el cual es necesario colocar el bit C a cero o uno lógico dependiendo si deseamos que la posición libre sea cubierta por uno de los dos valores.

La instrucción SWAPF no modifica el estado de los bits Z, DC y C pero eso no implica que la aplicación de la instrucción sobre el STATUS vaya a lograr el intercambio de los nibles. Al aplicar las instrucciones seguramente los bits IRP, RP1, RP0, Z, DC y C se intercambiarán uno a uno pero los bits -T0 y -PD no seran intercambiados ya que son de lectura.

Cuando una instrucción es ejecutada puede afectar el estado del STATUS, este hecho resulta aparentemente irrelevante mas la ayuda que presta es valiosa en la elaboración de soluciones por ejemplo:

Considere la instruccion:

movf REGISTRO,1. (o lo que es lo mismo movf REGISTRO, F)

La instrucción lleva el contenido desde el file REGISTRO hacia el W (registro de trabajo) y nuevamente lo deja donde lo encontró (REGISTRO). Esta operación parece intracendente pero es una forma bastante práctica y simplificada de preguntar si el valor contenido en el file REGISTRO es 0. Porque recordemos que el bit Z se pone a 1 si la última instrucción aplicada en la ALU dio como resultado 0. En este caso en particular solo se movio entonces si se movio un 0 y se devolvio a su posición seguramnente el bit Z será colocado 1.

Ejercicios

Ejercicio 1: Suponga una lampara que debe ser prendida o apagada desde tres puntos. Diseñe un programa que la encienda si y solo si hay dos interruptores activados.

Supongamos que tenemos los tres interruptores dispuestos en el PORTB (RB0,RB1 y RB2) y definimos la línea RB3 como salida tendríamos la siguiente tabla de verdad:

OUT	IN		
RB3	RB2	RB1	RB0
0	0	0	0
0	0	0	1
0	0	1	0
1	0	1	1
0	1	0	0
1	1	0	1
1	1	1	0
0	1	1	1

En función a la tabla de verdad anterior podriamos establecer el siguiente algoritmo:

1. Configurar RB0-2 como entrada digital y RB3 como salida digital

2. W=PB
3. Si (W = 0x03) o (W= 0x05) o (W= 0x06) RB3=1
4. sino RB3=0
5. Ir paso 2

a) Definiciones previas.

Como se puede apreciar en el algoritmo anterior es necesario contar con instrucciones que nos permitan realizar bifurcaciones en el programa. Para ello revisaremos las instrucciones de control de salto y bifurcación.

BTFSS Bit Test f, Skip if Set

Sintaxis: [etiqueta] SBTFSS f,d
Operandos: 0 <= f <= 127
0<= b <= 7
Operación: Salta si es (f<b>)=1

Bits afectados en el STATUS: None

Descripción: Si el bit 'b' del registro 'f' es 0 entonces la siguiente instrucción es ejecutada . Si el bit 'b' del registro 'f' es 1 la siguiente instrucción es descartada y una instrucción NOP es ejecutada en su lugar lo que ocasiona que esta instrucción ocupe 2 ciclos de instrucción.

Ejemplo HERE BTFSS FLAG, 1
FALSE GOTO PROCESS_CODE
TRUE .....

Antes de la instrucción : PC= la dirección de la etiqueta HERE
Despues de la instrucción: si FLAG<1> =0 PC=dirección de la etiqueta FALSE
si FLAG<1> =1 PC=dirección de la etiqueta TRUE

BTFSC Bit Test f, Skip if Clear

Es la instrucción complementaria a la anterior, la sintaxis y demas valores son exactamente los mismos solo que en este caso el salto se produce cuando el bit 'b' del registro 'f' es 0.

SUBWF Substrae W de f

Sintaxis: [etiqueta] SUBWF f,d
Operandos: 0 <= f <= 127 d= [0,1]
Operación: (f) – (W)-> (destino)

Bits afectados en el STATUS: C, DC, Z

Descripción: Substrae (usa en método de complemento a 2) el registro W del registro 'f'. Si 'd' es 0el resultado es almacenado en W. Si 'd' es 1el resultado es almacenado devuelta en el registro 'f'.

Creación de una variable
La mayor parte de los programas manejan datos, los datos son guardados en variables. Las variables siempre se crean en una zona de memoria tipo RAM. En el caso del PIC16F877 disponemos de espacio en todos los bancos de memoria a partir de la posicion 0x20. En el programa podemos crear una variable en la posición la 0x20 para almacenar datos o efectuar operaciones aritmetico lógicas en ella :

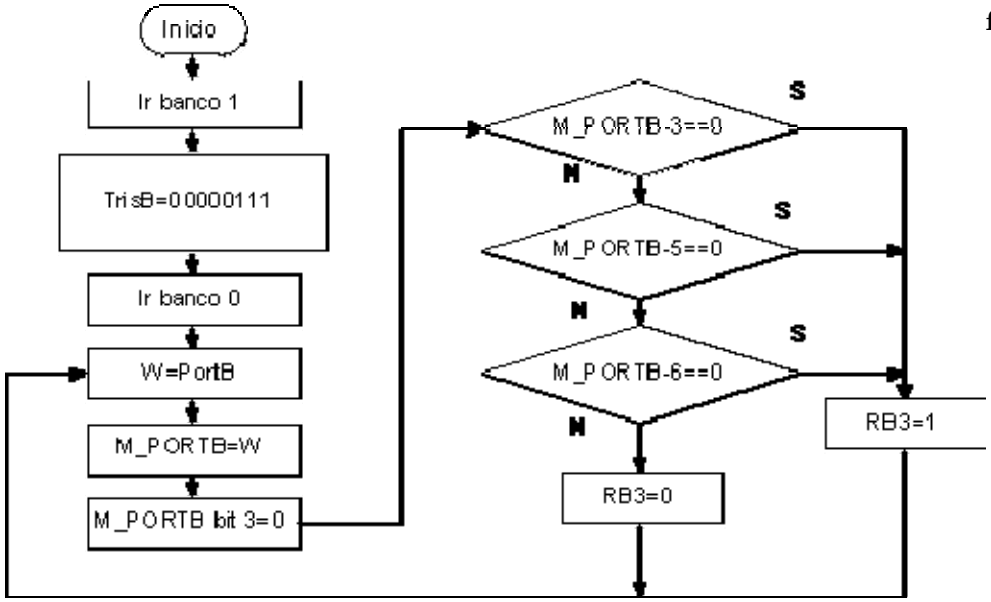
```
movwf 0x20 ; RAM[0x20]=W
bsf 0x20, 5 ;RAM[0x20] bit 5 =1
```

Pese a que el manejo es correcto este no resulta el mas apropiado sobre todo si el programa es extenso. Para hacerlo simple nos apoyamos en la directiva de compilación EQU que permite definir equivalencias:

```
VARIABLE EQU 0x20

movwf VARIABLE ; VARIABLE]=W
bsf VARIABLE, 5 ;VARIABLE bit 5 =1
```

b) Diseño del programa

Diagrama de flujo:



c) Trabajo en el MPLAB

1. Ingrese al MPLAB:
Cree un projecto en la carpeta de trabajo c:\archivos de programa\curso\
Asigne como nombre lamparin.pjt.

Abra un nuevo archivo e ingrese el siguiente código:

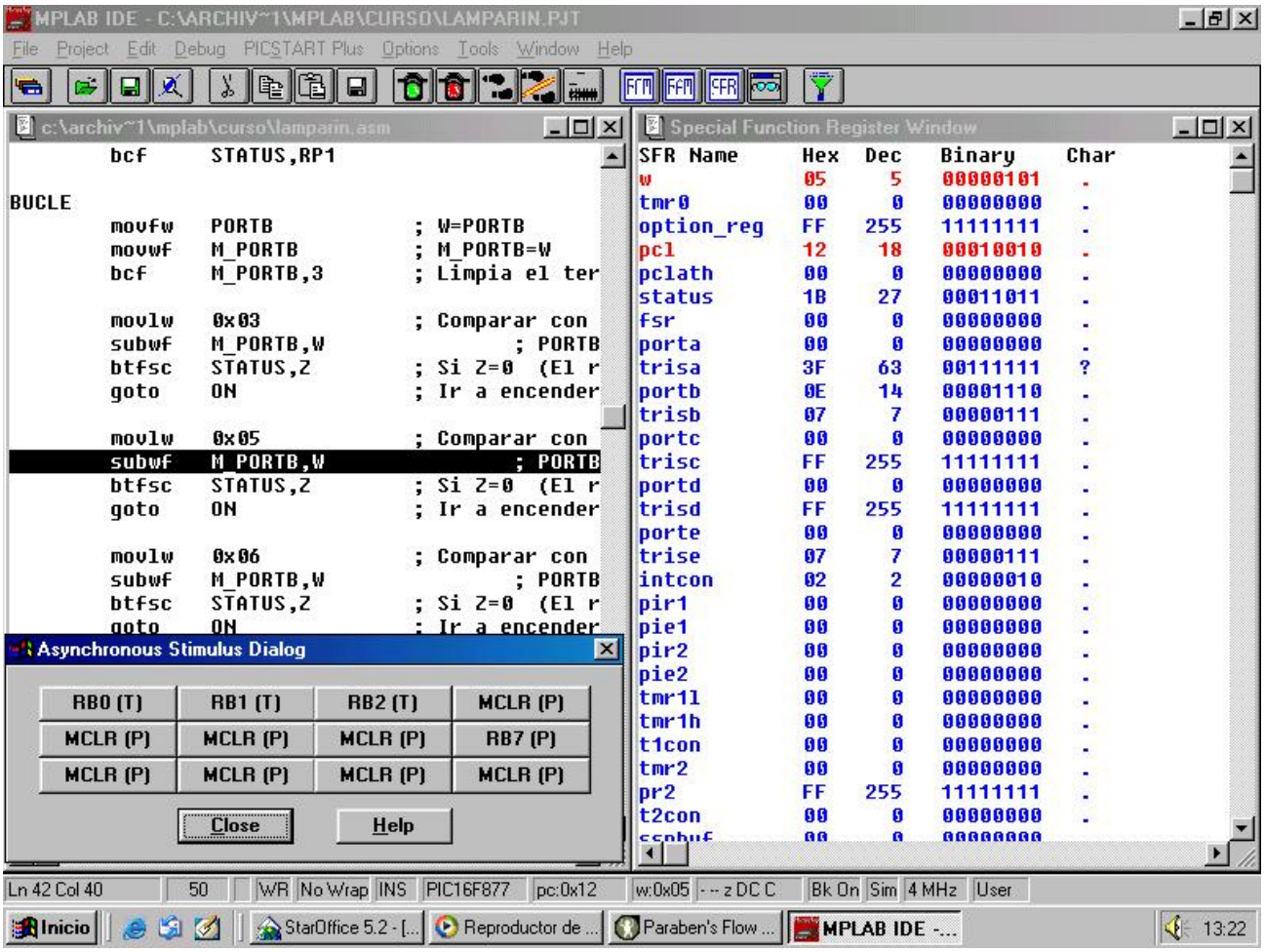
```
    list p=16F877
    include "p16f877.inc"

    CONF_ADCON1 EQU b'00000110'
    M_PORTB EQU 20

org 0x000 ; Origen del codigo
    nop ; No operacion
    nop ; No operacion
    bsf STATUS,RP0 ; Ir banco 1
    bcf STATUS,RP1

    movlw CONF_ADCON1 ; Configurar el PORTA como digital
    movwf ADCON1

    movlw b'00000111' ; RB2-0 entrada RB3 salida
    movwf TRISB

    bcf STATUS,RP0 ; Ir banco 0
    bcf STATUS,RP1

BUCLE
    movfw PORTB ; W=PORTB
    movwf M_PORTB ; M_PORTB=W
    bcf M_PORTB,3 ; Limpia el tercer bit

    movlw 0x03 ; Comparar con 3
    subwf M_PORTB,W ; PORTB-3
    btfsc STATUS,Z ; Si Z=0 (El resultado es no es 0)
    goto ON ; Ir a encender

    movlw 0x05 ; Comparar con 5
    subwf M_PORTB,W ; PORTB-5
    btfsc STATUS,Z ; Si Z=0 (El resultado es no es 0)
    goto ON ; Ir a encender

ON bsf PORTB,3 ; RB3=1
    nop
    goto BUCLE ; Ir bucle
END ; Fin del programa
```

2. Simulación

Al igual que en la primera simulación proceda a abrir la ventana de los registros especiales de funcion SFR.
Ingrese al menú **WINDOWS** elija **SPECIAL FUNCTION REGISTERS**

También habilite la ventana correspondiente ASYNCRONUS STIMULUS
Ingrese al menú **DEBUG** elija **SMULATOR STIMULUS** elija **ASYNCRONUS STIMULUS**

Empiece la simulación.
Con ayuda del Asynchronous Stimulus asigne al PORTB el valor 0x05 luego 0x03 y 0x06. La salida RB3=1
Nota.- No olvide revisar el bit Z del STATUS el bit 2



Ejercicio 2 : Diseñar un programa que simule a un comparador de 4 líneas.

Considerando las líneas de entrada:

Dato B				Dato A			
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
B3	B2	B1	B0	A3	A2	A1	A0

Y las líneas de salida:

Condición	RA2	RA1	RA0
A<B	0	0	1
A=B	0	1	0
A>B	1	0	0

a) Definiciones previas.

Para solucionar el problema nos apoyaremos en el uso de la instrucción SUBWF. La definición de la instrucción es la siguiente:

SUBWF Substrae W de f

Sintaxis: [etiqueta] SUBWF f,d
Operandos: 0 <= f <= 127 d= [0,1]
Operación: (f) – (W)-> (destino)

Bits afectados en el STATUS: C, DC, Z

Descripción: Substrae (usa en método de complemento a 2) el registro W del registro 'f'. Si 'd' es 0 el resultado es almacenado en W. Si 'd' es 1el resultado es almacenado devuelta en el registro 'f'.

Ejemplos: **SUBWF REG1,1**

Antes de la instrucción	**Después de la instrucción:**
REG1 = 3 **REG1>W** W = 2 C = ? Z = ?	REG1 = 1 W = 2 C = 1; **El resultado es positivo** Z = 0
REG1 = 2 **REG=W** W = 2 C = ? Z = ?	REG1 = 0 W = 2 C = 1; **El resultado es cero** Z = 1
REG1 = 1 **REG1<W** W = 2 C = ? Z = ?	REG1 = 0xFF W = 2 C = 0; **El resultado es negativo** Z = 0

b) Diseño del programa

Al aplicar la instrucción SUBWF sobre dos números (REG1 y W) podemos identificar cual de los dos números es mayor. El resultado de la diferencia se almacena en REG1 y los bits C y Z nos indican la relación de desigualdad.
Los números a comparar comparten el mismo registro (PORTB). Por tanto antes de aplicar la subtracción será necesario separarlos. Por eso copiaremos el valor de PORTB a dos registros (o files) libres. En el caso del dato A (nible bajo) bastará una Y-lógica con el valor 0x0F para separar el valor. En el caso del dato B (nible alto) primero debemos aplicar una instrucción swap (que invierte el orden de los nibles) y después una Y-lógica con 0x0F. El formato de ambas instrucciones se muestra a continuación:

SWAPF Intercambia los nibles de f

Sintaxis: [etiqueta] SWAPF f,d
Operandos: 0 <= f <= 127 d [0,1]
Operación: (f<3:0>) -> (destino<7:4>), (f<7:4>) -> (destino<3:0>)

Bits afectado en el STATUS: Ninguno

Descripción: El nible superior e inferior del registro 'f' son intercambiados. Si 'd' es 0 el resultado es almacenado en el registro W . Si 'd' es 1 el resultado es almacenado en el registro 'f'.

Ejemplo: SWAPF REG, 0

Antes de la instrucción: REG1 = 0xA5
Después de la instrucción REG1 = 0xA5 , W = 0x5A

ANDWF Y-lógica de W con f

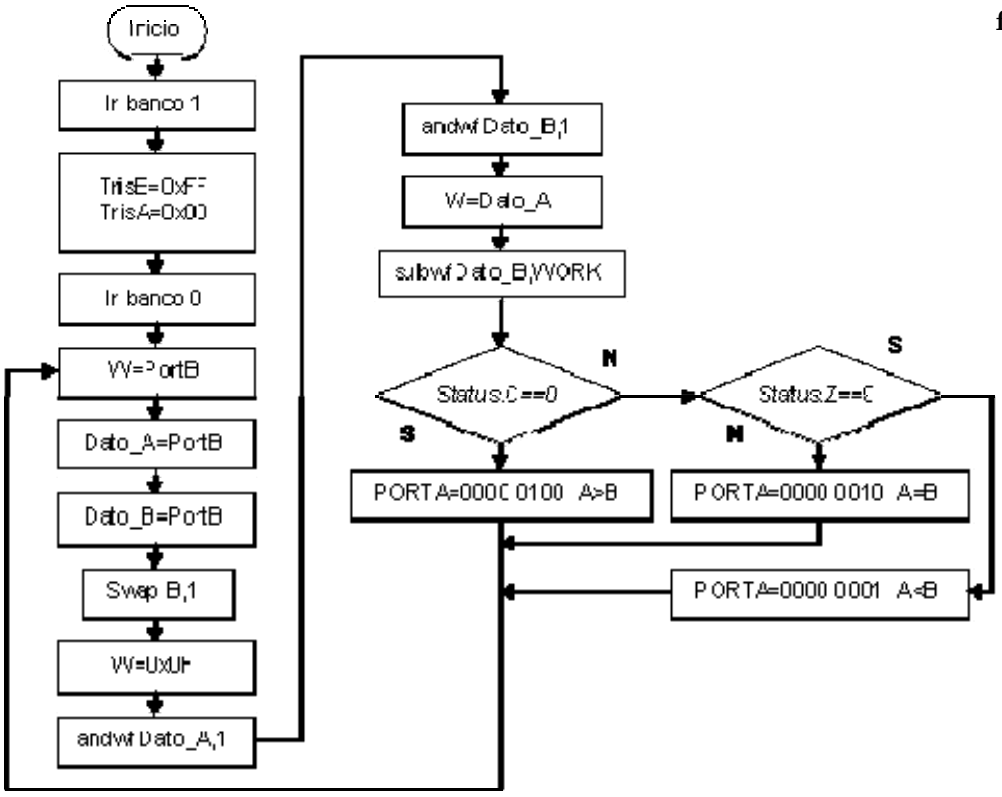
Sintaxis: [etiqueta] ANDWF f,d
Operandos: 0 <= f<= 127 d [0,1]
Operación: (W) .AND. (f) -> (destino)

Bits afectados en el STATUS: Z Codificación: 00 0101 dfff ffff

Descripción: Y-lógica del registro W con el registro 'f'. Si 'd' es 0 el resultado es almacenado en W. Si 'd' es 1 el resultado es almacenado en el registro 'f'.

Ejemplo: ANDWF FSR, 1
Antes de la instrucción: W = 0x17 FSR = 0xC2
Después de la instrucción: W = 0x17 FSR = 0x02

Diagrama de flujo:



c) Trabajo en el MPLAB

1. **Ingrese al MPLAB:**
   Cree un projecto en la carpeta de trabajo c:\archivos de programa\curso\
   Asigne como nombre compara4.pjt.

Abra un nuevo archivo e ingrese el siguiente código:

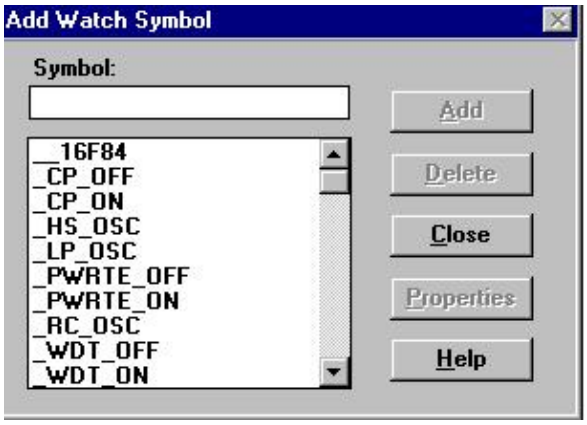
```
list p=16F877
include "p16f877.inc"
CONF_ADCON1 EQU b'00000110' ; PA como puerto digital
DATO_A EQU 0x20 ; Dato A
DATO_B EQU 0x21 ; Dato B
org 0x000 ; Origen del codigo
nop ; No operacion
nop ; No operacion


bsf STATUS,RP0 ; Ir banco 1
bcf STATUS,RP1


movlw CONF_ADCON1 ; Configurar el PORTA como digital
movwf ADCON1
movlw 0xFF ; PORTB como entrada
movwf TRISB
```

```
movlw 0x00 ; PORTA como salida
movwf TRISA
bcf STATUS,RP0 ; Ir banco 0
bcf STATUS,RP1
BUCLE
movf PORTB,W ; W=PORTB
movwf DATO_A ; RAM[DATO_A]=W
movwf DATO_B ; RAM[DATO_B]=W

swapf DATO_B,F ; Invertimos los nibles Ej: si 0xA5 => 0x5A
movlw 0x0F ; W=0x0F
andwf DATO_A,F ; RAM[DATO_A]= 0x0F AND RAM[DATO_A]
andwf DATO_B,F ; RAM[DATO_B]= 0x0F AND RAM[DATO_B]

movf DATO_A,W ; W=RAM[DATO_A]
subwf DATO_B,W ; W=RAM[DATO_B]-RAM[DATO_A](W)

btfsc STATUS,C ; ACARREO=0 SALTA
goto EVALUA1
movlw b'00000100' ; A>B => PB=100
goto ESCRIBE
EVALUA1
btfsc STATUS,Z ; CERO=0 SALTA
goto EVALUA2
movlw b'00000001' ; A>B => PB=001
goto ESCRIBE
EVALUA2
movlw b'00000010' ; A=B => PB=010
ESCRIBE
movwf PORTA ; PA=Resultado
goto BUCLE

END
```

2. Simulación

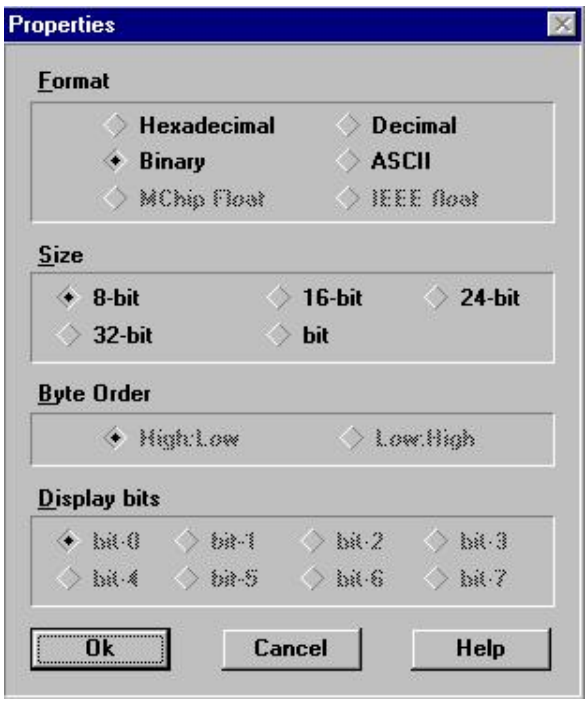
Hay varias formas de simular el proyecto. En esta ocasión vamos a crear una ventana particular que visualice los valores relevantes al proyecto.

Ingrese al menú **WINDOWS** elija **WATCH WINDOWS** y allí elija **NEW WATCH WINDOWS**

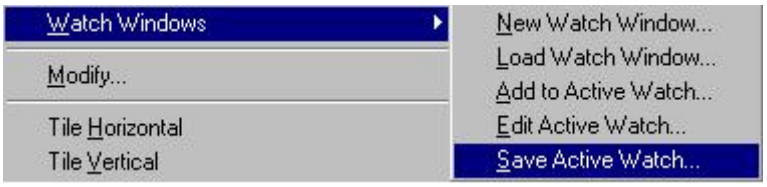


Alli debe seleccionar los registros siguientes
trisa, trisb, PORTA, PORTB, status, DATO_A, DATO_B y w.

Ingrese a:
Menú WINDOWS-> WATCH WINDOWS -> Edit Active Watch
Elija el registro **STATUS** y presione el botón Properties. Asegurece que el formato de displayado sea binario (**Binary**)

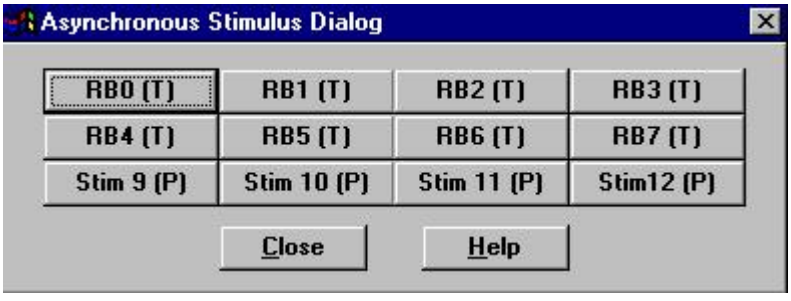


NOTA.- En el menú WINDOWS->WATCH WINDOWS hay otras opciones.



Si desea eliminar algún valor de los incluidos en la ventana puede usar la opción Edit. Si desea guardar la ventana que ha creado elegirá la opción Save y para cargarla posteriormente empleará la opción Load.

También habilite la ventana del estimulo asíncrono. Asigne los botones a los pines del puerto B (RB0-7). Configure los botones para que trabajen en modo **TOGGLE**.



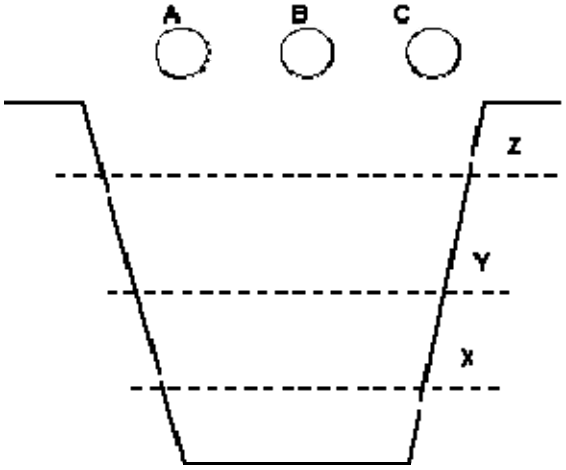
Empiece la simulación.
Con ayuda del Asynchronous Stimulus asigne al PORTB el valor 0x55. La salida del PORTA debería ser A=B. Luego cambie el PORTB a 0x72 la salida debería ser A<B. Finalmente 0x5A la salida debería ser A>B También usar la animación para verificar el

funcionamiento del programa. Elija Ud. los valores. **Nota.- No olvide Z es el bit 2 y C es el bit 0 del STATUS**

Ejercicio3:

Se tiene tres válvulas (A,B y C) que alimentan un tanque, el tanque a su vez tiene una salida. Existen 3 sensores de nivel (X,Y y Z). Cuando el tanque está vacio los 3 sensores estan a 0-lógico y es necesario activar el trabajo de las tres bombas. Cuando se llena 1/3 del tanque el sensor X pasa a 1-lógico y la bomba C deja de funcionar. Cuando se llenan 2/3 del tanque el sensor Y esta activado y la bomba B deja de funcionar. Cuando esta lleno el tanque el sensor Z se activa y la bomba A deja de funcionar. Una vez que el tanque esta lleno este empieza a expulsar el líquido acumulado. Cuando los 3 sensores pasan a 0-lógico la secuencia antes descrita se repite ANTES NO.

Solucione el ejercicio para ello se le proporcionan los siguientes elementos:



Salidas			Entradas		
C	B	A	Z	Y	X
RC2	RC1	RC0	RB2	RB1	RB0
1	1	1	0	0	0
1	1	0	0	0	1
No permitido			0	1	0
1	0	0	0	1	1
No permitido			1	0	0
No permitido			1	0	1
No permitido			1	1	0
0	0	0	1	1	1

Algoritmo

1. Configurar PORTB como entrada y PORTC como salida
2. Si PORTB != 0 entonces Ir paso2
3. Abrir las 3 válvulas (PORTC=0x07)
4. Si PORTB != 0x01 entonces Ir paso4

5. Abrir 2 válvulas (PORTC= 0x06)
6. Si PORTB != 0x03 entonces Ir paso6
7. Abrir 1 válvulas (PORTC= 0x04)
8. Si PORTB != 0x07 entonces Ir paso8
9. Cerrar todas las válvulas (PORTC= 0x00)
10. Ir paso 2

Elabore el diagrama de flujo, el código del programa y pruebe la simulación en el MPLAB

Módulo 2: Manejo de Temporizadores

Módulo Timer 0

El módulo Timer0 puede ser usado para generar periodos de tiempo (si funciona como temporizador) o puede ser usado para registrar el paso de eventos (si trabaja como contador de eventos). Al igual que en el módulo anterior existen registros que controlan el funcionamiento del módulo timer 0:

Posmem	Banco 0	Banco 1	Posmem
0x01	TMR0	OPTION_REG	0x81
	....	.....	
0x0B	INTCON<TOIF>	INTCON<TOIF>	0x8B
	.....	.....	

NOTA: La misma distribución se repite en los bancos 2 y 3

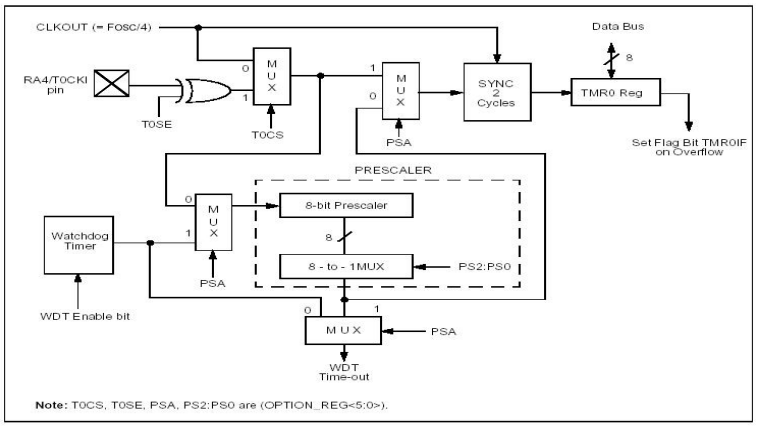
La operación implica la siguiente secuencia:

- Ingresar al banco 1
- Configurar el modulo timer 0 (como contador de eventos o timer)
- Regresar al banco 0
- Cargar el valor del TMR0 (inicializar la cuenta)

Las principales características del modulo timer 0 son.

- Puede ejecutar hasta 256 cuentas (0-255) debido a que el registro TMR0 es de 8 bits.
- El registro TMR0 puede ser leído para saber cual es valor actual de las cuentas o puede ser escrito para colocar un valor inicial.
- Posee un bloque de preescalamiento que permite ampliar el rango de las cuentas.
- Selector interno para definir si trabaja como temporizador o como contador de eventos
- Genera una señal de interrupción cuando se produce un desborde en el registro TMR0 (cuando pasa de 0xFF a 00). Bit T0IF del registro INTCON
- Selector para indicar si detecta flanco de subida o bajada cuando opera como contador de eventos. Bit T0SE del registro OPTION_REG (OPTION para fines prácticos lo llamaremos OPTION_REG en adelante).

Diagrama de Bloques del TMR0



Como se aprecia en la parte superior derecha esta presente un MUX. El MUX es controlado por el bit T0CS, si es 0 el módulo opera como temporizador; si es 1 como contador de eventos. El bit TOCS pertenece al registro OPTION_REG y es el bit 5.

Cuando el módulo funciona como timer el registro TMR0 se incrementa en cada ciclo de instrucción (siempre y cuando el Preescalamiento este deshabilitado). Cuando el registro TMR0 es escrito el microcontrolador debe esperar 2 ciclos de instrucción para que comenzar la cuenta.

Cuando el módulo trabaja como contador de eventos el registro TMR0 incrementa su valor cada vez que aparece un flanco en el pin RA4/TOCKI. La selección del tipo de flanco (subida o bajada) dependerá de la programación del bit TOSE (registro OPTION_REG bit 4). Si TOSE es 0-lógico trabaja con flanco de subida si es 1-lógico con flanco de bajada.

El preescaler es un módulo compartido por el WATCHDOG y el Timer 0. El preescaler es como un divisor de frecuencia programable. Como se aprecia puede conectarse en serie al modulo timer 0 (dependiendo de los valores de TOCS y PSA). Supongamos que el módulo timer 0 funciona como contador de eventos y el preescaler esta habilitado. El valor es 1:1, eso significa que cada pulso que ingrese incrementará el valor en el registro TMR0 (el valor máximo de cuentas será 256) . Si el preescaler vale 1:8 por cada 8 eventos que sucedan solo aumentará una cuenta en el registro TMR0 (el valor máximo de cuentas será 8x256). El valor del preescaler depende de los bits PS2 (bit 2), PS1(bit 1) y PS0 (bit 0) del registro OPTION_REG. El bit PSA (bit 3) del registro INTCON define si el el preescaler funciona con el Watchdog o con el Timer 0.

Observemos el detalle del registro INTCON:

	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
	bit 7							bit 0
bit 7	**RBPU**							
bit 6	**INTEDG**							
bit 5	**T0CS**: Bit selector de fuente para el TMR0							

	1 = Clock externo, pin RA4/T0CKI						
	0 =Clock interno(CLKOUT)						
bit 4	**T0SE**:Bit selector de flanco						
	1 = Incrementa en flanco de bajada en pin T0CKI						
	0 = Incrementa en flanco de subida en pin T0CKI						
bit 3	**PSA**: Bits de asignación del preescaler						
	1 = Prescaler es asignado al WATCHDOG						
	0 = Prescaler es asignado al modulo Timer0						
bit 2-0	**PS2:PS0**: Bits selectores relacion de trabajo						
	Valor de los bits	Relación TMR0	RelaciónWDT				
	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				

Ejercicio 9

Diseñar un programa en base al PIC16F877 para contar eventos (flancos de bajada en RA4/T0CKI) y mostrar la cuenta en un display de 7 segmentos conectado al puerto B. Cuando las cuentas llegan a 9 pasan de nuevo a 0.

a) Definiciones previas.

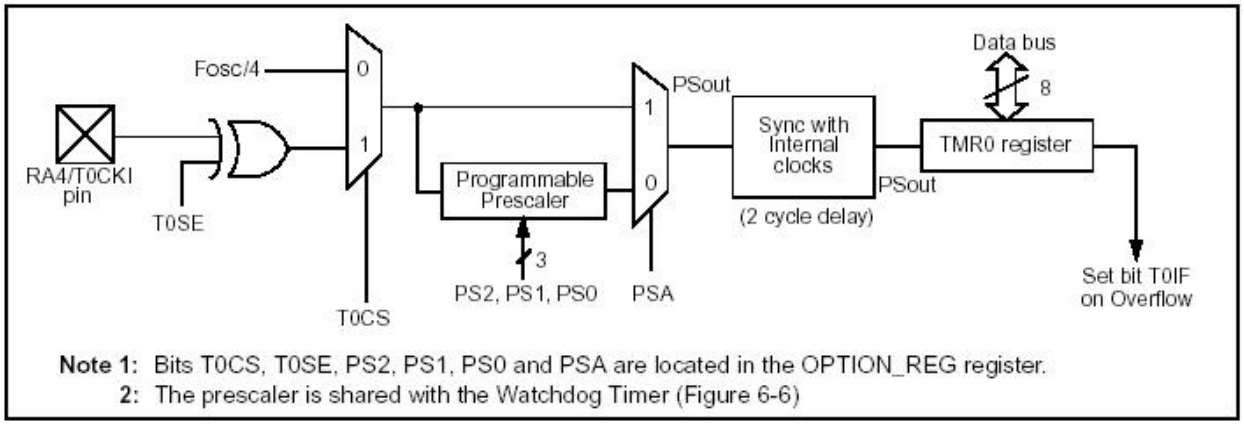
MODULO TMR0
Es un módulo que se encuentra en el PIC que se puede usar como temporizador o como contador de eventos. Cuando se emplea como contador de eventos se aplican los pulsos al pin RA4/TOCKI. En este tipo de programación se dice que trabaja como TOCK. Cuando se emplea como temporizador (timer) la entrada de valores la hace el oscilador (en nuestro caso un XTAL).

b) Diseño del programa

Como es necesario mostrar el número de eventos usaremos una tabla en el manejo de los display.

Car	**PORT B**

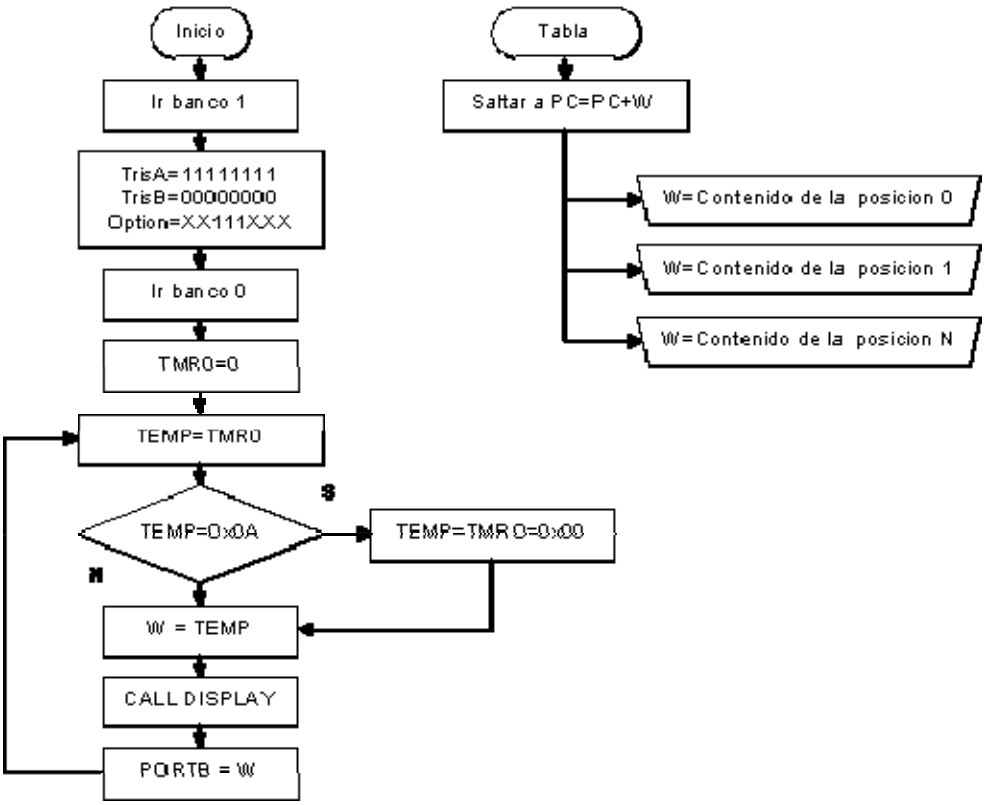
	Rb6	Rb5	Rb4	Rb3	Rb2	Rb1	Rb0
	G	F	E	D	C	B	A
0	0	1	1	1	1	1	1
1	0	0	0	0	1	1	0
2	1	0	1	1	0	1	1
3	1	0	0	1	1	1	1
4	1	1	0	0	1	1	0
5	1	1	0	1	1	0	1
6	1	1	1	1	1	0	1
7	0	0	0	0	1	1	1
8	1	1	1	1	1	1	1
9	1	1	0	1	1	1	1



Note 1: Bits T0CS, T0SE, PS2, PS1, PS0 and PSA are located in the OPTION_REG register.
2: The prescaler is shared with the Watchdog Timer (Figure 6-6)

Recordando el diagrama de bloques del Timer 0 vemos el detalle del módulo. Definimos la ruta como contador de eventos por eso será necesario colocar el bit TOCS a 1. No será necesario usar preescalamiento, por tanto el bit PSA estará a 1. El bit TOSE define si trabajamos con flanco de subida o bajada: si TOSE=1, detecta flanco de bajada. Si TOSE =0, detecta el flanco de subida. Según el planteamiento del programa será necesario usar el flaco de bajada (TOSE=1).

Diagrama de flujo:



Como se aprecia en el diagrama de flujo ingresamos al banco 1 a programar los puertos y el módulo TMR0 en los registros TRISA, TRISB y OPTION. Regresamos al banco 0, colocamos el TMR0 a 0 y entramos en un bucle. En el bucle leeremos el valor actual del TMRO lo copiamos a una variable temporal y observamos cual es el valor que presenta, si es igual a 10 reseteamos al TMR0 y a la variable temporal. Finalmente a través de una tabla visualizamos el valor de la variable TEMP. El valor del TMR0 se incrementa solo, depende de los pulsos que arriben al pin RA4/TOCKI.

c) Trabajo en el MPLAB

1. **Ingrese al MPLAB:**
   Cree un projecto en la carpeta de trabajo c:\archivos de programa\mplab\curso
   Asigne como nombre evento.pjt.
   Abra un nuevo archivo e ingrese el siguiente código:

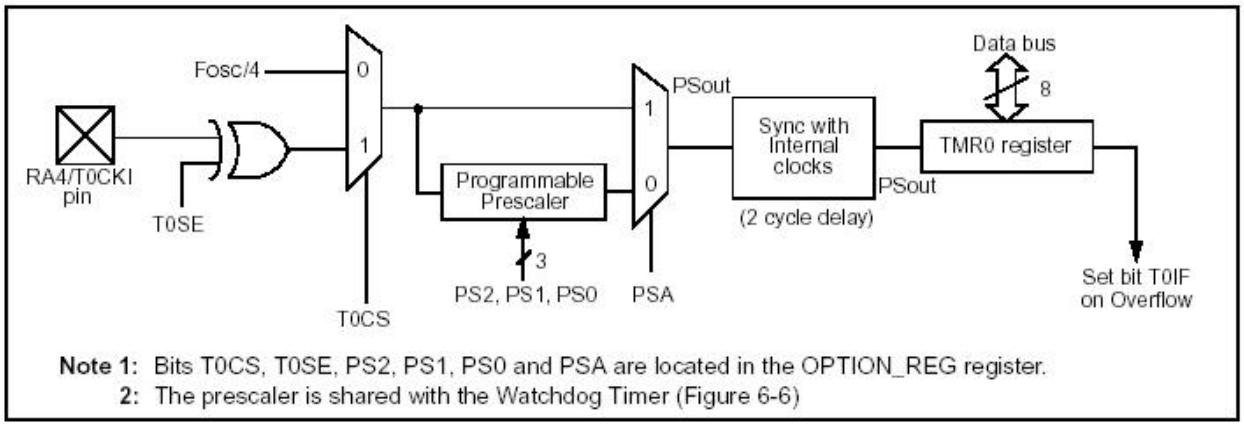
```
; REGISTRO OPTION
; -----------------------------------------------
; X X TOCS TOSE PSA PS2 PS1 PS0
; -----------------------------------------------
; 1 1 1 1 1 0 0 0 = 0xF8
; -----------------------------------------------
; C.EVEN. F.BAJ TMR0 Escalamiento

     list p=16f877 ;Comando que indica el Pic usado
   include "p16f877.inc" ;Etiquetas genéricas para el Pic16F877

   CONF_ADCON1 EQU b'00000110' ; PA entrada digital
   CONF_OPT EQU 0xF8 ; Valor a escribir en el registro de configuracion del TMR0
```

```
    LIMITE EQU 0x0A ; Limite de la cuenta
    TEMP EQU 0x20 ; Variable temporal

org 0x000 ;Inicio del programa en la posición cero de memoria
    nop ;Libre (uso del debugger)
    nop

_inicio
    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1

    movlw CONF_ADCON1 ;PA como entrada digital
    movwf ADCON1
    movlw 0xFF
    movwf TRISA ;PA entrada
    clrf TRISB ;PB salida
    movlw CONF_OPT
    movwf OPTION_REG ;Configuracion del TMRO

    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1

    clrf TMR0 ;TMR0=0

BUCLE
    movf TMR0,W ;W=TMR0
    movwf TEMP ;TEMP=W
    movlw LIMITE ;W=10
    xorwf TEMP,W ;W XOR TEMP
    btfss STATUS,Z ;EL resultado de la anterior instruccion es 0?
    goto SIGUE ;Z=0, NO es diferente de 0, TMRO = 0,1,2,3,4,5,6,7,8,9
    clrf TMR0 ;Z=1, SI vale 10, TMRO > 9, TMR0=0
    clrf TEMP ;Temp=0

SIGUE movf TEMP,W ;W=TEMP
    call DISPLAY ;Decodifica el valor de DISPLAY
    movwf PORTB ;Escribe el valor en PORTB
    goto BUCLE ;Salta a la etiqueta bucle

DISPLAY
    addwf PCL,f
    retlw b'01000000' ;0
    retlw b'01111001' ;1
    retlw b'00100100' ;2
    retlw b'00110000' ;3
    retlw b'00011001' ;4
    retlw b'00010010' ;5
    retlw b'00000010' ;6
    retlw b'01111000' ;7
    retlw b'00000000' ;8
    retlw b'00010000' ;9

END
```

2. Simulación
Habilite la ventana del ASINCHONUS STIMULUS, SPECIAL FUNTION REGISTER, organice las ventanas. A continuación asigne al botón 1 el valor TOCKI y al botón 2 el valor RA4. Asegúrece que ambos botones estén definidos como señales de pulso. A continuación simule el programa con ayuda del STEP (F7). No olvide presionar entre paso y paso (STEPS) cualquiera de los botones recientemente asignados. Notará como es que el pulso toma efecto 3 ciclos de instrucción después debido a que hemos elegido el flanco de bajada es decir el pulso demora 3 ciclos de instrucción. Si hubiéramos elegido el de subida la acción hubiera sido inmediata. Use la opción de animación para probar el incremento y como se resetea a 0 una vez que alcanzo el valor de 10

Ejercicio 10

Programar el TMR0 para generar un retardo de un segundo. A partir del cual se



incrementa un contador cuyo valor se muestra por el PORTC

a) Definiciones previas.

Para generar retardos es necesario usar el modulo timer 0 como temporizador por lo tanto es necesario colocar el **bit T0CS a 0**. Ahora debemos encontrar una expresión que nos ayude a calcular cuantos ciclos de instrucción necesitamos para generar un retardo de un segundo.

Un ciclo de instruccion = 4*Tosc (donde Tosc es el inverso de la frecuencia del clock que usa el PIC)
Si lo multiplicamos por un numero 'X'; tendremos un retardo de
Retardo= Un ciclo de instruccion * X; (como el modulo Timer 0 lleva la cuenta en el registro TMR0)

Retardo_T0= 4*Tosc* TMR0

Bajo esta expresión debemos considerar que el tiempo máximo sería:
Retardo_T0 =4*Tosc*256 Ahora si tenemos un clock de 4MHz conectado al PIC tendriamos: 256 us.

Ademas sabemos que el modulo timer 0 posee un preescaler que sirviria para amplia el retardo. Si lo usamos debemos configurar el bit **PSA a 0** . Si seteamos el bit **PS2, PS1 y PS0 a 1** el preescaler tendria un valor de 256.

Retardo_T0_Pre= 4*Tosc*TMR0*Preescaler.

Retardo_T0_Pre= 4.0.25us*256*256 =65536 us =65.536 ms

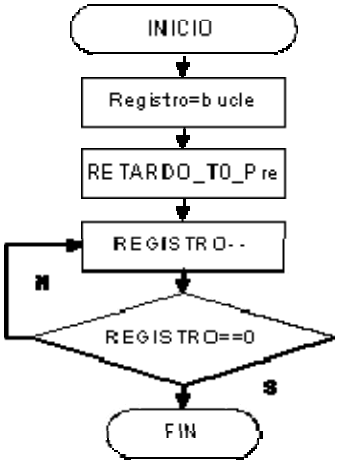
Con esto no alcanzamos a generar un segundo. Sin embargo podemos usar un registro que sirva para efectuar varios bucles. En tal caso, podríamos lograr retardos mayores:

Retardo=Bucle*Retardo_T0_Pre
Retardo=Bucle*65,536ms

Como el Retardo debe ser 1seg

1000ms =Bucle*65,536

Bucle=15.25



Como vemos el generar el retardo no es tan dificil si consideramos este procedimiento desde luego que nos faltaría definir una función que de nos indique cuando el TMRO ha generado las 256 cuentas.

Antes de dar la forma final al programa vamos a examinar ese punto.

Existe un bit T0IF (Timer 0 interrup flag) que se pone a 1 siempre que hay un desborde en el Registro TMR0 es decir cuando pasa de 255 a 0. Esta característica nos puede ayudar mucho ya que una vez producido el desborde el bit T0IF permanece en 1. Es necesario resetear el bit T0IF.

Y lo mas importante es que debemos cargar el TMR0 con el valor apropiado para conseguir el retardo deseado.

Suponga que necesita un retardo de 10 us. Entonces nuestro algoritmo sería:

1. T0IF=0
2. Colocar TMR0 a 246
3. Esperar a que T0IF sea 1.

Puede llamar la atención que hallamos cargado el TMR0 a 245. Observe:

TMR0	**246**	**247**	**248**	**249**	**250**	**251**	**252**	**253**	**254**	**255**	**0**
Cuenta:	0 1	1 2	2 3	3 4	4 5	5 6	6 7	7 8	8 9	9 10	10 T0IF=1

Como se puede apreciar si cargamos el TMRO con 246 10 ciclos de instruccion (10 us) mas tarde obtendremos el retardo necesario si revisamos el valor de T0IF. Para fines prácticos equivale a restar el retardo de 256.

Por tanto no existe una sino múltiples soluciones considerando este último punto y la fórmula :

Retardo= 4*Tosc*TMR0*Preescaler*Bucle.

Observe el siguiente cuadro (considere que los valores siempre deben ser enteros):

Opcion	Ciclo de instrucción	TMR0	Preescaler	Bucle	Retardo
1	1us	256	256	15	983040 us
2	1us	256	256	16	1048576 us
3	1us	122	128	64	999424 us

La tabla muestra que los primeros valores que calculamos no han sido tan exactos. Si empleamos la opción 3 estaríamos mas cerca al segundo.

b) Diseño del programa

Algoritmo
De acuerdo a las consideraciones expuestas el registro OPTION_REG debería configurarse así:

X X TOCS TOSE PSA PS2 PS1 PS0
1 1 0 0 1 1 1 0

XTAL Preescaler 128

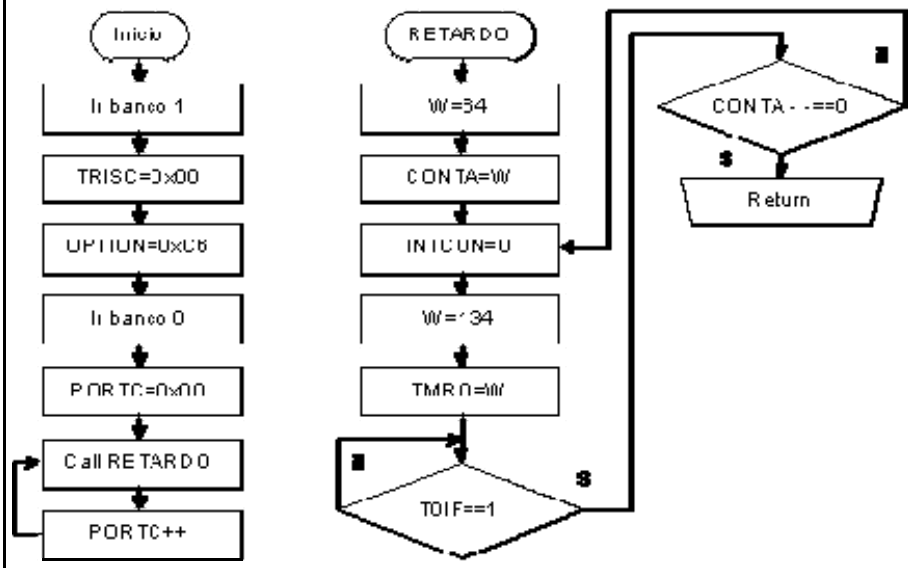
Programa Principal

1. Ir banco 1
2. TRISC=0
3. OPTION_REG=b'11001110'
4. Ir banco 0
5. PORTC=0
6. Call RETARDO
7. PORTC=PORTC+1
8. Ir paso 6

Rutina Retardo

1. CONTA=64
2. T0IF=0
3. TMR0=134 (122 cuentas)
4. Si T0IF es 1. Ir paso 6
5. sino ir paso 4
6. CONTA=CONTA -1
7. Si CONTAes 0 ir paso 9
8. sino Ir paso 4
9. Retorno

agrama de flujo:



Código del Programa

```
  list p=16f877 ;Comando que indica el Pic usado
    include "p16f877.inc" ;Etiquetas genéricas para el Pic16F877

    CONTA EQU 0x20 ;Variable CONTA en dirección 0x20 hexadecimal de ;memoria RAM

org 0x00 ;Inicio del programa en la posición cero de memoria
    nop ;Libre (uso del debugger)
    nop

_inicio
    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1

    clrf TRISA ;PORTA salida
    clrf TRISB ;PORTB salida
    clrf TRISC ;PORTC salida
    clrf TRISD ;PORTD salida
    clrf TRISE ;PORTE salida

    movlw b'11000110' ;Configuración del modulo TMR0
    movwf OPTION_REG ;Preescaler = 128

    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    clrf PORTC ;PORTC = 0
_bucle
    call _retardo ;Llama la rutina de retardo
    incf PORTC,F ;Incrementa el valor del PORTC
    goto _bucle ;Ir _bucle

_retardo ;T = 4 * Tosc * Valor de TMR0 * Preescaler
    movlw d'64' ;Cargar el valor de CONTA para 1 segundo
    movwf CONTA
_espera1
    clrf INTCON ;Deshabilitar interrupciones
    movlw d'134' ;Cargar el valor de TMR0 para 122 cuentas
    movwf TMR0 ;(Complemento)
_espera
```

```
    btfss INTCON,T0IF ;Esperar desborde del TMR0
    goto _espera
    decfsz CONTA,F ;Decrementar el registro CONTA hasta cero
    goto _espera1 ;Si no es cero: ir a _espera1
    return ;retorno de call
end
```

c) Trabajo en el MPLAB

Ingrese al MPLAB:

Cree un projecto en la carpeta de trabajo c:\archivos de programa\mplab\curso
Asigne como nombre m2p1.pjt.
Abra un nuevo archivo e ingrese el código.
Proceda a enlazar el ICD debugger al la tarjeta demoboard y grabe el programa
Ejecute el programa y verifique si es que aumenta la cuenta que se muestra en el PORTC cada segundo
Proceda a correr el programa paso a paso.

Estructura Interna y Funcionamiento del TMR1

El TMR1 es un Temporizador/Contador con un tamaño de 16 bits, lo que requiere el uso de dos registros concatenados de 8 bits: TMR1H : TMR1L, que son los encargados de guardar el valor del contaje en cada momento. Dicho valor evoluciona desde 0000h hasta FFFFh. Momento en el cual se activa el señalizador TMR1IF y se regresa al valor inicial 0000h.

El valor contenido en TMR1H : TMR1L puede ser leído o escrito y los impulsos de reloj que originan el contaje ascendente pueden provenir del exterior o de la frecuencia de funcionamiento del microcontrolador (Fosc/4)
El Timer1 tiene tres formas de funcionamiento:

a) Temporizador
b) Contador Síncrono
c) Contador Asíncrono

Como Temporizador el valor concatenado TMR1H : TMR1L se incrementa con cada ciclo de instrucción (Fosc/4). En el modo contador, el incremento se puede producir con los flancos ascendentes de un reloj externo, cuya entrada se aplica a las líneas RC0 y RC1 de la puerta C, o por impulsos aplicados en la línea RC0.

T1CON: REGISTRO DE CONTROL DEL TIMER1 (DIRECCION 10h)

—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

bit 7 bit 0
bit 7-6 Bits no implementados. Leído como **0**
bit 5-4 **T1CKPS1:T1CKPS0**: El Preescaler es un divisor de la frecuencia de los impulsos que se aplican al TIMER1

11 = 1:8 Valor de Preescaler
10 = 1:4 Valor de Preescaler
01 = 1:2 Valor de Preescaler
00 = 1:1 Valor de Preescaler

bit 3 **T1OSCEN**: Control de habilitación para el oscilador del Timer1
1 = El Oscilador es habilitado (RC0 y RC1 = entradas del oscilador externo)
0 = El oscilador trabaja en otro modo(RC0 = entrada del oscilador externo)

bit 2 **T1SYNC**: Determina la posible sincronización o no de los impulsos del reloj externo con los del reloj interno.
TMR1CS = 1

1 = No sincronización con un reloj externo

0 = Sincronización con un reloj externo TMR1CS = 0

Este bit es ignorado cuando el Timer1 usa el reloj interno

bit 1 **TMR1CS**: Selecciona la fuente de los impulsos de contaje

1 = Reloj externo desde el pin RC0/T1OSO/T1CKI (en el flanco de subida)

0 = Reloj interno (FOSC/4)

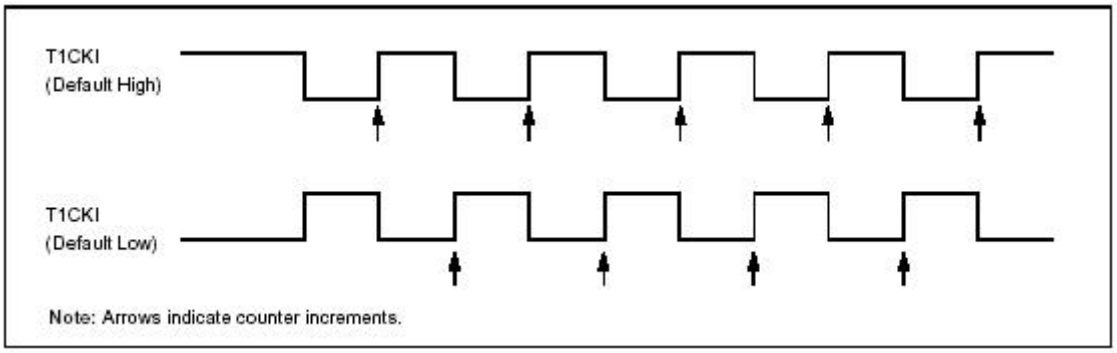
bit 0 **TMR1ON**: gobierna el permiso o la prohibición de funcionamiento del Timer1.

1 = Habilitar el Timer1

0 = Detener el Timer1

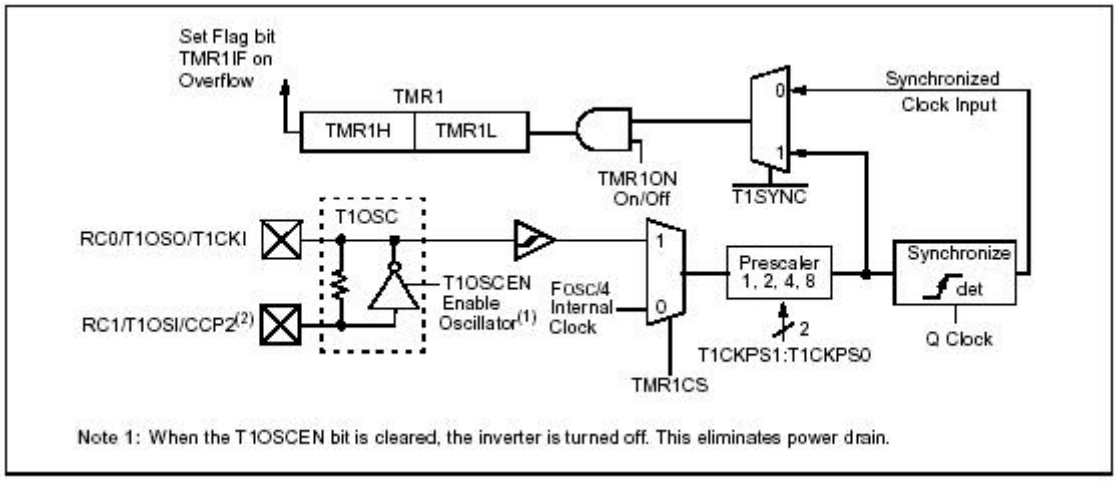
a).- Operación del Timer1 en modo Temporizador

Este modo es seleccionado limpiando el bit TMR1CS (T1CON<1>). En este modo la entrada de reloj al timer es Fosc/4. El bit T1SYNC (T1CON<2>) no tiene efecto desde el reloj interno; es siempre síncrono.



b) Timer1 en modo Contador Sincrono

Este modo es seleccionado seteando el bit TMR1CS. En este modo el timer incrementa cada flanco de subida de una entrada de reloj externo en el pin RC1, cuando el bit T1OSCEN es seteado. O cuando una entrada de reloj ingresa por el pin RC0, cuando el bit T1OSCEN es limpiado. Si el T1SYNC es limpiado, entonces el reloj externo es sincronizado con la fase del reloj interno.



c.- Timer1 en modo Contador Asíncrono

Si el bit T1SYNC (T1CON<2>) es seteado, el reloj externo no es sincronizado. El timer continua un incremento asíncrono a la fase del reloj interno. El timer continuará funcionando durante el modo SLEEP y podrá generar una interrupción de desborde; el cual podría despertar al procesador del modo SLEEP.

Lectura y escritura del Timer 1 en modo contador Asíncrono

Leer los registros TMR1H TMR1L mientras el timer esta corriendo desde un reloj asíncrono externo garantizará una lectura válida (tener cuidado con el hardware). Sin embargo el usuario tendrá en mente que leer 16 bits del timer en 2 registros de 8 bits puede causar molestia si el timer desborda mientras se produce la lectura.

Para escribir es recomendable que el usuario detenga el timer y escriba recién los valores deseados.

Oscilador para el TIMER1

Un oscilador de cristal puede ser construido entre los pines T1OSI (entrada) y T1OSO (salida amplificada). Esto es habilitado seteando el bit de control T1OSCEN (T1CON<3>). El oscilador puede ser hasta de 200 Khz. Este continuará oscilando durante el modo SLEEP.

Osc Type	Freq.	C1	C2
LP	32 kHz	33 pF	33 pF
	100 kHz	15 pF	15 pF
	200 kHz	15 pF	15 pF
These values are for design guidance only.			
Crystals Tested:			
32.768 kHz	Epson C-001R32.768K-A	± 20 PPM	
100 kHz	Epson C-2 100.00 KC-P	± 20 PPM	
200 kHz	STD XTL 200.000 kHz	± 20 PPM	

Ejercicios

Ejercicio 11

Programar el TMR1 para generar un retardo de 524.2 ms. Cada vez que concluya el tiempo se activará el PORTC de forma escalonada

a) Definiciones previas.

El modulo timer 1 cuenta con dos registros TMR1H y TMR1L que sumados nos pueden proveer de 2^16 cuentas esto es 65536 cuentas. Si tenemos un ciclo de instrucción de demanda 1 us (XTAL de 4MHZ). El retraso sera de 65.536 ms. Para alcanzar el valor que deseamos emplearemos el preescaler. Si lo colocamos a 1:8 obtendremos el valor deseado:

Retardo=4*Tosc*TMR1*Preescaler
Retardo=4*0.25us * 65636*8
Retardo=524,288 ms

Al igual que en el caso anterior nos apoyaremos en el bit de desborde del modulo timer 1 TMR1F. Como el valor lo vamos a mostrar en el PORTC como una escalera que se incrementa cada 524,2 ms usaremos una tabla para decodificar el valor a mostrar en el PORTC.

b) Diseño del programa

Algoritmo
Deshabilitamos el modulo timer 0 (OPTION_REG=0x80) y habilitamos el modulo timer 1(preescaler 1:8, deshabilitamos oscilador, no hay sincronismo, el clock es interno, el modulo timer 1 esta apagado).

X X T1CKPS1 T1CKPS0 T1OSCEN - T1SYNC TMR1CS TMR1ON
0 0 1 1 0 0 0 0

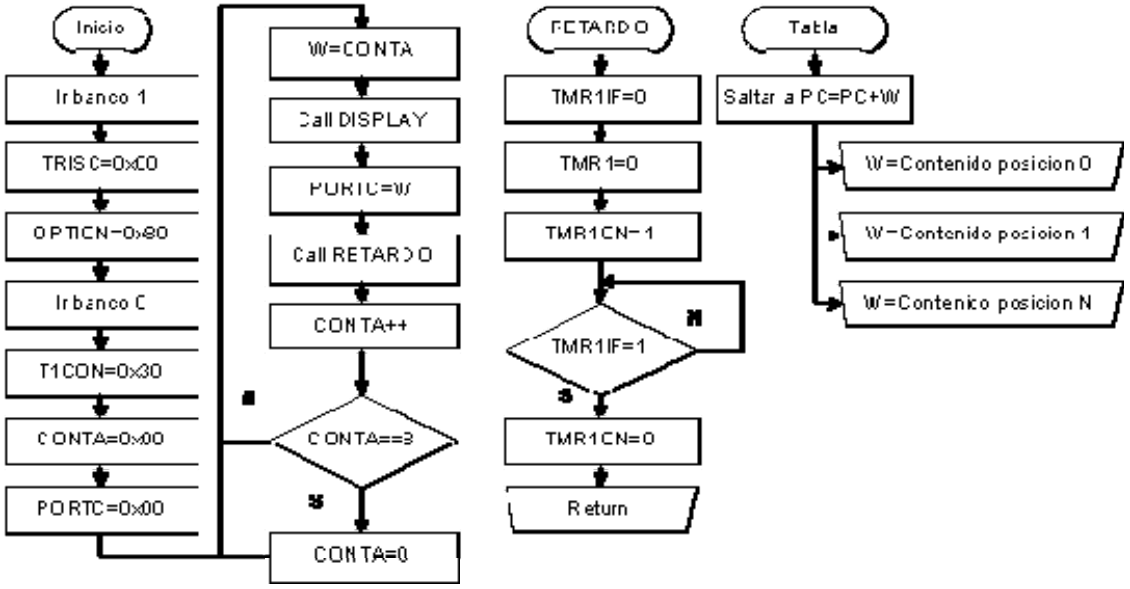
Programa Principal
1. Ir banco 1
2. TRISC=0
3. OPTION_REG=b'10000000'
4. Ir banco 0
5. T1CON=b'00110000'
6. CONTA=0, PORTC =0;
7. W=CONTA (indice)
8. CALL DISPLAY
9. PORTC=W
10 CALL RETARDO

11. CONTA=CONTA +1
12. Si CONTA es 8, entonces CONTA =0, Ir paso 7
13. Sino CONTA =CONTA+1, Ir paso 7

Retardo
1.TMR1IF=0
2. TMR1=0
3. TMR1ON=1
4. Si TMR1IF es 1, entonces TMR1ON=1, return
5. Ir paso 4

Display
1. PCL=PCL+W
2. Return con el valor literal indicado por el indice W

agrama de flujo:



Código del Programa

```
    list p=16f877 ;Comando que indica el Pic usado
    include "p16f877.inc" ;Etiquetas genéricas para el Pic16F877

    CONTA EQU 0x20 ;Variable CONTA en dirección 0x20 hexadecimal
                   ;de memoria RAM

org 0x000 ;Inicio del programa en la posición cero de memoria
    nop ;Libre (uso del debugger)
    nop

_inicio
    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1

    clrf TRISA ;PORTA salida
    clrf TRISB ;PORTB salida
    clrf TRISC ;PORTC salida
    clrf TRISD ;PORTD salida
    clrf TRISE ;PORTE salida
```

```
    movlw b'10000000' ;configuración del registro OPTION
    movwf OPTION_REG

    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1

    movlw b'00110000' ;Contador interno de 16 bits
    movwf T1CON

    clrf CONTA ;CONTA = 0
    clrf PORTC ;PORTC = 0

_bucle
    movf CONTA,W ;Cargar el valor de CONTA en W
    call _display ;Llama a la tabla _display
    movwf PORTC ;Al retornar de la subrutina el valor de W se saca
                ;por el PORTC
    call _retardo ;Llama a la subrutina retardo
    incf CONTA,F ;incrementa el valor de CONTA
    movlw d'8' ;Verifica si ha llegado a 8
    subwf CONTA,W
    btfss STATUS,Z
    goto _bucle ;Si no es 8: ir _bucle
    clrf CONTA ;Si es 8: CONTA = 0
    goto _bucle ;Ir _bucle

_retardo
    bcf PIR1,TMR1IF ;Borrar la bandera de desborde
    clrf TMR1L ;Limpiar los registros de conteo
    clrf TMR1H
    bsf T1CON,TMR1ON ;Habilita el TMR1
_espera
    btfss PIR1,TMR1IF ;Verificar el desborde
    goto _espera ;Si no ir _espera
    bcf T1CON,TMR1ON ;Si desborda: limpiar bandera de desborde
    return ;Retorno

_display
    addwf PCL,F ;pcl + W >>>> W
    ;El PCL se incrementa con el valor de W
    ;proporcionando un salto
    retlw b'10000000' ;retorna con valores para PORTC
    retlw b'11000000'
    retlw b'11100000'
    retlw b'11110000'
    retlw b'11111000'
    retlw b'11111100'
    retlw b'11111110'
    retlw b'11111111'

end
```

c) Trabajo en el MPLAB

Ingrese al MPLAB:

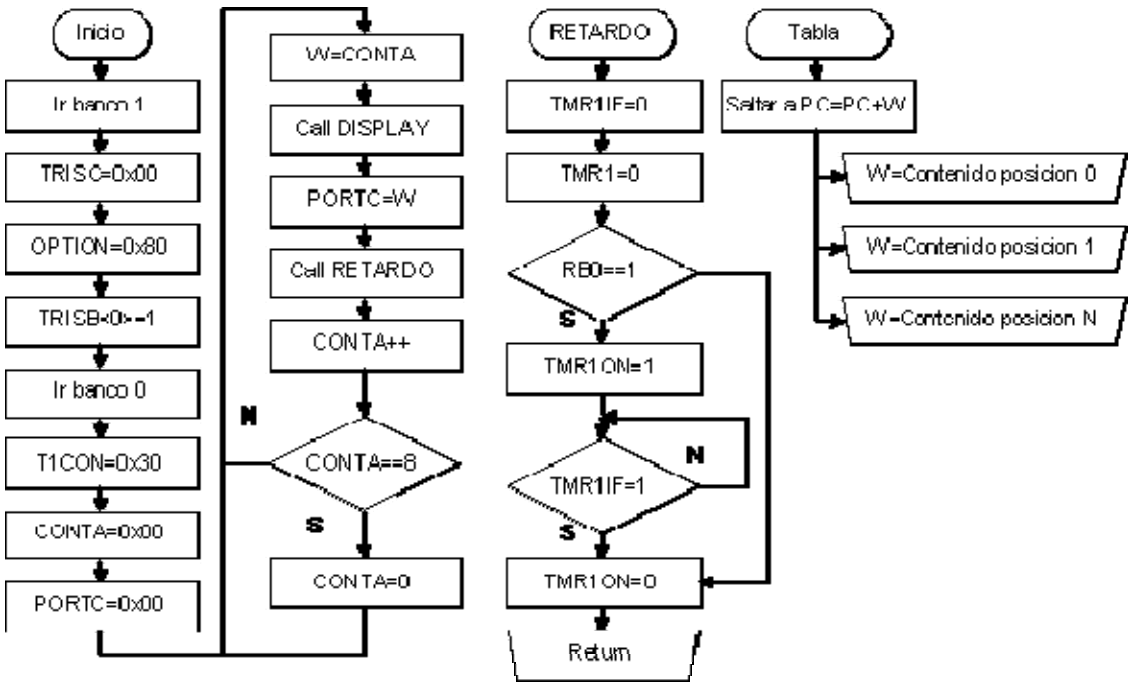
Cree un projecto en la carpeta de trabajo c:\archivos de programa\mplab\curso
Asigne como nombre m2p2.pjt.
Abra un nuevo archivo e ingrese el código.
Proceda a enlazar el ICD debugger al la tarjeta demoboard y grabe el programa
Corra el programa y verifique si cada 524,2 ms el valor en el PORTC crece como una escalera
Proceda a correr el programa paso a paso.

Ejercicio 12

Modificar el programa anterior para que lea el pin RB0, cuando se pulse deshabilitara el TMR1 y si se deja de pulsar reanudara el timer.

El ejercicio es similar al anterior. Configuramos el pin RB0 como entrada (en TRISB). El valor por defecto en el RB0 es 1-lógico en la tarjeta demoboard. Si la cuenta esta subiendo (con intervalos de 524 ms) y presionamos el pulsador conectado a RB0 (pasa a 0-lógico) deshabilitamos el TMR1 con la ayuda del bit TMR1ON (deshabilitado). En consecuencia la cuenta seguirá siendo visualizada e incrementada por el resto del programa pero como es muy rápida no podremos verla a simple vista por que el TMR1 ahora no genera el tiempo de 524 ms. Para solucionar el problema adicionaremos código a la rutina de retardo. Revisaremos el valor del RB0 dependiendo si es 0 detenemos la cuenta caso contrario el módulo seguirá generando los retardos. Observe el diagrama de flujo y el código.

Diagrama de Flujo



Código del Programa

```
    list p=16f877 ;Comando que indica el Pic usado
    include "p16f877.inc" ;Etiquetas genéricas para el Pic16F877

    CONTA EQU 0x20 ;Variable CONTA en dirección 0x20 hexadecimal
                   ;de memoria RAM

org 0x000 ;Inicio del programa en la posición cero de memoria
    nop ;Libre (uso del debugger)
    nop

_inicio
    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1

    clrf TRISA ;PORTA salida
    clrf TRISB ;PORTB salida
    clrf TRISC ;PORTC salida
    clrf TRISD ;PORTD salida
    clrf TRISE ;PORTE salida
```

```
    bsf TRISB,0 ;RB0 entrada

    movlw b'10000000' ;configuración del registro OPTION
    movwf OPTION_REG

    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1

    movlw b'00110000' ;Contador interno de 16 bits
    movwf T1CON

    clrf CONTA ;CONTA = 0
    clrf PORTC ;PORTC = 0

_bucle
    movf CONTA,W ;Cargar el valor de CONTA en W
    call _display ;Llama a la tabla _display
    movwf PORTC ;Al retornar de la subrutina el valor de W se saca ;por el PORTC
    call _retardo ;Llama a la subrutina retardo
    incf CONTA,F ;Incrementa el valor de CONTA
    movlw d'8' ;Verifica si ha llegado a 8
    subwf CONTA,W
    btfss STATUS,Z
    goto _bucle ;Si no es 8: ir _bucle
    clrf CONTA ;Si es 8: CONTA = 0
    goto _bucle ;Ir _bucle

_retardo
    bcf PIR1,TMR1IF ;Borrar la bandera de desborde
    clrf TMR1L ;Limpiar los registros de conteo
    clrf TMR1H

    btfss PORTB,0
    goto _deshabilitar ;Ir _deshabilitar
    bsf T1CON,TMR1ON ;Habilita el TMR1

_espera
    btfss PIR1,TMR1IF ;Verificar el desborde
    goto _espera ;Si no ir _espera

_deshabilitar
    bcf T1CON,TMR1ON ;Si desborda: limpiar bandera de desborde
    return ;Retorno

_display
    addwf PCL,F ;pcl + W >>>> W
    ;El PCL se incrementa con el valor de W
    ;proporcionando un salto
    retlw b'10000000' ;retorna con valores para PORTC
    retlw b'11000000'
    retlw b'11100000'
    retlw b'11110000'
    retlw b'11111000'
    retlw b'11111100'
    retlw b'11111110'
    retlw b'11111111'

end
```

Módulo 3: Convertidor Análogo Digital

Introduccion

Un convertidor análogo digital tiene como entrada un nivel de voltaje (valor analógico) y produce en su salida un número binario de n bits proporcional al nivel de la entrada (valor digital). Los convertidores de señal análogo a digital abrevian ADC o A/D.

Uno de los parámetros que definen al A/D es la **resolución** como la mínima variación de voltaje en la entrada que produce cambio del valor digital en la salida. Por ejemplo un convertidor de 10 bits tiene un total de 2^{10} valores (1024 valores de 0 a 1023).

Si tenemos 10V a la entrada la resolución seria de 9,765mV. En este caso el voltaje es de 10V a 0V pero pueden variar. Por ejemplo si tenemos de 10v a 5v la resolución será:

Resolución= (10v – 5v)/1024=4.88 mV
Una formula para el calculo será:
Resolución= (Vref2-Vref1)/1024
donde las tensiones de referencia son 10V y 5V.

Descripción General

El módulo convertidor Análogo Digital (A/D) del PIC 16F877 tiene 8 canales de entrada. La conversión de la señal analógica aplicada (a uno de los canales) se plasma en número binario de 10 dígitos. El módulo A/D posee voltajes de referencia que pueden ser seleccionados para emplear las tensiones VDD, VSS del microcontrolador o puede emplear tensiones aplicadas a los pines RA2 o RA3 (incluso es posible establecer combinaciones de los anteriores valores).

Para operar el modulo ADC contamos con 4 registros:

- Registro de resultado de byte alto de la conversión A/D (ADRESH). Banco 0, 0x1E
- Registro de resultado de byte bajo de la conversión A/D (ADRESL). Banco 1, 0x9E
- Registro 0 de control del módulo A/D (ADCON0). Banco 0, 0x1F
- Registro 1 de control del móodulo A/D (ADCON1). Banco 1, 0x9F

El detalle del registro ADCON0 se muestra a continuación:

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
bit 7							bit 0

bit 7-6 ADCS1:ADCS0: A/D Conversion Clock Select bits (ADCON0 bits in bold)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	**00**	FOSC/2
0	**01**	FOSC/8
0	**10**	FOSC/32
0	**11**	FRC (clock derived from the internal A/D RC oscillator)
1	**00**	FOSC/4
1	**01**	FOSC/16
1	**10**	FOSC/64
1	**11**	FRC (clock derived from the internal A/D RC oscillator)

bit 5-3 CHS2:CHS0: Analog Channel Select bits
000 = Channel 0 (AN0)
001 = Channel 1 (AN1)
010 = Channel 2 (AN2)
011 = Channel 3 (AN3)
100 = Channel 4 (AN4)
101 = Channel 5 (AN5)
110 = Channel 6 (AN6)
111 = Channel 7 (AN7)

Note: The PIC16F873A/876A devices only implement A/D channels 0 through 4; the unimplemented selections are reserved. Do not select any unimplemented channels with these devices.

bit 2 GO/$\overline{\text{DONE}}$: A/D Conversion Status bit
When ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion which is automatically cleared by hardware when the A/D conversion is complete)
0 = A/D conversion not in progress

bit 1 Unimplemented: Read as '0'

bit 0 ADON: A/D On bit
1 = A/D converter module is powered up
0 = A/D converter module is shut-off and consumes no operating current

Y el detalle del registro ADCON1 se muestra a continuación:

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7 **ADFM:** A/D Result Format Select.bit
1 = Right justified. Six (6) Most Significant bits of ADRESH are read as '0'.
0 = Left justified. Six (6) Least Significant bits of ADRESL are read as '0'.

bit 6 **ADCS2:** A/D Conversion Clock Select bit (ADCON1 bits in shaded area and in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	00	FOSC/2
0	01	FOSC/8
0	10	FOSC/32
0	11	FRC (clock derived from the internal A/D RC oscillator)
1	00	FOSC/4
1	01	FOSC/16
1	10	FOSC/64
1	11	FRC (clock derived from the internal A/D RC oscillator)

bit 5-4 **Unimplemented:** Read as '0'

bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits

bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C / R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8 / 0
0001	A	A	A	A	VREF+	A	A	A	AN3	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	AN3	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	AN3	VSS	2 / 1
011x	D	D	D	D	D	D	D	D	—	—	0 / 0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6 / 2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6 / 0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5 / 1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4 / 2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3 / 2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2 / 2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1 / 0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1 / 2

A = Analog input D = Digital I/O
C / R = # of analog input channels / # of A/D voltage references

El registro ADCON1 configura las funciones de los pines de entrada al módulo. Como se aprecia se puede configurar los pines del puerto A como entradas analógicas inclusive la línea RA3 puede funcionar como el votaje de referencia.

Los registros ADRESH:ADRESL contienen el resultado de la conversión (10 bits). Cuando se ha completado una conversión el resultado es almacenado en ADRESH:ADRESL y además el bit GO/-DONE (registro ADCON bit 2) se pone a 0-lógico y el bit ADIF (registro PIR1 bit 7) se pone como 1-lógico. El registro PIR1 ocupa la posición 0x0C del banco 0. He aquí parte del registro que volveremos a tocar en el módulo de interrupciones:

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

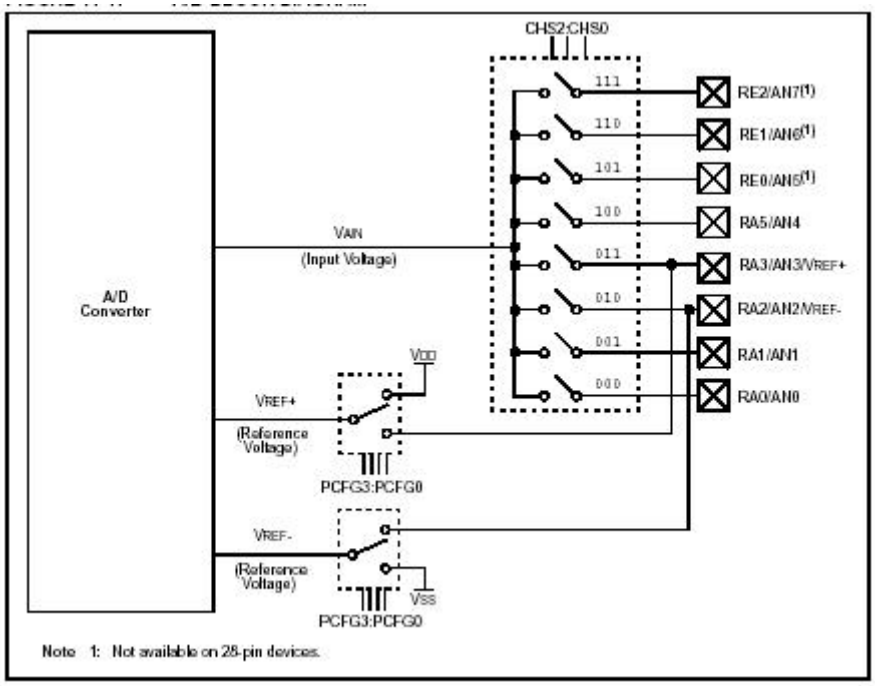
bit 7 **PSPIF:** Parallel Slave Port Read/Write Interrupt Flag bit(1)
1 = A read or a write operation has taken place (must be cleared in software)
0 = No read or write has occurred
Note 1: PSPIF is reserved on PIC16F873A/876A devices; always maintain this bit clear.

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

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

El diagrama de bloques del módulo A/D es:



Después que el módulo A/D ha sido configurado, es necesario esperar un periodo de tiempo para que la señal sea adquirida (antes de las conversiones empiecen). Cada uno de los canales de entrada tiene su correspondiente bit de configuración en los registros TRIS y estos han de ser puestos como entradas.

Una vez que el periodo de adquisición ha terminado la conversión A/D puede empezar. Los siguientes pasos muestran la secuencia de uso:

1. Configure el módulo A/D: (lo cual significa)
- Configurar los pines de entrada de los canales analógicos a usar. Configure los voltajes de referencia. (en el registro ADCON1)
- Seleccione el canal de entrada al módulo A/D (en el registro ADCON0)
- Seleccione el clock de conversión A/D (en el registro ADCON0)
- Ponga a funcionar el módulo A/D (en el registro ADCON0)

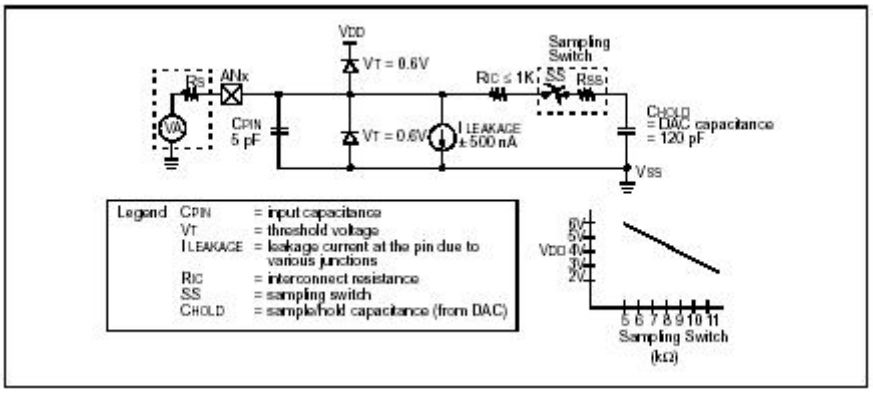
1. Configure la interrupción del A/D si lo desea:
- ADIF=0 -lógico (bit que indica si se produjo una conversión)
- ADIE=1-lógico (habilitador de interrupción del modulo analógico)
- PEIE=1 lógico (habilitador de interrupción de periféricos)
- GIE=1-lógico (Habilitador general de interrupciones)

1. Espere por el tiempo de adquisición (es el tiempo que se demora en trabajar el bloque de sampling and hold)
2. Comienzo de la conversión
   GO/-DONE=1-lógico (en el registro ADCON0)
3. Esperar a que la conversión se complete. Para saber si la conversión termino podemos:
- Revisar el bit GO/-DONE esperando que sea de nuevo 0 -lógico
- O esperar al flag de ADIF sea 1-lógico (puede emplearse como interrupción)

4. Leer el resultado del A/D en el par de registros (ADRESH:ADRESL). No debemos olvidar colocar el bit ADIF a 0-lógico (si se requiere)
5. Para la siguiente conversión, regrese al paso 1 o paso 2 dependiendo si solo se usa uno o mas canales.

Requerimientos para la adquisición A/D

Para que el modulo convertido A/D trabaje apropiadamente la carga del capacitor que muestrea la señal análoga debe ser la máxima .



Como se muestra en la figura, la impedancia de la fuente (Rs) y la impedancia del switch interno (Rss) afecta el tiempo requerido para la carga del capacitor. La máxima impedancia recomendada para la fuente analógica es de 10 K ohmios. Cuando la impedancia es menor el tiempo de adquisición es menor y por tanto la respuesta es mejor. Luego que el canal de entrada es seleccionado (o ha sido cambiado) no olvide esperar un tiempo a que la adquisición del dato sea hecha antes de que la conversión propiamente dicha empiece.

Para calcular el mínimo tiempo requerido en la adquisición podemos usar la ecuación:

$$
\begin{aligned}
T_{ACQ} &= \text{Amplifier Settling Time} + \text{Hold Capacitor Charging Time} + \text{Temperature Coefficient} \\
&= T_{AMP} + T_C + T_{COFF} \\
&= 2\mu s + T_C + [(\text{Temperature} - 25^\circ C)(0.05\mu s/^\circ C)] \\
T_C &= C_{HOLD}(R_{IC} + R_{SS} + R_S)\ln(1/2047) \\
&= -120pF\,(1k\Omega + 7k\Omega + 10k\Omega)\ln(0.0004885) \\
&= 16.47\mu s \\
T_{ACQ} &= 2\mu s + 16.47\mu s + [(50^\circ C - 25^\circ C)(0.05\mu s/^\circ C)] \\
&= 19.72\mu s
\end{aligned}
$$

Selección del clock de conversión Analógica Digital

Existe otro parámetro que es importante mencionar y es el tiempo de conversión de A/D POR BIT (definida como T_{AD}). La conversión A/D requiere un mínimo de 12 T_{AD} por 10 bits de conversión, La fuente para el clock en la conversión A/D se selecciona por software. Hay 7 posibles valores para la selección del T_{AD}:

- 2Tosc
- 4Tosc
- 8Tosc
- 16Tosc
- 32Tosc
- 64Tosc
- Oscilador RC interno que tiene el modulo A/D (crea un retardo de 2us a 6 us).

Para el correcto funcionamiento del módulo el clock de conversión debe ser seleccionado para asegurar un T_{AD} de 1.6 us como mínimo.

La siguiente tabla muestra el resultado de varios T_{AD} calculados para diferentes clock aplicados al microcontrolador.

AD Clock Source (TAD)		Maximum Device Frequency
Operation	ADCS2:ADCS1:ADCS0	Max.
2TOSC	000	1.25 MHz
4TOSC	100	2.5 MHz
8TOSC	001	5 MHz
16TOSC	101	10 MHz
32TOSC	010	20 MHz
64TOSC	110	20 MHz
RC(1, 2, 3)	x11	(Note 1)

Note 1: The RC source has a typical TAD time of 4 µs, but can vary between 2-6 µs.
2: When the device frequencies are greater than 1 MHz, the RC A/D conversion clock source is only recommended for SLEEP operation.
3: For extended voltage devices (LC), please refer to the Electrical Characteristics (Sections 17.1 and 17.2).

Configuración de los pines de los puertos para que trabajen de forma analógica

Los registros ADCON1 y TRIS controlan la operación de los pines de los canales A/D. Los pines que se empleen como entradas deben ser configurados en los registros TRIS como 1-lógico. Si en el registro TRIS se coloca a 0-lógico (como línea de salida) el módulo A/D convertirá el voltaje presente a la salida del pin. La operación de conversión del A/D es independiente del estado de los bits CHS2:CHS0 y de los bits de los registros TRIS

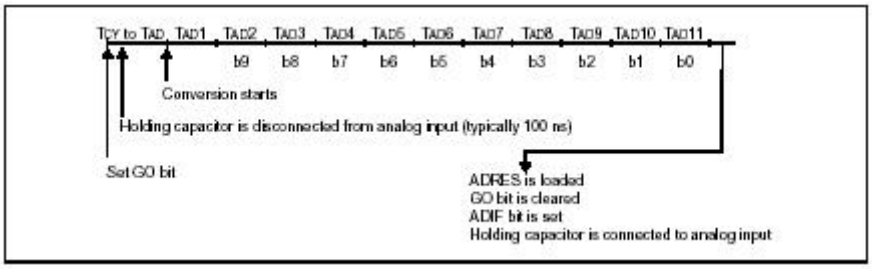
Nota

1. Si el puerto A ha sido configurado como analógico y tratamos de leer el registro del puerto (PORTA) lo que encontraremos serán 0-lógicos. Los pines configurados como salida digital serán leídos como entradas analógicas con el valor de voltaje presente en las líneas.
2. Si hubiera niveles analógicos en cualquier pin definido como entrada digital (incluyendo los pines AN7:AN0). Pueden causar en el buffer de entrada un consumo de corriente que esté fuera del rango de las especificaciones y por tanto dañar el microcontrolador.

Conversiones A/D

Si iniciamos una conversión y colocamos a 0-lógico el bit GO/-DONE conseguimos abortar la conversión A/D que se este llevando a cabo en ese momento. El resultado de la conversión no aparecerá en los registros ADRESH:ADRESL y se mantendrá el último valor convertido.

Después que la conversión A/D es abortada, la siguiente adquisición en el canal seleccionado empieza automáticamente. El bit GO/-DONE puede ser colocado a 1 para empezar la conversión.

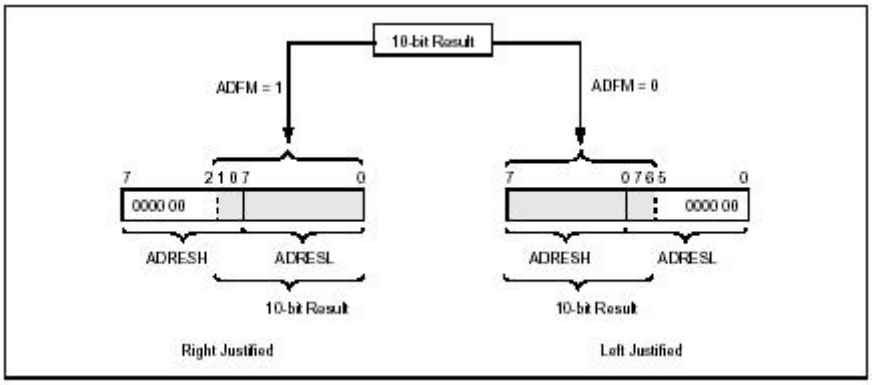


Como se aprecia en la figura después que el bit GO es colocado a 1-lógico comienza la conversión y al inicio hay que desconectar el condensador lo cual demanda como máximo 1 TAD, luego vienen los 10TAD correspondientes a los 10 bits del A/D y se necesita un TAD mas para depositar el resultado en los registros ADRES, colocar el bit GO/-DONE de nuevo a 0-lógico y el bit ADIF a 1-lógico.

Nota. EL bit GO/-DONE y el bit ADON están en el registro ADCON0 pero no debe ser activados a la vez, esto es en la misma instrucción.

Registros que almacenan el resultado de la conversión

El par de registros ADRESH:ADRESL almacenan el resultado de la conversión A/D. Este par de registros ocupan 16 bits. EL módulo A/D tiene la flexibilidad de colocar el resultado justificado a la derecha o a la izquierda de esos 16 bits (formato). El bit que selecciona el formato es el ADFM (registro ADCON1 bit 7). La figura muestra como el detalle de la justificación.



Los bits extras (6 bits) son llenados con 0-lógicos. Si en un programa no se emplea el módulo A/D es posible usar los registros ADREH :ADRESL como si fueran registros de propósito general (registros de 8 bits).

Operación del módulo A/D durante la operación SLEEP

El módulo A/D puede seguir operando durante el modo SLEEP, esto requiere que el clock A/D sea la red RC (ADCS1:ADCS0=111). Cuando el clock RC es seleccionado, el módulo A/D espera un cliclo de instrucción antes de empezar la conversión. Esto le permite a la instrucción SLEEP ser ejecutada eliminando todos los ruidos digitales producidos por el swticheo en la conversión.

Cuando la conversión es completada, el bit GO/-DONE es colocado a 0-lógico y el resultado es cargado a los registros ADRES. Si está habilitada la interrupción del módulo A/D el microcontrolador se despierta o sale del modo SLEEP.

Si las interrupciones del módulo A/D no están habilitadas, el módulo A/D será apagado pese a que el bit ADCON sigue aun en 1-lógico. Cuando se apaga el modulo A/D se tiene el mínimo consumo de corriente.

Nota. Para que el módulo A/D opere durante el modo SLEEP la fuente del clock A/D debe ser siempre el RC (ADCS1:ADCS0=11). Para permitir que la conversión ocurra durante el

modo SLEEP asegurece de colocar la instrucción SLEEP inmediatamente después de colocar el bit GO/-DONE a 1-lógico.

Efectos en el RESET

Después de un RESET el módulo A/D esta apagado y si había una conversión en curso ésta es abortada. Todos los pines que entran al modulo A/D son configurados como entradas analógicas. El valor presente en ADRESH:ADRESL no se modifica después de un RESET. Luego de encender el microcontrolador el valor presente en ADRESH:ADRESL es aleatorio (basura).

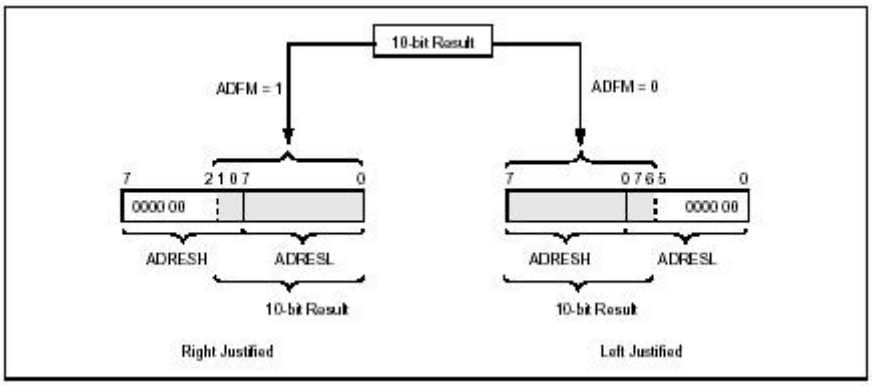
Ejercicios

Ejercicio 13

Diseñar un programa que permita leer el voltaje aplicado al canal 0 del módulo ADC, convertirlo a un valor digital de 10 bits y mostrar los ocho bits mas significativos en el PORTC

a)Definiciones previas

El módulo del ADC cuenta con dos registros ADRESH y ADRESL donde se almacena automaticamente la conversión digital de 10bits.
Sin embargo es posible seleccionar a cual de los dos registros se va a leer como ceros los 6 bits que restan.



Con ayuda del bit ADFM del registro ADCON1 podemos seleccionar la distribución de los 10 bits. Por ejemplo si ADFM=1 los bits estaran asignados entre los 8 bits de ADRESL y los 2 menos significativos de ADRESH.
Por otro lado si ADFM=0 los bits estarán asignados a los 8 bits de ADRESH y los 2 mas significativos de ADRESL. Para el programa a diseñar vamos a asignar el valor de 0 a ADFM.

Luego es importante recordar que el conversor necesita un tiempo de adquisición antes de empezar la conversión por lo que es recomendable generar por software un retardo que como mínimo debe ser de 19,72 us. En el programa a diseñar realizaremos un retardo con ayuda del TMR0 y asignar el preescaler 1:256 hasta que desborde, obteniendo asi un retardo de 65 us. Si bien es cierto este retardo esta sobredimensionado, es importante recalcar que el tiempo de adquisición puede crecer en función a la impedancia de entrada. (max impedancia recomendada = 10kohm)

b)Diseño del programa

La configuración del registro ADCON0 sera de la siguiente manera:

ADCS1 ADCS0 CHS2 CHS1 CHS0 GO/DONE X ADON
0 1 0 0 0 0 0 1

La configuración del registro ADCON1 será de la siguiente manera:

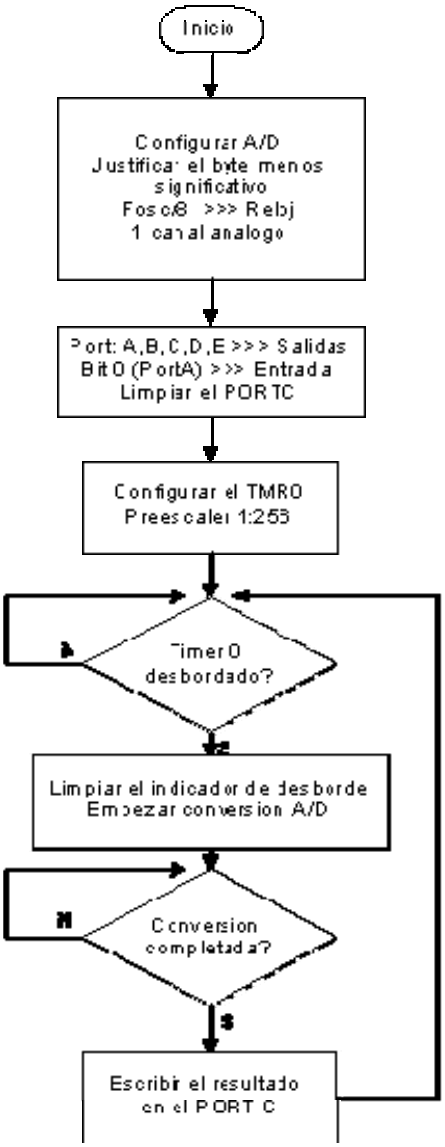
ADFM X X X PCFG3 PCFG2 PCFG1 PCFG0
0 0 0 0 1 1 1 0

La configuración del registro OPTION_REG sera de la siguiente manera:

X X TOCS TOSE PSA PS2 PS1 PS0
0 0 0 0 0 1 1 1

Preescaler 256
Programa principal
1.- Ir banco 0
2.- ADCON= b'01000001'
3.- Ir banco 1
4.- Puertos: A,B,C,D,E >> Salidas
5.- Linea AN0 como entrada
6.- OPTION_REG = b'00000111'
7.- ADCON1 = b'00001110'
8.- Banco 0
9.- Limpiar PuertoC
10.- Preguntar si TMR0 desbordo INTCON<TOIF> si no esperar
11.- Limpiar indicador de desborde
12.- Empezar conversion
13.- Preguntar si termino la conversion. ADCON<GO>si no esperar
14.- PORTC = ADRESH mover el dato al puertoC
15.- Ir paso 10

Diagrama de flujo



Código del programa:

```
    list p=16f877 ;Comando que indica el Pic usado
    include "p16f877.inc" ;Etiquetas genéricas para el Pic16F877

org 0x000 ;Inicio del programa en la posición cero de memoria
    nop ;Libre (uso del debugger)
    nop

_inicio
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1

    movlw b'01000001' ;A/D conversion Fosc/8
    movwf ADCON0

    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1

    clrf TRISA ;PORTA salida
```

```
    clrf TRISB ;PORTB salida
    clrf TRISC ;PORTC salida
    clrf TRISD ;PORTD salida
    clrf TRISE ;PORTE salida

    movlw b'00000111'
    movwf OPTION_REG ;TMR0 preescaler, 1:156

    movlw b'00001110' ;A/D Port AN0/RA0
    movwf ADCON1

    bsf TRISA,0 ;RA0 linea de entrada para el ADC
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    clrf PORTC ;Limpiar PORTC

_bucle
    btfss INTCON,T0IF
    goto _bucle ;Esperar que el timer0 desborde
    bcf INTCON,T0IF ;Limpiar el indicador de desborde
    bsf ADCON0,GO ;Empezar la conversion A/D

_espera
    btfsc ADCON0,GO ;ADCON0 es 0? (la conversion esta completa?)
    goto _espera ;No, ir _espera
    movf ADRESH,W ;Si, W=ADRESH
    movwf PORTC ;Muestra el resultado en PORTC

goto _bucle ;Ir bucle
    end
```

Ejercicio 14

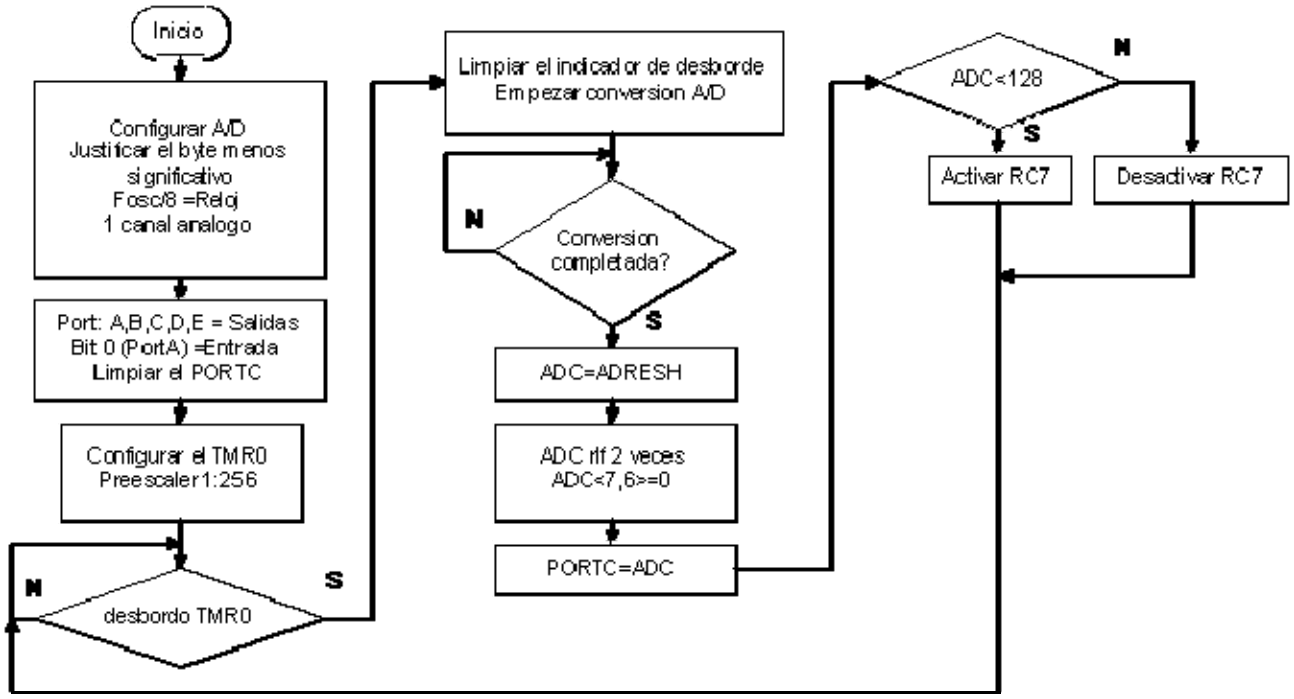
Elaborar un programa que lea el canal 0 del modulo ADC y que muestre el resultado en los 6 bits menos significativos del PORTC (RC0-RC5) adicionalmente active un pin del Pic (RC7) cuando el valor adquirido por el ADC sea mayor a 512 y desactive cuando sea menor a ese valor.

a) Definiciones previas:
Para el desarrollo del presente ejercicio es importante recordar la instrucción “rrf f,d” que permite rotar los bits de un registro hacia la derecha, de manera que el dato que obtengamos del ADC se rote a la derecha y podamos obtener la data en los 6 bits menos significativos del PuertoC.

b) Diseño del programa
Algoritmo
1.- Ir banco 0
2.- ADCON= b'01000001'
3.- Ir banco 1
4.- Puertos: A,B,C,D,E >> Salidas
5.- Linea AN0 como entrada
6.- OPTION_REG = b'00000111'
7.- ADCON1 = b'00001110'
8.- Banco 0
9.- Limpiar PuertoC
10.- Preguntar si TMR0 desbordo INTCON<TOIF> si no esperar
11.- Limpiar indicador de desborde
12.- Empezar conversion
13.- Preguntar si termino la conversion. ADCON<GO>si no esperar

14.- Registro ADC = ADRESH
15.- Rotar 2 veces a la derecha el registro ADC
16.- Preguntar: ADC es mayor a 128, si no PORTC<7>=0
17.- PORTC<7>=1
18.- Ir paso 10

Diagrama de Flujo:



Código del programa:

```
    list p=16f877 ;Comando que indica el Pic usado
    include "p16f877.inc" ;Etiquetas genéricas para el Pic16F877

    ADC EQU 0x20

org 0x00 ;Inicio del programa en la posición cero de memoria
    nop ;Libre (uso del debugger)

_inicio
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    movlw b'01000001' ;A/D conversion Fosc/8
    movwf ADCON0
    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1

    clrf TRISA ;PORTA salida
    clrf TRISB ;PORTB salida
    clrf TRISC ;PORTC salida
    clrf TRISD ;PORTD salida
    clrf TRISE ;PORTE salida

    movlw b'00000111'
    movwf OPTION_REG ;TMR0 preescaler, 1:156
    movlw b'00001110' ;A/D Port AN0/RA0
```

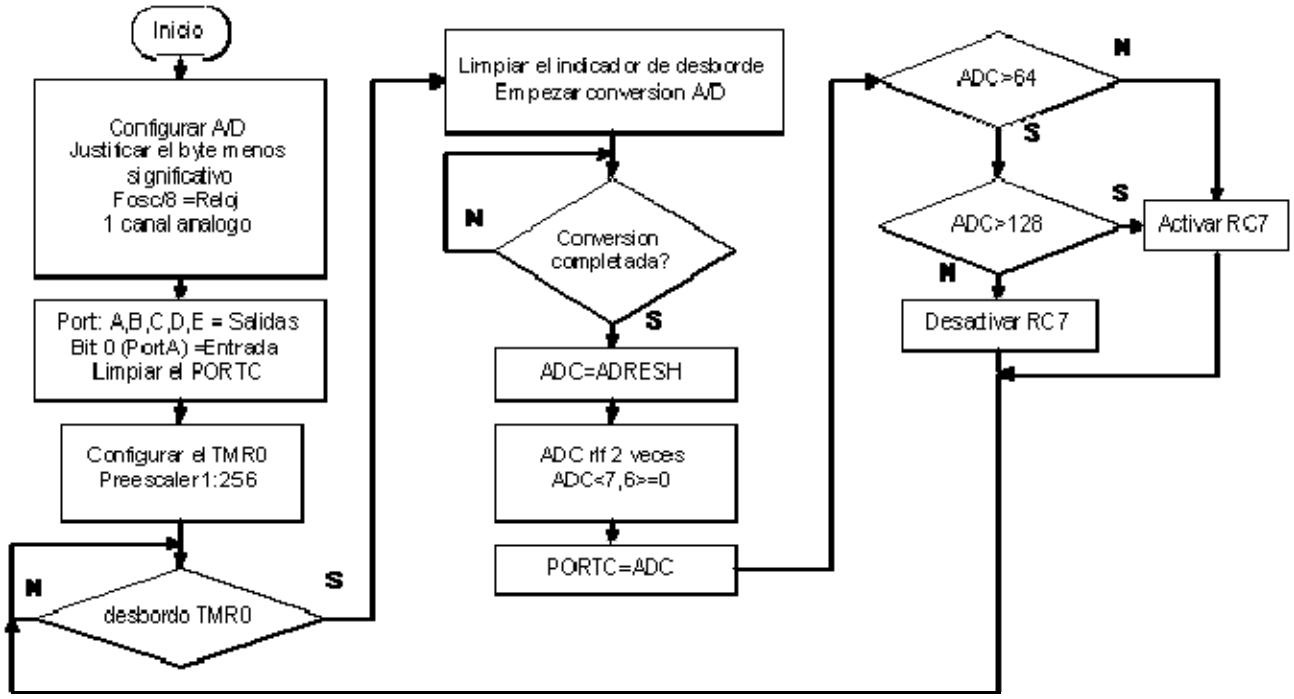
```
    movwf ADCON1
    bsf TRISA,0 ;RA0 linea de entrada para el ADC
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    clrf PORTC ;Limpiar PORTC

_bucle
    btfss INTCON,T0IF
    goto _bucle ;Esperar que el timer0 desborde
    bcf INTCON,T0IF ;Limpiar el indicador de desborde
    bsf ADCON0,GO ;Comenzar conversion A/D

_espera
    btfsc ADCON0,GO ;ADCON0 es 0? (la conversion esta completa?)
    goto _espera ;No, ir _espera
    movf ADRESH,W ;Si, W=ADRESH
    movwf ADC ;ADC=W
    rrf ADC,F ;ADC /4
    rrf ADC,F
    bcf ADC,7
    bcf ADC,6
    movfw ADC ;W = ADC
    movwf PORTC ;PORTC = W
    movlw D'32' ;Comparamos el valor del ADC para saber si es menor
que 128
    subwf ADC,W
    btfss STATUS,C ;Es mayor a 128?
    goto _desactivar ;No, desactivar RC7
    bsf PORTC,7 ;Si, RC7 = 1 logico
    goto _bucle ;Ir bucle
_desactivar
    bcf PORTC,7 ;RC7 = 0 logico
    goto _bucle ;Ir bucle
    end
```

Ejercicio 15

Un tanque es empleado para almacenar agua y se ha conectado un sensor para saber el nivel de agua. El sensor se ha conectado al canal 0 del Pic 16F877 y un indicador es accionado por el pin RC7. Desarrollar un programa que informe si el nivel de agua esta en el rango de 64 a 128 unidades del ADC RC7=0 (RC7=1 fuera del rango)

El ejercicio es un ejemplo simple de una aplicación a un tanque que necesita controlar el nivel de líquido. Y se tiene un setpoint entre 64 y 128. Es decir dentro de este rango el indicador (RC7) estará apagado.
Cuando el valor del adc se encuentre fuera del rango entre 64 y 128; el indicador (RC7) que representa una alarma se encenderá.

Diagrama de flujo



Código del programa

```
    list p=16f877 ;Comando que indica el Pic usado
    include "p16f877.inc" ;Etiquetas genéricas para el Pic16F877

    ADC EQU 0x20
org 0x000 ;Inicio del programa en la posición cero de memoria
    nop ;Libre (uso del debugger)
_inicio
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    movlw b'01000001' ;A/D conversion Fosc/8
    movwf ADCON0
    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1
    clrf TRISA ;PORTA salida
    clrf TRISB ;PORTB salida
    clrf TRISC ;PORTC salida
    clrf TRISD ;PORTD salida
    clrf TRISE
    movlw b'00000111'
    movwf OPTION_REG ;TMR0 preescaler, 1:156
    movlw b'00001110' ;A/D Port AN0/RA0
    movwf ADCON1
    bsf TRISA,0 ;RA0 linea de entrada para el ADC
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    clrf PORTC ;Limpiar PORTC
_bucle
    btfss INTCON,T0IF
    goto _bucle ;Esperar que el timer0 desborde
    bcf INTCON,T0IF ;Limpiar el indicador de desborde
    bsf ADCON0,GO ;Start A/D conversion
_espera
    btfsc ADCON0,GO ;ADCON0 es 0? (la conversion esta completa?)
    goto _espera ;No, ir _espera
```

```
    movf ADRESH,W ;Si, W=ADRESH
    movwf ADC ;ADC=W
    rrf ADC,F ;ADC /4
    rrf ADC,F
    bcf ADC,7
    bcf ADC,6
    movfw ADC ;W = ADC
    movwf PORTC ;PORTC = W
    movlw D'16' ;Comparamos el valor del ADC para saber si es menor
que 64
    subwf ADC,W
    btfss STATUS,C ;Es mayor a 64?
    goto _activar ;No, ir _activar
    goto _mayor ;Si, es mayor
_activar
    bsf PORTC,7 ;RC7 = 1 logico
    goto _bucle ;Ir bucle
_mayor
    movlw D'32' ;Comparamos el valor del ADC, es menor que 128?
    subwf ADC,W
    btfss STATUS,C ;Es mayor a 128?
    goto _desactivar ;No, ir desactivar
    goto _activar ;Si, ir _activar
_desactivar
    bcf PORTC,7 ;RC7 = 0 logico
    goto _bucle ;Ir bucle
end
```

Módulo 4: Comunicación Serie Asíncrona

Generalidades

Entre las herramientas que disponen los PIC16F8x se encuentra el USART, llamado SCI (Serial Comunications Interface), puede funcionar como un sistema de comunicación bidireccional, adaptándose a multitud de periféricos y dispositivos que transfieren información de forma serial, tales como un ordenador. También puede trabajar en modo unidireccional para soportar periféricos como memorias, conversores, etc.
El USART (Transmisor/Receptor Síncrono/Asíncrono Serie) puede trabajar de dos formas:

- Asíncrono (Bidireccional)
- Síncrono (Unidireccional)

En el modo asíncrono, la comunicación serie del USART en los PIC16F8x esta soportada por las líneas RC6/TX/CK y RC7/RX/DT por las que se mueven los bits a la frecuencia interna de reloj. En el modo síncrono, los bits de información circulan en ambos sentidos por la línea DT a la frecuencia de los impulsos que genere el maestro por la linea CK.

En el presente módulo vamos a enfocar el estudio del modo asíncrono.

En esta forma de comunicación serie, se usa la norma RS-232-C, donde cada palabra de información o dato se envía independientemente de los demás. Suele constatar de 8 o 9 bits y van precedidos por un bit de START (inicio) y detrás de ellos se coloca un bit de STOP (parada), de acuerdo con las normas del formato estandar NRZ (Non Return-to-Zero) Los bits se transfieren a una frecuencia fija y normalizada. La USART transmite y recibe primero el bit menos significativo.

La USART en modo asíncrono contiene los siguientes elementos:

- Generador de Baudios
- Circuito de Muestreo
- Transmisor Asíncrono
- Receptor Asíncrono

Generador de Baudios

Para el protocolo asíncrono RS-232-C, la frecuencia en baudios (bits por segundo) a la que se realiza la transferencia se debe efectuar a un valor normalizado: 330, 600, 1200, 2400, 4800, 9600, 19200, 38400, etc. Para generar esta frecuencia, el USART dispone de un generador de frecuencia en Baudios, BRG, cuyo valor es controlado por el contenido grabado en el registro SPBRG.

Aparte del valor X cargado en el registro SPBRG, la frecuencia en baudios del generador depende del bit BRGH del registro TXSTA <2>. En el caso de que BRGH = 0 se trabaja en

baja velocidad y si BRGH = 1 se trabaja en alta velocidad. Según este bit se obtendrá el valor de una constante K necesaria en la determinación de la frecuencia de funcionamiento.

$$Frecuencia\,(baudios) = \frac{Fosc}{(K*(X+1))}$$

X es el valor cargado en el registro SPBRG

BRG

Si BRG = 0, baja velocidad y K = 64

Si BRG = 1, baja velocidad y K = 16

$$X = \frac{Fosc}{Frecuencia*K} - 1$$

TABLE 10-3: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 0)

BAUD RATE (K)	FOSC = 20 MHz			FOSC = 16 MHz			FOSC = 10 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	1.221	1.75	255	1.202	0.17	207	1.202	0.17	129
2.4	2.404	0.17	129	2.404	0.17	103	2.404	0.17	64
9.6	9.766	1.73	31	9.615	0.16	25	9.766	1.73	15
19.2	19.531	1.72	15	19.231	0.16	12	19.531	1.72	7
28.8	31.250	8.51	9	27.778	3.55	8	31.250	8.51	4
33.6	34.722	3.34	8	35.714	6.29	6	31.250	6.99	4
57.6	62.500	8.51	4	62.500	8.51	3	52.083	9.58	2
HIGH	1.221	-	255	0.977	-	255	0.610	-	255
LOW	312.500	-	0	250.000	-	0	156.250	-	0

BAUD RATE (K)	FOSC = 4 MHz			FOSC = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	0.300	0	207	0.3	0	191
1.2	1.202	0.17	51	1.2	0	47
2.4	2.404	0.17	25	2.4	0	23
9.6	8.929	6.99	6	9.6	0	5
19.2	20.833	8.51	2	19.2	0	2
28.8	31.250	8.51	1	28.8	0	1
33.6	-	-	-	-	-	-
57.6	62.500	8.51	0	57.6	0	0
HIGH	0.244	-	255	0.225	-	255
LOW	62.500	-	0	57.6	-	0

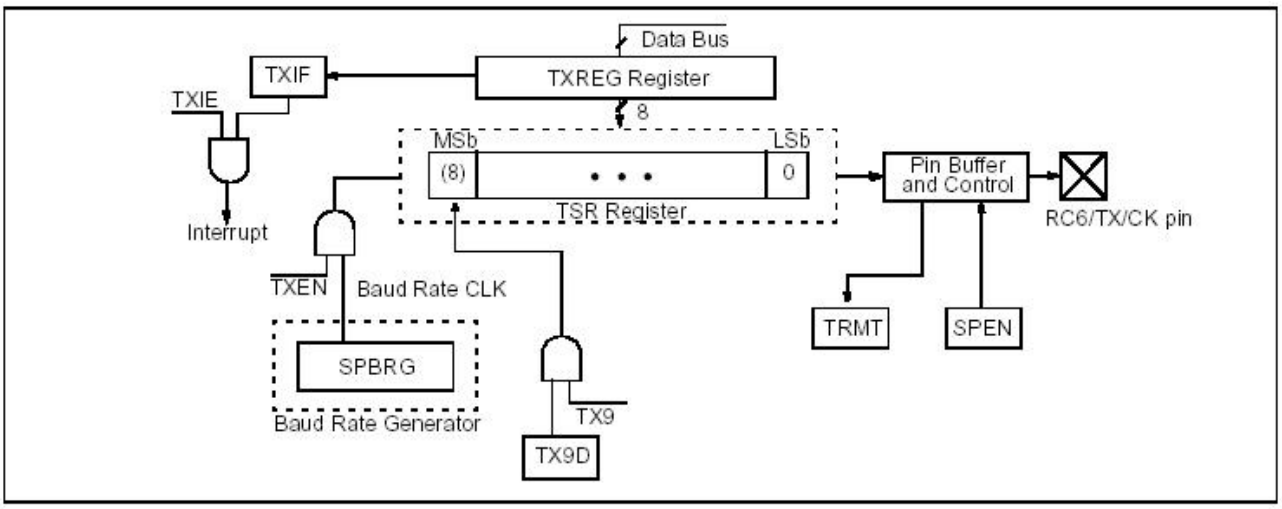
TABLE 10-4: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 1)

BAUD RATE (K)	FOSC = 20 MHz			FOSC = 16 MHz			FOSC = 10 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	-	-	-	-	-	-	-	-	-
2.4	-	-	-	-	-	-	2.441	1.71	255
9.6	9.615	0.16	129	9.615	0.16	103	9.615	0.16	64
19.2	19.231	0.16	64	19.231	0.16	51	19.531	1.72	31
28.8	29.070	0.94	42	29.412	2.13	33	28.409	1.36	21
33.6	33.784	0.55	36	33.333	0.79	29	32.895	2.10	18
57.6	59.524	3.34	20	58.824	2.13	16	56.818	1.36	10
HIGH	4.883	-	255	3.906	-	255	2.441	-	255
LOW	1250.000	-	0	1000.000		0	625.000	-	0

BAUD RATE (K)	FOSC = 4 MHz			FOSC = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-
1.2	1.202	0.17	207	1.2	0	191
2.4	2.404	0.17	103	2.4	0	95
9.6	9.615	0.16	25	9.6	0	23
19.2	19.231	0.16	12	19.2	0	11
28.8	27.798	3.55	8	28.8	0	7
33.6	35.714	6.29	6	32.9	2.04	6
57.6	62.500	8.51	3	57.6	0	3
HIGH	0.977	-	255	0.9	-	255
LOW	250.000	-	0	230.4	-	0

Trasmisor Asíncrono

La figura muestra el diagrama por bloques de la sección de transmisión del USART en modo asíncrono.



El dato que se desea transmitir por el USART se deposita en el registro TXREG y a continuación se traspasa al registro de desplazamiento TSR, que va sacando los bits secuencialmente y a la frecuencia establecida. Ademas, antes de los bits del dato de información incluye un bit de inicio y después de sacar todos los bits añade un bit de parada. El USART receptor recibe, uno a uno, los bits, elimina los de control y los de información una vez que han llenado el registro de desplazamiento RSR los traslada automáticamente al registro RCREG, donde quedan disponibles para su posterior procesamiento.

Si observamos el diagrama de bloques de la sección transmisora del USART. El núcleo esta constituido por el registro de desplazamiento TSR, que obtiene el dato desde el registro TXREG y luego lo va desplazando y sacando bit a bit, en serie, por la línea RC6/TX/CK. El primer bit que sale es el de menos peso. El dato a transferir se carga por software en TXREG y se transfiere al TSR en cuanto se haya transmitido el bit de parada del dato anterior. LA transferencia entre los dos registros se realiza en un ciclo y entonces el señalizador TXIF se pone a 1, para advertir que el registro de transmisión se ha vaciado. También en este momento puede producirse una interrupción si se ha posibilitado el uso de interrupciones. Cuando se escribe otro dato sobre TXREG, l señalizador TXIF se pone a 0. El bit TRMT sirve para indicar el estado del registro TSR y vale 1 cuando esta vacío.

La secuencia de pasos a seguir para una transmisión en el USART es la como sigue:

1. Configurar las líneas RC6/TX/CK como salida y RC7/RX/DT como entrada.
2. Asignar SYNC=0 y SPEN=1 para activar el USART como asíncrono.
3. Si se va a trabajar con interrupción, asignar TXIE=1, además de habilitar las interrupciones.
4. Si el dato consta de 9 bits, en lugar de los 8 típicos, asignar el bit TX9=1. El noveno bit se colocará en TX9D (TXSTA)

5. Se carga el valor adecuado en el registro SPBRG, para producir la frecuencia de trabajo deseada. Hay que controlar el bit BRGH (alta y baja velocidad)
6. Activar la transmisión con TXEN = 1. El bit TXIF tendra valor 1; ya que TXREG se encuentra vacio.
7. Cargar en TXREG el dato a transmitir. Comienza la transmisión.

TXSTA: TRANSMIT STATUS AND CONTROL REGISTER (ADDRESS 98h)

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

bit 7 **CSRC:** bit de selección de reloj
Modo asíncrono: no incluye
Modo síncrono
1 = Modo maestro (reloj generado internamente desde BRG)
0 = Modo esclavo (reloj generado por una fuente externa)
bit 6 **TX9 :** Habilita el bit 9 de transmisión
1 = Selecciona transmisión de 9 bits
0 = Selecciona transmisión de 8 bits
bit 5 **TXEN**: Activa la transmisión
1 = Transmisión activada
0 = Transmisión desactivada

Nota: SREN/CREN anula TXEN en modo síncrono.

bit 4 **SYNC**: Bit de selección del modo del USART
1 = Modo síncrono
0 = Modo asíncrono

bit 3 **Unimplementado: Leido como** '0'

bit 2 **BRGH**:Bit de selección de la velocidad de baudios

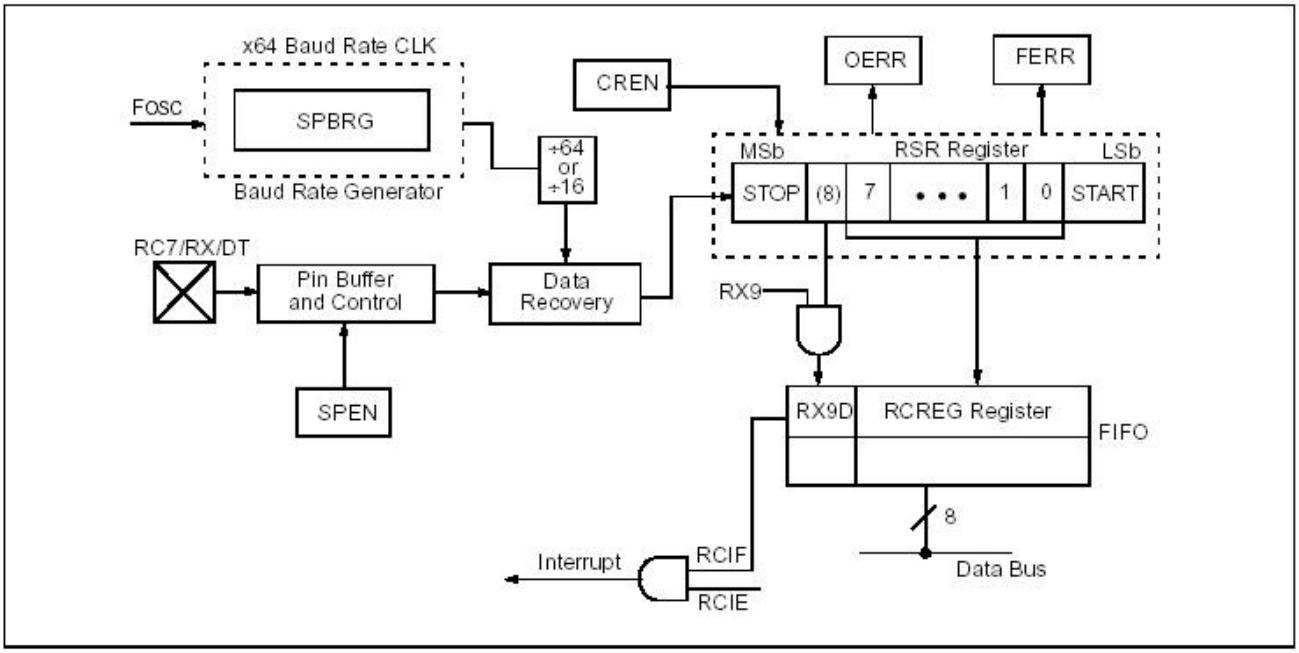
Modo asíncrono:
1 = Alta velocidad
0 = Baja velocidad
Modo Sincrono:
No se usa en este modo

bit 1 **TRMT**: Bit de estado del registro de desplazamiento de transmisión
1= TSR vacío
0 = TSR no vacío

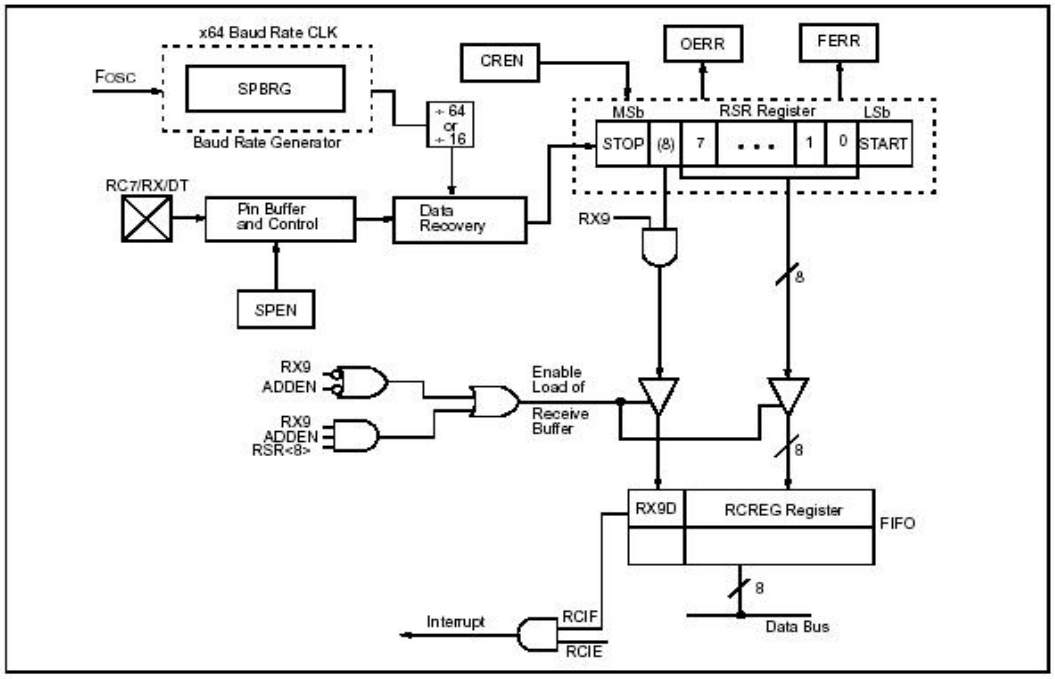
bit 0 **TX9D:** Bit 9 del dato a transmitir (puede ser el bit de paridad)

Receptor Asíncrono

La figura muestra el diagrama de bloques de la sección receptora del USART



Los datos se reciben en serie, bit a bit, por la linea RC7/RX/DT y se van introduciendo secuencialmente en el registro de desplazamiento RSR que funciona a una frecuencia 16 veces más rapida que la de trabajo (baud rate). Cuando el dato consta de 9 bits hay que programar el bit RX9 = 1 y el noveno bit de información se colocará en el bit RX9D del registro RCSTA.

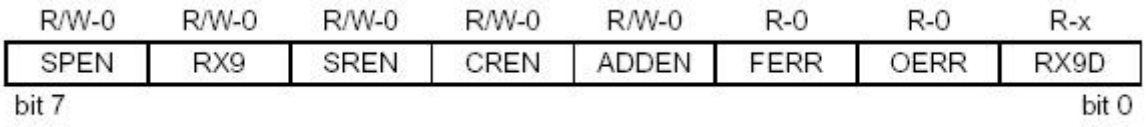


Una vez que el modo asíncronoha sido seleccionado, la recepción es habilitada colocando a 1-lógico el bit CREN (RCSTA<4>). El corazón de la recepción serial es el registro de corrimiento de recepción (RSR). Después de detectar el bit de parada la data presente en RSR es transferida al registro RCREG(si es que está vacío). Si la transferencia ha sido completada con éxito el RCIF (PIR1<5>) es puesto a 1-lógico. Si lo deseamos podemos usar

esta característica para trabajar interrupciones. Para ello deberemos habilitar el bit RCIE (PIE1<5>). El bit RCIF solo es de lectura y colocado a 0-lógico por hardware ; es decir cuando el registro RCREG esta vacío. Al parecer el registro RCREG es un solo registro pero no es asi; el registro RCREG es una suerte de pila de dos posiciones. Por tanto es capaz de almacenar 2 datos (bytes) y mantener un tercer dato en RSR. Si la pila está llena y se llena el RSR (tercer dato) es decir llega el bit de stop de ese tercer dato el microcontrolador procede a colocar a 1-lógico el bit OERR (over run error bit) y el valor presente en RSR se pierde. De darse esta situación lo que debemos hacer es rescatar los dos datos que estan en la pila de RCREG. Luego resetear el OERR, para eso es necesario resetear el bit CREN (ponerlo a 0-lógico y luego a 1-lógico). Cuando se activa el bit OERR las transferencias de RSR a RCREG son deshabilitadas por lo tanto debemos cumplir con hacer el reset del bit CREN para normalizar la recepción. Hay otro bit que también es importante señalar y es el bit FERR (RCSTA<2>) error de frame o trama o marco. El bit FERR se coloca a 1-lógico si detecta un 0-lógico como bit de parada. Por lo tanto antes de leer lo que contiene el registro RCREG es necesario revisar el valor de los bits FERR y OERR.

La siguiente es la secuencia de pasos a realizar para configurar la recepción asíncrona:

1. Inicializar el registro SPBRG con el valor apropiado que genere los baudios necesarios. No olvide colocar un valor al bit BRGH en función a si va o no a transmitir en alta velocidad.
2. Habilite la puerta serial asíncrona colocando a 0-lógico el bit SYNC y a 1-lógico el bit SPEN.
3. Si se va a trabajar con interrupción, asignar RCIE=1, además de habilitar las interrupciones.
4. Habilite la recepción colocando a 1-lógico el bit CREN.
5. El bit RCIF se colocará a 1-lógico cuando un dato llegue completo al microcontrolador y una interrupción se generará si es que se ha seteado el bit RCIE.
6. Lea el registro RCSTA para obtener el valor del noveno bit (si es que esta trabajndo con 9 bits de datos) y determine si hubo error en la comunicación (revise los bits OERR y FERR).
7. Si no hubo error lea los 8 bits de datos del registro RCREG.
8. Si no hubo algun error resetee el bit CREN.



bit 7 **SPEN:** Habilitación del puerto serie
1 = Puerto serie habilitado (configures RC7/RX/DT and RC6/TX/CK pins as serial port pins)
0 = Puerto serie deshabilitado

bit 6 **RX9**: Habilita el bit 9 de recepción
1 = Selecciona recepción de 9 bits
0 = Selecciona recepción de 8 bits

bit 5 **SREN**: Configura la recepción sencilla
Modo asíncrono no incluye
Modo síncrono maestro
1 = Habilita recepción sencilla
0 = Deshabilita recepción sencilla
Modo síncrono esclavo no se utiliza

bit 4 **CREN**: Configura la recepción continua
Modo asíncrono:
1 = Habilita modo de recepción continua
0 = Deshabilita recepción continua
Modo síncrono:
1 = Habilita recepción continua hasta que el bit CREN es borrado
0 = Deshabilita recepción continua

bit 3 **ADDEN:** Detección de dirección
Modo asíncrono con 9 bits (RX9 = 1):
1 =Activa la detección de dirección, activa la interrupción y descarga el buffer de recepcion al activarse
RSR<8>
0 =Desactiva la detección de dirección, todos los bits son recibidos y el bit 9 puede ser utilizado como bit de
paridad

bit 2 **FERR**: Bit de error de trama
1 = Error de trama (puede ser actualizado leyendo el registro RCREG y recibir el siguiente dato válido
0 = No hay error de trama

bit 1 **OERR**: Bit de error de sobrepasamiento
1 = Error de sobrepasamiento (puede ser borrado escribiendo un 0 en el bit CREN)
0 = No hay error de sobrepasamiento

bit 0 **RX9D:** Bit 9 del dato recibido (Puede ser el bit de paridad)

Ejercicios

Ejercicio 16

Transmitir muestras analógicas provenientes del canal AN= del ADC del PIC hasta la computadora y visualizar el dato por el hyper terminal de Windows.

a) Definiciones previas

- Para el desarrollo de este programa es recomendable contar con un resonador por que nos brinda una frecuencia mas exacta que la nos da una red RC. En los ejercicios que se muestran a continuación asumimos que estamos trabajando con un cristal de 4MHz.
- Necesitamos de un programa que nos permita recibir o transmitir datos al microcontrolador por eso emplearemos el Hyper terminal que es un accesorio de comunicaciones que encontramos en Windows. La configuración es bastante sencilla y la configuración debe coincidir con la que hemos seleccionado para en el PIC.
- Este programa lee un dato del canal AN0 del A/D y lo transmite a la PC a (2400 baudios:8 bits de datos:ninguna paridad:1 bit de parada y ningún control de flujo). El manejo del A/D es similar a los programas anteriores con la diferencia que no usamos el timer 0 para generar el retardo (mínimo 19,2 us) para el tiempo de adquisición sino que las instrucciones que empleamos en el programa nos permiten asegurar el tiempo apropiado.

b) Diseño del programa

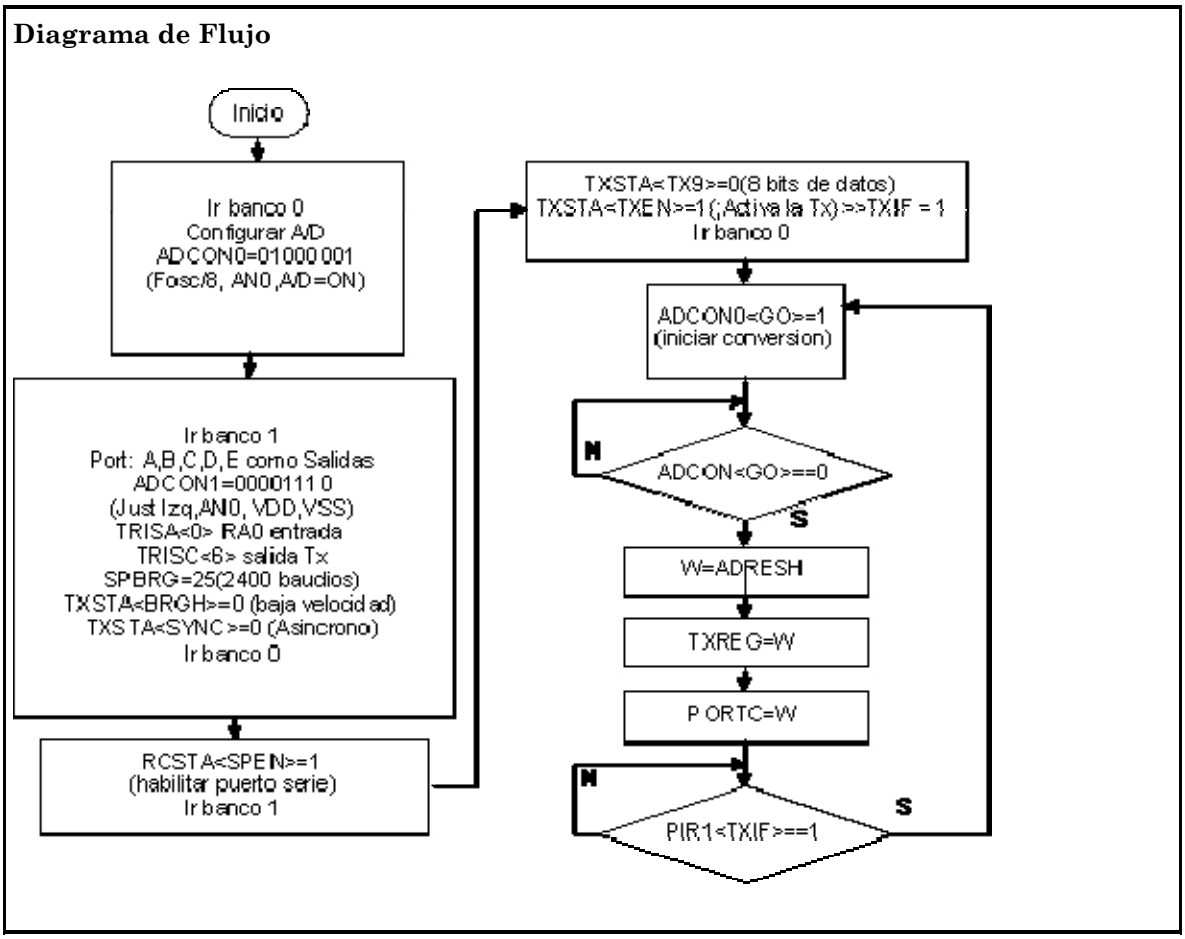
Algoritmo

1. Ir banco 0.
2. **ADCON0: ADCS1 ADCS0 CH2 CH1 CH0 GO/-DONE ----- ADON**

0 1 0 0 0 0 --- 1
Seleccionamos el canal 0 del A/D (CH2-0), encendemos el modulo (ADCON=1) y elegimos el TAD Fosc/8 (ADS1:0)

3. Ir banco 1.
4. **TRISA=TRISB=TRISC=TRISD=TRISE=Salidas**
5. **ADCON1: ADMF ADCS2 ------ ------ PCFG3 PCFG2 PCFG1 PCFG0**
   0 0 0 0 1 1 1 0
   Justificamos el resultado de los registros ADRES a la izquierda (ADMF=0). Solo el pin RA0 será canal analógico y las tensiones de referencia para el ADC serán $V_{REF+}=V_{DD}$ y $V_{REF-}= V_{SS}$
6. **TRISA<0>=0**, configurar el pin RA0 como entrada.
7. **TRISC<6>=1**, configurar el pin RC6 como salida para la Tx serial.
8. **SPBRG =25** (si y solo si el Cristal es de 4MHz)para lograr una frecuencia de 2400 baudios (referirse a la tabla).
9. **TXSTA<BRGH>=0** para baja velocidad de acuerdo a la tabla del registro SPBRG
10. **TXSTA<SYNC>=0** configuramos en modo asíncrono
11. Ir banco 0
12. **RCSTA<SPEN>=1** habilitamos la puerta serial.
13. Ir banco 1
14. **TXSTA<TX9>=0** , elegimos 8 bits de datos para la transmisión.
15. TXSTA<TXEN>=1, habilita la transmisión, automáticamente el bit TXIF se va a 1-lógico
16. **GO/-DONE=1**. Iniciar la conversión A/D.
17. Si GO/-DONE es 1 ir paso 17
18. Si GO/-DONE es 0 ir paso 19
19. TXREG=PORTC=ADRESH
20. Si TXIF=0 Ir paso 20
21. Si TXIF=1 Ir paso 16

Diagrama de Flujo



Código del programa

```
    include "p16f877.inc"

    ADDR_L equ 0x20
    DATA_L equ 0x21
org 0x00
    nop
    nop

    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1

    movlw b'01000001' ;A/D conversion Fosc/8
    movwf ADCON0

    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1

    clrf TRISA ;PORTA salida
    clrf TRISB ;PORTB salida
    clrf TRISC ;PORTC salida
    clrf TRISD ;PORTD salida
    clrf TRISE

    movlw b'00001110' ;A/D Port AN0/RA0
    movwf ADCON1

    bsf TRISA,0 ;Canal AN0 como entrada
```

```
    bcf TRISC,6 ;RC6/TX salida, pin de transmisión
    movlw d'12' ;2400 baud rate Xtal=4Mhz
    movwf SPBRG
    bcf TXSTA,BRGH ;Selección de baja velocidad
    bcf TXSTA,SYNC ;Modo asíncrono

    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1

    bsf RCSTA,SPEN ;habilita el puerto serie

    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1

    bcf TXSTA,TX9 ;8 bits de datos a transmitir
    bsf TXSTA,TXEN ;Activa la transmisión serial, TXIF = 1

    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
_adc
    bsf ADCON0,GO ;Start A/D conversion
_espera
    btfsc ADCON0,GO ;ADCON0 es 0? (la conversion esta completa?)
    goto _espera ;No, ir _espera
    movf ADRESH,W ;Si, W=ADRESH
    movwf TXREG ;TXREG = W
    movwf PORTC ;PORTC = W
_esperatx
    btfss PIR1,TXIF ;Espera hasta que transmisión culminó
    goto _esperatx
    goto _adc ;Ir adc

end
```

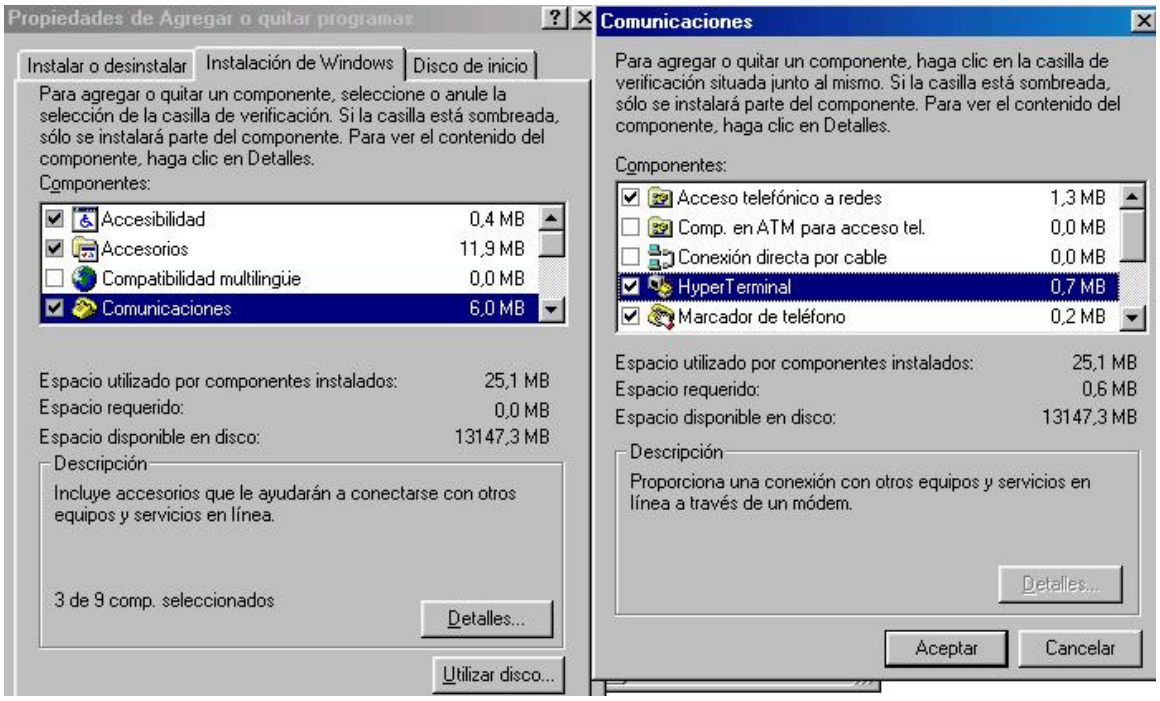
Configurar el Hyper terminal.

Abrir el hyper terminal. Ingresar a:

Inicio->Accesorios->Comunicaciones->Hyper terminal

Si el Hyper terminal no esta instalado

Ingrese a:
Inicio-> Configuración panel de control -> Agregar quitar programas
Elija la pestaña Instalación de Windows. Ahora seleccione Comunicaciones y habilite el Hyper terminal.

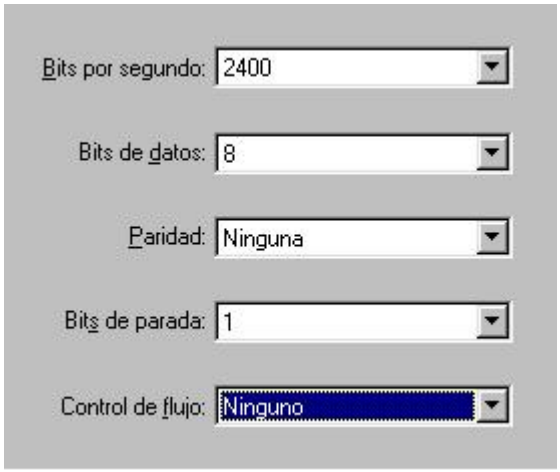


Una vez que abra el hyper terminal observará una ventana como la que se muestra. Elija el ícono para crear una conexión. A continuación le aparecerá una ventana de mensajes: **Necesita instalar un modem antes de poder hacer una conexión Desea hacerlo hora?** Elija el botón "No"	

Ahora elija un icono que identifique la conexión. Asigne como nombre: "**demo**" A continuación aparece una ventana para seleccionar el puerto con el que operará la conexión. Como el ICD esta conectado al COM1 elegiremos en esta ventana el **"COM2"** Presione el botón **Aceptar**	

Ahora proceda a configurar el puerto serial propiamente dicho.
Bits por segundo : 2400
Bits de datos : 8
Paridad : Ninguna
Bit de parada : 1
Control de flujo : ninguno

Presione el botón **Aceptar.**



Si todo ha terminado sin novedad aparecerá una ventana como la que se muestra en este momento podrá usted conectarse con algún dispositivo que tenga conectado al COM2 y que transmita bajos parámetros establecidos (2400:8:0:1:Ninguna)

Trabajo en el MPLAB

Ingrese al MPLAB cree un proyecto
c:\Archivos de programa\MPLAB\m4e1.pjt

Ingrese el código correspondiente al programa y adicione el nodo al proyecto.
Compile. Si no hay errores proceda a grabar el programa en el PIC con ayuda del ICD debugger.
Conecte el demoboard a la PC (COM2) y revise el funcionamiento del programa.

Ejercicio 17
Recibir datos provenientes de la computadora de forma serial y mostrarlos en el PORTC

a) Definiciones previas

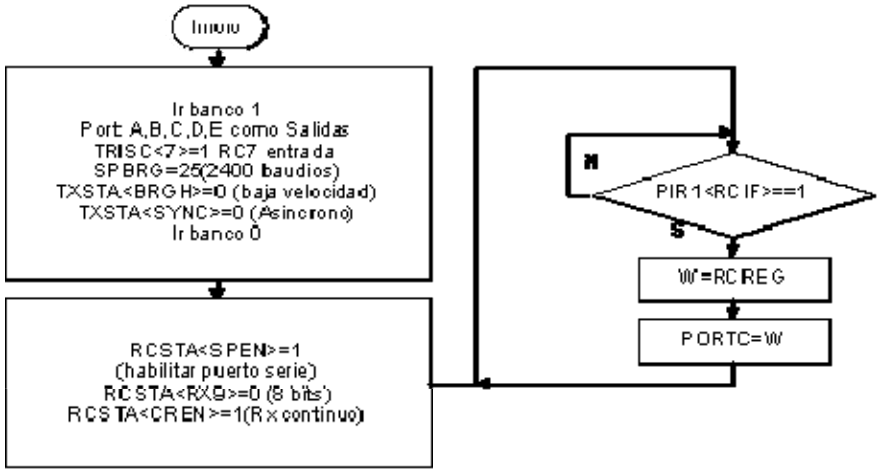
Para el desarrollo del programa es importante conocer la frecuencia del XTAL con el que funcionará el microcontrolador. En nuestro caso es de 4MHz. Ahora a partir de eso definimos la velocidad de transmisión que será de 2400 bits por segundo. Por tanto Ese valor esta tipificado como de baja velocidad y el bit BRGH ha de ser 0. En base a las tablas mostradas en el apartado teórico elegimos 25 para el registro SPBRG. Colocamos el bit SYNC a 0 para una transmisión asíncrona. Este programa solo recepciona datos por tanto configuramos el pin RC7 como entrada. El diagrama de bloques del USART nos musetra que hay 2 bits que controlan el trabajo de recepción; SPEN es una suerte de compuerta que debemos habilitar y el bit CREN también ya que indica que se llevará una recepción continua de datos. En el diagrama de bloque observamos la aparición de otros bits como es el caso del OERR y el FERR que sirve para la detección de errores. También figura el bit TX9 que define si la transmisión se realiza con 8 ó 9 bits de datos, es este caso optaremos por 8 bits de datos. Finalmente existe un bit el RCIF que se pone a 1 cuando un byte ha llegado exitosamente, usaremos esta característica para recoger el valor del USART y lo pasaremos al PORTC. El proceso se realiza en forma continua.

b) Diseño del programa

Algoritmo

1. Ir banco 1.
2. Configurar TRISA=TRISB=TRISC=TRISD=TRISE como salidas
3. Colocar el pin RC7 como entrada
4. Configurar la frecuencia de recepción. Considerando un XTAL de 4MHz tenemos SPBRG=25
5. Configurar para que trabaje a baja velocidad, BRGH=0
6. Configurar el modo asíncrono, SYNC=0
7. Ir banco 0
8. Habilitar la compuerta serie de entrada, SPEN=1
9. Configurar 8 bits de datos, RX9=0
10. Configurar recepción continua, CREN=1
11. Si PIR<RCIF>=0 Ir paso 11
12. Si PIR<RCIF>=1 Ir paso 13
13. PORTC=RCREG
14. Ir paso 11

Diagrama de Flujo



Código del programa

```
    include "p16f877.inc"

org 0x00
    nop
    nop

    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1

    clrf TRISA ;PORTA salida
    clrf TRISB ;PORTB salida
    clrf TRISC ;PORTC salida
    clrf TRISD ;PORTD salida
    clrf TRISE
    bsf TRISC,7 ;RC7/Rx entrada, pin de recepción
    movlw d'23' ;2400 baud rate Xtal=4Mhz
    movwf SPBRG

    bcf TXSTA,BRGH ;Selección de baja velocidad
    bcf TXSTA,SYNC ;Modo asíncrono

    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1

    bsf RCSTA,SPEN ;habilita el puerto serie

    bcf RCSTA,RX9 ;8 Bits de datos
    bsf RCSTA,CREN ;Para Rx Continuo
_espera
    btfss PIR1,RCIF ;Pregunta si el buffer de RX es full
    goto _espera ;No, ir _espera

    movf RCREG,W ;Si, W=RCREG y pone a cero el RCIF
    movwf PORTC ;PORTC = W
    goto _espera

end
```

Trabajo en el MPLAB

Ingrese al MPLAB cree un proyecto

c:\Archivos de programa\MPLAB\m4e2.pjt

Ingrese el código correspondiente al programa y adicione el nodo al proyecto.
Compile. Si no hay errores proceda a grabar el programa en el PIC con ayuda del ICD debugger.

Conecte el demoboard a la PC (COM2) y revise el funcionamiento del programa. No olvide abrir una sesión (Hyper Terminal) para recibir los datos. Cada dato que ingrese por medio del teclado será visualizado en el PORTC del PIC16F877. Pruebe con los números 0,1,2,3,4 y 5 si todo marcha bien observara números binarios consecutivos en el PORTC debido a son consecutivos en la tabla de codigos ASCII.

Ejercicio 18
Diseñe un programa en el PIC 16F877 que reciba datos de la computadora, los muestre por el PORTC y devuelva el mismo dato a la computadora para poder visualizar los datos por el hyper terminal de Windows

a) Definiciones previas

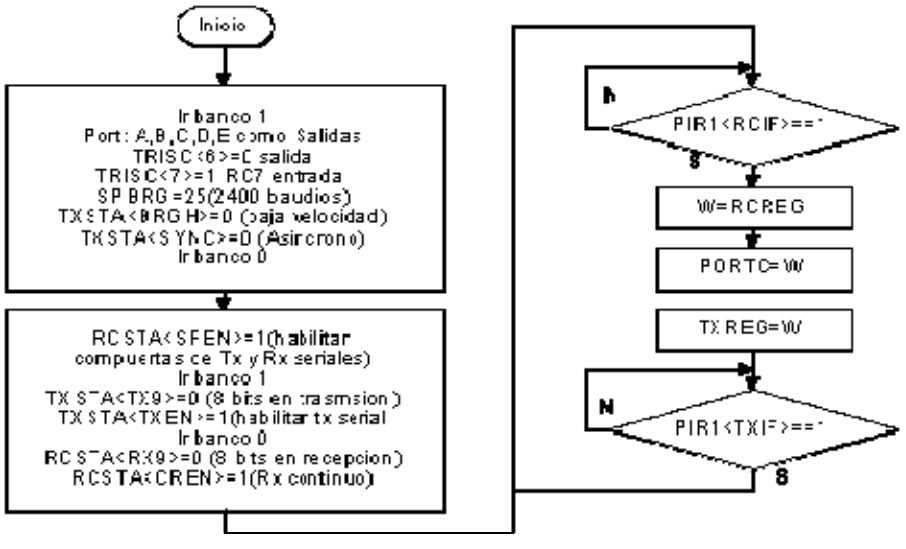
El programa lo que pretende es crear una suerte de "echo" de tal forma que el valor enviado al presionar una tecla sea visualizado en el PORTC del microcontrolador. A continuación ese mismo valor que fue recibido será devuelto a la PC (es como si rebotará), es decir es un "echo". En este programa debemos conjuncionar tanto la transmisión como la recepción serial.

b) Diseño del programa

Algoritmo

1. Ir banco 1.
2. Configurar TRISA=TRISB=TRISC=TRISD=TRISE como salidas
3. Colocar el pin RC6 como salida
4. Colocar el pin RC7 como entrada
5. Configurar la frecuencia de recepción. Considerando un XTAL de 4MHz tenemos SPBRG=25
6. Configurar para que trabaje a baja velocidad, BRGH=0
7. Configurar el modo asíncrono, SYNC=0
8. Ir banco 0
9. Habilitar la compuerta serie de entrada y la compuerta serie de salida, SPEN=1
10. Ir banco 1
11. Configurar 8 bits de datos en transmisión , TX9=0
12. Habilitar transmisión serie, TXEN=1
13. Ir banco 0
14. Configurar 8 bits de datos en recepción , RX9=0
15. Configurar recepción continua, CREN=1
16. Si PIR<RCIF>=0 Ir paso 11
17. Si PIR<RCIF>=1 Ir paso 13
18. PORTC=RCREG
19. TXREG=RCREG
20. Si PIR<TXIF>=0 Ir paso 20
21. Si PIR<TXIF>=1 Ir paso 16

Diagrama de Flujo



Código del programa

```
    include "p16f877.inc"
org 0x00
    nop
    nop

    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1
    clrf TRISA ;PORTA salida
    clrf TRISB ;PORTB salida
    clrf TRISC ;PORTC salida
    clrf TRISD ;PORTD salida
    clrf TRISE
    bcf TRISC,6 ;RC6/TX salida, pin de transmisión
    bsf TRISC,7 ;RC7/Rx entrada, pin de recepción
    movlw d'25' ;2400 baud rate Xtal=4Mhz
    movwf SPBRG
    bcf TXSTA,BRGH ;Selección de baja velocidad
    bcf TXSTA,SYNC ;Modo asíncrono
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1

    bsf RCSTA,SPEN ;habilita el puerto serie

    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1

    bcf TXSTA,TX9 ;8 bits de datos a transmitir
    bsf TXSTA,TXEN ;Activa la transmisión serial, TXIF = 1

    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1

    bcf RCSTA,RX9 ;8 Bits de datos
    bsf RCSTA,CREN ;Para Rx Continuo
_esperarx
    btfss PIR1,RCIF ;Pregunta si el buffer de RX es full
    goto _esperarx ;No, ir _espera

    movf RCREG,W ;Si, W=RCREG y pone a cero el RCIF
    movwf PORTC ;PORTC = W
    movwf TXREG ;Transmitir dato

_esperatx
    btfss PIR1,TXIF ;Espera hasta que transmisión culminó
    goto _esperatx ;No, ir _esperatx

    goto _esperarx ;Si, ir _esperarx
    end
```

Trabajo en el MPLAB

Ingrese al MPLAB cree un proyecto
c:\Archivos de programa\MPLAB\m4l4.pjt

Conecte el demoboard a la PC (COM2) y revise el funcionamiento del programa con ayuda del Hyper terminal. Si todo marcha bien notara como el carácter que digita aparece en el PORTC y ademas es puesto en la ventana del Hyper terminal.

Módulo 5: Manejo de interrupciones

Interrupciones

La familia Pic16F87x tiene 13 fuentes de interrupciones los de 28 pines y 14 los de 40 pines. Al aceptarse una interrupción se salva el valor del PC (contador de porgrama) en la pila y se carga aquel con el valor 0004h, que es el Vector de Interrupcion. La mayoría de los recursos o periféricos de que disponen los Pic16F87x son capaces de ocacionar una interrupción, si se programan adecuadamente los bits de los registros que pasamos a describir a continuación.

1. Desbordamiento del TMR0
2. Activación de la patita de interrupción RB0/INT
3. Cambio de estado de una de las cuatro patitas de mas peso del puerto B
4. Finalización de la escritura de un byte en la EEPROM

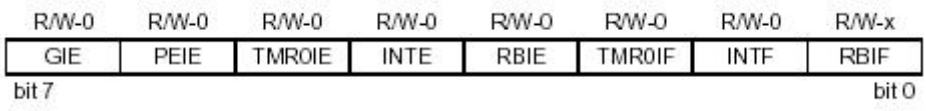
Hasta aqui se tienen las mismas causas de interrupción del Pic16F84

5. Desbordamiento del Timer1
6. Desbordamiento del Timer2
7. Captura o comparación en el módulo CCP1
8. Captura o comparación en el módulo CCP2
9. Transferencia en la puerta serie Síncrona
10. Colisión de bus en la puerta serie Síncrona
11. Fin de la transmisión en el USART
12. Fin de la recepción en el USART
13. Fin de la conversión en el conversor A/D
14. Transferencia en la puerta paralela esclava (Esta causa de interrupción no esta disponible en los Pic16F87x de 28 patitas)

Registro de Control de Interrupciones (INTCON)

Es un registro que podemos leer o escribir y lo lo encontramos en cualquiera de los cuatro bancos, ocupando las direcciones 0x0Bh, 0x8Bh, 0x10Bh, 0x18Bh, respectivamente. Tiene la mision de controlar las interrupciones provocadas por el TMR0, cambio de estado en las cuatro líneas de mas peso de la puerta B y activación de la patita RB0/INT. El bit PEIE actua como una segunda llave parcial de permiso o prohibición de las causas de interrupción que nos estan contenidas en INTCON y que las provocan los restantes periféricos del microcontrolador. GIE es el bit de permiso global de todas las interrupciones.

INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)



bit 7 **GIE:** Bit de permiso global de interrupciones
1 = Enables all unmasked interrupts

0 = Disables all interrupts
bit 6 **PEIE**: Bit de permiso de los perifericos que no se controlan con INTCON
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts
bit 5 **TMR0IE**: Bit de permiso de interrución del TMR0
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt
bit 4 **INTE** : Bit de permiso de la interrución externa por RB0/INT
1 = Enables the RB0/INT external interrupt
0 = Disables the RB0/INT external interrupt
bit 3 **RBIE**: Bit de permiso de interrución por cambio en RB4-RB7
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt
bit 2 **TMR0IF**: Señalizador de desbordamiento en TMR0
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow
bit 1 **INTF**: Señalizador de activación de la patita RB0/INT
1 = The RB0/INT external interrupt occurred (must be cleared in software)
0 = The RB0/INT external interrupt did not occur
bit 0 **RBIF**: Señalizador de cambio en RB4 - RB7
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

Registro de permiso de interrupciones 1 (PIE1)

Contiene los bits que permiten (1) o prohiben (0) las interrupciones provocadas por los periféricos internos del microcontrolador y que no estaban contempladas en INTCON.

Ocupa la dirección 8Ch y para que cumplan su función los bits de PIE1 es necesario que PEIE=1 en INTCON <6>. El bit PSPIE solo es valido en los modelos de 40 pines, manteniéndose a 0 en los que tienen 28 pines.

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

Note 1: PSPIE is reserved on PIC16F873A/876A devices; always maintain this bit clear.

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

Registro de permiso de interrupciones 2 (PIE2)

Contiene los bits de permiso de interrupción de las tres causas que no figuran en PIE1. La de fin de escritura en la EEPROM, colisión de bus en el modo SSP y producción de una captura o una comparación en el módulo CCP2.

PIE2 REGISTER (ADDRESS 8Dh)

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
—	CMIE	—	EEIE	BCLIE	—	—	CCP2IE
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'
bit 6 **CMIE**: Comparator Interrupt Enable bit
1 = Enables the Comparator interrupt
0 = Disable the Comparator interrupt
bit 5 **Unimplemented:** Read as '0'
bit 4 **EEIE**: EEPROM Write Operation Interrupt Enable bit
1 = Enable EEPROM write interrupt
0 = Disable EEPROM write interrupt
bit 3 **BCLIE**: Bus Collision Interrupt Enable bit
1 = Enable bus collision interrupt
0 = Disable bus collision interrupt
bit 2-1 **Unimplemented:** Read as '0'
bit 0 **CCP2IE** : CCP2 Interrupt Enable bit
1 = Enables the CCP2 interrupt
0 = Disables the CCP2 interrupt

Registros de los señalizadores de interrupciones 1 y 2 (PIR1 y PIR2)

En correspondencia con los bits de permiso/prohibición de las causas de interrupción recogidas en los registros PIE1 y PIE2, existen otros dos registros, PIR1 y PIR2, cuyos bits actuan de señalizadores del momento en el que se origina la causa que provoca la interrupción, independientemente de si está permitida o prohibida. Ocupan las direcciones 0Ch y 0Dh, respectivamente.

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:** Parallel Slave Port Read/Write Interrupt Flag bit(1)
1 = A read or a write operation has taken place (must be cleared in software)
0 = No read or write has occurred
Note 1: PSPIF is reserved on PIC16F873A/876A devices; always maintain this bit clear.

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 bit
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

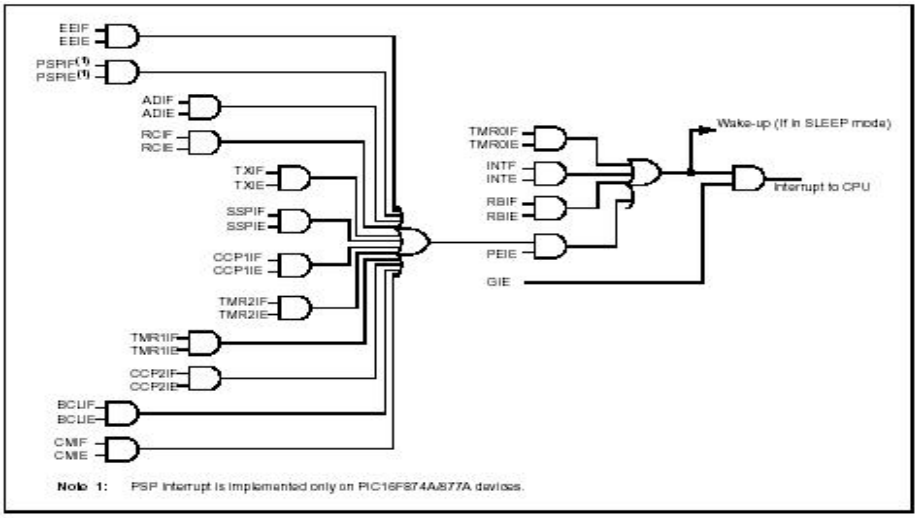
PIR2 REGISTER (ADDRESS 0Dh)

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
—	CMIF	—	EEIF	BCLIF	—	—	CCP2IF
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'
bit 6 **CMIF**: Comparator Interrupt Flag bit
1 = The Comparator input has changed (must be cleared in software)
0 = The Comparator input has not changed
bit 5 **Unimplemented:** Read as '0'
bit 4 **EEIF**: EEPROM Write Operation Interrupt Flag bit
1 = The write operation completed (must be cleared in software)
0 = The write operation is not complete or has not been started
bit 3 **BCLIF**: Bus Collision Interrupt Flag bit
1 = A bus collision has occurred in the SSP, when configured for I^2C Master mode
0 = No bus collision has occurred
bit 2-1 **Unimplemented:** Read as '0'
bit 0 **CCP2IF**: CCP2 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

Lógica de Interrupciones:



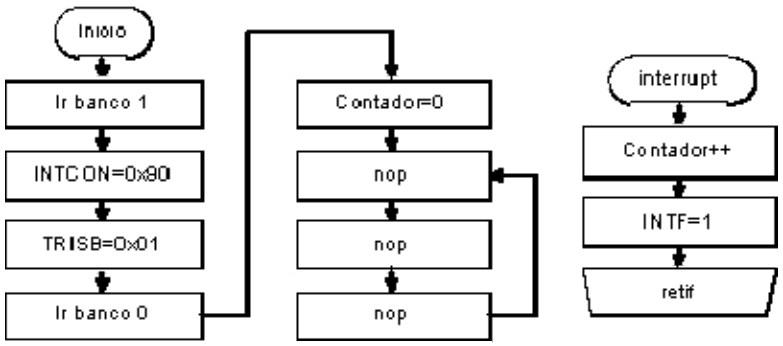
Ejercicios

Ejercicio 20
Diseñar y simular un programa que trabaja la interrupción RB0/INT para ir acumulando los flancos de subida que vayan ingresando por el pin RB0. En este primer ejemplo el bucle principal del programa no "tareas".

a) Diseño del programa MPLAB

Condiciones:
- Debe existir una variable "contador" que acumule los pulsos que llegan al RB0/INT. Al inicio del programa la variable ha de ser colocada a 0.
- Los pulsos son detectados por el flanco de subida.
- En el programa principal debe existir un bucle que ejecute instrucciones de no operación en forma infinita.

Diagrama de flujo El diagrama de flujo comienza con la configuración del microcontrolador, ingreso al banco 1, allí se configuran los registros INTCON, TRISB y el bit INTEDG del OPTION a fin de detectar pulsos de entrada en RB0/INT por flanco de subida. La rutina de interrupción incrementa la variable CONTADOR en una unidad, y además resetea el flag INTF.



Los valores para los bits del registro INTCON son:

GIE EEIE T0IE INTE RBIE T01F INTF RBIF
1 0 0 1 0 0 0 0 = 0x 90

Código del programa

```
    list p=16f877
    include "p16f877.inc"

    CONTADOR EQU 0x20 ;POSMEM en la que usaremos una variable

    ORG 0 ;Vector de RESET
    nop
    nop
    goto INICIO ;Alta a la direccion donde se inicia el programa
    ORG 4 ;Vector de INTERRUPCION
    incf CONTADOR,1 ;Contador++
    bcf INTCON,INTF ;Resetea el flag de la interrupcion RB0/INT
    retfie ;Fin rutina de interupcion
INICIO
    bcf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1

    clrf TRISA ;PORTA salida
    clrf TRISB ;PORTB salida
    clrf TRISC ;PORTC salida
    clrf TRISD ;PORTD salida
    clrf TRISE ;PORTD salida
    bsf TRISB,0 ;RB0 como entrada
    movlw 0x90
    movwf INTCON ;Configura INTCON
    bsf STATUS,INTEDG ;Detecta flancos de subida que llegan a RB0/INT

    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    clrf CONTADOR ;Contador=0

BUCLE
    nop ;No operacion
```

```
    nop
    goto BUCLE ;Ir a BUCLE

END
```

b) Simulación MPLAB

Ingrese al MPLAB, cree un proyecto. Luego ingrese el código que se mencionado anteriormente, guárdelo en un archivo . Compile el programa para verificar que este libre de errores.

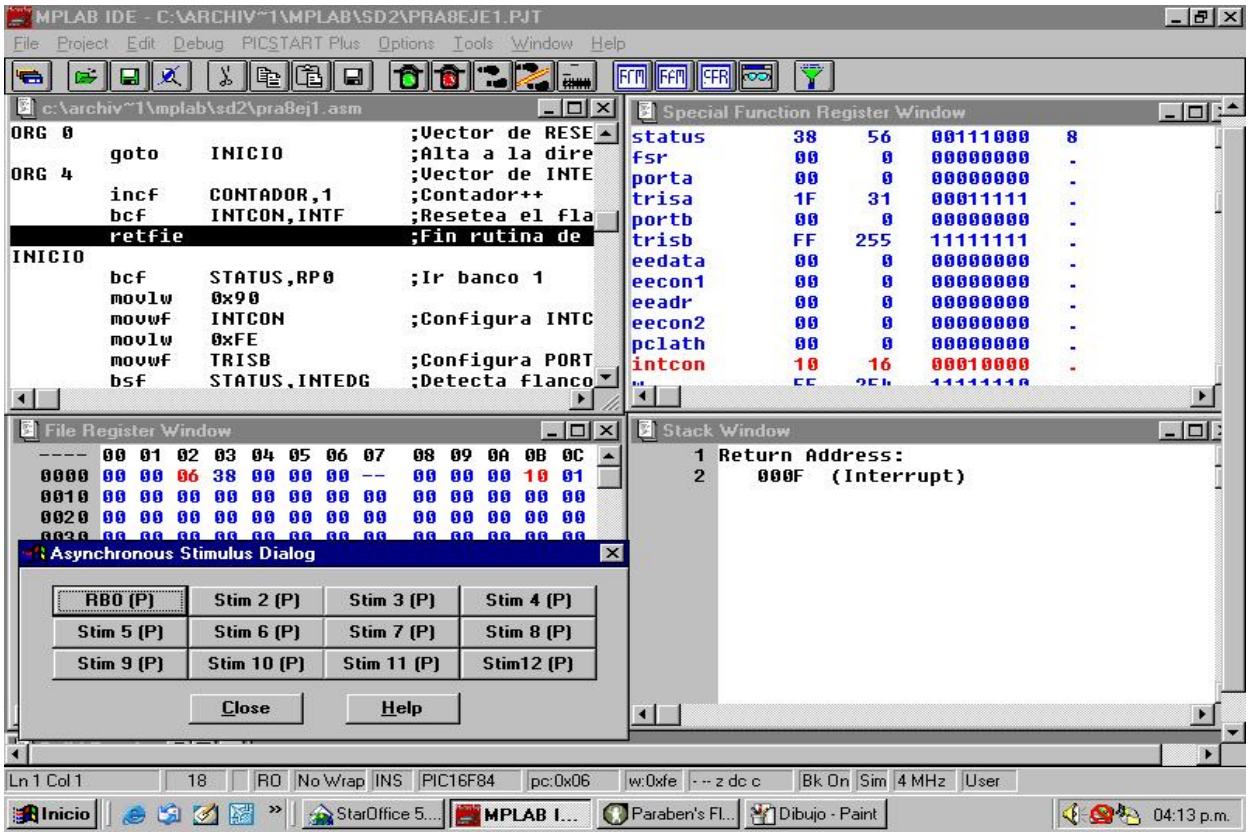
- Asigne un botón al RB0/INT en el ASYNCHRONUS STIMULUS.
  Para hacer esta tarea:
- Ingresar al menú -> DEBUG -> STIMULUS SIMULATOR -> ASYNCHRONUS STIMULUS
- Para asociar Stim 1(P) a RBO haga clic derecho sobre el botón Stim 1 (P). Elija la opción Assing PIN. Seleccione RB0
- Asegurece que el tipo de estimulo a ser aplicado sea tipo PULSE.

Abra las ventanas siguientes:

- Stack Window
- Special Funcion Register
- File Register

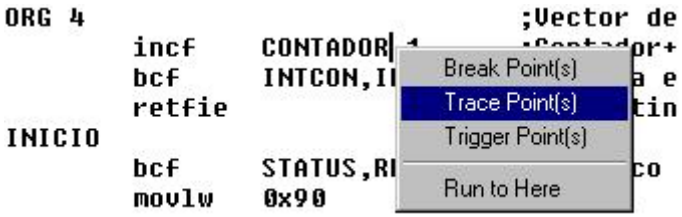
omode las en la ventana a fin que pueda tener una visión de todas ellas a la vez.

- Proceda a simular el programa ingrese al modo STEP (F7, avance paso a paso). Avance hasta ingresar a la etiquta “BUCLE”.
- Con ayuda del mouse simule un pulso en RB0/INT (haga un click en: RBO (P), ventana ASYNCHRONUS STIMULUS). Observara como es que simulador salta a la rutina de interrupción e incrementa el valor de la cuenta CONTADOR (registro 0x0C).
- Observe el lo que sucede en la ventana STACK WINDOWS.



TRACE POINT

- Resetee el programa (DEBUG->RUN->RESET)
- Coloque el mouse sobre la línea "bcf INTCON,INTF" , click izquierdo, BREAK POINT
- Cree un punto de revisión (TRACE POINT)
- Coloque el mouse sobre la instrucción "**incf CONTADOR,1**", click izquierdo, TRACE POINT



- Abra la ventana TRACE WINDOWS (menú Windows -> Trace Memory)
- Ahora corra el programa (click sobre el semáforo verde). Ingrese 5 pulsos en el pin RB0/INT del microcontrolador.
- Detenga el programa (click en el SEMAFORO ROJO).
- Observe que aparece en la ventana TRACE MEMORY.

NOTA.- Los TRACE POINTS siempre se muestran en COLOR VERDE. Es posible usar el TRACE POINT en combinación con el BREAK POINT. Desde luego que solo visualizara los datos acumulados hasta encontrar el BREAK POINT.

OBSERVACIONES.-La ventana TRACE MEMORY reporta los cambios operados sobre la linea que ha sido marcada con el TRACE POINTS. Como nuestro objetivo era examinar el valor de CONTADOR pusimos el TRACE POINTS sobre la instrucción "**incf CONTADOR,1".**

CUESTIONARIO: Use el simulador para responder las siguientes preguntas:

a) ¿Que es lo que pasa si elimina y/o comenta la línea?

bcf INTCON,INTF En la rutina de interrupción.¿ Porque?

b) ¿Qué es lo que pasaría en la ventana STACK del microcontrolador?

c) ¿Qué es lo que pasa con el bit **GIE** del registro **INTCON** cuando entramos a la rutina de interrupción?
Explique que utilidad tiene esto.

d) Si reemplazamos la instrucción:

retfie por la instrucción return.
¿Funcionaría el programa? ¿Porque?. Sugerencia. Observe el valor de GIE.

e) Reemplace la rutina de interrupción para que quede así:

```
ORG 4
incf CONTADOR,1
bcf INTCON,INTF
bsf INTCON,GIE
goto BUCLE
```

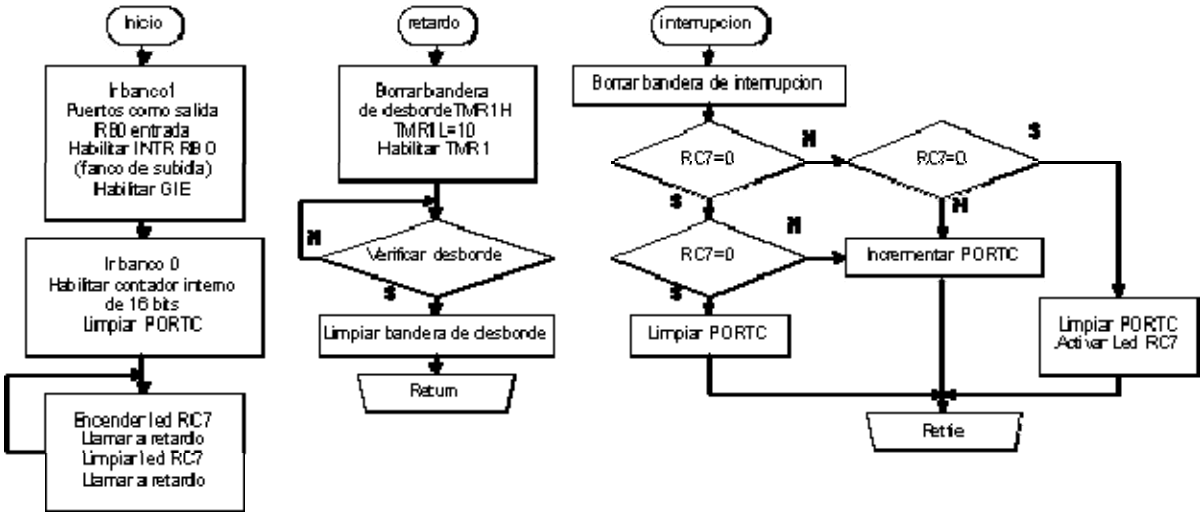
¿Funcionaria el programa? Porque ? Cual es la intención en la ejecución de bsf INTCON, GIE?
¿Qué se puede ver en la ventana de STACK WINDOWS
¿Qué pasa cuando se presenta el overflow del STACK?

Ejercicio 21

Configurar el TMR1 para generar espacios de tiempo de 500ms. Cada vez que se cumpla el tiempo invertir el valor del RC7 (similar a un led intermitente). Emplear la interrupción RB0 para incrementar el valor del PORTC cada vez que llegue una petición.

Definiciones Previas

El PIC16F877 cuenta con 14 causas que pueden originar una interrupción; el uso de interrupciones es importante porque permite realizar varias tareas a la vez. Cuando se produce una interrupción se salva el valor del contador de programa y se carga con el valor 0x04 donde debe estar la rutina de interrupción. Una vez concluída la rutina de interrupción restablece el valor del contador del programa.



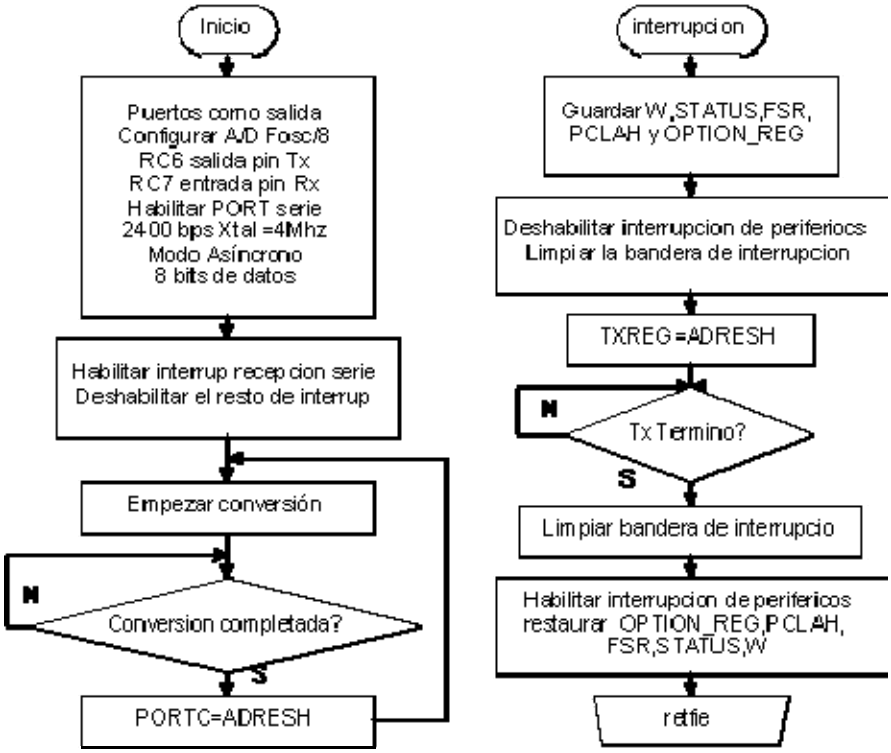

```
     list p=16f877 ;Comando que indica el Pic usado
    include "p16f877.inc" ;Etiquetas genéricas para el Pic16F877
org 0x00 ;Inicio del programa en la posición cero de memoria
    nop ;Libre (uso del debugger)
    goto _inicio ;Ir _inicio
org 0x04 ;Vector de interrupcion
    goto _interrupción ;Ir rutina de interrupcion

_inicio
    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1

    clrf TRISA ;PORTA salida
    clrf TRISB ;PORTB salida
    clrf TRISC ;PORTC salida
    clrf TRISD ;PORTD salida
    clrf TRISE ;PORTE salida
    bsf TRISB,0 ;RB0 entrada
    clrf INTCON ;Deshabilitar interrupciones
    clrf PIE1
    clrf PIE2
    bsf INTCON,INTE ;Habiltar la interrupción de RB0
    bcf OPTION_REG,INTEDG ;Flanco de subida en RBO
    bsf INTCON,GIE ;Habilitador general de interrupciones

    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1

    movlw b'00110000' ;Contador interno de 16 bits
    movwf T1CON

    clrf PORTC ;PORTC=0
_bucle
    bsf PORTC,7 ;RC7 = 1
    call _retardo ;Retardo de 500 ms
    bcf PORTC,7 ;RC7 = 0
    call _retardo ;Retardo de 500 ms
    goto _bucle ;Ir bucle

_retardo
    bcf PIR1,TMR1IF ;Borrar la bandera de desborde
```

```
    clrf TMR1H ;TMR1H = 0
    movlw 0xa ;TMR1L = 10
    movwf TMR1L
    bsf T1CON,TMR1ON ;Habilita el TMR1
_espera
    btfss PIR1,TMR1IF ;Verificar el desborde
    goto _espera ;Si no ir _espera
    bcf T1CON,TMR1ON ;Si desborda: limpiar bandera de desborde
    return ;Retorno

_interrupción ;Rutina de interrupcion
    bcf INTCON,INTF ;Borra flag de interrupcion RBO
    btfsc PORTC,7 ;RC7 = 0
    goto _compara ;No, ir _compara
    movlw 0x7f ;PORTC = 0x7f?
    subwf PORTC,W
    btfss STATUS,Z
    goto _salida ;No, ir _salida
    clrf PORTC ;Si, PORTC = 0
    retfie ;Fin interrupcion

_compara
    movlw 0Xff ;PORTC = 0xff?
    subwf PORTC,W
    btfss STATUS,Z
    goto _salida ;No, ir _salida
    clrf PORTC ;Si, PORTC = 0
    bsf PORTC,7 ;RC7 = 1
    retfie ;Fin interrupcion

_salida
    incf PORTC,F ;PORTC = PORTC + 1
    retfie ;Fin interrupcion
    END
```

Ejercicio 22
Configurar el canal AN0 del ADC para mostrar en forma permanente el valor de conversión en el puerto C. Cada vez que llegue una interrupción provocada por el hiperterminal el microcontrolador enviará a la PC el valor actual del ADC.

Durante la ejecucion de las interupciones se suele cambiar algunos valores por efecto de la ejecución de una instruccion (por ejemplo el STATUS se modifica cuando ejecutamos una instrucción). El contexto o los registros importantes deben guardarse al inicio de la rutina de interrupción y deben restituirse antes de salir de la misma. Por eso es que el fabricante sugiere guardar al menos los siguientes registros:

```
MOVWF   W_TEMP          ;Copy W to TEMP register
SWAPF   STATUS,W        ;Swap status to be saved into W
CLRF    STATUS          ;bank 0, regardless of current bank, Clears IRP,RP1,RP0
MOVWF   STATUS_TEMP     ;Save status to bank zero STATUS_TEMP register
MOVF    PCLATH, W       ;Only required if using pages 1, 2 and/or 3
MOVWF   PCLATH_TEMP     ;Save PCLATH into W
CLRF    PCLATH          ;Page zero, regardless of current page
:
:(ISR)                  ;(Insert user code here)
:
MOVF    PCLATH_TEMP, W  ;Restore PCLATH
MOVWF   PCLATH          ;Move W into PCLATH
SWAPF   STATUS_TEMP,W   ;Swap STATUS_TEMP register into W
                        ;(sets bank to original state)
MOVWF   STATUS          ;Move W into STATUS register
SWAPF   W_TEMP,F        ;Swap W_TEMP
SWAPF   W_TEMP,W        ;Swap W_TEMP into W
```

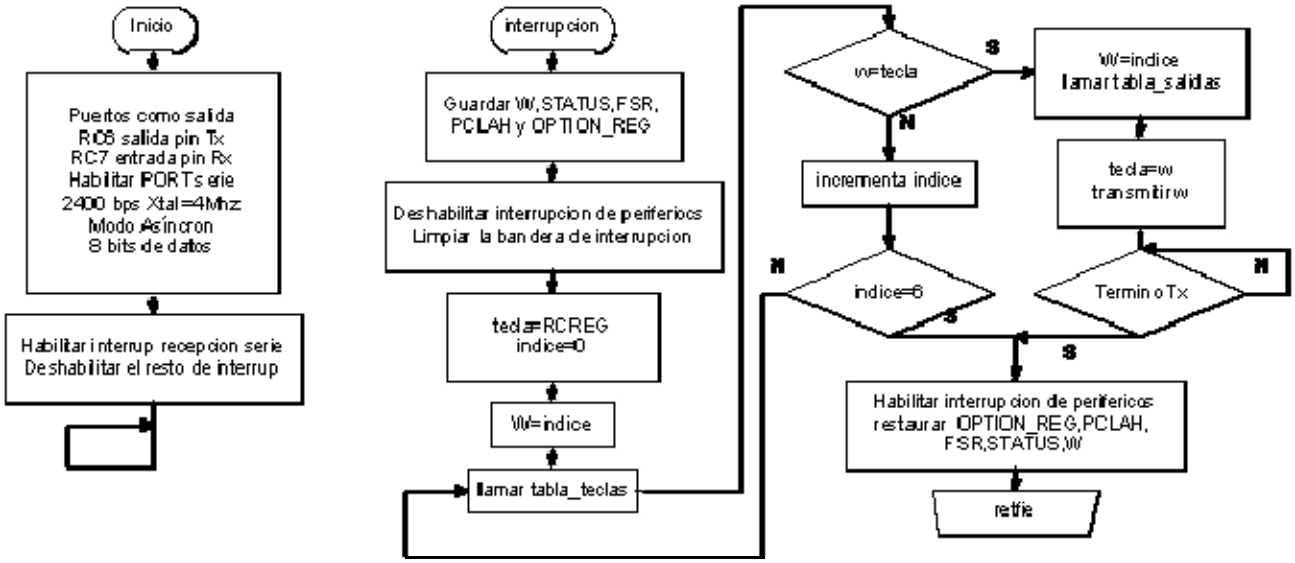



```
    list p=16f877 ;Comando que indica el Pic usado
    include "p16f877.inc" ;Etiquetas genéricas para el Pic16F877

    _int_save_W EQU 0x20 ;Var para guardar W
    _int_save_STATUS EQU 0x21 ;Var para guardar STATUS
    _int_save_FSR EQU 0x22 ;Var para guardar FSR
    _int_save_PCLATH EQU 0x23 ;Var para guardar PCLATH
    _int_save_OPTION_REG EQU 0x24 ;Var para guardar OPTION_REG

org 0x00 ;Inicio del programa en la posición cero de memoria
    nop ;Libre (uso del debugger)
    goto _inicio ;Ir _inicio

org 0x04 ;Vector de interrupcion
```

```
    movwf _int_save_W ;Guardar W
    swapf STATUS,W ;Guardar STATUS
    movwf _int_save_STATUS
    swapf FSR,W ;Guardar FSR
    movwf _int_save_FSR
    swapf PCLATH,W ;Guardar PCLATH (Pagina de programa)
    movwf _int_save_PCLATH
    swapf OPTION_REG,W ;Guardar OPTION_REG (Bancos)
    movwf _int_save_OPTION_REG

    goto _interrupción ;Ir rutina de interrupcion

_inicio
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1

    movlw b'01000001' ;A/D conversion Fosc/8
    movwf ADCON0

    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1

    clrf TRISA ;PORTA salida
    clrf TRISB ;PORTB salida
    clrf TRISC ;PORTC salida
    clrf TRISD ;PORTD salida
    clrf TRISE ;PORTE salida
    bcf TRISC,6 ;RC6/TX salida, pin de transmisión
    bsf TRISC,7 ;RC7/Rx entrada, pin de recepción
    movlw d'25' ;2400 baud rate Xtal=4Mhz
    movwf SPBRG

    bcf TXSTA,BRGH ;Selección de baja velocidad
    bcf TXSTA,SYNC ;Modo asíncrono

    movlw b'00001110' ;A/D Port AN0/RA0
    movwf ADCON1

    bsf TRISA,0 ;RA0 linea de entrada para el ADC

    clrf INTCON ;Deshabilitar interrupciones
    clrf PIE1
    clrf PIE2

    bsf PIE1,RCIE ;Habilita INTR Recepcion Serial
    bsf INTCON,PEIE ;Habilita INTR perifericos
    bsf INTCON,GIE ;Habilitador general de interrupciones

    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1

    bsf RCSTA,SPEN ;habilita el puerto serie

    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1

    bcf TXSTA,TX9 ;8 bits de datos a transmitir
    bsf TXSTA,TXEN ;Activa la transmisión serial, TXIF = 1

    bcf STATUS,RP0 ;Ir banco 0
```

```
    bcf STATUS,RP1
    bcf RCSTA,RX9 ;8 Bits de datos
    bsf RCSTA,CREN ;Para Rx Continuo

    clrf PORTC ;PORTC=0

_bucle
    bsf ADCON0,GO ;Start A/D conversion
_espera
    btfsc ADCON0,GO ;ADCON0 es 0? (la conversion esta completa?)
    goto _espera ;No, ir _espera

    movf ADRESH,W ;Si, W=ADRESH
    movwf PORTC ;Muestra el resultado en PORTC

    goto _bucle ;Ir bucle

_interrupción ;Rutina de interrupcion

    bcf INTCON,PEIE ;Deshabilita INTR perifericos
    movfw ADRESH ;W= ADC
    movwf TXREG ;Transmitir dato, TXREG = W

_esperatx
    btfss PIR1,TXIF ;Espera hasta que transmisión culminó
    goto _esperatx ;No, ir _esperatx

    movf RCREG,W ;Limpia el flag de interrupcion RCIF =0
    bsf INTCON,PEIE ;Habilita INTR perifericos

    swapf _int_save_OPTION_REG,W ;Restaurar valor de OPTIO_REG
    movwf OPTION_REG
    swapf _int_save_PCLATH,W ;Restaurar valor de PCLATH
    movwf PCLATH
    swapf _int_save_FSR,W ;Restaurar valor de FSR
    movwf FSR
    swapf _int_save_STATUS,W ;Restaurar valor de STATUS
    movwf STATUS
    swapf _int_save_W,W
    retfie
END
```

Diseñe un programa en el Pic16F877 que funcione con la interrupción del USART. Si desde el hiperterminal se presiona la tecla ‘A’ RC0 = 1, ‘a’ RC0 = 0, ‘B’ RC1 = 1, ‘b’ RC1 = 0, ‘C’ RC2 = 1, ‘c’ RC2 = 0.




```
 list p=16f877 ;Comando que indica el Pic usado
     include "p16f877.inc" ;Etiquetas genéricas para el Pic16F877
      _int_save_W EQU 0x20 ;Var para guardar W
    _int_save_STATUS EQU 0x21 ;Var para guardar STATUS
     _int_save_FSR EQU 0x22 ;Var para guardar FSR
     _int_save_PCLATH EQU 0x23 ;Var para guardar PCLATH
     _int_save_OPTION_REG EQU 0x24 ;Var para guardar OPTION_REG
    tecla EQU 0x25
    indice EQU 0x26

org 0x00 ;Inicio del programa
    nop ;Libre (uso del debugger)
    goto _inicio ;Ir _inicio

org 0x04 ;Vector de interrupcion
    movwf _int_save_W ;Guardar W
    swapf STATUS,W ;Guardar STATUS
    movwf _int_save_STATUS
    swapf FSR,W ;Guardar FSR
    movwf _int_save_FSR
    swapf PCLATH,W ;Guardar PCLATH (Pagina de programa)
    movwf _int_save_PCLATH
    swapf OPTION_REG,W ;Guardar OPTION_REG (Bancos)
    movwf _int_save_OPTION_REG
    goto _interrupción ;Ir rutina de interrupcion

_inicio
    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP   1
    clrf TRISA ;PORTA salida
    clrf TRISB ;PORTB salida
    clrf TRISC ;PORTC salida
    clrf TRISD ;PORTD salida
    clrf TRISE ;PORTE salida

    bcf TRISC,6 ;RC6/TX salida, pin de transmisión
    bsf TRISC,7 ;RC7/Rx entrada, pin de recepción
    movlw d'25' ;2400 baud rate Xtal=4Mhz
    movwf SPBRG
    bcf TXSTA,BRGH ;Selección de baja velocidad
```

```
    bcf TXSTA,SYNC ;Modo asíncrono

    clrf INTCON ;Deshabilitar interrupciones
    clrf PIE1
    clrf PIE2

  bsf PIE1,RCIE ;Habilita INTR Recepcion Serial
    bsf INTCON,PEIE ;Habilita INTR perifericos
    bsf INTCON,GIE ;Habilitador general
                              ;de interrupciones

    t-size: 8pt">bcf STATUS,RP0 ;Ir banco 0
  bcf STATUS,RP1
    bsf RCSTA,SPEN ;habilita el puerto serie

    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1

    bcf TXSTA,TX9 ;8 bits de datos a transmitir
    bsf TXSTA,TXEN ;Activa la transmisión serial TXIF=1

    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    bcf RCSTA,RX9 ;8 Bits de datos
    bsf RCSTA,CREN ;Para Rx Continuo

    clrf PORTC ;PORTC=0

_bucle goto _bucle ;Ir bucle

_interrupción ;Rutina de interrupcion
    bcf INTCON,PEIE ;Deshabilita INTR perifericos
    movf RCREG,W ;Limpia el flag de interrup RCIF =0
    movwf tecla
    clrf indice
 _busca_tecla
    movfw indice
    call _teclas
    subwf tecla,W
    btfsc STATUS,Z
    goto _sigue
    incf indice,F
    movlw D'6'
    subwf indice,W
    btfsc STATUS,Z
    goto _retorno
    goto _busca_tecla
_sigue
    movf indice,W
    call _salidas
    movfw tecla
    movwf TXREG ;Transmitir dato, TXREG = W

_esperatx
    btfss PIR1,TXIF ;Espera hasta que transmisión culminó
    goto _esperatx ;No, ir _esperatx

_retorno
    bsf INTCON,PEIE ;Habilita INTR perifericos
    swapf _int_save_OPTION_REG,W ;Restaurar valor de OPTIO_REG
    movwf OPTION_REG
```

```
    swapf _int_save_PCLATH,W ;Restaurar valor de PCLATH
    movwf PCLATH
    swapf _int_save_FSR,W ;Restaurar valor de FSR
    movwf FSR
    swapf _int_save_STATUS,W ;Restaurar valor de STATUS
    movwf STATUS
    swapf _int_save_W,W
    retfie

_teclas
    addwf PCL,F ;pcl + W >>>> W
                         ;El PCL se incrementa con el valor
                         ;de W proporcionando
                         ;un salto
    retlw A'A' ;retorna con valores para PORTC
    retlw A'B'
    retlw A'C'
    retlw A'a'
    retlw A'b'
    retlw A'c'

_salidas
    addwf PCL,F ;pcl + W >>>> W
                        ;El PCL se incrementa con el valor
                        ;de W proporcionando
                        ;un salto
    goto _on_RC0
    goto _on_RC1
    goto _on_RC2
    goto _off_RC0
    goto _off_RC1
    goto _off_RC2

_on_RC0 bsf PORTC,0
    return
_on_RC1 bsf PORTC,1
    return
_on_RC2 bsf PORTC,2
    return
_off_RC0 bcf PORTC,0
    return
_off_RC1 bcf PORTC,1
    return
_off_RC2 bcf PORTC,2
    return

END
```

Módulo 6: Memoria EEPROM

Memoria de Datos EEPROM y Memoria FLASH de Programa

La memoria EEPROM(llamada de datos) y la memoria FLASH (memoria de programa) pueden ser leídas y escritas durante la ejecución de un programa. Trabajan con la tensión de alimentación del microcontrolador (VDD). Para operar ambos bloques de memoria disponemos de los siguientes registros especiales de función (SFR):

EECON1
EECON2
EEDATA
EEDATH
EEADR
EEADRH

La memoria de datos (EEPROM) tiene una palabra de 8 bits. Cuando accedemos a este bloque de memoria el registro EEDATA contiene los 8 bits de datos y el registro EEADR contiene la dirección que deseamos trabajar. El PIC16F877 tiene 256 posiciones de memoria de datos (EEPROM) numeradas de 0x00 a 0xFF (de 0 a 255). La memoria de datos (EEPROM) es una memoria no volátil que se graba eléctricamente, es útil por que nos permite almacenar valores o parámetros que sean persistentes en la ejecución de la aplicación. Esto es, que no se borren frente a la pérdida de energía o reset.

En la memoria FLASH (programa) podemos leer o escribir palabras. El acceso a la memoria de programa se hace para el cálculo del checksum y la calibración de tablas de almacenamiento. La escritura de un byte o de una palabra borra automáticamente la localización seleccionada y coloca en ella el dato elegido. La escritura de la memoria de programa cesa todas las operaciones hasta que es completada. La memoria de programa no puede ser accedidas durante un ciclo de escritura por tanto ninguna instrucción puede ser ejecutada. Durante la operación de escritura el oscilador continua trabajando los demás módulos. Si se sucede una interrupción esta queda en espera (en una suerte de cola) hasta que el ciclo de escritura de la memoria de programa haya terminado. Terminado el ciclo, ejecuta una instrucción mas (que se cargo antes de que empiece la grabación) y luego salta al vector de interrupción.

Cuando trabajamos la memoria FLASH (programa) los registros EEDATH:EEDATA sirven para almacenar el contenido, recordemos que la palabra de la memoria de programa es de 14 bits. Los registros EEADRH:EEADR sirven para direccionar la posición de memoria (13 bits) dado que las direcciones varían de 0 a 8K (0x000 a 0x3FFF). Los bits superiores que no se emplean se llenan con ceros.

Los valores escritos en la memoria de programa no tiene que ser necesariamente valores correspondientes a instrucciones. Por tanto los 14 bits almacenados pueden servir como parámetros, números seriales, paquetes de 7 bits ASCII , etc. La ejecución de una posición de la memoria de programa que contenga datos resulta en un código NOP.

El registro EEADR
Sirve para direccionar (apuntar) a una posición de las 256 disponibles para la memoria EEPROM o para las 8192 de la memoria de programa (FLASH). El registro EEADRH contiene los valores mas significativos, la EEADR los 8 menos significativos.

Los registros EECON1 y EECON2

El registro EECON1 es el registro que contiene los bits de control del proceso de lectura o escritura en la memoria EEPROM. El registro EECON2 es un registro que se usa en la secuencia de escritura y que físicamente no existe, si se lee se encuentra que esta lleno de ceros.

En el registro EECON1 esta el bit EEPGD (posición 7) que indica si el trabajo a realizar se lleva a efecto sobre la memoria de datos o sobre la memoria de programa. Cuando es colocado a cero las operaciones serán sobre la EEPROM (memoria de datos) y si es colocado a 1 las operaciones son sobre la memoria de programa (FLASH)

Los bits WR y RD del registro EECON1 sirven para controlar y revisar el estado de la operaciones de escritura y lectura respectivamente. Para iniciar cualquier de esas operaciones se coloca a 1-lógico el bit de la operación. Cuando se ha completado un ciclo de lectura o escritura los bits se colocan a 0-lógico nuevamente. Por software solo se puede colocar a 1-lógico.

El bit WREN (write enable) se coloca a 1 para habilitar la operación de escritura esto para evitar escrituras espurías en el EEPROM. Cuando el microcontrolador es energizado el bit WEN permanece a 0-lógico.

En el registro EECON1encontramos el bit WRERR que es una bandera que se activa (coloca a 1-lógico) cuando se ha producido un error en la escritura de la EEPROM que puede deberse a un RESET, la activación de WDT. Siempre es recomendable revisar el bit WRERR para saber si la grabación termino exitosamente si no es así se procede a reescribir el dato. Los valores de EADR y EEDATA permanecen iguales así como el bit EEPGD.

El bit EEIF es una bandera que se coloca a 1-lógico cuando la operación de escritura ha terminado, debe ser colocada a 0 por software. La bandera se puede usar cuando deseamos trabajar interrupciones. A continuación se presenta el detalle del registro EECON1:

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

EECON1 REGISTER (ADDRESS 18Ch)

bit 7 **EEPGD** : Bit selector de memoria Programa(FLASH)/Data (EEPROM)
1 = Acceso a la memoria de programa (FLASH)
0 = Acceso a la memoria de datos (EEPROM)
Este bit no puede ser cambiado una vez que la operación de escritura esta en progeso
bit 6-4 **No implementados:** Se leen como '0'
bit 3 **WRERR:** Bandera de Error de EEPROM
1 = Si la operación de escritura ha terminado premauramente
si se ha producido algun MCLR(RESET) o reset de Watchdog durante la escritura
0 = Completo la operación de escritura
bit 2 **WREN:** Bit habilitador de ciclo de escritura en EEPROM
1 = Permite un cliclo de escritura
0 = Inhabilita la escritura en la EEPROM
bit 1 **WR:** Bit de control de escritura
1 = Inicia un ciclo de escritura. El bit es puesto a 0 por hardware una vez completada la escritura
El bit WR solo puede ser puesto a 1-logico por software.
0 = Ha sido completado un ciclo de escritura en EEPROM
bit 0 **RD:** Bit de control de lectura

1 = Inicializa la lectura de la EEPROM; RD es puesto a 0 por hardware.
El bit RD solo puede ser puesto a 1-logico por software.
0 = No se ha empezado un ciclo de lectura en la EEPROM

Operación de lectura de la memoria de datos EEPROM

La secuencia de pasos para efectuar una secuencia de lectura en la memoria EEPROM es la siguiente:

- Cargar el registro EEADR con la dirección que nos interesa acceder.
- Colocar a 0-lógico el bit EEPGD del registro EECON1 para indicar que vamos a trabajar en la memoria de datos(EEPROM).
- Colocar a 1-lógico el bit RD del registro EECON1.
- En el siguiente ciclo de instrucción podemos obtener el contenido en el registro EEDATA.

El registro EEDATA contendrá el valor que no escribió hasta que se lea el valor o se vuelva a ejecutar la operación de escritura. El siguiente segmento de código muestra un ciclo de lectura:

```
bsf STATUS,RP1 ;Ir banco 2
bcf STATUS,RP0

movlw DATA_EE_ADDR ;EEADR=direccion a leer
movwf EEADR

bsf STATUS, RP0 ;Ir banco 3

bcf EECON1,EEPGD ;Indicar trabjo en la memoria de datos EEPROM
bsf EECON1,RD ;Iniciar el ciclo de lectura

bcf STATUS,RP0 ;Ir banco 2
movf EEDATA,W ;W=EEDATA
```

Operación de escritura en la memoria de datos EEPROM

Al igual que en el caso anterior antes de realizar una operación debemos cargar la dirección de la memoria en el registro EEADR y el dato debe ser cargado en el registro EEDATA. El siguiente segmento de código muestra la secuencia de grabación a ser operada:

```
    bsf STATUS,RP1 ;Ir banco 2
    bcf STATUS,RP0
    movlw DATA_EE_ADR ;EEADR=DATA_EE_ADR
    movwf EEADR

    movlw DATA_EE_DATA ;EEADR=DATA_EE_DATA
    movwf EEDATA

    bsf STATUS,RP0 ;Ir banco 3
    bcf EECON1,EEPGD ;Indicar trabajo en la memoria de datos
    bsf EECON1,WREN ;Habilitar la escritura

    bcf INTCON,GIE ;Deshabilitar las interupciones (si es que estan
habilitadas)

    movlw 0x55 ;EECON2=0x55
    movwf EECON2
    movlw 0xAA ;EECON2=0xAA
```

```
    movwf EECON2
    bsf EECON1,WR ;Empieza ciclo de grabacion
    bsf INTCON,GIE ;Habilitar interrupciones(si es que estuvieron
habilitadas)
    sleep ;Esperar a que termine el ciclo de escritura
    bcf EECON1,WREN ;Deshabilitar ciclos de escritura
```

El fabricante indica que no se llevara a cabo el ciclo de grabación si en forma previa no se escribe 0x55 y 0xAA en el registro EECON2, después se puede habilitar el ciclo de escritura con el bit WR. Esta secuencia se debe realizar cada vez que almacenamos un dato en la EEPROM. Este mecanismo tiene por objetivo evitar la escritura accidental de datos indeseados. El bit WREN debe ser puesto a 0-lógico todo el tiempo excepto cuando actualicemos datos en la EEPROM. El bit WREN no se coloca a 0 por hardware.

Luego que se inicia la escritura si cambiamos el valor de WREN no detiene la grabación. El bit WR solo debe ser colocado a 1-lógico después de que WREN ha sido puesto a 1-lógico. El bit WR y el bit WREN no deben ser colocados a 1-lógico en la misma instrucción.

Cuando se completa un ciclo de escritura, el bit WR es colocado a 0 por hardware y el bit EEIF se coloca a 1-lógico. El bit EEIF debe ser nuevamente colocado a 0-lógico por software.

Protección contra escrituras espurias

Hay condiciones en las cuales se desea proteger el dispositivo contra posibles grabaciones indeseadas para ello existen mecanismos. Cuando se energiza el microcontrolador (Power-Up) el bit WREN es puesto a 0-lógico, ademas hay un temporizador Power-Up timer que demora 72 us antes de que se efectúe cualquier instrucción en el microcontrolador. El bit PWRTE de la palabra de configuración debe ser puesto a 1-lógico al momento de grabar el PIC para que el temporizador Power-Up funcione. Finalmente existe la secuencia de grabación y el bit WREN que controla el ciclo de escritura.

Ejercicios

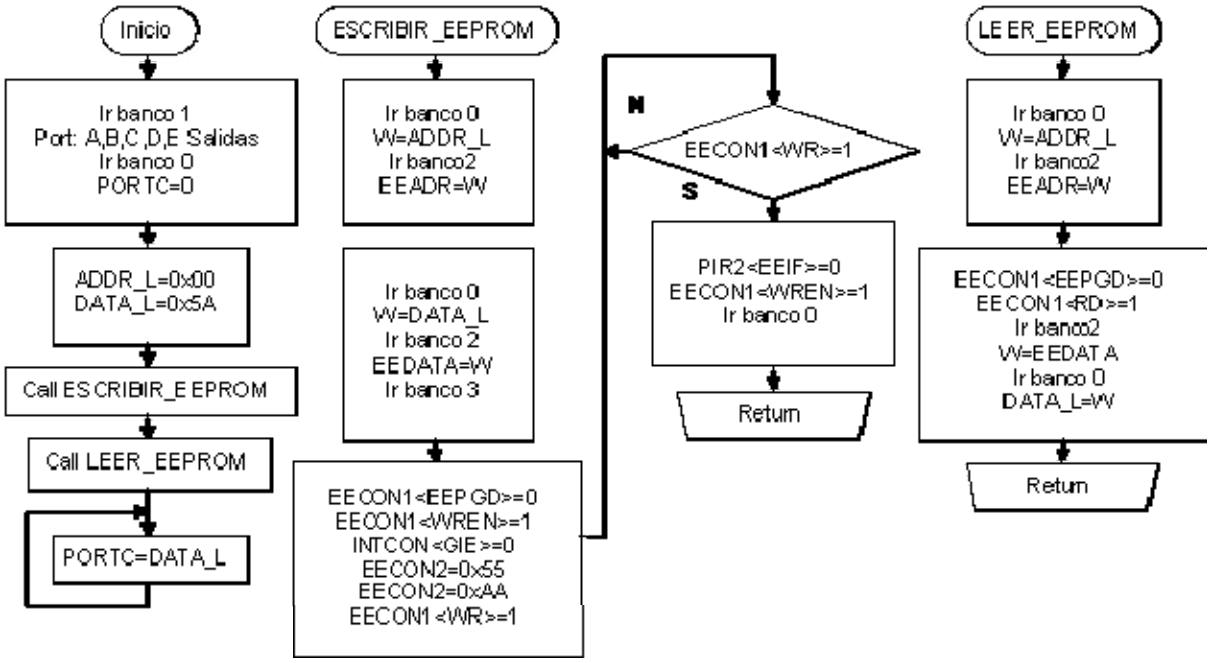
Ejercicio 24

Diseñe un programa que permita escribir y leer un byte en la posición 0x00 de la memoria EEPROM

a) Algoritmo

Para la implementación de programa usaremos dos rutinas: una que graba (ESCRIBIR_EEPROM) y una que lee (LEER_EEPROM). En ambos casos definimos dos variables ADDR_L (0x20) y DATA_L(0x21). El proceso de lectura o grabación se realiza conforme lo establecido por el fabricante. Antes de llamar a la rutina ESCRIBIR_EEPROM es necesario cargar la dirección ADDR_L y el dato que deseamos grabar. Antes de invocar a la rutina LEER_EEPROM debemos cargar la dirección a leer, al terminar la rutina almacenamos el valor en DATA_L para que sea usado desde el programa principal.

a) Diagrama de Flujo



b) Código del programa

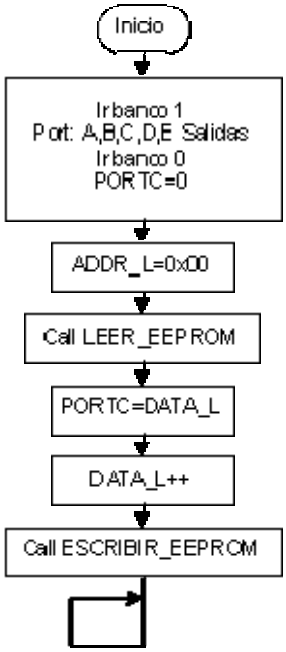
```
        list p=16f877 ;Comando que indica el Pic usado
    include "p16f877.inc" ;Etiquetas genéricas para el Pic16F877
    ADDR_L equ 0x20
    DATA_L equ 0x21
    org 0x000 ;Inicio del programa en la posición cero de memoria
    nop ;Libre (uso del debugger)
    nop
_inicio
    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1
    clrf TRISA ;PORTA salida
    clrf TRISB ;PORTB salida
    clrf TRISC ;PORTC salida
    clrf TRISD ;PORTD salida
    clrf TRISE
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    clrf PORTC ;Limpiar PORTC
    clrf ADDR_L ;ADDR_L = 0x00
    clrf DATA_L ;Limpiar registro de datos
    movlw 0x5A ;DATA_L = 0x5A
    movwf DATA_L
    call ESCRIBIR_EEPROM ;Llamar rutina de grabación
    clrf EEDATA ;Limpiar EEDATA
    call LEER_EEPROM ;Llamar rutina de lectura
_bucle
    movf DATA_L,W ;W = DATA_L
    movwf PORTC ;PORTC = W
    goto _bucle ;Ir _bucle

ESCRIBIR_EEPROM
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    movf ADDR_L,W ;EEADR = ADDR_L
```

```
    bcf STATUS,RP0 ;Ir banco 2
    bsf STATUS,RP1
    movwf EEADR
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    movf DATA_L,W ;EEDATA = DATA_L
    bcf STATUS,RP0 ;Ir banco 2
    bsf STATUS,RP1
    movwf EEDATA
    bsf STATUS,RP0 ;Ir banco 3
    bsf STATUS,RP1
    bcf EECON1,EEPGD ;Apuntar a la memoria EEPROM
    bsf EECON1,WREN ;Habilitar escritura
    bcf INTCON,GIE ;Deshabilita interrupciones
    movlw 55h
    movwf EECON2 ;Escribe 55 hexadecimal
    movlw 0xAA
    movwf EECON2 ;Escribe AA hexadecimal
    bsf EECON1,WR ;Habilita el bit de escritura
    ; bsf INTCON,GIE ;Habilita interrupciones
_bucle1
    btfsc EECON1,WR ;Espera el final de grabación
    goto _bucle1 ;Si no termina la grabación: Ir _bucle
    bcf PIR2,EEIF ;Si termina Borra bandera de interrupción
    bcf EECON1,WREN ;Deshabilitar escritura
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    return ;Retorno
LEER_EEPROM
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    movf ADDR_L,W ;Cargar dirección a leer
    bcf STATUS,RP0 ;Ir banco 2
    bsf STATUS,RP1
    movwf EEADR
    bsf STATUS,RP0 ;Ir banco 3
    bsf STATUS,RP1
    bcf EECON1,EEPGD ;Apunta a la memoria EEPROM
    bsf EECON1,RD ;Habilita ciclo de lectura
    bcf STATUS,RP0 ;Ir banco 2
    bsf STATUS,RP1
    movf EEDATA,W ;W = EEDATA (leer dato de EEPROM)
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    movwf DATA_L ;DATA_L = W (almacena dato de EEPROM)
    return ;Retorno
    end
```

Ejercicio 25 (Modulo 6 Ejercicio 2)

Elaborar un programa que lea la posición 0 de la memoria EEPROM, muestre el dato en el PORTC, incremente el valor del dato en una unidad, grabe el nuevo valor en la posición 0x00 de la memoria EEPROM. Demostrar que quitando la alimentación al microcontrolador se tiene el dato almacenado.

a) Diagrama de Flujo



Nota. -Los diagramas de flujo de las rutinas ESCRIBIR_EEPROM y LEER_EEPROM son los mismos que los que se mostraron en el programa anterior

b) Código del programa

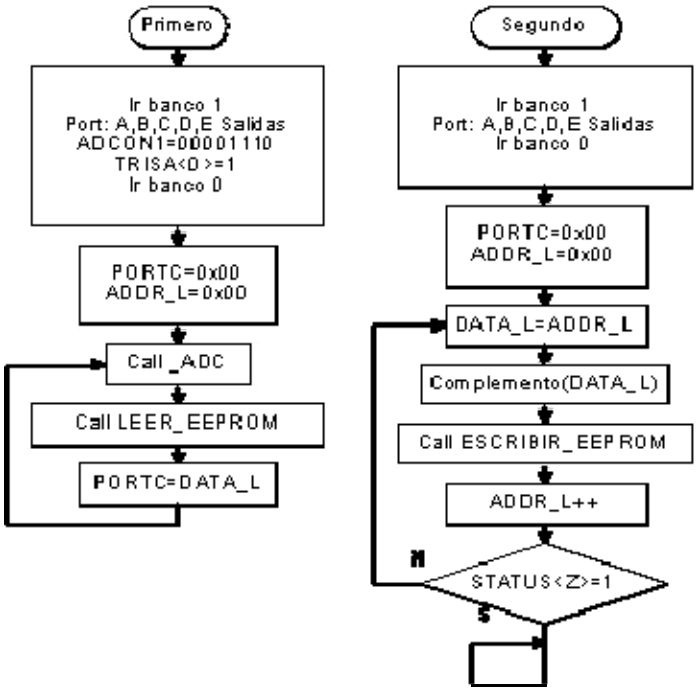
```
    list p=16f877 ;Comando que indica el Pic usado
    include "p16f877.inc" ;Etiquetas genéricas para el Pic16F877
    ADDR_L equ 0x20
    DATA_L equ 0x21
    org 0x00 ;Inicio del programa en la posición cero de memoria
    nop ;Libre (uso del debugger)
    nop
    nop
_inicio
    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1
    clrf TRISA ;PORTA salida
    clrf TRISB ;PORTB salida
    clrf TRISC ;PORTC salida
    clrf TRISD ;PORTD salida
    clrf TRISE
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    clrf PORTC ;Limpiar PORTC
    clrf ADDR_L ;ADDR_L = 0x00 (Variables en banco2)
    call LEER_EEPROM ;Llamar rutina de lectura (Regresa en el banco0)
    movfw DATA_L ;W = DATA_L
    movwf PORTC ;PORTC = W
    incf DATA_L,F ;incremento del dato leído de la EEPROM
    call ESCRIBIR_EEPROM ;Llamar rutina de grabación
_bucle
    goto _bucle ;Ir _bucle

ESCRIBIR_EEPROM

      bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    movf ADDR_L,W ;EEADR = ADDR_L
```

```
    bcf STATUS,RP0 ;Ir banco 2
    bsf STATUS,RP1
    movwf EEADR
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    movf DATA_L,W ;EEDATA = DATA_L
    bcf STATUS,RP0 ;Ir banco 2
    bsf STATUS,RP1
    movwf EEDATA
    bsf STATUS,RP0 ;Ir banco 3
    bsf STATUS,RP1
    bcf EECON1,EEPGD ;Apuntar a la memoria EEPROM
    bsf EECON1,WREN ;Habilitar escritura
    bcf INTCON,GIE ;Deshabilita interrupciones
    movlw 55h
    movwf EECON2 ;Escribe 55 hexadecimal
    movlw 0xAA
    movwf EECON2 ;Escribe AA hexadecimal
    bsf EECON1,WR ;Habilita el bit de escritura
    ; bsf INTCON,GIE ;Habilita interrupciones
_bucle1
    btfsc EECON1,WR ;Espera el final de grabación
    goto _bucle1 ;Si no termina la grabación: Ir _bucle
    bcf PIR2,EEIF ;Si termina Borra bandera de interrupción
    bcf EECON1,WREN ;Deshabilitar escritura
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    return ;Retorno

LEER_EEPROM

    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    movf ADDR_L,W ;Cargar dirección a leer
    bcf STATUS,RP0 ;Ir banco 2
    bsf STATUS,RP1
    movwf EEADR
    bsf STATUS,RP0 ;Ir banco 3
    bsf STATUS,RP1
    bcf EECON1,EEPGD ;Apunta a la memoria EEPROM
    bsf EECON1,RD ;Habilita ciclo de lectura
    bcf STATUS,RP0 ;Ir banco 2
    bsf STATUS,RP1
    movf EEDATA,W ;W = EEDATA (leer dato de EEPROM)
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    movwf DATA_L ;DATA_L = W (almacena dato de EEPROM)
    return ;Retorno
    end
```

Ejercicio 26
Diseñe dos programas:

- **El primero graba 255 en la posición 0x00 de la EEPROM, luego 254 en la posición 0x01, y asi sucesivamente hasta 0 en la posición 0xFF.**
- **El segundo programa lee el canal AN0 del ADC y según esa lectura obtiene una dirección de 8 bits que usa para extraer un dato de la EEPROM y lo muestra en el PORTC.**

a) Diagrama de Flujo



Nota:- Los diagrama de flujos de las rutina LEER_EEPROM y ESCRIBIR_EEPROM ya fueron explicados. Escribir EEPROM sigue la logica de los programas expuestos con anterioridad.

b) Código del programa

PRIMER PROGRAMA

```
    list p=16f877 ;Comando que indica el Pic usado
    include "p16f877.inc" ;Etiquetas genéricas para el Pic16F877
    ADDR_L equ 0x20
    DATA_L equ 0x21
    org 0x000 ;Inicio del programa en la posición cero de memoria
    nop ;Libre (uso del debugger)
    nop
    nop
_inicio
    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1
    clrf TRISA ;PORTA salida
    clrf TRISB ;PORTB salida
    clrf TRISC ;PORTC salida
    clrf TRISD ;PORTD salida
    clrf TRISE
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    clrf PORTC ;Limpiar PORTC
    clrf ADDR_L ;ADDR_L = 0x00
_graba
    movfw ADDR_L
    movwf DATA_L
    comf DATA_L,F
    call ESCRIBIR_EEPROM ;Llamar rutina de grabación
    incf ADDR_L,F
    movfw ADDR_L
    btfss STATUS,Z
```

```
    goto _graba
_bucle
    goto _bucle
ESCRIBIR_EEPROM

    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    movf ADDR_L,W ;EEADR = ADDR_L
    bcf STATUS,RP0 ;Ir banco 2
    bsf STATUS,RP1
    movwf EEADR
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    movf DATA_L,W ;EEDATA = DATA_L
    bcf STATUS,RP0 ;Ir banco 2
    bsf STATUS,RP1
    movwf EEDATA
    bsf STATUS,RP0 ;Ir banco 3
    bsf STATUS,RP1
    bcf EECON1,EEPGD ;Apuntar a la memoria EEPROM
    bsf EECON1,WREN ;Habilitar escritura
    bcf INTCON,GIE ;Deshabilita interrupciones
    movlw 55h
    movwf EECON2 ;Escribe 55 hexadecimal
    movlw 0xAA
    movwf EECON2 ;Escribe AA hexadecimal
    bsf EECON1,WR ;Habilita el bit de escritura
    ; bsf INTCON,GIE ;Habilita interrupciones
_bucle1
    btfsc EECON1,WR ;Espera el final de grabación
goto _bucle1 ;Si no termina la grabación: Ir _bucle
    bcf PIR2,EEIF ;Si termina Borra bandera de interrupción
    bcf EECON1,WREN ;Deshabilitar escritura
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    return ;Retorno
    end
```

SEGUNDO PROGRAMA

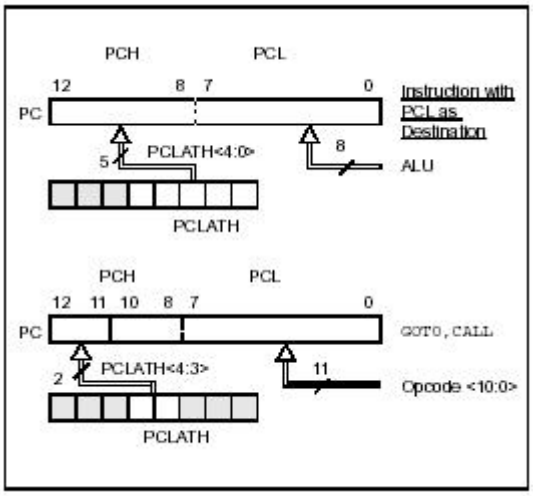
```
    list p=16f877 ;Comando que indica el Pic usado
    include "p16f877.inc" ;Etiquetas genéricas para el Pic16F877

    ADDR_L equ 0x20
    DATA_L equ 0x21

    org 0x000 ;Inicio del programa en la posición cero de memoria
    nop ;Libre (uso del debugger)
    nop
    nop
_inicio
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    movlw b'01000001' ;A/D conversion Fosc/8
    movwf ADCON0
    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1
    clrf TRISA ;PORTA salida
    clrf TRISB ;PORTB salida
    clrf TRISC ;PORTC salida
    clrf TRISD ;PORTD salida
```

```
    clrf TRISE
    movlw b'00001110' ;A/D Port AN0/RA0
    movwf ADCON1
    bsf TRISA,0
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    clrf PORTC ;Limpiar PORTC
    clrf ADDR_L ;ADDR_L = 0x00
_leer
    call _adc
    call LEER_EEPROM
    movf DATA_L
    movwf PORTC
    goto _leer
_adc
    bsf ADCON0,GO ;Start A/D conversion
_espera
    btfsc ADCON0,GO ;ADCON0 es 0? (la conversion esta completa?)
    goto _espera ;No, ir _espera
    movf ADRESH,W ;Si, W=ADRESH
    movwf ADDR_L ;Muestra el resultado en PORTC
    return
LEER_EEPROM

    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    movf ADDR_L,W ;Cargar dirección a leer
    bcf STATUS,RP0 ;Ir banco 2
    bsf STATUS,RP1
    movwf EEADR
    bsf STATUS,RP0 ;Ir banco 3
    bsf STATUS,RP1
    bcf EECON1,EEPGD ;Apunta a la memoria EEPROM
    bsf EECON1,RD ;Habilita ciclo de lectura
    bcf STATUS,RP0 ;Ir banco 2
    bsf STATUS,RP1
    movf EEDATA,W ;W = EEDATA (leer dato de EEPROM)
    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    movwf DATA_L ;DATA_L = W (almacena dato de EEPROM)
    return ;Retorno

end
```

Módulo 7: Manejo de Páginas de Memoria y Watch Dog

PCL Y PCLATCH

El contador de programa (PC) es un registro que tiene 13 bits para direccionar las 8K posiciones de memoria que tiene el microcontrolador. Los bits menos significativos se almaceman en el registro PCL, que es un registro que se puede leer o escribir. Los bits mas significativos PC<12:8> no se pueden leer pero pueden ser **ESCRITOS INDIRECTAMENTE** a traves del registro PCLATCH. Como lo muestra la figura:



Cuando se produce un reset o empieza a funcionar el microcontrolador los 5 bits superiores del PC son puestos a 0-lógico. Los valores en el registro PCLATH son 0. El microcontrolador tiene 8K posiciones de memoria distribuidos en 4 paginas de 2K cada una; por tanto los bits 4 y 3 del registro PCLATH nos ayudarán a saltar de una a otra página.

El registro PLATCH es un buffer que cuyos valores podemos modificar a voluntad para seleccionar una página solo en el momento que se lleve a efecto una instrucción CALL o una instrucción GOTO los valores del PCLATH (3 y 4) serán escritos en el contador de programa (PC).

La Pila

El PIC16F877 tiene una pila de 8 niveles en hardware y es exclusiva para el PC. Existe un puntero de pila que apunta a la última posición disponible (en un reset apunta a la posición 0). El puntero de pila no se puede leer ni escribir. Generalmente cuando usamos una pila empleamos las instrucciones PUSH (colocar en pila) y POP (extraer de la pila). En el caso del PIC no hay tales instrucciones . La colocación de valores dentro de la pila se hacen a través de la instrucción CALL. La extracción se hace por medio de las instrucciones RETURN, RETLW y RETFIE. **El PCLATH no se ve afectado por la extracción o colocación de valores en la pila condición que debemos tener presente al momento de hacer uso de varias páginas de memoria.**

La pila opera como un buffer circular, es decir que si se hacen 9 operaciones PUSH estaremos sobreescribiendo la primera posición de la pila. Si fueran 10 perderiamos los dos primeros valores.

Paginación de la memoria de programa

Los microcontroladores son capaces de direccionar hasta 8K posiciones de memoria de programa. Las instrucciones CALL y GOTO nos permiten saltar a una posicion dentro de un espacio de 2K por que solo tienen 11 bits para direccionar. **Cuando hacemos una instrucción CALL o GOTO los dos bits superiores del PCLATH (PCLATH<4:3>) son cargados a los bits 12 y 11 del contador de programa**. Por tanto antes de hacer un CALL o un GOTO debemos poner alli la combinación apropiada para saltar con exito. En los programas que hemos venido desarrollando no habia sido necesario modificar el PCLATH<4:3> por que los programas ocupan menos de una página y ademas era conveniente centrarse en el manejo de los módulos del PIC16F877.Si se hace una instrucción RETURN de una subrutina o una interrupción los 13 bits del PC son cargados nuevamente por tanto la manipulación de PCLATH<4:3> no es requerida.

Las páginas de memoria son como se muestran:

Pagina 0		**Pagina 1**		**Pagina 2**		**Pagina 3**
0 (0x00)		2048(0x800)		4096(0x1000)		6144(0x1800)
2047(0x7FF)		4095(0xFFF)		6143(0x17FF)		8191(0x1FFF)

La combinación de los bits 4 y 3 del PCLATCH nos permite seleccionar una página de memoria como se muestra:

Pagina	**PCLATH<4>**	**PCLATH<3>**
Pagina 0	0	0
Pagina 1	0	1
Pagina 2	1	0
Pagina 3	1	1

Siempre que deseamos hacer un salto de página debemos cargar el PCLATCH con la combinación apropiada. Y cuando regresamos de la llamada es necesario "liberar la página". Esto es restituir los valores para que el PCLATCH este en la página previa a la llamada:

; Supongamos que estamos en la página 0 y vamos a llamar a una rutina que esta en la página 3

```
; PCLATCH<4:3> = <0:0>

org 0x00 ;Pagina 0
............
    nop
    bsf PCLATCH,4 ; Pagina 3 => PCLATH<4:3>= <1:1>
    bsf PCLATCH,3
    call _pagina_3 ; Va a ejecutar la rutina pagina_3 en la pagina 3
    bcf PCLATCH,4 ; Pagina 0 => PCLATH<4:3>= <0:0>
    bcf PCLATCH,3 ; LIBERAR LA PAGINA
```

Ejercicio 27
Desarrolle un programa que desarrolle la secuencia

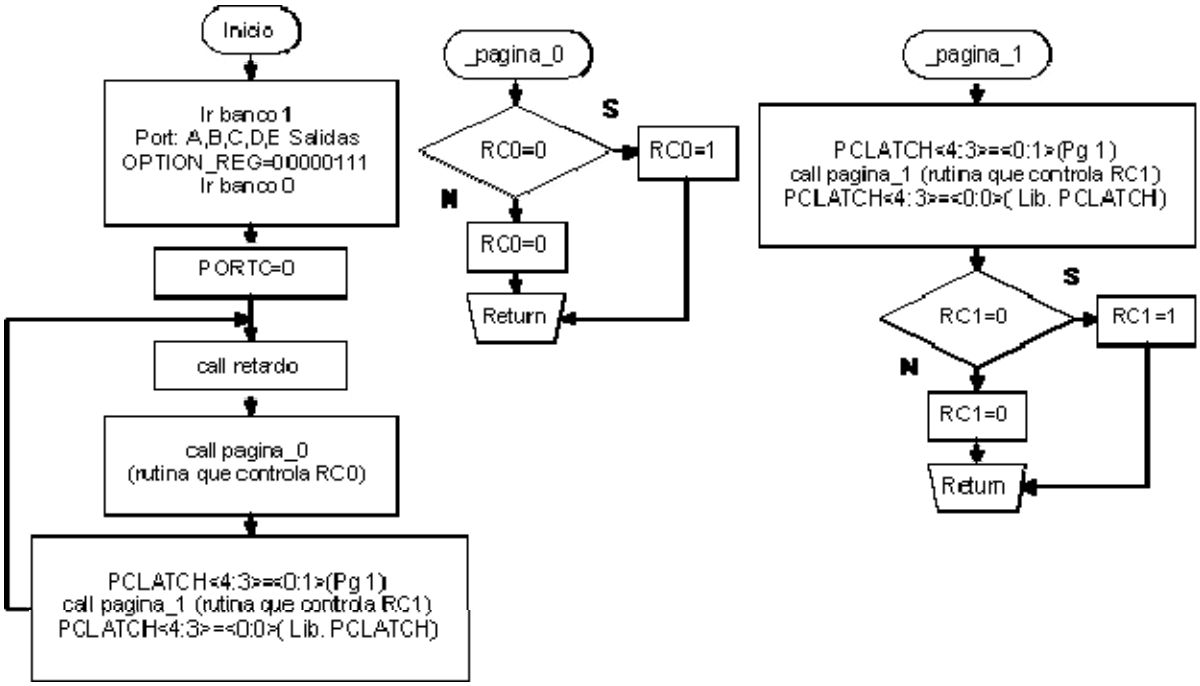
- **Retardo de mas de 1 segundo Led conectado a RC0=ON**
- **Retardo de mas de 1 segundo Led conectado a RC1=ON**
- **Retardo de mas de 1 segundo Led conectado a RC0=OFF**
- **Retardo de mas de 1 segundo Led conectado a RC1=OFF**

Definiciones previas
Las condiciones seran las siguiente: Debe existir la rutina de retardo en base al TMR0 y el código debe estar en la página 0. La rutina que controla RC0 debe estar en la página 0. La rutina que controla RC1 debe estar en la página 1 y debe llamar a la rutina retardo de la página 0. Si hacemos un resumen de las rutinas y las asociamos a las páginas de memoria tendriamos:

Pagina 0	***Pagina 1***
org 0x00 configurar bucle call retardo call rc0 call rc1 goto bucle retardo{ .....} rc0{.....}	**org 0x800** rc1{..... call Retardo ..... }

Diseño del programa

a) Diagrama de flujo



Como se observa el manejo de páginas es sencillo, basta con cargar en el PCLATCH los valores correspondientes a la página hacer la llamada a la función y desde luego LIBERAR LA PAGINA para evitar la pérdida del contador de programa. Si no liberamos la página y hacemos una instrucción goto o una instrucción call el contador de programa cargará los valores del PCLATH. En este caso en particular iría de nuevo a la página 1.

Nota:- No se incluye el diagrama de flujo del retardo por cuanto ya se ha tocado en el módulo de timers.

b) Código de programa

```
list p=16f877
include "p16f877.inc" ;Etiquetas genericas para el Pic16F877
TEMP EQU 0x20 ;Variable para temporizacion

;=====================================================================
==
org 0x0 ;PAGINA 0
;=====================================================================
==
    nop
    nop

    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1

    clrf TRISA
    clrf TRISB
    clrf TRISC
    clrf TRISD
    clrf TRISE

    movlw b'00000111' ;TMR0 preescaler 1:256
    movwf OPTION_REG

    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1

    clrf PORTC ;PORTC=0
_bucle
    call _retardo ;Llama a retardo (pagina 0 de memoria)
    call _pagina_0 ;Enciende o apaga el bit 0 del PORTC

;--------------------------------;
; Llamada a rutina en pagina 1 ;
;--------------------------------;
    bcf PCLATH,4 ;PCLATH=Pagina 1
    bsf PCLATH,3
    call _pagina_1 ;Enciendo o apaga el bit 1 del PORTC
    bcf PCLATH,4 ;PCLATH=Pagina 0, LIBERAR PCLATH
    bcf PCLATH,3

    goto _bucle ;Ir bucle

;=====================================================================
==
_retardo
    movlw d'25'
    movwf TEMP ;TEMP=25
_retardo_0
```

```
    bcf INTCON,T0IF
_retardo_1
    btfss INTCON,T0IF
    goto _retardo_1
    decfsz TEMP
    goto _retardo_0
    return

;=====================================================================
==
_pagina_0 ;PAGINA 0
    btfss PORTC,0 ;Es el valor de PORTC<0>=1
    goto _on_0 ;PORTC<0>=1
    bcf PORTC,0
    return
_on_0 bsf PORTC,0 ;PORTC<0>=0
return

;=====================================================================
==
org 0x800 ;PAGINA 1
;=====================================================================
==
_pagina_1
;---------------------------------------;
; Llamada a rutina _retardo en pagina 0 ;
;---------------------------------------;
    bcf PCLATH,4 ;PCLATH=Pagina 0
    bcf PCLATH,3
    call _retardo
    bcf PCLATH,4 ;PCLATH=Pagina 1, LIBERAR PCLATH
    bsf PCLATH,3

    btfss PORTC,1 ;Es el valor de PORTC<1>=1
    goto _on_1 ;PORTC<1>=1
    bcf PORTC,1 ;PORTC<1>=0
    return
_on_1 bsf PORTC,1
    return

    end
```

Metodología de acceso a funciones por medio de una solo página

Como hemos visto el acceso a funciones o rutinas que se encuentran en otras páginas es relativamente sencillo. Ahora suponga que tiene una rutina llamada 'A' que se encuentra en la página 0, esa rutina es llamada desde varias varias páginas como lo muestra la siguiente figura:

	Pagina 0		***Pagina 1***		***Pagina 2***		***Pagina 3***
	..... A nop ..... return		..... call A ..... call A		..... call A ..... call A		..... call A ..... call A

Ahora suponga que por algun motivo debe mover la rutina A de la página 0 a la página 3. Esta situacion podria llevar a multiples problemas por que todas las llamadas que se hace a lo largo del programa deben ser modificadas. Hay una forma practica de evitar este inconveniente.

La forma es similar a crear un indice de funciones. Suponga que creamos el indice en la página 0 entonces siempre que hacemos un call ingresamos a la página 0 (PCLATCH<4:3>=<0:0>) desde allí recién saltamos con un goto al lugar donde propiamente se encuentra el código de la función.

Observe el segmento de código:

```
org 0x00 ;Pagina 0
........
_a
    bcf PCLATH,4
    bcf PCLATH,3
    goto _a_code
    ........
_a_code
    nop
    ...
    return

org 0x800
.......
    bcf PCLATH,4 ;Pagina 0
    bcf PCLATH,3
    call A
    bcf PCLATH,4 ;Liberar pagina
    bsf PCLATH,3
    ........

org 0x1000 ;Pagina 2
    .......
    bcf PCLATH,4 ;Pagina 0
    bcf PCLATH,3
    call A
    bsf PCLATH,4 ;Liberar pagina
    bcf PCLATH,3
    ........
```

Entonces cuando queremos llamar a la rutina A lo que hacemos es cargar el PCLATH para ir a la página 0 y, luego hacemos el CALL. El CALL almacenará la siguiente dirección. Entramos al página 0 y volvemos a cargar el PCLATH con la página donde esta el código propiamente dicho de la rutina A (página 0) que es _A_code. Saltamos a esa dirección con ayuda de la instrucción GOTO (no coloca nada en la pila del PC). Cuando termina _A_code con la instrucción return extrae de la pila el PC que le permite regresar a la página original de la llamada (no regresa a la página 0).

Ahora si quieramos mudar el código de la rutina A de la página 0 a la 3 tendriamos:

```
org 0x00 ;Pagina 0
........
_a
    bsf PCLATH,4
    bsf PCLATH,3
    goto _a_code
........
```

```
org 0x800
.......
    bcf PCLATH,4 ;Pagina 0
    bcf PCLATH,3
    call A
    bcf PCLATH,4 ;Liberar pagina
    bsf PCLATH,3
........

org 0x1000 ;Pagina 2
.......
    bcf PCLATH,4 ;Pagina 0
    bcf PCLATH,3
    call A
    bsf PCLATH,4 ;Liberar pagina
    bcf PCLATH,3
........

org 0x1800 ;Pagina 3
_a_code
    nop
    ...
    return
```

Note como ahora el resto del código de programa no sufre modificación alguna solo hemos cambiado la dirección en el indice que esta en la página 0.

Ejercicios

Ejercicio 28

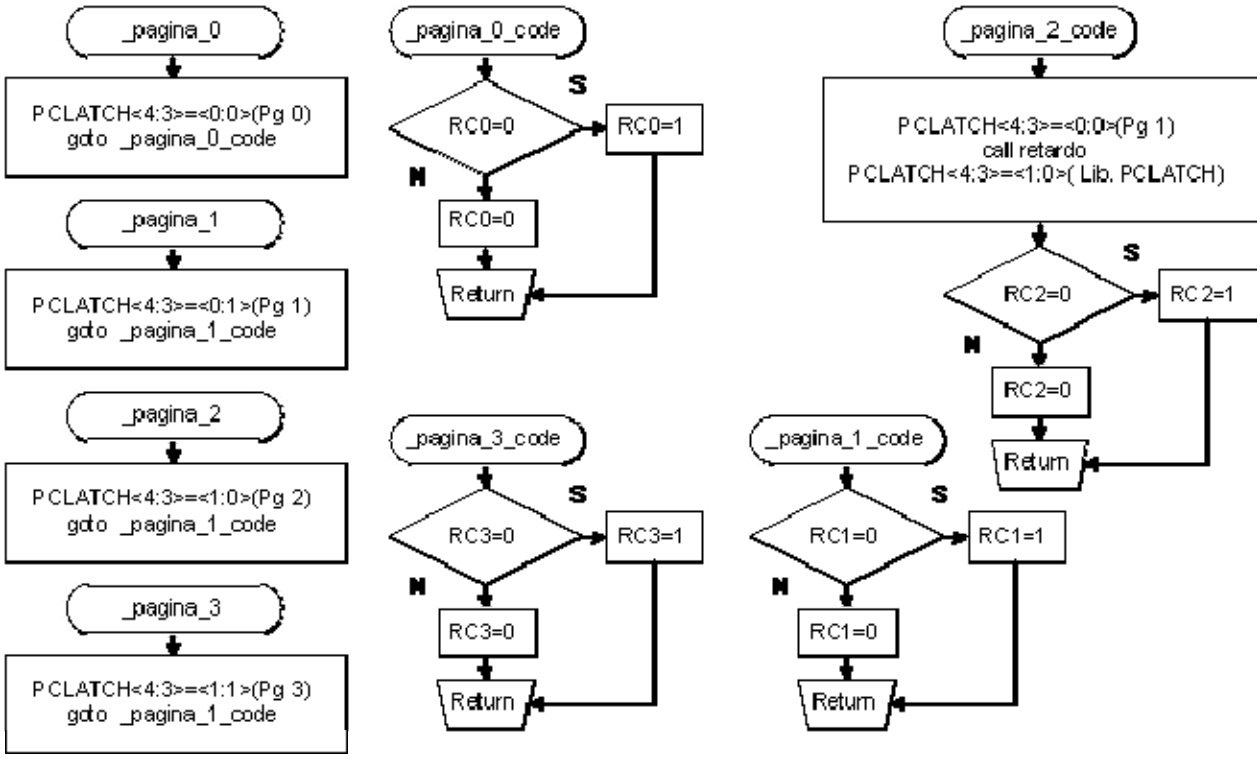
Desarrolle un programa que desarrolle la secuencia

- **Retardo de mas de 1 segundo Led conectado a RC0=ON**
- **Retardo de mas de 1 segundo Led conectado a RC1=ON**
- **Retardo de mas de 1 segundo Led conectado a RC2=ON**
- **Retardo de mas de 1 segundo Led conectado a RC3=ON**
- **Retardo de mas de 1 segundo Led conectado a RC0=OFF**
- **Retardo de mas de 1 segundo Led conectado a RC1=OFF**
- **Retardo de mas de 1 segundo Led conectado a RC2=OFF**
- **Retardo de mas de 1 segundo Led conectado a RC3=OFF**

Definiciones previas
En el programa usaremos un indice de páginas. Las llamadas seran hechas desde la página 0 a las rutinas que controlan el RC0(_pagina_0) , RC1(_pagina_1), RC2(_pagina_2), RC3 (_pagina_3). La rutina de RC2 (_pagina_2) llama a la función de retardo de la página 0.

Diseño del programa

a)Diagrama de flujo



b)Código del programa

```
list p=16f877
include "p16f877.inc" ;Etiquetas genericas para el Pic16F877
TEMP EQU 0x20 ;Variable para temporizacion

;=====================================================================
==
org 0x0 ;PAGINA 0
;=====================================================================
==
nop
nop

    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1

    clrf TRISA
    clrf TRISB
    clrf TRISC
    clrf TRISD
    clrf TRISE

    movlw b'00000111' ;TMR0 preescaler 1:256
    movwf OPTION_REG

    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1

    clrf PORTC ;PORTC=0
_bucle
;---------------------------------;
```

```
; Llamada a rutina en pagina 0 ;
;--------------------------------;

call _retardo ;Llama a retardo (pagina 0 de memoria)
    bcf PCLATH,4 ;PCLATH=Pagina 0, LIBERAR PCLATH
    bcf PCLATH,3
call _pagina_0 ;Enciendo o apaga el bit 0 del PORTC
    bcf PCLATH,4 ;PCLATH=Pagina 0, LIBERAR PCLATH
    bcf PCLATH,3

;--------------------------------;
; Llamada a rutina en pagina 1 ;
;--------------------------------;
call _retardo ;Llama a retardo (pagina 0 de memoria)
    bcf PCLATH,4 ;PCLATH=Pagina 0, LIBERAR PCLATH
    bcf PCLATH,3
call _pagina_1 ;Enciendo o apaga el bit 1 del PORTC
    bcf PCLATH,4 ;PCLATH=Pagina 0, LIBERAR PCLATH
    bcf PCLATH,3

;--------------------------------;
; Llamada a rutina en pagina 2 ;
;--------------------------------;
;El RETARDO esta dentro de la rutina _pagina_2
call _pagina_2 ;Enciendo o apaga el bit 2 del PORTC
    bcf PCLATH,4 ;PCLATH=Pagina 0, liberar PCLATH
    bcf PCLATH,3

;--------------------------------;
; Llamada a rutina en pagina 3 ;
;--------------------------------;
call _retardo
    bcf PCLATH,4 ;PCLATH=Pagina 0, LIBERAR PCLATH
    bcf PCLATH,3
    call _pagina_3 ;Enciendo o apaga el bit 2 del PORTC
    bcf PCLATH,4 ;PCLATH=Pagina 0, liberar PCLATH
    bcf PCLATH,3

goto _bucle ;Ir bucle

;=====================================================================
==
_pagina_0_code ;PAGINA 0
    btfss PORTC,0 ;Es el valor de PORTC<0>=1
    goto _on_0 ;PORTC<0>=1
    bcf PORTC,0
    return
_on_0 bsf PORTC,0 ;PORTC<0>=0
    return

;=====================================================================
==
; DEFINICIONES DE SALTOS. Similar a declarar funciones
;=====================================================================
==

_pagina_0
    bcf PCLATH,4 ;Pagina 0
    bcf PCLATH,3
    goto _pagina_0_code
```

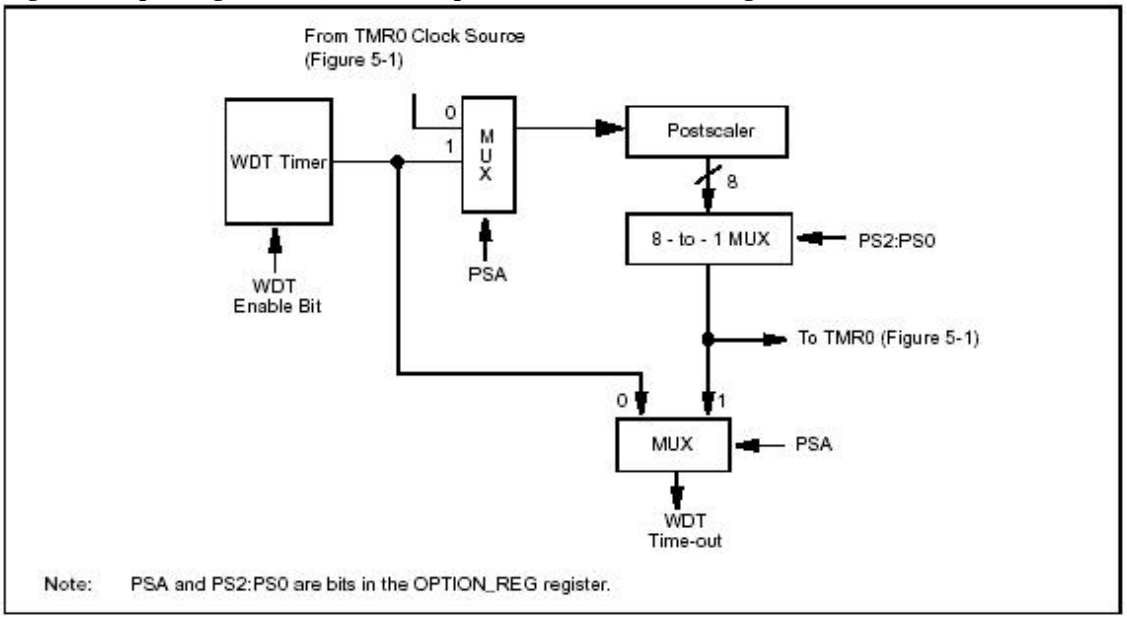
```
_pagina_1
    bcf PCLATH,4 ;Pagina 1
    bsf PCLATH,3
    goto _pagina_1_code
_pagina_2
    bsf PCLATH,4 ;Pagina 2
    bcf PCLATH,3
    goto _pagina_2_code

_pagina_3
    bsf PCLATH,4 ;Pagina 3
    bsf PCLATH,3
    goto _pagina_3_code

_retardo
    bsf PCLATH,4 ;Pagina 3
    bsf PCLATH,3
    goto _retardo_code

;=================================================================
==
org 0x800 ;PAGINA 1
;=================================================================
==
_pagina_1_code
    btfss PORTC,1 ;Es el valor de PORTC<1>=1
    goto _on_1 ;PORTC<1>=1
    bcf PORTC,1 ;PORTC<1>=0
    return
_on_1 bsf PORTC,1
    return

;=================================================================
==
org 0x1000 ;PAGINA 2
;=================================================================
==
_pagina_2_code

;---------------------------------;
; Llamada a rutina _retardo ;
;---------------------------------;
    bcf PCLATH,4 ;PCLATH=Pagina 0
    bcf PCLATH,3
    call _retardo
    bsf PCLATH,4 ;PCLATH=Pagina 2, LIBERAR PCLATH
    bcf PCLATH,3

    btfss PORTC,2 ;Es el valor de PORTC<2>=1
    goto _on_2 ;PORTC<2>=1
    bcf PORTC,2 ;PORTC<2>=0
    return
_on_2 bsf PORTC,2
    return

;=================================================================
==
org 0x1800 ;PAGINA 3
;=================================================================
==
_pagina_3_code
```

```
    btfss PORTC,3 ;Es el valor de PORTC<2>=1
    goto _on_3 ;PORTC<3>=1
    bcf PORTC,3 ;PORTC<3>=0
    return
_on_3 bsf PORTC,3
    return

;=====================================================================
==
_retardo_code
    movlw d'25'
    movwf TEMP ;TEMP=25
_retardo_0
    bcf INTCON,T0IF
_retardo_1
    btfss INTCON,T0IF
    goto _retardo_1
    decfsz TEMP
    goto _retardo_0
    return

end
```

PERRO GUARDIAN (WDR: WATCHDOG TIMER)

El WDT de los PIC16F87x es un contador que funciona con los impulsos de su propio oscilador y que provoca un reset cuando se desborda en funcionamiento normal. Si el desbordamiento se produce cuando en microcontrolador se halla en estado de reposo, se despierta y sigue su comportamiento normal. En la figura se muestra un esquema con los bloques principales que constituyen el perro guardian.

Las instrucciones CLRWDT y SLEEP borran o ponen a cero el valor del contaje del WDT y el postdivisor. Si se ejecuta la instrucción CLRWDT y el predivisor de frecuencia esta asignado al perro guardian, se borra, pero no cambia su configuración.



MODO DE REPOSO O DE BAJO CONSUMO

En este modo especial de funcionamiento del microcontrolador se introduce cuando se ejecuta la instrucción SLEEP. Esta forma de trabajo se caracteriza por su bajo consumo y pareciera que el PIC se ha "congelado". Las líneas de E/S digitales que se utilizaban mantienen su estado, las que no se empleaban reducen al mínimo su consumo, se detienen los temporizadores y tampoco opera el conversor A/D.

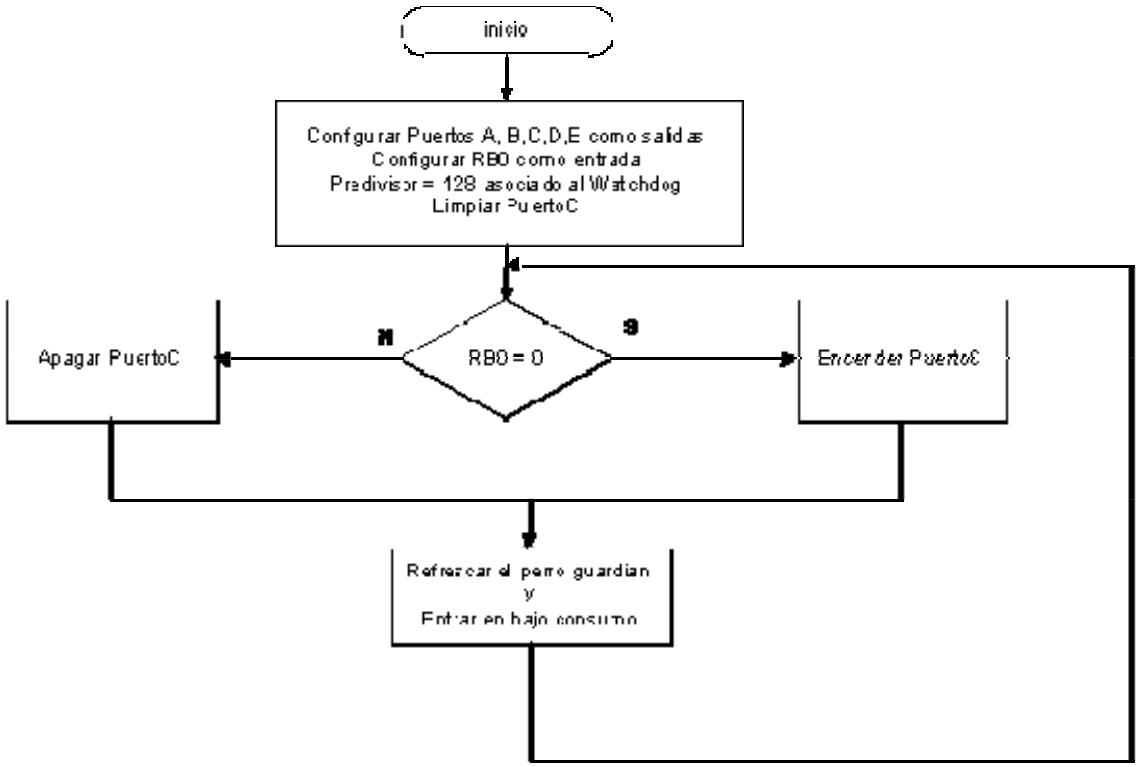
Al entrar en el modo de reposo, si estaba funcionando el WDT se borra pero sigue trabajando. Para salir de ese estado ("despertar") y pasar a ejecutar la instrucción direccionada por PC+1 existen varias causas.

1. Activacion externa de la patita MCLR#
2. Desbordamiento del WDT que sigue trabajando en reposo
3. Generacion de interrupcion por la activacion del pin RB0/INT o por cambio de estado en los 4 pines de mas peso del puerto B
4. Interrupcion originada por alguno de los nuevos perifericos como:

a) Lectura o escritura en la puerta paralela (PSP)
b) Interrupción del Timer1
c) Interrupción del módulo CCP en modo captura
d) Disparo especial del TMR1 funcionando en el modo asíncrono con reloj externo.
e) Interrupción en el módulo de comunicación SSP (Start/Stop)
f) Transmisión o recepción del MSSP en modo escalvo (SPI/I2C)
g) Transmisión o recepción del USART
h) Fin de la conversión A/D
i) Fin de la operación de escritura sobre la EEPROM.

Ejercicio

Ejercicio 29
Según el valor del interruptor RB0, se encenderán o apagarán los leds del puertoC pero, antes de volver a mirar el valor de dichos interruptores, se introducirá al microcontrolador en estado de reposo, del cual despertara al desbordarse el perro guardian, iniciandose de nuevo el proceso.




```
list p=16f877 ;Comando que indica el Pic usado
include "p16f877.inc" ;Etiquetas genéricas para el Pic16F877

org 0x000 ;Inicio del programa en la posición cero de memoria
    nop ;Libre (uso del debugger)
    nop

goto _inicio

org 0x005

_inicio
    bsf STATUS,RP0 ;Ir banco 1
    bcf STATUS,RP1

    clrf TRISA ;PORTA salida
    clrf TRISB ;PORTB salida
    clrf TRISC ;PORTC salida
    clrf TRISD ;PORTD salida
    clrf TRISE ;PORTE salida

    bsf TRISB,0 ;RB0 como entrada

    movlw b'11001111' ;Preescaler = 128
    movwf OPTION_REG ;Asociado al perro guardian

    bcf STATUS,RP0 ;Ir banco 0
    bcf STATUS,RP1
    clrf PORTC ;PORTC = 0
    ; clrwdt
    _bucle
    btfsc PORTB,0 ;RB0 = 0 ?
```

```
    goto _apagar ;no
    goto _encender ;si
_apagar
    clrf PORTC ;limpiar puertoC
    goto _reposo ;ir _reposo
_encender
    movlw 0xff
    movwf PORTC ;poner a 1 todo el puertoC
    goto _reposo ;ir _reposo
_reposo
    clrwdt ;refresca el perro guardian
    sleep ;entrar en bajo consumo
goto _bucle ;ir a _bucle

end ;Fin de programa
```

ANEXO 1 Tabla de códigos ASCII

0		32		64	@	96	`	128	Ç	160	á	192	└	224	α
1	☺	33	!	65	A	97	a	129	ü	161	í	193	┴	225	ß
2	☻	34	"	66	B	98	b	130	é	162	ó	194	┬	226	Γ
3	♥	35	#	67	C	99	c	131	â	163	ú	195	├	227	π
4	♦	36	$	68	D	100	d	132	ä	164	ñ	196	─	228	Σ
5	♣	37	%	69	E	101	e	133	à	165	Ñ	197	┼	229	σ
6	♠	38	&	70	F	102	f	134	å	166	ª	198	╞	230	µ
7	•	39	'	71	G	103	g	135	ç	167	º	199	╟	231	τ
8	◘	40	(	72	H	104	h	136	ê	168	¿	200	╚	232	Φ
9	○	41	)	73	I	105	i	137	ë	169	⌐	201	╔	233	Θ
10	◙	42	*	74	J	106	j	138	è	170	¬	202	╩	234	Ω
11	♂	43	+	75	K	107	k	139	ï	171	½	203	╦	235	δ
12	♀	44	,	76	L	108	l	140	î	172	¼	204	╠	236	∞
13	♪	45	-	77	M	109	m	141	ì	173	¡	205	═	237	ø
14	♫	46	.	78	N	110	n	142	Ä	174	«	206	╬	238	ε
15	☼	47	/	79	O	111	o	143	Å	175	»	207	╧	239	∩
16	►	48	0	80	P	112	p	144	É	176	░	208	╨	240	≡
17	◄	49	1	81	Q	113	q	145	æ	177	▒	209	╤	241	±
18	↕	50	2	82	R	114	r	146	Æ	178	▓	210	╥	242	≥
19	!!	51	3	83	S	115	s	147	ô	179	│	211	╙	243	≤
20	¶	52	4	84	T	116	t	148	ö	180	┤	212	╘	244	⌠
21	§	53	5	85	U	117	u	149	ò	181	╡	213	╒	245	⌡
22	▬	54	6	86	V	118	v	150	û	182	╢	214	╓	246	÷
23	↨	55	7	87	W	119	w	151	ù	183	╖	215	╫	247	≈
24	↑	56	8	88	X	120	x	152	ÿ	184	╕	216	╪	248	°
25	↓	57	9	89	Y	121	y	153	Ö	185	╣	217	┘	249	∙
26	→	58	:	90	Z	122	z	154	Ü	186	║	218	┌	250	·
27	←	59	;	91	[	123	{	155	¢	187	╗	219	█	251	√
28	∟	60	<	92	\	124	¦	156	£	188	╝	220	▄	252	n
29	↔	61	=	93	]	125	}	157	¥	189	╜	221	▌	253	2
30	▲	62	>	94	^	126	~	158	₧	190	╛	222	▐	254	■
31	▼	63	?	95	_	127	⌂	159	ƒ	191	┐	223	▀	255	

ANEXO 3

Relación de ejercicios

Ejercicio 0: Desarrollar un programa que configure las líneas del puerto A como entrada y las líneas del puerto B como salida. Y que muestre en forma permamente la entrada del puerto A en el puerto B.

Ejercicio 1: Suponga una lampara que debe ser prendida o apagada desde tres puntos. Diseñe un programa que la encienda si y solo si hay dos interruptores activados.

Ejercicio 2: Diseñar un programa que simule a un comparador de 4 líneas.

Ejercicio 3: Se tiene tres válvulas (A,B y C) que alimentan un tanque, el tanque a su vez tiene una salida. Existen 3 sensores de nivel (X,Y y Z). Cuando el tanque está vacio los 3 sensores estan a 0-lógico y es necesario activar el trabajo de las tres bombas. Cuando se llena 1/3 del tanque el sensor X pasa a 1-lógico y la bomba C deja de funcionar. Cuando se llenan 2/3 del tanque el sensor Y esta activado y la bomba B deja de funcionar. Cuando esta lleno el tanque el sensor Z se activa y la bomba A deja de funcionar. Una vez que el tanque esta lleno este empieza a expulsar el líquido acumulado. Cuando los 3 sensores pasan a 0-lógico la secuencia antes descrita se repite ANTES NO.

Ejercicio 4: Diseñar un programa que configure el RB0 como entrada y el RC0 como salida y probarlo en el demoboard.

Ejercicio 5: Diseñar un programa que muestre en el puerto C los cuatro bits mas significativos activados si RB0 es 0 y los bits menos significativos activados si RB0 es 1.

Ejercicio 6 : Diseñar un programa que muestre el corrimiento de un bit en el puerto C. El corrimiento tendrá lugar cada vez que se pulse RB0

Ejercicio 7: Diseñar un programa que lea los 4 bits inferiores del puerto A y muestre el dato en un display de 7 segmentos que se encuentra conectado en el puerto C.

Ejercicio 8: Resuelva el Ejercicio 3 haciendo uso de tablas

Ejercicio 9: Diseñar un programa en base al PIC16F877 para contar eventos (flancos de bajada en RA4/T0CKI) y mostrar la cuenta en un display de 7 segmentos conectado al puerto B. Cuando las cuentas llegan a 9 pasan de nuevo a 0.

Ejercicio 10: Programar el TMR0 para generar un retardo de un segundo. A partir del cual se incrementa un contador cuyo valor se muestra por el PORTC

Ejercicio 11: Programar el TMR1 para generar un retardo de 524.2 ms. Cada vez que concluya el tiempo se activará el PORTC de forma escalonada

Ejercicio 12 : Modificar el programa anterior para que lea el pin RB0, cuando se pulse deshabilitara el TMR1 y si se deja de pulsar reanudara el timer.

Ejercicio 13: Diseñar un programa que permita leer el voltaje aplicado al canal 0 del módulo ADC, convertirlo a un valor digital de 10 bits y mostrar los ocho bits mas significativos en el PORTC

Ejercicio 14: Elaborar un programa que lea el canal 0 del modulo ADC y que muestre el resultado en los 6 bits menos significativos del PORTC (RC0-RC5) adicionalmente active un pin del Pic (RC7) cuando el valor adquirido por el ADC sea mayor a 512 y desactive cuando sea menor a ese valor.

Ejercicio 15: Un tanque es empleado para almacenar agua y se ha conectado un sensor para saber el nivel de agua. El sensor se ha conectado al canal 0 del Pic 16F877 y un indicador es accionado por el pin RC7. Desarrollar un programa que informe si el nivel de agua esta en el rango de 64 a 128 unidades del ADC RC7=0 (RC7=1 fuera del rango)

Ejercicio 16: Transmitir muestras analógicas provenientes del canal AN0 del ADC del PIC hasta la computadora y visualizar el dato por el hyper terminal de Windows.

Ejercicio 17: Recibir datos provenientes de la computadora de forma serial y mostrarlos en el PORTC

Ejercicio 18: Diseñe un programa en el PIC 16F877 que reciba datos de la computadora, los muestre por el PORTC y devuelva el mismo dato a la computadora para poder visualizar los datos por el hyper terminal de Windows

Ejercicio 19: N/A. Eliminado.

Ejercicio 20: Diseñar y simular un programa que trabaja la interrupción RB0/INT para ir acumulando los flancos de subida que vayan ingresando por el pin RB0. En este primer ejemplo el bucle principal del programa no “tareas”.

Ejercicio 21: Configurar el TMR1 para generar espacios de tiempo de 500ms. Cada vez que se cumpla el tiempo invertir el valor del RC7 (similar a un led intermitente). Emplear la interrupción RB0 para incrementar el valor del PORTC cada vez que llegue una petición.

Ejercicio 22: Configurar el canal AN0 del ADC para mostrar en forma permanente el valor de conversión en el puerto C. Cada vez que llegue una interrupción provocada por el hiperterminal el microcontrolador enviará a la PC el valor actual del ADC.

Ejercicio 23: Diseñe un programa en el Pic16F877 que funcione con la interrupción del USART. Si desde el hiperterminal se presiona la tecla ‘A’ RC0 = 1, ‘a’ RC0 = 0, ‘B’ RC1 = 1, ‘b’ RC1 = 0, ‘C’ RC2 = 1, ‘c’ RC2 = 0.

Ejercicio 24: Diseñe un programa que permita escribir y leer un byte en la posición 0x00 de la memoria EEPROM

Ejercicio 25: Elaborar un programa que lea la posición 0 de la memoria EEPROM, muestre el dato en el PORTC, incremente el valor del dato en una unidad, grabe el nuevo valor en la posición 0x00 de la memoria EEPROM. Demostrar que quitando la alimentación al microcontrolador se tiene el dato almacenado.

Ejercicio 26: Diseñe dos programas:

- El primero graba 255 en la posición 0x00 de la EEPROM, luego 254 en la posición 0x01, y asi sucesivamente hasta 0 en la posición 0xFF.

- El segundo programa lee el canal AN0 del ADC y según esa lectura obtiene una dirección de 8 bits que usa para extraer un dato de la EEPROM y lo muestra en el PORTC.

Ejercicio 27: Desarrolle un programa que desarrolle la secuencia

- Retardo de mas de 1 segundo Led conectado a RC0=ON
- Retardo de mas de 1 segundo Led conectado a RC1=ON
- Retardo de mas de 1 segundo Led conectado a RC0=OFF
- Retardo de mas de 1 segundo Led conectado a RC1=OFF

Ejercicio 28: Desarrolle un programa que desarrolle la secuencia

- Retardo de mas de 1 segundo Led conectado a RC0=ON
- Retardo de mas de 1 segundo Led conectado a RC1=ON
- Retardo de mas de 1 segundo Led conectado a RC2=ON
- Retardo de mas de 1 segundo Led conectado a RC3=ON
- Retardo de mas de 1 segundo Led conectado a RC0=OFF
- Retardo de mas de 1 segundo Led conectado a RC1=OFF
- Retardo de mas de 1 segundo Led conectado a RC2=OFF
- Retardo de mas de 1 segundo Led conectado a RC3=OFF

Ejercicio 29: Según el valor del interruptor RB0, se encenderán o apagarán los leds del puertoC pero, antes de volver a mirar el valor de dichos interruptores, se introducirá al microcontrolador en estado de reposo, del cual despertara al desbordarse el perro guardian, iniciandose de nuevo el proceso.