MANUAL DE MICROCONTROLADORES PIC

Contenido:

1.1 Arquitectura Harvard La arquitectura tradicional:

La arquitectura tradicional de computadoras y microprocesadores se basa en el esquema propuesto por John Von Neumann, en el cual la unidad central de proceso, o CPU, esta conectada a una memoria única que contiene las instrucciones del programa y los datos (figura 1.1.1). El tamaño de la unidad de datos o instrucciones esta fijado por el ancho del bus de la memoria. Es decir que un microprocesador de 8 bits, que tiene además un bus de 8 bits que lo conecta con la memoria, deberá manejar datos e instrucciones de una o más unidades de 8 bits (bytes) de longitud. Cuando deba acceder a una instrucción o dato de más de un byte de longitud, deberá realizar más de un acceso a la memoria. Por otro lado este bus único limita la velocidad de operación del microprocesador, ya que no se puede buscar de memoria una nueva instrucción, antes de que finalicen las transferencias de datos que pudieran resultar de la instrucción anterior. Es decir que las dos principales limitaciones de esta arquitectura tradicional son :

a) que la longitud de las instrucciones esta limitada por la unidad de longitud de los datos, por lo tanto el microprocesador debe hacer varios accesos a memoria para buscar instrucciones complejas,

b) que la velocidad de operación (o ancho de banda de operación) esta limitada por el efecto de cuello de botella que significa un bus único para datos e instrucciones que impide superponer ambos tiempos de acceso.

La arquitectura von Neumann permite el diseño de programas con código automodificable, práctica bastante usada en las antiguas computadoras que solo tenían acumulador y pocos modos de direccionamiento, pero innecesaria, en las computadoras modernas.

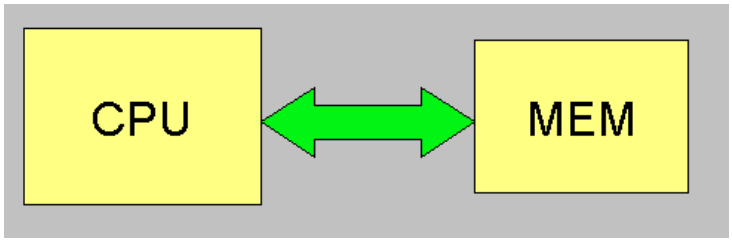


FIG. 1.1.1 Arquitectura Von Newmann

La arquitectura Harvard y sus ventajas:

La arquitectura conocida como Harvard, consiste simplemente en un esquema en el que el CPU esta conectado a dos memorias por intermedio de dos buses separados. Una de las memorias contiene solamente las instrucciones del programa, y es llamada Memoria de Programa. La otra memoria solo almacena los datos y es llamada Memoria de Datos (figura 1.1.2). Ambos buses son totalmente independientes y pueden ser de distintos anchos. Para un procesador de Set de Instrucciones Reducido, o RISC (Reduced Instrucción Set Computer), el set de instrucciones y el bus de la memoria de programa pueden diseñarse de manera tal que todas las instrucciones tengan una sola posición de memoria de programa de longitud. Además, como los buses son independientes, el CPU puede estar accediendo a los datos para completar la ejecución de una instrucción, y al mismo tiempo estar leyendo la próxima instrucción a ejecutar. Se puede observar claramente que las principales ventajas de esta arquitectura son:

a) que el tamaño de las instrucciones no esta relacionado con el de los datos, y por lo tanto puede ser optimizado para que cualquier instrucción ocupe una sola posición de memoria de programa, logrando así mayor velocidad y menor longitud de programa,

b) que el tiempo de acceso a las instrucciones puede superponerse con el de los datos, logrando una mayor velocidad de operación.

Una pequeña desventaja de los procesadores con arquitectura Harvard, es que deben poseer instrucciones especiales para acceder a tablas de valores constantes que pueda ser necesario incluir en los programas, ya que estas tablas se encontraran físicamente en la memoria de programa (por ejemplo en la EPROM de un microprocesador).

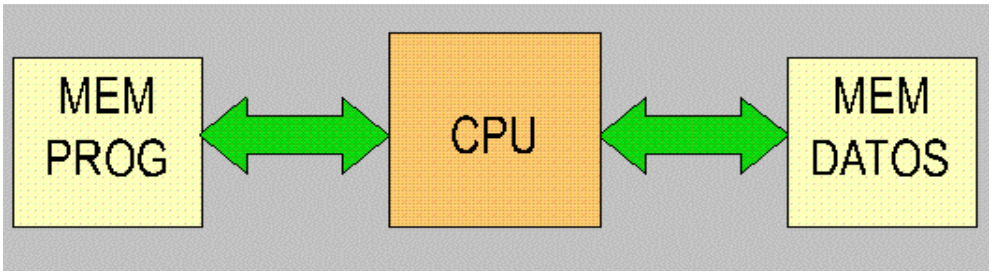
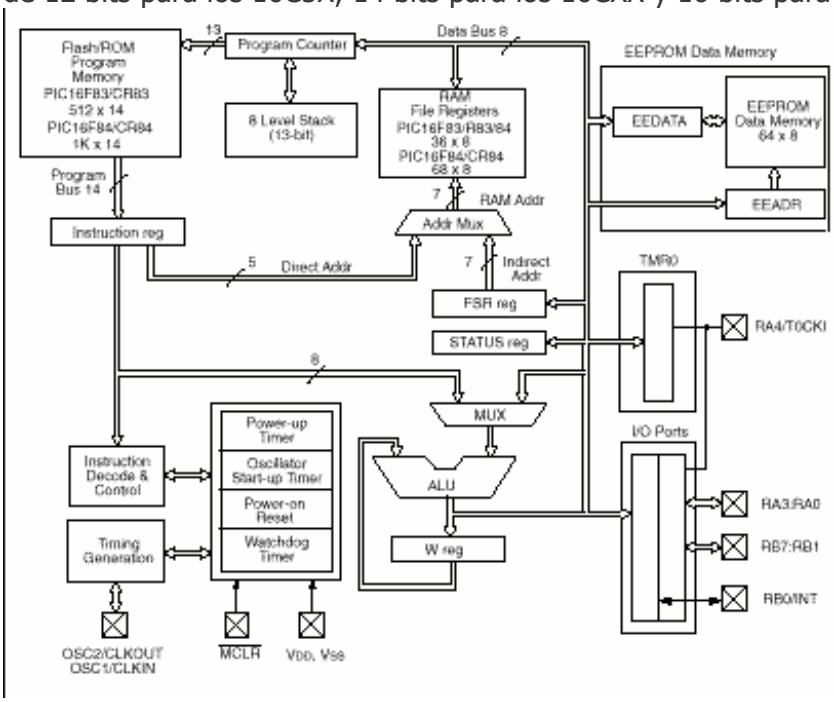


FIG. 1.1.2 Arquitectura Harvard

Los microcontroladores PIC 16C5X, 16CXX y 17CXX poseen arquitectura Harvard, con una memoria de datos de 8 bits, y una memoria de programa que, según el modelo, puede ser de 12 bits para los 16C5X, 14 bits para los 16CXX y 16 bits para los 17CXX.



1.2 Diagrama de bloques

Diagrama de bloques de los microcontroladores PIC16F8X

1.3 Mapas de memoria Memoria Interna (RAM)

Organización

La memoria interna de datos, también llamada archivo de registros (register file), esta dividida en dos grupos: los registros especiales, y los registros de propósito generales. Los primeros ocupan las 11 posiciones primeras que van desde la 00 a la 0B, y los segundos las posiciones que siguen, o sea de la 08 a la 4F.
Los registros especiales contienen la palabra de estado (STATUS), los registros de datos de los tres puertos de entrada salida (Puerto A, Puerto B, Puerto C), los 8 bits menos significativos del program counter (PC), el contador del Real Time Clock/Counter (RTCC) y un registro puntero llamado File Select Register (FSR). La posición 00 no contiene ningún registro en especial y es utilizada en el mecanismo de direccionamiento indirecto.

Los registros de propósito general se dividen en dos grupos : los registros de posición fija y los bancos de registros. Los primeros ocupan las 8 posiciones que van de la 08 a la 0F. los bancos de registros consisten en hasta cuatro grupos o bancos de 16 registros cada uno, que se encuentran superpuestos en las direcciones que van de la 10 a la 1F. Se puede operar con un solo banco a la vez, el cual se selecciona mediante los bits 5 y 6 del File Select Register (FSR)

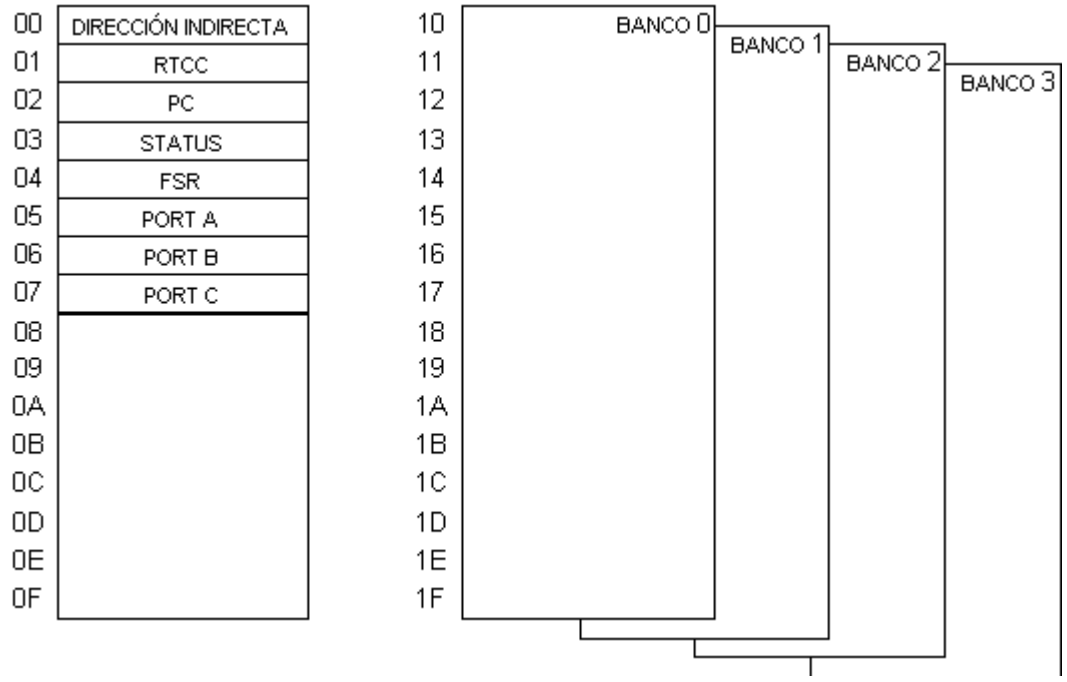


FIG. 1.3.1 Organización de la memoria Interna (RAM) en la familia PIC16C5X

Memoria de Programa

Organización

La memoria de programa, que en los PIC16C5X puede ser de 512 a 2K instrucciones, debe ser considerada a los efectos de la programación, como compuesta por secciones o páginas de 512 posiciones. A su vez cada página debe considerarse dividida en dos

mitades de 128 posiciones cada una. Esto se debe, como se verá, a las limitaciones de direccionamiento de las instrucciones de salto

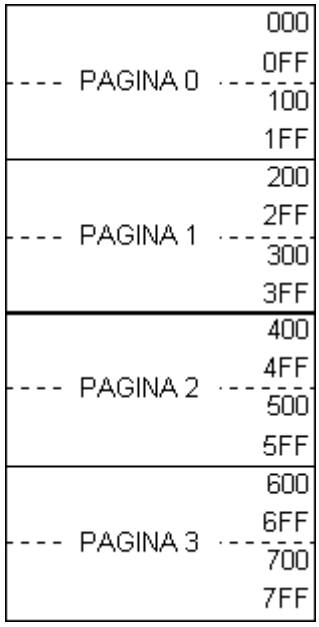
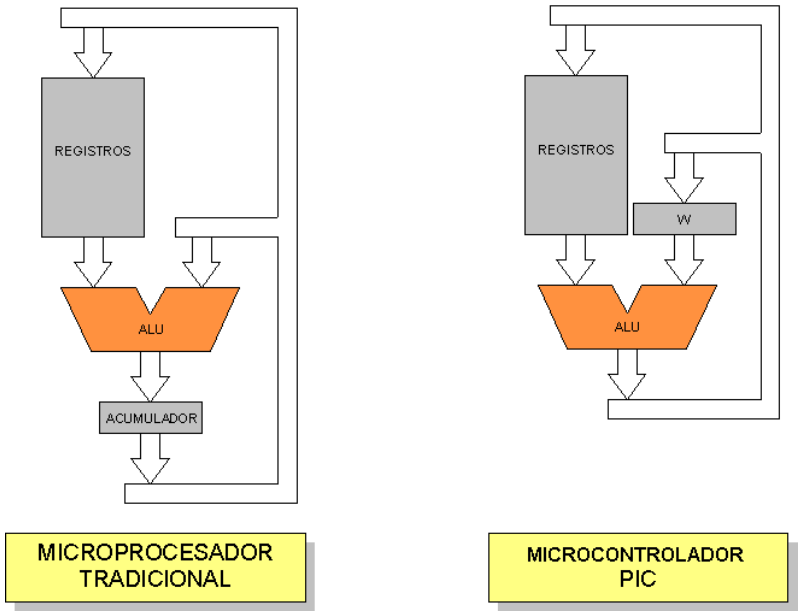


FIG. 1.3.2 Organización de la memoria de programa en la familia PIC16C5X

1.4 Registros de funciones especiales Camino de los datos y registro W

La figura 1.4.2 representa un diagrama simplificado de la arquitectura interna del camino de los datos en el CPU de los microcontroladores PIC. Este diagrama puede no representar con exactitud el circuito interno de estos microcontroladores, pero es exacto y claro desde la óptica del programador. La figura 1.4.1 representa el mismo diagrama para un microprocesador ficticio de arquitectura tradicional. Se puede observar que la principal diferencia entre ambos radica en la ubicación del registro de trabajo, que para los PIC's se denomina W (Working Register), y para los tradicionales es el Acumulador (A).



En los microcontroladores tradicionales todas las operaciones se realizan sobre el acumulador. La salida del acumulador esta conectada a una de las entradas de la Unidad Aritmética y Lógica (ALU), y por lo tanto éste es siempre uno de los dos operandos de cualquier instrucción. Por convención, las instrucciones de simple operando (borrar, incrementar, decrementar, complementar), actúan sobre el acumulador. La salida de la ALU va solamente a la entrada del acumulador, por lo tanto el resultado de cualquier operación siempre quedara en este registro. Para operar sobre un dato de memoria, luego realizar la operación siempre hay que mover el acumulador a la memoria con una instrucción adicional.

En los microcontroladores PIC, la salida de la ALU va al registro W y también a la memoria de datos, por lo tanto el resultado puede guardarse en cualquiera de los dos destinos. En las instrucciones de doble operando, uno de los dos datos siempre debe estar en el registro W, como ocurría en el modelo tradicional con el acumulador. En las instrucciones de simple operando el dato en este caso se toma de la memoria (también por convención). La gran ventaja de esta arquitectura es que permite un gran ahorro de instrucciones ya que el resultado de cualquier instrucción que opere con la memoria, ya sea de simple o doble operando, puede dejarse en la misma posición de memoria o en el registro W, según se seleccione con un bit de la misma instrucción. Las operaciones con constantes provenientes de la memoria de programa (literales) se realizan solo sobre el registro W.

En la memoria de datos de los PIC's se encuentran ubicados casi todos los registros de control del microprocesador y sus periféricos autocontenidos, y también las posiciones de memoria de usos generales. En el caso de los 16C5X, algunos registros especiales de solo escritura (TRIS y OPTION) no están accesibles dentro del bloque de memoria de datos, sino que solo se pueden cargar desde el registro W por medio de instrucciones especiales.

Contador de Programa

Este registro, normalmente denominado PC, es totalmente equivalente al de todos los microprocesadores y contiene la dirección de la próxima instrucción a ejecutar. Se incrementa automáticamente al ejecutar cada instrucción, de manera que la secuencia natural de ejecución del programa es lineal, una instrucción después de la otra. Algunas instrucciones que llamaremos de control, cambian el contenido del PC alterando la secuencia lineal de ejecución. Dentro de estas instrucciones se encuentran el GOTO y el CALL que permiten cargar en forma directa un valor constante en el PC haciendo que el programa salte a cualquier posición de la memoria. Otras instrucciones de control son los SKIP o "salteos" condicionales, que producen un incremento adicional del PC si se cumple una condición especifica, haciendo que el programa saltee, sin ejecutar, la instrucción siguiente.

El PC es un registro de 9 bits en los 16C54/55, 10 bits en el 16C56, y 11 bits en el 16C57, lo que permite direccionar respectivamente 512, 1024 o 2048 posiciones de memoria de programa.

Al resetearse el microprocesador, todos los bits del PC toman valor 1, de manera que la dirección de arranque del programa es siempre la ultima posición de memoria de programa. En esta posición se deberá poner una instrucción de salto al punto donde verdaderamente se inicia el programa.

A diferencia de la mayoría de los microprocesadores convencionales, el PC es también accesible al programador como registro de memoria interna de datos, en la posición de 02. Es decir que cualquier instrucción común que opere sobre registros puede ser utilizada para alterar el PC y desviar la ejecución del programa. El uso indiscriminado de este tipo de instrucciones complica el programa y puede ser muy peligroso, ya que puede producir comportamientos difíciles de predecir. Sin embargo, algunas de esta instrucciones utilizadas con cierto método, pueden ser muy útiles para implementar poderosas estructuras de control tales como el goto computado. Como el microprocesador opera con datos de 8 bits, y la memoria de datos es también de 8 bits, estas instrucciones solo pueden leer o modificar los bits 0 a 7 del PC.

Stack

En los microcontroladores PIC el stack es una memoria interna dedicada, de tamaño limitado, separada de las memorias de datos y de programa, inaccesible al programador, y organizada en forma de pila, que es utilizada solamente, y en forma automática, para guardar las direcciones de retorno de subrrutinas e interrupciones. Cada posición es de 11 bits y permite guardar una copia completa del PC. Como en toda memoria tipo pila, los datos son accedidos de manera tal que el primero que entra es el ultimo que sale.

En los 16C5X el stack es de solo dos posiciones, mientras que en los 16CXX es de 8 posiciones y en los 17CXX es de 16 posiciones. Esto representa, en cierta medida, una limitación de estos microcontroladores, ya que no permite hacer uso intensivo del anidamiento de subrutinas. En los 16C5X, solo se pueden anidar dos niveles de subrutinas, es decir que una subrutina que es llamada desde el programa principal, puede a su vez llamar a otra subrutina, pero esta ultima no puede llamar a una tercera, porque se desborda la capacidad del stack, que solo puede almacenar dos direcciones de retorno. Esto de hecho representa una traba para el programador y además parece impedir o dificultar la programación estructurada, sin embargo es una buena solución de compromiso ya que estos microcontroladores están diseñados para aplicaciones de alta velocidad en tiempo real, en las que el overhead (demoras adicionales) que ocasiona un excesivo anidamiento de subrutinas es inaceptable. Por otra parte existen técnicas de organización del programa que permiten mantener la claridad de la programación estructurada, sin necesidad de utilizar tantas subrutinas anidadas.

Como ya se menciono anteriormente, el stack y el puntero interno que lo direcciona, son invisibles para el programador, solo se los accede automáticamente para guardar o rescatar las direcciones de programa cuando se ejecutan las instrucciones de llamada o retorno de subrutinas, o cuando se produce una interrupción o se ejecuta una instrucción de retorno de ella.

Palabra de Estado del Procesador

La palabra de estado del procesador contiene los tres bits de estado de la ALU (C, DC y Z), y otros bits que por comodidad se incluyeron en este registro.

7 6 5 4 3 2 1 0 Registro STATUS

El bit Z indica que el resultado de la ultima operación fue CERO. El bit C indica acarreo del bit más significativo (bit 7) del resultado de la ultima operación de suma. En el caso de la resta se comporta a la inversa, C resulta 1 si no hubo pedido de préstamo. El bit DC (digit carry) indica acarreo del cuarto bit (bit 3) del resultado de la ultima operación de suma o

resta, con un comportamiento análogo al del bit C, y es útil para operar en BCD (para sumar o restar números en código BCD empaquetado). El bit C es usado además en las operaciones de rotación derecha o izquierda como un paso intermedio entre el bit 0 y el bit 7.

El bit PD (POWER DOWN) sirve para detectar si la alimentación fue apagada y encendida nuevamente, tiene que ver con la secuencia de inicialización, el watch dog timer y la instrucción sleep, y su uso se detallara en la sección referida al modo POWER DOWN. El bit TO (TIME-OUT) sirve para detectar si una condición de reset fue producida por el watch dog timer, esta relacionado con los mismos elementos que el bit anterior y su uso se detallara en la sección referida al WATCH DOG TIMER. Los bits de selección de pagina PA0/PA1/PA2 se utilizan en las instrucciones de salto GOTO y CALL, y se explicaran con detalle en la sección referida a las instrucciones de control, y a la organización de la memoria de programa. En realidad en el 16C54 estos bits no se usan y sirven para propósitos generales. En el 16C57 el PA0 si se usa pero los otros dos no. En el 16C55 se utilizan PA0 y PA1. PA2 esta reservado para uso futuro y en cualquiera de los PIC 16C5X sirve para propósitos generales.

OTROS REGISTROS ESPECIALES

Las ocho primeras posiciones del área de datos están reservadas para alojar registros de propósito especial, quedando las restantes libres para contener los datos u operandos que se desee (registros de propósito general).

El registro INDF que ocupa la posición 0 no está implementando físicamente y, como se ha explicado, se le referencia en el direccionamiento indirecto de datos aunque se utiliza el contenido de FSR.
En la dirección esta el registro TAR0 (Temporizador) que puede ser leído y escrito como cualquier otro registro. Puede incrementar su valor con una señal externa aplicada al pin T0CKI o mediante el oscilador interno.

El PC ocupa la posición 2 del área de datos en donde se halla el registro PCL al que se añaden 3 bits auxiliares y se conectan con los dos niveles de la Pila en las instrucciones CALL y RETLW.

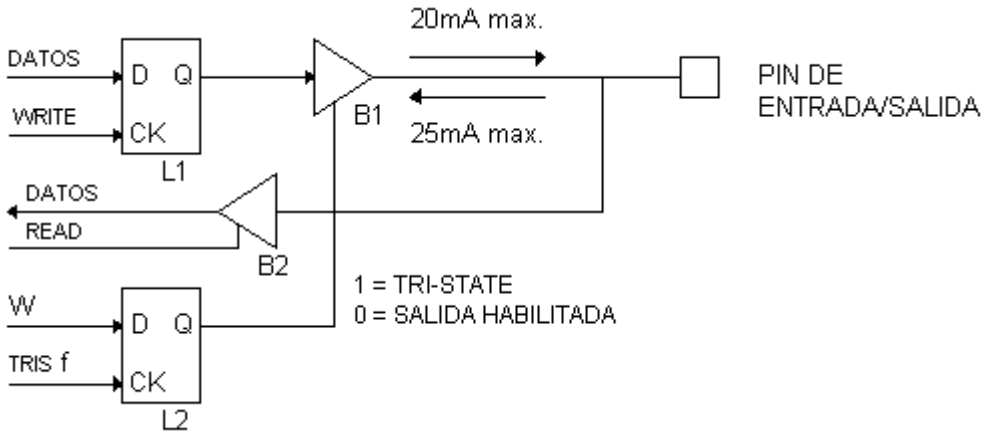
El registro de Estado ocupa la posición 3 y entre sus bits se encuentran los señalizadores C, DC y Z y los bits PA1 y PA0 que seleccionan la página en la memoria de programa. El bit 7 (PA2) no está implementando en los PIC de la gama baja.

FRS se ubica en la dirección 4 y puede usarse para contener las dirección del dato en las instrucciones con direccionamiento indirecto y también para guardar operandos en sus 5 bits de menos peso.

Los registros que ocupan las posiciones 5 ,6 y 7 soportan los Puertos A, B y C de E/S. Pueden ser leídos y escritos como cualquier otro registro y manejan los valores de los bits que entran y salen por los pines de E/S del microcontrolador.

1.5 Puertos de entrada / salida Los microprocesadores PIC16C5X tienen dos o tres puertos de entrada/salida paralelo de usos generales llamados Puerto A, Puerto B y Puerto C. El Puerto A es de cuatro bits y los demás son de 8 bits cada uno. El Puerto C solamente esta disponible en el 16C55 y el 16C57.

Circuito equivalente
El circuito equivalente de un bit cualquiera de un puerto de entrada salida es el siguiente



Circuito equivalente de puerto I/O

El latch L1 corresponde a un bit del registro de datos del puerto, mientras que L2 es un bit del registro de control de tristate del mismo. B1 es el buffer tristate de salida que tiene capacidad de entregar 20 mA y drenar 25 mA. B1 es controlado por L2. Si L2 tiene cargado un "1", B1 se encuentra en tri-state, es decir con la salida desconectada (en alta impedancia), y el puerto puede ser usado como entrada. Si L2 tiene cargado un "0", la salida de B1 esta conectada (baja impedancia) y el puerto esta en modo de salida. B2 es el buffer de entrada, es decir el que pone los datos en el bus interno del microcontrolador cuando se lee el registro de datos del puerto. Puede verse que el dato leído es directamente el estado del pin de entrada

1.6 Diagrama lógico

Diagrama lógico para los microcontroladores PIC16C54/56

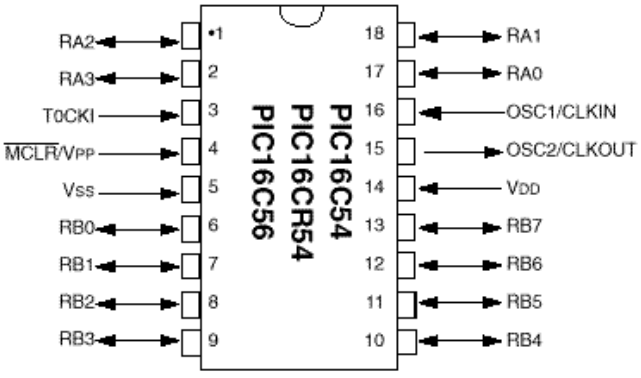
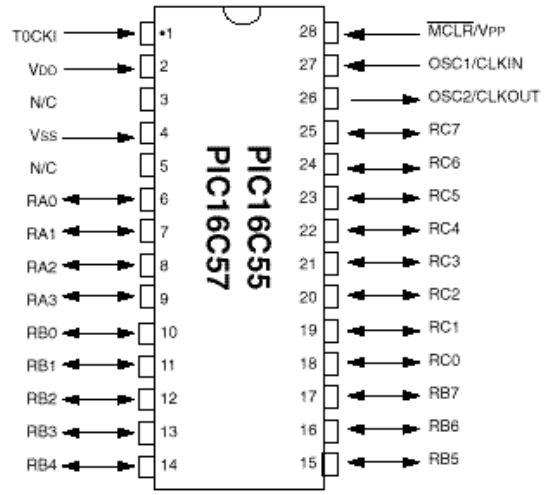


Diagrama lógico para los microcontroladores PIC16C55/57



1.7 Temporizador/Contador (RTCC) Este dispositivo, llamado Real Time Clock / Counter, es básicamente un contador de 8 bits, constituido por el registro operacional RTCC que se encuentra en la posición 01 de la memoria de datos. Este registro puede usarse para contar eventos externos por medio de un pin de entrada especial (modo contador) o para contar pulsos internos de reloj de frecuencia constante (modo timer). Además, en cualquiera de los dos modos, se puede insertar un prescaler, es decir un divisor de frecuencia programable que puede dividir por 2, 4, 8, 16, 32, 64, 128 o 256. Este divisor puede ser utilizado alternativamente como prescaler del RTCC o como postscaler del Watch Dog Timer, según se lo programe.

Para su programación se dispone de dos registros: el RTCC ya mencionado y el registro OPTION. Este último no es accesible como memoria de datos, no se lo puede leer de ninguna manera, y solo se lo puede escribir con la instrucción especial OPTION (familia PIC16C5X). Este registro contiene los bits necesarios para seleccionar modo contador o modo timer, flanco de conteo en modo contador, prescaler para RTCC o para WDT y constante de división del prescaler, según el siguiente esquema:

U-0	U-0	W-1	W-1	W-1	W-1	W-1	W-1
—	—	T0CS	T0SE	PSA	PS2	PS1	PS0
bit7	6	5	4	3	2	1	bit0

bit 7-6: **Unimplemented.**

bit 5: **T0CS**: Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

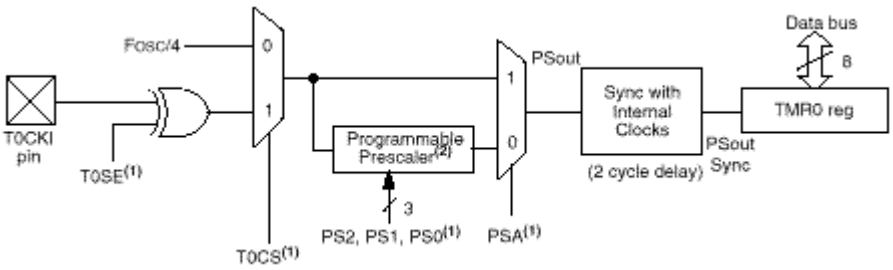
bit 4: **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

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

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

Bit Value	Timer0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

A continuación se muestra un circuito equivalente del RTCC (TMR0) y el prescaler.



En el esquema se puede observar claramente como operan los bits de configuración T0SE, T0CS y PSA, en cualquiera de sus combinaciones. Se observa además que en la entrada del contador RTCC hay un circuito de sincronización que introduce una demora de dos ciclos del clock de instrucciones (Fosc. / 4). Al escribir sobre el RTCC automáticamente se resetea este circuito, por lo tanto solo se incrementara dos ciclos después.

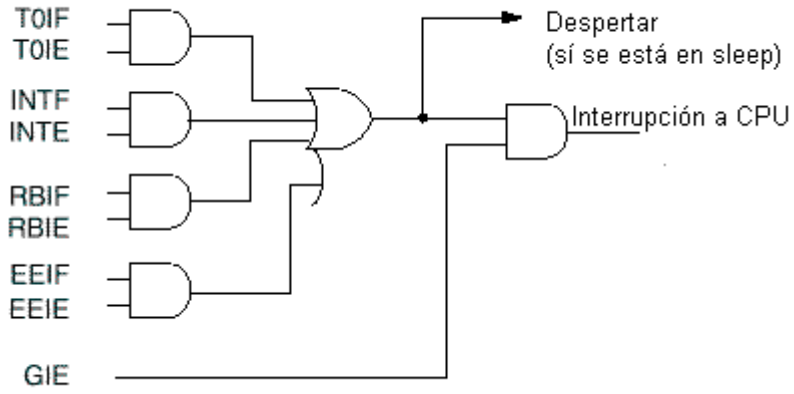
El prescaler es un contador asincrónico de 8 bits más un multiplexor 8 a 1 comandado por los bits PS0 a PS2, que permite seleccionar como salida a cualquiera de los bits del contador. Al escribir sobre el RTCC, si este esta programado para operar con prescaler (PSA=0), se borra automáticamente el prescaler. Las instrucciones CLRWDT y SLEEP borran el prescaler, si este esta programado para operar como postscaler del watch dog timer (PSA=1).

1.8 Interrupciones Los 16CXX agregan la posibilidad de contar con sistema de interrupciones. Este sistema consiste en un mecanismo por el cual un evento interno o externo, asincrónico respecto del programa, puede interrumpir la ejecución de éste produciendo automáticamente un salto a una subrutina de atención, de manera que pueda atender inmediatamente el evento, y retomar luego la ejecución del programa exactamente en donde estaba al momento de ser interrumpido. Este mecanismo es muy útil por ejemplo para el manejo de timers o rutinas que deben repetirse periódicamente (refresh de display, antirebote de teclado, etc.), detección de pulsos externos, recepción de datos, etc.

Existen de tres a doce eventos que pueden generar interrupciones en los PIC16CXX existentes hasta el momento, pero nada impide que puedan agregarse más en versiones futuras.

Funcionamiento
En los 16CXX las interrupciones se comportan casi exactamente igual que las subrutinas. Desde el punto de vista del control del programa, al producirse una interrupción se produce el mismo efecto que ocurriría si el programa tuviese un CALL 0004h en el punto en que se produjo la interrupción. En uno de los registros de control del sistema de interrupciones existe un bit de habilitación general de interrupciones GIE, que debe ser programado en 1 para que las interrupciones puedan actuar. Al producirse una interrupción, este bit se borra automáticamente para evitar nuevas interrupciones. La instrucción RETFIE que se utiliza al final de la rutina de interrupción, es idéntica a un retorno de subrutina, salvo que además coloca en uno automáticamente el bit GIE volviendo a habilitar las interrupciones. Dentro de la rutina de interrupción, el programa deberá probar el estado de los flags de interrupción de cada una de las fuentes

habilitadas, para detectar cual fue la que causo la interrupción y así decidir que acción tomar.



Lógica de interrupciones para los µcontroladores PIC16F8X

Fuentes
La señal que produce la interrupción es en realidad una sola, que resulta de la combinación de todas las fuentes posibles y de los bits de habilitación. Existen dos grupos de fuentes, unas que se habilitan con solo colocar en uno el bit GIE, y otras que además necesitan que este puesto a uno el bit PEIE. En algunas versiones de los 16CXX solo existe el primer grupo. Además, cada fuente de interrupciones tiene su respectivo bit de habilitación individual.

Las fuentes de interrupción varían con cada versión, y pueden ser por ejemplo:

· Interrupción externa por pin RB0/INT
· Desborde del Timer 0 o RTCC
· Cambio en el estado de los bits 4 a 7 del puerto B
· Desborde del timer 1
· Desborde del timer 2
· Interrupción del capture/compare 1
· Interrupción del capture/compare 2
· transmisión o recepción de un caracter por la interface serie sincrónica
· transmisión o recepción de un caracter por la interface serie asincrónica
· Fin de conversión A/D
· Lectura/escritura del puerto paralelo de comunicación con otros microprocesadores
· Escritura de EEPROM finalizada

Programa fuente:

El programa fuente esta compuesto por una sucesión de líneas de programa. Cada línea de programa esta compuesta por 4 campos separados por uno o más espacios o tabulaciones. Estos campos son:

[Etiqueta] Comando [Operando(s)] [;Comentario]

La etiqueta es opcional. El comando puede ser un mnemónico del conjunto de instrucciones. El operando esta asociado al comando, si no hay comando no hay operando, e inclusive algunos comandos no llevan operando. El comentario es opcional para el compilador aunque es buena práctica considerarlo obligatorio para el programador.

La etiqueta, es el campo que empieza en la primer posición de la línea. No se pueden insertar espacios o tabulaciones antes de la etiqueta sino será considerado comando. Identifica la línea de programa haciendo que el compilador le asigne un valor automáticamente. Si se trata de una línea cuyo comando es una instrucción de programa del microcontrolador, se le asigna el valor de la dirección de memoria correspondiente a dicha instrucción (location counter). En otros casos se le asigna un valor de una constante, o la dirección de una variable, o será el nombre de una macroinstrucción, etc.

El comando puede ser un código mnemónico de instrucción del microcontrolador, o una directiva o pseudoinstrucción para el compilador. En el primer caso será directamente traducido a código de maquina, en el segundo caso será interpretado por el compilador y realizara alguna acción en tiempo de compilación como ser asignar un valor a una etiqueta, etc.

El campo de parámetros puede contener uno o más parámetros separados por comas. Los parámetros dependen de la instrucción o directiva. Pueden ser números o literales que representen constantes o direcciones.

El campo de comentario debe comenzar con un caracter punto y coma. No necesita tener espacios o tabulaciones separándolo del campo anterior, e incluso puede empezar en la primer posición de la línea. El compilador ignora todo el texto que contenga la línea después de un caracter punto y coma. De esta manera pueden incluirse líneas que contengan solo comentarios, y es muy buena práctica hacer uso y abuso de esta posibilidad para que los programas resulten autodocumentados.

2.1 Conjunto de instrucciones El conjunto de instrucciones de los microprocesadores PIC 16C5X consiste en un pequeño repertorio de solo 33 instrucciones de 12 bits, que pueden ser agrupadas para su estudio en tres a cinco grupos. En este curso se ha optado por clasificarlas, desde el punto de vista del programador, en cinco categorías bien definidas de acuerdo con la función y el tipo de operandos involucrados. En primer lugar se agrupan las instrucciones que operan con bytes y que involucran algún registro de la memoria interna. En segundo lugar se analizaran las instrucciones que operan solo sobre el registro W y que permiten cargarle una constante implícita o incluida literalmente en la instrucción (literales). En tercer lugar se agrupan las instrucciones que operan sobre bits individuales de los registros de la memoria interna. En cuarto lugar se clasifican las instrucciones de control de flujo del programa, es decir las que permiten alterar la secuencia lineal de ejecución de las instrucciones. Por último se agrupan unas pocas instrucciones que llamaremos especiales, cuyas funciones o tipos de operandos son muy específicos y no encajan en ninguna de las clasificaciones anteriores.

Instrucciones de Byte que operan con Registros
Estas instrucciones pueden ser de simple o doble operando de origen. El primer operando de origen será siempre el registro seleccionado en la instrucción, el segundo, en caso de existir, será el registro W. El destino, es decir donde se guardara el resultado, será el registro seleccionado o el W, según se seleccione con un bit de la instrucción.

El formato genérico de estas instrucciones es el siguiente :

0	1	2	3	4	5	**6**	**7**	**8**	**9**	**10**	**11**
						d	**f**	**f**	**f**	**f**	**f**

Los bits 0 a 4 (5 bits), denominados "f" permiten seleccionar uno de 32 registros de la memoria interna. El bit 5, denominado "d", permite especificar el destino del resultado. Si d = 1 el resultado se guardara en el registro seleccionado. Si d = 0 el resultado se guardara en W. Los bits 6 a 11 identifican la instrucción especifica a realizar.

Las instrucciones siguientes son las tres operaciones lógicas de doble operando :

ANDWF f,d ;operación AND lógica, destino = W ☐ f
IORWF f,d ;operación OR lógica, destino = W ☐ f
XORWF f,d ;operación XOR lógica, destino = W ☐ f

Los nombres mnemónicos de estas instrucciones provienen de : AND W con F, Inclusive OR W con F y XOR W con F.
Las que siguen son las cuatro operaciones aritméticas y lógicas sencillas de simple operando :

MOVF f,d ;movimiento de datos, destino = f
COMF f,d ;complemento lógico, destino = NOT f
INCF f,d ;incremento aritmético, destino = f + 1
DECF f,d ;decremento aritmético, destino = f - 1

Los mnemónicos de estas instrucciones provienen de : MOVe File, COMplement File, INCrement File y DECrement File.
En las siete instrucciones anteriores el único bit afectado de la palabra de estado del procesador es el Z, que se pone en 1 si el resultado de la operación es 00000000, y se pone en 0 si el resultado tiene cualquier otro valor.

A continuación siguen las dos instrucciones de rotación de bits a través del CARRY :

RLF f,d ;rotación a la izquierda, destino = f ROT ☐
RRF f,d ;rotación a la derecha, destino = f ROT ☐

En estas operaciones (Rotate Left File y Rotate Right File) los bits son desplazados de cada posición a la siguiente, en sentido derecho o izquierdo. El desplazamiento es cerrado, formando un anillo, con el bit C (CARRY) de la palabra de estado.

En estas dos instrucciones, el único bit afectado de la palabra de estado del procesador es el bit C, que tomará el valor que tenia el bit 7 o el bit 0, según sea el sentido del desplazamiento.

Estas instrucciones son muy útiles para la manipulación de bits, y además para realizar operaciones aritméticas, ya que en numeración binaria, desplazar un número a la izquierda es equivalente a multiplicarlo por 2, y hacia la derecha, a dividirlo por 2.

La instrucción siguiente realiza el intercambio de posiciones entre los cuatro bits menos significativos y los cuatro más significativos (nibble bajo y nibble alto).

SWAPF f,d ;intercambia nibbles, destino = SWAP f

Esta instrucción (SWAP File) no afecta ninguno de los bits de la palabra de estado del procesador.

Esta instrucción es muy útil para el manipuleo de números BCD empaquetados, en los que en un solo byte se guardan dos dígitos BCD (uno en cada nibble).

Las dos operaciones que siguen son la suma y la resta aritméticas :

ADDWF f,d ;suma aritmética, destino = f + W
SUBWF f,d ;resta aritmética, destino = f - W

Estas operaciones (ADD W a F y SUBstract W de F) afectan a los tres bits de estado C, DC y Z.
El bit Z se pone en 1 si el resultado de la operación es 00000000, y se pone en 0 si el resultado tiene cualquier otro valor.

La suma se realiza en aritmética binaria pura sin signo. Si hay un acarreo del bit 7, es decir que el resultado es mayor que 255, el bit C (carry) resulta 1, en caso contrario resulta 0. Si hay un acarreo del bit 3, es decir que la suma de las dos mitades (nibbles) menos significativas (bits 0 a 3) resulta mayor que 15, se pone en 1 el bit DC (digit carry), en caso contrario se pone en 0.

Ejemplos :

```
      1010 0010                1101 0000
    + 0100 1111 C DC Z       + 0110 1111 C DC Z
      1111 0001 0  1  0        0011 1111 1  0  0
```

La resta se realiza sumando, en binario puro sin signo, el registro f más el complemento a dos de W (el complemento a 1, o complemento lógico, más 1)

Ejemplos :

```
 f □             0100 0100               0010 1000
W□             - 0010 1000  C DC Z     - 0100 0100  C DC Z
                 0001 1100  1  0  0      1110 0100  0  1  0
```

□□
□
equivalente a :

```
       f □       0100 0100               0010 1000
cmp.2 W □      + 1101 1000 C DC Z      + 1011 1100 C DC Z
                 0001 1100 1  0  0       1110 0100 0  1  0
```

Los bits de estado C y DC toman el valor normal correspondiente a la suma de f con el complemento a 2 de W. De esta manera el significado para la operación de resta resulta

invertido, es decir que C (carry) es 1 si no hubo desborde en la resta, o dicho de otra manera, si el contenido de W es menor que el de f. El bit DC se comporta de manera similar, es decir que DC es 1 si no hubo desborde en la mitad menos significativa, lo que equivale a decir que el nibble bajo del contenido de W es menor que el del registro f.

Las instrucciones que siguen son de simple operando, pero son casos especiales ya que el destino es siempre el registro seleccionado :

CLRF f ;borrado de contenido, f = 0
MOVWF f ;copia contenido W □ f, f = W

La instrucción CLRF (CLeaR File) afecta solo al bit Z que resulta siempre 0.
La instrucción MOVWF (MOVe W a F) no afecta ningún bit de la palabra de estado.

Instrucciones de Byte que operan sobre W y Literales
Estas instrucciones se refieren todas al registro W, es decir que uno de los operandos de origen y el operando de destino son siempre el registro W. En las instrucciones de este grupo que tienen un segundo operando de origen, este es siempre una constante de programa literalmente incluida en la instrucción, llamada constante literal o simplemente literal.

El formato genérico de estas instrucciones es el siguiente :

0	1	2	3	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
				k	**k**	**k**	**k**	**k**	**k**	**k**	**k**

Los bits 0 a 7 especifican la constante literal de 8 bits que se utilizara en la operación.

Las tres instrucciones que siguen son las operaciones lógicas tradicionales, similares a las que ya vimos anteriormente, pero realizadas entre una constante de programa y el registro W :

IORLW k ; operación OR lógica, W = W □ k
ANDLW k ; operación AND lógica, W = W □ k
XORLW k ; operación XOR lógica, W = W □ k

En estas tres instrucciones (Inclusive OR Literal W, AND Literal W y XOR Literal W) el único bit afectado de la palabra de estado del procesador es el Z, que se pone en 1 si el resultado de la operación es 00000000, y se pone en 0 si el resultado tiene cualquier otro valor.

La instrucción que sigue sirve para cargar una constante de programa en el registro W :

MOVLW k ;carga constante en W, W = K

Esta (MOVe Literal W) instrucción no afecta ninguno de los bits de estado del procesador.

La instrucción que sigue (CLeaR W) no correspondería incluirla en este grupo, y pertenece en realidad al primero, el de las instrucciones que operan sobre registros, ya que se trata

de un caso especial de la instrucción CLRF, con destino W, y f = 0. La incluimos aquí porque como se le ha asignado un mnemónico particular referido específicamente al registro W, creemos que, desde el punto de vista del programador, es más útil verla dentro del grupo de instrucciones referidas a W.

CLRW ;borra el contenido de W, W = 0

Al igual que en la instrucción CLRF, el único bit de estado afectado es el Z que resulta 1.

Instrucciones de Bit
El formato genérico de estas instrucciones es el siguiente :

0	1	2	3	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
				b	**b**	**b**	**f**	**f**	**f**	**f**	**f**

Los bits 0 a 4 (5 bits), denominados "f", permiten seleccionar uno de 32 registros de la memoria interna. Los bits 5 a 7, denominados "b", permiten especificar el numero de bit (0 a 7) sobre el que se operara. Estas instrucciones operan solamente sobre el bit especificado, el resto de los bits del registro no son alterados. Estas instrucciones no tienen especificación de destino, ya que el mismo es siempre el registro seleccionado.

BCF f,b ;borra el bit b de f ;bit f(b) = 0
BSF f,b ;coloca en uno el bit b de f ;bit f(b) = 1

Estas instrucciones (Bit Clear File y Bit Set File) no afectan ningún bit de la palabra de estado del procesador.

Instrucciones de Control

GOTO k ;salto a la posición k (9 bits) del programa

Esta es la típica instrucción de salto incondicional a cualquier posición de la memoria de programa (que en la mayoría de los microprocesadores convencionales se llama JUMP). La constante literal k es la dirección de destino del salto, es decir la nueva dirección de memoria de programa a partir de la cual comenzarán a leerse las instrucciones después de ejecutar la instrucción GOTO. Esta instrucción simplemente carga la constante k en el registro PC (contador de programa). La única complicación de esta instrucción es que la constante k es de solo 9 bits, mientras que el registro PC es de 11 bits, ya que en el 16C57 debe permitir direccionar una memoria de programa de 2 K. Los dos bits faltantes, bit 9 y 10 del PC, son tomados respectivamente de los bits de selección de página PA0 y PA1 de la palabra de estado. Este comportamiento particular hace que la memoria de programa aparezca como dividida en paginas de 512 posiciones como se vera más adelante. El programador debe tener en cuenta que antes de ejecutar una instrucción GOTO es posible que haya que programar los bits PA0 y PA1.

La que sigue es la instrucción de llamado a subrutina:

CALL k ;salto a la subrutina en la posición k (8 bits)

Su comportamiento es muy similar al de la instrucción GOTO, salvo que además de saltar guarda en el stack la dirección de retorno de la subrutina (para la instrucción RETLW). Esto lo hace simplemente guardando en el stack una copia del PC incrementado, antes de que el mismo sea cargado con la nueva dirección k. La única diferencia con la instrucción GOTO respecto de la forma en la que se realiza el salto, es que en la instrucción CALL la constante k tiene solo 8 bits en vez de 9. En este caso también se utilizan PA0 y PA1 para cargar los bits 9 y 10 del PC, pero además el bit 8 del PC es cargado siempre con 0. Esto hace que los saltos a subrutina solo puedan realizarse a posiciones que estén en las primeras mitades de las paginas mencionadas. El programador debe tener en cuenta este comportamiento y asegurarse de ubicar las posiciones de inicio de las subrutinas en las primeras mitades de las paginas.

La instrucción que aparece a continuación es la de retorno de subrutina:

RETLW k ;retorno de subrutina con constante k, W = k

Esta (RETurn con Literal in W) instrucción produce el retorno de subrutina con una constante literal k en el registro W. La operación que realiza consiste simplemente en sacar del stack un valor y cargarlo en el PC. Ese valor es el PC incrementado antes de realizar el salto, de la ultima instrucción CALL ejecutada, por lo tanto es la dirección de la instrucción siguiente a dicho CALL.. Dado que el stack es de 11 bits, el valor cargado en el PC es una dirección completa, y por lo tanto se puede retornar a cualquier posición de la memoria de programa, sin importar como estén los bits de selección de pagina. Esta instrucción además carga siempre una constante literal en el registro W. Ya que esta es la única instrucción de retorno de subrutina de los PIC16C5X, no hay en estos microprocesadores forma de retornar de una subrutina sin alterar el registro W. Por otro lado, y con una metodología especial de programación, un conjunto de sucesivas instrucciones RETLW puede ser usado como una tabla de valores constantes incluida en el programa (Ej. : tablas BCD/7 seg., hexa/ASCII, etc.).

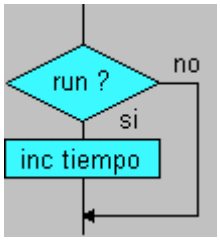
A continuación se presentan las dos únicas instrucciones de "salteo" (skip) condicional. Estas instrucciones son los únicos medios para implementar bifurcaciones condicionales en un programa. Son muy generales y muy poderosas ya que permiten al programa tomar decisiones en función de cualquier bit de cualquier posición de la memoria interna de datos, y eso incluye a los registros de periféricos, los puertos de entrada/salida e incluso la palabra de estado del procesador. Estas dos instrucciones reemplazan y superan a todo el conjunto de instrucciones de salto condicional que poseen los microprocesadores sencillos convencionales (salto por cero, por no cero, por carry, etc.).

BTFSC f,b ;salteo si bit = 0, bit = f(0) □ saltea
BTFSS f,b ;salteo si bit = 1, bit = f(1) □ saltea

BTFSC (Bit Test File and Skip if Clear) saltea la próxima instrucción si el bit b del registro f es cero. La instrucción BTFSS (Bit Test File and Skip if Set) saltea si el bit es 1. Estas instrucciones pueden usarse para realizar o no una acción según sea el estado de un bit, o, en combinación con GOTO, para realizar una bifurcación condicional.

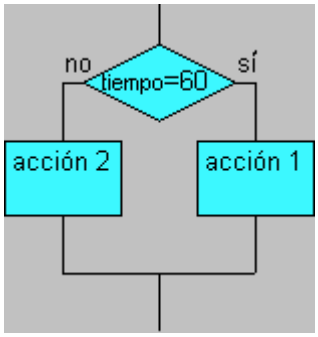
Ejemplo 1 :

```
- - - - - - - - - - - - -
- - - - - - - - - - - - -
btfsc flags,run    ;sí ha arrancado el reloj
incf   tiempo      ;incremento contador de tiempo
- - - - - - - - - - - - -
- - - - - - - - - - - - -
```



Ejemplo 2 :

```
    - - - - - - - - - - - - -
    - - - - - - - - - - - - -
    movf tiempo,w     ;testeo por tiempo = 60
    xorlw 60
    btfss STATUS,Z
    goto acc_2          ;salto si tiempo <> 60
     - - - - - - - - - - - - -  ;acción 1
     - - - - - - - - - - - - -
     - - - - - - - - - - - - -
     goto acc_fin
acc_2
      - - - - - - - - - - - - -  ;acción 2
      - - - - - - - - - - - - -
      - - - - - - - - - - - - -
acc_fin   ;acá se unen los caminos
```



Las instrucciones que siguen son casos especiales de las de incremento y decremento vistas anteriormente. Estas instrucciones podrían categorizarse dentro del grupo de instrucciones orientadas a byte sobre registros (primer grupo), ya que efectivamente operan sobre los mismos, y el formato del código de la instrucción responde al de ese grupo, pero, a diferencia de las otras, pueden además alterar el flujo lineal del programa y por eso se les incluyó en este grupo.

DECFSZ f,d ;decrementa y saltea sí 0, destino= f - 1, = 0 □ saltea
INCFSZ f,d ;incrementa y saltea sí 0, destino= f + 1, = 0 □ saltea

Estas dos instrucciones (DECrement File and Skip if Zero, e INCrement File and Skip if Zero) se comportan de manera similar a DECF e INCF, salvo que no afectan a ningún bit de la palabra de estado. Una vez realizado el incremento o decremento, si el resultado es 00000000, el microprocesador salteara la próxima instrucción del programa. Estas instrucciones se utilizan generalmente en combinación con una instrucción de salto

(GOTO), para el diseño de ciclos o lazos (loops) de instrucciones que deben repetirse una cantidad determinada de veces.

Ejemplo:

```
        clrf 10                 ;pongo cero en la posición 10 de la memoria interna
loop                            ;lo que sigue se ejecutará 256 veces
 .....
 .....
 .....

        incfsz 10,1             ;incremento la posición 10 hasta que llegue a 0
        goto loop               ;si no llego a cero voy a repetir la secuencia
                                 ;cuando llegue a cero salteo el goto
 ..... ;y sigue la continuación del programa
 .....
 .....
```

Instrucciones Especiales

En este grupo se reunieron las instrucciones que controlan funciones específicas del microprocesador o que actúan sobre registros especiales no direccionados como memoria interna normal.

La instrucción que sigue es la típica NO OPERATION, existente en casi todos los microprocesadores.

```
 NOP        ;no hace nada, consume tiempo
```

Esta instrucción solo sirve para introducir una demora en el programa, equivalente al tiempo de ejecución de una instrucción. No afecta ningún bit de la palabra de estado.

La siguiente es una instrucción especifica de control de los puertos de entrada/salida.

```
 TRIS f ;carga el tristate control, TRISf = W
```

Esta instrucción (TRIState) carga el registro de control de los buffers tristate de un puerto de entrada salida (data dirección register), con el valor contenido en W. El parámetro f debe ser la dirección de memoria interna del puerto, aunque el valor W no será cargado en el puerto sino en el registro de tristate del mismo. Los valores validos para f son 4 y 5 en los 16C54/56 y 4, 5 y 6 en los 16C55/57. Esta instrucción no afecta ningún bit de la palabra de estado.

La siguiente instrucción sirve para programar el registro OPTION que controla el RTCC y prescaler

```
 OPTION  ;carga el registro OPTION, OPTION = W
```

El registro OPTION no es accesible como memoria interna y solo se lo puede programar con esta instrucción. Esta instrucción no afecta ningún bit de la palabra de estado.

La instrucción que sigue borra el contador del watch dog timer. Este registro tampoco esta accesible como memoria, y esta es la única instrucción que lo modifica.

CLRWDT ;borra el watch dog timer, WDT = 0

Esta instrucción, además, coloca en uno los bits PD (power down) y TO (time-out) de la palabra de estado.

La siguiente es una instrucción especial de control del microcontrolador que lo pone en el modo power down. En este modo el microprocesador se detiene, el oscilador se apaga, los registros y puertos conservan su estado, y el consumo se reduce al mínimo. La única forma de salir de este estado es por medio de un reset o por time-out del watch dog timer.

SLEEP ;coloca el µC en modo sleep, WDT = 0

Esta instrucción, además, borra el bit PD (power down) y setea el bit TO (time-out) de la palabra de estado.

Resumen de instrucciones (clasificación según el fabricante en tres grupos):

Instrucciones orientadas a byte:

Mnemonic, Operands		Description	Cycles	12-Bit Opcode MSb		LSb	Status Affected
ADDWF	f,d	Add W and f	1	0001	11df	ffff	C,DC,Z
ANDWF	f,d	AND W with f	1	0001	01df	ffff	Z
CLRF	f	Clear f	1	0000	011f	ffff	Z
CLRW	–	Clear W	1	0000	0100	0000	Z
COMF	f, d	Complement f	1	0010	01df	ffff	Z
DECF	f, d	Decrement f	1	0000	11df	ffff	Z
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	0010	11df	ffff	None
INCF	f, d	Increment f	1	0010	10df	ffff	Z
INCFSZ	f, d	Increment f, Skip if 0	1(2)	0011	11df	ffff	None
IORWF	f, d	Inclusive OR W with f	1	0001	00df	ffff	Z
MOVF	f, d	Move f	1	0010	00df	ffff	Z
MOVWF	f	Move W to f	1	0000	001f	ffff	None
NOP	–	No Operation	1	0000	0000	0000	None
RLF	f, d	Rotate left f through Carry	1	0011	01df	ffff	C
RRF	f, d	Rotate right f through Carry	1	0011	00df	ffff	C
SUBWF	f, d	Subtract W from f	1	0000	10df	ffff	C,DC,Z
SWAPF	f, d	Swap f	1	0011	10df	ffff	None
XORWF	f, d	Exclusive OR W with f	1	0001	10df	ffff	Z

Instrucciones orientadas a bit:

Mnemonic, Operands		Description	Cycles	12-Bit Opcode MSb		LSb	Status Affected
BCF	f, b	Bit Clear f	1	0100	bbbf	ffff	None
BSF	f, b	Bit Set f	1	0101	bbbf	ffff	None
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	0110	bbbf	ffff	None
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	0111	bbbf	ffff	None

Instrucciones orientadas a literal y control:

Mnemonic, Operands		Description	Cycles	12-Bit Opcode MSb		LSb	Status Affected
ANDLW	k	AND literal with W	1	1110	kkkk	kkkk	Z
CALL	k	Call subroutine	2	1001	kkkk	kkkk	None
CLRWDT	k	Clear Watchdog Timer	1	0000	0000	0100	$\overline{TO}$, $\overline{PD}$
GOTO	k	Unconditional branch	2	101k	kkkk	kkkk	None
IORLW	k	Inclusive OR Literal with W	1	1101	kkkk	kkkk	Z
MOVLW	k	Move Literal to W	1	1100	kkkk	kkkk	None
OPTION	k	Load OPTION register	1	0000	0000	0010	None
RETLW	k	Return, place Literal in W	2	1000	kkkk	kkkk	None
SLEEP	–	Go into standby mode	1	0000	0000	0011	$\overline{TO}$, $\overline{PD}$
TRIS	f	Load TRIS register	1	0000	0000	0fff	None
XORLW	k	Exclusive OR Literal to W	1	1111	kkkk	kkkk	Z

En esta tabla de resumen del conjunto de instrucciones se pueden observar los mnemónicos, la explicación, el número de ciclos, el código de máquina y los bits afectados del registro STATUS para cada una de las instrucciones.

2.2 Modos de direccionamiento

Direccionamiento de la memoria de datos (RAM)

La memoria interna se direcciona en forma directa por medio de los 5 bits "f" contenidos en las instrucciones que operan sobre registros. De esta manera se puede direccionar cualquier posición desde la 00 a la 1F. Como se vió en el capítulo correspondiente a los mapas de memoria, las direcciones 10 a 1F corresponden a los bancos de registros, por lo tanto, en los microcontroladores que tengan más de un banco, antes de acceder a alguna variable que se encuentre en esta zona, el programador deberá asegurarse de haber programado los bits de selección de banco en el registro FSR. Los registros especiales y de uso general de la posición 00 a la 0f están presentes en todos los PIC16C5X, al igual que el banco 0 de registros. Los bancos 1, 2 y 3 de registros están presentes solo en el 16C57.

El registro FSR, además de servir para seleccionar el banco activo, sirve como puntero para direccionamiento indirecto. La posición 00 del mapa de RAM es la llamada dirección indirecta. Sí en cualquier instrucción se opera con la dirección 00, en realidad se estará operando con la dirección a donde apunte el contenido del FSR. Por ejemplo si el FSR contiene el valor 14, una instrucción que opere sobre la dirección 0, operara en realidad sobre la dirección 14. Se puede decir en este ejemplo que la posición 14 de memoria fue direccionada en forma indirecta a través del puntero FSR.

Ejemplo :

; Esta porción de programa borra 5 posiciones de memoria a partir de la dirección 12

```
FSR  equ  04                ;(definición al comienzo del programa)
 .....................................
 .....................................

        movlw  5            ;prepara para repetir 5 veces
        movwf  08           ;(el registro 08 es el contador del loop)
        movlw  12h          ;apunta a la dirección 12h
```

```
        movwf  FSR  ;
loop:
        clrf  0              ;borra una posición de memoria
        incf  FSR            ;apunta a la siguiente
        decfsz  08           ;si todavía no borra todas
        goto  loop           ;sige borrando
 .....
 .....
```

El direccionamiento indirecto es muy útil para el procesamiento de posiciones consecutivas de memoria, como en el ejemplo, o para el direccionamiento de datos en subrutinas.

Direccionamiento de la memoria de programa (EPROM, OTP)

La instrucción GOTO dispone solo de 9 bits en el código de operación para especificar la dirección de destino del salto. Al ejecutar una instrucción GOTO el microprocesador toma los dos bits que restan para completar la dirección de 11 bits, de los bits 5 y 6 de la palabra de estado. Estos últimos son llamados bits de selección de página (PA0 y PA1). El programador deberá asegurarse de que estos dos bits tengan el valor correcto antes de toda instrucción GOTO. Ver figura 2.2.1

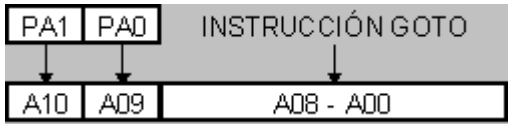


FIG 2.2.1 Direccionamiento directo con instrucción GOTO

Deberá tenerse en cuenta además que es posible avanzar de una página a otra en forma automática cuando el PC se incrementa. Esto ocurre si el programa empieza en una página y sigue en la siguiente. Sin embargo, al incrementarse el PC desde la última posición de una página a la primera de la siguiente, **los bits PA0 y PA1 no se modifican**, y por lo tanto sí se ejecuta una instrucción GOTO, CALL o alguna que actúe sobre el PC, esta producirá un salto a la página anterior, a menos que el programador tenga la precaución de actualizar el valor de dichos bits. Por este motivo es conveniente dividir el programa en módulos o rutinas que estén confinados a una página.

En el caso de la instrucción CALL, el direccionamiento se complica un poco más, ya que la misma solo dispone de 8 bits para especificar la dirección de destino salto. En este caso también se utilizan los mismos bits de selección de página para completar los bits décimo y decimoprimero de la dirección, pero falta el noveno bit. En estas instrucciones este bit se carga siempre con 0, lo que implica que solo se pueden realizar saltos a subrutina a las mitades inferiores de cada página. En este caso también el programador deberá asegurarse que el estado de los bits PA0 y PA1 sea el correcto al momento de ejecutarse la instrucción. Ver figura 2.2.2

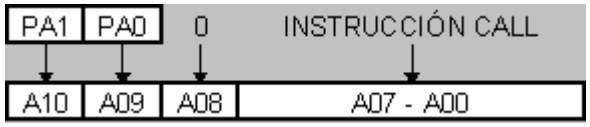


FIG. 2.2.2 Direccionamiento directo con instrucción CALL

Las instrucciones que operan sobre el PC como registro y alteran su contenido provocando un salto, responden a un mecanismo muy similar al de las instrucciones CALL para la formación de la dirección de destino. En este caso los bits 0 a 7 son el resultado de la instrucción, el bit 8 es 0 y los bits restantes se toman de PA0 y PA1.

Este mecanismo se llama paginado, y a pesar de que representa una complicación bastante molesta para el programador, resulta muy útil ya que permite ampliar la capacidad de direccionamiento de memoria de programa para las instrucciones de salto

2.3 Herramientas de desarrollo

UNA MIRADA RAPIDA AL MPLAB

Qué es el MPLAB ?

EL MPLAB es un "Entorno de Desarrollo Integrado " (Integrated Development Environment,
IDE) que corre en "Windows ", mediante el cual Usted puede desarrollar aplicaciones para los microcontroladores de las familias PIC 16/17.
EL MPLAB le permite a Usted escribir, depurar y optimizar los programas (firmware) de sus diseños con PIC 16/17. EL MPLAB incluye un editor de texto, un simulador y un organizador de proyectos. Además, el MPLAB soporta el emulador PICMASTER y a otras herramientas de desarrollo de Microchip como el PICSTART - Plus.

De que forma le ayuda el MPLAB ?

Con el MPLAB Usted puede:

- Depurar sus programas fuente.
- Detectar errores automáticamente en sus programas fuente para editarlos.
- Depurar los programas utilizando puntos de corte (breakpoints) mediante valores de los registros internos.
- Observar el flujo del programa con el simulador MPLAB -SIM, ó seguirlo en tiempo real utilizando el emulador PICMASTER.
- Realizar medidas de tiempo utilizando un cronómetro.
- Mirar variables en las ventanas de observación.
- Encontrar respuestas rápidas a sus preguntas, utilizando la Ayuda en línea del MPLAB.

LAS HERRAMIENTAS DEL MPLAB

El Organizador de Proyectos (Proyect Manager).

El organizador de proyectos (Proyect Manager) es parte fundamental de MPLAB. Sin crear un proyecto Usted no puede realizar depuración simbólica. Con el Organizador de Proyectos (Proyect manager) puede utilizar las siguientes operaciones:

- Crear un proyecto.

- Agregar un archivo de programa fuente de proyecto.
- Ensamblar o compilar programas fuente.
- Editar programas fuente.
- Reconstruir todos los archivos fuente, o compilar un solo archivo.
- Depurar su programa fuente.

Software ensamblador:

El software ensamblador que presenta Microchip viene en dos presentaciones, una, para entorno DOS llamado MPASM.EXE y la otra, para entorno Windows llamado MPASMWIN.EXE
Las dos presentaciones soportan a TODOS los microcontroladores de la familia PIC de Microchip.

El conjunto de instrucciones de los microcontroladores PIC es en esencia la base del lenguaje ensamblador soportado por este software.

Directivas de uso frecuente:
Son instrucciones para el compilador.

#DEFINE
ej. #define <nombre> [<valor a remplazar>]
explicación: declara una cadena de texto como substituto de otra

END
ej. end
explicación: indica fin de programa

EQU
ej. status equ 05
explicación: define una constante de ensamble

INCLUDE
ej. include <PIC16F84.h>
explicación: incluye en el programa un archivo con código fuente

ORG
ej. org 0x100
explicación: ensambla a partir de la dirección especificada

Para información más completa referirse a la guía rápida del MPASM.

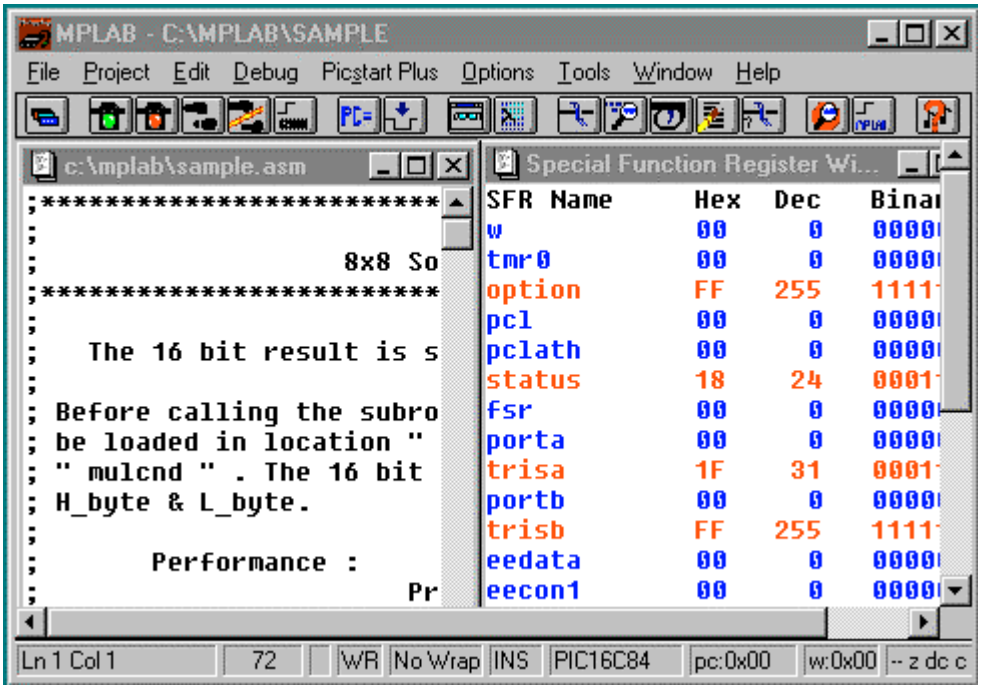
Una vez instalado adecuadamente el MPLAB, para realizar la simulación de un programa deben seguirse los siguientes pasos:

Edite en un archivo de texto el siguiente programa:

```
;ejemplo:
status  equ    0x03      ;hace equivalencia entre el símbolo status indicándolo como 3 en
hexadecimal
Cont   equ    0x20
F       equ    1
        org 0              ;indica posición de memoria desde donde se ensambla
Inicio
        movlw  0x0F    ;carga de w con el valor constante 15 (literal)
        movwf  Cont    ;el contenido de w se pasa al reg. CONT
Loop
        decfsz  Cont,F  ;decremento de Cont y elude siguiente si=0
        goto  Loop      ;salto incondicional a Loop
        goto  $          ;Salto incondicional aqui mismo
        end              ;Fin del código
```

Lista de pasos:

1. Haga doble click en el ícono correspondiente a MPLAB.
2. Crear el archivo fuente correspondiente (menú File...New Source).
3. Salve el archivo (con extensión .ASM) una vez terminada su edición (menú FILE...Save).
4. Debe a continuación crearse un nuevo proyecto (menú Project...New Project).
5. Cuando aparezca la ventana de New Project editar las cajas de texto:
 Project path and Name y Development Mode, hacer click en <OK>.
6. En la siguiente ventana Edit Project, hacer click en la sección Non-project files sobre el nombre del archivo fuente realizado en los pasos 2 y 3.
7. Haga click en el botón <=add y luego de que éste aparezca en la sección Project Files haga click sobre el botón <OK>.
8. Salvar el proyecto (en el menú Project...Save project).
9. Realizar la "construcción de todo el proyecto" (menú Project...Build All).
10. En esta etapa se realiza en forma automática el ensamble de nuestro programa fuente y el vaciado de éste en memoria de simulación. El proceso de ensamble generará un archivo de errores en caso de que estos existan, sí es así deben corregirse directamente sobre el archivo fuente, salvar las correcciones y reconstruir el proyecto (menú Project...Build All). <<<En esta etapa del proceso ya se tiene el entorno listo para la simulación>>>



Vista típica del entorno MPLAB

Como en la mayoría de las aplicaciones Windows la pantalla se divide en varias secciones:

1. Barra de título: Se observa el nombre del proyecto
2. Barra de menus: Acceso a las diferentes opciones del entorno
3. Barra de herramientas: Cada ícono ejecuta las acciones correspondientes
4. Barra de estados: Indica el estado del entorno y sus ventanas

Simulación:

1. Resetear el procesador (menú Debug...Run...Reset) ó con F6 ó con el ícono correspondiente en la barra de herramientas.

2. Crear una nueva ventana donde se incluyan las variables que queremos tener en cuenta (Window...New Watch Window)

3. Empezar a correr paso a paso el programa haciendo el seguimiento detallado de todos y cada uno de los pasos (menú Debug...Run...Step) ó con la tecla F7 ó con el ícono correspondiente en la barra de herramientas.

El proceso de simulación nos permite detectar y corregir problemas de lógica, problemas de situaciones que no hayamos tenido en cuenta que son errores que no pueden ser detectados en el momento del ensamble del programa.

Nota: El programa MPLAB puede obtenerse en forma gratuita de la dirección:
http://www.microchip.com/10/Tools/mTools/MPLAB/index.htm

3.1 Subrutinas y llamados

IMPORTANCIA DE LAS RUTINAS (*)

La mayoría de los microcontroladores incluyen en su repertorio de instrucciones algunas que permiten saltar a una rutina y, cuando se complementa su ejecución, retornar al programa principal

El empleo de subrutinas aporta muchas ventajas entre las que se destacan las siguientes:

1. Se pueden escribir como subrutinas secciones de código y ser empleadas en muchos programas (por ejemplo, la subrutina de exploración de un teclado).

2. Dan a los programas un carácter modular, es decir, se pueden codificar diferentes módulos para usarlos en cualquier programa.

3. Se reduce notablemente el tiempo de programación, la detección de errores, usando repetidamente una subrutina.

4. El código es más fácil de interpretar, dado que las instrucciones de las subrutinas no aparecen en el programa principal. Solo figuran las llamadas CALLs.

LAS INSTRUCCIONES CALL Y RETURN (*)

La instrucción CALL (llamada la subrutina) consigue que la ejecución del programa continúe en la dirección donde se encuentra la subrutina a la que hace referencia. Es similar a GOTO pero coloca en la pila la dirección de la siguiente instrucción que se debe ejecutar después de la CALL.

La subrutina finaliza con la instrucción RETURN (Retorno de la subrutina) que retoma la dirección guardada en le pila y la coloca en el contador del programa PC continuando el flujo de control con la instrucción que sigue a la CALL.

En la familia PIC de gama media la pila tiene ocho niveles de memoria del tipo FIFO (primero en entrar, último en salir). Si se produce la llamada a una subrutina durante la ejecución de otra subrutina, la dirección de retorno de esta segunda es colocada en la cima de la pila sobre la dirección anterior. Esta segunda dirección es la primera en salir de la pila mediante la instrucción RETURN.

Con la pila de ocho niveles, una subrutina puede llamar a otra y ésta, a su vez, llamar a otra hasta un máximo de ocho. La gama baja sólo puede realizar dos llamadas de este tipo al poseer una pila de sólo dos niveles.

Las subrutinas deben colocarse al comienzo de las páginas debido a que el bit 8 del contador del programa es puesto a 0 por la instrucción CALL (o por cualquier instrucción que modifica el PC). Las subrutinas deben colocarse en la mitad inicial de las páginas (las 256 palabras).

** Tomado de: Microcontroladores PIC, la solución en un chip, Angulo y otros, Sección 5.1*

3.2 Consulta a tablas En muchas ocasiones es necesario para un programador efectuar una coincidencia entre alguna cantidad de valores conocidos y un número desconocido que se tiene como índice, por ejemplo, basados en el contenido de una posición de memoria RAM (índice) se puede obtener de una serie consecutiva de datos almacenados

en memoria de programa (a estos datos "conocidos" almacenados se le denomina tabla), el dato desplazado ***n*** posiciones adelante del comienzo de esta tabla, este número ***n*** corresponde al contenido de la posición de memoria RAM ó índice.

Programa ejemplo:

```
offset    equ    0Ch    ;posición de memoria RAM
w         equ    0      ;destino W
f         equ    1      ;destino F
.................
.................
.................
   movf     offset,w  ;tomamos a W el número n utilizado como índice
   call     tabla     ;posición en donde se encuentra la serie de datos
                      ;en este sitio luego del retorno de la subrutina se tiene en W el dato
leído de la tabla
.................
.................
.................
tabla
   addwf    PCL,f   ;se suma al PC el contenido de W obteniendo como resultado un salto
indexado
   retlw      30h    ;sí el contenido de W sumado al PCL es 0 se retorna en esta posición,
W=30h
   retlw      31h    ;sí el contenido de W sumado al PCL es 1 se retorna en esta posición,
W=31h
   retlw      32h    ;sí el contenido de W sumado al PCL es 2 se retorna en esta posición,
W=32h
   retlw      33h    ;sí el contenido de W sumado al PCL es 3 se retorna en esta posición,
W=33h
   retlw      34h    ;sí el contenido de W sumado al PCL es 4 se retorna en esta posición,
W=34h
   retlw      35h    ;sí el contenido de W sumado al PCL es 5 se retorna en esta posición,
W=35h
   .                 ;...
   .
   .
```

Finalmente y luego de observar el ejemplo anterior, podemos anotar que antes de hacer el llamado a la subrutina **tabla**, se debe cargar en el registro de trabajo W el valor del índice y una vez se retorne de dicha subrutina, es en este mismo registro de trabajo en donde se obtiene el resultado de la consulta a la tabla (vemos que la sucesión de instrucciones **retlw k** se encuentra en memoria de programa).

3.3 Conversión a ASCII

El conjunto de carácteres ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) es el código de representación en hexadecimal del alfabeto, los números del 0 al 9 y los principales símbolos de puntuación y algunos carácteres de control. Ver Tabla 3.3.1

Least Significant Character	Most Significant Character								
	Hex	0	1	2	3	4	5	6	7
	0	NUL	DLE	Space	0	@	P	`	p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	"	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	Bell	ETB	'	7	G	W	g	w
	8	BS	CAN	(	8	H	X	h	x
	9	HT	EM	)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[	k	{
	C	FF	FS	,	<	L	\	l	|
	D	CR	GS	-	=	M	]	m	}
	E	SO	RS	.	>	N	^	n	~
	F	SI	US	/	?	O	_	o	DEL

. **Tabla 3.3.1 Conjunto de carácteres ASCII** Como se observa en la tabla 3.3.1, podemos dividir a cada caracter representado en hexadecimal como una parte alta de 3 bits (**Most significant character**) y una parte baja de 4 bits (**Least significant character**), o sea, la representación total se hace con solo 7 bits.

De los problemas usuales en la programación está el convertir un número hexadecimal representado en 8 bits a dos carácteres ASCII los cuales sean la representación de dicho número para permitir la visualización en terminales de datos tales como Monitores de video o Impresoras entre otras.

Ejemplo:
Se quiere representar el número hexadecimal **70h** cuya representación en binario es **01110000b** como los dos carácteres ASCII "7" y "0", gráficamente:

7	**0**	en hexadecimal (8 bits)
□□	□□	
"7"	**"0"**	en ascii (16 bits)
□□□□	□□	
37h	**30h**	ascii en hexadecimal (16 bits)

Transportándolo a un programa:

```
NumHex   equ    0Ch          ; posición donde se almacena el número a convertir
AsciiH   equ    0Dh          ; posición donde se almacena el resultado parte alta
```

```
AsciiL      equ     0Eh             ; posición donde se almacena el resultado parte baja
.....
.....
            movlw   0Fh              ; dato para enmascarar parte alta
            andwf   NumHex,0        ; se enmascara la parte alta del número hexa y pasa a
W
            iorlw   30h              ; convierte  el número en ascii
            movwf   AsciiL          ; el número queda salvado en la variable de salida
            movlw   0F0h             ; dato para enmascarar parte baja
            andwf   NumHex,1        ; se enmascara la parte baja del número hexa y
queda allí
            swapf   NumHex,0        ; se invierten parte alta y baja
            iorlw   30h              ; convierte  el número en ascii
            movwf   AsciiL          ; el número queda salvado en la variable de salida.
.....
.....
```

Se debe tener en cuenta que el ejemplo anterior funciona en forma correcta siempre y cuando lo nibbles del número hxadecimal a convertir, estén en el rango de 0 a 9, debe realizarse un tratamiento adicional a estos si se encuentran en el rango de Ah a Fh. Realice en un programa esta condición.

3.4 Ramificación múltiple

Cuando se tiene que solucionar un diagrama de flujo como el de la figura 3.4.1 en el cual tenemos tres posibles respuestas a una pregunta, se plantean las soluciones aquí presentadas.

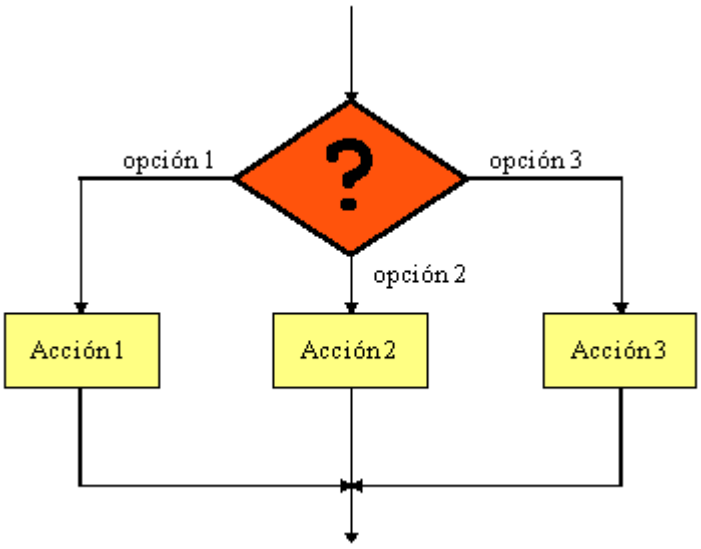


FIG. 3.4.1 Tres posibilidades para una pregunta. Solución #1

Una de las formas de solucionar en un programa este problema es:

Determinando para la opción 1, la opción 2 y la opción 3 un valor consecutivo como:

```
   opción1   equ   0
   opción2   equ   1
   opción3   equ   2
```

Uno de estos posibles valores llevarlos a W y en una parte del programa tratarlos así:

```
Decisión:                  ;sitio en donde la pregunta "?" tendría solución
   addwf   PCL,1
   goto      Acción1
   goto      Acción2
   goto      Acción3
Acción1:
.....                      ;instrucciones correspondientes a la Acción 1
.....
.....
   goto encuentro
Acción2:
.....                      ;instrucciones correspondientes a la Acción 2
.....
.....
   goto encuentro
Acción3:
.....                      ;instrucciones correspondientes a la Acción 3
.....
.....
encuentro:                 ;sitio de encuentro luego de una de las acciones
.....                      ;continuación del programa
.....
```

Solución #2

Otra forma posible es comparando una por una los valores de las diferentes opciones almacenadas en memoria RAM en una variable llamada **OPCION**

```
   movlw    Opción1
   xorwf    OPCION,0    ;se realiza la verificación del contenido de OPCION con respecto
a W
   btfsc     STATUS,Z    ;Verificando la bandera Z
   goto      Acción1
   movlw    Opción2
   xorwf    OPCION,0    ;se realiza la verificación del contenido de OPCION con respecto
a W
   btfsc     STATUS,Z    ;Verificando la bandera Z
   goto      Acción2
   movlw    Opción3
   xorwf    OPCION,0    ;se realiza la verificación del contenido de OPCION con respecto
a W
   btfsc     STATUS,Z    ;Verificando la bandera Z
   goto      Acción3
Acción1:
```

```
.....          ;instrucciones correspondientes a la Acción 1
.....
.....
   goto encuentro
Acción2:
.....          ;instrucciones correspondientes a la Acción 2
.....
.....
   goto encuentro
Acción3:
.....          ;instrucciones correspondientes a la Acción 3
.....
.....
encuentro:      ;sitio de encuentro luego de una de las acciones
.....          ;continuación del programa
.....
```

Aunque este último método es más largo que el anterior, permite que los valores de las diferentes opciones no sean consecutivos entre si.

3.5 Aritmética

Dentro de los microcontroladores PIC se cuenta con instrucciones aritméticas tales como ADDWF y ADDLW, SUBWF y SUBWF, para efectuar operaciones de suma y resta respectivamente e instrucciones tales como RLF y RRF para realizar operaciones de rotación a traves del carry con las cuales se pueden realizar divisiones entre 2 y multiplicaciones por 2 respectivamente, hasta este punto podríamos ver el conjunto de instrucciones un poco limitado, sin embargo, utilizando técnicas avanzadas de programación podemos obtener operaciones más complejas. Una buena cantidad de ellas la obtenemos de dos notas de aplicación de Microchip en formato PDF.
Los siguientes archivos pueden observarse con el Acrobat Reader®
Rutinas matemáticas para PIC16C5X/16CXX: PIC16C5X / 16CXX Math Utility Routines
Rutinas matemáticas generales Math Utility Routines

3.6 Temporización Existen momentos dentro de la programación en los que se necesita realizar un retardo de tiempo. Los retardos de tiempo se pueden obtener mediante hardware o por medio de ciclos repetitivos
basados en software. La precisión de los retardos generados por software depende en esencia del
tipo de oscilador que se utilice como base de tiempo en el microcontrolador, la mayor precisión se obtiene de los cristales de cuarzo.

La velocidad a la que se ejecuta el código (instrucciones) depende de la velocidad del oscilador y del número de ciclos de máquina ejecutados. Las instrucciones necesitan 1 ó 2 ciclos de máquina para ser ejecutadas. Un ciclo de máquina es un tiempo utilizado por el microcontrolador para realizar sus operaciones internas y equivale a cuatro ciclos del oscilador. Por tanto: $T_{ciclo\ máq.} = 4 * T_{osc}$ ☐☐☐ $T_{ciclo\ máq} = 4 / f_{osc}$ El número de ciclos de máquina utilizados por una instrucción para ser ejecutada depende de la misma. Las instrucciones que modifican el contador de programa necesitan dos (2) ciclos de máquina, mientras que todas las demás necesitan tan solo uno (1).

El hecho de generar ciclos repetitivos por medio del programa y calcular el tiempo total de ejecución nos puede ayudar a generar tiempos precisos.

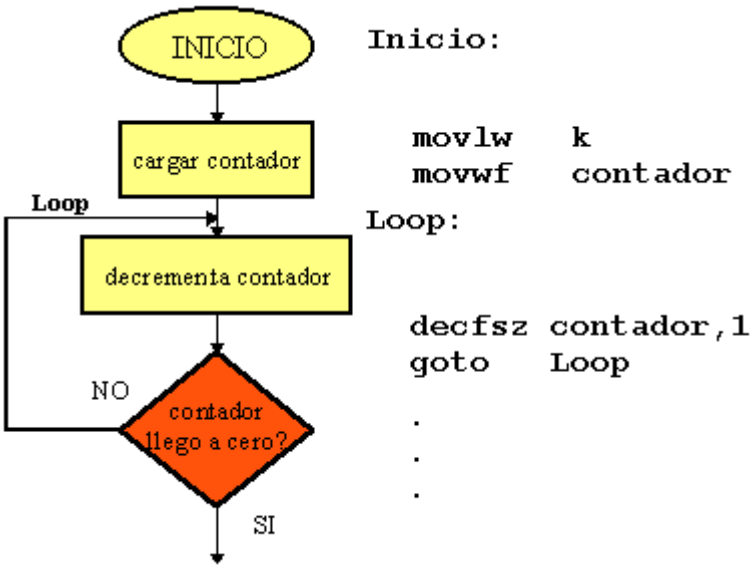


FIG. 3.6.1 Ciclo repetitivo de retardo
El ciclo repetitivo de retardo de la figura 3.6.1 se tomará un número de ciclos así:

Operación	**# de ciclos**
la carga de k en W	1
la carga de W en el contador	1
el decremento del contador mientras no llegue a cero	k-1
el decremento del contador cuando llegue a cero	2
el salto a Loop	2 * (k-1)
Total:	**3*k+1**

Por cada instrucción agregada debe incluirse en la cuenta total el número de ciclos correspondiente a dicha instrucción.

Trabajando a 4 Mhz y asumiendo que k se remplaza por el valor 15_d en el ejemplo tendríamos un tiempo igual a:
Número de ciclos = (3*15) +1 = 46 ciclos de máquina,
$T_{ciclo\ máq.}$= 4 / 4 Mhz = 1 µ segundo, el tiempo total del ejemplo entonces será 46 µsegundos.

4.1 Operaciones Entrada / Salida Objetivos:

- Verificar el modo en el que se debe programar el sentido de los puertos
- Realizar la entradas por puerto mediante la lectura de interruptores "dip-switch"
- Escribir sobre un puerto de salida visualizando sobre LEDs

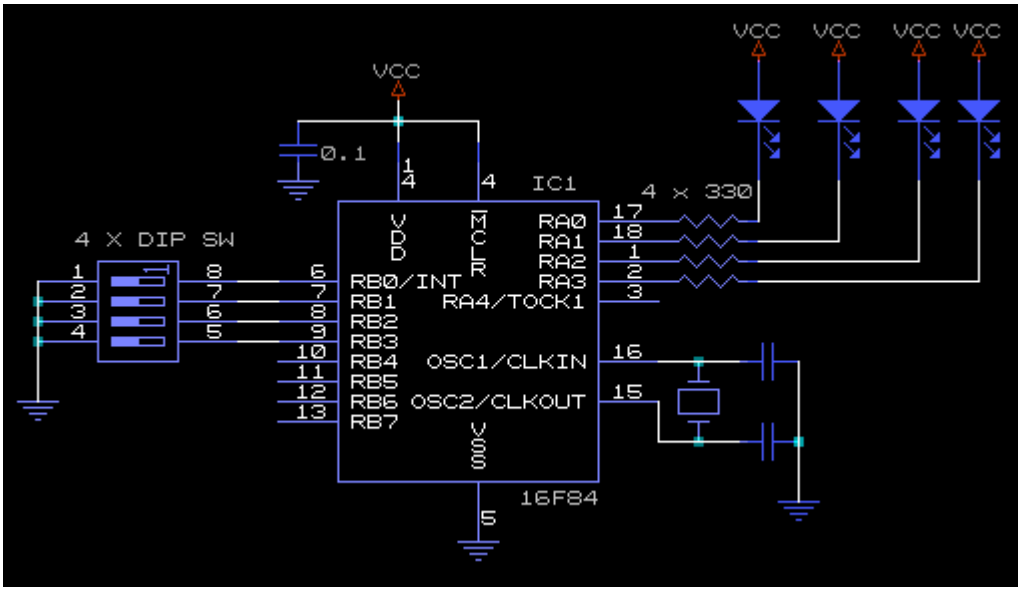


FIG. 4.1.1 Hardware para ejercicio Entrada/Salida Procedimiento:
En el proceso de utilización de un puerto debe tenerse en cuenta como primera instancia la
programación del sentido en que dicho puerto va a utilizarse. Una vez energizado el microcontrolador todos y cada uno de los puertos quedan programados como entrada, entonces,
tan solo deben programarse los que se quieren utilizar como salida.

En el hardware de la figura 4.1.1 se observa que se han colocado 4 dip switch al puerto B y estos
no poseen resistencia de pull up lo cual nos obliga a habilitar las resistencias internas con las que
cuenta el microcontrolador PIC16F84, el programa debe entonces en un repetitivo infinito leer el
nivel lógico que colocan los switch y pasar este resultado al puerto A complementando el estado de
la información puesto que de acuerdo a la disposición de los LEDs un estado bajo en el puerto
enciende el LED correspondiente y por ende un estado alto en el puerto, apaga el LED.

Programa:

```
status    equ   03h
optionr   equ   81h
trisa     equ   85h
porta     equ   05h
trisb     equ   86h
portb     equ   06h
;
Inicio:
    bsf   status,5  ;se pasa al banco 1 de RAM
    clrf  trisa     ;se programa el puerto A como salida
```

```
    movlw 0Fh       ;dato para la programación del puerto B
    movwf trisb     ;parte alta como salida y parte baja como entrada
    bcf   optionr,7 ;se habilitan resistencias de Pull Up
    bcf   status,5  ;se pasa al banco 0 de RAM
Loop:
    comf  portb,0   ;se lee el puerto B, se complementa su valor y el    ;resultado pasa a W
    movwf  porta    ;se pasa el resultado de W al puerto A
    goto   Loop     ;ejecuta un ciclo infinito
    end
```

En un proceso de lectura de interruptores, casi siempre cuando se detecta un cambio en el estado, es aconsejable amortiguar la lectura de estos con un retardo de software.

Dependiendo de la calidad del interruptor el tiempo del retardo puede estar al rededor de 50 mS. En el caso de este ejercicio en particular no es requerido puesto que un cambio en el interruptor debe reflejarse inmediatamente en el puerto de salida. Se debe tener en cuenta que nunca una entrada debe quedar al aire puesto que los microcontroladores PIC son hechos con tecnología CMOS. Es por este motivo que en el programa se programó la parte alta del puerto B como salida.

4.2 Visualización 7 segmentos Objetivos:

- Realizar la decodificación de BCD a 7 segmentos por software
- Multiplexar en el tiempo la información para 2 dígitos 7 segmentos

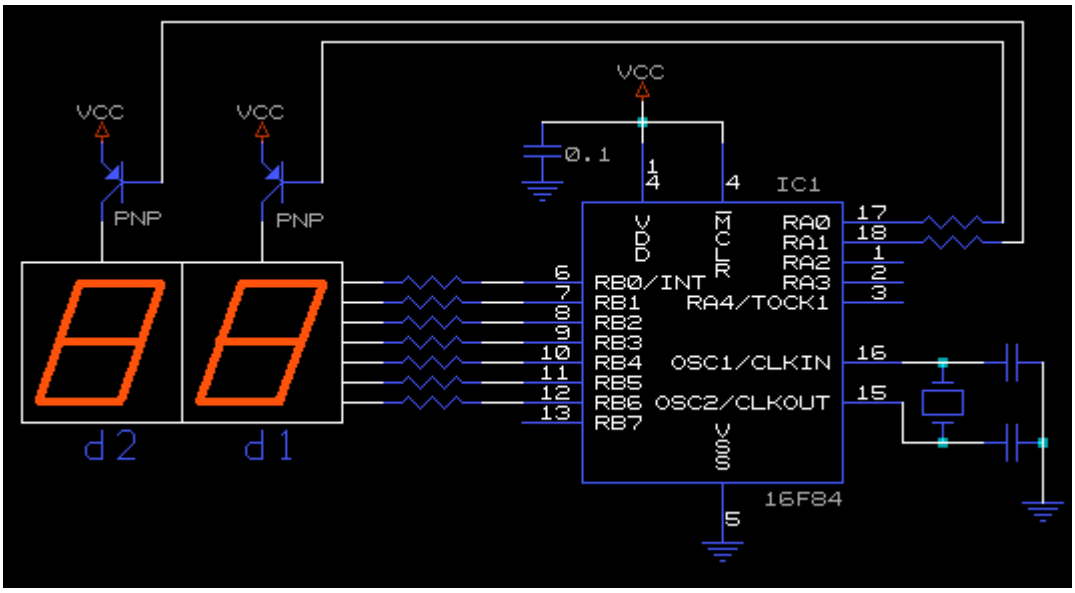


FIG. 4.2.1 Diagrama eléctrico para visualización dinámica en display 7 segmentos de dos dígitos

Procedimiento:
El hecho de visualizar datos en display de 7 segmentos de la forma en que se muestra en la figura 4.2.1 obliga a interconectar entre si los pines correspondientes a los segmentos del dígito 1 (d1) con los pines de los segmentos del dígito 2 (d2), de esta manera se ahorran líneas de conexión.

El método utilizado para la visualización dínamica consiste en visualizar cada dígito durante un instante de tiempo y a una velocidad tal que gracias a la persistencia del ojo el efecto final es que todos los dígitos están encendidos al tiempo

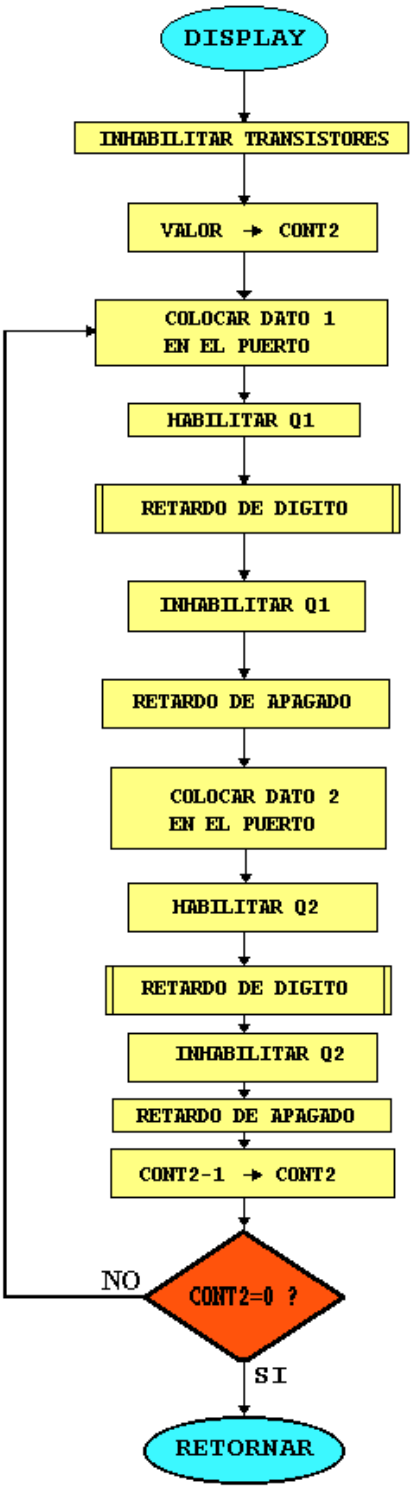


FIG. 4.2.2 Diagrama de flujo para visualización dinámica en display 7 segmentos de dos dígitos

Listado del programa del display en assembler

```
;rutina de display dinámico de dos dígitos
#define BANK0  bcf STATUS,RP0
#define BANK1  bsf STATUS,RP0
Cont2     equ     0x0D
Dato1     equ     0x0E
Dato2     equ     0x0F
Del1      equ     0x10
Del2      equ     0x11
Display
        BANK1
        clrf    TRISA          ;puerto A como salida
        clrf    TRISB          ;puerto B como salida
 BANK0
        movlw   0x03           ;inhabilita transistores
        movwf   PORTA
;
        movlw   .10            ;valor de repeticiones
        movwf   Cont2
LoopDisp
;
;Sacar al puerto el Dato 1 por un tiempo específico
;
        movf    Dato1,W        ;Dato para decodificar
        call    Tabla          ;Decodificación del dato
        movwf   PORTB          ;Dato decodificado a puerto
        bcf     PORTA,0        ;Habilita Q dato 1
        call    RetDig         ;Retardo de dígito
        bsf     PORTA,0        ;Inhabilita Q dato 1
        nop                    ;Retardo de apagado
        nop
        nop
        nop
;
;Sacar al puerto el Dato 2 por un tiempo específico
;
        movf    Dato2,W        ;Dato para decodificar
        call    Tabla          ;Decodificación del dato
        movwf   PORTB          ;Dato decodificado a puerto
        bcf     PORTA,1        ;Habilita Q dato 2
        call    RetDig         ;Retardo de dígito
        bsf     PORTA,1        ;Inhabilita Q dato 2
        nop                    ;Retardo de apagado
        nop
        nop
        nop
;
        decfsz Cont2,F         ;Decrementa Cont2, elude sig. sí cero
```

```
        goto   LoopDisp      ;Repite ciclo
        return
Tabla
        addwf  PCL,F
        retlw  0x01          ;Cuando el dígito es 0
        retlw  0x4F          ;Cuando el dígito es 1
        retlw  0x12          ;Cuando el dígito es 2
        retlw  0x06          ;Cuando el dígito es 3
        retlw  0x4C          ;Cuando el dígito es 4
        retlw  0x24          ;Cuando el dígito es 5
        retlw  0x20          ;Cuando el dígito es 6
        retlw  0x0F          ;Cuando el dígito es 7
        retlw  0x00          ;Cuando el dígito es 8
        retlw  0x04          ;Cuando el dígito es 9
        retlw  0x08          ;Cuando el dígito es A
        retlw  0x60          ;Cuando el dígito es B
        retlw  0x31          ;Cuando el dígito es C
        retlw  0x42          ;Cuando el dígito es D
        retlw  0x30          ;Cuando el dígito es E
        retlw  0x38          ;Cuando el dígito es F
;
RetDig
        movlw  2
        movwf  Del1
Loop1
        movlw  .50
        movwf  Del2
Loop2
        decfsz Del2,F
        goto   Loop2
        decfsz Del1,F
        goto   Loop1
        return
;
```

El listado anterior debe colocarse en un archivo llamado **display.asm,** *en el programa que se va a utilizar incluirlo con una línea del tipo:* #include <display.asm> *y llamarlo desde el programa con una línea:* call display *sin olvidar cargar previamente las variables* **dato1** *y* **dato2** *con los valores hexadecimales que se desea visualizar.*

4.3 Servicio de interrupción Objetivos:

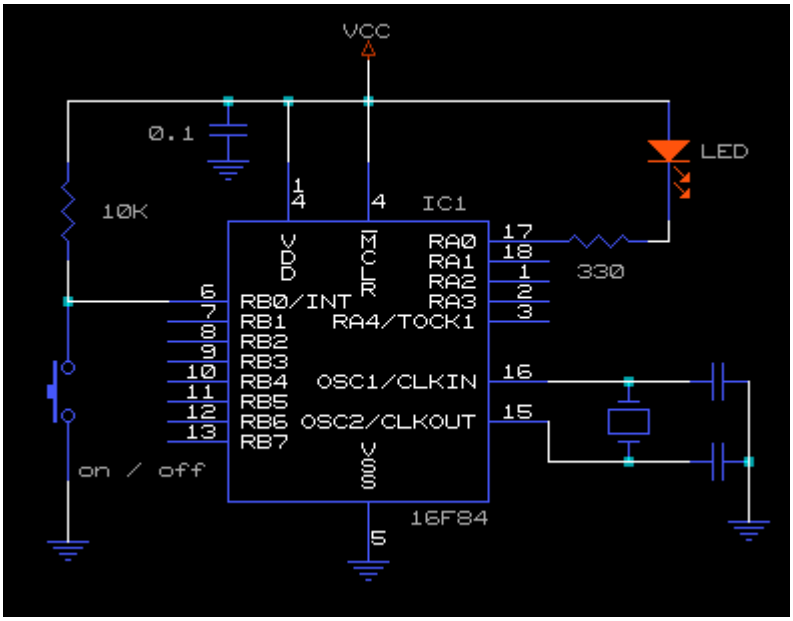
- Encender y apagar un LED como respuesta a un estímulo de interrupción
- Determinar la forma en que debe colocarse el código en el programa fuente para aceptar y atender una rutina de interrupción

Procedimiento:

Para ejemplificar el uso de un servicio de interrupción se ha dispuesto el hardware de la figura 4.3.1 en el cual se decide utilizar la interrupción externa **INT** (**RB0**) en un

PIC16F84, esta interrupción está vectorizada a la dirección de memoria de programa 004h, dentro de la atención a esta interrupción se opta por complementar el estado del LED colocado al puerto RA0 cada vez que esta sea atendida.

Diagrama electrónico:



IG. 4.3.1 Hardware para ejemplo de interrupción

Diagrama de flujo:

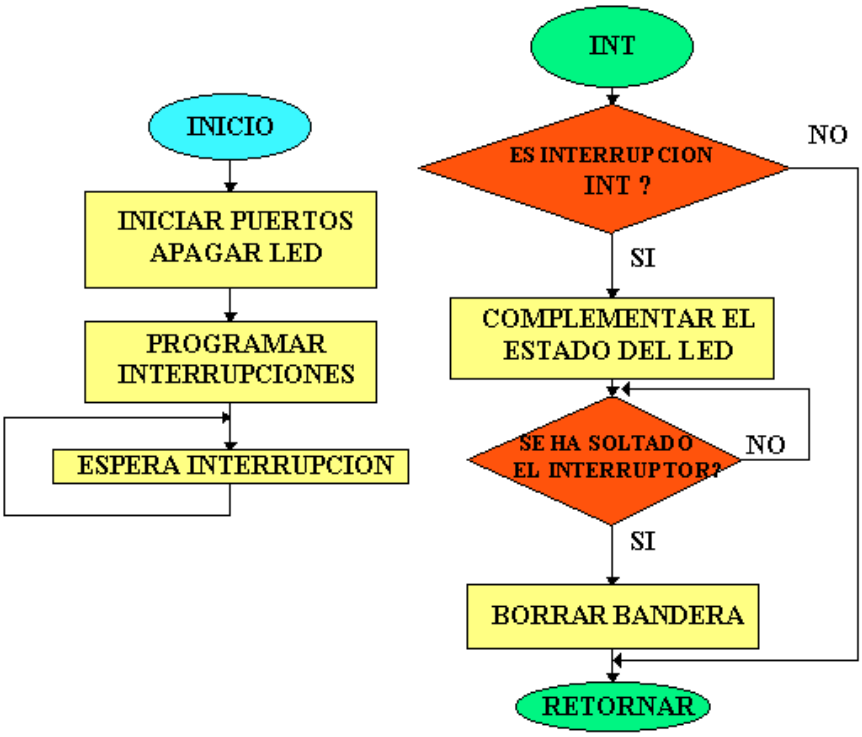


FIG. 4.3.2 Diagrama de flujo para acción de interrupción
Listado del programa en assembler:

```
;Programa para realizar el apagado y encendido de un LED colocado en el ;Puerto A0
basado en la interrupción externa INT (RB0)
;
list p=16F84
status   equ   0x03
porta    equ   0x05
portb    equ   0x06
intcon   equ   0x0B
optionr  equ   0x81
trisa    equ   0x85
trisb    equ   0x86
#define  LED   porta,0
#define  BANK1 bsf status,5
#define  BANK0 bcf status,5
        org   000h    ;Indica al ensamblador la dirección de memoria de
                      ;la sig. instrucción
        goto  Inicio
        org   004h    ;Indica al ensamblador la dirección de memoria de
                      ;la sig. instrucción
Interrupcion
     btfss intcon,1    ;es interrupción INT?
     retfie            ;retorna de interrupción y GIE=1
     btfsc porta,0     ;probar estado actual del LED
     goto  Prender     ;va a encender el LED
Apagar
     bsf   porta,0     ;apaga el LED
     goto  Espera
Prender
    bcf    porta,0     ;enciende el LED
Espera
    btfss  portb,0     ;espera a que se suelte el pulsador
    goto   Espera
    bcf    intcon,1    ;borra bandera INT
    retfie             ;retorna de interrupción y GIE=1
;Programa principal
Inicio
    BANK1              ;selección del banco 1
    bcf    trisa,0     ;selecciona porta,0 como salida
    BANK0              ;selección de banco 0
    bsf    porta,0     ;apagar LED
;programación de interrupción
    bsf    intcon,4    ;activar interrupción INT
    BANK1              ;selección banco 1
    bcf    optionr,6   ;selección del flanco de bajada en el pin INT
    BANK0
    bsf    intcon,7    ;Habilitar interrupciones globales
```

```
goto   $          ;queda a la espera de interrupción
end
```

El símbolo $ significa la dirección de memoria de programa en donde se encuentra éste (ciclo infinito de espera)
Debe notarse la ubicación de la rutina de interrupción a partir de la posición de memoria de programa 004h.

4.6 Conversión A/D Objetivos:

- Realizar la observación práctica de la utilización del conversor A/D en un PIC16C71
- Configurar uno de los canales de entrada al conversor A/D (canal 0)

Procedimiento:
Para realizar la observación práctica mencionada en los objetivos, se realiza el montaje de un voltimetro digital AC el cual utiliza el canal análogo 0 como entrada de la muestra DC tomada de un divisor de voltaje. De acuerdo a la relación del divisor de tensión conformado con el reostato **Ajuste** y de las resistencias, debe obtenerse un voltaje máximo de 5 voltios en el pin RA0, correspondiente a la máxima entrada de voltaje alterno colocado en **VAC in** .

Diagrama eléctrico:

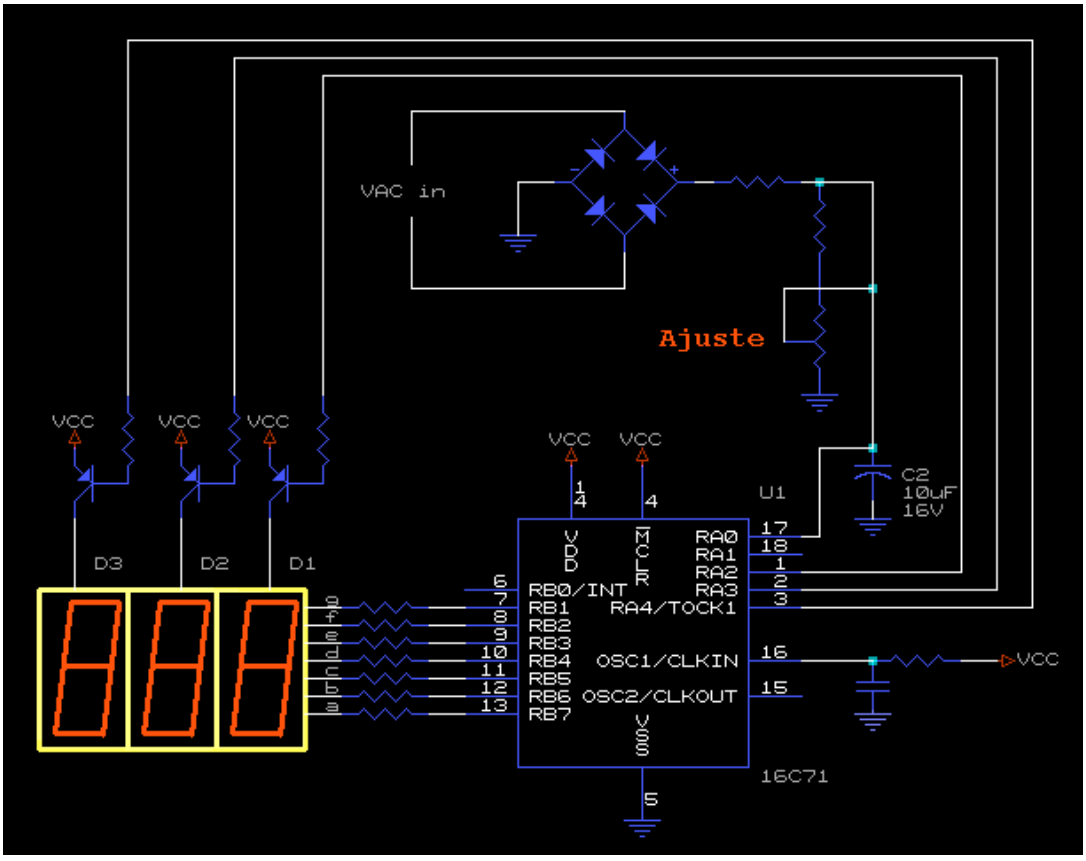


FIG. 4.6.1 Hardware utilizado para el voltimetro AC
Listado del programa en assembler

```
 LIST P=16C71
INDF         EQU     H'0000'
TMR0          EQU     H'0001'
PCL          EQU     H'0002'
STATUS       EQU     H'0003'
FSR          EQU     H'0004'
PORTA        EQU     H'0005'
PORTB        EQU     H'0006'
ADCON0       EQU     H'0008'
ADRES        EQU     H'0009'
PCLATH       EQU     H'000A'
INTCON       EQU     H'000B'
;
OPTION_R     EQU     H'0081'
TRISA        EQU     H'0085'
TRISB        EQU     H'0086'
ADCON1       EQU     H'0088'
;=====================================================
============
W            EQU     H'0000'
F            EQU     H'0001'
#define      _z      STATUS,2
#define      _c      STATUS,0
#define      RP0     STATUS,5
#define      RP1     STATUS,6
#define      BANK1   bsf  RP0
#define      BANK0   bcf  RP0
;
H_BYTE       equ      10         ;Variable IN de la rutina B2_BCD
L_BYTE       equ      11         ;Variable IN de la rutina B2_BCD
R0           equ      12         ;MSByte DE LA RUTINA B2_BCD
R1           equ      13
R2           equ      14         ;LSByte DE LA RUTINA B2_BCD
OnDyn        equ      15
count        equ      16
temp         equ      17
DIG3         equ      18         ;MSD DE LA RUTINA DE DISPLAY
DIG2         equ      19
DIG1         equ      1A         ;LSD DE LA RUTINA DE DISPLAY
PERSIST      equ      1B
CounterDig   equ      1C
DELAY        equ      1D         ;RETARDO VISUALIZACION
DELAY2       equ      1E
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  org 0
```

```
  goto Main
;*********************************************************************
****
;Conversión de Binario a BCD 5 dígitos
;Entrada: Un binario de 16 Bits en H_Byte y L_Byte
;Salida: R0, R1 y R2; R0 es el MSD
;*********************************************************************
****
B2_BCD
   bcf      _c
   movlw    .16
   movwf    count
   clrf     R0
   clrf     R1
   clrf     R2
Loop16
   rlf      L_BYTE
   rlf      H_BYTE
   rlf      R2
   rlf      R1
   rlf      R0
;
   decfsz   count
   goto     adjDEC
   retlw    0
;
adjDEC
   movlw    R2
   movwf    FSR
   call     adjBCD
;
   movlw    R1
   movwf    FSR
   call     adjBCD
;
   movlw    R0
   movwf    FSR
   call     adjBCD
;
   goto     Loop16
;
adjBCD
   movlw    3
   addwf    0,W
   movwf    temp
   btfsc    temp,3
   movwf    0
   movlw    30
```

```
   addwf    0,W
   movwf    temp
   btfsc    temp,7
   movwf    0
   retlw    0
;#######################################################
####################
Main
;Inicialización de puertos
    BANK1
    CLRF    TRISB      ;Programación del puerto B como salida
    MOVLW   03
    MOVWF   TRISA      ;Programación del puerto A, RA0 y RA1 como entrada
    BANK0
    MOVLW   0FF
    MOVWF   PORTA      ;Inicialización de los puertos A y B
    MOVWF   PORTB
;Inicialización del módulo A/D Canal 0 activo
    BANK1
    MOVLW   B'000000010'
    MOVWF   ADCON1
    BANK0
    MOVLW   0C1
    MOVWF   ADCON0
;inicialización de dígitos
    clrf    DIG1
    clrf    DIG2
    clrf    DIG3
INI_AD
    MOVLW   .5
    MOVWF   DELAY2
    BSF     ADCON0,2
VISUALIZAR
    MOVLW   .255         ;RETARDO DE DISPLAY
    MOVWF   DELAY
LOOP_VISUAL
    CALL    DISPLAY
    DECFSZ  DELAY,F
    GOTO    LOOP_VISUAL
    DECFSZ  DELAY2,F
    GOTO    VISUALIZAR
TEST_FIN_AD
    BTFSC   ADCON0,1     ;SÍ ESTÁ EN 0 NO HA TERMINADO LA CONVERSION
    GOTO    HEX_BCD      ;CONVERSION Y ADECUACION DE DATOS A VER
    GOTO    VISUALIZAR   ;SEGUIR CON LOS DATOS YA PREDETERMINADOS
;
DISPLAY
    MOVLW  03
```

```
    MOVWF  CounterDig
    MOVLW  0FB
    MOVWF  OnDyn
    MOVLW  DIG1       ;Posición dígito menos significativo (LSD)
    MOVWF  FSR
Salt1
    MOVLW  10
    MOVWF  PERSIST
    MOVF   INDF,W
    DECF   FSR,F
    CALL   Tabla
    MOVWF  PORTB
    MOVF   OnDyn,W
    MOVWF  PORTA
    BSF    _c
    RLF    OnDyn,F
Decre
    DECFSZ PERSIST,F
    GOTO   Decre
    NOP
    NOP
    MOVLW  0FFh
    MOVWF  PORTA
    DECFSZ CounterDig,F
    GOTO   Salt1
    NOP
    NOP
    NOP
    RETURN
Tabla
    ADDWF  PCL,F
    RETLW  03h     ;0
    RETLW  9Fh     ;1
    RETLW  25h     ;2
    RETLW  0Dh     ;3
    RETLW  99h     ;4
    RETLW  49h     ;5
    RETLW  41h     ;6
    RETLW  1Fh     ;7
    RETLW  01h     ;8
    RETLW  19h     ;9
    RETLW  0FDh    ;SEG g
    RETURN
;*****************************************************************************
****
HEX_BCD
    BCF    ADCON0,1 ;FIN DE CONVERSIÓN
    MOVF   ADRES,W  ;RESULTADO DE LA CONV. A/D
```

```
    SUBLW  0FBh
    BTFSS  _c  ;
    GOTO   OVERFLOW
;SITUACION NORMAL
    MOVF   ADRES,W
    MOVWF  L_BYTE   ;PARA LA CONVERSION B2_BCD
    CLRF   H_BYTE   ;PARA LA CONVERSION B2_BCD
    CALL   B2_BCD   ;CONVERTIR DATO
    MOVF   R2,W
    ANDLW  0Fh
    MOVWF  DIG1
    SWAPF  R2,W
    ANDLW  0Fh
    MOVWF  DIG2
    MOVF   R1,W
    MOVWF  DIG3
    GOTO   INI_AD
    MOVLW  0Ah      ;representa overflow en la medición
    MOVWF  DIG3
    MOVWF  DIG2
    MOVWF  DIG1
    GOTO   INI_AD
    end
```

El programa presentado hace uso de los recursos que posee el PIC16C71 el cual posee 4 canales análogos y un conversor A/D de 8 bits; para mayor información ver las especificaciones técnicas de este dispositivo.

4.7 Comunicación serial

Objetivos:

- Verificar la comunicación serial síncrona y asíncrona
- Comprobar los algoritmos de comunicación serial

En vista de que algunos de los elementos de la familia PIC16CXXX no poseen periféricos de comunicación serial, este capítulo hará referencia al desarrollo del algoritmo como tal simulando los pines de comunicación serial con puertos del microcontrolador.

Comunicación serial síncrona:
La comunicación síncrona se caracteriza porque los pulsos de sincronización deben ser transmitidos a lo largo de la línea de comunicación entre el transmisor y el receptor. Dentro de los varios tipos de comunicación serial síncrona vamos a notar el protocolo I^2C ó de dos hilos y el protocolo SPI ó de tres hilos.

	Línea(s) de datos	**Línea de reloj**
I^2C	SDA (serial data)	SCL
SPI	SO (serial out), SI (serial in)	SCK

Tabla 4.7.1 Nomenclatura de los pines de comunicación (síncronos)

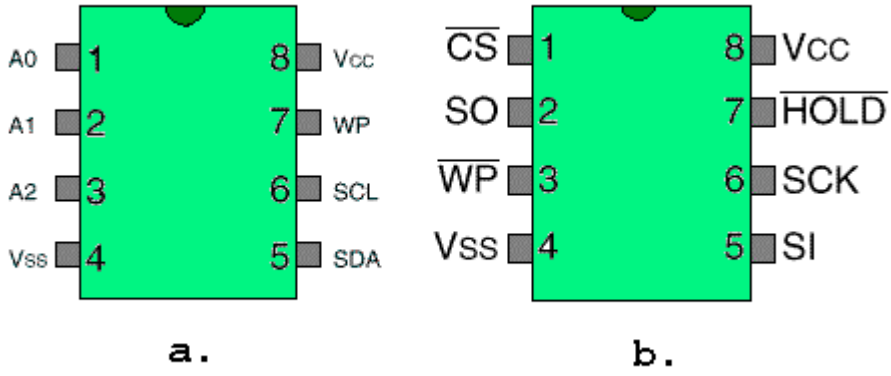


FIG. 4.7.1 Disposición típica de pines en dispositivos síncronos (a) I^2C, (b) SPI

I^2C

El bus I^2C es un bus diseñado para que sobre éste puedan colocarse varios dispositivos dentro de la misma tarjeta electrónica (comunicación multipunto), cada dispositivo tendrá una dirección lógica asignada físicamente mediante los pines A0, A1 y A2 de acuerdo al nivel lógico al que estos sean alambrados. ver estos pines en la figura **4.7.1 a**.

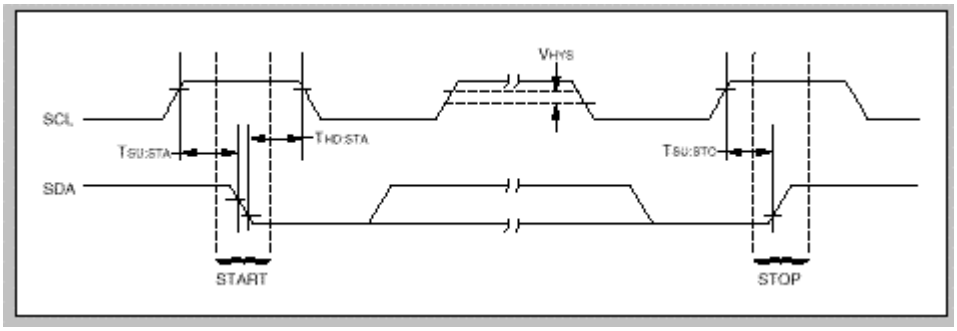


FIG. 4.7.2 Bits de START y STOP del protocolo I^2C

En las figuras **4.7.2** y punto se observa la forma en que las señales SCL y SDA deben ser manejadas. Para iniciar la comunicación sobre un dispositivo I^2C debe realizarse la secuencia denominada bit de START que consiste en pasar la línea de datos SDA de nivel alto a bajo mientras que la línea SCL permanece en alto. Para la culminar la comunicación con el dispositivo I^2C debe ejecutarse la secuencia denominada bit de STOP la cual consiste en pasar la línea de datos SDA de nivel bajo a alto mientras que la línea de reloj SCL permanece en alto. Un bit de datos es aceptado por el dispositivo mientras que sobre la línea de datos SDA permanece el nivel adecuado al bit en cuestión, y sobre la línea de reloj SCL se lleva a cabo un pulso, es decir, el paso de nivel de bajo a alto y luego de alto a bajo. Los tiempos implicados en esta secuencia dependen básicamente del fabricante del dispositivo.

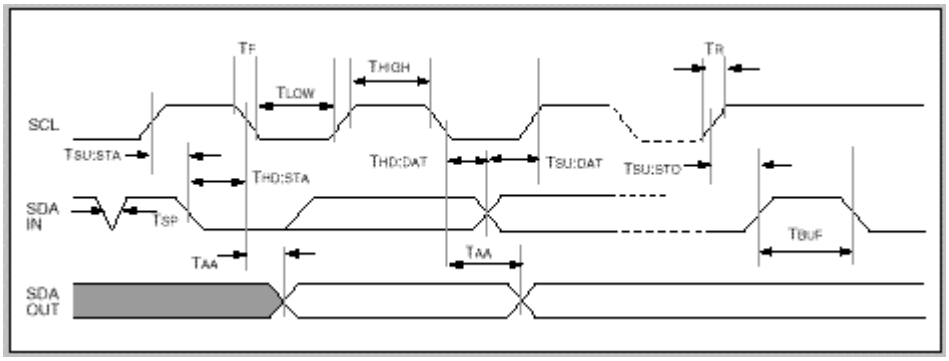


FIG. 4.7.3 Temporización en el bus I^2C

SPI

El bus SPI es un bus diseñado para que sobre éste se coloque un dispositivo maestro y un dispositivo esclavo (comunicación punto a punto) ver figura **4.7.1 b**. Con relación al bus I2C podemos notar que éste soporta mayor velocidad de comunicación.

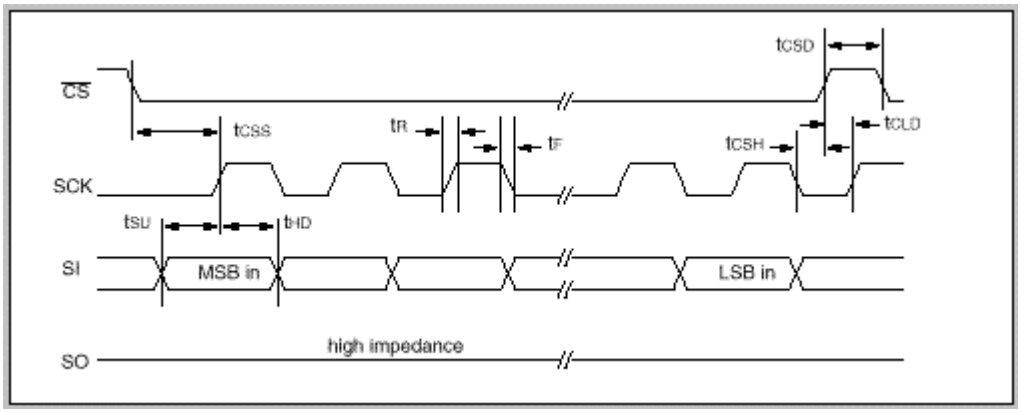


FIG. 4.7.4 Entrada de datos a dispositivo SPI

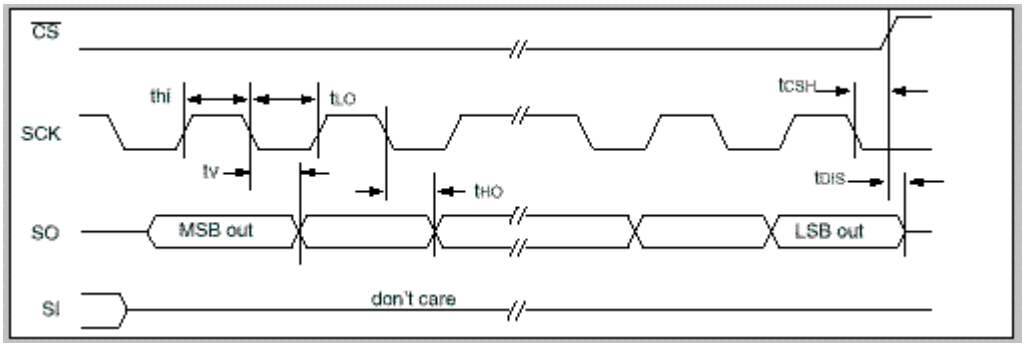


FIG. 4.7.5 Salida de datos de dispositivo SPI

El dispositivo SPI posee como observamos el figura 4.7.1, una línea de selección CS la cual debe pasar al nivel lógico activo (en este caso bajo) para poder realizar la comunicación con el dispositivo. Desde este punto de vista podríamos colocar sobre un bus de este tipo varios dispositivos, pero utilizando un dispositivo decodificador adicional.

Otra línea podemos observar es la línea HOLD la cual permite al procesador detener momentáneamente la comunicación, ver figura **4.7.6**

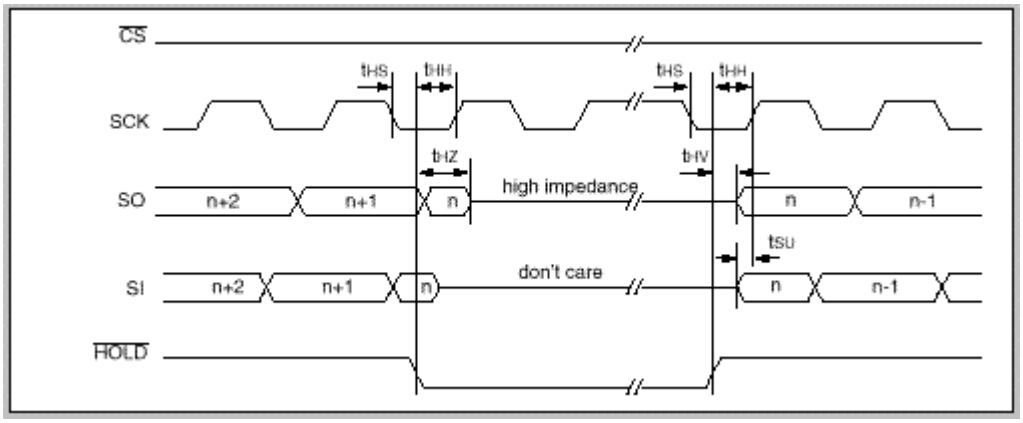


FIG. 4.7.6 Utilización de la línea HOLD

Comunicación serial asíncrona:
En este tipo de comunicación tanto el transmisor como el receptor tienen incluido el reloj de sincronización de tal forma que no se transmite a lo largo de la línea de comunicación.

Datos de Entrada	Datos de Salida
RxD	TxD

Tabla 4.7.2 Nomenclatura de los pines básicos (asíncronos)

Para saber algunas generalidades del puerto serial asíncrono: The serial port,
Este artículo ha sido tomado de: http://www.lvr.com en donde pueden encontrarse temas muy interesantes acerca del mundo de la computación y las diferentes conexiones paralelas, serie, etc.

Algunas notas de aplicación de comunicación en formato **PDF**:

Comunicación serial síncrona SPI: AN530 AN560
Comunicación serial sincrona I2C: AN515 AN567
Comunicación serial asíncorna: AN510 AN555

El código fuente de las notas de aplicación comprimidas en formato **ZIP:**

Comunicación serial síncrona SPI: AN530 AN560
Comunicación serial sincrona I2C: AN515 AN567
Comunicación serial asíncorna: AN510 AN555

Aplicación de Transmisión y Recepción con PIC16F84 simulando el puerto serial asíncrono:
Código fuente: TXRXASIN.TXT (4KB)

Aplicación de Escritura y Lectura sobre dispositivo I2C simulando el puerto serial síncrono:

Código fuente: SERIAL.TXT (5KB)

5.1 Exploración de teclado

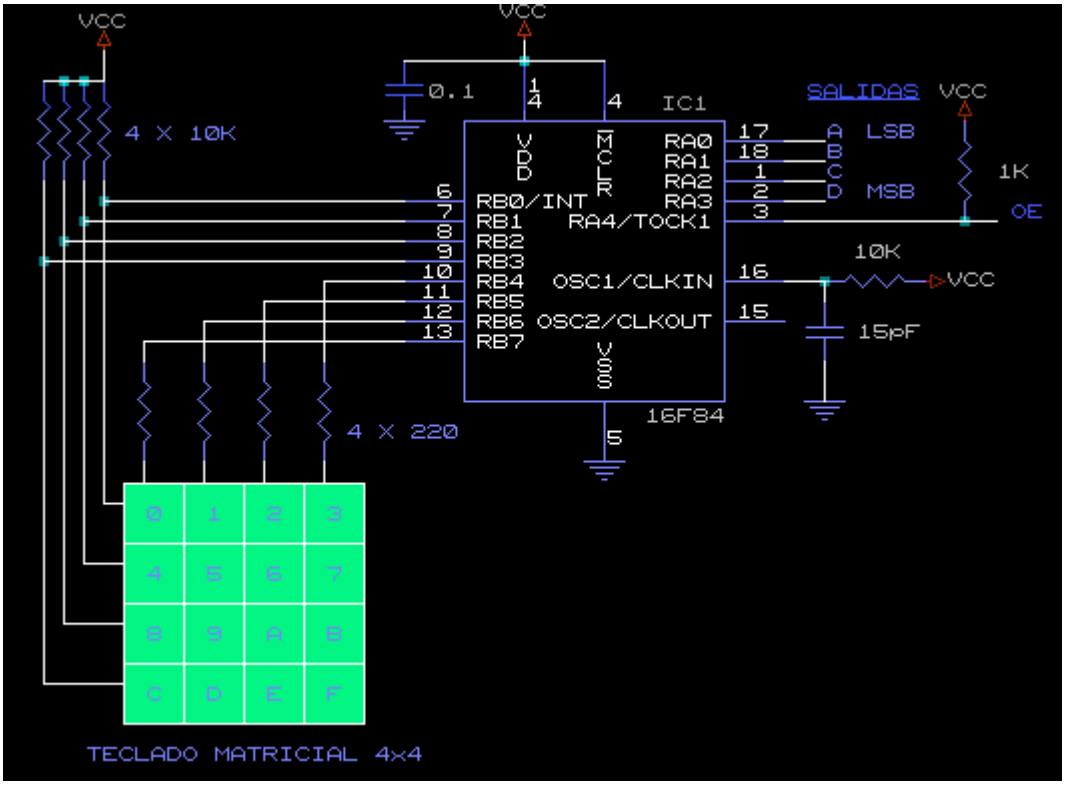


FIG. 5.1.1 Hardware correspondiente al experimento de exploración de teclado

Para la lectura del teclado debemos tener en cuenta la disposición de las filas y las columnas como se observa en la FIGURA 5.1.2 con la cual realizando la operación allí descrita se debe obtener un número consecutivo de las teclas en la organización aquí mostrada. Luego, mediante el acceso a una tabla se decodifica la tecla leída para obtener el patrón final observado en el diagrama del hardware FIG. 5.1.1.
Ej. Sí se oprimiése la tecla C del teclado (FIG. 5.1.1), el código de exploración correspondiente a esta es el 13d (FIG. 5.1.2) que debe ser representado como el 1100b en las salidas DCBA (FIG. 5.1.1).

	1	2	3	4
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

Tecla=#Filax4+#Columna

FIG. 5.1.2 Distribución del teclado, numeración en filas y columnas y la fórmula para hallar la tecla oprimida

En este experimento se realiza la emulación del integrado decodificador de teclado 74C922 en cuanto a su funcionamiento, pero de acuerdo a la configuración de hardware de la FIG. 5.1.1.

Listado del programa para exploración de teclado:

```
   list p=16f84
#include <p16f84.inc>    ;archivo de encabezado por Microchip®
;
;ESTE PROGRAMA EMULA UN 74C922 DECODIFICADOR DE TECLADO
;
CONTFIL EQU  0x12      ;Contador de Filas
CONTCOL EQU  0x13      ;Contador de Columnas
COLKBD EQU  0x14       ;DATO EN COLUMNAS
Temp EQU 0x15
R1 EQU 0x16            ;Variable para Retardo
R2 EQU 0x17            ;Variable para Retardo
R3 EQU 0x18            ;Variable para Retardo
R4 EQU 0x19            ;Variable para Retardo
COUNT EQU 0x1A
CHAR EQU 0x1B          ;Almacenamiento temporal SCAN
AUX EQU 0x1C           ;Variable Auxiliar

#define _z STATUS,2
#define _c STATUS,0
#define OE PORTA,4
#define BANK0  bcf STATUS,RP0
#define BANK1  bsf STATUS,RP0
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
   ORG 0x00
MAIN
   BANK1
   CLRF TRISA
   BANK0
   CLRF PORTA
   NOP
   NOP
Muestre
   BCF OE
RUSCAN
   CALL SCAN
   XORLW 0x00      ;espera una Tecla
   BTFSC _z
   GOTO Muestre
```

```
    MOVWF PORTA
    MOVLW .50
    MOVWF COUNT
LOOPSCAN
    CALL DEL5MS
    DECFSZ COUNT,1
    GOTO LOOPSCAN
    GOTO RUSCAN
;*********************************************************************
***
DEL5MS
    MOVLW .12
    MOVWF R1
    MOVLW 7
    MOVWF R2
    MOVLW 1
    MOVWF R3
    MOVLW 1
    MOVWF R4
LOOPDEL5
    DECFSZ R1,F
    GOTO LOOPDEL5
    DECFSZ R2,F
    GOTO LOOPDEL5
    DECFSZ R3,F
    GOTO LOOPDEL5
    DECFSZ R4,F
    GOTO LOOPDEL5
    NOP
    RETURN
;*****************************************************************
;RETORNA W=00 NO HAY TECLA OPRIMIDA,
;RETORNA W=COD SI TECLA OPRIMIDA.
;*************************************************************
SCAN
    BANK1
    MOVLW 0x0F        ;el puerto que lee teclado <0:3> filas (in)
    MOVWF TRISB
    BANK0
    MOVLW 0x01
    MOVWF CONTCOL
    MOVLW 0x7F
    MOVWF COLKBD
RSTFIL
    CLRF CONTFIL      ;RESET CONT FILAS
    MOVF COLKBD,W
    MOVWF PORTB       ;COLOCAR UN CERO EN COLUMNAS
    nop
```

```
    nop
    nop
    MOVF PORTB,W       ;LEER FILAS DE TECLADO
    MOVWF AUX
    RLF AUX,F
    RLF AUX,F
    RLF AUX,F
    RLF AUX,F
TESTFIL
    RLF AUX,F
    BTFSS _c
    GOTO ACERTADO
    INCF CONTFIL,F
    MOVF CONTFIL,W
    XORLW 0x04
    BTFSS _z
    GOTO TESTFIL

    BSF _c
    RRF COLKBD,F      ;rotacion del cero a colocar
    INCF CONTCOL,F
    MOVF CONTCOL,W
    XORLW 0x05
    BTFSS _z
    GOTO RSTFIL
    RETLW 0x00

ACERTADO
    MOVF CONTFIL,W
    XORLW 0x00
    BTFSC _z
    GOTO ESCERO
    MOVLW 0x00
MUL
    ADDLW 0x04
    DECFSZ CONTFIL
    GOTO MUL
SUMACOL
    ADDWF CONTCOL,W
    CALL TABKBD
    RETURN
TABKBD
    addwf PCL,F
    retlw 0       ;inválido
    retlw 0x10
    retlw 0x11
    retlw 0x12
    retlw 0x13
    retlw 0x14
```

```
    retlw 0x15
    retlw 0x16
    retlw 0x17
    retlw 0x18
    retlw 0x19
    retlw 0x1A
    retlw 0x1B
    retlw 0x1C
    retlw 0x1D
    retlw 0x1E
    retlw 0x1F
ESCERO
    MOVLW 0x00
    GOTO SUMACOL
    end
```

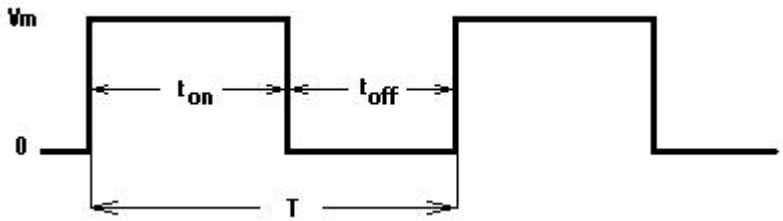
5.2 Control de motor paso a paso

Para conocer las generalidades de los motores paso a paso visite:

1. Jones on Stepping Motors
2. Tom Porter's Motor Control Web Page
3. Ian Harries on Stepping Motors
4. Tony Mercer's web Pages

5.4 Regulación de velocidad de motor D.C.

Por medio de la presente práctica se pretende hacer la variación de la velocidad a un motor DC aplicando un voltaje variable a este por medio del método de modulación por ancho de pulso o PWM.



Onda rectangular y sus características

El método de modulación por ancho de pulso está basado en la obtención de un voltaje DC variable a partir de una onda rectangular de frecuencia constante y ciclo útil variable, de tal manera que:

Vdc = (Ciclo útil) * Vm

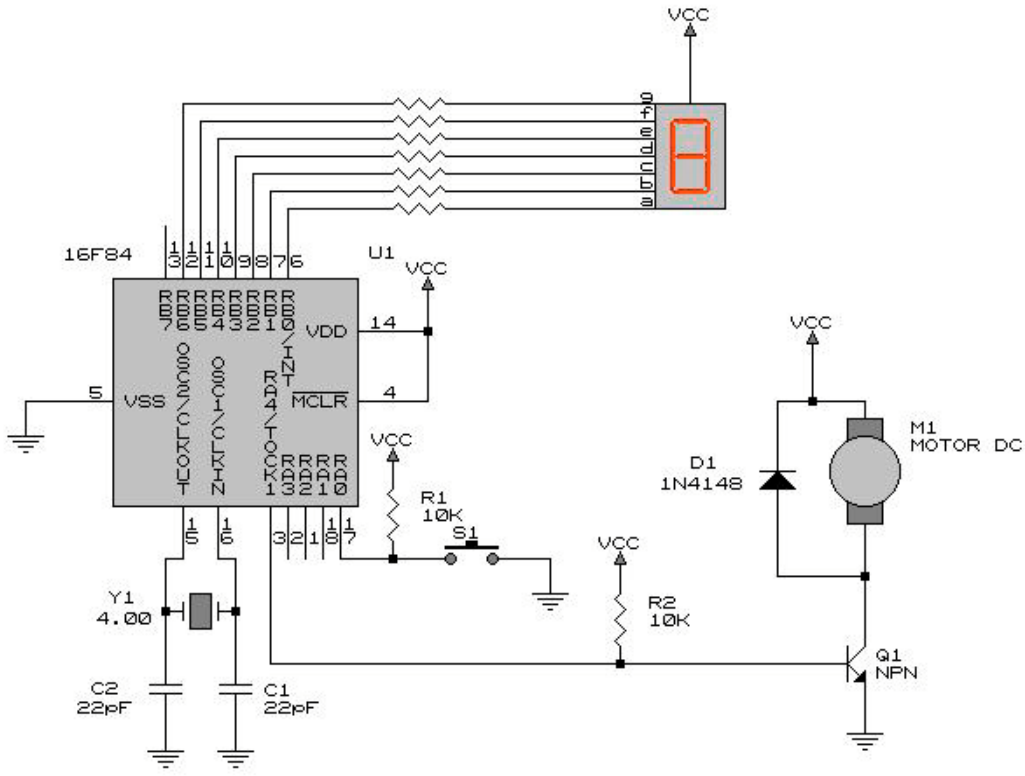
ó

Vdc = (t_{on}/T) * Vm

Siendo:
t_{on} *el tiempo en alto de la onda cuadrada*
T *el periodo de la onda rectangular* (T = t_{on} + t_{off})
Vm *el voltaje máximo de la onda*

Se puede concluir a partir de esta simple ecuación que sí hacemos variar el ciclo útil de la onda rectangular obtendremos una variación en el voltaje promedio y sí este es aplicado como alimentación del motor DC, el efecto será el de la variación de la velocidad.

Para la aplicación de este principio nos basamos en el siguiente circuito:



El programa que comanda sobre este Hardware funciona basado en el pulsador S1 como control de 10 pasos discretos de velocidad los cuales son indicados sobre el display 7 segmentos (0 a 9) y cuyo efecto final se observa directamente en la velocidad del motor M1.

Listado del programa

```
LIST P= 16F84
#INCLUDE<P16F84.INC>
DIG EQU 0CH
VROFF EQU 0DH
VRON EQU 0EH
CONT1 EQU 0FH
CONT2 EQU 10H
X EQU .250
Y EQU .12
;*****************************************************************
INICIO
        BSF STATUS,RP0 ; PROGRAMACIÓN DEL SENTIDO DE PUERTOS
        CLRF TRISB
        MOVLW B'11100001'
        MOVWF TRISA
        BCF STATUS,RP0
        MOVLW 00H
        MOVWF DIG
        BCF PORTA,4
DIS_LOOP:
        CALL DISPLAY ; VISUALIZACIÓN DE DIGITO
        BTFSC PORTA,0 ; LA TECLA ESTA OPRIMIDA?
        GOTO RMOTOR
        CALL RETAR ; TIEMPOS DE ANTIREBOTE
        CALL RETAR
        CALL RETAR
```

```
        CALL RETAR
        CALL RETAR
        CALL RETAR
        CALL RETAR
        CALL RETAR
        CALL RETAR
        CALL RETAR
        CALL RETAR
        INCF DIG,1
        MOVLW .11
        XORWF DIG,0
        BNZ RMOTOR
        CLRF DIG
RMOTOR
        CALL MOTOR
        GOTO DIS_LOOP
;*****************************************************************
***********
DISPLAY
        MOVF DIG,0
        CALL TABLA
        MOVWF PORTB
        RETURN
;*****************************************************************
***********
TABLA: ; TABLA DE DATOS PARA DECODIFICACIÓN A SIETE SEGMENTOS
        ADDWF PCL,1
        RETLW 0x3F
        RETLW 0x06
```

```
        RETLW 0x5B
        RETLW 0x4F
        RETLW 0x66
        RETLW 0x6D
        RETLW 0x7D
        RETLW 0x07
        RETLW 0x7F
        RETLW 0x6F
        RETLW 0x77
;*****************************************************************
************
MOTOR: ; ACTIVA MOTOR CON TIEMPOS: Ton Y Toff
        MOVF DIG,0
        SUBLW .10
        MOVWF VROFF
        CLRW
        XORWF DIG,0
        BZ OFFMOTOR
        MOVF DIG,0
        MOVWF VRON
        BSF PORTA,4 ; ENCIENDE EL MOTOR
MOTOR1:
        CALL RETAR
        DECFSZ VRON,1
        GOTO MOTOR1
OFFMOTOR
        BCF PORTA,4 ; APAGA MOTOR
        MOVF DIG,0
```

```
        XORLW .10
        BZ MOTOR3
MOTOR2
        CALL RETAR
        DECFSZ VROFF,1
        GOTO MOTOR2
MOTOR3
        RETURN
;*****************************************************************
*************
RETAR
        MOVLW X
        MOVWF CONT1
CICLO1
        MOVLW Y
        MOVWF CONT2
CICLO2
        DECFSZ CONT2,1
        GOTO CICLO2
        DECFSZ CONT1,1
        GOTO CICLO1
        RETURN
;*****************************************************************
**************
        END
```

5.5 Periféricos Inteligentes: Módulo LCD

Los módulos LCD están compuestos básicamente por una pantalla de cristal líquido y un circuito microcontrolador especializado el cual posee los circuitos y memorias de control necesarias para desplegar el conjunto de caracteres **ASCII,** un conjunto básico de caracteres japoneses, griegos y algunos símbolos matemáticos por medio de un circuito denominado generador de caracteres. La lógica de control se encarga de mantener la

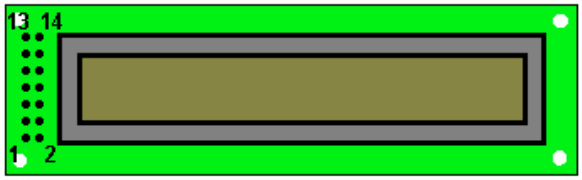
información en la pantalla hasta que ella sea sobreescrita o borrada en la memoria RAM de datos..

La pantalla de cristal líquido está conformada por una ó dos líneas de 8, 16, 20, 24 ó 40 caracteres de 5x7 pixels c/u.
El microcontrolador especializado puede ser el modelo **HITACHI 44780**, ó el modelo **HITACHI 44100**. También existen módulos LCD con IC's implantados directamente sobre el PCB (**POWERTIP**®).

Estos módulos poseen a través de estos CI's una interfese paralela para ser comandada desde un microcontrolador, microprocesador ó inclusive se puede realizar el control de este desde el puerto paralelo de un PC.

El microcontrolador y la pantalla de cristal líquido están colocados sobre un circuito impreso (PCB) y se interconectan con el mundo exterior (µC, µP o PP del PC) a través de un conector de 14 pines, el cual puede obtenerse en dos presentaciones: una línea y dos líneas teniendo ***la siguiente distribución***:

Conector de 1 línea por 14



Conector de 2 líneas por 7

Descripción de pines de los módulos LCD

Pin número	Símbolo	Función
1	Vss	Tierra ó Masa
2	Vdd	Alimentación + 5 VDC
3	Vo	Voltaje de ajuste de contraste
4	RS	Selección de Dato / Comando
5	R/W	Lectura / Escritura
6	E	Habilitador
7	DB0	1a línea de datos (LSB)
8	DB1	2a línea de datos
9	DB2	3a línea de datos
10	DB3	4a línea de datos
11	DB4	5a línea de datos
12	DB5	6a línea de datos
13	DB6	7a línea de datos
14	DB7	8a línea de datos (MSB)

En las siguientes notas de aplicación se muestra de manera más completa el debido uso y la explicación del manejo de módulos LCD:

Manejo de módulo LCD con µC **PIC** en PDF	00587b.PDF	**351KB**	
Código Manejo de módulo LCD con µC **PIC** (comprimido)	00587b.ZIP	**45KB**	
Interfase de un módulo LCD a µC **Motorola**	AN1745.PDF	**158KB**	

Bibliografia

Introducción a los microcontroladores, José Adolfo González V., McGraw Hill
Microcontroladores PIC, Tavernier, Editorial Paraninfo
Microcontroladores PIC, La solución en un solo chip, Angulo y otros, Editorial Paraninfo
Microcontroladores PIC, Diseño de aplicaciones, Angulo y otros, McGraw Hill
Cursos sobre Microcontroladores PIC, Niveles Básico y Avanzado, Tekcien Ltda.
Embedded Control Handbook, Microchip
PIC 16/17 microcontroller data Book, Microchip
MPASM assembler. User's Guide, Microchip
MPLAB IDE User's Guide, Microchip.
www.microchip.com