Curso de PICs para Estudiantes y Aficionados

Para un buen aprendizaje práctico, este manual se complementa con un CD y un video que le enseñan los fundamentos sobre microcontroladores PIC y lo guían paso a paso para que arme su propio cargador universal y aprenda a realizar sus primeros programas. Presentando este manual en cualquiera de nuestros distribuidores autorizados en América Latina (búsque el más cerca de su zona en **www.webelectronica.com.ar**) puede adquirirlos a precio promocional. El costo del CD + el video (presentando este manual) es el siguiente: Argentina: $20; México: $150; Otros Países: U$S18.

En Argentina llame al: (011) 4301-8804. En México llame al: (0155) 5787-1779

Arquitectura de los PICs

Bases Generales

Sepa qué es un PIC, cómo se compone y para qué puede utilizarlo.

¿Qué es un PIC?

Los circuitos integrados programables (Programmable Integrated Circuits = PIC) son componentes sumamente útiles en la Electrónica de Consumo. Aún cuando son conocidos desde hace más de veinte años, existen en la actualidad nuevos tipos que cumplen con una serie de requisitos y características sumamente útiles. Como una primera aproximación podemos definir a un PIC como "un chip que me permite obtener un circuito integrado a mi medida", es decir puedo hacer que el PIC se comporte como un procesador de luminancia o un temporizador o cualquier otro sistema mediante un programa que le grabo en una memoria ROM interna.

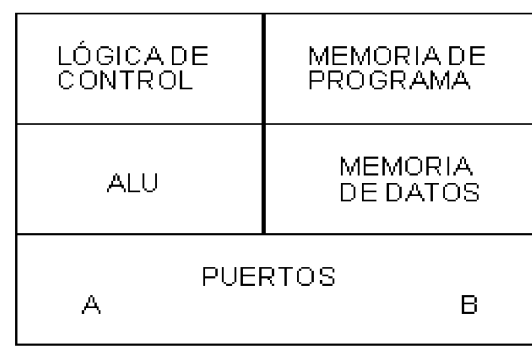
Los microcontroladores PIC son en el fondo procesadores similares a otros tipos, como por ejemplo la familia de los microprocesadores X86, 80486, Pentium y muchos otros que usan una arquitectura interna del tipo Von Neumann. En este tipo de arquitectura los datos y la memoria del programa se encuentran en el mismo espacio de direcciones.

En realidad un microprocesador y un microcontrolador no son la misma cosa. Los PICs son microcontroladores, es decir, una unidad que posee en su interior al microprocesador y a los elementos indispensables para que pueda funcionar como una minicomputadora en un solo chip.

Un microprocesador es solamente la unidad central de procesos o CPU, la memoria, los puertos y todos los demás periféricos son exteriores. La programación de un microprocesador es, por lo tanto, una tarea compleja porque deben controlarse todos estos dispositivos externos.

Un microcontrolador integra la CPU y todos los periféricos en un mismo chip. El programador se desentiende de una gran cantidad de dispositivos y se concentra en el programa de trabajo. Esta circunstancia da lugar a una gran pérdida de tiempo porque los datos tienen que ser retirados de la memoria y llevados a la CPU (Central Processor Unit) y viceversa. Esto significa que la computadora dedica la mayor parte del tiempo al transporte de datos de ida o de vuelta, en lugar de usar este tiempo para trabajar sobre los datos.

Los PICs emplean un conjunto de instrucciones del tipo RISC (Reduced Instruction Set Computer). Con el RISC se suele ejecutar la mayoría de las instrucciones con un solo pulso del clock. Con las instrucciones que se usan en otros equipos del tipo CISC (Complex Instruction Set Computer), se logran instrucciones más poderosas, pero a costa de varios ciclos del clock. En el bien conocido procesador 68HC11 de Motorola se requieren a veces hasta 5 ciclos del clock para ejecutar una instrucción.



ARQUITECTURA SIMPLIFICADA DEL PIC16F84

Figura 1

A los fines prácticos nos vamos a referir a los microcontroladores como bloques que poseen una memoria de

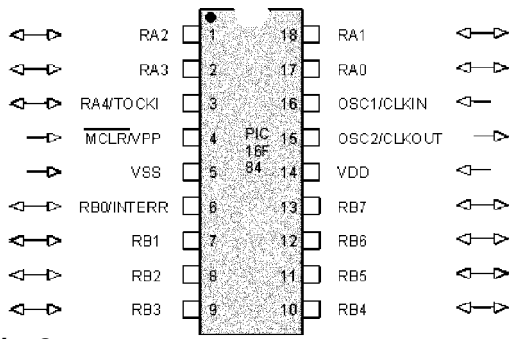


Fig. 2

programa que es el lugar donde deben alojarse los datos que le indiquen al chip qué es lo que debe hacer; una memoria de datos donde ingresen las señales que debe procesar el programa, una unida aritmética y lógica donde se desarrollen todas las tareas, una unidad de control que se encargue de supervisar todos los procesos y puertos de entrada y salida para que el PIC tenga contacto con el exterior (figura 1).

Un microcontrolador como cualquier circuito integrado analógico tiene entradas, salidas y algunos componentes exteriores necesarios para procesar las señales de entrada y convertirlas en las señales de salida (figura 2). El 16F84 requiere un cristal con dos capacitores y como mínimo un resistor para el reset. Por supuesto necesita una tensión de fuente de 5V (VDD) aplicada con respecto al terminal de masa (VSS). Posee dos puertos de salida, el A y el B, cuyos terminales son marcados RA0 al RA4 y RB0 al RB7. Estos puertos pueden ser programados como de entrada o de salida. El terminal 4 opera como reset pero también cumple funciones de carga de memoria de programa cuando es excitado con pulsos de 15V. El terminal RA4 (pata 3) también tiene funciones como entrada de un temporizador y RBO (pata 6) cumple también funciones como entrada de interrupción.

Ahora bien, la mayoría de los microcontroladores (sean de Microchip, o de National, Motorola, Philips, etc.) se comportan de forma similar, por ello nos vamos a referir a los microcontroladores PIC16F84 cuya arquitectura interna puede observarse en la figura 3.

Observe primero los bloques externos. Existe un cristal que se conecta en OSC1 y OSC2 para generar el CLOCK del sistema. Luego una señal de entrada llamada MCLR negada, que es un nombre de fantasía para nuestro conocido RESET (debido a que esa pata tiene un doble uso) y, por último, dos puertos paralelos de I/O (entrada o salida) llamados puerto A y puerto B. Una de las patas del puerto A puede ser utilizada como entrada de interrupciones (esta pata especial hace que el microprocesador deje de realizar la tarea que estaba ejecutando y pase a realizar otra tarea alternativa; cuando la termina vuelve a su programa original).

Analicemos el bloque más grande (temporizadores), en éste observamos un grupo de bloques dedicados a mejorar el funcionamiento pero sin influir directamente en el flujo de señales. Vemos un temporizador de encendido,

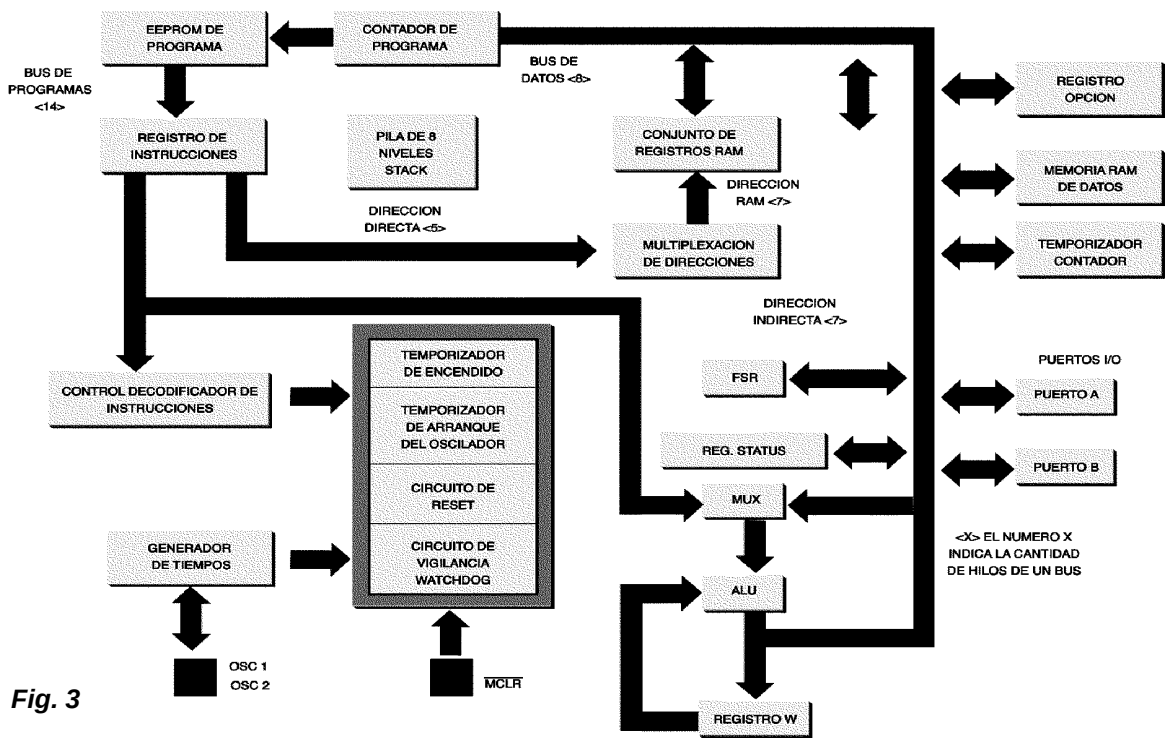


Fig. 3

un temporizador de arranque del oscilador de CLOCK, un circuito de reset y un circuito llamado de vigilancia o WATCHDOG. Los dos primeros bloques procuran un arranque ordenado para no producir una carga al mismo tiempo sobre la fuente. Por último, existe un circuito con un nombre curioso: "perro guardián". Su función es estar vigilante el máximo de tiempo que tarda el microprocesador en completar su programa (o mejor sería decir, la derivación más larga de su programa) y en caso de superarse ese tiempo, provocar un reset automático porque el microprocesador se quedó trabado en alguna parte de su programa. También se dice que el microprocesador se quedó colgado o congelado.

Este bloque de circuitos no trabaja independientemente sino que requiere conexiones al exterior y al interior del dispositivo. Por ejemplo, no siempre son utilizados y es el programa quien determina su utilización y además ajusta sus parámetros. Esto se realiza a través del bloque de control o decodificador de instrucciones.

Analicemos ahora la sección de arriba a la izquierda en donde observamos la memoria de programa, el contador de programa, el registro de instrucciones y la pila o STACK de 8 niveles. Cuando hablamos de registros nos referimos a pequeñas unidades de memoria transitoria, construida por lo general con un registro de desplazamiento como los analizados en "el rey micro". Son memorias volátiles que se utilizan para guardar información por un tiempo mínimo, con el fin de realizar una operación compleja de varios pasos.

El contador de programa es el responsable de que el microprocesador vaya analizando las instrucciones en orden ascendente. Este guarda el número de instrucción en el STACK y la instrucción misma la pasa al registro de instrucciones desde donde se envía al resto del microprocesador. El STACK es, en realidad, una pila de registros (en nuestro ejemplo hay 8), debido a que el programa puede tener derivaciones (en la jerga LOOPS, rulos o subprogramas). Cuando se termina de ejecutar un loop se debe volver al mismo punto del programa en donde se había producido la bifurcación y eso es posible porque ese número de instrucción quedó guardado en uno de los registros de la pila. Es común que un loop tenga, a su vez, un loop secundario y cuando se ejecuta ese loop secundario se debe volver al mismo punto del loop primario, eso se consigue con guardar ese número de instrucción del loop secundario en otro registro de la pila.

Analicemos ahora la sección inferior derecha. En ese sector se ubican los bloques responsables de efectuar operaciones matemáticas y lógicas binarias; recordemos que el nombre ALU proviene de Aritmetic Logic Unite (unidad arimética y lógica). En este sector es imprescindible utilizar un registro ya que una operación aritmética o lógica siempre se efectúa entre dos números. Los números binarios que deben procesarse se toman de la memoria de datos, el primero se acumula en el registro de trabajo o registro W (de Work = trabajo) el segundo es el presente en el instante en que se invoca la memoria de datos. Como las operaciones pueden ser encadenadas (cuando el resultado sirve como operando de la siguiente operación, tal como el caso de un producto) el registro W tiene un retorno a la ALU.

Vemos además que la ALU está comandada por el bloque MUX (MUltipleXador). En efecto, la ALU requiere que se le envíen números para procesar que le lleguen desde la memoria de datos, pero antes se la debe predisponer para que efectúe la operación requerida (comparación, rotación de dígitos, etc.).

El registro de estado o estatus colabora durante las operaciones matemáticas. Piense cómo opera Ud. para realizar una resta: primero ubica el primer número, luego el segundo y después comienza a analizar los bits menos significativos (las unidades), pero si el número de arriba es menor que el número de abajo, entonces toma prestado de la columna de las decenas, luego debe recordar esto porque el número de arriba en la columna de las decenas se redujo en una unidad. En realidad, aunque se trate de una operación entre dos números su ejecución requiere guardar lo que se llama acarreo en otro registro y éste no es otra cosa más que el registro STATUS.

El PIC16C84 contiene además de todo lo visto una memoria RAM de registros que puede ser llamada desde el registro de instrucción a través de un multiplexador de direcciones. Esta sección sólo se utiliza en desarrollos avanzados. Ahora bien, un microcontrolador sin programa no sabe hacer nada, es como un niño recién nacido; tiene algunos reflejos condicionados como el de succión que le permite alimentarse pero no sabe hacer más que eso. Nosotros deberemos enseñarle a realizar acciones y lo vamos a hacer como a un bebé, paso a paso. Su capacidad de aprendizaje es enorme y sumamente variada. Le vamos a enseñar a llorar a intervalos regulares, a encender luces, a sumar, a restar, etc.

Enseñarle significa programarlo y eso se realiza con una plaqueta de programación que depende de cada marca y modelo de microcontrolador. Antiguamente los microprocesadores tenían una ventanita transparente y era necesario exponerlos a la luz ultravioleta para borrar su memoria e introducir un programa nuevo. Actualmente cuentan con memorias que no tienen este requisito. Basta con cargarlos con un programa para que se borre el

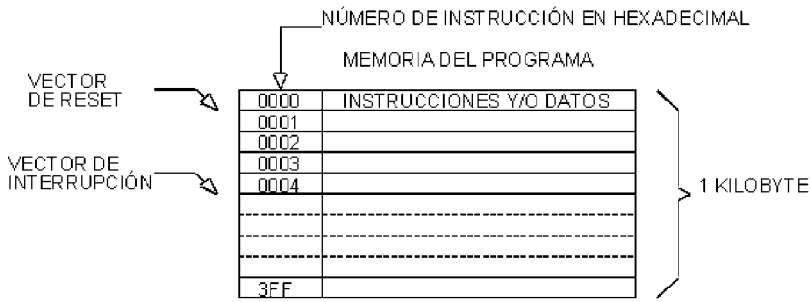


Fig. 4 MAPA DE MEMORIA DE PROGRAMA DEL PIC16F84

anterior. Esto significa que, con el mismo integrado, podremos construir diferentes dispositivos que realicen funciones totalmente distintas unas de otras.

La Memoria de Programa

Es una memoria EEPROM, es decir, de lectura solamente (ROM = Read Only Memory) que se programa por tensión (no necesita luz ultravioleta); es decir que basta con introducir los datos con cierto nivel de tensión para que éstos borren el programa anterior y graben uno nuevo.

¿Por qué esta memoria se llama ROM, si se pueden grabar datos sobre ella?

Se llama ROM porque para grabarla se debe conectar el PIC al programador; luego de que el PIC coloca estos datos en la plaqueta del dispositivo, sólo pueden ser leídos, ya que entonces forman el programa del PIC.

Esta memoria (figura 4) tiene una longitud de 1 Kbyte con palabras de 14 bits. Digamos que tiene un ancho de 14 bits y una altura de 1.000 Bytes o que es una memoria de 1.000 x 14. Observe que los números de instrucción en hexadecimal van desde el 000 al 3FF, lo cual implica que existen 1.040 posiciones de memoria, valor obtenido empleando la fórmula:

$$3 \times 16^2 + 16 \times 16^1 + 16 \times 16^0$$

Observe que dos de las posiciones de memoria tienen las indicaciones “vector de reset” y “vector de interrupción”.

Eso significa que, cuando se provoca un reset, el microprocesador vuelve a la posición 000 del programa y cuando se produce una interrupción, a la posición 004. Estos retornos forzados deben ser considerados al diseñar el programa del microprocesador; es decir que el reset se produce porque la señal externa pone el contador de programa en 000 y todo el programa se reinicia. En cambio, cuando ingresa una señal por la pata de interrupción el contador de programa va a 004 y la acción que, en general ocurre, es que se comienza a leer un subprograma particular. Cuando este subprograma termina, el contador de programa recupera el número que tenía en el momento de arribar la interrupción.

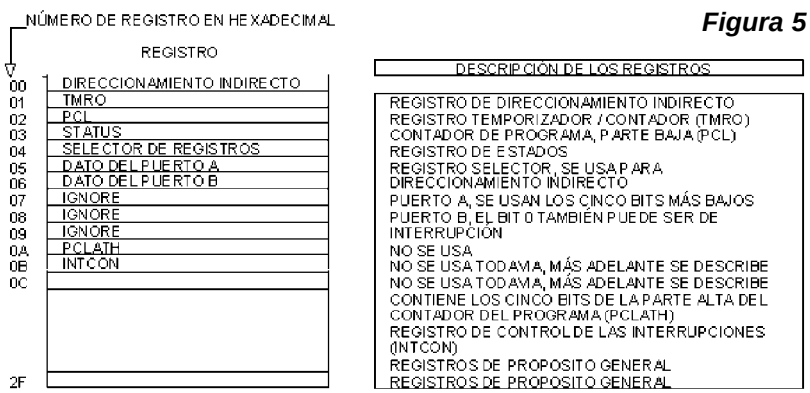
La Memoria de Datos

La RAM (Random Acces Memory = memoria de acceso aleatorio, figura 5) es una memoria de lectura y escritura de 128 posiciones pero que sólo tiene implementados las primeras 48 posiciones (desde 00 a 2F en hexadecimal). De estos 48 registros, los primeros 12 son fijos y cumplen un propósito determinado, en tanto que desde el 13 hasta el 48 son registros de propósito general, en donde el programa puede indicar que se almacene un dato para ser tomado más tarde.

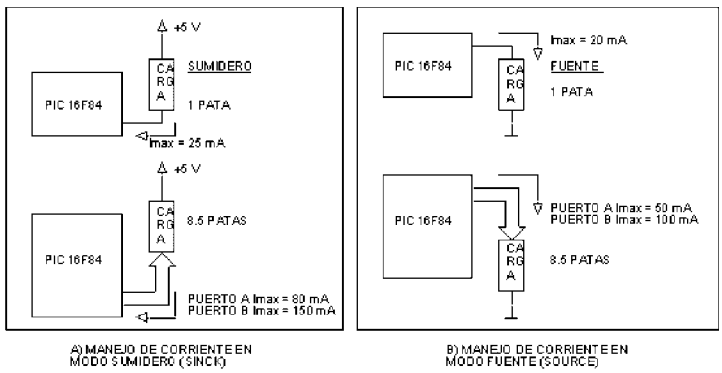
Los Puertos del PIC

El PIC16C84 tiene dos puertos paralelos de entrada

Figura 5



Memoria de Datos del PIC16F84 de 64 bytes, de los cuales no tiene disponibles las primeras 12 posiciones



MANEJO DE CORRIENTE DE LOS PUERTOS

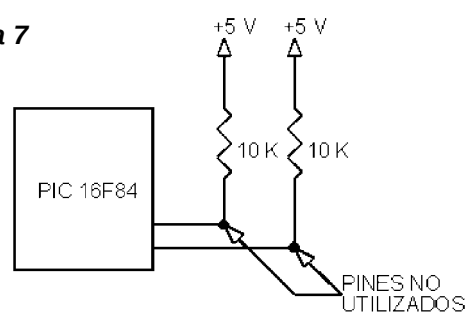
Fig. 6

o salida: el puerto "A" de 8 patas y el "B" de 5 patas. Cada pata puede ser predispuesta por el programa para operar como de entrada o de salida. Cada pata tiene un resistor de pull-up (resistor conectado a fuente) interno que puede ser desconectado mediante el programa. Estos resistores se desconectan automáticamente si una pata se predispone como pata de salida debido a que las salidas ya tienen posibilidad de entregar corriente desde fuente con un transistor. Todos los resistores de pull-up se conectan o desconectan al mismo tiempo (no existe un comando que los conecte independientemente).

Como puerto de salida, una pata puede tomar 25mA del circuito o entregar 20mA al mismo, sin embargo, en el puerto "A" sólo se pueden consumir 80mA en total o entregar 50mA, esto significa que sólo algunas patas pueden trabajar al máximo porque si todas lo hicieran (y son 8) el consumo total sería de 25 x 8 = 200mA. El puerto "B" tiene otras características máximas, ya que en total puede tomar 150mA o entregar 100mA. Como vemos, las salidas admiten suficiente carga como para alimentar directamente a un led (figura 6).

Los puertos no utilizados siempre se deben conectar a la fuente de 5V a través de un resistor de 10kΩ debido a que se trata de un dispositivo CMOS que, de otro modo, podría deteriorarse por captación electrostática (figura 7).

Figura 7



CONEXION DE PUERTOS NO UTILIZADOS

La pata 3 perteneciente al puerto "A" puede ser configurada como de entrada-/salida o como de arranque de un temporizador/contador (figura 8). Cuando se programa como entrada esta pata funciona como un disparador de SCHMITT o Schmitt trigger ideal para reconocer señales distorsionadas o con crecimiento lento. Esta misma pata también tiene una característica distinta cuando opera como salida. Ella es la única que trabaja a colector abierto, es decir, que no puede emplearse como fuente, en este caso siempre se utilizará un resistor externo.

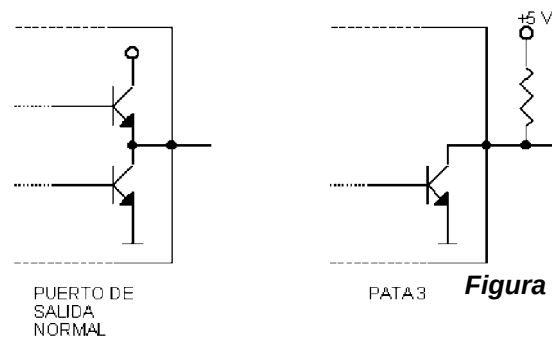


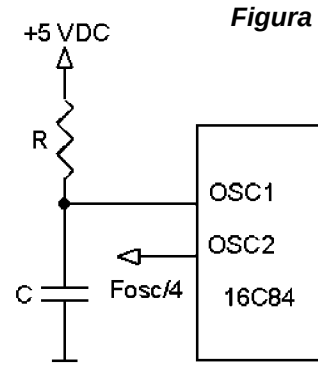
Figura 8

CARACTERISTICAS ESPECIALES DE LA PATA 3

El Clock

Los PIC poseen un oscilador configurable por programa de características muy amplias. Cuando no se requiere mucha precisión se puede trabajar con un oscilador a RC conectado según la figura 9.

Figura 9



CLOCK A RC

Para circuitos que requieran una gran precisión se puede trabajar con un cristal de frecuencia baja, media o alta (figura 10). Como máximo el PIC16C84 puede trabajar con un cristal de 10MHz. Internamente la frecuencia del cristal se divide por 4, por lo tanto, es muy común la utilización de un cristal de 4MHz para obtener un CLOCK interno de 1MHz que garantiza que cada instrucción dure exactamente 1mS. Para temporizadores de período largo se utilizan cristales de baja frecuencia.

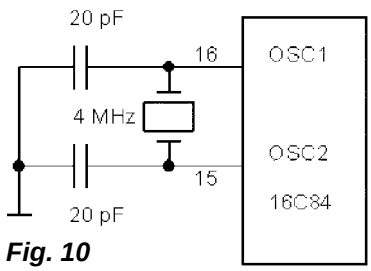


Fig. 10

CLOCK A CRISTAL

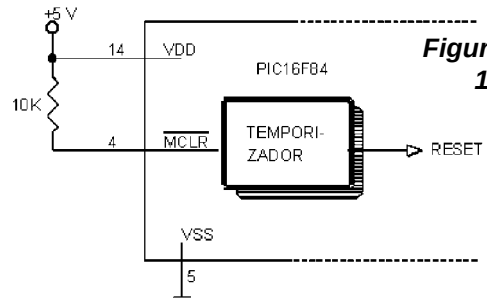


Figura 11

RESET AUTOMATICO

El Reset

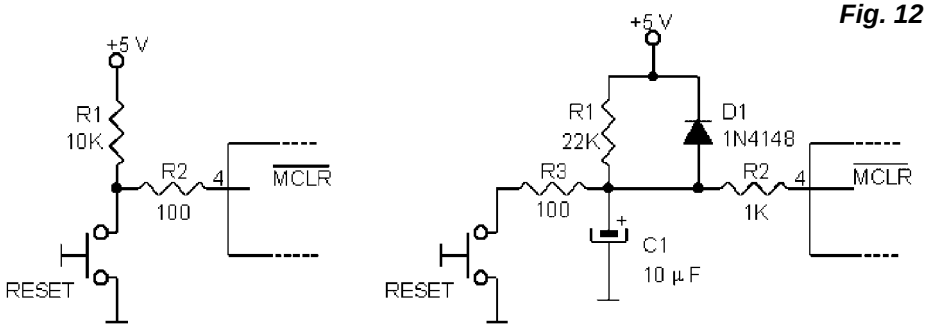
El PIC "se resetea" cuando la pata 4 (MCLR negada) se pone a potencial bajo. Para simplificar el circuito de reset el PIC posee un temporizador interno que permite realizar un reset automático cuando se aplica tensión de 5V. En estos casos el circuito externo de reset sólo implica el uso de un resistor de 10kΩ entre la pata 4 y fuente tal como se muestra en la figura 11.

En muchos circuitos es necesario realizar un reset manual y para ello existen dos posibilidades, una es utilizar sólo el temporizador interno (por programa) y la otra es agregar una constante de tiempo exterior como se muestra en la figura 12. En el segundo circuito C1 provee un retardo al encendido o posterior al pulsado de reset porque C1 se cargará lentamente a través de R1 con una constante de tiempo de 22kΩ x 10µF = 220mS.

El resistor R3 limita la corriente de descarga de C1 a valores compatibles con sus características de corriente de pico máxima.

D1 descarga a C1 cuando la tensión de fuente decae para permitir un reset inmediato cuando la fuente se apaga y se enciende en rápida sucesión. R2 limita la corriente de reset, tomada desde el microprocesador. Este segundo sistema se suele utilizar cuando se requiere un reseteado remoto a través de varios metros de cable que podrían captar zumbido (C1 reduce la impedancia del circuito de reset).

Fig. 12



DOS CIRCUITOS DE RESET MANUAL

Programas Cargadores de PICs

Hardware y Software para Cargar Microcontroladores

Tenga circuitos gratuitos para cargar programas en la memoria de un PIC.

INTRODUCCIÓN

Para que un PIC funcione como nosotros queremos es preciso "cargarle" un programa en la "memoria de programa". El programa debe estar en un lenguaje comprensible por el chip o sea un lenguaje de máquina en código binario aunque en realidad al PIC debemos introducirle un programa en código hexadecimal e internamente lo convierte a binario para realizar su interpretación.

El programa que debemos cargarle al PIC en código hexadecimal tiene que tener la terminación "hex".

Sin embargo, normalmente se debe escribir "editar" el

programa en un lenguaje entendible por el usuario, normalmente conocido como "assembler". Los PIC basan su programación en un set reducido de instrucciones que permiten editar el programa generando un archivo con extensión "asm".

Existen programas que se encargan de "traducir" el programa escrito en lenguaje assembler (asm) a programa escrito en lenguaje de máquina (hex); a estos programas se los denomina traductores o ensambladores.

Por lo tanto, para escribir un programa y cargarlo en un PIC precisamos:

• Un utilitario o programa que nos permita escribir el juego de sentencias a cargar en la memoria de programa del PIC. Este utilitario es un "editor" y como ejemplo tenemos el MPLAB.

• Un utilitario o programa que nos permita "traducir" el programa escrito en assembler a lenguaje de máquina para que podamos cargarlo en el PIC. A ese programa se lo denomina "ensamblador" y como ejemplo tenemos al MPASM.

• Un utilitario para "cargar" el archivo que tiene el programa en lenguaje de máquina (archivo con extensión hex) en la memoria del PIC. A estos programas se los denomina "cargadores" y como ejemplo tenemos al NOPPP, al PIC PRO y a tantos otros, muchos de los cuales mencionaremos más adelante.

Pero también existen otros utilitarios que permiten "simular" y "emular" un programa para ver cómo se comporta antes de cargarlo en el PIC.

El simulador permite detectar errores en el programa (el MPLAB permite simular) para poder corregirlos. El emulador permite que "una computadora se comporte como PIC" siendo las patas del puerto, las correspondientes a las del PIC.

Esto quiere decir que si queremos hacer circuitos con PICs, para poder predisponerlo adecuadamente precisaremos:

• Programa MPLAB: Para editar (escribir) un programa en lenguaje assembler y así generar el archivo con extensión asm. El MPLAB también me permite "simular" el funcionamiento del programa para detectar errores y corregirlos.

• Programa MPASM: Para "ensamblar" o convertir el archivo asm en archivo hex con el objeto de cargarlo en el PIC.

• Programa NOPPP: O cualquier otro cargador que permita ingresar el archivo con extensión hex en la memoria de programa del PIC. También se puede emplear cualquier otro cargador, hasta iuncluso se podrían cargar los datos manualmente, tal como veremos oportunamente. Todos los programas cargadores precisarán de un circuito "eléctrico" o hardware para poder realizar la función de carga.

Los Cargadores de PICs

Dijimos que un cargador de PIC debe poseer por un lado el circuito que permita cargar el programa desde una computadora (hardware) y el programa que permita manejar a dicho circuito para realizar la operación de carga (software). El circuito debe ser muy simple.

Vamos a suponer que no tenemos un dispositivo programador a nuestra disposición; es decir que deberemos armar también un dispositivo programador de nuestros PICs que se conecte al puerto paralelo de una PC (el puerto de impresora).

Nuestra intención es que construya un dispositivo para programar PIC que sea muy sencillo de modo que se pueda armar en el mismo protoboard y que se conectará a la PC por intermedio de una manguera.

Como todos sabemos, un dispositivo para una PC (hardware), siempre necesita un programa que lo controle (software). Nuestra intención es que Ud. no compre nada más que lo imprescindible; así que vamos a usar un software gratuito llamado NOPPP que puede bajarse por Internet. Ud puede bajar este programa, el MPLAB, el MPASM y otros programas y utilitarios, siguiendo los pasos dados en la página de contenidos especiales de Saber Electrónica y, lo que es más importante, en forma totalmente **GRATUITA.**

Para ello diríjase a: **www.webelectronica.com.ar** (si no tiene Internet o no sabe cómo se hace, lleve este escrito a cualquier cybercafé y pídale al encargado que le baje los programas). Haga doble click sobre el ícono password y luego digite la clave **aiwa15**.

Recuerde que cuando decimos "programar" nos estamos refiriendo a diseñar un programa para un PIC y a la persona que realizó ese trabajo la llamamos "programador"; cuando decimos "cargar" nos estaremos refiriendo a llenar con datos la memoria de un PIC y el dispositivo que realiza esta función lo llamaremos "cargador de PICs". Aunque parezca increíble nuestro amplio idioma no tiene una palabra precisa que diferenciara a la acción del aparato y se suscitaban graves confusiones al leer, por algo tan simple. En realidad, sí estaba acuñado un término práctico para el dispositivo: "quemador de PICs", pero el

mismo sólo se puede aplicar a aquellos PICs que sólo se pueden grabar una sola vez. Por otro lado, el término se puede confundir con un dispositivo que funcionando mal quema a los PICs en lugar de cargarlos, tal como el autor mencionará con respecto a un programador comercial.

Navegando por Internet es posible encontrar gran cantidad de programas cargadores de PICs, nosotros haremos referencia a los cargadores:

* ***Cargador NOPPP***
* ***Cargador PROPIC***
* ***Cargador Universal de PICs y EEPROM***
* ***Cargador PIC800***
* ***Cargador PP84***

Obviamente, todos estos circuitos puede "bajarlos libremente" pudiendo tener más información y los enlaces a las páginas web de cada uno de estos cargadores en la página de contenidos especiales que hemos dado anteriormente. Estos cargadores sirven para ser utilizados en diferentes tipos de PICs a saber:

NOPPP *es un programa de uso libre ampliamente difundido que sirve para cargar PICs del tipo 16C84, 16F83 y 16F84, pero también se dán otros cargadores con sus respectivos programas, que sirven para otros componentes.*

PICPRO *es un programador avanzado en relación con el NOPPP y sirve para ser montado cuando no se quieren correr riesgos con la PC que se está utilizando para cargar PICs y cuando se desea tener un trabajo libre de errores (el NOPPP es un programa para aprendizaje).*

El Programador Universal *sirve para programar: 12C508, 12C509, 12C67X, 16C55X, 16C61, 16C62X, 16C71, 16C71X, 16C8X, 16F8X, 16F873/4/6/7, 16C73/74/76/77. También programa memorias EEPROM de la serie 24LCXX.*

PIC800 *es un programador para 16C84, 16F84, 16F873/74/76/77.*

PP84 *es un programador sencillo del tipo NOPPP de buenas prestaciones.*

Un circuito con PIC siempre se compone de dos secciones. El circuito del dispositivo deseado y los necesarios componentes periféricos imprescindibles para que el PIC funcione, a saber clock y reset (la primer sección cambia con cada proyecto, en tanto que la segunda puede ser común a muchos proyectos). Ambos circuitos periféricos tienen diferentes opciones: el clock puede ser a cristal cuando se requiera precisión, o a RC (como en nuestro caso) cuando sólo se requiere que la secuencia de encendido tenga un tiempo aproximado. En cuanto al reset, éste puede ser externo con un pulsador o interno y automático, cada vez que se conecta la alimentación de 5V.

Una vez definidas estas cosas se puede dibujar el circuito completo, tal como lo mostramos en la figura 13.

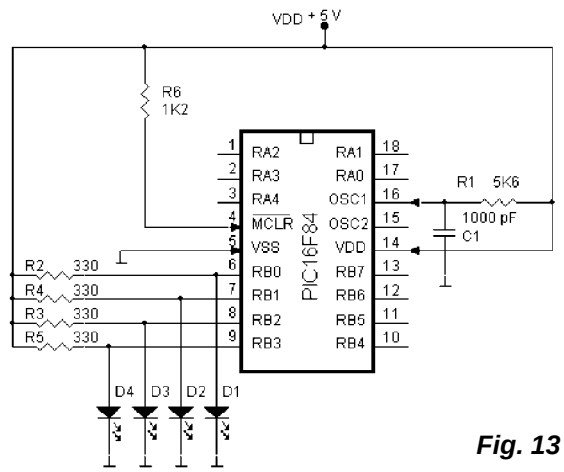


Fig. 13

Observemos primero la sección periférica imprescindible. El clock está generado por un oscilador RC realizado con el PIC, R1 y C1 (elegidos para obtener una secuencia completa que dure un segundo aproximadamente), en tanto que el reset se realiza en forma automática, cada vez que se enciende el equipo, por intermedio de R6. Nuestro circuito de aplicación no puede ser más sencillo, sólo utilizamos cuatro LEDs sobre las patas RA0 a RA3 (puerto "A" tomado parcialmente) y sus correspondientes resistores de pull up (resistores a fuente). Nada más simple y fácil de armar en un panel protoboard.

¿Qué ocurriría si compro los componentes, los armo y pruebo directamente el circuito?

No va a funcionar porque como el PIC no está instruido para realizar ninguna función y los LEDs van a permanecer todos encendidos. El PIC tiene una memoria interna que debe cargarse con números binarios de 8 dígitos; exactamente tiene 48 posiciones de memoria (numeradas desde el 0 hasta el 2F en números hexadecimales), en donde se le indicará qué tarea debe realizar o qué datos fijos debe cargar para ejecutar esas tareas.

En nuestro caso activar transistores internos en algunas patas del puerto "A" (predispuestas como salidas) de modo de generar cortocircuitos momentáneos a masa en forma secuencial.

El dispositivo con un zócalo para conectar el PIC e instruirlo (cargarle los datos en la memoria de programna) se llama genéricamente “Programador de PICs”, pero nosotros en este artículo convenimos en llamarlo “Cargador de PICs” y es como un apéndice de nuestra PC, conectado con un cable al puerto paralelo de la misma.

Si la PC tiene dos puertos paralelos de salida se usará uno para la impresora y el otro para nuestro programador, si sólo tiene uno, se desconectará provisoriamente la impresora para conectar el programador, o mejor aún, se conectarán ambos dispositivos a través de una caja selectora que se consigue en los negocios de computadoras y se usa para conectar dos impresoras a la misma PC.

Si Ud. tiene un mínimo conocimiento sobre computadoras, sabrá que un dispositivo conectado a la PC es totalmente inútil si no está acompañado de un programa instalado en el disco rígido de la misma. En algunos casos se necesitan dos programas, a saber: el driver del dispositivo y un programa de aplicación que utilice dicho dispositivo. En el caso de dispositivos que se conectan en el puerto paralelo, el programa driver no es necesario porque dicho puerto ya está debidamente habilitado para usar la impresora. Lo que sí se requiere obligatoriamente, es un software de aplicación del programador que suele proveerlo el fabricante del mismo.

Programadores y software de aplicación de los mismos hay muchos. Algunos son muy simples y económicos (tan económicos que muchos se entregan gratuitamente por Internet, es decir que el autor regala el software y da las explicaciones para armar el hardware) y otros son muy complejos y caros. La diferencia entre unos y otros suele ser la posibilidad de aceptar más tipos de PICs (además del 16C84 y 16F84 existen muchos otros) e inclusive microprocesadores o memorias de otras marcas. También se diferencian en la velocidad a la cual cargan el PIC; los hay de alta productividad que trabajan con un elevado flujo de datos y otros más lentos que sólo sirven para tareas de aprendizaje.

En nuestro caso vamos a trabajar con un software que se llama NOPPP, que es absolutamente gratuito y se puede bajar desde Internet. Este software está previsto para ser usado sólo con los PIC16C84, PIC16C83 y PIC16F84 que son los más utilizados.

Con respecto al hardware, Ud. puede usar el que propone el autor del software que es muy sencillo o usar otro que le propongo yo más adelante y que tiene algunas ventajas con respecto a una mejor forma de los pulsos de programación. El NOPPP es suficientemente sencillo como para implementarlo en un panel protoboard, pero como es un hardware que deberemos usar más adelante para programar otros PICs conviene armarlo en forma más definitiva utilizando una plaqueta de circuito impreso o una plaqueta ojalillada. Si Ud. sólo desea armar el circuito de este artículo puede armar tanto el cargador como el circuito de aplicación en dos lugares distintos del mismo protoboard.

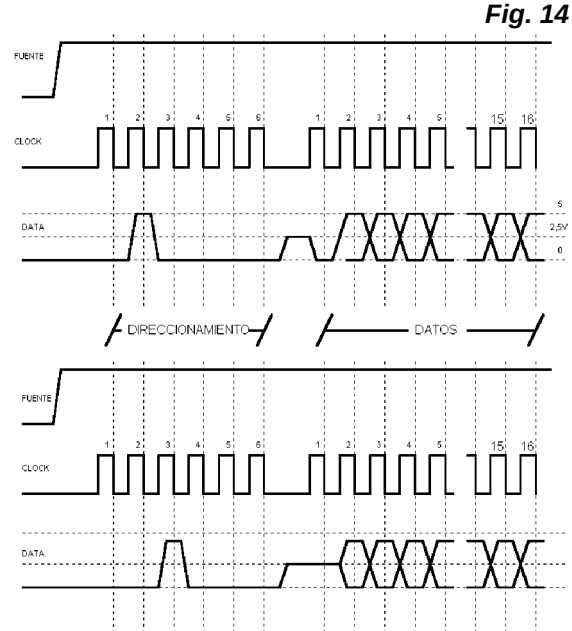
El PIC, como una memoria, tiene una pata que predispone el dispositivo para leer o para escribir. Si la pata 4 del PIC está a un potencial comprendido entre 13 y 14 V, el PIC está preparado para escribir los datos que provienen de la PC. Si la pata 4 está por debajo de 6V, el dispositivo está previsto para ser leído. Los datos a leer o escribir se ponen/obtienen de la pata 13 del PIC con la pata 12 del mismo que opera como clock.

Aquellos que conocen el proceso de carga de datos en una memoria serie, no tendrán mayores inconveniente en entender la frase anterior. Para aquellos que no conocen el proceso explicamos lo siguiente:

Un PIC se lee/escribe accediendo a las diferentes posiciones de memoria por la misma pata por la que se obtienen/ingresan los datos (la pata 13). La señal primero elige la posición de memoria a ser leída/escrita, y luego que esa posición está accesible se escriben/leen los datos. El clock que se coloca en la pata 12 sirve para indicar en qué momento se debe transferir la información. Los datos pueden estar sobre la pata 13 todo el tiempo que Ud. desee, ya que no serán ni leídos ni escritos por la PC hasta que se produzca un cambio de estado (de alto para bajo) en la pata 12.

En la figura 14 mostramos las formas de señal indica-

Fig. 14



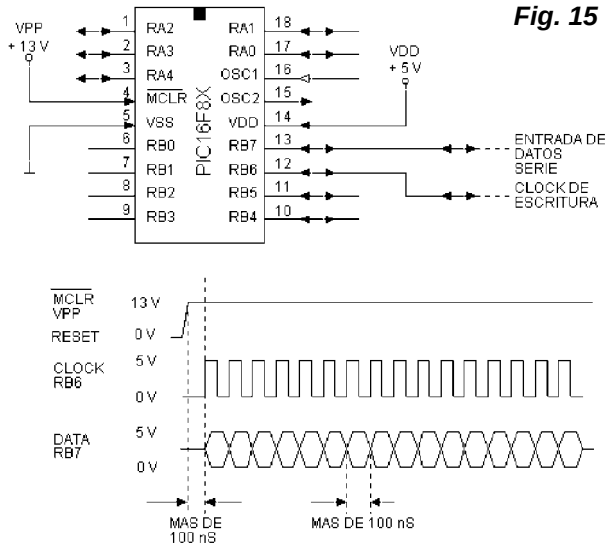


Fig. 15

das por Microchip para leer/grabar un PIC.

El lector observará que dentro del comando existe una primera sección de 6 bits que indica la posición de memoria a ser leída/escrita y luego la línea de datos permanece en estado de alta impedancia por un intervalo de tiempo que depende de la operación (durante ese tiempo el clock está apagado). Por último ingresan/egresan los datos. Las conexiones de fuente y las señales deben ser aplicadas según una secuencia predeterminada que debe respetarse invariablemente. Los nombres de las señales se aclaran en la figura 15.

A) El primer paso es colocar el PIC en el zócalo del programador con señales y fuentes a potencial de masa.

B) Levantar la tensión de fuente VDD a un potencial de 5V ± 0,2V por la pata 14 (VDD).

C) Levantar la tensión de fuente VPP a un potencial de 13V ± 0,3V por la pata 4 (MCLR NEGADO).

D) Esperar en esas condiciones un tiempo superior a 1mS.

E) Posicionar el primer dato en la pata 13 (RB7) con un potencial alto (mayor a 4 V) o bajo (menor a 1V).

F) Cuando la pata 12 (RB6) pase a un estado bajo, inferior a 1V, el dato se carga en la memoria.

G) Continuar cargando los datos con el mismo criterio a un ritmo tal que el dato este presente por lo menos durante 100nS.

H) Cuando todos los datos fueron cargados se debe esperar 1 segundo.

I) Desconectar la fuente de 13V.

J) Desconectar la fuente de 5V.

K) Retirar el micro grabado.

Por sobre todas las cosas debe respetarse el criterio de no sacar un PIC de su zócalo con las fuentes conectadas, porque puede desprogramarse o, peor aún, dañarse definitivamente. También es muy importante respetar las tensiones de fuente y no demorar la llegada de datos, luego conectar VPP (porque podrían cargarse datos falsos por interferencias electromagnéticas).

Como usted puede observar de la descripción anterior la entrada RB6 es utilizada como clock y la RB7 como entrada de datos en una clásica operación de transferencia de datos en serie.

Observe que hay un tiempo entre el momento en que MCLR NEGADO sale de la condición de reset hasta que aparece el primer dato o el primer pulso de clock. También hay un tiempo mínimo para la permanencia de un dato en la entrada (la representación usada para un dato significa que el mismo puede ser alto o bajo, depende del bit que se esté grabando). Lo que no hay es un tiempo máximo pero evidentemente cuando mayor es este tiempo más lenta será la carga del programa.

El tiempo más adecuado depende de factores tales como el largo del cable utilizado para conectar el programador a la PC. Si el cable es largo, los pulsos tienden a deformarse y atenuarse, sobre todo cuando son de corta duración (100nS equivalen a una frecuencia de 10MHz). Para evitar problemas, todos los programadores trabajan a velocidades inferiores a la máxima, sobre todo considerando que la capacidad de memoria no es muy grande y para el uso no comercial no son imprescindibles grandes velocidades de grabación.

Un cargador de PIC debe permitir también una operación de verificación para cuando el usuario tiene dudas respecto de la condición de un determinado PIC (vacío o lleno). Solo que el programa almacenado debe ser inviolable en caso de que la persona que lo cargó así lo haya dispuesto. Esto se llama predisponer el PIC y no sólo se lo predispone para hacerlo inviolable, sino que además se determinan otros importantes parámetros de funcionamiento como por ejemplo que esté predispuesto para un clock a RC o que contenga un temporizador de reset interno.

Las predisposiciones no forman parte del programa del PIC aunque pueden incluirse junto con éste. Algunos cargadores de PICs preguntan sobre la predisposición antes de grabar el PIC. La pregunta sobre la predisposición deseada aparece en la pantalla de la PC y uno elige de un menú de opciones. En otros casos, las predisposiciones se escriben antes del programa en una secuencia

perfectamente predeterminada que debe respetarse a ultranza. Las predisposiciones son tres y las vamos a analizar por orden:

a) La prohibición de lectura. El PIC se programa pero el programa no puede ser leído. Cada tanto aparece información en Internet sobre alguna empresa dedicada a leer PICs protegidos, sin embargo, aún no he conseguido leer ningún PIC protegido. Aclaremos que el programa protegido no se puede leer, pero el PIC se puede volver a grabar. El autor escuchó también una especie que dice que el PIC16Fxx fue creado por una supuesta filtración en el secreto del programa de los PIC16Cxx, pero el autor no tiene pruebas de la certeza de esta noticia.

b) Habilitación del timer de reset. En el pinup del PIC se puede observar que la pata llamada MCLR negado (4) tiene un doble uso. En principio sirve para predisponer el PIC en grabación o lectura, como ya fuera comentado; pero cuando su tensión cae por debajo de 1V, el PIC se resetea y comienza la lectura del programa por el principio. Este tipo de reset se llama reset a pulsador y se utiliza para provocar alguna acción como, por ejemplo, comenzar una secuencia de encendidos de LEDS o encender un LED por un tiempo predeterminado. Este modo de funcionamiento requiere una acción externa de reset (apretar un pulsador). La misma acción de reset se puede conseguir en forma automática cada vez que se conecta la fuente de 5V. Este modo de trabajar sólo requiere que la pata MCLR (pata 4) esté permanentemente conectada a 5V (con un puente o un resistor de 1kΩ) y que el PIC esté predispuesto con el Power-up Timer en ON (temporizador de encendido conectado). Cuando el temporizador está habilitado, luego de conectar la fuente de 5V, comienza un conteo interno (que dura 72ms) durante el cual el PIC pone todas sus compuertas internas en cero (reset) y recién después comienza a efectuar los pasos que le indica el programa. A propósito, el nombre MCLR proviene de MASTER CLEAR (literalmente, limpiador maestro).

c) Circuito de vigilancia (perro guardián). En realidad el PIC tiene un reset extra que opera luego de un cierto tiempo si no se realiza la operación final del programa. Es como una especie de rutina automática de descongelamiento que opera en caso de falla del programa. Eventualmente puede ingresar algún pulso de energía al sistema que envíe al programa a un loop (rizo) eterno (una derivación o camino cerrado sobre sí mismo). Una vez que el programa ingresó en ese camino no puede salir de él salvo que se produzca un reset. Si pasa un tiempo considerable sin que se llegue a la última sentencia del programa, el PIC analiza su predisposición y si el Watchdog timer (literalmente perro guardián) está habilitado provoca un reset programado. En los programas más simples y donde se producen loops infinitos a propósito (de los que sólo se sale tocando un pulsador), se suele dejar el perro guardián desconectado para evitar un funcionamiento errático. Cada cargador de programas tiene un software diferente y esto implica que las predisposiciones anteriores, también llamadas fusibles de predisposición se accionen con sentencias diferentes que serán explicadas más adelante.

EL CARGADOR NOPPP

El NOPPP es un software para un cargador muy simple y efectivo. Realmente no tiene defectos importantes y nos permite cargar los PIC 16C83, 16C84 y 16F84 en forma muy económica ya que el hardware correspondiente sólo requiere dos diodos rápidos de señal, un BC548 y 4 resistores, además de una fuente regulada doble de 13 y 5V y un conector para el puerto paralelo de la PC. Más adelante veremos el circuito completo pero para entender el funcionamiento del cargador de programas nada mejor que el circuito simplificado de la figura 16. A propósito, el nombre del software proviene de las iniciales de NO Piece Programer Pic, es decir: programador para PIC sin piezas en alusión a los muy pocos componentes que requiere. Las patas de comunicación con el puerto paralelo de la PC están indicadas como 1J1, 2J1, etc, ya con J1 designamos al conector de entrada y el número inicial indi-

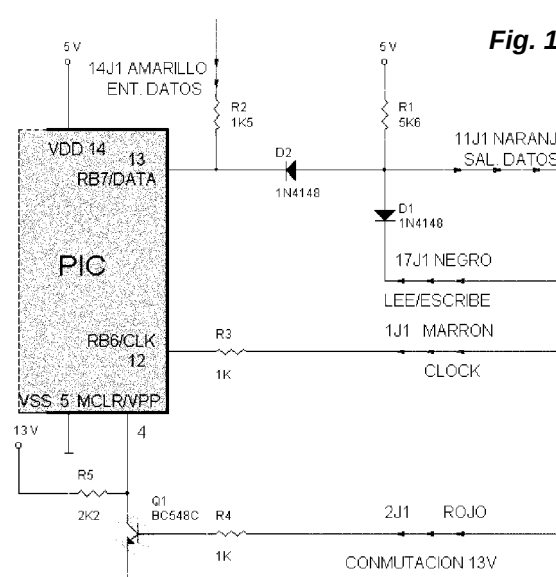


Fig. 16

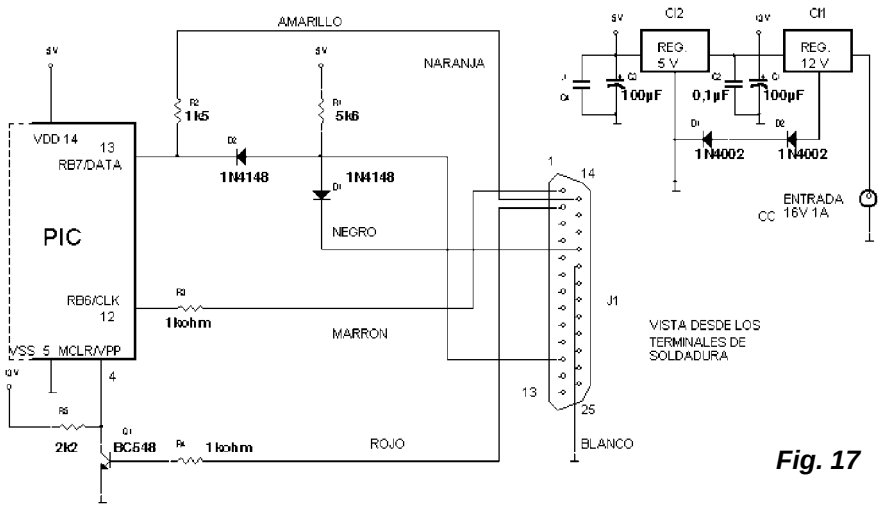


Fig. 17

garse completo.

Ahora que conocemos el funcionamiento del cargador de programas NOPPP vamos a completar el circuito con una fuente de alimentación y el conector para PC (figura 17).

Observe que las fuentes están conformadas por dos reguladores de 5 y de 12V pero, como necesitamos un regulador de 13V, realizamos una pequeña modificación en el regulador de 12V; agregando los diodos D1 y D2 la tensión de salida se incrementa a un valor de 13,2V. El otro cambio importante es el agregado de capacitores de filtro que, como Ud. observa, siempre se ubican de a dos: un electrolítico para filtrar las bajas frecuencias y un cerámico disco para las altas frecuencias.

ca a qué pata se dirige el cable (también incluimos el color del mismo). Los datos (oscilando de 0 o 5V) ingresan en sucesión por el cable 14J1 y, atravesando el resistor R2, llegan a la pata 13 del PIC. La pata 13 perteneciente al puerto "B" se comporta como una pata de entrada de datos en tanto el cable 2J1 se encuentra en el estado bajo es decir que la pata 4 Vpp está a un potencial de 13V aproximadamente.

Al mismo tiempo, el cable 17J1 se manda a potencial de masa para que los datos entrantes no salgan a su vez por el cable 11J1; de este modo, el diodo D1 no permite que el potencial del cable supere la tensión de barrera del diodo (es decir que el diodo D1 opera como una llave). Como ya sabemos los datos deben ser validados por medio de un cambio de estado de la señal de clock que ingresa desde la PC por el cable 1J1.

Fórmese una imagen mental del flujo de datos. Piense en la PC como si bombeara datos al PIC por el cable 14J1; el PIC, para no inflarse, los devuelve por el cable 11J1. En realidad, los datos ingresan por la pata 13 del PIC y se instalan en la memoria; pero a continuación, el programa de carga verifica que el dato esté en la posición de memoria correspondiente y si así ocurre, se habilita la carga del siguiente dato. La secuencia es tal que:

A) se direcciona una posición de la memoria,
B) se graba,
C) se verifica esta última grabación y si es correcta,
D) se habilita al programa para cargar el siguiente dato.

De acuerdo al programa de carga, si falla la carga de un dato se puede seguir con los otros y al final se intenta la carga del dato que no se cargó. En otros, una falla de verificación significa que el programa debe volver a car-

Por último se agrega un conector del tipo DB21 macho para conectar el dispositivo directamente a la salida de la impresora de una PC. En el circuito dibujamos el conector visto por el lado de las patas de conexiones y el código de colores de cable adecuado para usar un cable plano de 5 hilos (negro, marrón, rojo, naranja y amarillo) al cual se le retuerce por encima otro de color blanco que opera como masa y blindaje. Todo este circuito es sumamente sencillo y si Ud. sólo quiere conocer los PICs mediante este manual técnico, pero no se va dedicar a trabajar permanentemente con ellos, puede armarlo en un módulo de armado sin soldaduras (protoboard) junto con el circuito de un destellador rítmico (figura 18).

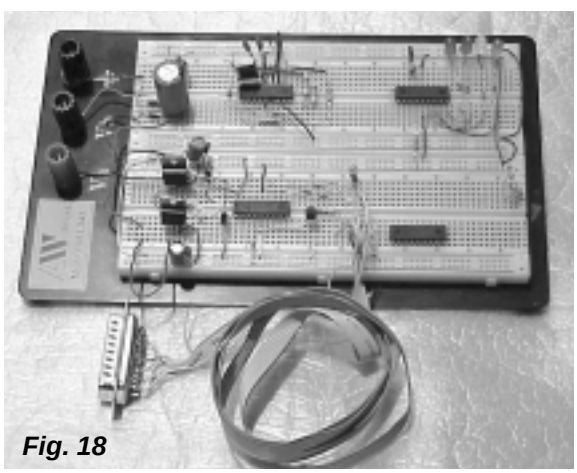
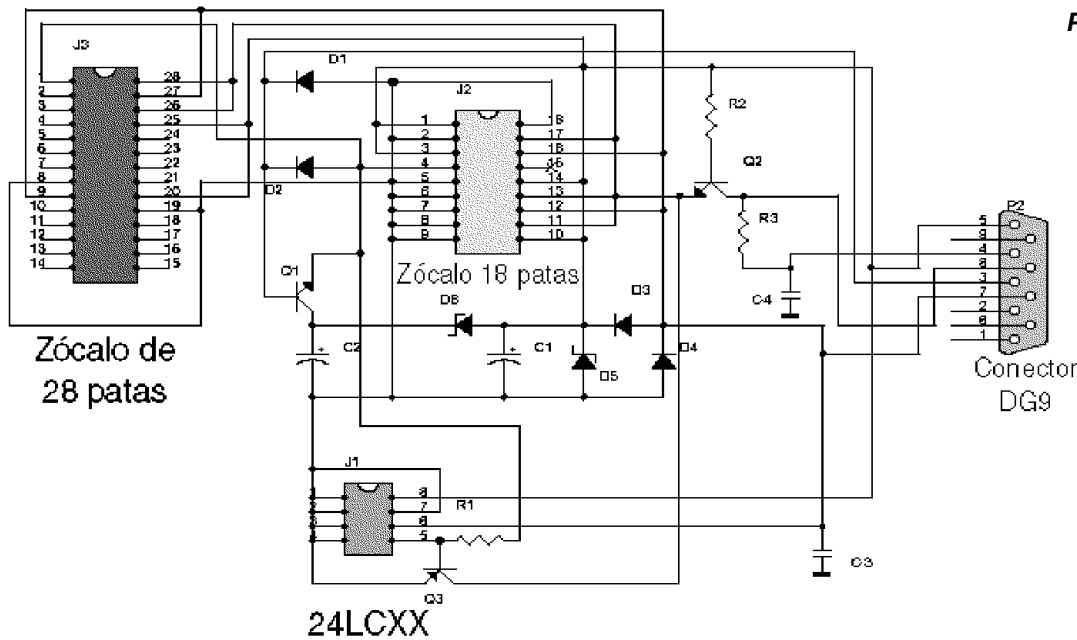


Fig. 18

Fig. 19



CARGADOR UNIVERSAL DE PICS

En varias direcciones de internet aparece el circuito que describiremos a continuación, adjudicándole la autoría a diferentes personas. La versión que reproducimos es en base al trabajo de un tal "Droky" de RaDiKAl ByTEs. Nuestros técnicos han realizado algunas modificaciones y el resultado es el siguiente:

Con este proyecto se puede realizar un programador para PICs y EEPROMs de la serie 24LXxx que funciona bien y requiere pocos componentes externos.

El circuito está basado en el "Luddi" o programador de JDM (vea: www.jdm.com), siendo compatible con diferentes softwares cargadores. Se ha probado en diferentes computadoras desde un Pentium 100 hasta un Pentium III de 850MHz sin inconvenientes. No requiere de ningún tipo de adaptador, y en tres zócalos que dispone, podemos programar:

- *PIC12C508 /A, PIC12C509 /A*
- *PIC12C67x*
- *PIC16C55x*
- *PIC16C61*
- *PIC16C62x*
- *PIC16C71*
- *PIC16C71x*
- *PIC16C8x*
- *PIC16F8x*
- *PIC16F873/4/6/7 (Modelos de 28 patillas)*
- *PIC16C73B/74B/76/77*
- *EEPROMs de la serie 24LCxx*

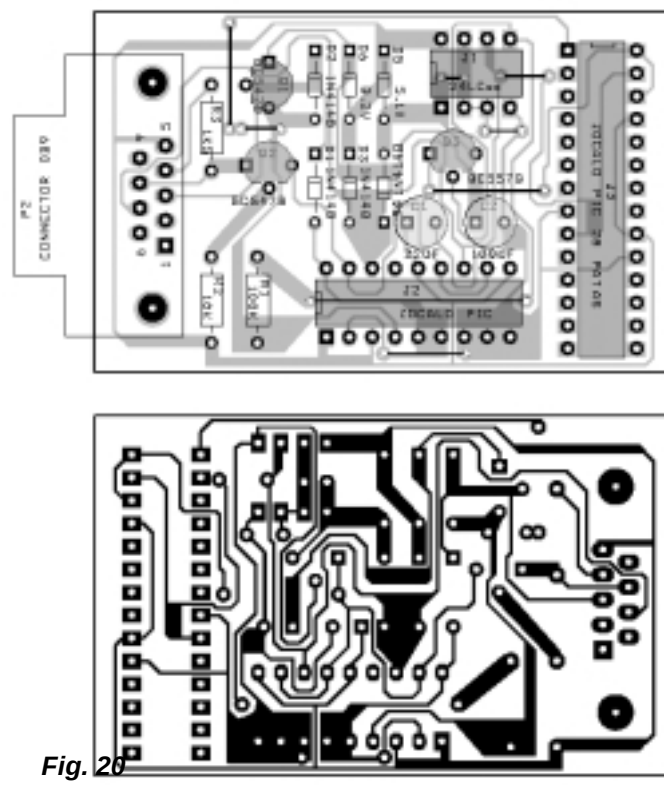


Fig. 20

En la figura 19 se muestra el circuito del cargador y en la 20 la placa de circuito impreso propuesta.

El tamaño del cargador es bastante reducido y posee bases (zócalos) para la mayoría de los chips que puede programar (siempre en encapsulado DIL no en TSOP, SMD ni otros formatos).

Si bien es posible utilizar distintos tipos de softwares, hemos realizado experiencias con el italque, el desden, el pix y el icprog. Es precisamente el icprog el que permite "cargar" a casi todos los chips indicados y lo puede bajar gratuitamente siguiendo los pasos que mencionamos anteriormente, utilizando la clave de acceso: aiwa15.

El programa pix se puede usar para los PIC16F8x y PIC16C8x, es rápido y permite bastantes opciones. Para las EEPROM 24LCxx, pueden emplearse tanto el Icprog como el PIX .

Para los 12C508/A y 12C509 /A lo más seguro es usar el prog508 y el prog509 (vea los foros PSX) y que se pueden descargar de la página de JDM.

La lista de materiales del cargador universal es la siguiente:

C1 - 22µF , 16v Tantalio/Electrolítico
C2 - 100µF , 16v Tantalio/Electrolítico
C3, C4 - 0,001µF - Cerámico
D1,D2,D3,D4 - 1N4148
D5 - 5.1V 1/2W
D6 - 8.2V 1/2W
J1 - Zócalo 24LCXX 8 Patillas
J2 - Zócalo PIC 18 Patillas
J3 - Zócalo PIC 28 Patillas
P2 - CONNECTOR DB9 Hembra
Q1,Q2 - BC548B
Q3 - BC558B
R1 - 100kΩ
R2 - 10kΩ
R3 - 1k5

LOS OTROS CARGADORES:

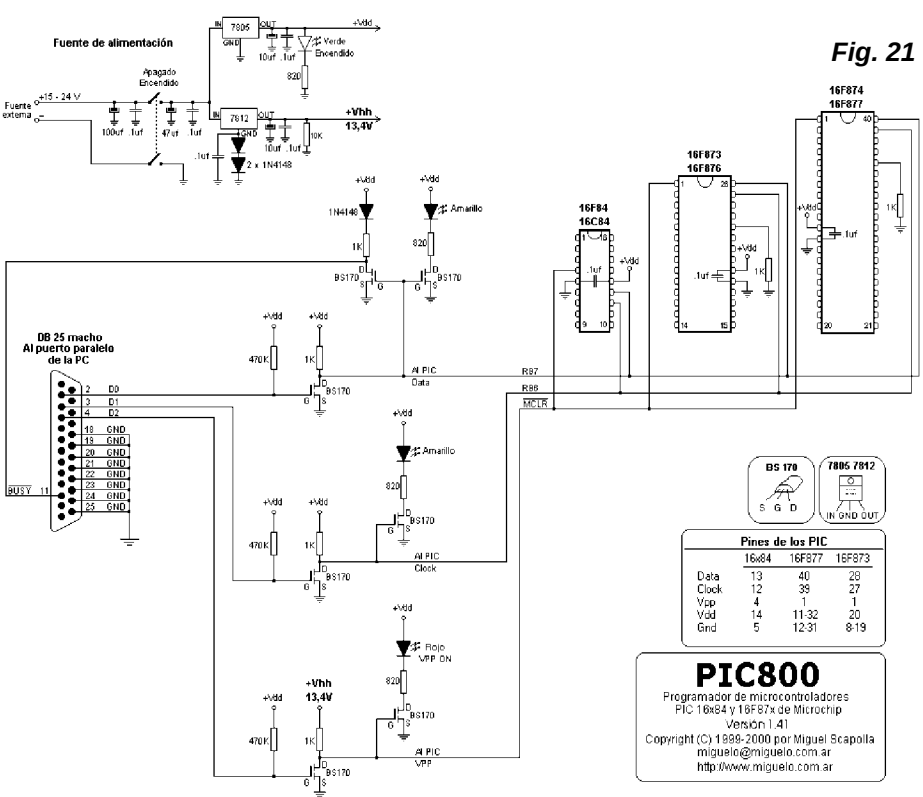
Como hemos dicho, existen muchos circuitos cargadores de PICs que se pueden bajar de Internet.

En la figura 21 se observa el PIC800 que es un programador para 16C84, 16F84, 16F873/74/76/77 que está presentado en la web por Miguel Scapolla (www.miguelo-.com.ar).

El PICPRO es un programador avanzado en relación con el NOPPP y sirve para ser montado cuando no se quieren correr riesgos con la PC que se está utilizando para cargar PICs y cuando se desea tener un trabajo libre de errores (el NOPPP es un programa para aprendizaje). Su circuito se lo puede ver en la figura 22. Por último, el PP84 es un programador sencillo del tipo NOPPP de buenas prestaciones (figura 23).

Si desea obtener los programas que permitan manejar a estos cargadaores, puede buscarlos en Internet o consultar a: **ateclien-@webelectronica.com.ar**

Fig. 21



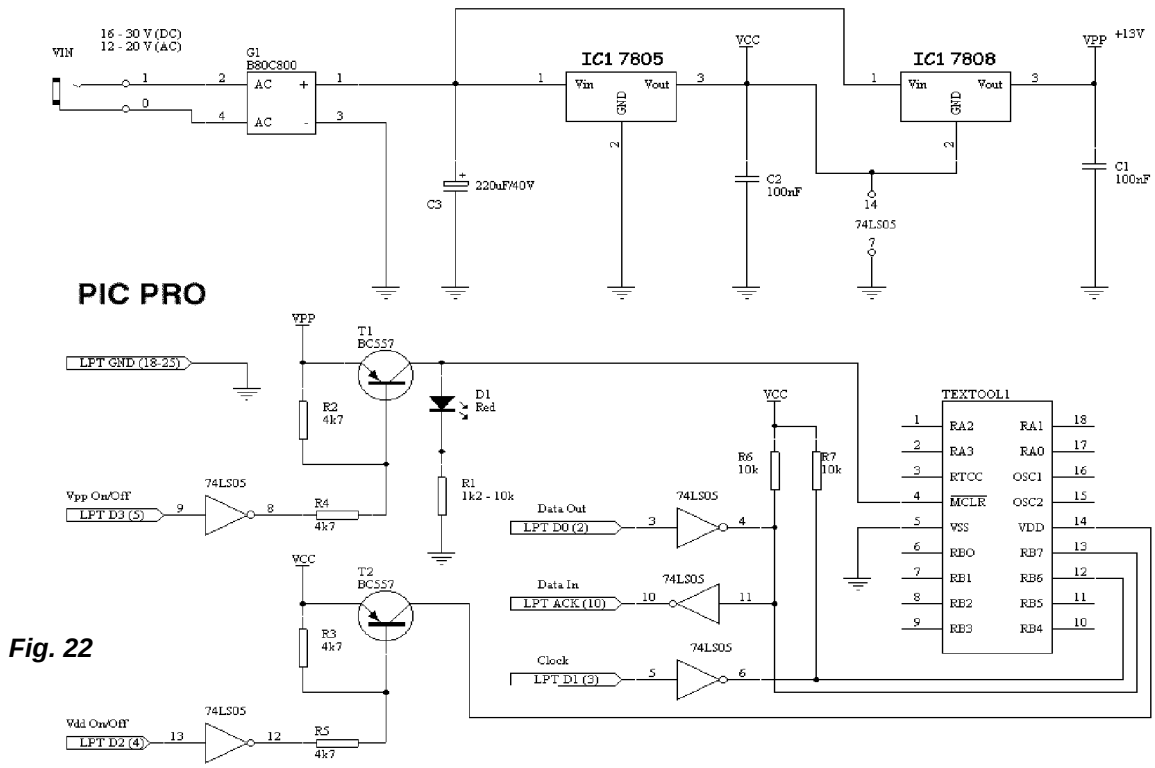
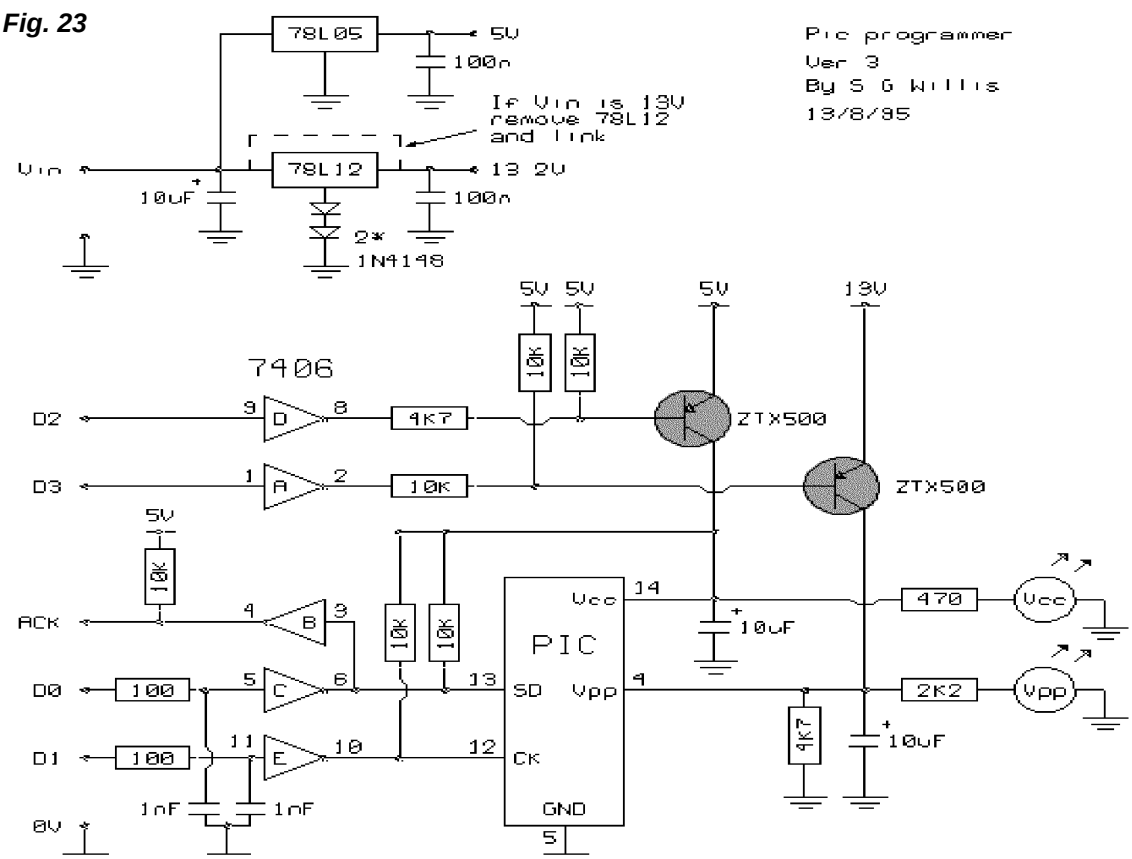

16 - 30 V (DC)
12 - 20 V (AC)
VIN
G1
B80C800
AC
AC
IC1 7805
Vin
Vout
GND
VCC
IC1 7808
VPP +13V
C3 220uF/40V
C2 100nF
C1 100nF
74LS05
PIC PRO
LPT GND (18-25)
VPP
T1
BC557
R2 4k7
D1 Red
R1 1k2 - 10k
Vpp On/Off
LPT D3 (5)
R4 4k7
T2
BC557
R3 4k7
Vdd On/Off
LPT D2 (4)
R5 4k7
R6 10k
R7 10k
Data Out
LPT D0 (2)
Data In
LPT ACK (10)
Clock
LPT D1 (3)
TEXTOOL1
RA2
RA3
RTCC
MCLR
VSS
RB0
RB1
RB2
RB3
RA1
RA0
OSC1
OSC2
VDD
RB7
RB6
RB5
RB4

Fig. 22

Fig. 23


78L05
5V
100n
If Vin is 13V
remove 78L12
and link
Vin
78L12
13.2V
10uF
2*
1N4148
100n
Pic programmer
Ver 3
By S G Willis
13/8/95
7406
D2
D3
ACK
D0
D1
0V
9K7
10K
ZTX500
100
1nF
PIC
Vcc
SD
Vpp
CK
GND
470
2K2
4K7
10uF

El Set de Instrucciones

Manejo de las Instrucciones de un PIC

Aprenda a manejar el set RISC de un PIC para realizar sus propios programas.

INTRODUCCIÓN

Una instrucción es un comando, parte del programa, que se le aplica al PIC para que lo interprete y ejecute su cumplimiento. La instrucción se compone de una serie de bits presentada en secciones o posiciones perfectamente delimitadas, que el procesador interpreta para cumplir una orden deseada, quedando establecidas las variables que se modifican.

Dicho de otra forma, las instrucciones poseen un formato de acuerdo a un sistema de codificación. El formato cambia de acuerdo con la operación que se va a realizar.

Cada instrucción, para la línea baja de los PICs tipo 16C5X, consiste en una palabra de 12 bits que codifica en un solo bloque, la orden, el operador y la ubicación del resultado o el salto (en caso de que lo hubiere). En los microcontroladores PICs tipo 16X84 cada instrucción tiene una longitud de 14 bits.

Los bits que actúan como datos de la memoria EPROM se reciben en el decodificador de instrucciones, y operan con el contador de programa y el registro de trabajo W, para acceder a lugares específicos del microcontrolador, tales como la ALU, posiciones de memoria, registros, etc.

Como sabemos, los PICs 16X84, entre otros, manejan un set reducido de instrucciones (35 instrucciones en lo que se denomina RISC) que presentan una codificación muy particular llamada “formato de la instrucción”.

Cada instrucción posee su formato y es totalmente definido por MICROCHIP.

En la figura 24 podemos apreciar “la sintaxis” de una instrucción del set. Se trata de la instrucción que suma el contenido del registro de trabajo con el contenido de un registro cuya dirección está determinada por el operador “f”. En esta figura, en primer lugar se observa la “sintaxis”, es decir, la forma en que el programador escribirá la instrucción en el lenguaje “entendible por el operador” (el programa en .asm). La codificación es el formato de la palabra que define la instrucción y que veremos en detalle luego. Note que para que se complete la instrucción se necesita una sola palabra y un solo ciclo del contador de programas.

Instrucción: SUMA

Suma el contenido del registro de trabajo con el contenido de la posición de memoria definida por "f"

Sintaxis:	ADDWF f,d
Codificación:	0001 11d1 ffff
Palabras:	1
Ciclos:	1
Operación:	(w) + (f) d
Modifica el Status:	C, DC, Z

Figura 24

Como veremos, el resultado de la operación se guarda en un sitio definido por el programador (dependiendo de qué estado tome el bit “d”) y que esta instrucción modifica los bits C, DC y Z del registro de estado (STATUS).

Esto quiere decir que cuando coloco la instrucción:

ADDWF f,d

El ensamblador generará el código:

0001 11df ffff

En este código, los seis bits de mayor peso (bits 6 al 11 en la figura 25) definen la operación que va a realizar la instrucción de acuerdo a lo requerido por el programa-

Codigo de la instrucción ADDWf

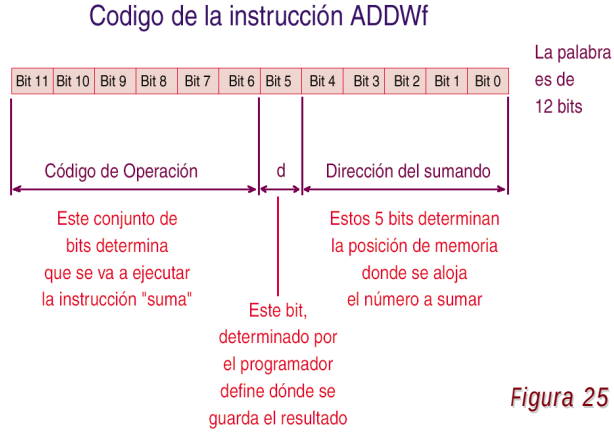


Figura 25

dor. Aquí se define una orden que el PIC interpreta y ejecuta sobre un operando determinado, cuya dirección la busca según lo indicado por los 5 bits menos significativos (bits 0 al 4 en la figura 25).

El bit 5 es un código de reconocimiento de destino y determina en qué lugar se va a alojar el resultado de la operación. Si este bit es “0” el destino de la operación será puesto en el registro de trabajo (W). Si el bit “d” es puesto a “1” el destino será el de la posición de memoria definido por “f”.

Los cinco bits de menor peso en el formato de la instrucción representan la dirección donde está guardado el operando (f), que está en la memoria de datos y por tener cinco bits sólo podemos direccionar una de las 32 posiciones de memoria de datos.

La mayoría de las instrucciones se realizan en un ciclo de contador de programa (ciclo de instrucción) excepto las instrucciones de salto que necesitan dos ciclos para ejecutarla. Se determina el ciclo de instrucción dividiendo por cuatro la frecuencia del oscilador, elegida para el funcionamiento del microcontrolador tal como se observa en la figura 26.

Es decir, la señal que proviene del oscilador externo, conectado a los pines OSC1/CLKIN y OSC2/CLKOUT del microcontrolador, se divide en cuatro ciclos, obteniéndose así la señal requerida por el procesador interno para realizar las operaciones. De esta manera se puede realizar la búsqueda y ejecución de la instrucción.

El reloj de instrucción es el ciclo interno que posee el microcontrolador para cronometrar el tiempo de ejecución de las instrucciones.

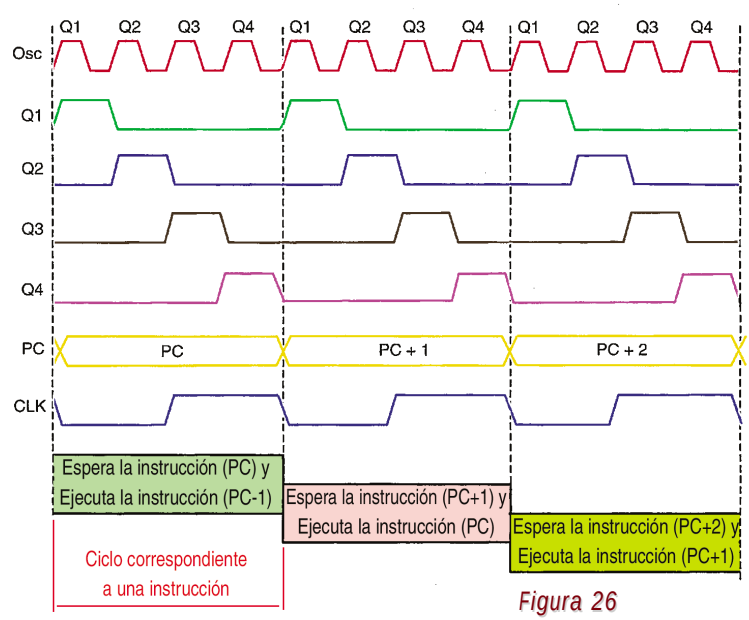


Figura 26

Los pulsos entrantes del reloj son divididos por 4, generando diferentes señales denominadas Q1, Q2, Q3 y Q4. El estado Q1 hace incrementar el contador de programa, Q2 y Q3, se encargan de la decodificación y ejecución de la instrucción y por último, Q4 es la fase de búsqueda de la instrucción. El código se almacena en el registro de instrucciones.

El Set de Instrucciones del 16X84

Vimos cómo es la estructura de una instrucción, razón por la cual le recomendamos que lea nuevamente el comienzo de este capítulo si no entiende lo que a continuación expresaremos.

Los códigos (denominados mnemónicos) que simbolizan un conjunto de instrucciones, representan la tarea que debe hacer el microcontrolador una vez que las analice en función del operando.

Un mnemónico ayuda a recordar el significado que tiene la instrucción.

Para poder analizar al conjunto de instrucciones que conforman el set RISC, se los suele agrupar teniendo en cuenta el tipo de operación que realizan, así es común que se las presente bajo cuatro posibles formas, a saber:

1. *Instrucciones orientadas a registros.*
2. *Instrucciones orientadas a bits.*
3. *Instrucciones con literales.*
4. *Instrucciones de control y especiales.*

En la figura 27 vemos un cuadro donde se reproducen las instrucciones orientadas a registros, la figura 28 muestra las operaciones orientadas a bits y la figura 29 nos permite conocer cuáles son las instrucciones con literales y de control.

Sobre las notas a las que hacemos referencia en dichas figuras, debe tener en cuenta lo siguiente:

Nota 1. Al modificar un registro de entrada/salida (E/S) con una operación sobre él mismo (por ejemplo MOVF PORTB,1), el valor utilizado es el que se encuentre presente en las patas del PORTB. Por ejemplo, si el biestable tiene un "1" para una pata configurada como entrada y se pone a nivel bajo desde el exte-

rior, el dato se volverá a escribir como un "0".

Nota 2. Si se ejecuta esta instrucción sobre el TMR0 y d=1, será borrado el divisor de frecuencia (preescaler), si está asignado al TMR0.

Nota 3. Si se modifica el Contador de Programa (PC ó CP) o es verdadera una condición de prueba, la instrucción requiere dos ciclos de máquina. El segundo ciclo se ejecuta como un NOPPP.

Figura 27

Instrucciones Orientadas a Registros

CODIGO MNEMONICO	DESCRIPCION	CODIGO DE OPERACION	MODIFICACION DE LOS FLAGS	NOTAS
ADDWF f,d	(W)+(f) → (destino)	00 0111 DFFF FFFF	C, DC, Z	1,2
ANDWF f,d	(W) AND (f) → (destino)	00 0101 DFFF FFFF	Z	1,2
CLRF f	00 → (f)	00 0001 1FFF FFFF	Z	2
CLRW	00 → (W)	00 0001 0000 0011	Z	1,2
COMF f,d	Complemento de f [(#f)→ (destino)]	00 1001 DFFF FFFF	Z	1,2
DECF f,d	(f)-1 → destino	00 0011 DFFF FFFF	Z	1,2,3
DECFSZ f,d	(f)-1 → destino y si resultado es 0 salta	00 1011 DFFF FFFF	Ninguno	1,2
INCF f,d	(f)+1 → destino	00 1010 DFFF FFFF	Z	1,2,3
INCFSZ f,d	(f)+1 → destino y si resultado es 0 salta	00 1111 DFFF FFFF	Ninguno	1,2
IORWF f,d	(W) OR (f) → destino	00 0100 DFFF FFFF	Z	1,2
MOVF f,d	Mueve f → destino	00 1000 DFFF FFFF	Z	
MOVWF f	(W) → (f)	00 0000 1FFF FFFF	Ninguno	1,2
NOP	No operación	00 0000 0xx0 0000	Ninguno	1,2
RLF f,d	Rota f a la izquierda a través del carry → destino	00 1101 DFFF FFFF	C	1,2
RRF f,d	Rota f a la derecha a través del carry → destino	00 1100 DFFF FFFF	C	1,2
SUBWF f,d	(f)–(W) → (destino)	00 0010 DFFF FFFF	C,DC,Z	1,2
SWAPF f,d	Intercambia los niveles de f → destino	00 1110 DFFF FFFF	Ninguno	
XORWF f,d	(W) XOR (f) → (destino)	00 0110 DFFF FFFF	Z	

Figura 28

Instrucciones orientadas a Bits

CODIGO MNEMONICO	DESCRIPCION	CODIGO DE OPERACION	MODIFICACION DE LOS FLAGS	NOTAS
BCF f,b	Pone a 0 el bit b del registro f	01 00bb bfff ffff	Ninguno	1,2
BSF f,b	Pone a 1 el bit b del registro f	01 01bb bfff ffff	Ninguno	1,2
BTFSC f,b	Skip si el bit b del reg f es 0	01 10bb bfff ffff	Ninguno	3
BTFSS f,b	Skip si el bit b del reg f es 1	01 11bb bfff ffff	Ninguno	3

Figura 29 **Instrucciones con literales y de control**

CODIGO MNEMONICO	DESCRIPCION	CODIGO DE OPERACION	MODIFICACION DE LOS FLAGS	NOTAS
ADDLW K	(W)+ K → (W)	11 111x kkkk kkkk	C,DC,Z	
ANDLW K	(W) AND K → (W)	11 1001 kkkk kkkk	Z	
CALL K	Llamada a subrutina	10 0kkk kkkk kkkk		
CLRWDT	Clear del temporizador del WD	00 0000 0110 0100	Ninguno	
GOTO K	Go To dirección	10 1kkk kkkk kkkk	Z	
IORLW K	(W) OR K → (W)	11 1000 kkkk kkkk	Ninguno	
MOVLW K	K → (W)	11 00xx kkkk kkkk	Ninguno	
RETFIE	Retorno de una interrupción	00 0000 0000 1001	Ninguno	
RETLW K	Retorno con un literal en W	11 01xx kkkk kkkk	Ninguno	
RETURN	Retorno de una subrutina	00 0000 0000 1000	C,DC,Z	
SLEEP	Modo Standby	00 0000 0110 0011	Z	
SUBLW K	K – (W) → W	11 110x kkkk kkkk		
XORLW K	(W) XOR K → (W)	11 1010 kkkk kkkk		

Programación de PICs

Aprendiendo a Programar: Carga y Programación de PICs

Sepa cómo es la estructura de un programa y cómo se deben utilizar las instrucciones.

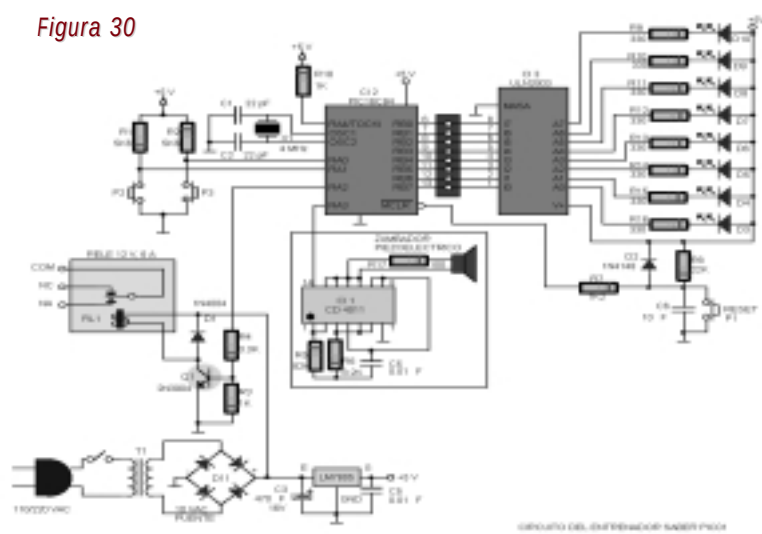
Circuito de un Entrenador

En Saber Electrónica hemos desarrollado un "entrenador" para aprender a programar y cargar PICs, este entrenador se muestra en la figura 30.

Antes de dar los primeros ejemplos es conveniente tener en cuenta los siguientes conceptos:

1) En todo programa se debe decir cuál es el tipo de PIC que se empleará, hay que incluir sus librerías y configurarlo:

Figura 30



Ejemplo:

```
list            p=16f84
include         <p16f84.inc>
__config_RC_OSC &_WDT_OFF
```

2) Al escribir un programa se realizan las sentencias en columnas.

La primera columna se utiliza para nombrar variables o colocar etiquetas. La segunda columna se utiliza para aplicar la instrucción. La tercer columna contiene los datos necesarios para que pueda ejecutarse dicha instrucción. La cuarta columna contiene datos útiles para el programador pero que no son tenidos en cuenta por el microcontrolador.

Ejemplo:

Voy a definir a la variable M en la posición de memoria expresada en número decimal '26':

1ª Col.	2ª Col.	3ª Col.	4ªCol.
M	equ	D'26'	;definí a la variable M en ;la posición 26.

M es la variable que definí

equ es la instrucción que significa asignar o definir

D '26' es la posición de memoria expresada en número decimal.

3) Se emplean signos para efectuar determinadas consideraciones, por ejemplo:

; se emplea para colocar observaciones. El programa no toma en cuenta todo lo que está en una línea luego de dicho signo.

: se utiliza normalmente para definir etiquetas, que son lugares a donde va el programa cuando así lo requiere.

'' se utiliza para decir que lo que está entre ellas es el número y puede estar expresado en lenguaje decimal, binario o hexadecimal.

. se emplea para definir un número en decimal.

Ejemplo:

```
R       equ     .28     ;definí a la variable R en la posición
                        ;de memoria 28
lazo:   rlf     R,f     ;roto el contenido de  R y el resultado
                        ;queda en R
```

4) Un programa siempre "debería" comenzar con la instrucción org y terminar con la instrucción end.

Este manual no pretende ser un "tratado" de programación y su objetivo es capacitar al estudiante y aficionado en el uso y carga de programa de un PIC. Si Ud. desea obtener una capacitación completa le recomendamos estudiar el "Curso Completo de PICs" preparado por Editorial Quark y que se compone de dos textos, un CD y un video de entrenamiento.

PROGRAMAS DE PRÁCTICA

Sea el programa:

```
                                ; Primer programa de práctica

        List    p = 16C84       ; voy a utilizar el PIC16C84
ptob    equ     0x06            ; inicializo la variable ptob en
                                ; la dirección 06h
Reset   org     h'0'            ; comienza el programa
Inicio  movlw   h'0'            ; cargo a W con 0
        tris    ptob            ; mando el contenido de W a ptob
        movlw   h'0f'           ; cargo a W con el número binario 15
ciclo   nop                     ; rutina nula
        goto    ciclo           ; vaya a ciclo
```

El programa comienza con un **(;)**, por lo tanto, lo que sigue en el renglón es tomado como un comentario. Los signos **=** forman un resalte para indicar el inicio del programa y ayudan a darle una distribución agradable a la vista. El programa se lista en cuatro columnas; la primera sirve para colocar las "variables" que utilizaremos como registros y las "etiquetas" que son ubicaciones del programa adónde se debe ir cuando el operando de una instrucción así lo requiera.

En la segunda columna se coloca la instrucción y en la tercera el operando de la instrucción. La cuarta columna siempre va precedida de (;) y se utiliza para colocar observaciones que le sirvan al programador como guía para saber qué quiso hacer o qué función cumple esa sentencia, obviamente, al compilar esa instrucción, las observaciones no son tenidas en cuenta.

List p = 16C84

Es el encabezado del programa que le indica al ensamblador qué tipo de PIC se está utilizando para que éste pueda reconocer qué set de instrucción debe utilizar.

ptob equ 06

Nombré a la variable ptob y la coloqué en la dirección

06 de la RAM, “esta dirección está reservada para el PUERTO B”, es decir, ratifico que ptob es el registro del PORT B. Cuando, más adelante, deba enviar información al puerto b, sólo debo mencionar ptob.

Reset org 0

Significa que al realizarse el reset, el programa comienza por la posición 0 de la memoria del programa. Pero de inmediato pasa a la posición 1 que tiene escrita la siguiente sentencia:

Inicio movlw 0

Con esto se carga el registro w con el hexadecimal 0 (es decir el binario 00000000)

tris ptob

Esta instrucción envía la información del registro W al puerto B para indicarle que todos sus pines son de salida (si se hubiera cargado el binario 11111111 todos los pines serían de entrada y si se hubiera cargado 11001010 algunos serían entradas y otros salidas).

movlw 0f

Carga el registro W con el hexadecimal 15 que equivale al binario 00001111 y

movlw ptob

Envía el valor cargado al puerto **"B"** que producirá un estado alto en RB0, RB1, RB2 y RB3 y un estado bajo en RB4, RB5, RB6 y RB7. La información del puerto pasa al buffer que lleva las patas 15, 16, 17 y 18 de un PIC16C84 a masa encendiendo los leds D7, D8, D9 Y D10.

Ciclo nop

Realiza una rutina nula, es decir, que no efectúa operación alguna.

goto ciclo

Envía el programa hacia la etiqueta **"ciclo"**. Las dos últimas operaciones hacen que al ejecutarse un programa, éste se quede en un lazo que se llama “loop cerrado”.

La única manera de salir de este loop es pulsando **RESET**. Entonces se observa que los leds se apagan hasta que se suelta el pulsador y el programa comienza nuevamente por la etiqueta **RESET**.

Obviamente, este programa debe ser editado en un utilitario adecuado (MPLAB, por ejemplo) y luego debe ser compilado (utilizando el MPASM o el mismo MPLAB) para obtener el archivo .hex que me permitirá cargar el PIC que deberé colocar en el circuito de la figura 1 para verificar que realmente “hace” lo que estamos diciendo.

Para cargar el PIC con el programa .hex se utiliza un prototipo adecuado (cargador de PICs) que consiste en un circuito que es manejado por un programa para permitir la carga. Todo este proceso se explica con total claridad en el primer texto de esta serie, titulado: “Todo Sobre PICs”.

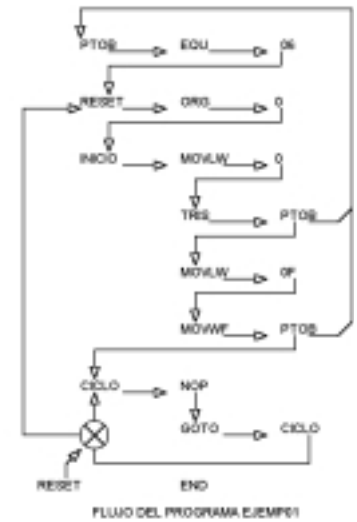


Figura 31

En la figura 31 se puede observar un diagrama de flujo que refleja el funcionamiento del programa que acabamos de explicar.

¿Cuánto tardan en encenderse los leds luego de soltar el botón de reset?

Si se observa el circuito, se verá un cristal de clock de 4MHz (0,25µS de período); como internamente existe un divisor x4 cada operación se realizará en 0,25 x 4 = 1µS. Si contamos las sentencias hasta llegar a cargar el puerto **B**, veremos que hay 5 (cinco renglones de programa); por lo tanto, la demora es de 5µS.

****

Veamos ahora algunos ejemplos de programas sencillos que rápidamente podemos llevar a la práctica:

Programa que permite el encendido de los dos bits menos significativos del port B (figura 32)

```
list      p=16f84
include   <p16F84.inc>
org       0
movlW     B'00000000'
tris      PORTB
movlw     B'00000011'
movwf     PORTB
end
```

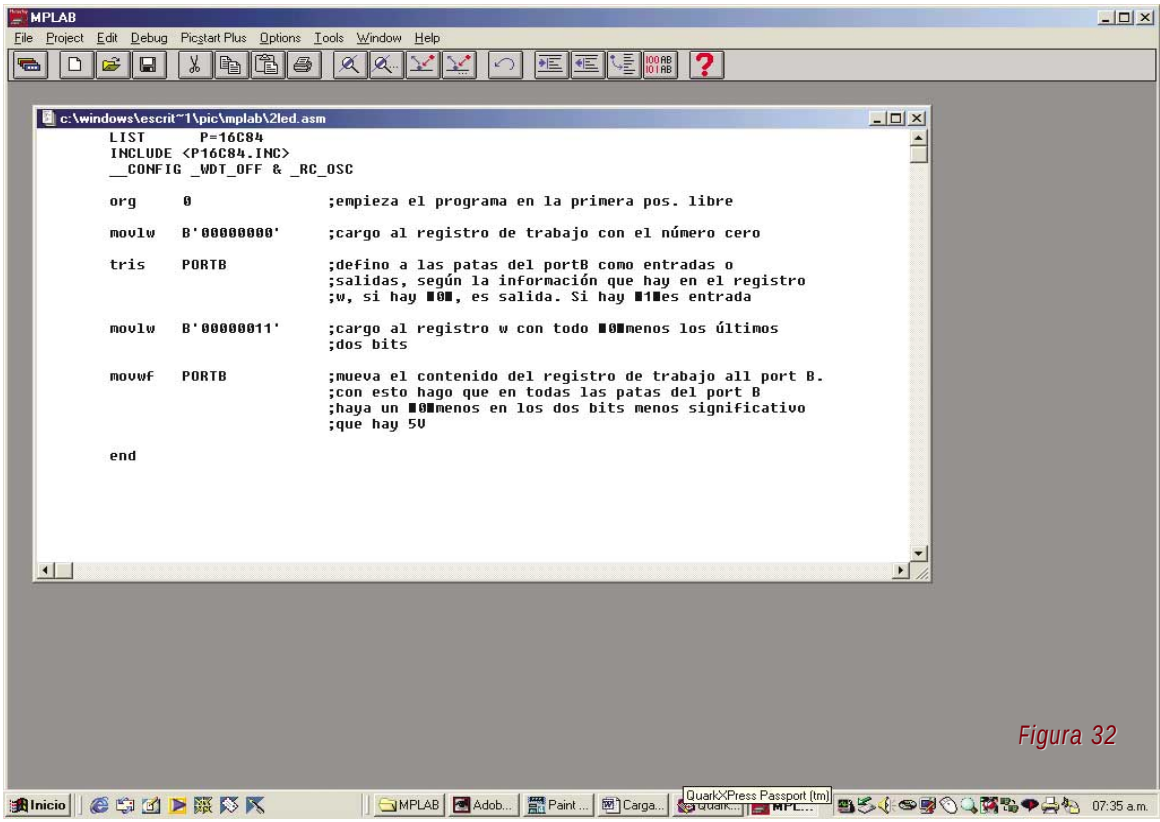


Figura 32

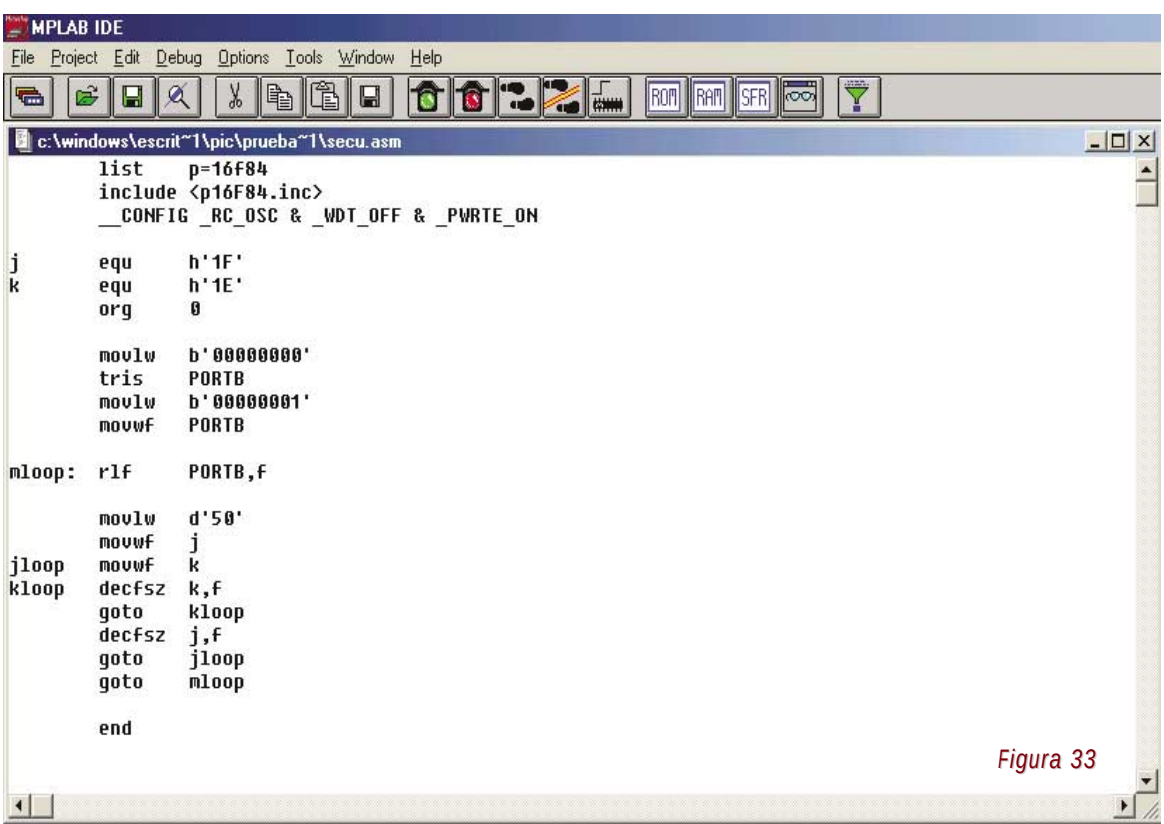


Figura 33

En la figura 33 se observa el programa correspondiente a un secuenciador de 8 canales que hemos editado en el programa MPLAB.

```
list        p=16f84
include     <p16f84.inc>
__CONFIG _RC_OSC & _WDT_OFF & _PWRTE_ON
```

Este juego de sentencias indica que se va a utilizar el PIC 16F84, se pide que se carguen las librerías del PIC y se configura al programa para trabajar con clock RC, el temporizador WDT apagado y el temporizador libre activo.

```
j     equ     h'11'
k     equ     h'12'
```

Defino a las variables j y k en las posiciones de memoria de dato 11 y 12 (en hexadecimal) respectivamente.

```
      org     0
```

Comienza el programa

```
      movlw   b'00000000'
      tris    PORTB
```

Con estas dos instrucciones digo que todas las patas del PORTB serán salidas.

```
      movlw   b'00000001'
      movwf   PORTB
```

Mandé un “1” a la pata B0, es decir que si hay un led conectado entre esa pata y masa, el mismo se encenderá.

```
mloop:      rlf     PORTB,f
```

En esta sentencia coloqué una etiqueta (una marca) que será la posición a la que irá el programa en algún momento, cuando se le dé la instrucción. Luego con la instrucción rlf digo que se rote hacia la izquierda el contenido del registro PORTB, es decir que luego de esta instrucción ahora mandé un “1” al bit B1 y todas las demás patas quedan en cero, es decir, se apaga el led conectado en B0 y se enciende el led conectado en B1 (pata 7 del integrado).

```
      movlw   d'50'
      movwf   j
```

Puse en la posición de memoria j el número 50

```
jloop movwf   k
```

Puse en la posición de memoria k el número 50, además en esta sentencia coloqué una etiqueta y en algún momento le diré al programa que vaya hacia esa dirección.

```
kloop decfsz  k,f
      goto    kloop
```

Ahora doy la instrucción para que se decremente el contenido de la variable k y si ese contenido es “0” entonces que salte una instrucción; sino es “0” entonces mando al contador de programa a la etiqueta kloop para que se haga un nuevo decremento. Es decir, estoy haciendo un lazo cerrado para “perder tiempo”

```
      decfsz  j,f
      goto    jloop
```

Entro en un nuevo lazo cerrado cuando k=0, en este caso para decrementar el contenido de la variable j.

Con este juego de instrucciones cuento “50 veces 50” (cuento hasta 2500) y con cada cuenta consumo 4 ciclos de reloj de modo que tardé 10.000 ciclos en terminar el lazo completo. Si la frecuencia de reloj es de 10.000Hz, entonces tardaré un segundo en completar el doble lazo.

```
      goto    mloop
```

Ahora mando al contador de programa a la posición donde está la etiqueta mloop para que se haga una nueva rotación del contenido del PORTB, es decir, se encenderá ahora otro led, apagándose el anterior. Esta rutina se sigue indefinidamente, es decir, tenemos un secuenciador de 8 canales.

```
      end
```

Indica que finalizó el programa.

De más está decir que ésta es una forma de programar “no muy adecuada” pero que sirve perfectamente para que el lector aprenda a estructurar sus propios proyectos.

Es aconsejable que al editar un programa lo haga en el MPLAB (siguiendo los pasos que explicaremos más adelante), pues de esta manera tendrá la oportunidad de poder comprobar si ha cometido errores o no en su trabajo. El MPLAB es ideal porque la forma de manejarlo es sencilla y sistemática.

A los fines prácticos, en la página siguiente damos el programa correspondiente a un semáforo muy sencillo. Le sugiero que intente seguir paso a paso cada instrucción para comprender su estructura.

MPLAB

File Project Edit Debug Picstart Plus Options Tools Window Help

c:\windows\escrit~1\pic\prueba~1\semafor.asm

```
        list    p=16f84
        include <p16f84.inc>
        __CONFIG _RC_OSC & _WDT_OFF & _PWRTE_ON

j       equ     h'1F'
k       equ     h'1E'

        org     0

        movlw   b'00000000'
        tris    PORTB
mloop:  movlw   b'00000001'
        movwf   PORTB

        movlw   d'200'
        movwf   j
jloop   movwf   k
kloop   decfsz  k,f
        goto    kloop
        decfsz  j,f
        goto    jloop
        rlf     PORTB,f
        movlw   d'50'
        movwf   j
aloop   movwf   k
bloop   decfsz  k,f
        goto    bloop
        decfsz  j,f
        goto    aloop
        rlf     PORTB,f
        movlw   d'200'
        movwf   j
cloop   movwf   k
dloop   decfsz  k,f
        goto    dloop
        decfsz  j,f
        goto    cloop
        goto    mloop

        end
```

Figura 34

Inicio MPLAB Adob... Paint ...

Programa Secuenciador de 4 Canales

```
list        P=16C84                 ;el dispositivo usa el 16C84
;-----------------------------------------------------------------------------
ptob        equ         06          ;el puerto B se ubica en 06 de la RAM.
Rot         equ         0d          ;al registro ubicado 0d se lo llama Rot.
Reg1        equ         0e          ;al registro ubicado en 0e se lo llama reg1
reg2        equ         0f          ;al registro ubicado en 0f se lo llama reg2
reg3        equ         10          ;al registro ubicado en 10 se lo llama reg3
grueso      equ         00          ;se asignan nombres a los literales.
medio       equ         40          ;se asignan nombres a los literales.
fino        equ         50          ;se asignan nombres a los literales.
;-----------------------------------------------------------------------------
reset       org         0           ;reset en dirección 00h
            goto        comenzar    ;se lleva el control a comienzo
comenzar    movlw       00          ;se carga w con 00h
            tris        ptob        ;se programa el puerto "b" como salida.
ppal        movlw       01          ;se carga w con 01h
            movwf       rot         ;se descarga w en el registro rot
                                    ;preparándolo para la rotación.
sec         movf        rot,0       ;el contenido de rot pasa a w
            movwf       ptob        ;enciende el led conectado donde indica w
            call        demora      ;se mantiene encendido el led.
            rlf         rot         ;se mueven los bits de "rot".
            btfss       rot,4       ;se comprueba el 5º bit.
            goto        sec         ;si el 5º bit es cero  se reenvía a sec.
            goto        ppal        ;si el 5º bit es uno se reenvía a ppal para
                                    ;comenzar una nueva secuencia.
demora      movlw       grueso      ;recarga w con el número hexadecimal
                                    ;“grueso” es decir, con 30
            movwf       reg1        ;se vuelca “grueso” en el registro
                                    ;reg 1
dem3        movlw       medio       ;se carga w con el número hexadecimal
                                    ;“medio” es decir con 40
            movwf       reg2        ;se vuelca "medio" en el registro
                                    ;reg2.
dem2        movlw       fino        ;se carga w con el número hexadecimal
                                    ;"fino" es decir con 50.
            movwf       reg3        ;se vuelca "fino" en el registro reg3.
dem1        decfsz      reg3        ;se decrementa el registro reg3 en una
                                    ;unidad y si el resultado es cero se salta a
                                    ;la siguiente instrucción, si no se sigue
                                    ;decrementando.
            goto        dem1        ;retorno a la frecuencia etiqueta dem1.
            decfsz      reg2        ;idem con reg2.
            goto        dem2        ;retorno a la etiqueta dem2.
            decfsz      reg3        ;idem con reg1.
            goto        dem1        ;retorno a la etiqueta dem3.
            retlw       0           ;se carga w con 0 y se retorna al
                                    ;programa principal.
            end
```

Manejo del MPLAB

Edición y Simulación de Programas

Sepa cómo editar un programa y encontrar errores con el MPLAB.

INTRODUCCIÓN

MPLAB es un entorno de desarrollo integrado que le permite escribir y codificar los microcontroladores PIC de Microchip para ejecutarlos. El MPLAB incluye un editor de texto, funciones para el manejo de proyectos, un simulador interno y una variedad de herramientas que lo ayudarán a mantener y ejecutar su aplicación. También provee una interfase de usuario para todos los productos con lenguaje Microchip, programadores de dispositivos, sistemas emuladores y herramientas de tercer orden.

El MPLAB está diseñado para ser ejecutado bajo Windows 3.11, y puede operar con Windows 95, 98 y superiores (vea www.microchip.com). Asume que el usuario ya conoce el entorno de Windows y sabe manejarlo. La guía que describimos le permitirá realizar las siguientes tareas:

- *Manejar el escritorio MPLAB*
- *Crear un nuevo archivo de código fuente para el ensamble e ingresarlo a un nuevo proyecto para el 16F84*
- *Identificar y corregir los errores simples*
- *Ejecutar el simulador interno*

Para que Ud. aprenda a programar sus microcontroladores, damos a continuación, paso a paso, las instrucciones de instalación y uso de la aplicación:

INSTALACIÓN

Descargue los archivos del software de instalación y ejecute el archivo MPxxxxx.EXE. Estos archivos puede obtenerlos por medio de Internet en la dirección:

http//www.microchip.com/10/Tools

Estos archivos pueden ser transferidos a disquetes si desea instalar el MPLAB en otra computadora. De acuerdo a la versión que haya descargado, los nombres de los archivos pueden variar levemente. Por ejemplo, la versión 4.00 del MPLAB tendría los siguientes archivos:

MP40000.EXE
MP40000.WO2
MP40000.WO3
MP40000.WO4
MP40000.WO5
MP40000.WO6

Si Ud. lo prefiere, puede venir a nuestras oficinas con este libro y 6 disquetes vírgenes y tendrá la oportunidad de llevarse dicho programa sin cargo.

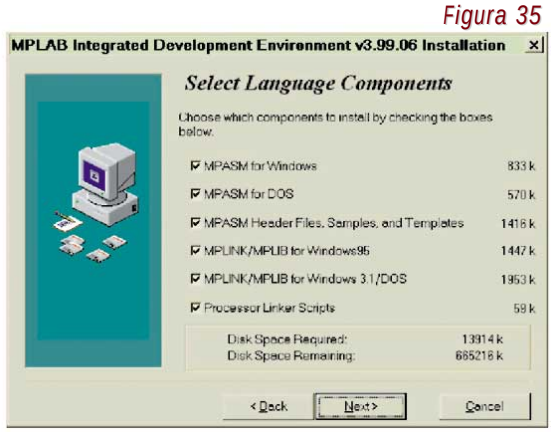
Copie el contenido de los 6 disquetes en el disco rígido de su PC (en un lugar que pueda identificar).

Cuando ejecute el archivo .EXE, comenzará la instalación del MPLAB en su sistema. Seguidamente deberá elegir los componentes del MPLAB que desea instalar en su sistema. A menos que haya comprado un programador o emulador del dispositivo, sólo debiera instalar las siguientes herramientas del software:

Archivos MPLAB IDE
Archivos MPASM/MPLINK/MPLIB
Archivos de Protección del Simulador MPLAB-SIM
Archivos de Ayuda (vea la figura 1 que aparecerá cuando ejecute el programa).

Luego aparecerá el menú de la figura 35 que le permitirá seleccionar los componentes de lenguaje Microchip que desee instalar. Usualmente debiera seleccionarlos todos (por defecto). Al hacer “doble click” en MP40000.EXE, el instalador le va diciendo lo que debe

Figura 35



hacer. Luego de instalarlos, ejecute MPLAB.EXE o clique el ícono MPLAB para iniciar el sistema.

Aparecerá el escritorio del MPLAB (figura 36).

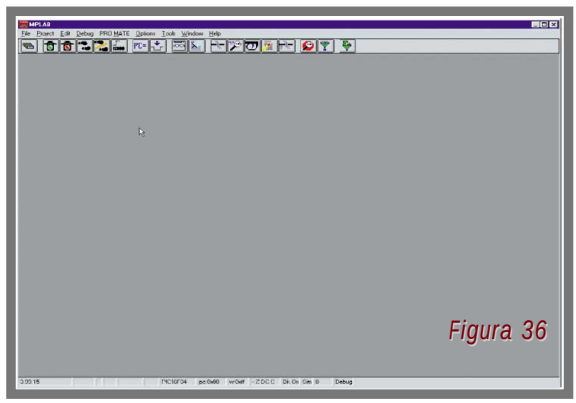


Figura 36

Configurar el Modo de Desarrollo

El escritorio básico del MPLAB se asemeja al de las aplicaciones de Windows (como pudo ver en la pantalla de la figura 36). Tiene una barra de menú en el margen superior, una barra de herramientas y también una barra de estado en el margen inferior. Podrá advertir que la barra de estado incluye información sobre cómo se ha configurado el sistema.

Nota: El *"modo de desarrollo"* determina la herramienta, debe elegir alguna, que ejecutará el código. Para esta guía, usaremos el simulador de software *MPLAB-SIM*. Si sabe del tema y tiene un emulador, en este capítulo encontrará más información para cambiar a una de sus operaciones. Al seleccionar el ítem del menú ***"Opciones>Modo de Desarrollo"***, aparecerá una caja de diálogo semejante a la de la figura 37.

El **MPLAB** es un producto en constante evolución, de modo que pueden aparecer sutiles diferencias entre la pantalla que usted vea y la que mostramos aquí. Seleccione el ícono próximo al Simulador MPLAB-SIM (MPLAB-SIM Simulator) y elija 16F84 (que corresponde a un tipo de PIC) en la lista de procesadores disponibles que pueden ser resistidos por el simulador. Clique 16F84 y luego presione el botón ***"Reset"***. De este modo se iniciará el simulador, y debería aparecer en la barra de estado "16F84" y "Sim". Se encuentra así en el modo simulador para el 16F84.

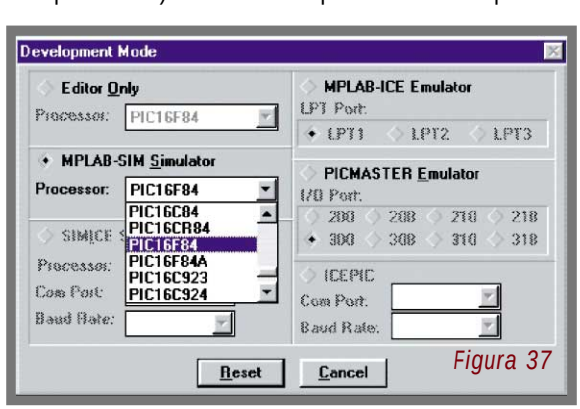


Figura 37

Cómo Crear un Proyecto Simple

El simulador se ejecutará desde el mismo archivo, llamado ***"archivo hex"***, el cual puede ser programado en el micro PIC. Para que se ejecute el simulador, primero deberá crear un archivo de código fuente y realizar el montaje del código fuente.

A continuación explicaremos cómo llevar a cabo este proceso:

Nota: El lenguaje ensamblador produce, entre otros elementos, un archivo **hex**. Este archivo tiene la extensión (***.hex***). A este archivo lo llamaremos:

tutor84.hex.

Más adelante este archivo puede ser cargado directamente en el programador del dispositivo sin usar el ensamblador o un proyecto del MPLAB. Este archivo también puede ser cargado por otros programadores de tercer orden.

Seleccione ***"Archivo>Nuevo (File>New)"*** en el menú y aparecerá la caja de diálogo de la figura 38.

Clique en el botón ***Sí***, seguidamente aparecerá un diálogo de exploración de Windows estándard. Decida dónde desea crear su proyecto (en qué carpeta o lugar de su disco rígido lo va a guardar) y recuerde dónde lo ubicó. Más tarde necesitará esta información. Esta guía usa un directorio en ***c: \temp\tutorial*** y crea el archivo de proyecto llamado ***tutor84.pjt*** (figura 39). ***"PJT"*** es el su-

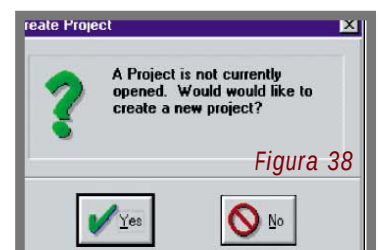


Figura 38

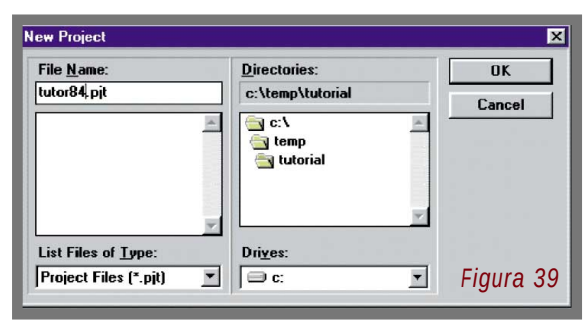


Figura 39

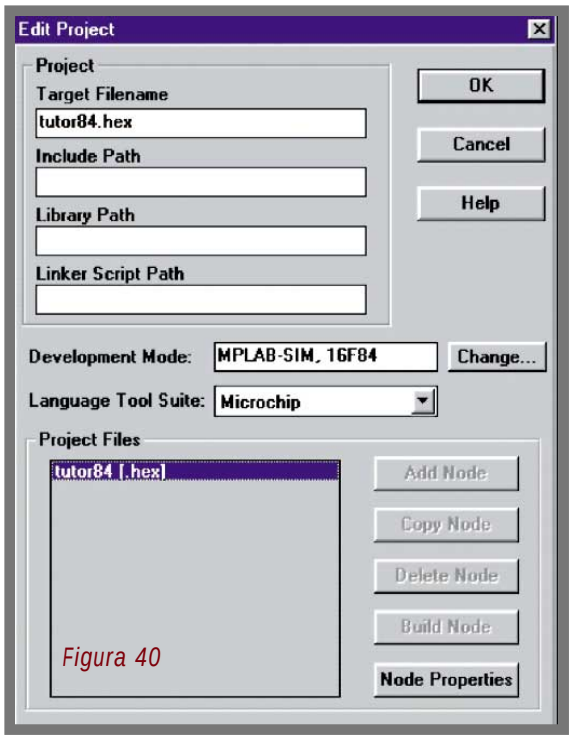


Figura 40

fijo estándar para los archivos de proyectos en el MPLAB.

El prefijo del nombre de archivo del proyecto, en este caso "tutor84", será el prefijo por defecto de muchos de los archivos que el MPLAB usará o creará para esta guía. Clique ***"Aceptar (OK)"*** para que aparezca el diálogo Proyecto MPLAB. Este diálogo puede parecer confuso, pero en realidad es muy simple.

Nota: El simulador, los programadores y los sistemas emuladores que operan con el MPLAB usan un archivo ***hex*** creado por el ensamble, la compilación y/o el "linking" del código fuente. Algunas herramientas diferentes pueden crear archivos hex, al tener en cuenta que estas herramientas formarán parte de cada proyecto.

Los proyectos le dan la flexibilidad para describir cómo se construirá la aplicación y qué herramientas se usarán para crear el archivo ***.hex***. Obviamente, en la guía nos ocuparemos de todos estos detalles.

El diálogo ***"Editar Proyecto"*** será semejante al mostrado en la figura 40.

Advierta que el nombre del archivo de destino ya ha sido completado. Ya conoce el modo de desarrollo que configuramos previamente y asume que usaremos la serie de herramientas de lenguaje Microchip. En la ventana ***"Archivos de Proyecto (Files proyect)"***, encontrará ***tutor84. [hex]***. Al destacar este nombre, se podrá utilizar el ícono ***"Propiedades del Nodo (Node Properties)"***. Seguidamente debe indicarle al MPLAB cómo crear el archivo hex. Hágalo clicando el botón "Propiedades del Nodo". Aparecerá el diálogo "Propiedades del Nodo". Este diálogo contiene todas las configuraciones por defecto para una herramienta de lenguaje -en este caso **MPASM**, como podrá ver en en el ángulo superior derecho del diálogo. En su forma más simple, el proyecto contiene un archivo hex creado desde un archivo fuente de ensamble. Esta será la configuración por defecto cuando aparezca el diálogo "Propiedades del Nodo (Node Properties)", vea la figura 41.

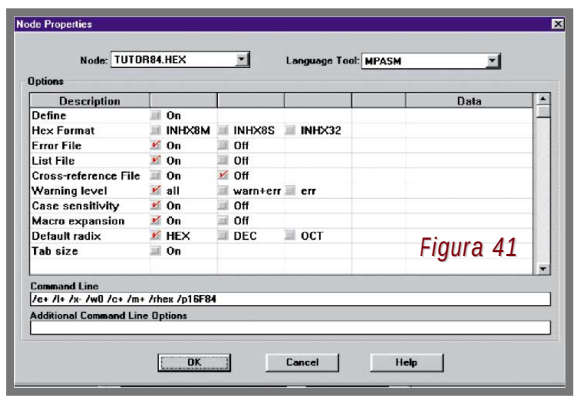


Figura 41

Nota: Como puede ver, hay una cantidad de filas y columnas en este diálogo.

Cada fila usualmente corresponde a un ***"cambio"***, aquellos elementos que se establecen en la línea de comando cuando se invoca una herramienta. De hecho, la configuración de estos cambios se refleja en la ventana ***"Línea de Comando (comand line)"***, próxima al margen inferior de la pantalla. Esta es la línea de comando que se usará cuando se invoque el MPASM desde el MPLAB. Por el momento, puede usar las configuraciones por defecto, pero cuando ya sepa construir una aplicación, probablemente deseará cambiar algunas.

Al clicar el botón ***"Aceptar (OK)"***, aplicará estas configuraciones, y retornará al diálogo ***"Editar Proyecto (Edit Project)",*** con el ícono (botón) ***"Agrega Nodo (Add Node)"*** disponible. Presione el botón ***"Agregar Nodo"***. Aparecerá el diálogo de exploración de Windows estándar, con el mismo directorio usado para el proyecto. Ingrese el nombre de archivo: ***tutor84.asm*** y presione ***"Aceptar"***. Retornará al diálogo ***"Editar Proyecto"***, donde podrá ver ***"tutor84.asm"*** añadido debajo del archivo hex, que indica que es un nodo concurrente. Al presionar ***"Aceptar"***, retornará al escritorio MPLAB con un archivo de código fuente abierto y aún sin nombre.

Cómo Crear un nuevo Archivo Fuente Simple

Clique dentro del espacio en blanco de la ventana de archivo creada. Seguramente se llamará "***Sin título (Un-***

titled)". De este modo accederá al "***foco"*** de la ventana. Use la opción de menú "***Archivo>Guardar como...***", y guarde el archivo vacío como ***tutor84.asm***. Cuando abra el diálogo de exploración estándar, encontrará su ubicación en el directorio del proyecto. Ingrese el nombre de archivo y presione "***Aceptar***". Vea la figura 42.

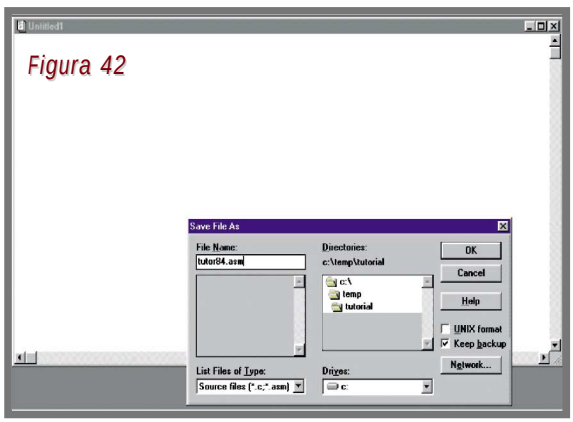


Figura 42

Ahora estarán disponibles el escritorio MPLAB y la ventana de archivo vacío, pero el nombre de la ventana de archivo reflejará su nuevo nombre.

Nota: El nombre del archivo fuente y el nombre del proyecto ("tutor84" en esta guía) deben ser iguales en este tipo de proyectos. Hay otros proyectos de archivo múltiple que usan el "linker" y permiten que el nombre del archivo de salida sea diferente al del archivo de entrada (hay una guía aparte para los proyectos de archivos múltiples que usan el linker).

El ***MPASM*** siempre creará un archivo ***hex*** de salida con el mismo nombre que el archivo fuente, y esta configuración no puede modificarse. Si cambia el nombre del archivo fuente, también deberá cambiar el nombre del proyecto. Ahora ya está listo para escribir el código que almacenará en el PIC para que cumpla una función determinada.

Ingresar el Código Fuente

Use el mouse para ubicar el cursor al comienzo de la ventana de archivo vacío **tutor84.asm**, e ingrese el texto de la tabla 1, exactamente como está escrito en cada línea. No debe ingresar los comentarios (los textos que siguen a cada punto y coma).

Este código es un programa muy simple que incrementa un contador y lo "resetea" a un valor predeterminado cuando el contador vuelve a cero.

Nota: Todos los rótulos comienzan en la primera columna, y la última línea tiene una directiva "end". Las páginas de datos del micro PIC contienen información completa sobre instrucciones con ejemplos para su uso.

Guarde el archivo usando la función de menú ***"Archivo>Guardar" (File>Save).***

Ensamble del Archivo Fuente

El ensamble del archivo puede realizarse de varias maneras. Aquí describiremos un método. Use el ítem de menú ***"Proyecto>Construir todo (Proyect>Build All)"***. De este modo ejecutará el lenguaje ensamblador MPASM en el trasfondo usando las configuraciones guardadas con el proyecto anteriormente. Una vez completado el

Tabla 1

```
        list    p=16f84
        include <p16F84.inc>

c1      equ     h'0c'   ; Establece el contador de variable temp c1 en la dirección 0x0c
        org     h'00'   ; Establece la base de memoria del programa en el vector reset 0x00 reset
        goto    start   ; Ir a inicio en el programa principal

        org     h'04'   ; Establece la base de memoria del programa al comienzo del código del
                        ; usuario
start
        mov1w   h'09'   ; Inicializa el contador a un valor arbitrario mayor que cero
        movwf   c1      ; Guarda el valor en la variable temp definida
loop
        incfsz  c1,F    ; Incrementa el contador, ubica los resultados en el registro de archivos
        goto    loop    ; loop hasta que el contador se completa

        goto    bug     ; Cuando el contador se completa, va a start para reiniciar
        end
```

proceso de ensamble, aparecerá la ventana *"Resultados de Construcción (Build Results"* (figura 43):

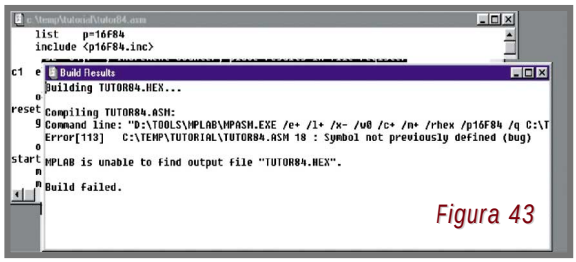


Figura 43

Ha ingresado intencionalmente al menos "un error" si ingresó el código tal como lo hemos escrito en el paso anterior. El último "goto" en el programa refiere a un rótulo inexistente llamado "bug". Dado que este rótulo no ha sido definido previamente, el lenguaje ensamblador informará el error. También podría relevar otros errores.

Haga un doble click sobre el mensaje de error. De este modo ubicará el cursor en la línea que contiene el error en el código fuente. Cambie "bug" por "start". Use la ventana ***"Resultados de construcción (Build Results)"*** para hacer una búsqueda de errores, y reparar los que aparecieran en el código fuente. Reensamble el archivo ejecutando la función de menú ***"Proyecto>Construir todo"***. Este procedimiento puede demandar un par de repeticiones.

Nota: Cuando reconstruya un proyecto, todos sus archivos fuente serán guardados en el disco.

Luego de reparar todos los problemas en el código fuente, la ventana "Resultados de construcción" mostrará el mensaje ***"Construcción completada exitosamente (Build completed successfully)"***. Ya ha completado un proyecto que puede ejecutarse usando el simulador.

Ejecución de su Programa

Use ***"Debug>Ejecutar>Reset (Debug>Run>Reset)"*** para iniciar el sistema. El contador del programa se reseteará a cero, que es el vector de reset en el 16F84.

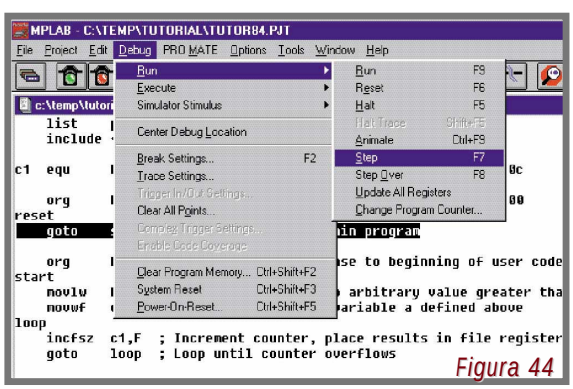


Figura 44

La línea del código fuente en esta dirección será destacada con una barra oscura. También advertirá que en la barra de estado, la PC se establecerá en 0x00.

Use el ítem de menú ***"Debug>Ejecutar>Paso (Debug>Run>step)"*** (figura 44). Al hacerlo, el contador del programa avanzará hasta la siguiente ubicación de instrucción. La barra oscura seguirá el código fuente y el contador del programa desplegado en la barra de estado avanzará hasta "4".

Cuando ejecute el ítem de menú "Debug>Ejecutar->Paso", advierta la aparición de un texto en el lado derecho del ítem de menú que dirá "F7". El mismo equivale a "tecla de función siete" en su teclado. Muchas funciones del MPLAB se asignan a "teclas-especiales". Estas teclas cumplen la misma función que los ítems de menú a los cuales corresponden. Presione F7 varias veces y verá cómo el contador del programa y la barra avanzan a través del programa.

Ejecute el ítem de menú ***"Debug>Ejecutar>Ejecutar (Debug>Run>Run)"*** o presione F9 para inciar la ejecución del programa desde la la ubicación actual del contador. Los colores de la barra de estado cambiarán, para indicar que el programa está ejecutando las instrucciones. Ninguno de los campos de la barra de estado se actualizará mientras el programa esté en ejecución.

Detenga el programa ejecutando el ítem de menú ***"Debug>Run>Halt (detener)"*** o presionando F5. La barra de estado volverá a su color original, y el contador del programa y otras informaciones de su estado serán actualizadas.

Nota: Otra manera de ejecutar funciones es usar la barra de herramientas ubicada en el margen superior de la pantalla. Si ubica el cursor sobre los ítems de la barra de herramientas, podrá ver el nombre de su función en la barra de estado. El botón de la izquierda es un botón estándar ***"cambiar barra de herramientas (change tool bar)"*** que le permite desplegar las barras de herramientas disponibles. Estas pueden ser personalizadas, como podrá advertirlo en la sección "Algunas Sugerencias" al final de esta guía. En la barra de herramientas de Debug, la luz verde es equivalente a F9 (Ejecución) y la luz roja equivale a F5 (Detención).

Hasta aquí, aprendió a instalar el programa y crear un archivo para *"aprender a programar"*. Dimos un ejemplo de programación y ya sabemos cómo *"correr el programa en la PC"* para saber si todo está bien, antes de escribir dicho programa en nuestro PIC. Sin embargo, cuando los programas son más largos, es posible cometer errores que pueden ser evitados. A continuación explicamos el procedimiento.

Abrir Otras Ventanas Para el Seguimiento de Errores

Hay muchas maneras de visualizar el programa y su ejecución usando el MPLAB. Por ejemplo, este programa está destinado a incrementar un contador temporario pero, *¿cómo puede asegurarse que se está produciendo dicho incremento?* Una manera es abrir e inspeccionar una ventana de registro de archivo. Puede hacerlo ejecutando el ítem de menú ***"Ventana>registros de archivo (Window>File Register)"***. Aparecerá una pequeña ventana con todos los registros de archivo o el RAM del 16F84.

Presione F7 (ejecute instrucción por instrucción, si piensa anularlo) varias veces, y observe la actualización de valores en la ventana de registro de archivo. Hemos colocado la variable del contador en la ubicación de dirección 0x0C. Mientras el contador temporario se incremente, este incremento se reflejará en la ventana de registro de archivo. Los registros de archivo cambian de color cuando su valor cambia, de modo que los cambios puedan advertirse fácilmente en la inspección. De todos modos, en muchos programas complejos, varios valores pueden cambiar, así resultará más difícil focalizar las variables que le interesan. Usando una ventana de observación especial, este problema puede solucionarse.

Ahora, ya sabemos mejor qué es lo que estamos haciendo:

Programamos algo que incrementa en "1" cada vez que viene una señal" y podemos verificarlo virtualmente, antes de programar el PIC.

Creación de una Ventana de Observación

Ejecute el ítem de menú **"Ventana>Nueva ventana de observación (Window>New Watch Window)"**. Aparecerá el diálogo "Agregar Símbolo de Observación (Add Watch Symbol)" (figura 45). Tipee "c1" en la caja de nombre de símbolo para que la lista se despliegue hasta el símbolo deseado. Selecciónelo, presione el botón ***"Agregar (Add)"***, y luego el botón ***"Cerrar (Close)"***. Aparecerá en su escritorio MPLAB una ventana de observación que mostrará el valor actual del valor "c1" del contador temporario.

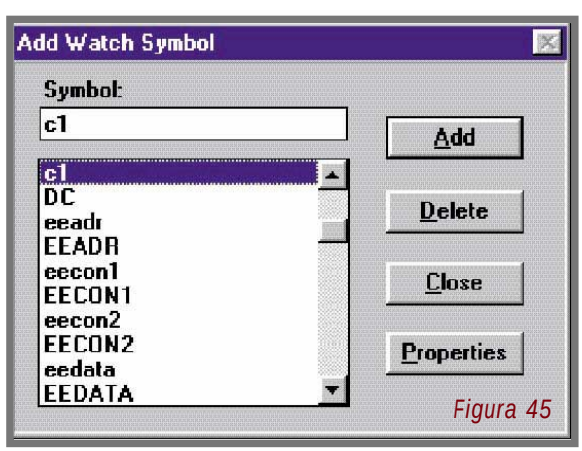


Figura 45

Presione F7 varias veces para advertir cómo se actualiza la ventana de observación mientras el valor del contador se incrementa. Si dejó la ventana de registro de archivo abierta, la misma también será actualizada (mueva una de ellas para que pueda ver las dos en la pantalla).

Puede guardar la ventana de observación y sus configuraciones al ejecutar el ítem ***"Guardar observación (Save Watch)"*** debajo del botón del sistema, ubicado en el ángulo superior izquierdo de la ventana de observación. Al clicar este botón, se desplegará un menú en cascada. Seleccione "Guardar observación" y aparecerá el diálogo de exploración estándar ubicado en el directorio del proyecto. Elija algún nombre arbitrario y presione "Aceptar (OK)".

Si no nombra la ventana de observación, el MPLAB lo hará por usted. La ubicación y el estado en la pantalla de la ventana abierta o cerrada serán guardados con el proyecto, de modo que la próxima vez que abra su proyecto, sus ventanas de observación aparecerán restauradas.

Nota: También puede editar ventanas de observación luego de crearlas. Use el botón del sistema y seleccione ***"Agregar Observación (Add Watch)"*** para que aparezca un diálogo mediante el cual podrá agregar más ítems. Con la tecla ***"Ins"*** podrá hacer lo mismo. Si desea borrar un ítem, selecciónelo y presione la tecla Suprimir (Delete); la observación referida desaparecerá de la ventana. Puede seleccionar ***"Editar observación (Edit Watch)"*** en el menú del sistema para cambiar el modo en el cual se muestra el ítem (en hex, binario, como una variable de 16-bit en vez de 8-bit, etc.).

Cómo se Marca un Punto de Interrupción

Presione F5 ("Debug>Ejecutar>Detener") para asegurarse que el procesador del simulador se ha detenido. Clique dentro de la ventana del código fuente la línea siguiente al rótulo "start", que dice "movlw 0'09'". Presione el botón derecho del mouse para que aparezca el menú de la figura 46.

Clique el ítem de menú ***"Punto(s) de interrupción (Break Point)"***. El menú desaparecerá y la línea donde se ubicó el cursor cambiará de color, para indicar que ha sido establecido un punto de interrupción en dicha ubica-

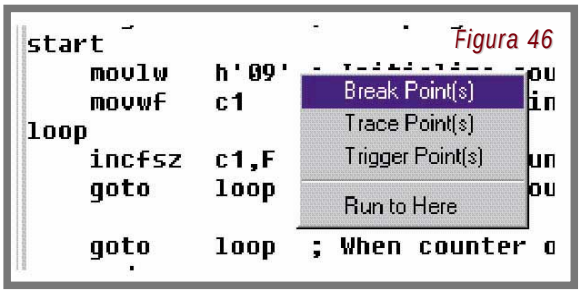


Figura 46

ción. Presione F6 o ejecute el ítem de menú ***"Debug>Ejecutar>Reset (Debug>Run>Reset)"*** para resetear el sistema. Luego, presionando F9, ejecute el sistema. El programa se ejecutará y se detendrá en la instrucción ubicada luego del punto de interrupción. "c1", como aparece en la ventana de observación o en la de registro de archivo, si la tiene aún abierta, reflejará el estado reset de cero, el modo instrucción por instrucción ejecutará la carga y c1 luego reflejará un valor de 0x09. Presione F9 varias veces y advierta que la barra de estado cambia de color mientras el programa se esté ejecutando, y luego retorna a su color original cuando el procesador se detiene.

Algunas Sugerencias:

PUNTOS DE INTERRUPCION - Puede marcar puntos de interrupción en la ventana "Ventanas>Memoria de programa (Window>Program Memory)", en la ventana de archivo fuente (en este caso tutor84.asm), o en la ventana "Ventanas>Listado Absoluto (Windows>Absolute)".

ARCHIVOS FUENTE - Use "Ventana>Ventana de Proyecto (Window>Projet Window)" para que aparezca una lista de sus archivos fuente. Puede hacer un doble click sobre el nombre de un archivo para trasladarlo al editor.

ERRORES MPASM - Si el MPASM le da un error, haga un doble click sobre el error en la ventana de error para ir al error en el código fuente. Si tiene múltiples errores, siempre elija el primer error —generalmente un error causa errores subsiguientes y al reparar el primero corregirá los restantes.

CONFIGURACIÓN DE BITS Y EL MODO DEL PROCESADOR - La configuración de bits en el archivo fuente no determinará el modo del procesador para el simulador (o los emuladores).

Use "Opciones>Configurar Procesador>Hardware (Options>Procesor Setup>Hardware)" para estas configuraciones. Aún cuando puede establecer estos bits en el archivo fuente del MPASM o del MPLAB-C17, el MPLAB no cambia automáticamente los modos. Por ejemplo, la configuración de bit Activar Observación de Dog Timer puede hacerse de tal modo que, cuando programe un dispositivo, el Dog Timer sea activado. En el MPLAB también necesitará acceder al diálogo "Opciones>Configurar Procesador>Hardware" para activar el WDT para el simulador o el emulador. Esto le permitirá hacer un seguimiento de errores con el WDT activado o desactivado sin cambiar su código fuente.

OPCIONES - Use "Opciones>Configurar Entorno (Options>Enviroment Setup)" para hacer lo siguiente:

** Establecer teclas de Mapa Europeo para funciones MPLAB y caracteres ASCII especiales*

** Cambiar la fuente de la pantalla o el tamaño de la fuente*

** Posicionar la barra de herramientas a un lado o al pie de la pantalla*

** Modificar la barra de herramientas*

** Cambiar la cantidad de caracteres desplegados para los rótulos.*

ARCHIVOS MAPA - Use el diálogo "Proyecto>Editar Proyecto" y cambie las Propiedades del Nodo del MPASM para producir un archivo MAP llamado tutor84.map. Luego de construir el proyecto, busque el tutor84.map para ver información de la construcción.

MENUES DESACTIVADOS - Si encuentra menúes desactivados (opacados), verifique que no haya ingresado el modo "sólo Editor (Editor Only)".

Si está seguro de que ha realizado correctamente la configuración, intente salir del MPLAB y reinicie el programa. ✪

Importante: Este manual se complementa con bibliografía sobre reparación de equipos sin cargo. Por ser comprador de esta edición, Ud. tiene acceso a información y programas **GRATIS**. Para acceder a los mismos, vaya a nuestro sitio:**www.webelectronica-.com.ar**. Luego haga click en el ícono password y digite la clave: **aiwa15**

ISBN: 987-9227-92-1

Editorial Quark SRL - Herrera 761, (1295) Bs. As. Argentina - **Director:** Horacio D. Vallejo - **Tel.:** (005411) 4301-8804
En Internet: www.webelectronica.com.ar - **Impresión:** Talleres Gráficos Conforti, Bs. As. - **octubre de 2002**.
Distribución en Argentina: Capital - Carlos Cancellaro e Hijos SH, Gutenberg 3258, Capital - Interior: Bertrán S.A.C., Av. Vélez Sarsfield 1950, Capital