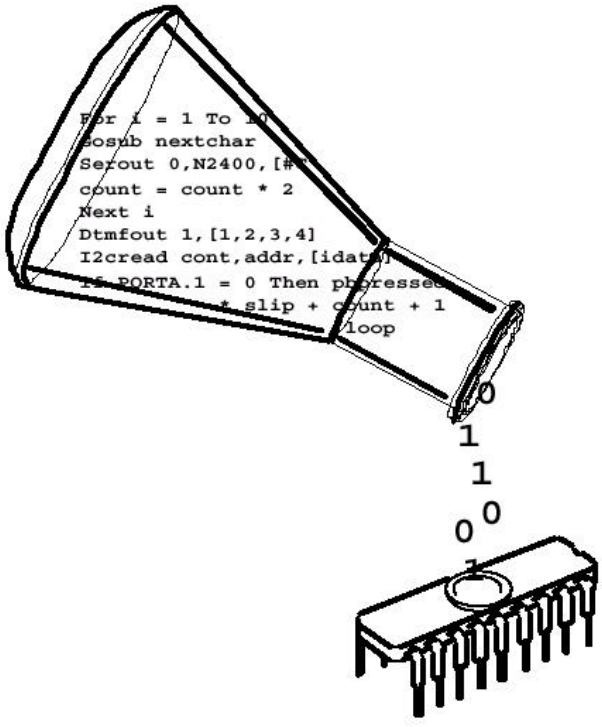
Traducido al castellano por Luis Frino
Daprotis 7292 Tel 0223 4792596 Mar del Plata Argentina
www.frino.com.ar donino@sinectis.com.ar

Compilador PicBasic Pro



microEngineering Labs, Inc.

1. INTRODUCCION

El compilador PicBasic Pro (PBP) es nuestro lenguaje de programación de nueva generación que hace mas fácil y rápido para usted programar micro controladores Pic micro de Microchip Technology .
El lenguaje Basic es mucho más fácil de leer y escribir que el lenguaje ensamblador Microchip.

El PBP es similar al "BASIC STAMP II" y tiene muchas de las librerías y funciones de los BASIC STAMP I y II. Como es un compilador real los programas se ejecutan mucho más rápido y pueden ser mayores que sus equivalentes STAMP.
PBP no es tan compatible con los BASIC STAMP como nuestro compilador PicBasic es con el BS I. Decidimos mejorar el lenguaje en general. Una de estas decisiones fue agregar IF ...THEN...ELSE...ENDIF en lugar de IF.. THEN (GOTO) de los Stamps. Estas diferencias se ven luego en este manual.
PBP por defecto crea archivos que corren en un PIC 16F84-04/P con un reloj de 4 Mhz. Solamente muy pocas partes son necesarias capacitores de dos capacitores de 22 pf para el cristal de 4Mhz un resistor de 4.7K en el pin/MCLR y una fuente de 5 volt. Otros micros PIC además del 16F84, así como otros osciladores de frecuencias distintas pueden ser usados por este compilador.

1.1. LOS MICRO

El PBP produce código que puede ser programado para una variedad de micro controladores PIC que tengan de 8 a 68 pins y varias opciones en el chip incluyendo convertidores A/D, temporizadores y puertos seriales.

Hay algunos micros PIC que no trabajaran con el PBP, por ejemplo las series PIC 16C5X incluyendo el PIC 16C54 Y PIC 15C58. Estos micro PIC están basados en el viejo núcleo de 12 bit en lugar del núcleo más corriente de 14 bit. El PBP necesita alguna de las opciones que solamente están disponibles con el núcleo de 14 bit como el stack (pila)de 8 niveles.

Hay muchos micros PIC, algunos compatibles pin a pin con la serie 5 X, que pueden ser usados con el PBP. La lista incluye:

PIC16C554, 556, 558, 61, 62(A),
620, 621, 622, 63, 64(A), 65(A),
71, 710, 711, 715, 72, 73(A), 74(A),
84, 923, 924,
PIC16F83 y 84,
PIC12C671 y 672
PIC14C000,

Microchip sigue agregando otros. Para reemplazo directo de un PIC166C54 o 58, el PIC16C554, 558, 620 y 622 funcionan bien con el compilador y tienen aproximadamente el mismo precio.*

Para propósitos generales de desarrollo usando el PBP, el PIC16F84 (o PIC16C84 si el F84 no está disponible) es la elección común de micro PIC. Este micro controlador de 18 pin usa tecnología flash (EEPROM) para permitir rápido borrado y reprogramación para acelerar la depuración de programas.

Con el clic de un mouse en el software, el PIC16F84 puede ser borrado instantáneamente y luego ser reprogramado una y otra vez. Otros micros PIC de las series 12C67X, 16C55X, 16C6X, 16C7X y 16C9X son programables una vez (OTP) o tienen una ventana de cuarzo en su parte superior (JW) para permitir el borrado exponiéndolo a una luz ultravioleta durante varios minutos.

El PIC16F84 (y 'C84) además, contiene 64 bytes de memoria de datos no volátil que puede ser usada para archivar el datos de programa y otros parámetros, aun cuando no haya energía. A ésta área de datos, se puede acceder simplemente usando las órdenes "Read" y "Write" del PBP. (El código programa es permanentemente guardado en el espacio de código del micro PIC, tanto si hay o no energía.)

Usando el 'F84 para el testeo del programa inicial , el proceso de depuración puede ser más rápido. Una vez que las rutinas principales de un programa estén operando satisfactoriamente, se puede utilizar un micro PIC con mayor capacidad o las opciones expandidas del compilador.

Si bien muchas de las opciones del micro PIC serán discutidas en este manual , para completar la información del micro PIC, es necesario obtener las apropiadas hojas de datos del micro PIC o el CD-ROM de Microchip Technology. Refiérase al Apéndice B para informarse como contactarnos.
El precio de venta es dictado por Microchip Technology Inc. Y sus distribuidores.

1.2. ACERCA DE ESTE MANUAL

Este manual no es un tratado completo del lenguaje BASIC. Describe el conjunto de instrucciones del micro PIC y brinda ejemplos de cómo utilizarlo. Si no está familiarizado con la programación de BASIC , deberá obtener un libro sobre dicho tema. O intentarlo directamente. BASIC está diseñado como un lenguaje fácil de utilizar y, hay ejemplos adicionales de programas en el disco que pueden ayudarlo a comenzar.
La próxima sección de este manual abarca la instalación del PBP y la escritura del primer programa. Siguiendo a esto, una sección que describe diferentes opciones para la compilación de programas. Después son explicadas las bases de la programación , seguidas por una sección de referencia en donde cada comando del PBP es detallada en una lista. La sección de referencia muestra cada prototipo de comando, una descripción del comando y algunos ejemplos. Las llaves ({}), indican los parámetros opcionales.
El resto del manual, provee información para programadores avanzados - todo el trabajo interno que el compilador hace.

2. EMPEZANDO DESDE EL PRINCIPIO

2.1. INSTALACIÓN DEL SOFTWARE

El software PBP debe ser copiado a su disco rígido antes de usarlo. Cree un subdirectorio en su disco llamado PBP u otro nombre a su elección, tipeando md PBP
en el prompt de DOS. Copie todos los archivos desde el diskette adjunto ,en el subdirectorio recientemente creado, tipeando:
xcopy a:*.* PBP /s
La opción /s asegurará que todos los subdirectorios necesarios serán creados dentro del subdirectorio PBP.
Si el archivo es comprimido (.ZIP) o ejecutable (.EXE), usted necesita descomprimirlo (unzip) usando la opción -d para asegurarse que los subdirectorios sean recreados.
Asegurese que FILES y BUFFERS sean fijados en por lo menos 50 en su archivo CONFIG.SYS . Dependiendo de cuantos FILES y BUFFERS ya estén en uso por su sistema, puede ser necesario aumentar su número.

2.2. SU PRIMER PROGRAMA

Para operar el PBP , necesitará un editor ó procesador de texto para crear su programa fuente , algún tipo de programador de micros PIC como el *EPIC Plus Pocket PICmicro Programmer* y el propio PBP. Por supuesto , también necesita un PC.

La secuencia de eventos es similar a la siguiente :

Primero cree el archivo fuente BASIC para el programa , usando su editor o procesador de texto preferido. Si lo desea , EDIT (incluído en DOS) ó NOTEPAD (incluído en WINDOWS) , pueden ser utilizados.El nombre del archivo fuente debe terminar con la extensión .BAS (pero no es excluyente). El archivo de texto creado debe ser texto ASCII puro .No debe contener códigos especiales insertados por procesadores de texto para sus propósitos específicos .Normalmente se tiene la opción de grabar el archivo como texto ASCII puro en la mayoría de los procesadores de texto.

El siguiente programa provee un buen primer testeo de un micro PIC en el mundo real .Puede tipearlo o simplemente obtenerlo del subdirectorio SAMPLES incluído en el diskette distribuido con el PBP .El archivo es BLINK.BAS . El archivo fuente BASIC debe ser creado o movido al mismo directorio donde se encuentra el archivo PBP.EXE

Ejemplo de programa para hacer parpadear un LED conectado al puerto PORTB.0 , aproximadamente una vez por segundo.

```
loop: high PORTB.0      ´ enciende el LED
pause 500               ´ demora de .5 segundos
low PORTB.0             ´ apaga el LED
pause 500               ´ demora de .5 segundos
goto loop               ´ vuelve a loop y hace parpadear el LED para siempre
end
```

Una vez que Ud. esté convencido que el programa que ha escrito funcionará sin errores puede compilarlo ingresando PBP seguido del nombre de su archivo de texto en el prompt de DOS .Por ejemplo , si el archivo de texto que Ud. creo se llama BLINK.BAS , ingrese :

PBP blink

El compilador mostrará un mensaje de inicialización y procesará su archivo .Si lo acepta , creará un archivo de código fuente ensamblado (en este caso BLINK.ASM) y automáticamente invocará al ensamblador para completar la tarea .Si todo funciona bien , se crea un archivo de código microPIC (en este caso BLINK.HEX). Si existen errores , se emitirá un listado de los mismos , que deberán ser corregidos en su archivo fuente BASIC antes de ser compilados nuevamente.

Para ayudarlo a asegurarse que su archivo original funcione sin errores , es mejor comenzar escribiendo y probando pequeñas partes de su programa y no escribir 100000 líneas de programa y luego tratar de depurarlas de principio a fin.

Si Ud, no le indica otra cosa , el PBP ,por defecto , crea código para el PIC16F84 . Para compilar códigos para otros micros PIC , simplemente use la opción -p en la línea de comandos , como se describe mas adelante , para especificar otro tipo de procesador .Por ejemplo si intenta usar el programa BLINK.BAS en un PIC16C74 , compile usando el comando

PBP - p16c74 blink

2.3. PROGRAMANDO EL MICRO

Hay otros dos pasos para colocar su programa compilado dentro del micro controlador Picmicro y testearlo.

El PBP genera archivos de 8 bit standard de INTEL (.HEX) que pueden ser usados con cualquier programador de microsPic incluyendo nuestro programador EPIC Plus Pocket PICmicro Programmer.
Los microPic no pueden ser programados con cables de programación BASIC Stamp.
El siguiente es un ejemplo de cómo un microPic puede ser programado usando nuestro programador EPIC .

Asegúrese de que no haya microsPic instalados en el zócalo de programación del programador EPIC.
Conecte el programador EPIC al puerto paralelo de impresora del PC usando un cable de impresora DB25 macho a DB25 hembra.

Enchufe el adaptador de AC en la pared y luego en el programador EPIC (o coloque dos baterías de 9 volt en el programador y conecte el jumper “Batt ON”).

El LED en el programador puede estar encendido o apagado en este momento. No coloque un microPic en el zócalo de programación cuando el LED esté encendido o antes de que el software de programación dé comienzo.

Ingrese: EPIC.

En el prompt de DOS para comenzar el software de programación. El software EPIC puede ser iniciado desde una sección de DOS pura o desde una sección de DOS bajo Windows ó OS/2. (No se recomienda utilizarlo bajo Windows porque altera el sistema de tiempos y juega con los puertos lo que puede causar errores de programación).

El software de EPIC busca donde está conectado el programador EPIC y lo deja listo para programar un microPIC. Si no se encuentra un programador EPIC chequea todas las conexiones y verifica que no hay un microPic o algún adaptador conectado al programador. Tipeando: EPIC / ?

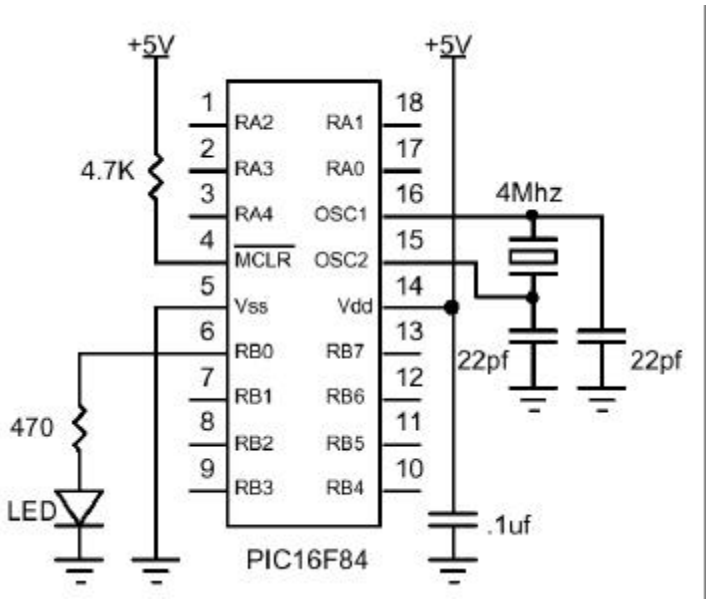
En el prompt de DOS mostrará una lista de opciones disponibles para el software de EPIC.
Una vez que se muestra la pantalla de programación, use el mouse para hacer click sobre OPEN archivo ó apriete Alt-O en su teclado. Use el mouse o el teclado para seleccionar BLINK.HEX ó cualquier otro archivo que usted desee programar dentro del microPic desde la caja de diálogo.

El archivo se cargará y usted verá una lista de números en la ventana de la izquierda. Esto es su programa en código de microPic. A la derecha de la pantalla se muestra información de la configuración que será programada dentro del microPic. Verifique que es correcta antes de proseguir.
En general, el oscilador debe ser colocado en XT para un cristal de 4 Mhz y el Watchdog Timer debe ser colocado en ON para programas de PBP. Mas importante, Code Protect debe ser OFF cuando se programa cualquier micro con ventana de cuarzo (JW). Usted no puede borrar un microPic con ventanas que ha sido protegido con código.

Cuando todo parece maravilloso, es tiempo de insertar un microPic dentro del zócalo de programación y hacer click sobre Program ó teclear Alt-P en el teclado. El microPic primero será chequeado para asegurarse de que esté en blanco y luego su código será programado dentro de él. Si el microPic no está en blanco y es un 16F84 ó un 16C84, usted simplemente puede elegir programarlo sin borrarlo primero. Una vez que se completó el programa y se apagó el LED es tiempo de probar su programa.

2.4 ESTA VIVO

El esquema de ejemplo siguiente da una idea de los pocos elementos que se necesitan conectar a un microPic para hacerlo funcionar. Básicamente se necesita un resistor de pull-up en la línea /MCLR, un cristal de 4 Mhz con 2 capacitores y una fuente de 5 volt. Hemos agregado un LED y un resistor para proveer la salida para el programa BLINK.



Construya y verifique este simple circuito en una tarjeta y enchufe el microPic que usted programó. Nuestra línea de tarjetas PICProto es perfecta para este tipo de cosas.

Conecte una fuente de alimentación su microPIC arrancará parpadeando el LED aproximadamente una vez por segundo. Si no lo hace verifique todas las conexiones y asegúrese que hay 5 volt en los pins apropiados de su microPic

Con este simple comienzo ,puede crear su propio mundo de aplicaciones.

2.5. TENGO PROBLEMAS

Los programas mas comunes para hacer funcionar los microPic involucran asegurarse que los pocos componentes externos sean de valor apropiado y están conectados adecuadamente. Las siguientes son algunas claves para ayudar a hacer funcionar todo.

Asegúrese que el pin /MCLR está conectada a 5 volt a través de algún tipo de circuito protector o simplemente con un resistor de 4.7K. Si deja el pin sin conectar su nivel flota y algunas veces el microPic puede fallar. El microPic tiene un circuito interno de power-on-reset (puesta a cero al encender) y en general solo es necesario un resistor interno de pull-up es adecuado. En algunos casos el microPic puede no arrancar apropiadamente y se necesita un circuito interno. En los manuales encontrará mayor información.

Asegúrese de usar un cristal en buen estado con capacitores de valor adecuados conectados a él . Los valores de los capacitores pueden ser difíciles de leer. Si los valores son muy diferentes el oscilador no arrancará ni trabajará en forma apropiada. Un cristal de 4 Mhz con 2 capacitores cerámicos de 22pf es un buen comienzo para la mayoría de los microPic. Una vez mas, busque en los manuales información adicional en éste tema.

Asegúrese de que la fuente de alimantación es apropiada para la tarea. Aunque el microPic consume muy poca potencia la fuente debe estar muy bien filtrada. Si el microPic está controlando dispositivos que consumen bastante corriente de su fuente cuando ellos se encienden o apagan pueden causar un glitch (ruido ó interferencia) en las líneas de alimentación causando que el microPic deje de funcionar apropiadamente. Aun un visor LED puede crear un drenaje instantáneo en una pequeña fuente de alimentación (como una batería de 9 volt) y causar el microPic falle.

Cheque las hojas de datos del microPic. Algunos dispositivos tienen opciones que pueden interferir con algunas operaciones. Los PIC16C62X (16C620, 621 y 622) son un buen ejemplo. Estos microPic tienen comparadores analógicos en PORTA. Cuando estos micros arrancan, PORTA se coloca en modo analógico. Esto hace que la función en PORTA trabaje de una manera no esperada. Para cambiar el pin a digital, simplemente agregue la línea.

CMCON = 7

Al principio de su programa .Cualquier micro ŒC con entradas analógicas como la serie16C7xx arrancará en modo analógico .Deberá colocarlos en modo digital si intenta usarlos en ese modo.

Otro ejemplo de desastre potencial es el pin 4 PORTA , exhibe un comportamiento inusual cuando es usado como salida .Esto sucede porque tiene una salida a colector abierto , en lugar del estado bipolar del resto de los pins .Esto significa que puede ser llevada a tierra cuando se coloca en 0 . pero “flotará” cuando se la coloque en 1 , en lugar de ir a un estado alto .Para hacer que este pin funcione de manera correcta , agregue un resistor de pull-up entre el pin y 5 volt . El valor del resistor puede estar entre 1 K y 33 K . dependiendo de la conexión de entrada .Este pin actua como cualquier otro cuando se lo usa como entrada.

Todos los pins del microPIC se colocan como entradas cuando arranca .Si necesita que un pin actue como salida , coloquelo como salida antes de usarlo ó use un comando de PBP que lo haga por Ud .Una vez más , puede revisar las hojas de datos de los microsPIC .

Comience en pequeño .Escriba programas cortos para probar opciones de las que no está seguro ó con las que puede tener problemas .Una vez que estas funcionen, puede seguir adelante.

Trate de hacer las cosas de otra manera . Algunas veces , aquello que trata de hacer debería funcionar , pero no lo hace . Normalmente hay mas de una forma de confeccionar un programa . Trate de llegar utilizando un ángulo diferente.

2.6. ESTILO DE CÓDIGO

Escribir programas legibles y mantenibles es un arte . Existen algunas técnicas simples que ud. debe seguir y se convertirá en un artista.

2.6.1 COMENTARIOS

Use comentarios . Aunque sea perfectamente obvio para Ud, lo que está escribiendo , alguien más puede leerlo (ó Ud. mismo más tarde) y puede no iener idea de lo que Ud. hizo .Aunque los comentarios usan espacio en el archivo fuente BASIC , no lo hacen en el microPIC .

Haga que los comentarios le digan algo útil acerca de lo que el programa está haciendo .Un comentario como " colocar pin 0 en 1 " simplemente explica la sintaxis del lenguaje , pero no le dice nada acerca de la necesidad de hacerlo . "Encender el LED de batería baja" puede ser más útil .

Un bloque de comentarios en el comienzo del programa y antes de cada sección de código puede describir que sucederá con más detalle que un simple espacio después de cada línea .No incluya un bloque de comentarios en lugar de comentarios individuales de línea; use ambos .

En el comienzo del programa que hará el programa , autor y fecha .Puede ser útil para listar informaciones de revisión y fechas . Especificar las conexiones de cada pin puede ser útil para recordar para que equipo en especial se programó .Si se utilizó un cristal no standard , u opciones especiales de compilación asegurese de indicarlo .

2.6.2. NOMBRES DE PIN Y DE VARIABLE

Haga que los nombres sean algo más coherente que Pin0 o B1 . Además de un uso libre de comentarios , nombres descriptivos ayuda mucho a la legibilidad . El siguiente fragmento lo demuestra :

```
BattLED var portb.0      ´ LED de batería baja
Level var byte           ´ la variable contiene el nivel de batería
If level < 10 then       ´ si el nivel de batería es bajo
High battLED             ´ enciende el LED
Endif
```

2.6.3. ETIQUETAS

Las etiquetas (labels) deben indicar algo mas significativo que " etigueta 1" ó "aquí" .Aún una etiqueta "loop" es más descriptiva (pero poco) .Usualmente la línea ó rutina a la que se está saltando hace algo único .Trate de dar un indicio de su función con la etiqueta y luego siga con un comentario .

2.6.4. GOTO

Trate de no usar demasiados GOTO . Aunque pueden ser un mal necesario , trate de minimizar su uso en lo posible .Trate de escribir su código en secciones lógicas y no ir saltando a cualquier lado . Usar GOSUB puede ser útil para esto .

3. OPCIONES DE LÍNEA DE COMANDO

3.1. USO

El PBP puede ser llamado desde la línea de comando del DOS usando el siguiente formato:
PBP opciones nombre de archivo.

Ninguna o varias opciones pueden ser usadas para modificar la manera en que PBP compila el archivo especificado. Las opciones comienzan con un signo menos (-) o una barra invertida (/). El carácter siguiente al signo menos o la barra es una letra que selecciona la opción. Puede haber más caracteres si la opción requiere mas información. Cada opción debe estar separada por un espacio, pero no puede haber ningún espacio dentro de una opción.

Opciones múltiples pueden ser usadas al mismo tiempo, por ejemplo:

PBP -p16c71 -ampasm blink

Causará que el archivo BLINK.BAS sea compilado usando MPASM sea usado como ensamblador y dirigido a un procesador PIC16C71.

El primer elemento que no comienza con un signo menos se toma como nombre de archivo. Si no se especifica extensión se usa la extensión .BAS por defecto. Si se especifica una ruta, en ese directorio es buscado el nombre del archivo. No importa donde se encuentre el archivo fuente, los archivos generados por PBP se colocan en el directorio actual.

Por defecto, PBP automáticamente arranca el ensamblador (PM.EXE) si la compilación se efectúa sin error. PBP espera encontrar PM.EXE en el mismo directorio que PBP.EXE. Si la compilación tiene errores o se usa la opción -S no arranca el ensamblador.

Si PBP es llamado sin parámetro o sin nombre de archivo se muestra una pantalla de ayuda.

3.2. OPCIONES

Opción	Descripción
A	Usa un ensamblador diferente
C	Inserta líneas fuente como comentarios dentro del archivo ensamblador
H(?)	Muestra pantalla de ayuda
I	Usa una ruta diferente para Include
L	Usa un archivo diferente para Library
O	Pasa la opción al ensamblador
P	Especifica el procesador
S	Saltea la ejecución del ensamblador cuando está hecho.
V	Modo con comentario

3.2.1 Opción -A

PBP tiene la posibilidad de usar tanto PM, que está incluído con PBP ó MPASM de Microchip's como ensamblador. Para especificar MPASM, use -ampasm en la línea de comandos

PBP -ampasm nombre de archivo

Si no se especifica un ensamblador en la línea de comando, se usa PM. Vea la sección de programación en lenguaje ensamblador para más información.

3.2.2. Opción -C

La opción -C causa que PBP inserte las líneas del archivo fuente como comentario en el archivo fuente en lenguaje ensamblador. Esto puede ser útil como una herramienta de depuración o una herramienta de aprendizaje ya que muestra la instrucción PBP seguida por las instrucciones en lenguaje ensamblador que genera.

PBP -C nombre de archivo

3.2.3 Opción -H ó -?

La opción -H ó -? causa que PBP muestre una pantalla de ayuda. Esta pantalla de ayuda también se muestra si no se especifica opción o nombre de archivo en la línea de comandos

3.2.4 Opción -I

La opción -I le permite seleccionar la ruta include usada por el PBP.

3.2.5 Opción -L

La opción -L permite seleccionar la librería usada por el PBP. Esta opción generalmente es innecesaria ya que el archivo de librería por defecto se coloca en un archivo de configuración para cada micro controlador. Para mas información acerca de la librería PBP, vea las secciones avanzadas de éste manual.

PBP -lpbpps2 nombre del archivo

Este ejemplo hace que PBP compile el nombre del archivo usando la librería PicStic2.

3.2.6 Opción -O

La opción -O causa que las letras que la sigan sean pasadas al ansamblador en su línea de comando como opciones. Algunas opciones de PM se muestran en la siguiente tabla:

PM opción	Descripción
OD	Genera listados, Tabla de símbolos y archivos de mapeado
OL	Solamente genera listados

PBP -ol nombre de archivo

Este ejemplo le dice a PBP que genere un archivo nombre de archivo.lst después de una compilación satisfactora.
Se puede pasar mas de una opción -O al ensamblador a la vez.
El manual del ensamblador Macro Picmicro en el disco contiene mas información.

3.2.7 Opción -P

Si no se le indica otra cosa, PBP compila programas para el PIC16F84. Si el programa está destinado a otro procesador su nombre debe ser especificado en la línea de comando usando la opción -P.

Por ejemplo si el programa PBP está destinado a un procesador PIC16C74 la línea de comando debería ser la siguiente:

PBP -p16c74 nombre de archivo

3.2.8 Opción -S

Normalmente cuando PBP compila exitosamente un programa automáticamente arranca el ensamblador. Esto se hace para convertir la salida ensamblada el PBP a una imagen ejecutable. La opción -S evita esto dejando la salida del PBP en un archivo .ASM.

Ya que -S evita que sea llamado el ensamblador todas la opciones que son pasadas al ensamblador usando la opción -O son anuladas.

PBP -S nombre de archivo

3.2.9 Opción -V

La opción -V coloca al PBP en modo de comentarios el que presenta mayor información durante la compilación del programa.

PBP -V nombre de archivo

4. BASES DEL PBP

4.1 IDENTIFICADORES

Un identificador es simplemente un nombre. Son usados en PBP como etiquetas de líneas y nombres de variables. Un identificador es cualquier secuencia de letras, dígitos y símbolos, aunque no deben comenzar con un dígito. Los identificadores no distinguen las letras mayúsculas de las minúsculas, por lo que etiqueta, ETIQUETA, Etiqueta son todas tratadas como equivalentes. Aunque las etiquetas pueden tener cualquier número de caracteres de longitud PBP solamente reconoce los primeros 32.

4.2 ETIQUETAS DE LÍNEA (LABELS)

Para marcar líneas que el programa puede desear referenciar con comandos GOTO ó GOSUB, PBP usa etiquetas de línea. PBP no permite número de línea y no requiere que cada línea sea etiquetada. Cualquier línea PBP puede comenzar con una etiqueta de línea que es simplemente un identificador seguido por un punto y coma (;)

```
here: Serout 0, N2400, [“Hello, World!”, 13, 10]
Goto here
```

4.3 VARIABLES

Variables es donde se guardan datos en forma temporaria en un programa PBP. Son creadas usando la palabra clave VAR. Pueden ser bits , bytes ó word. Espacio para cada variable es automáticamente destinado en la memoria del micro controlador por PBP. El formato para crear una variable es el siguiente:
Etiqueta VAR tamaño (.modificadores)
Etiqueta es cualquier identificador excluyendo palabras claves como se describe anteriormente. Tamaño es bit, byte ó word. Modificadores opcionales agregan control adicional acerca de cómo se crea la variable. Algunos ejemplos de creación de variables son:

```
Dog var byte
Cat var bit
W0 var word
```

No hay variables predefinidas de usuarios de PBP. Por razones de compatibilidad existen dos archivos que crean las variables standard usadas con BASIC stamps:“bs1defs.bas” y “bs2defs.bas”. Para usar uno de estos archivos agregue la línea

```
Include “bs1defs.bas”
ó
Include “bs2defs.bas”
```

cerca del comienzo del programa PBP. Estos archivos contienen numerosas declaraciones VAR que crean todas las variables de BASIC Stamps y definiciones de pin.

De cualquier manera, en lugar de usar estos archivos envasados le recomendamos que cree su propias variables usando nombres con significado para usted.

El número de variables disponibles depende de la cantidad de RAM en un dispositivo en particular y el tamaño de las variables y los arrays .PBP reserva aproximadamente 24 posiciones RAM para su propio uso. También puede crear variables temporarias adicionales para usar en ordenamiento de ecuaciones complejas.

4.4 ALIAS

VAR también puede ser usado para crear un alias para una variable. Esto es muy útil para acceder al interior de una variable.

```
fido var dog          ´ fido es otro nombre de dog
b0 var w0.byte0       ´ b0 es el primer byte de word w0
b1 var w1.byte1       ´ b1 es el segundo byte de word w0
flea var dog.0        ´ flea es bit0 de dog
```

Modificador	Descripción
BIT0 O 0	Crea alias al bit 0 de byte o word
BIT1 O 1	Crea alias al bit 1 de byte o word
BIT2 O 2	Crea alias al bit 2 de byte o word
BIT3 O 3	Crea alias al bit 3 de byte o word
BIT4 O 4	Crea alias al bit 4 de byte o word
BIT5 O 5	Crea alias al bit 5 de byte o word
BIT6 O 6	Crea alias al bit 6 de byte o word
BIT7 O 7	Crea alias al bit 7 de byte o word
BIT8 O 8	Crea alias al bit 8 de word
BIT9 O 9	Crea alias al bit 9 de word
BIT10 O 10	Crea alias al bit 10 de word
BIT11 O 11	Crea alias al bit 11 de word
BIT12 O 12	Crea alias al bit 12 de word
BIT13 O 13	Crea alias al bit 13 de word
BIT14 O 14	Crea alias al bit 14 de word
BIT15 O 15	Crea alias al bit 15 de word
BYTE0 O LOWBYTE	Crea alias al lowbyte de word
BYTE1 O HIGHBYTE	Crea alias al highbyte de word

4.5 ARRAYS (ARREGLOS)

Los arreglos de variables pueden ser creados en una manera similar a las variables.
Etiqueta VAR tamaño (número de elementos)

Etiqueta es cualquier identificador, excluyendo palabras claves, como se describió anteriormente. Tamaño es BIT, BYTE ó WORD. Número de elementos es cuantos lugares en el arreglo se desean. Algunos ejemplos de creación de arreglo son los siguientes:

sharks var byte[10]
fish var bit [8]

La primera ubicación dentro del arreglo es el elemento cero. En el arreglo fish anterior los elementos están numerados fish (0) a fish (7) conteniendo 8 elementos en total .
Dada la forma en que los arreglos están localizados en memoria hay límites de tamaño para cada tipo.

Tamaño	Número máximo de elementos
BIT	128
BYTE	64
WORD	32

Vea la sección de memoria para mas información.

4.6 CONSTANTES

Las llamadas constantes pueden ser creadas de manera similar a las variables. Puede ser mas conveniente usar un nombre de constante en lugar de un número constante. Si el número necesita ser cambiado, únicamente puede ser cambiando en un lugar del programa donde se define la constante. No pueden guardarse datos variables dentro de una constante.
Etiqueta CON expresión constante
Algunos ejemplos son:

Mice con 3
Traps con mice *1000

4.7 SÍMBOLOS

SYMBOL provee otro método para renombrar (darle alias) a variables y constantes. SYMBOL no puede ser usado para crear una variable. Use VAR para crear una variable
SYMBOL lion = cat ´ cat fue previamente creada usando VAR
SYMBOL mouse = 1 ´ igual que mouse con 1

4.8 CONSTANTES NUMÉRICAS

PBP permite definir constantes numéricas en tres bases: decimal, binario y hexadecimal. Valores binarios son definidos usando el prefijo "%" y valores hexadecimales usando el prefijo "$". Los valores decimales se toman por defecto y no requieren prefijo.

´ valor decimal 100
%100 ´ valor binario para el decimal 4.
$100 ´ valor hexadecimal para el decimal 256.

Para facilitar la programación, los caracteres son convertidos en sus equivalentes ASCII. La constante debe ser puesta entre comillas y contener sólo un caracter (de lo contrario, ellas son una sarta de constantes).

"A" ' ASCII valor para el decimal 65
"d" ' ASCII valor para el decimal 100

4.9. SARTA DE CONSTANTES:

PBP no provee capacidad de manejo de sartas, pero las sartas pueden ser usados con algunos comandos. Una sarta contiene uno o más caracteres y es delimitado entre comillas. No se soportan secuencias de escape para caracteres no-ASCII (aunque, la mayoría de los comandos PBP tienen este manejo incorporado)

"Hello" ' String (forma abreviada de "H", "e", "l", "l", "o")

Las sartas son usualmente tratadas como una lista de valores de caracteres individuales.

4.10. PINS:

A los pins se puede acceder de diferentes modos. El mejor camino para especificar un pin para una operación, es simplemente usar sus nombres PORT y un número de bit:
PORTB.1= ' Colocar PORTB, bit 1 a 1

Para recordar fácilmente para qué puede ser usado un pin, debe serle asignado un nombre usando el comando VAR. De esta manera, el nombre puede ser utilizado luego en cualquier operación:

```
Led var PORTA.O        ' Renombra PORTA.O como led
High led               ' Coloca led (PORTA:O) en valor alto
```

Para compatibilidad con el BASIC Stamp, los pins usados en los comandos del PBP pueden, además, ser referidos por un número, 0-15. Estos pins estan físicamente distribuidos sobre diferentes puertos del hardware del micro PIC, dependiendo de cuántos pins tiene el microcontroladores

N°. Pins del micro PIC	0-7	8-15
8-pin	GPIO*	GPIO*
18-pin	PORTB	PORTA*
28-pin (excepto 14C000)	PORTB	PORTC
28-pin (14C000)	PORTC	PORTD

40-pin	PORTB	PORTC

*GPIO y PORTA no tienen 8 I/O pins.

Si un conector no tiene ocho pins, como el PORTA, sólo los números de pin que existen pueden ser utilizados, por ejemplo 8-12. Usar los números de pin 13-15 no tendrá un efecto perceptible.

Este número de pin, 0-15, no tiene relación con el número físico del pin de un micro PIC. Dependiendo de cada micro PIC, el pin número 0 podría ser el pin físico 6,21 o 33, pero en cada caso esto apunta a PORTB.0 (o GPIO.0 para dispositivos 8-pin, o PORTC.0 para un PIC14C000).

Los pins pueden ser referenciados por un número (0-15), un nombre (Ej: Pin0, si uno de los archivos bsdefs.bas son incluídos o si usted los tiene definidos), o un nombre completo de bit (Ej: PORTA.1).
A cualquier pin o bit del microcontrolador se puede acceder usando el método anterior.

Los nombres de los pin (Ej: Pin0) no son automáticamente incluídos en su programa. En la mayoría de los casos, usted define los nombres de los pins como desee, usando el comando VAR:

Led var PORTB.3

Sin embargo, dos archivos de definicines han sido provistos para mejorar la compatibilidad del BASIC Stamp. Los archivos “bs1defs.bas” o “bs2defs.bas” pueden ser incluidos en el programa PBP para proveer nombres de pin y bit, compatibles con los nombres del BASIC Stamp.

Include “bs1defs.bas”
ó
Include “bs2defs.bas”

BS1DEFS.BAS define Pins , B0 - B13 , W0 - W6 y muchos otros nombres de pin y variables de BS1.
BS2DEFS.BAS define Ins , Outs , B0 - B25 , W0 - W12 y muchos otros nombres de pin y variables de BS1.

Cuando arranca un microPIC , todos los pins son colocados en entrada .Para usar un pin como salida , el pin o port debe ser colocado como salida o se debe usar un comando que automaticamente coloque al pin como salida.

Para colocar un pin o port como salida (ó entrada) debe dar valores al registro TRIS . Colocando el bit de TRIS como 0 , hace su pin una salida , y colocandolo en 1 lo hace una entrada .Por ejemplo :

TRISA = %00000000 ´ O TRISA = 0
Coloca todos los pins PORTA como salidas.

TRISB = % 11111111 ´ O TRISB = 1
Coloca todos los pins PORTB como entradas.

TRISC = % 10101010
Coloca todos los pins pares como salidas y los impares como entradas.
Cada bit individual puede ser manejado de la misma manera

TRISA.0 = 0
Coloca el PORTA , pin 0 como salida. Todos los demás pin permanecen sin cambio.

Los nombres de variables de BASIC Stamp Dirs, Dirh, Dirl y Dir 0 - Dir 15 no están definidos y no deben ser usados con el PBP, en su lugar debe ser usado TRIS, pero tiene el estado opuesto de Dirs.
Esto no trabajará en PBP

Dir0 = 1 ´ coloca el pin PORTB.0 como salida.
En su lugar
TRISB.0 = 0 ´ coloca el pin PORTB.0 como salida.

O simplemente use un comando que automáticamente coloque la dirección del pin.

4.11. COMENTARIOS

Un comentario de PBP comienza con la palabra clave REM o el apóstrofe (´). Todos los demás caracteres de esa línea se ignoran.

REM es una única palabra clave y no es una abreviación de REMark, por lo tanto, los nombres de variables pueden comenzar con REM (aunque REM por sí mismo no es válido).

4.12. DECLARACIONES MÚLTIPLES

Para permitir programas mas compactos y agrupamientos lógicos de comandos relacionados, PBP soporta el uso de (:) para separar comandos ubicados en la misma línea. Los siguientes dos ejemplos son equivalentes.

```
W2 = W0
W0 = W1
W1 = W2
```

Es lo mismo que:

```
W2 = w0 : W0 = W1 : W1 = W2
```

En los dos casos, el tamaño del código generado es el mismo.

4.13. CARÁCTER DE EXTENSIÓN DE LÍNEA

El número máximo de caracteres que puede aparecer en una línea PBP es 256. Declaraciones muy largas pueden ser extendidas a la línea siguiente usando el carácter (_) al final de cada línea a ser continuada.

```
Branch B0 , [label0, label1, label2, _
Label3, label4]
```

4.14. INCLUDE

Se puede agregar archivos fuente BASIC a un programa PBP usando INCLUDE. Usted puede tener su rutina standard, definiciones u otros archivos que desee guardar en forma separada. Los archivos de definición de modo serial y de stamp son ejemplo de esta. Estos archivos pueden ser incluidos en programas donde ser necesario, pero no en programas donde no se los necesita.

Las líneas de código fuente del archivo incluido son insertadas dentro del programa exactamente donde se coloca el INCLUDE.

```
INCLUDE “modedefs.bas”
```

4.15. DEFINE

Algunos elementos, como el oscilador y las ubicaciones de los pin LCD, están predefinidas en PBP. DEFINE le permite a un programa PBP cambiar estas definiciones si así lo desea.

Define puede ser usado para cambiar el valor predefinido del oscilador, los pins de DEBUG y el baud rate y las ubicaciones de los pin LCD además de otras cosas. Estas definiciones deben estar en mayúsculas

DEFINE BUTTON_PAUSE 50	´ demora en el anti-rebote del botón en ms
DEFINE CHAR_PACING 1000	´ paso de la salida serial en us
DEFINE DEBUG_REG _PORTL	´ depuracion del pin port
DEFINE DEBUG_BIT 0	´ depuracion del pin bit
DEFINE DEBUG_BAUD 2400	´ depuracion del baud rate
DEFINE DEBUG_MODE 1	´ modo depuracion: 0=CIERTO,1=INVERTIDO
DEFINE DEBUG_PACING 1000	´ paso de depuracion en us
DEFINE HSER_RCSTA 90 h	´ setear registro receive
DEFINE HSER_TXSTA 20 h	´ setear registro transmit
DEFINE HSER_BAUD 2400	´ setear baud rate
DEFINE HSER_EVEN 1	´ usar solo si se desea paridad par
DEFINE HSER_ODD 1	´ usar solo si se desea paridad impar
DEFINE I2C_INTERNAL 1	´ usar para EEPROM interno en 16CEXX y 12CEXX
DEFINE I2C_SLOW 1	´ usar para OSC > 8Mhzcon dispositivos de velocidad standard
DEFINE LCD_DREG PORTB	´ port de data LCD
DEFINE LCD_DBIT 0	´ datos LCD comenzando en bit 0 o 4
DEFINE LCD_RSREG PORTB	´ port de selección de registro LCD
DEFINE LCD_RSBIT 4	´ bit de selección de registro LCD
DEFINE LCD_EREG PORTB	´ port de habilitacion LCD
DEFINE LCD_EBIT 5	´ bit de habilitacion LCD
DEFINE LCD_BITS 4	´ tamaño 4 u 8 de bus de LCD
DEFINE LCD_LINES 2	´ numero de lineas en LCD
DEFINE OSC 4	´ 3 4 8 10 12 16 20
DEFINE OSCCAL_1K 1	´ setea OSCCAL para PIC12C671
DEFINE OSCCAL_2K 1	´ setea OSCCAL para PIC12C672

4.16. OPERADORES MATEMÁTICOS

PBP efectua todas las operaciones matemáticas en urden jerárquico .Esto significa que existe precedencia para los operadores .Multiplicación y división son efectuados antes que suma y resta , por ejemplo..Para asegurarse que las operaciones son efectuadas en el orden que se desea , use paréntesis para agrupar las operaciones.

A = (B + C) * (D - E)

Todas las operaciones matemáticas se realizan sin signo y con una precisión de 16 bit.
Los operadores soportados son :

Operador matemático	Descripción
+	Suma
-	Resta
*	Multiplicacion
**	16 bits superiores de la multiplicacion
*/	16 nits medios de la multiplicacion
/	Division
//	Resto (módulo)
<<	Desplazamiento izquierdo
>>	Desplazamiento derecho
ABS	Valor absoluto
COS	Coseno
DCD	2m decodificador
DIG	Digito
MAX	Maximo *
MIN	Minimo *
NCD	Codificar
REv	Invertir bits
SIN	Seno
SQR	Raiz cuadrada
&	Bit inteligente AND
÷	Bit inteligente OR
^	Bit inteligente EXCLUSIVE OR
~	Bit inteligente NOT
& /	Bit inteligente NOT AND
÷ /	Bit inteligente NOT OR

^ /	Bit inteligente NOT EXCLUSIVE OR

La implementacion difiere del BASIC Stamp

4.16.1. Multiplicación

PBP efectua multiplicaciones 16 x 16 bits .El operador ´* ´ devuelve los 16 bit inferiores del resultado de 32 bit. Esta es la multiplicación típica encontrada en los lenguajes de programación.

El operador ´** ´devuelve los 16 bit superiores del resultado de 32 bit. Estos dos operadores pueden ser utilizados en conjunto para realizar multiplixaxiones de 16 x 16 bit que produzcan resultados de 32 bit .

```
W1 = W0 * 1000      ´ multiplica el valor de W0 por 1000 y coloca el resultado en W1
W2 = W0 ** 1000     ´ W0 por 1000 y coloca los 16 bit superiores (que deben ser 0 ) en W2
El operador */      ´ los 16 bit medios del resultado de 32 bit.
W3 = W1 */ W0       ´ multiplica W1 por W0 y coloca los 16 bit medios en W3
```

4.16.2 División

PBP efectua divisiones de 16 x 16 bit . El operador ´/ ´devuelve el resultado de 16 bit . El operador ´// ´ devuelve el resto (módulo del número).

```
W1 = W0 / 1000 ´ Divide el valor de W0 por 1000 y coloca el resultado en W1
W2 = W0 // 1000 ´ Divide el valor de W0 por 1000 y coloca el resto en W2
```

4.16.3 Desplazamiento

Los operadores ´<<´ y ´>> ´ desplazan un valor hacia la izquierda ó derecha respectivamente , 1 a 15 veces .Los bits desplazados se colocan en 0 .

```
B0 = B0 << 3    ´ Desplaza B0 tres lugares a la izquierda ( igual a multiplicar por 8)
W1 = W0 >> 1    ´ Desplaza W0 un lugar a la derecha y pone el resultado en W1 (igual a dividir por 2)
```

4.16.4. ABS

ABS devuelve el valor absoluto de un número.
Si un byte es mayor de 127 (bits altos) ABS devuelve un valor 256 - .
Si un word es mayor de 32767 (bits altos) , devuelve un valor 65536 - .

```
B1 = ABS B0
```

4.16.5. COS

COS el coseno en 8 bit de un valor dado. El resultado está dado en forma de dos complementos . (p.ej. -127 a 127) .Usa una tabla de cuarto de onda para encontrar el resultado . El coseno comienza con un valor en radianes binarios , 0 a 255 , en lugar de los comunes 0 a 358 grados.

```
B1 = COS B0
```

4.16.6. DCD

DCD devuelve el valor decodificado de un número de bit . Cambia un número de bit (0 a 15) por un número binario con ese bit seteado en 1. Todos los demás bits son 0 .

B0 = DCD 2 ´ setea B0 como % 00000100

4.16.7. DIG

DIG devuelve el valor de un dígito decimal . Simplemente se le indica el número de dígito a conocer (0 - 4 , siendo 0 el primero de la derecha) y ya está.

```
B0 = 123          ´ setea B0 en 123
B1 = B0 DIG 1     ´ setea B1 en 2 (dígito 1 de 123)
```

4.16.8. MAX y MIN

MAX y MIN devuelven el máximo y mínimo ,respectivamente , de dos números . Se usan normalmente para limitar números a un valor.

```
B1 = B0 MAX 100       ´ setea B1 al mayor de B0 y 100 (B0 debe estar entre 100 y 255)
B1 = B0 MIN 100       ´ setea B1 al menor de B0 y 100(B1no puede ser mayor de 100)
```

4.16.9. NCD

NCD devuelve el número de prioridad de bit codificado (1-16) de un valor . Se usa para encontrar el bit codificado con 1 de un valor dado . Devuelve 0 si no existen bits con valor 1 .

```
B0 = NCD %01001000 ´ setea B0 en 7
```

4.16.10. REV

REV invierte el orden de los bits inferiores de un valor .El número de bits a ser invertidos es de 1 a 16 .

```
B0 = %10101100 REV 4 ´ setea B0 a %10100011
```

4.16.11. SIN

SIN el seno en 8 bits de un valor . El está dado en dos complementos (p.ej. -127 a 127) .Usa una tabla de cuarto de onda para encontrar el resultado . Comienza con un valor en radianes binarios , 0 a 255 , en lugar de los usuales 0 a 359 grados.

```
B1 = SIN B0
```

4.16.12. SQR

SQR devuelve la raíz cuadrada de un valor . Como PBP dolo trabaja con enteros , el resultado será siempre un entero en 8 bits no mayor que el resultado actual .

```
B0 = SQR W1 ´ setea B0 con la raíz cuadrada de W1
```

4.16.13. Operadores de bit inteligente (BITWISE)

Estos operadores actuan sobre cada bit de un valor en forma booleana .Pueden ser usados para aislar bits o para agregar bits dentro de un valor .

```
B0 = B0 & %00000001    ´ aisla el bit 0 de B0
B0 = B0 ÷ %00000001    ´ setea el bit 0 de B0
```

```
B0 = B0 ^ %00000001     ´ invierte el estado del bit 0 de B0
```

4.17. Operadores de comparación

Se usan en declaraciones IF ... THEN para comparar una expresión con otra .Los operadores soportados son :

Operador	Descripción
= o ==	Igual
<> o !=	No igual
<	Menor
>	Mayor
<=	Menor o igual
>=	Mayor o igual

If i > 10 then loop

4.18. Operadores lógicos

Los operadores lógicos difieren de las operaciones de bit inteligente. Entregan un resultado CIERTO / FALSO de su operación .Valores 0 son tratados como falso. Cualquier otro valor es cierto. Se usan junto a operadores de comparación en una declaración IF .. THEN .Los operadores soportados son :

Operador	Descripción
AND o &&	AMD logico
OR o I I	OR logico
XOR o ^ ^	OR exclusivo logico
NOT AND	NAND logico
NOT OR	NOR logico
NOT XOR	NXOR logico

If (A == big) AND (B > mean) then run

Asegúrese de usar paréntesis para indicarle a PBP el orden en que quiere que se realicen las operaciones.

5. REFERENCIA DE DECLARACIONES PBP

@	Inserta una linea de codigo ensamblador
ASM...ENDASM	Inserta una seccion de codigo ensamblador
BRANCH	GOTO computado(equiv. a ON..GOTO)
BRANCHL	BRANCH fuera de pagina(BRANCH largo)
BUTTON	Anti-rebote y auto-repeticion de entrada en el pin especificado
CALL	Llamada a subrutina de ensamblador
CLEAR	Hace cero todas las variables
COUNT	Cuenta el numero de pulsos en un pin
DATA	Define el contenido inicial en un chip EEPROM
DEBUG	Señal asincronica de salida en un pin fijo y baud
DISABLE	Deshabilita el procesamiento de ON INTERRUPT
DTMFOUT	Produce tonos en un pin
EEPROM	Define el contenido inicial en un chip EEPROM
ENABLE	Habilita el procesamiento de ON INTERRUPT
END	Detiene la ejecucion e ingresa en modo de baja potencia
FOR...NEXT	Ejecuta declaraciones en forma repetitiva
FREQOUT	Produce hasta 2 freuencias en un pin
GOSUB	Llama a una subrutina BASIC en la etiqueta especificada
GOTO	Continua la ejecucion en la etiqueta especificada
HIGH	Hace alto la salida del pin
HSERIN	Entrada serial asincronica(hardware)
HSEROUT	Salida serial asincronica(hardware)
I2CREAD	Lee bytes de dispositivo I2C
I2CWRITE	Graba bytes en dispositivo I2C
IF..THEN..ELSE..ENDIF	Ejecuta declaraciones en forma condicional
INPUT	Convierte un pin en entrada
(LET)	Asigna el resultado de una expresion a una variable
LCDOUT	Muestra caracteres en LCD
LOOKDOWN	Busca un valor en una tabla de constantes
LOOKDOWN2	Busca un valor en una tabla de constantes o variables
LOOKUP	Obtiene un valor constante de una tabla
LOOKUP2	Obtiene un valor constante o variable de una tabla

LOW	Hace bajo la salida de un pin
NAP	Apaga el procesador por un corto periodo de tiempo
ON INTERRUPT	Ejecuta una subrutina BASIC en un interrupt
OUTPUT	Convierte un pin en salida
PAUSE	Demora (resolucion 1mseg.)
PAUSEUS	Demora (resolucion 1 useg.)
PEEK	Lee un byte del registro
POKE	Graba un byte en el registro
POT	Lee el potenciometro en el pin especificado
PULSIN	Mide el ancho de pulso en un pin
PULSOUT	Genera pulso hacia un pin
PWM	Salida modulada en ancho de pulso a un pin
RANDOM	Genera numero pseudo-aleatorio
RCTIME	Mide el ancho de pulso en un pin
READ	Lee byte de un chip EEPROM
RESUME	Continua la ejecucion despues de una interrupcion
RETURN	Continua en la declaracion que sigue al ultimo GOSUB
REVERSE	Convierte un pin de salida en entrada o uno de entrada en salida
SERIN	Entrada serial asincronica (tipo BS!)
SERIN2	Entrada serial asincronica (tipo BS2)
SEROUT	Salida serial asincronica (tipo BS1)
SEROUT2	Salida serial asincronica (tipo BS2)
SHIFTIN	Entrada serial sincronica
SHIFTOUT	Salida serial sincronica
SLEEP	Apaga el procesador por un periodo de tiempo
SOUND	Genera un tono o ruido blanco en un pin
STOP	Detiene la ejecucion del programa
SWAP	Intercambia los valores de dos variables
TOGGLE	Hace salida a un pin y cambia su estado
WHILE..WEND	Ejecuta declaraciones mientras la condicion sea cierta
WRITE	Graba bytes a un chip EEPROM
XIN	Entrada X - 10
XOUT	Salida X - 10

5.1. @

@ declaracion

Cuando se usa al comienzo de una línea , provee un atajo para insertar una declaracion en lenguaje ensamblador en un programa PBP. Este atajo se puede usar libremente para unir codigo ensamblador con declaraciones PBP .

```
i var byte
Rollme var byte
For i = 1 to 4
@ rlf _rollme, F              ‘ rotar byte a la izquierda una vez
next i
```

El atajo @ tambien se puede usar para incluir rutinas en lenguaje ensamblador en otro archivo .Por ejemplo :

@ Include “ fp.asm “

@ resetea a 0 la página del registro antes de ejecutar la instrucción en lenguaje ensamblador .La página del registro no debe ser alterada usando @
Vea la seccion de programacion del ensamblador para mayor informacion.

5.2. ASM..ENDASM

ASM
ENDASM

Estas instrucciones le dicen a PBP que el codigo entre estas dos líneas esta en lenguaje ensamblador y no debe ser interpretado como declaraciones PBP .Se puede usar estas dos instrucciones libremente para mezclar código ensamblador con declaraciones PBP .

El tamaño máximo para una sección de texto ensamblador es 8 K . Este es el tamaño máximo para el fuente actual , incluyendo comentarios , no el código generado .Si el bloque de texto es mayor , divídalo en múltiples secciones ASM ..ENDASM o incluyalo en un archivo separado .

ASM resetea a 0 el registro de página .Debe asegurarse que el registro de página sea 0 antes de ENDASM si el código de ensamblador lo ha alterado .
Vea la sección de programación de ensamblador para mas información .

```
ASM
Bsf PORTA,0 ;setea bit 0 en PORTA
Bcf PORTB.0 ;setea bit 0 en PORTB
ENDASM
```

5.3. BRANCH

BRANCH index , [etiqueta { ,etiqueta .....}]

Causa que el programa salte a una posición diferente , basada en una variable indexada. Es similar al ON...GOTO de otros BASIC.

Index selecciona una etiqueta de una lista .La ejecucion comienza en la etiqueta especificada .Por ejemplo , si Index es 0 , el programa salta a la primer etiqueta especificada en la lista ,si Index es 1 , salta a la segunda y así sucesivamente ..Si Index es mayor ó igual al número de etiquetas ,no se toma ninguna acción y la ejecución continúa con la declaración siguiente al BRANCH .Se pueden usar hasta 256 etiquetas en un BRANCH .

Etiqueta debe estar en la misma página de código que la instrucción BRANCH . Si no está seguro de esto , use BRANCHL .

BRANCH B4 , [dog,cat,fish]
´ igual que :
´ if B4=0 then dog (goto dog)
´ if B4=1 then cat(goto cat)
´ if B4=2 then fish (goto fish)

5.4. BRANCHL

BRANCHL Index , [etiqueta {, etiqueta ....}]

BRANCHL trabaja en forma similar a BRANCH ,haciendo que el programa salte a una localización determinada , basándose en una variable indexada .Las principales diferencias son que puede saltar a una etiqueta ubicada en otra página de código y que genera un código dos veces mayor en tamaño al de BRANCH ..Si está seguro que las etiquetas están en la misma página que el BRANCH ó si el microcontrolador no tiene más que una página de código (2K ó menos de ROM) , use BRANCH para minimizar el uso de memoria .

Index selecciona una etiqueta de una lista .La ejecucion comienza en la etiqueta especificada .Por ejemplo , si Index es 0 , el programa salta a la primer etiqueta especificada en la lista ,si Index es 1 , salta a la segunda y así sucesivamente ..Si Index es mayor ó igual al número de etiquetas ,no se toma ninguna acción y la ejecución continúa con la declaración siguiente al BRANCHL .Se pueden usar hasta 128 etiquetas en un BRANCHL .

BRANCHL B4 , [dog,cat,fish]

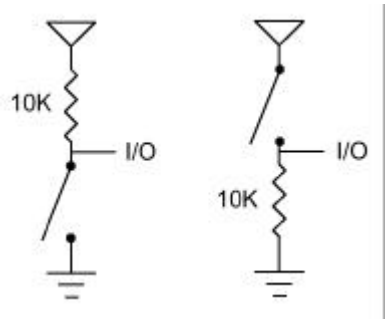
´ igual que :
´ if B4=0 then dog (goto dog)
´ if B4=1 then cat(goto cat)
´ if B4=2 then fish (goto fish)

5.5. BUTTON

BUTTON , Pin , Down , Delay , Rate , Bvar , Action , Etiqueta

Lee Pin y opcionalmente ejecuta anti-rebote y auto-repetición . Pin automáticamente se toma como entrada .Pin debe ser una constante , 0 - 15 , o una variable que contenga un número 0 - 15 (p.ej. B0) ó un número de pin (p.ej. PORTA ,0)

Down	Estado del pin cuando se oprime el pulsador (0 ..1)
Delay	Contador de ciclos antes de que comience la auto-repetición(0..255). Si es 0 ,no se efectua anti-rebote ni auto.repetición .Si es 255 se eliminan rebotes , pero no auto-repetición.
Rate	Valor de auto-repetición (0..255)
Bvar	Variable con tamaño de byte usada internamente para conteo de demoras y repeticiones,Dene ser inicializada a 0 antes de ser usada y no ser usada en cualquier lugar del programa.
Action	Estado del pulsador a ser actuado.
Etiqueta	La ejecución comienza en esta etiqueta si es cierto Action.



´ goto notpressed if button not pressed on Pin2
BUTTON PORTB ,2,0,100,10,b2,0,notpressed

BUTTON necesita ser usado dentro de un loop para auto-repetición para funcionar adecuadamente. BUTTON permite eliminar rebotes , demorando la ejecución de un programa por un período de milisegundos para permitir que los contactos se asienten .La demora por defecto es 10 ms. .Para cambiarlo a otro valor use DEFINE .

´ setea la demora de anti-rebote a 50 ms
DEFINE BUTTON_PAUSE 50

BUTTON_PAUSE debe estar en mayúsculas.
En general , es más fácil leer el estado del pin con un IF..THEN que usar el comando BUTTON .

IF PORTB,2 = 1 THEN notpressed

5.6. CALL CALL etiqueta

Ejecuta la subrutina ensamblador llamada etiqueta.

Normalmente se usa GOSUB para ejecutar una subrutina PBP .La principal diferencia entre GOSUB y CALL , es que con ésta última no se chequea la existencia de etiquetas hasta el momento de ensamblar .Usando CALL se puede acceder a una etiqueta en una sección de lenguaje ensamblador , lo que es inaccesible mediante PBP .

```
CALL pass        ´ ejecuta la subrutina ensamblada ,denominada _pass
```

5.7. CLEAR

CLEAR

Coloca en cero todos los registros en cada banco . Coloca en cero todas las variables ,incluyendo las del sistema .Esto no se hace automáticamente al comenzar un programa en PBP , como sucede en BASIC Stamps .Por lo general ,las variables deben ser colocadas en un estado inicial apropiado por el programa , y no usando CLEAR .
CLEAR ´ Coloca todas las variables en cero

5.8. COUNT COUNT Pin,Period,Var

Cuenta el numero de pulsos en un Pin , durante un período Period ,y guarda el resultado en Var .Pin es automáticamente colocado como entrada .Pin debe ser una constante , 0-15 , ó una variable que contenga un número de 0 a 15 (p.ej. B0) .ó un numero de pin .

La resolución de Period está dada en milisegundos .Sigue la frecuencia del oscilador basado en DEFINE OSC .

VCOUNT chequea el estado de Pin mediante un loop y cuenta las transiciones de bajo a alto .Con un oscilador de 4 Mhz chequea el estado del pin cada 20 us .Con un oscilador de 20 Mhz chequea el estado cada 4 us .De esto ,se infiere que la mayor frecuencia de pulsos que puede ser contada ,es de 25 Khz con un oscilador de 4 Mhz y de 125 Khz con un oscilador de 20 Mhz si la frecuencia tiene un ciclo útil del 50 % (los tiempos altos son iguales a los bajos).

```
´ cuenta el número de pulsos en Pin1 en 100 ms
COUNT PORTB.1,100,W1          ´ determinar la frecuencia en un Pin
COUNT PORTA.2,1000,W1         ‘ contar por 1 segundo
Serout PORTB.0,N2400, [ W1 ]  ‘ enviar el resultado por RS-232
```

5.9. DATA

DATA { @ location , } constante { ,constante }

Guarda constantes en un chip EEPROM cuando este dispositivo se programa por primera vez .Si se omite el valor opcional location , la primer declaración de DATA comienza a almacenarse en la dirección 0 y las declaraciones siguientes , en las direcciones siguientes .

Si existe un valor location este indica la dirección de comienzo donde se almacenará la información .Una etiqueta opcional se le puede asignar a la dirección de comienzo , para futuras referencias del programa , Constante puede ser una constante numérica ó una sarta de constantes .Solo se guarda el byte menos significativo del valor numérico , excepto que se use el modificador WORD .

Las sartas se guardan como bytes consecutivos de valores ASCII .No se agregan terminadores ni se completa el largo , DATA solo funciona con micro controladores con EEPROM incorporado como el PIC16F84 y PIC16C84 .Dado que el EEPROM es una memoria no-volatil , los datos permanecen intactos aún cuando se quite la energía .

Los datos se guardan dentro del EEPROM una sola vez en el momento en que se programa el micro controlador , no cada vez que se ejecuta el programa .
WRITE se usa para colocar los valores en el EEPROM en el momento de ejecución .

´ guardar 10,20 y 30 comenzando en la posición 5
DATA @5,10,20,30

´ asignar una etiqueta a un word en la próxima ubicación
dlabel DATA word $1234 ´ guarda $34, $12

´ saltear 4 posiciones y guardar 10 ceros
DATA (4) , 0 (10)

5.10. DEBUG

DEBUG item{ ,item ..}

Envia uno ó más items a un pin predefinido con un baud rate predefinido en formato standard asincrónico , usando 8 bits de datos ,sin paridad y con 1 bit de parada (stop bit) (8N1) ..El pin ,automáticamente se convierte en salida .

Si un signo (#) precede a un item ,se envía serialmente la representación ASCII para cada dígito .DEBUG soporta los mismos modificadores de datos que SEROUT2.

DEBUG es una de varias funciones seriales asincronicas pre-construidas .Es la más pequeña de las rutinas seriales generadas por software .Puede ser usada para enviar información de depuración (variables ,posición de marcadores , etc).a un programa terminal como HyperTerm .También se puede usar cuando se desee salida serial sobre un pin determinado y con un baud rate determinado .

Los pin y baud rate seriales son especificadas usando DEFINEs:

‘ Set Debug pin port
DEFINE DEBUG_REG PORTB ‘ Set Debug pin bit
DEFINE DEBUG_BIT 0

‘ Set Debug baud rate
DEFINE DEBUG_BAUD 2400

‘ Set Debug mode: 0= cierto, 1= invertido
DEFINE DEBUG_MODE 1

DEBUG asume un oscilador de 4 Mhz, cuando está generando su tiempo de bit. Para mantener el tiempo apropiado del baud rate con otros valores de osciladores, asegurese de DEFINE el seteo de OSC al valor de oscilador deseado.

En algunos casos, la tasa de transmisión de las instrucciones de DEBUG podrían presentar los caracteres demasiado rápidamente al dispositivo receptor. Un DEFINE agrega una demora de caracteres para las transmisiones seriales de salida. Esto permite un tiempo adicional entre los caracteres a medida que son transmitidos. La demora de caracter DEFINE permite un atraso de 1 a 65,535 microsegundos (.001 a 65.535 milisegundos) entre cada carácter transmitido.
Por ejemplo, para pausar 1 milisegundo entre la transmisión de cada caracter:

DEFINE DEBUG_PACING 1000

Si bien los chips convertidores de nivel RS-232 son comunes y económicos, gracias a la implementación de corriente RS-232 y a las excelentes especificaciones I/O del micro PIC ; la mayoría de las aplicaciones no requieren convertidores de nivel. Se puede usar TTL invertido (DEBUG_MODE =1) Se sugiere un resistor limitador de corriente (se supone que RS-232 es tolerante a los cortocircuitos).

´ Enviar el texto “B0=” seguido por el valor decimal de B0 y un avance de línea (linefeed) serialmente a la salida

```
DEBUG “B0=” ,dec B0,10
```

5.11. DISABLE

DISABLE

DISABLE interrumpe el procesamiento siguiente a la instrucción. Pueden ocurrir otras Interrupciones, pero el manipulador de interrupciones del BASIC en el PBP, no se ejecutará hasta que se encuentre un ENABLE .

DISABLE y ENABLE son pseudo-operaciones en el sentido que dan direcciones de compilador, en lugar de producir código. Vea ON INTERRUPT para más información.

```
DISABLE ‘ Deshablita interrupciones en el handler

Myint: led = 1     ‘ enciende el LED cuando es interrumpido
Resume             ‘ Vuelve al programa principal
Enable             ‘ ENABLE interrumpe después del handler
```

5.12. DTMFOUT

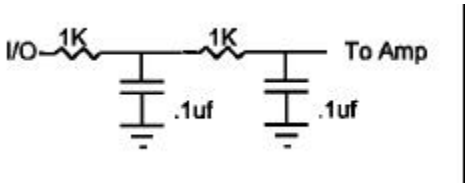
DTMFOUT Pin, { Onms ,Offms,} [Tone {,Tone}]

Produce una secuencia DTMF Touch Tone en Pin ,Pin automáticamente se convierte en salida . Pin debe ser una constante , 0 - 15 , ó una variable que contenga un número de 0 a 15 (p.ej. B0)ó un número de pin (p.ej. B0)

Onms es el número de milisegundos que suena cada tono y Offms es el número de milisegundos de pausa entre cada tono .Si no están especificados , por defecto Onms es 200 ms y Offms es 50 ms.

Tones tiene un valor de 0 - 15 .Los tonos de 0 - 9 son los mismos que en un teclado telefónico .Tone 10 es la clave * , Tone 11 es la clave # ,y los Tones 12 - 15 corresponden a las teclas extendidas A -D .

DTMFOUT usa FREQOUT para generar los tonos duales .FREQOUT genera tonos usando una forma de modulación de ancho de pulso .Los datos en bruto que salen del pin son bastante horribles .Usualmente se necesita algún tipo de filtro para suavizar la señal hasta una forma de onda senoidal quitándole algunas armónicas generadas:



DTMFOUT trabaja mejor con un oscilador de 20 Mhz .También puede trabajar con uno de 10 Mhz y aún con uno de 4 Mhz , aunque será muy difícil de filtrar y tendrá muy baja amplitud .Cualquier otra frecuencia causará que DTMFOUT genere una frecuencia proporcional al oscilador comparado a 20 Mhz , lo que no será muy útil para enviar touch tones .

```
´ enviar DTMF tones para 212 en Pin1
DTMFOUT PORTB.1 , [ 2,1,2 ]
```

5.13. EEPROM

EEPROM {Location ,} [constante {,constante ...}]

Guarda constantes en un chip EEPROM . Si se omite el valor opcional Location ,la primera declaración se guarda en la dirección 0 del EEPROM y las subsiguientes en las siguientes direcciones del mismo .Si se indica un valor Location , éste indica la dirección de comienzo para guardar los datos .

Constante puede ser una constante numérica ó una sarta de constantes .Solo se guardan los bytes menos significativos de los valores numéricos . Las sartas son guardadas como bytes consecutivos d valores ASCII .No se agregan automáticamente terminadores ,ni se completa el largo .

EEPROM solo trabaja con micro controladores con EEPROM incorporado como el PIC16F84 y PIC16C84 . Dado que el EEPROM es una memoria no volátil , los datos permanecerán intactos aún sin alimentación .

Los datos son guardados en el EEPROM solo una vez , cuando el micro controlador es programado , no cada vez que se ejecuta el programa .Se puede usar WRITE para colocar valores en el EEPROM en el momento de la ejecución .

´ Guardar 10 ,20 , 30 comenzando en la dirección 5

EEPROM 5, [10,20,30]

5.14. ENABLE

ENABLE

ENABLE interrumpe el procesamiento que fue previamente deshabiliyado con DISABLE ,siguiendo a esta instrucción.

DISABLE y ENABLE son pseudo-operaciones en el sentido que dan direcciones de compilador, en lugar de producir código. Vea ON INTERRUPT para más información.

```
DISABLE ‘ Deshablita interrupciones en el handler
Myint: led = 1 ‘ enciende el LED cuando es interrumpido
Resume ‘ Vuelve al programa principal
Enable ‘ ENABLE interrumpe después del handler
```

5.15. END

END

Detiene la ejecución del proceso y entra en modo de baja potencia .Todos los pins de I/O permanecen en el estado en que se encuentran ,END trabaja ejecutando una instrucción SLEEP continua dentro de un loop .

Un END , STOP ó GOTO deben ser colocados al final de un programa para evitar pasar del límite de la misma u comience nuevamente .END

5.16. FOR .. NEXT

FOR Count = Start TO End {STEP {-} Inc}
{Body}
NEXT {Count}

El loop FOR .. NEXT permite a los programas ejecutar un número de declaraciones (Body) un numero de veces , usando una variable como contador . Debido a su complejidad y versatilidad , es mejor describirla paso a paso .

El valor de Start se asigna a la variable índice ,Count , que puede ser una variable de cualquier tipo .
Se ejecuta el Body . Body es opcional y puede ser omitido (quizás por un loop de demora).
El valor de Inc es sumado a (ó restado si se especifica "-") Count .Si no se define una cláusula STEP , se incrementa Count en uno .

Si Count no pasó End ó desbordó el tipo de variable , la ejecución vuelve al paso 2) .
Si el loop necesita contar más de 255 (Count > 255) , se debe usar una variable de tamaño word .

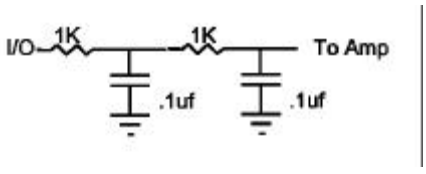
```
FOR i=1 TO 10 ´ cuenta de 1 a 10
Serout 0,N2400, [ # i," "] ´ envía cada número al pin0 en forma serial

NEXT i ´ vuelve y efectúa la próxima cuenta
Serout 0,N2400, [ 10 ] ´ envía un avance de línea
FOR B2=20 TO 10 STEP -2 ´ cuenta de 20 a 10 de a 2
Serout 0,N2400, [ # B2 , " "] ´ envía cada número al pin0 en forma serial

NEXT B2 ´ vuelve y efectúa la próxima cuenta
Serout 0,N2500, , [ 10 ] ´ envía un avance de línea
```

5.17. FREQOUT

FREQOUT Pin,Onms,Frequency1{,Frequency2}

Produce la ó las frecuencias especificadas en el Pin , durante Onms milisegundos .Pin se convierte automáticamene en salida .Pin puede ser una constante , 0-15 , ó una variable que contenga un número 0 - 15 .(p.ej. B0) ó un número de pin (p.ej. PORTA.0) .

Puede producir una ó dos frecuencias de 0 a 32767 Hz al mismo tiempo .
FREQOUT genera tonos usando una forma de modulación de ancho de pulso . Los datos en bruto que salen del pin son bastante horribles .Usualmente se necesita algún tipo de filtro para suavizar la señal hasta una forma de onda senoidal quitándole algunas armónicas generadas:



FREQOUT trabaja mejor con un oscilador de 20 Mhz .También puede trabajar con uno de 10 Mhz y aún con uno de 4 Mhz , aunque será muy difícil de filtrar y tendrá muy baja amplitud .Cualquier otra frecuencia causará que FREQOUT genere una frecuencia proporcional al oscilador comparado a 20 Mhz .

```
´ Enviar un tono de 1 Khz al Pin1 durante 2 segundos
FREQOUT PORTB.1 ,2000,1000
```

5.18. GOSUB

GOSUB etiqueta

Salta a la subrutina indicada en la etiqueta , guardando su dirección de regreso en la pila (stack) .A diferencia del GOTO , cuando se llega a un RETURN ,la ejecución sigue con la declaración siguiente al último GOSUB ejecutado .

Se puede usar un número ilimitado de subrutinas en un programa y pueden estar anidadas .En otras palabras , las subrutinas pueden llamar a otra subrutina .Cada anidamiento no debe ser mayor de cuatro niveles .

```
GOSUB beep              ´ ejecuta la subrutina beep
beep: high 0            ´ enciende el LED conectado a Pin0
sound 1, [ 80 , 10 ]    ´ hace sonar el parlante conectado a Pin1
low 0                   ´ apaga el LED conectado a Pin0
return                  ´ vuelve a la rutina principal
```

5.19. GOTO

GOTO etiqueta

La ejecución del programa continua en la declaración de la etiqueta .
GOTO send ´ salta a la declaración etiquetada send

send: serout 0,N2400, [” Hi”] ´ envía “ Hi” como salida al Pin0 en forma serial

5.20. HIGH

HIGH Pin

Hace de valor alto el Pin especificado y lo convierte automáticamente en salida .Pin puede ser una constante , 0 - 15 , ó una variable que contenga un número de 0-15 (p.ej. B0) ó un número de Pin (p.ej. PORTA.0)

```
HIGH 0              ´ convierte Pin0 en salida y lo coloca en valor alto (5 volt)
HIGH PORTA.0        ´ convierte PORTA,Pin0 en salida y lo coloca en valor alto (5 volt)
led var PORTB.0     ´ define el pin LED
HIGH led            ´ convierte el Pin LED en salida y lo coloca en valor alto (5 volt)
```

Como alternativa , si el pin ya es salida , hay una forma más rápida y corta de setearlo en valor alto (desde un codigo generado standpoint) :

```
PORTB.0 = 1         ´ setea PORTB Pin0 a valor alto .
```

5.21. HSERIN

HSERIN {ParityLabel , }{Timeout ,Label ,} [Item { . . . }]

Recibe uno ó más Items de un port serial (de hardware) en dispositivos que soportan comunicaciones seriales asincrónicas por hardware .

HSERIN es una de varias funciones seriales asincrónicas pre-construídas .Sólo puede ser usada en dispositivos que posean hardware USART .Vea la hoja de datos del dispositivo para información de los pin seriales de entrada y otros .Los parametros seriales y el baud-rate son especificados usando DEFINE :

´ coloque el registro receptor en receptor habilitado
DEFINE HSER_RCSTA 90h

´ coloque el registro de transmisión en transmisión habilitada
DEFINE HSER_TSTA 20h

´ coloque baud rate
DEFINE HSER_BAUD 2400

HSERIN asume un oscilador de 4 Mhz cuando calcula el baud rate .Para mantener una relación de baud rate apropiada con otros valores de oscilador ,use DEFINE para especificar el nuevo valor OSC .
Timeout y Label pueden ser incluídos en forma opcional para permitir al programa continuar si un carácter no es recibido dentro de un límite de tiempo . Timeout está especificado en unidades de 1 milisegundo .

El formato por defecto de los datos seriales es 8N1 , 8 bits de datos ,sin paridad y 1 stop bit .7E1 (7 bits de datos , paridad par , 1 stop bit) ó 7 O 1 (7 bits de datos , paridad impar ,1 stop bit) pueden ser habilitados usando los siguientes DEFINEs :

´ use solo si se desea paridad par
DEFINE HSER_EVEN 1

´ use solo si se desea paridad impar
DEFINE HSER_ODD 1

El seteo de paridad igual que todos los DEFINE HSER afectan tanto a HSERIN como a HSEROUT
Se puede incluir ParityLabel como opcional en la declaración .El programa continuará en este punto si se recibe un carácter con error de paridad .Solo debe ser usado si se habilitó paridad con un DEFINE anterior .

Dado que la recepción serial se realiza por hardware ,no es posible invertir los niveles para eliminar un driver RS - 232 .Por esto debe usarse un driver adecuado con HSERIN .
HSERIN soporta los mismos modificadores de datos que SERIN2 .Refierase a la sección de SERIN2 para mayor información .

HSERIN [B0 , dec W1]

5.22. HSEROUT

HSEROUT [Item {,Item }]

Envía uno ó más Items al port serial de hardware en dispositivos que soportan comunicaciones seriales asincrónicas por hardware .

HSEROUT es una de varias funciones seriales asincrónicas pre-construídas .Sólo puede ser usada en dispositivos que posean hardware USART .Vea la hoja de datos del dispositivo para información de los pin seriales de entrada y otros .Los parametros seriales y el baud-rate son especificados usando DEFINE :

´ coloque el registro receptor en receptor habilitado
DEFINE HSER_RCSTA 90h

´ coloque el registro de transmisión en transmisión habilitada
DEFINE HSER_TSTA 20h

´ coloque baud rate
DEFINE HSER_BAUD 2400

HSEROUT asume un oscilador de 4 Mhz cuando calcula el baud rate .Para mantener una relación de baud rate apropiada con otros valores de oscilador ,use DEFINE para especificar el nuevo valor OSC .

El formato por defecto de los datos seriales es 8N1 , 8 bits de datos ,sin paridad y 1 stop bit .7E1 (7 bits de datos , paridad par , 1 stop bit) ó 7 O 1 (7 bits de datos , paridad impar ,1 stop bit) pueden ser habilitados usando los siguientes DEFINEs :

´ use solo si se desea paridad par
DEFINE HSER_EVEN 1

´ use solo si se desea paridad impar
DEFINE HSER_ODD 1

El seteo de paridad igual que todos los DEFINE HSER afectan tanto a HSERIN como a HSEROUT
Dado que la recepción serial se realiza por hardware ,no es posible invertir los niveles para eliminar un driver RS - 232 .Por esto debe usarse un driver adecuado con HSEROUT .
HSEROUT soporta los mismos modificadores de datos que SEROUT2 .Refiérase a la sección de SEROUT2 para mayor información .

´ enviar el valor decimal de B0 seguido por un linefeed a través del USART

HSEROUT [dec B0 , 10]

5.23. I2CREAD

I2CREAD DataPin ,ClockPin,Control,{Address,}[Var {,Var ...}] { . Label }

Envía los bytes de Control y opcionalmente los de Address , a través del ClockPin y el DataPin y guarda los bytes recibidos dentro de Var .ClockPin y dataPin pueden ser constantes , 0-15 , una variable que contenga un número (p.ej. B0) , ó un número de Pin (p.ej. PORTA.0)

I2CREAD y I2CWRITE pueden ser usados para leer y grabar datos de un EEPROM serial usando una interfase I2C de 2 cables , como Microchip 24LC01B ó similar .Esto permite guardar datos en una memoria externa no volátil , para que sean mantenidos aún sin energía conectada .Estos comandos funcionan en modo I2C master y también son usados para comunicarse con otros dispositivos con interfase I2C , como sensores de temperatura y convertidores A/D .

Los 7 bits superiores del byte de Control contienen el código de control junto con la selección del chip e información adicional de dirección ., dependiendo de cada dispositivo .El bit inferior es una bandera interna que indica si es un comando de lectura ó escritura y no se debe usar .
Este formato para el byte de Control es diferente al usado por el PBP original .Asegúrese de usar este formato en operaciones PBP I2C .

Por ejemplo , cuando comunicamos con un 24LC01B , el código de control es %1010 y no se usa la selección de chip , por lo que el byte de Control será %10100000 ó $A0 .Algunos formatos de Control son :

Dispositivo	Capacidad	Control	Tamaño dirección
24LC01B	128 bytes	%1010xxx0	1 byte
24LC02B	256 bytes	%1010xxx0	1 byte
24LC04B	512 bytes	%1010xxb0	1 byte
24LC08B	1 Kbytes	%1010xbb0	1 byte
24LC16B	2 Kbytes	%1010bbb0	1 byte
24LC32B	4 Kbytes	%1010ddd0	2 bytes
24LC65	8 Kbytes	%1010ddd0	2 bytes

bbb = bits de selección de block (direcciones de orden alto)
ddd = bits de selección de dispositivo
xxx = no importa

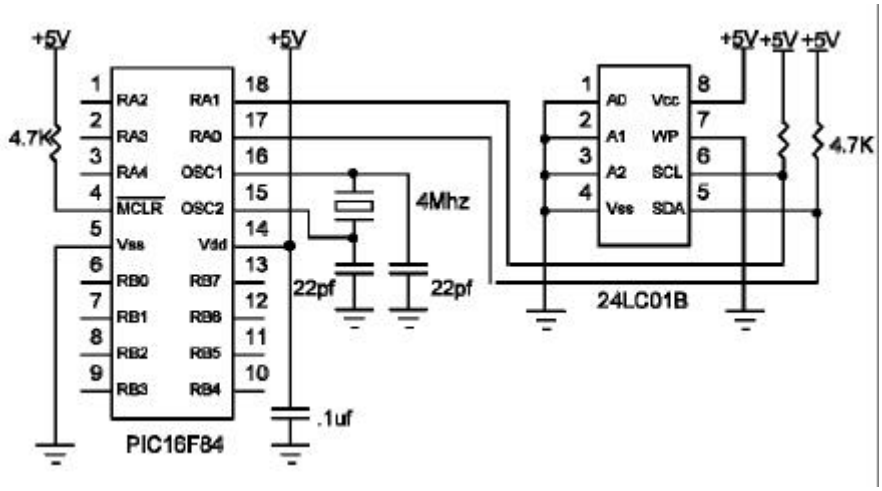
El tamaño de dirección enviado (byte ó word) es determinado por el tamaño de la variable usada .Si se usa una variable con tamaño byte se envía una dirección de 8 bits.Si se envía una variable de tamaño word ,se envía una dirección de 16 bits. Asegúrese de usar una variable apropiada al dispositivo a comunicar .
Si se especifica Var con tamaño word , se leen 2 bytes y se guarda primero el de mayor orden y luego el de orden inferior dentro de Var .Este orden es el inverso al que se osa normalmente con variables .
Si se usa la opción Label , se saltará a ella , si no se recibe un reconocimiento del dispositivo I2C .
Las instrucciones I2C pueden ser usadas para acceder al EEPROM incorporado en los dispositivos 12CExxx y 16CExxx .Simplemente especifique los nombres de pin de las líneas internas adecuadas como parte del comando I2C y coloque el siguiente DEFINE en el principio del programa .

DEFINE I2C_INTERNAL 1

Vea las hojas de datos de Microchip para más información .
El tiempo de las instrucciones I2C es tal que los dispositivos de velocidad standard (100 Khz) pueden ser accedidos a velocidad de clock de hasta 8 Mhz .Dispositivos rápidos (400 Khz) pueden ser usados hasta 20 Mhz .Si se desea acceder un dispositivo de velocidad standard a 8 Mhz , se debe usar el siguiente

DEFINE en el programa :
DEFINE I2C_SLOW 1

El clock I2C y las líneas de datos pueden ser empujados a Vcc con un resistor de 4-7 K de acuerdo al siguiente esquema ,ya que ambos trabajan en modo de colector abierto .



addr var byte
cont con %10100000
addr =17 ´ coloca la dirección en 17

´ lee datos de la dirección 17 y los deja en B2
I2CREAD PORTA.0,PORTA.1,cont,addr, [B2]

Vea el libro Microchip " NON VOLATILE MEMORY PRODUCTS" para mayor información de este ú otros dispositivos que pueden ser usados con los comandos I2CREAD y I2CWRITE .

5.24. I2CWRITE

I2CWRITE DataPin ,ClockPin,Control,{Address,}[Value {,Value ...}] { . Label }

Envía los bytes de Control y opcionalmente los de Address , a través del ClockPin y el DataPin seguidos por Value .ClockPin y DataPin pueden ser constantes , 0-15 , una variable que contenga un número (p.ej. B0) , ó un número de Pin (p.ej. PORTA.0)

El tamaño de dirección enviado (byte ó word) es determinado por el tamaño de la variable usada .Si se usa una variable con tamaño byte se envía una dirección de 8 bits.Si se envía una variable de tamaño word ,se envía una dirección de 16 bits. Asegúrese de usar una variable apropiada al dispositivo a comunicar .

Cuando se escribe un EEPROM serial , es necesario esperar 10 ms (dependiendo del dispositivo) para completar la grabación , antes de intentar comunicarse nuevamente con el dispositivo .Si se intenta un I2CWRITE ó I2CREAD antes que se complete la grabación , se ignorará el acceso .

Aunque una sola declaración I2CWRITE puede ser usada para grabar múltiples bytes simultaneamente , se puede violar los requerimientos de tiempo de grabación para los EEPROM seriales .Algunos permiten grabar múltiples bytes en una página simple antes de necesitar una espera .Revise la hoja de datos del dispositivo que esté usando .La opción de grabación múltiple puede ser útil con dispositivos I2C que no deban esperar entre grabaciones .

Si se usa la opción Label , se saltará a ella , si no se recibe un reconocimiento del dispositivo I2C .

Las instrucciones I2C pueden ser usadas para acceder al EEPROM incorporado en los dispositivos 12CExxx y 16CExxx .Simplemente especifique los nombres de pin de las líneas internas adecuadas como parte del comando I2C y coloque el siguiente DEFINE en el principio del programa .

DEFINE I2C_INTERNAL 1

Vea las hojas de datos de Microchip para más indormación .
El tiempo de las instrucciones I2C es tal que los dispositivos de velocidad standard (100 Khz) pueden ser accedidos a velocidad de clock de hasta 8 Mhz .Dispositivos rápidos (400 Mhz) pueden ser usados hasta 20 Mhz .Si se desea acceder un dispositivo de velocidad standard a 8 Mhz , se debe usar el siguiente

DEFINE en el programa :
DEFINE I2C_SLOW 1

Vea el siguiente comando I2CREAD

```
addr var byte
cont con %10100000
addr =17 ´ coloca la dirección en 17

´ envía el byte 6 a la dirección 17
I2CWRITE PORTA.0,PORTA.1,cont,addr, [ 6 ]

Pause 10          ´ espera 10 ms que se complete la grabación
addr =1           ´ coloca la dirección en 1

´ envía el byte en B2 a la dirección 1
I2CWRITE PORTA.0,PORTA.1,cont,addr, [ B2 ]

Pause 10          ´ espera 10 ms que se complete la grabación
```

5.25 IF ...THEN

```
IF Comp { AND/OR Comp ... } THEN Label
IF Comp { AND/OR Comp ... } THEN
Declaración
ELSE
Declaración
ENDIF
```

Efectúa una ó más comparaciones .Cada término Comp puede relacionar una variable con una constante ú otra variable e incluye uno de los operadores listados anteriormente .

IF ... THEN evalúa la comparación en términos de CIERTO o FALSO .Si lo considera cierto , se ejecuta la operación posterior al THEN . Si lo considera falso , no se ejecuta la operación posterior al THEN .Las comparaciones que dan 0 se consideran falso .Cualquier otro valor es cierto .Todas las comparaciones son sin signo , ya que PBP solo soporta operaciones sin signo .

Asegúrese de usar paréntesis para especificar el orden en que se deben realizar las operaciones .De otra manera , la prioridad de los operadores lo determina y el resultado puede no ser el esperado .

IF..THEN puede operar de dos maneras. De una forma, el THEN en un IF..THEN es esencialmente un GOTO. Si la condición es cierta, el programa irá hacia la etiqueta que sigue al THEN. Si la condición es falsa, el programa va a continuar hacia la próxima línea después del IF..THEN. Otra declaración no puede ser puesta después del THEN; sino que debe ser una etiqueta.

```
‘ si el botón conectado al pin 0 es oprimido (0), salta a la etiqueta pushd
If Pin0 = 0 Then pushd

‘ si el valor en la variable B0 es mayor ó igual a 40 salta a old
If B0 >=40 Then old

‘si PORTB, pin 0 es alto (1), salta a itson
If PORTB.0 Then itsonIf (B0 = 10) AND (B1 = 20) Then loop
```

En la segunda forma, IF..THEN puede ejecutar condicionalmente un grupo de declaraciones que sigan al THEN. Las declaraciones deben estar seguidas por un ELSE o un ENDIF para completar la estructura.

```
If B0 <> 10 Then
B0 = B0 + 1
B1 = B1 - 1
Endif
If B0 = 20 Then
led = 1
Else
led = 0
Endif
```

5.26. INPUT

INPUT Pin

Convierte el Pin especificado en una entrada. Pin debe ser una constante, 0-15, o una variable que contenga un número 0-15 (p. ejj., B0) o el nombre de un pin (p. ej., PORTA.0).
INPUT 0 ‘ convierte el Pin0 en entrada

INPUT PORTA.0 ‘ convierte el PORTA, pin 0 en entrada
En forma alternativa, el pin puede ser colocado como entrada de una forma más rápida y corta (desde un código generado standpoint):

TRISB.0 =1 ‘ Setea el PORTB, pin 0 como entrada

Todos los pins en un port pueden ser colocados como entradas seteando el registro TRIS completo de una sola vez:

TRISB = %11111111 ‘ Setea todo el PORTB como entrada

5.27. { LET }

{LET} Var = Value

Asigna un Value a una Variable. El Value puede ser una constante, otra variable o el resultado de una expresión. Refiérase a la sección previa acerca de operadores para más información. La palabra clave LET , por sí misma , es opcional.

```
LET B0 = B1 * B2 + B3
B0 = Sqr W1
```

5.28. LCDOUT

LCDOUT Item{ , Item...}

Muestra Items en un visor de cristal líquido inteligente (LCD). PBP soporta módulos LCD con un controlador Hitachi 44780 o equivalente. Estos LCD, usualmente, tienen un cabezal de 14 o 16 pins simples o duales en un extremo.

Si el signo (#) está colocado antes de un Item, la representación ASCII para cada dígito es enviada al LCD. LCDOUT también puede usar cualquiera de los modificadores usados con SEROUT2. Vea la sección de SEROUT2 para más información.

Un programa debe esperar, por lo menos, medio segundo antes de enviar el primer comando a un LCD. Puede tomar bastante tiempo a un LCD arrancar.
Los comandos son enviados al LCD, enviando un $FE seguido por el comando. Algunos comandos útiles se muestran en la siguiente tabla:

Comando	Operación
$FE, 1	Limpia visor
$FE, 2	Vuelve a inicio (comienzo de la primera línea)
$FE, $0C	Cursor apagado
$FE, $0E	Subrayado del cursor activo
$FE, $0F	Parpadeo del cursor activo
$FE, $10	Mueve cursor una posición hacia la izquierda
$FE, $14	Mueve cursor una posición hacia la derecha
$FE, $C0	Mueve cursor al comienzo de la segunda línea

Note que hay un comando para mover el cursor al comienzo de la segunda línea en un visor de dos líneas. Para muchos LCD, los caracteres y líneas mostrados no son consecutivos en la memoria del visor - puede haber un salto entre las localizaciones. Para muchos visores 16x2, la primera línea comienza en $0 y la segunda, en $40. El comando:

LCDOUT $FE, $C0

hace que el visor comience a escribir caracteres en el principio de la segunda línea. Los visores 16x1 usualmente están formateados como visores de 8x2, con un salto entre las locaciones de memoria para los primeros y segundos caracteres de 8. Los visores de 4 líneas, también tienen un mapa de memoria no ordenado.

Vea la hoja de datos para el dispositivo LCD, en particular el que usted esté usando, para las locaciones de memoria de caracter y comandos adicionales.

```
LCDOUT $FE, 1, “Hello”          ‘limpia el visor y muestra “Hello”
LCDOUT B0, #B1
```

El LCD puede estar conectado al micro Pic, usando un bus de 4 bit o uno de 8 bit. Si se usa un bus de 8 bit, todos los 8 bits deben estar en un port. Si se usa un bus de 4 bit , debe estar conectado o a los 4 bit inferiores o a los 4 bit superiores de un port. Enable y Register Select deben estar conectados a algún pin del port . R/W debe estar colocado a tierra, ya que el comando de LCDOUT solamente es de grabación.

PBP supone que el LCD está conectado a pins específicos, a menos que se le diga de otra manera. Asume que el LCD va a ser usado con un bus de 4 bits, con las líneas de data DB4 - DB7 conectadas en el micro Pic a PORTA.0 - PORTA.3, Register Select a PORTA.4 y Enable a PORTB.3. Además, inicializa el LCD como un visor de dos líneas.

Para cambiar este seteo, coloque uno o más de los siguientes DEFINEs, todos en mayúsculas, en el comienzo de su programa PBP:

```
‘ Setea el port de datos LCD
DEFINE LCD_DREG PORTB
```

‘ Setea el bit de comienzo de datos (0 o 4) si el bus es de 4-bit
DEFINE LCD_DBIT 0

‘ Setea el port LCD Register Select
DEFINE LCD_RSREG PORTB

‘ Setea el bit LCD Register Select
DEFINE LCD_RSBIT 4

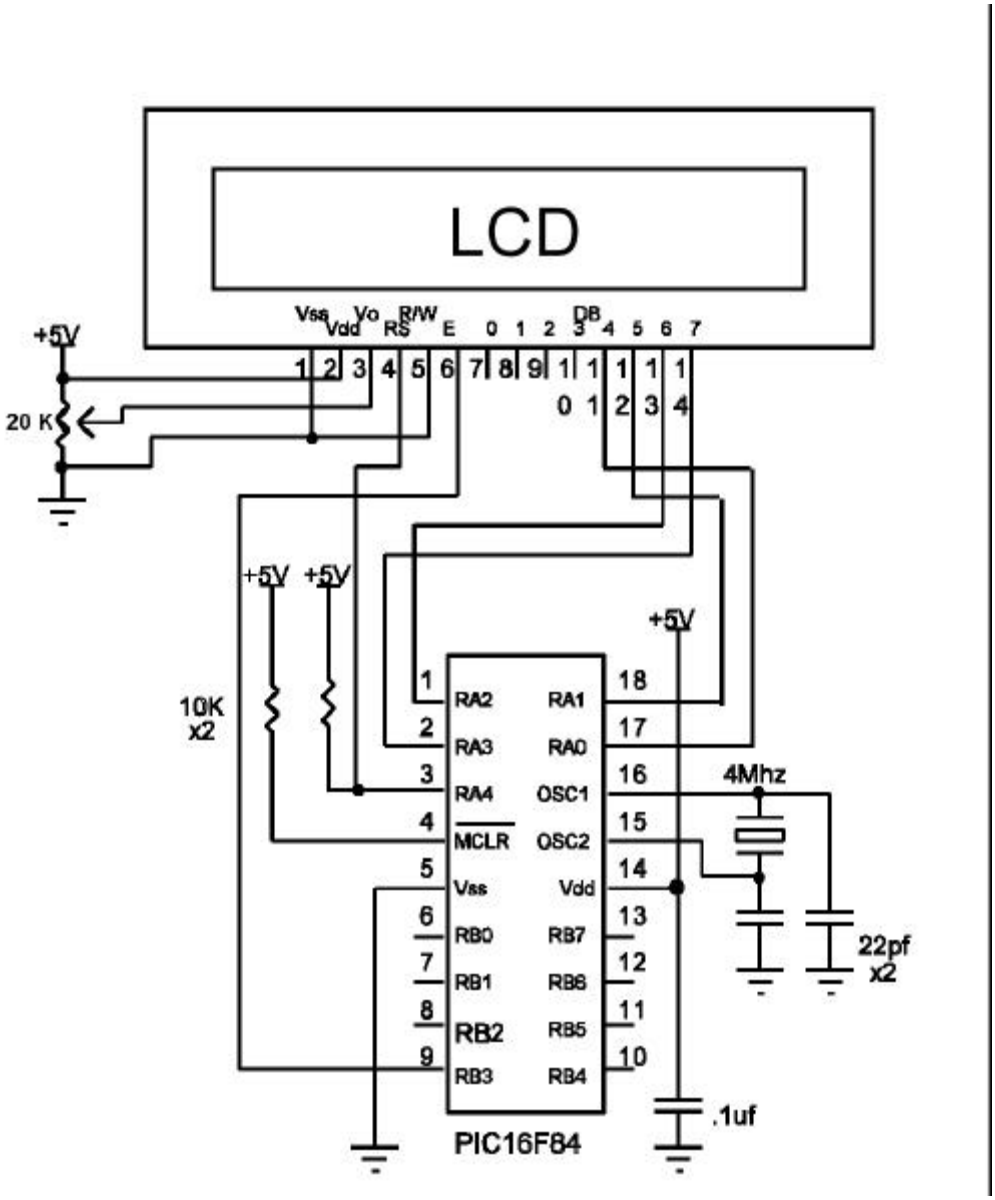
‘ Setea el port LCD Enable
DEFINE LCD_EREG PORTB

‘ Setea el bit LCD Enable
DEFINE LCD_EBIT 5

‘ Setea el tamaño del bus LCD (4 o 8 bits)
DEFINE LCD_BITS 4

‘ Setea el numero de lineas en el LCD
DEFINE LCD_LINES 2

Este seteo, le dirá a PBP que hay conectado un LCD de 2 líneas en modo de 4 bit con el bus de datos en los 4 bit inferiores de PORTB, Register Select en el PORTB.4, y Enable en el PORTB.5.
El siguiente esquema muestra una forma de conectar un LCD a un micro Pic:



5.29. LOOKDOWN

LOOKDOWN Search, [Constant{ , Constant...}] , Var

La declaración LOOKDOWN busca en una lista de 8 bit los valores Constant que coincidan con un valor Search. Si se encuentra, el índice de la constante es guardado en Var así, si el valor es el primero de la lista, Var = 0. Si es el segundo, Var = 1 y así, sucesivamente. Si no se encuentra, no se toma ninguna acción y Var permanece sin cambios.

La lista de constantes puede ser una mezcla de constantes numéricas y sartas. Cada carácter en una sarta es tratado como una constante separada con el valor del carácter ASCII. Las variables de array con índice variable no pueden ser usadas en LOOKDOWN, aunque variables de array con índice constantes son permitidas.

´ Obtiene un carácter hexadecimal de pin1 enforma serial

Serin 1,N2400,B0

´ Convierte el carácter hexadecimal en B0 a un valor decimal B1
LOOKDOWN B0, [“0123456789ABCDEF”], B1

´ Envía un valor decimal a pin0 en forma serial
Serout 0,N2400, [#B1]

5.30. LOOKDOWN2

LOOKDOWN2 Search, {Test} [Value{, Value...}] , Var

La declaración LOOKDOWN2 busca un valor Search en una lista de Values. Si lo encuentra, el índice de la constante es guardado en Var así, si el valor es el primero de la lista, Var = 0. Si es el segundo, Var = 1 y así, sucesivamente. Si no se encuentra, no se toma ninguna acción y Var permanece sin cambios.

El parámetro opcional Test puede ser usado para efectuar una busqueda distinta a la igualdad (“=”).Por ejemplo , se puede buscar el primer Value que sea mayor que el parámetro Search usando (“ > “) , Si no se indica nada , se asume (“=”) .

La lista de Values puede ser una mezcla de constantes numéricas y sartas en 16 bits y variables. Cada carácter en una sarta es tratado como una constante separada con el valor del carácter ASCII. No se pueden usar expresiones en una lisia de Values , aunque pueden ser usadas como valor Search Las variables de array con índice variable no pueden ser usadas en LOOKDOWN2, aunque variables de array con índice constantes son permitidas.

LOOKDOWN2 genera un código 3 veces más grande que LOOKDOWN .Si la lista consiste solamente de constantes y sartas de 8 bits , use LOOKDOWN .

```
LOOKDOWN2 W0,[512,W1,1024],B0
LOOKDOWN2 W0,<[10,100,1000],B0
```

5.31. LOOKUP

LOOKUP Index, [Constant{ , Constant...}] , Var

LOOKUP puede ser usado para obtener valores de una tabla de constantes de 8 bits ,Si Index es cero , Var toma el valor de la primer Constant .Si Index es 1 , Var toma el valor de la segunda Constant y así sucesivamente .Si Index es mayor ó igual que el número de entradas en la lista de constantes , no se toma ninguna acción y Var permanece sin cambios .

La lista de constantes puede ser una mezcla de constantes numéricas y sartas. Cada carácter en una sarta es tratado como una constante separada con el valor del carácter ASCII. Las variables de array con índice variable no pueden ser usadas en LOOKUP, aunque variables de array con índice constantes son permitidas.

```
For B0=0 to 5                ´ cuenta de 0 a 5
LOOKUP B0,[ “Hello “ ],B1    ´ obtiene el carácter B0 de la sarta y lodeja en B1
Serout 0,N2400, [ B1 ]       ´ envía el carácter en B1 al Pin0 en formaSerial
Next B0                      ´ va al segundo carácter
```

5.32. LOOKUP2

LOOKUP2 Index,[Value [,Value ...}] , Var

LOOKUP2 puede ser usado para obtener entradas de una tabla de Values ,Si Index es cero , Var toma el valor del primer Value .Si Index es 1 , Var toma el valor del segundo Value y así sucesivamente .Si Index es mayor ó igual que el número de entradas en la lista , no se toma ninguna acción y Var permanece sin cambios .

La lista de Values puede ser una mezcla de constantes numéricas y sartas en 16 bits y variables. Cada carácter en una sarta es tratado como una constante separada con el valor del carácter ASCII. No se pueden usar expresiones en una lisia de Values , aunque pueden ser usadas como valor Index Las variables de array con índice variable no pueden ser usadas en LOOKUP2, aunque variables de array con índice constantes son permitidas.

LOOKUP2 genera un código 3 veces más grande que LOOKUP .Si la lista consiste solamente de constantes y sartas de 8 bits , use LOOKUP .

```
LOOKUP2 B0 , [256,512,1024],W1
```

5.33. LOW

LOW Pin

Coloca el pin especificado en valor bajo y automáticamente lo convierte en salida .Pin puede ser una xonstante , 0-15 , +o una variable que contenga un número 0-15 (p.ej. B0) ó un nombre de pin (p.ej. PORTA.0)

```
LOW 0              ´ Coloca el Pin9 en salida y nivel bajo ( 9 volt)
LOW PORTA.0        ´ Coloca PORTA.0 como salida y en nivel bajo(0 volt)
Led var PORTB.0    ´ define un pin LED
LOW led            ´ coloca el pin LED como salida y en valor bajo(0 volt)
```

Si el pin ya es una salida , es más rápido y corto usar un código ya generado :

PORTB.0 = 0 ´ coloca en nivel bajo el pin0 de PORTB

5.34. NAP

NAP Period

Coloca al micro controlador en modo de baja potencia por períodos de tiempo reducidos . Durante este NAP , se reduce al mínimo el consumo de energía . Los períodos indicados son solo aproximados , porque el tiempo se deriva del Watchdog Timer que está controlado por R/C y puede variar de chip a chip y también con la temperatura . Como NAP usa el Watchdog Timer es independiente de la frecuencia del oscilador .

Period	Demora (aprox.) en milisegundos
0	18
1	36
2	72
3	144
4	288
5	576
6	1152
7	2304

NAP 7 ´ pausa en baja potencia por aprox. 2,3 segundos

5.35. ON INTERRUPT

ON INTERRUPT GOTO Label

Permite el manejo de las interrupciones del micro controlador por medio de una subrutina PBP .
Existen dos formas de manejar interrupciones usando PBP . La primera es escribir una subrutina de interrupción en lenguaje ensamblador .Esta es la forma de manejar interrupciones con la menor latencia y el menor overhead .Este método se discute más adelante , en la sección avanzada .

El segundo método es escribir un handler (manejador) de interrupciones PBP .Es similar a una subrutina PBP , pero termina con un RESUME .

Cuando ocurre una interrupción , se marca con una bandera .Cuando la ejecución de la declaración PBP que se estaba ejecutando termina , el programa salta al handler de interrupciones indicado en Label .Una vez que termina el trabajo del handler , una declaración RESUME envía el programa de vuelta a donde estaba cuando ocurrió la interrupción , tomando todo como lo dejó .

DISABLE y ENABLE permiten que distintas secciones de un programa PBP se ejecuten sin la posibilidad de ser interrumpidas .El lugar más notorio para usar DISABLE es justo antes del actual handler de interrupciones . O el handler puede ser colocado antes que la declaración ON INTERRUPT ya que la bandera de interrupciones no se chequea antes del primer ON INTERRUPT en un programa .

Latencia es el tiempo entre el pedido de interrupción y el momento en que se ingresa en el handler de interrupciones .Como las declaraciones de PBP no son reentrantes (p.ej. no se puede ejecutar una declaración mientras se está ejecutando una anterior) , puede existir una latencia considerable hasta que se ingrese a la rutina de interrupciones.

PBP no ingresará al handler BASIC de interrupciones hasta que haya terminado de ejecutar la declaración en curso .Si la declaración es PAUSE ö SERIN , puede demorarse bastante hasta que sea reconocida la interrupción .Se debe diseñar el programa tomando en cuenta esta latencia .Si esta es inaceptable y las interrupciones deben ser manejados más rapidamente , se debe usar una rutina en lenguaje ensamblador .

Overhead es otro tema .ON INTERRUPT agregará una instrucción después de cada declaración , para chequear si ocurre ó no una interrupción .DISABLE elimina la posibilidad de usar esta instrucción y ENABLE la habilita .Normalmente , las instrucciones adicionales no son un problema , pero programas largos en pequeñas computadoras si lo son .

Se puede usar más de un ON INTERRUPT en un programa :

```
ON INTERRUPT GOTO myint ´ el handler de interrupciones es myint
INTCON = %10010000 ´ habilita la interrupción RB0
. . . . . .
DISABLE                 ´ deshabilita las interrupciones en el
Handler
Myint : led=1           ´ enciende el LED con una interrupción
RESUME                  ´ vuelve al programa principal
ENABLE                  ´ habilita las interrupciones después del
Handler
```

Para deshabilitar permanentemente las interrupciones (ó hasta que se necesiten) , una vez que se usó ON INTERRUPT , coloque INTCON en $80

```
INTCON = $80
```

5.36. OUTPUT

OUTPUT Pin

Convierte el pin especificado en salida . .Pin puede ser una constante , 0 - 15 , ó una variable que contenga un número de 0-15 (p.ej. B0) ó un número de Pin (p.ej. PORTA.0)

OUTPUT 0 ´ convierte pin 0 en salida
OUTPUT PORTA.0 ´ convierte PORTA pin 0 en salida

En forma alternativa , el pin puede ser convertido en salida de una manera más rápida y corta (con un código generado standpoint).

TRISB.0 = 0 ´ setea PORTB pin 0 como salida

Tofos los pins de un port pueden ser seteados simultaneamente como salida usando el registro TRIS completo :

TRISB = %00000000 ´ setea todos los pins de PORTB como salidas

5.37. PAUSE

PAUSE Period

Detiene el programa por Period milisegundos .Period tiene 16 bit , por lo que los retardos pueden ser de hasta 65.535 milisegundos .(un poco mas de 1 minuto) .No coloca el micro controlador en modo de baja potencia como las otras funciones de retardo (NAP y SLEEP). Inclusive , consume mayor potencia, pero es más exacto .Tiene la misma precisión que el clock .

PAUSE asume la frecuencia de 4 Mhz del oscilador . Si se usa un oscilador de otra frecuencia ,se debe indicar usando el comando DEFINE OSC . Vea la sección sobre velocidad para mayores detalles .

PAUSE 1000 demora de 1 segundo

5.38. PAUSEUS

PAUSEUS Period

Detiene el programa por Period milisegundos .Period tiene 16 bit , por lo que los retardos pueden ser de hasta 65.535 milisegundos .No coloca el micro controlador en modo de baja potencia como las otras funciones de retardo (NAP y SLEEP). Inclusive , consume mayor potencia, pero es más exacto .Tiene la misma precisión que el clock .

PAUSE tiene un número mínimo de ciclos para operar .Como depende de la frecuencia del oscilador , no es posible obtener demoras menores a un número mínimo de microsegundos usando PAUSEUS .Para obtener demoras precisas , menores que esto use una rutina ensambladora tipo ASM...ENDASM .La tabla siguiente muestra el número mínimo de microsegundos obtenible para una determinada frecuencia de oscilador .

OSC	Demora mínima
3(3.58)	20us
4	24us
8	12us
10	8us
12	7us
16	5us
20	3us

PAUSEUS asume la frecuencia de 4 Mhz del oscilador . Si se usa un oscilador de otra frecuencia ,se debe indicar usando el comando DEFINE OSC . Vea la sección sobre velocidad para mayores detalles .

PAUSEUS 1000 demora de 1 segundo

5.39. PEEK

PEEK Address ,Var

Lee el registro del micro controlador en la dirección Address especificada y guarda la lectura en Var

.Opciones especiales del microPIC , como convertidores A/D y ports adicionales pueden ser leídos usando PEEK .

PEEK y POKE permiten acceso directo a los registros del microPIC incluyendo PORTA,PORTB,PORTC,PORTD,PORTE y sus registros asociados de dirección de datos (TRIS) .PEEK y POKE operan en todos los bits de un registro simultaneamente .Cuando se hace un POKE de datos a PORTA , se actualiza el port completo , no solamente un bit individual .

PEEK , PORTA,B0 ´ toma el estado actual de PORTA y lo coloca en B0

PBP puede acceder directamente a registros y bits sin necesidad de utilizar PEEK y POKE .Se recomienda usar el acceso directo y no PEEK y POKE .

B0 = PORTA toma el estado actual de PORTA y lo coloca en B0

5.40. POKE

POKE Address , Value

Graba Value en el registro del micro controlador en la dirección Address especificada .Opciones especiales del microPIC , como convertidores A/D y ports adicionales pueden ser leídos usando PEEK .

POKE $85,0 ´ graba 0 en el registro 85 hexadecimal (setea todo
PORTA como salidas)

PBP puede acceder directamente a registros y bits sin necesidad de utilizar PEEK y POKE .Se recomienda usar el acceso directo y no PEEK y POKE .

TRISA = 0 ´ setea todo PORTA como salidas)
PORTA.0 = 1 ´ setea alto el bit 0 de PORTA

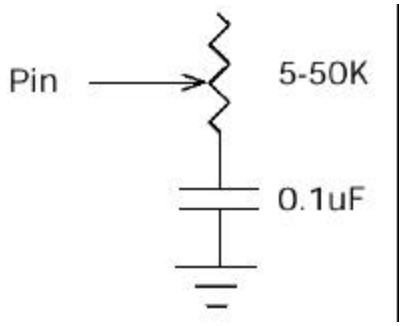
5.41. POT

POT Pin,Scale,Var

Lee un potenciómetro (ú otro dispositivo resistivo) en Pin . Pin puede ser una constante , 0 - 15 , ó una variable que contenga un número de 0-15 (p.ej. B0) ó un número de Pin (p.ej. PORTA.0)

La resistencia se mide tomando el tiempo de descarga de un capacitor a través de un resistor .
(5 K a 50 K) .Scale se usa para ajustar distintas constantes RC . Para constantes RC grandes , Scale debe ser baja (valor mínmo 1) . Para constantes RC pequeñas , Scale debe ser máxima (255) . Si el valor de Scale es correcto , Var debe ser cero para mínima resistencia y 255 para máxima resistencia .

Desafortunadamente , Scale debe ser determinada en forma experimental .Para esto , coloque el dispositivo a medir en máxima resistencia y mídalo con Scale=255 . En estas condiciones , Var tendrá un valor apropiado de Scale .(Este es el mismo tipo de proceso que efectúa la opción ALT-P en BS1).




```
POT,3,255,B0            ´ lee el potenciómetro en pin 3 para determinar Scale
Serout 0,N2400,[#B0]    ´ envía el valor del potenciómetro en forma serial al pin 0
```

5.42. PULSIN

PULSIN Pin,State,Var

Mide el ancho del pulso en Pin . Si State es cero se mide el ancho de un pulso bajo . Si State es uno , se mide el ancho de un pulso alto . El ancho medido se coloca en Var . Si el flanco del pulso no llega , ó el ancho del pulso es demasiado grande para ser medido , Var=0 .

Si se usa una variable de 8 bit , solo se usan los bits menos significativos de la medición de 16 bits . Pin puede ser una constante , 0 - 15 , ó una variable que contenga un número de 0-15 (p.ej. B0) ó un número de Pin (p.ej. PORTA.0)

La resolución de PULSIN depende de la frecuencia del oscilador . Si se usa un oscilador de 4 Mhz , el ancho de pulso se obtiene en incrementos de 10 us .Si se usa un oscilador de 20 Mhz ,el ancho de pulso tendrá una resolución de 2 us .Definir un valor de OSC no tiene efectos sobre PULSIN . La resolución siempre cambia con la velocidad del oscilador en uso .

´ mide el pulso alto en pin 4 y lo guarda en W3
PULSIN PORTB.4,1,W3

5.43. PULSOUT

PULSOUT Pin,Period

Genera un pulso en Pin , con un Period especificado . El pulso se genera activando dos veces el pin , por lo que la polaridad del pulso depende del estado inicial del pin . Pin puede ser una constante , 0 - 15 , ó una variable que contenga un número de 0-15 (p.ej. B0) ó un número de Pin (p.ej. PORTA.0)

La resolución de PULSOUT depende de la frecuencia del oscilador . Si se usa un oscilador de 4 Mhz , el Period del pulso generado estará en incrementos de 10 us .Si se usa un oscilador de 20 Mhz ,Period tendrá una resolución de 2 us .Definir un valor de OSC no tiene efectos sobre PULSOUT . La resolución siempre cambia con la velocidad del oscilador en uso .

´ envia un pulso de 1 mseg. a pin 5 (a 4 Mhz)
PULSOUT PORTB.5,100

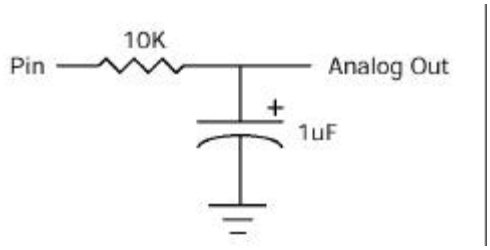
5.44. PWM

PWM Pin,Duty,Cycle

Envía un tren de pulsos modulados en ancho a Pin .Cada ciclo de PWM está compuesto de 256 pasos . El ciclo útil Duty para cada ciclo varía de 0 (0%) a 255 (100%) .El ciclo PWM es repetido Cycle veces . Pin puede ser una constante , 0 - 15 , ó una variable que contenga un número de 0-15 (p.ej. B0) ó un número de Pin (p.ej. PORTA.0)

Cycle depende de la frecuencia del oscilador .Con un oscilador de 4 Mhz , cada Cycle será de aproximadamente 5 mseg. de largo .Con un oscilador de 20 Mhz el largo aproximado será de 1 mseg. Definir un valor de OSC no tiene efecto sobre PWM . El tiempo de Cycle siempre cambia con la velocidad del oscilador en uso .

Pin se convierte en salida justo antes de la generación del pulso y vuelve a ser entrada , cuando cesa .La salida de PWM en un pin tiene mucho ruido , y no tiene forma de onda cuadrada .Es necesario usar algún tipo de filtro para convertirla en algo útil . Un circuito R/C se puede usar como un simple convertidor D/A .



´ envía una señal PWM con un ciclo útil del 50% al pin 7 , durante 100 ciclos
PWM PORTB.7,127,100

5.45. RANDOM

RANDOM Var

Efectúa una iteración pseudo-aleatoria en Var .Var debe ser una variable de 16 bit .No se pueden usar variables de array con índice variable ,pero se permite usar variables de array con índice constante .Var se

usa tanto como origen , como para guardar el resultado .El algoritmo pseudo-aleatorio usado tiene un paso de 65535 (el único número que no produce es el cero).

RANDOM W4 ´ coloca un número aleatorio en W4

5.46. RCTIME

RCTIME Pin,State,Var

Mide el tiempo que un Pin permanece en un estado State determinado .Básicamente es la mitad de un PULSIN . Pin puede ser una constante , 0 - 15 , ó una variable que contenga un número de 0-15 (p.ej. B0) ó un número de Pin (p.ej. PORTA.0)

RCTIME puede usarse para leer un potenciómetro (ó cualquier dispositivo resistivo) .La resistencia puede ser medida descargando un capacitor a través de un resistor (5 K a 50 K) y midiendo el tiempo de carga (ó viceversa)

La resolución de RCTIME depende de la frecuencia del oscilador . Si se usa un oscilador de 4 Mhz , el tiempo estará en incrementos de 10 us .Si se usa un oscilador de 20 Mhz ,el tiempo tendrá una resolución de 2 us .Definir un valor de OSC no tiene efectos sobre RCTIME. La resolución siempre cambia con la velocidad del oscilador en uso .

Si el pin no cambia de estado , se devuelve 0

```
Low PORTB.3                ´ descarga el capacitor para comenzar
Pause 10                   ´ descarga por 10 mseg.
RCTIME PORTB.3,0,W0        ´ lee el potenciómetro en pin 3
```

5.47. READ

READ Address,Var

Lee el EEPROM incorporado en la dirección Address , y guarda el resultado en Var .Esta instrucción solo puede ser usada con un microPIC que tenga un EEPROM incorporado como el PIC16F84 ó PIC16C84

```
READ 5,B2        ´ coloca en B2 el valor de la dirección 5 del EEPROM
```

5.48. RESUME

RESUME { Label }

Vuelve al lugar del programa que se abandonó , después que termina de procesarse una interrupción . RESUME es similar a RETURN ,pero es usado al final de un handler de interrupciones PBP .

Si se usa el Label opcional , la ejecución del programa va a continuar en este Label y no donde estaba el programa cuando ocurrió la interrupción .En este caso , cualquier otra dirección de retorno , no será accesible .Vea ON INTERRUPT para mayor información .

```
clockint: seconds=seconds+1      ´ cuenta tiempo
RESUME                           ´ vuelve al programa después de la
Interrupción
error: high errorled             ´ enciende el led de error
RESUME restart                   ´ vuelve a algun otro lugar
```

5.49. RETURN

RETURN

Vuelve desde una subrutina . Retoma la ejecución en la declaración que sigue al GOSUB que llamó la subrutina .

```
Gosub sub1 ´ va a la subrutina denominada sub1
. . .
sub1: serout 0,N2400,[“Lunch”]      ´ envia “Lunch” al pin 0 en forma serial
RETURN                              ´ vuelve al programa principal despues del gosub
```

5.50. REVERSE

REVERSE Pin

Si Pin es entrada , lo convierte en salida .Si es salida , lo convierte en entrada . Pin puede ser una constante , 0 - 15 , ó una variable que contenga un número de 0-15 (p.ej. B0) ó un número de Pin (p.ej. PORTA.0)

```
Output 4        ´ convierte pin 4 en salida
REVERSE 4       ´ cambia pin 3 a entrada
```

5.51. SERIN

SERIN Pin,Mode, {Timeout,Label,}{[Qual...],} {Item...}

Recibe uno ó más Items en Pin , en formato standard asincrónico , usando 8 bit de datos ,sin paridad y un stop bit (8N1) .SERIN es similar al comando Serin de BS1 con el agregado de Timeout . Pin automaticamente se convierte en entrada . Pin puede ser una constante , 0 - 15 , ó una variable que contenga un número de 0-15 (p.ej. B0) ó un número de Pin (p.ej. PORTA.0)

Los nombres Mode (p.ej. T2400) están definidos en el archivo MODEDEFS.BAS .Para usarlos ,agregue la línea :

Include “modedefs.bas”

al comienzo del programa PBP . BS1DEFS,BAS y BS2DEFS.BAS ya incluyen MODEDEFS.BAS .No lo incluya , si ya está usando alguno de estos archivos .Los números Mode pueden ser usados sin incluir este archivo .

Mode	Mode N°	Baud rate	State
T2400	0	2400	CIERTO
T1200	1	1200	
T9600	2	9600	
T300	3	300	
N2400	4	2400	FALSO
N1200	5	1200	
N9600	6	9600	

N300	7	300	

Timeout y Label son opciones que pueden ser incluídas para permitir al programa continuar si no se recibe un carácter durante un cierto tiempo . Timeout está especificado en unidades de 1 milisegundo .

La lista de Items de datos a ser recibida puede estar precedida por uno ó más calificadores encerrados entre corchetes . SERIN debe recibir estos bytes en un orden exacto , antes de recibir los datos .Si algún byte recibido no concuerda con el byte siguiente de la secuencia de calificación ,el proceso de calificación comienza nuevamente (p.ej. el próximo hyte recibido es comparado con el primer Item de la lista de calificación) .Un Qualifier puede ser constante,variable ó una sarta de constantes . Cada carácter de una sarta es tratado como un calificador individual .

Una vez que se completan los calificadores , SERIN comienza a guardar datos en la variable asociada con cada Item .Si el nombre de variable es único ,el valor del carácter ASCII recibido es guardado en la variable .Si la variable es precedida por el signo # . SERIN convierte un valor decimal en ASCII y guarda el resultado en esa variable .Todos los no-dígitos recibidos antes del primer dígito del valor decimal son ignorados y descartados .El carácter no-dígito que termina el valor decimal también se descarta .

Aunque los chips convertidores de nivel RS-232 son comunes y baratos , las excelentes especificaciones de I/O de los microPIC permiten ejecutar muchas aplicaciones sin usar convertidores de nivel .Más aún , se pueden usar entradas invertidas (N300...N9600) junto con un resistor limitador de corriente .

´espera hasta que carácter “A” sea recibido en forma serial en el pin 1 y coloca el próximo carácter en B0
SERIN 1,N2400, [“A”],B0

5.52. SERIN2

SERIN2 DataPin { FlowPin } ,Mode,{ParityLabel,} {Timeout,Label,}[Item...]

Recibe uno ó más Items en el Pin especificado en formato standard asincrónico .SERIN2 es similar al comando Serin de BS2 .DataPin es colocado como entrada en forma automática .FlowPin es opcional y es automáticamente colocado como salida .DataPin y FlowPin pueden ser una constante , 0 - 15 , ó una variable que contenga un número de 0-15 (p.ej. B0) ó un número de Pin (p.ej. PORTA.0)

El pin opcional de control de flujo FlowPin , puede ser incluído para ayudar a que los datos no desborden la capacidad del receptor .Si se usa , FlowPin es automáticamente habilitado para permitir la transmisión de cada carácter .Este estado habilitado es determinado por la polaridad del dato especificado en Mode .

Mode se usa para especificar el baud rate y los parámetros de operación de la transferencia serial .Los 13 bits de menor orden seleccionan el baud rate . Bit 13 selecciona paridad ó no paridad . Bit 14 selecciona nivel cierto ó invertido . Bit 15 no se usa .
Los bits de baud rate especifican el el tiempo de bit en microsegundos -20 .Para encontrar un valor dado , use la ecuación :

(1000000/baud)-20

Algunos baud rate standard se muestran en la tabla siguiente :

Baud rate	Bits 0 - 12
300	3313
600	1646
1200	813
2400	396
4800	188
9600	84
19200	32

Bit 13 selecciona paridad par (bit13=1) ó sin paridad (bit13=0) .Normalmente ,las transmisiones seriales son 8N1 (8 bit de datos, sin paridad ,1 stop bit) .Si se selecciona paridad ,los datos son recibidos como 7E1 (7 bit de datos , paridad par ,1 stop bit) .

Bit 14 selecciona el nivel de los pins de datos y de control de flujo .Si bit 14=0 , se reciben los datos en forma normal ,para usar con los drivers RS-232 .Si bit 14=1 , los datos se reciben invertidos .Esto se puede usar para evitar usar drivers RS-232

Algunos ejemplos de Mode son :

Mode = 84 (9600 baud ,sin paridad , cierto)
Mode = 16780 (2400 baud , sin paridad , invertido)
Mode = 27889 (300 baud , paridad par invertido)

Si se incluye *ParityLabel* , se saltará a la etiqueta indicada si se recibe un carácter con error de paridad. Solo debe ser usado con paridad par seleccionada (bit 13=1)

En forma opcional se puede incluir *Timeout* y *Label* para permitir que el programa continúe si no se recibe un carácter dentro de un cierto tiempo .Timeout se especifica en unidades de 1 milisegundo .

SERIN2 soporta distintos modificadores , que pueden ser combinados entre sí ,dentro de una declaración SERIN2 para obtener distintos formatos .

Modificador	Operación
BIN {1..16}	Recibe digitos binarios
DEC{1..5}	Recibe digitos decimales
HEX{1..4}	Recibe digitos hexadecimales
SKIP n	Saltea n caracteres recibidos
STR ArrayVar{}	Recibe una sarta de n caracteres ,opcionalmente terminada en el carácter c
WAIT ()	Espera por una secuencia de caracteres
WAITSTRArrayVar{ }	Espera por una sarta de caracteres

1) Una variable precedida por BIN va a recibir la representación ASCII de su valor binario ..Por ejemplo ,si está especificado BIN B0 y se recibe “1000” , B0 será 8.

2) Una variable precedida por DEC va a recibir la representación ASCII de su valor decimal ..Por ejemplo , si está especificado DEC B0 y se recibe “123 ” , B0 será 123.

3) Una variable precedida por HEX va a recibir la representación ASCII de su valor hexadecimal ..Por ejemplo, si está especificado HEX B0 y se recibe “FE ” , B0 será 254.

4) SKIP seguido por un contador , va a saltear esa cantidad de caracteres en el flujo de datos .Por ejemplo , SKIP 4 salteará 4 caracteres .

5) STR seguido por una variable de array , un contador y un carácter opcional de finalización ,va a recibir una sarta de caracteres .La longitud de la sarta está determinada por el contador ó cuando se encuentre el carácter opcional .

6) La lista de items de datos a ser recibidos ,puede estar precedida por uno ó más calificadores entre parentesis después del WAIT . SERIN2 debe recibir estos bytes en un orden exacto ,antes de recibir los datos .Si algún byte recibido no concuerda con el próximo en la secuencia de calificación ,recomienza el proceso de calificación .(p.ej. el el próximo byte recibido se compara con el primer item en la lista de calificadores). Un *Qualifier* puede ser constante , variable ó una sarta de constantes ,Cada carácter de una sarta es tratado como un calificador individual .

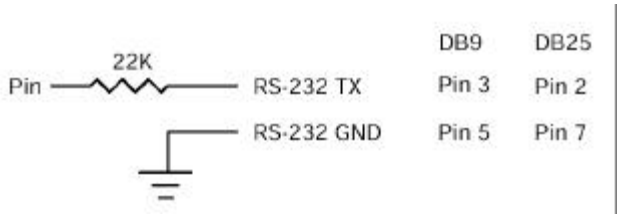
7) WAITSTR puede ser usado como WAIT anteriormente , para forzar a SERIN2 a esperar por una sarta de caracteres de un determinado largo ,antes de seguir adelante .

Una vez que los calificadores WAIT y WAITSTR están cumplimentados , SERIN” comienza a guardar los datos en las variables asociadas con cada Item .Si se usa solo el nombre de la variable , se guarda el valor del carácter ASCII . Si la variable está precedida por BIN,DEC ö HEX , SERIN2 convierte un valor binario , decimal ó hexadecimal en su equivalente ASCII y guarda el resultado en esa variable .Todos los no-digitos recibidos antes que el primer digito del valor decimal es ignorado y descartado .El carácter no-digito que termina el valor también es descartado .

BIN , DEC y HEX pueden estar seguidos por un número .reciben tantos digitos como hay en la entrada .Sin embargo , si un número sigue a un modificador ,SERIN2 siempre recibirá ese número de dígitos , salteando dígitos adicionales si es necesario .

SERIN2 asume un valor de oscilador de 4 Mhz cuando genera sus tiempos de bit .Para mantener los valores de baud rate adecuados con otro oscilador , asegúrese de usar DEFINE OSC con el nuevo valor de oscilador .

Aunque los chips convertidores de nivel RS-232 son comunes y baratos gracias a la implementación de corriente RS-232 y las excelentes especificaciones de I/O del microPIC , no se requieren convertidores de nivel en muchas aplicaciones . Se puede usar TTL invertido (Mode bit14 = 1) .Se sugiere el uso de un resistor limitador de corriente (se supone que RS-232 es tolerante a los cortocircuitos).



´ espera hasta que el carácter “A” sea recibido en forma serial en Pin1 y pone el próximo carácter en B0
SERIN2 1,16780,[wait (”A”),B0]

´ saltea 2 caracteres y toma un número decimal de 4 dígitos
SERIN2 PORTA.1 ,84,[skip 2,dec4 B0]
SERIN2 PORTA.1.0 ,84,100,tlabel,[wait (“x”,b0),str ar]

5.53. SEROUT

SEROUT Pin,Mode,[Item[,Item...}]

Envía uno ó más Items a Pin , en formato standard asincrónico usando 8 bits de datos , sin paridad y 1 stop bit (8N1) .SEROUT es similar al comando Serout de BS1 .Pin es automáticamente colocado como salida . Pin puede ser una constante , 0 - 15 , ó una variable que contenga un número de 0-15 (p.ej. B0) ó un número de Pin (p.ej. PORTA.0)

Los nombres Mode (p.ej. T2400) están definidos en el archivo MODEDEFS.BAS .
Para usarlos ,agregue la línea :

Include “modedefs.bas”

al comienzo de su programa PBP .

BS1DEFS.BAS y BS2DEFS.BAS ya incluyen MODEDEFS.BAS . No lo incluya ,si ya está usando uno de ellos .Los números Mode pueden ser usados sin incluir este archivo .

Mode	Mode N°	Baud rate	Estado
T2400	0	2400	LLEVADO A CIERTO
T1200	1	1200	
T9600	2	9600	
T300	3	300	
N2400	4	2400	LLEVADO A INVERTIDO
N1200	5	1200	
N9600	6	9600	
N300	7	300	
OT2400	8	2400	ABIERTO CIERTO
OT1200	9	1200	
OT9600	10	9600	
OT300	11	300	
ON2400	12	2400	ABIERTO INVERTIDO
ON1200	13	1200	
ON9600	14	9600	
ON300	15	300	

SEROUT soporta 3 tipos distintos de datos , que pueden ser combinados libremente dentro de una declaración SEROUT .

1) Una sarta de constantes es enviada como una sarta de caracteres literales .

2) Un valor numérico (constante ó variable) va a enviar el correspondiente carácter ASCII .Más aún , 13 es retorno de carro (Carriage Return ó CR) y 10 es avance de línea (Line Feed ó LF) .

3) Un valor numérico precedido por el signo # va a enviar la representación ASCII de su valor decimal .Por ejemplo , si W0=123 ,entonces #W0 (ó #123) va a enviar “1”,”2”,”3” .

SEROUT asume un valor de oscilador de 4 Mhz cuando genera sus tiempos de bit .Para mantener los valores de baud rate adecuados con otro oscilador , asegúrese de usar DEFINE OSC con el nuevo valor de oscilador .

En algunos casos ,los rangos de transmisión de SEROUT pueden presentar los caracteres demasiado rápidamente en el dispositivi receptor .Un DEFINE agrega tiempo entre caracteres en la transmisión de salida .Esto permite un tiempo adicional entre caracteres a medida que son transmitidos .Se puede lograr una demora entre cada carácter transmitido de 1 a 65535 microsegundos (.001 a 65,535 milisegundos) . Por ejemplo , para pausar 1 milisegundo entre cada carácter transmitido :

DEFINE CHAR_PACING 1000

Aunque los chips convertidores de nivel RS-232 son comunes y baratos gracias a la implementación de corriente RS-232 y las excelentes especificaciones de I/O del microPIC , no se requieren convertidores de nivel en muchas aplicaciones . Se puede usar TTL invertido (N300 ...N9600) .Se sugiere el uso de un resistor limitador de corriente (se supone que RS-232 es tolerante a los cortocircuitos).
SEROUT 0,N2400,[#B0,10] ´ envía el valor ASCII de B0 ,seguido por
un LF al pin 0 , en forma serial

5.54. SEROUT2

SEROUT2 DataPin { FlowPin } ,Mode,{Pace,} {Timeout,Label,}[Item...]
Envía uno ó más Items al Pin especificado en formato standard asincrónico .SEROUT2 es similar al comando Serout de BS2 .DataPin es colocado como salida en forma automática .FlowPin es opcional y es automáticamente colocado como entrada .DataPin y FlowPin pueden ser una constante , 0 - 15 , ó una variable que contenga un número de 0-15 (p.ej. B0) ó un número de Pin (p.ej. PORTA.0)

El pin opcional de control de flujo FlowPin , puede ser incluído para ayudar a que los datos no desborden la capacidad del receptor .Si se usa , los datos seriales no serán enviados hasta que FlowPin esté en el estado adecuado . .Este estado es determinado por la polaridad del dato especificado en Mode .
Como opción se puede incluir Timeout y Label para permitir continuar al programa si el FlowPin no cambia al estado de habilitación dentro de un cierto tiempo .Timeout esta especificado en unidades de 1 milisegundo .

En algunos casos ,los rangos de transmisión de SEROUT2 pueden presentar los datos demasiado rápidamente al dispositivo receptor .Puede no desearse usar un pin extra para control de flujo .La opción Pace se puede usar para agregar tiempo entre cada carácter durante la transmisión .La demora puede ser de 1 a 65535 milisegundos entre cada carácter transmitido .

Mode se usa para especificar el baud rate y los parámetros de operación de la transferencia serial .Los 13 bits de menor orden seleccionan el baud rate . Bit 13 selecciona paridad ó no paridad . Bit 14 selecciona nivel cierto ó invertido . Bit 15 selecciona si está abierto ó no.

Los bits de baud rate especifican el el tiempo de bit en microsegundos -20 .Para encontrar un valor dado , use la ecuación :

(1000000/baud)-20

Algunos baud rate standard se muestran en la tabla siguiente :

Baud rate	Bits 0 - 12
300	3313
600	1646
1200	813
2400	396
4800	188
9600	84
19200	32

Bit 13 selecciona paridad par (bit13=1) ó sin paridad (bit13=0) .Normalmente ,las transmisiones seriales son 8N1 (8 bit de datos, sin paridad ,1 stop bit) .Si se selecciona paridad ,los datos son enviados como 7E1 (7 bit de datos , paridad par ,1 stop bit) .

Bit 14 selecciona el nivel de los pins de datos y de control de flujo .Si bit 14=0 , se envian los datos en forma normal ,para usar con los drivers RS-232 .Si bit 14=1 , los datos se envian invertidos .Esto se puede usar para evitar usar drivers RS-232

Bit 15 selecciona si el pin de datos está siempre con carga (bit15=0) ó si queda abierto en uno de los estados (bit15=1).El modo abierto puede usarse para conectar varios dispositivos juntos en el mismo bus serial .

Algunos ejemplos de Mode son :

Mode = 84 (9600 baud ,sin paridad , cierto , con carga)
Mode = 16780 (2400 baud , sin paridad , invertido ,con carga)
Mode = 60657 (300 baud , paridad par, invertido, abierto)

SEROUT2 soporta distintos modificadores , que pueden ser combinados entre sí ,dentro de una declaración SEROUT2 para obtener distintos formatos .

Modificador	Operación
{I}{S}BIN {1..16}	Envia digitos binarios
{I}{S}DEC{1..5}	Envia digitos decimales
{I}{S}HEX{1..4}	Envia digitos hexadecimales
REP c n	Envia el caracter c repetido n veces
STR ArrayVar{}	Envia una sarta de n caracteres

1) Una sarta de constantes es enviada como una sarta de caracteres literales .

2) Un valor numérico (constante ó variable) va a enviar el correspondiente carácter ASCII .Más aún , 13 es retorno de carro (Carriage Return ó CR) y 10 es avance de línea (Line Feed ó LF) .

3) Un valor numérico precedido por BIN va a enviar la representación ASCII de su valor binario ..Por ejemplo ,si B0=8 , entonces BIN B0 va a enviar “1000”

4) Un valor numérico precedido por DEC va a enviar la representación ASCII de su valor decimal ..Por ejemplo ,si B0=123 , entonces DEC B0 va a enviar ”123 ”

5) Un valor numérico precedido por HEX va a enviar la representación ASCII de su valor hexadecimal ..Por ejemplo ,si B0=254 , entonces HEX B0 envia “FE ”.

6) REP seguido por un carácter y un contador , va a repetir el carácter la cantidad de veces que indique el contador .Por ejemplo , REP “0”´ 4 enviará “0000”

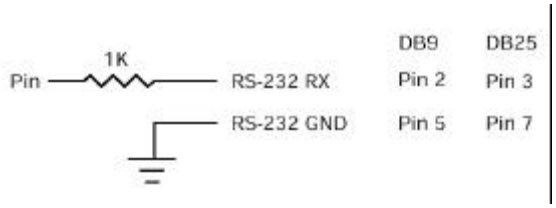
7) STR seguido por una variable de array , y un contador opcional ,va a enviar una sarta de caracteres .La longitud de la sarta está determinada por el contador ó cuando se encuentre un carácter 0 en la sarta .

BIN ,DEC y HEX pueden estar precedidos , ó seguidos por varios parámetros opcionales . Si alguno de ellos está precedido por una I (por indicado).la salida estará precedida por alguno de los símbolos % , # , $ para indicar que el valor siguiente es binario , decimal ó hexadecimal .

Si alguno está precedido por una s (por signo) ,la salida estará precedida por “ - “ , si el bit de alto orden del dato está alto .Esto permite la transmisión de números negativos .Recuerde que todas las operaciones u comparaciones de PBP son sin signo .Sin embargo ,las matemáticas sin signo pueden llevar a resultados con signo .Por ejemplo , B0 = 9 - 10 .El resultado de DEC B0 será “255” .Enviando sDEC B0 daría “-1” , dado que se envía el bit de alto orden .

BIN , DEC y HEX también pueden estar acompañados de un número .Normalmente , estos modificadores muestran exactamente tantos dígitos como sean necesarios (sin enviar los ceros a la izquierda).Sin embargo , si un npumero sigue a un modificador , SEROUT2 siempre ese número de dígitos , agregando tantos ceros al comienzo ,como sea necesario .Además , ajusta cualquier bit extra de orden superior.
P.ej. BIN6 8 sería enviado como “001000” y BIN2 8 como “00” .

Se puede usar cualquier combinación de modificadores simultáneamente .P.ej. ISDEC4 B0 .
Aunque los chips convertidores de nivel RS-232 son comunes y baratos gracias a la implementación de corriente RS-232 y las excelentes especificaciones de I/O del microPIC , no se requieren convertidores de nivel en muchas aplicaciones . Se puede usar TTL invertido (N300 ...N9600) .Se sugiere el uso de un resistor limitador de corriente (se supone que RS-232 es tolerante a los cortocircuitos).



´ envía el valor ASCII de B0 ,seguido por un LF al pin 0 , en forma serial a 2400 baud
SEROUT2 0,16780,[dec B0,10]

´ envía “B0 = 0” seguido por el valor binario de B0 ,a PORTA pin1 ,en forma serial , a 9600 baud
SEROUT2 PORTA.1,84 ,[“B0=0”,ihex4 B0]

5.55. SHIFTIN

SHIFTIN DataPin,ClockPin,Mode, [Var{}...]

El ClockPin , desplaza en forma sincrónica los bits en DataPin y guarda los bytes recibidos en Var . ClockPin y DataPin pueden ser una constante , 0 - 15 , ó una variable que contenga un número de 0-15 (p.ej. B0) ó un número de Pin (p.ej. PORTA.0)

(en forma opcional) especifica el número de bits a ser desplazado .Si no se especifica , se desplazan 8 bits , independientemente del tipo de variable .

Los nombres Mode (p.ej. MSBPRE) están definidos en el archivo MODEDEFS.BAS .
Para usarlos ,agregue la línea :

Include “modedefs.bas”

al comienzo de su programa PBP .

BS1DEFS.BAS y BS2DEFS.BAS ya incluyen MODEDEFS.BAS . No lo incluya ,si ya está usando uno de ellos .Los números Mode pueden ser usados sin incluir este archivo .

Mode	Mode N°	Operación
MSBPRE	0	Primero desplaza datos en el bit superior ,lee datos antes de mandar clock
LSBPRE	1	Primero desplaza datos en el bit inferior , lee datos antes de mandar clock
MSBPOST	2	Primero desplaza datos en el bit superior ,lee datos después de mandar clock
LSBPOST	3	Primero desplaza datos en el bit inferior , lee datos después de mandar clock

SHIFTIN 0,1,MSBPRE,[B0]

5.56. SHIFTOUT

SHIFTOUT DataPin,ClockPin,Mode, [Var{}...]

Desplaza en forma sincrónica el contenido de Var sobre DataPin y ClockPin . ClockPin y DataPin pueden ser una constante , 0 - 15 , ó una variable que contenga un número de 0-15 (p.ej. B0) ó un número de Pin (p.ej. PORTA.0)

(en forma opcional) especifica el número de bits a ser desplazado .Si no se especifica , se desplazan 8 bits , independientemente del tipo de variable .

Los nombres Mode (p.ej. LSBFIRST) están definidos en el archivo MODEDEFS.BAS .
Para usarlos ,agregue la línea :

Include “modedefs.bas”

al comienzo de su programa PBP .

BS1DEFS.BAS y BS2DEFS.BAS ya incluyen MODEDEFS.BAS . No lo incluya ,si ya está usando uno de ellos .Los números Mode pueden ser usados sin incluir este archivo .

Mode	Mode N°	Operación
LSBFIRST	0	Primero desplaza datos del bit inferior
MSBFIRST	1	Primero desplaza datos del bit superior

SHIFTOUT 0,1,MSBFIRST,[B0]
SHIFTOUT PORTA.1,PORTA.2,1,[wordvar 4]

5.57. SLEEP

SLEEP Period

Coloca al micro controlador en modo de baja potencia por Period segundos .Period tiene 16 bit ,por lo que los retardos pueden ser de hasta 65535 segundos (aprox. 18 horas).

SLEEP usa el WatchDog Timer ,por lo que es independiente de la frecuencia del oscilador utilizado .La granulación es aproximadamente 2.3 segundos y puede variar de acuerdo al dispositivo y la temperatura . Esta variación es distinta a la de BASIC Stamp . Se necesitó este cambio ,porque cuando el micro PIC pone a cero (resetea) el WatchDog Timer , también pone valores predefinidos en los registros internos .Estos valores pueden diferir de los esperados por su programa .Ejecutando el comando SLEEP sin calibrar , este paso se deja de lado .

SLEEP 60 ´ duerme por aprox. 1 minuto

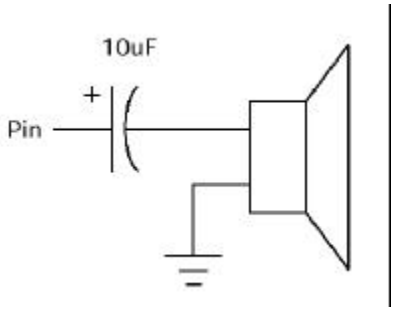
5.58. SOUND

SOUND Pin,[Note,Duration{,Note,Duration...}]

Gebera un tono y/o ruido blanco en el Pin especificado . Pin es automáticamente colocado como salida . Pin puede ser una constante , 0 - 15 , ó una variable que contenga un número de 0-15 (p.ej. B0) ó un número de Pin (p.ej. PORTA.0)

Note 0 es silencio .Nte 1-127 son tonos .Notes 128-255 son ruido blanco .Los tonos y el ruido blanco están en una escala ascendente (p.ej. 1 y 128 son las frecuencias menores , 129 y 266 las mayores).Note 1 es aprox. 78,74 Hz y Note 127 es aproz. 10000 Hz .
Duration es 0-255 y determina el largo de la nota ,en incrementos de 12 milisegundos .Note y Duration no necesitan ser constantes .

SOUND entrega como salida ondas cuadradas con nivel TTL .Gracias a las características del micro PIC , se puede manejar un parlante a través de un capacitor -El valor del capacitor debe ser determinado enfunción de las frecuencias a usar y la carga del parlante .Parlantes piezo eléctricos pueden ser conectados directamente .



SOUND PORTB.7,1[100,10,50,10] ´envía 2 sonidos consecutivos a pin7

5.59. STOP

STOP

Detiene la ejecución del programa , ejecutando un loop sin fin ,No coloca al micro controlador en modo de baja potencia .El micro controlador trabaja igual que siempre .

STOP ´ envía el programa a vía muerta

5.60. SWAP

SWAP Variable,Variable

Intercambia los valores de dos variables .Normalmente intercambiar los valores de dos variables es un proceso tedioso .SWAP lo hace con una sola declaración ,sin variables intermedias .Puede ser usado con variables de bit,vyte y word .Variables de array con índice variable no son permitidas ,pero sí variables de array con índice constante .

```
Temp = B0       ´ anteriormente
B0=B1
B1=Temo

SWAP B0 ,B1     ´ahora
```

5.61. TOGGLE

TOGGLE Pin

Invierte el estado del Pin especificado . Pin es automáticamente colocado como salida . Pin puede ser una constante , 0 - 15 , ó una variable que contenga un número de 0-15 (p.ej. B0) ó un número de Pin (p.ej. PORTA.0)

```
Low 0           ´ comienza Pin0 como bajo
TOGGLE 0        ´ cambia a alto el estado del pin 0
```

5.62. WHILE...WEND

```
WHILE Condition
Statement
WEND
```

Ejecuta las declaraciones Statement en forma repetida ,mientras la condición Condition sea cierta .Cuando Condition deja de ser cierta ,la ejecución continua con la declaración siguiente al WEND .Condition puede ser cualquier expresión de comparación .

```
I=1
WHILE i < = 10
Serout 0,N2400,[“No:”,#i,13,10]
WEND
```

5.63. WRITE

WRITE Address,Value

Graba valores Value en el EEPROM incorporado en la dirección Address especificada . .Esta instrucción solo puede ser usada con un microPIC que tenga un EEPROM incorporado como el PIC16F84 ó PIC16C84
Es usado para colocar datos en el EEPROM durante el momento de la ejecución .Para grabar datos en el EEPROM durante la programación ,se usan las declaraciones DATA y EEPROM .
Cada WRITE se auto regula en tiempo y toma aproximadamente 10 milisegundos ejecutarlo en un microPIC .

WRIT 5,B0 ´ envía el valor de B0 al EEPROM pin 5

5.64. XIN

XIN DataPin,ZeroPin,{Timeout,Label,} [Var{,...}]

Recibe datos X-10 y guarda el House Code y el Key Code en Var.

XIN se usa para recibir información de dispositivos X-10 .Los módulos X-10 están disponibles en muchos lugares y de distintos proveedores .Se requiere una interfase para conectar el micro controlador a la línea de AC .Se necesita un TW-523 para comunicaciones de dos vías para trabajar con XIN .Este dispositivo tiene la interfase a la línea de alimentación y aísla el micro controlador de la línea de AC .Como X-10 está patentado , esta interfase también cubre el licenciamiento .

DataPin es automáticamente convertido en entrada para recibir datos de la interfase X-10 .ZeroPin es automáticamente convertido en entrada para recibir el tiempo de cruce por cero de la interfase X-10 .Ambos pins pueden ser llevados a 5 volt con resistores de 4.7 K . DataPin y ZeroPin pueden ser una constante , 0 - 15 , ó una variable que contenga un número de 0-15 (p.ej. B0) ó un número de Pin (p.ej. PORTA.0)

En forma opcional se pueden incluir Timeout y Label para permitir continuar el programa ,si no se reciben datos en un lapso de tiempo determinado ..Timeout está especificado en medios ciclos de la línea de AC (aprox. 8.33 milisegundos).

XIN solamente procesa datos en el momento en que la línea de AC pasa por cero (en ese momento recibe ZeroPin).Si no hay transiciones en esta línea , XIN esperará que las haya .
Si Var tiene tamaño de word ,cada códigi House Code recibido , se guarda en el byte superior del word .Cada código Key Code recibido se guarda en el byte inferior del word .Si Var es un byte , solo se guarda el Key Code .

El House Code es un número 0-15 , que corresponde al juego House Code del módulo X-10 de A a P .
El Key Code puede ser el número de un módulo específico de X-10 ó la función que debe ser realizada por un módulo .En la práctica , primero se envía un comando especificando el número de módulo X-10 ,seguido por un comando especificando la función deseada .Algunas funciones operan en todos los módulos , por lo que el número de módulo es innecesario .Los ejemplos posteriores ayudarán a clarificar el tema .Key Code 0-15 corresponden a números de módulo 1-16 .
XOUT después lista las funciones y la información de conexionado .

```
Housekey var word

loop: XIN PORTA.2,PORTA.0,[housekey]                          ´ obtiene datos X-10

Lcdout $fe,1,"House=",#housekey.byte1,"Key=",#housekey.byte0  ´ muestra datos X-10 en un LCD

Goto loop                                                     ´ por siempre
```

XIN PORTA.2,PORTA.0,1,nodata,[housekey] ´ busca datos X-10 , si no hay va a nodata

5.65. XOUT

XOUT DataPin,ZeroPin,[HouseCode KeyCode{} {,...}]

Envia un HouseCode seguido por un KeyCode , repetidos un número Repeat de veces en formato X-10 .Si no se usa Repeat se asume 2 veces como mínimo .Repeat usualmente se usa con los comandos Bright y Dim .

XOUT se usa para enviar información de control a dispositivos X-10 .Los módulos X-10 están disponibles en muchos lugares y de distintos proveedores .Se requiere una interfase para conectar el micro controlador a la línea de AC .Se necesita un TW-523 para comunicaciones de dos vías , ó un PL-513 para enviar solamente , para trabajar con XIN .Estos dispositivos tienen la interfase a la línea de alimentación y aíslan el micro controlador de la línea de AC .Como X-10 está patentado , esta interfase también cubre el licenciamiento .

DataPin es automáticamente convertido en salida para enviar datos a la interfase X-10 .ZeroPin es automáticamente convertido en entrada para recibir el tiempo de cruce por cero de la interfase X-10 .Ambos pins pueden ser llevados a 5 volt con resistores de 4.7 K . DataPin y ZeroPin pueden ser una constante , 0 - 15 , ó una variable que contenga un número de 0-15 (p.ej. B0) ó un número de Pin (p.ej. PORTA.0)

XOUT solamente procesa datos en el momento en que la línea de AC pasa por cero (en ese momento recibe ZeroPin).Si no hay transiciones en esta línea , XIN esperará que las haya .
El Hpuse Code es un número 0-15 , que corresponde al juego House Code del módulo X-10 de A a P .El HouseCode apropiado debe ser enviado como parte de cada comando .

El Key Code puede ser el número de un módulo específico de X-10 ó la función que debe ser realizada por un módulo .En la práctica , primero se envía un comando especificando el número de módulo X-10 ,seguido por un comando especificando la función deseada .Algunas funciones operan en todos los módulos , por lo que el número de módulo es innecesario .Los ejemplos posteriores ayudarán a clarificar el tema .Key Code 0-15 corresponden a números de módulo 1-16 .

Los nombres Keycode (funciones) (p.ej. uniton) están definidos en el archivo MODEDEFS.BAS .Para usarlos ,agregue la línea :

Include “modedefs.bas”

al comienzo de su programa PBP .BS1DEFS.BAS y BS2DEFS.BAS ya incluyen MODEDEFS.BAS . No lo incluya ,si ya está usando uno de ellos .Los números KeyCode pueden ser usados sin incluir este archivo .

KeyCode	KeyCode N°	Operación
unitOn	%10010	Enciende el módulo
UnitOff	%11010	Apaga el módulo
UnitsOff	%11100	Apaga todos los módulos
LightsOn	%10100	Enciende módulos de luz
LightsOff	%10000	Apaga módulos de luz
Bright	%10110	Más brillo al nódulo de luz

Dim	%11110	Menos brillo al módulo de luz

Conectarse a la interfase X-10 requiere 4 conexiones .La salida de la interfase X-10 (cruce por cero y datos) es a colector abierto y necesita un resistor de aprox. 4.7 K conectado a 5 volt .La tabla siguiente muestra el xonexionado :

Cable N°	Color del cable	Conexión
1	Negro	Salida cruce por cero
2	Rojo	Comun cruce por cero
3	Verde	Común transmisión X-10
4	Amarillo	Entrada transmisión X-10

TW-523

Cable N°	Color del cable	Conexión
1	Negro	Salida cruce por cero
2	Rojo	Comun cruce por cero
3	Verde	Salida recepción X-10
4	Amarillo	Entrada transmisión X-10

```
house var byte
unit var byte

Inckude “modedefs,bas”

house=0          ´ coloca 0 en house (A)
unit=8           ´ coloca 8 en unit (9)

´ enciende unit 8 en house 0
        XOUT PORTA.1,PORTA.0,[house unit,house unitOn]

´ apaga todas las luces en house 0
        XOUT PORTA.1,PORTA.0 ,[house lightsOff]

´ parpadea la luz 0 cada 10 segundos
        XOUT PORTA.1,PORTA.0 ,[house0]

        loop XOUT PORTA.1 PORTA.0 ,[house]
        pause 10000                                  ´ espera 10 segundos

        XOUT PORTA.1 PORTA.0 ,[house]
        pause 10000                                  ´ espera 10 segundos

        Goto loop
```

6. ESTRUCTURA DE UN PROGRAMA COMPILADO

PBP está diseñado para ser usado fácilmente .Se puede compilar y ejecutar programas gracias al trabajo interno de PBP .Algunos solo tienen confianza en un producto cuando saben como trabaja internamente , otros son solo curiosos .

Esta sección es para ellos . Describe los archivos usados y generados por PBP y da una idea de lo que esta sucediendo .

6.1. APUNTAR A ENCABEZADOS DESTINO ESPECIFICOS

Se usan tres archivos de encabezados destino especificos cuando se compila un programa .Uno es usado por PBP y los otros dos están incluídos para uso del ensamblador .

Un archivo con el nombre del micro controlador ,seguido por la extensión .BAS contiene la información específica del chip necesaria para PBP .Incluye el perfil de memoria del chip , que librerías usa y la definición de las variables que necesita .Para el PIC16F84 ,el micro controlador por defecto , se llama 16F84.BAS .

Un archivo con el nombre del micro controlador ,seguido por la extensión .INC es incluído en el archivo generado .ASM para dar al ensamblador información del chip ,incluyendo los parámetros de configuración . .Para el PIC16F84 ,el micro controlador por defecto , se llama 16F84.INC .

Finalmente ,el ensamblador tiene su archivo include propio , que define las direcciones de los registros del micro controlador .Usualmente se lo llama P16F84.INC .

6.2. ARCHIVOS DE LIBRERÍA

Pbp tiene un juego de archivos de librería que contienen todo el código y los archivos de definición para un grupo particular de micro controladores .En el caso de los microPIC con núcleo de 14 bit , estos archivos comienzan con el nombre PBPPIC14 .

PBPIC14.LIB contiene todas las subrutinas en lenguaje ensamblador usadas por el compilador .
PBPPIC14.MAC contiene todas las macros que llaman a estas subrutinas .Muchos comandos de PBP consisten en una macro y una librería de subrutina asociadas .

PBPPIC14.RAM contiene las declaraciones VAR que direccionan la memoria necesitada por la librería .
PIC14EXT.BAS contiene las definiciones externas que le dicen a PBP todos los nombres de los registros de un microPIC con núcleo de 14 bits .

6.3. CODIGO GENERADO PBP

Un programa PBP compilado se construye en varias etapas .Primero PBP crea el archivo .ASM .Luego construye un archivo .MAC que contiene solo las macros (tomadas de la librería) usadas en el archivo ASM .Si hasta ese momento no existen errores , va al ensamblador .

El ensamblador genera su propio juego de archivos .Estos incluyen el archivo final .HEX y como opcionales , archivos de listado y depuración .

6.4. ESTRUCTURA DEL ARCHIVO .ASM

El archivo .ASM tiene su estructura propia . Todo debe ser hecho en un orden determinado para que el trabajo funcione correctamente .

El primer elemento colocado en el archivo es la definición de que ensamblador se usará , seguida por un INCLUDE para indicarle al ensamblador que microprocesador es el destino y darle alguna información básica , como los datos de configuración .

Después se listan todas las direcciones de variables y alias . Si se solicita , sigue la inicialización del EEPROM .

Luego se coloca un INCLUDE para el archivo de macros ,seguido por un INCLUDE para las librerías de subrutinas .

Finalmente se coloca el código del programa .Este código es simplemente una lista de las macros que fueron generadas a partir de las líneas PBP

7. OTRAS CONSIDERACIONES PBP

7.1. CUAN RAPIDO ES SUFICIENTEMENTE RAPIDO ?

Por defecto PBP genera programas sobre la base de un microPIC con un cristal de 4 Mhz ó un resonador cerámico .

Todas las instrucciones sensibles al tiempo , asumen un tiempo de instrucción de 1 microsegundo para sus demoras .Esto permite a PAUSE 1000 , por ejemplo , esperar 1 segundo y a los comandos SERIN y SEROUT , baud rates exactos .

Sin embargo , puede ser útil hacer funcionar al microPIC a otra frecuencia distinta de 4 Mhz .Aunque los programas compilados son rápidos , es mejor hacerlos funcionar más rápido . O quizás se desea entrada y salida serial con un baud rate de 19200 baud en lugar de usar el tope común de 9600 baud.

Los programas PBP pueden funcionar a frecuencias de clock distintas de 4 Mhz de dos maneras .La primera , es simplemente usar un oscilador de frecuencia distinta de 4 Mhz y no indicarselo a PBP .Esta es una técnica útil si usted prestó atención a lo que sucede con las instrucciones dependientes del tiempo .

Si desea hacer funcionar el bus serial a 19200 baud ,simplemente cambie el cristal de 4 Mhz ,por uno de 8 Mhz .Esto , en efecto , hace funcionar todo dos veces más rápido , incluyendo los comandos SERIN y SEROUT .Si le indica a los comandos SERIN y SEROUT que operen a 9600 baud , doblando la velocidad del oscilador , funcionarán a 19200 baud .

Sin embargo ,tenga en cuenta que comandos como PAUSE y SOUND se ejecutarán dos veces más rápido. PAUSE 1000 sólo esperará medio segundo con un cristal de 8 Mhz antes de permitir la continuación del programa .

La otra manera es usar una frecuencia de oscilador diferente e indicarselo al PBP .Esto se hace usando DEFINE . DEFINE , como se demostró con el comando LCDOUT anteriormente , se usa para indicarle a PBP que debe usar utros parámetros que no son los usados por defecto .
Normalmente , por defecto PBP usa un oscilador de 4 Mhz .Agregando la declaración :

DEFINE OSC 8

Cerca del comienzo del programa PBP , se asume que se usará un oscilador de 8 Mhz .Las definiciones aceptables son :

OSC	Oscilador usado
3	3.58 Mhz
4	4 Mhz
8	8 Mhz
10	10 Mhz
12	12 Mhz
16	16 Mhz
20	20 Mhz

Indicando a PBP la frecuencia del oscilador se le permite compensar y producir los tiempos correctos para COUNT ,DEBUG ,DTMFOUT ,FREQOUT ,HSERIN ,HSEROUT ,I2CREAD ,I2CWRITE

,LCDOUT ,PAUSE ,PAUSEUS ,SERIN ,SERIN2 ,SEROUT , SEROUT2 , SHIFTIN ,SHIFTOUT ,SOUND ,XIN y XOUT .

Cambiando la frecuencia del oscilador puede ser usado para mejorar la resolución de las instrucciones PULSIN ,PULSOUT y RCTIME .A 4 Mhz ,operan con una resolución de 10 microsegundos .Si se usa un cristal de 20 Mhz , la resolución será de 2 microsegundos .Pero , el ancho del pulso es medido en una variable de 16 bit .Con una resolución de 2 microsegundos , el máximo ancho de pulso medible será de 131070 microsegundos .

Yendo en la otra dirección y utilizando un oscilador de 32768 Khz es problemático .Puede ser deseable , si se desea reducir el consumo de potencia .Los comandos SERIN y SEROUT son inutilizables ,y el WatchDog Timer puede hacer que el programa recomience sólo en cualquier momento .Experimente si su aplicación funciona con esta velocidad del oscilador .

7.2. VALORES PREDETERMINADOS (SETEOS)DE CONFIGURACION

Como se mencionó anteriormente , los valores por defecto de configuración para un dispositivo en particular ,están en el archivo .INC con el mismo nombre del dispositivo .Estos valores pueden ser cambiados en el mismo momento en que el dispositivo es programado .

El oscilador por defecto es XT en muchos dispositivos -Este es el valor para un oscilador de 4 Mhz por defecto .Si se usa un oscilador más rapido , se debe cambiar a HS .Dispositivos con oscilador interno usan INTRC .

El WatchDog Timer es habilitado por PBP .Se usa , junto con el prescaler TMR0 , en las instrucciones NAP y SLEEP .Si estas instrucciones no se usan en el programa ,puede ser deshabilitado y el prescaler usado para otra función .

La protección de código Code Protect está habilitada por defecto , pero puede deshabilitarse cuando el dispositivo se está programando físicamente .
Vea la hoja de datos de Microchip para cada dispositivo en particular .

7.3. USO DE RAM

En general ,no es necesario conocer como PBP distribuye la memoria RAM .PBP se encarga de todos los detalles para que el programador no lo tenga que hacer .Sin embargo hay momentos en que es útil saber como lo hace .

Las variables son guardadas en los registros RAM del microPIC .La primer dirección disponible es $0C para el micro PIC16F84 y algunos de los microPIC menores ,y $20 para el PIC16C74 y otros microPIC mayores .Refiérase al manual de Microchip , para determinar la dirección de comienzo de los registros de cada micro ciontrolador .

Las variables son asignadas en memoria secuencialmente y en un orden particular .Primero los array de word (si existen) ,seguidos por los arrays de byte y bit .Luego se posicionan los word y bytes y finalmente los bits individuales .Los bits se empaquetan en bytes si es posible .Este orden brinda el mejor aprovechamiento de la memoria disponible .

Los arrays deben estar dentro de un banco .No deben cruzar los límites de un banco .Esto limita el tamaño de un array individual .Vea la sección previa sobre arrays para conocer estos límites .
Se puede sugerir a PBP en que banco debe colocar una variable :

```
penny VAR WORD BANK0
nickel VAR WORD BANK1
```

Si se hacen solicitudes de un banco específico , éstos son manejados primero .Si no existe lugar dentro del banco solicitado , se usa el primer espacio disponible y se avisa lo sucedido .

También se pueden solicitar direcciones específicas para las variables .En muchos casos , es mejor dejar que PBP maneje la distribución de memoria por usted .Pero en algunos casos ,como el manejo del registro W durante un handler de interrupciones es necesario definir una dirección fija .Esto puede ser hecho de una manera similar al direccionamiento de un banco :

W_store VAR BYTE $20

Varias variables de sistema ,usando 24 bytes de memoria ,son automáticamente distribuídas por el compilador para el uso de las subrutinas de librería .Estas variables están en el archivo PBPPIC14.RAM y deben estar en banco 0 .

Las variables de usuario usan el símbolo (_) , mientras que las variables de sistema no lo hacen ,por lo que no interfieren entre sí .

Las variables B0-B25 y W0-W12 de BASIC Stamp no son distribuídas automáticamente .Es mejor crear sus propias variables usando la instrucción VAR .Sin embargo , si desea que se creen estas variables, simplemente incluya el archivo apropiado , BS1DEFS.BAS o BS2DEFS.BAS al comienzo del programa PBP .Estas variables se colocan en un espacio separado y aparte de las demás variables que usted pueda crear posteriormente .Es diferente que en BS2 donde usar variables pre-confeccionadas y variables de usuario ocasiona problemas .

El compilador puede crear automáticamente variables adicionales temporarias ,para ayudar a resolver ecuaciones .Una lista de estas variables ,así como el mapa de memoria completo , están en los archivos generados .ASM y .LST .

7.4. PALABRAS RESERVADAS

Palabras reservadas son aquellas que son usadas por el compilador y no pueden ser definidas como nombres de variable ni etiqueta .Estas palabras reservadas pueden ser nombres de comandos ,pseudo-operaciones ,tipos de variables o los nombres de los registros del microPIC .

Las pseudo-operaciones , tipos de variables y palabras clave de comandos son listadas en cada sección .Los nombres de los registros del microPIC son definidos en el archivo PIC14EXT.BAS .Si se incluyen los archivos BS1DEFS.BAS y BS2DEFS.BAS ,las definiciones dentro de ellos , se convierten en palabras reservadas y no deben ser redefinidas .

7.5. VIDA DESPUES DE 2K

Si , hay vida después de 2K usando el compilador PBP .
Los microPIC tienen un espacio de código segmentado .Las instrucciones microPIC ,como Call y Goto ya tienen suficiente código como para llenar 2 K de espacio de programa .Para llegar al código que está fuera del límite de 2 K ,el registro PCLATH debe ser activado antes de cada Call ó Goto .

PBP automáticamente setea estos bits PCLATH por usted .Sin embargo , hay algunas restricciones . La librería PBP debe entrar completa dentro de la página 0 del espacio de código .Normalmente ,no es problema ,ya que la librería es el primer elemento en un programa PBP y la librería completa es menor de 2 K .Sin embargo , se debe prestar atención a esto ,si se usan librerías adicionales .

Los handler de interrupción de lenguaje ensamblador también deben entrar en la página 0 del espacio de código .Esto se logra colocándolos al principio del programa PBP .Vea la próxima sección sobre lenguaje ensamblador para mayor información .

Agregar instrucciones para setear los bits PCLATH agrega overhead al código generado .PBP va a setear los bits de PCLATH para cualquier código que cruce el límite de 2 K ó para cualquier referencia en los microPIC con más de 2 K de espacio de código .

Hay instrucciones PBP específicas para ayudar al uso de más de 2 K .
BRANCHL se creó para permitir saltos a etiquetas que están del otro lado del límite de 2 K .Si el microPIC tiene 2 K ó menos de espacio de código ,se debe usar BRANCH , ya que ocupa menos espacio que BRANCHL .Si el micro controlador tiene más de 2 K de espacio de código ,y no está seguro de que los saltos siempre serán en la misma página , use BRANCHL .
El ensamblador puede enviar un aviso acerca de que el límite de página ha sido cruzado .Esto es normal y es aconsejable que usted controle por cualquier BRANCH que cruce el límite de página .

8. PROGRAMACION EN LENGUAJE ENSAMBLADOR

Las rutinas en lenguaje ensamblador pueden ser una útil ayuda para un programa de PBP Compiler . Aunque muchas tareas pueden ser hechas completamente en PBP .hay casos en que puede ser necesario un trabajo más rápido ,ó usar un espacio de código más pequeño ó diferente a como lo hace el compilador. En esos momentos es útil tener las posibilidades de un ensamblador en línea .

Puede ser beneficioso escribir rápidamente un programa usando PBP y luego colocar unas pocas líneas de código ensamblador para aumentar su funcionalidad .Estos códigos adicionales pueden ser insertados directamente en el programa PBP ó incluídos en otro archivo .

8.1. DOS ENSAMBLADORES - SIN DEMORA

PBP primero compila el programa en lenguaje ensamblado y luego comienza automáticamente un ensamblador .Esto convierte la salida del ensamblador en el archivo final .HEX que puede ser programado dentro de un micro controlador .

Se puede usar dos ensambladores distintos con PBP : PM , nuestro PICmicro Macro Assembler , y MPASM ,el ensamblador de Microchip .PM está incluído con el compilador mientras que MPASM debe ser obtenido directamente de Microchip ,vía Internet ó está incluído en sus programadores microPic .

Hay beneficios y contras en el uso de cada uno de ellos .PM está a mano ,porque está incluído vomo parte de PBP .Es más rápido que MPASM y puede ensamblar programas más grandes en DOS .PM incluye un juego de instrucciones estilo 8501 que es más intuitivo que los mnemónicos usados por MPASM . Información completa acerca del PICmicro Macro Assembler vea el archivo PM.TXT en el disco.

MPASM , tiene la posibilidad de crear un archivo .COD .Este archivo contiene información adicional que puede ser muy útil con simuladores y emuladores .MPASM es también más compatible con la amplia variedad de ejemplos en lenguaje ensamblador que hay en Internet , y en los manuales de Microchip .

PBP por defecto usa PM .Para usar MPASM ,simplemente copie todos los archivos MPASM en su propio subdirectorio ,probablemente llamado MPASM .Este subdirectorio también debe estar en la ruta (PATH) de DOS .

MPASM se puede usar de dos maneras .Si se usa la opción de línea de comando “-ampasm”, MPASM será comenzado siguiendo a la compilación para completar el proceso .MPASM muestra su propia pantalla con el progreso de la tarea .

PBP -ampasm filename

En forma alternativa , la opción de línea de comando “-amp” comenzará MPASM en modo silencioso y solo mostrará si hay errores .Sin embargo el disparador consume memoria adicional que no está disponible para MPASM .

PBP -amp filename

Para disponer del máximo de memoria para MPASM , se debe usar la opción “- ampasm” ó la versión Windows de MPASM .
En cualquier caso , MPASM no está incluida en PBP y debe ser obtenida de Microchip .

8.2. PROGRAMANDO EN LENGUAJE ENSAMBLADOR

Los programa s PBP pueden una línea simple de lenguaje ensamblador precedida por un símbolo (@) ,ó una ó más líneas de código ensamblador precedidas por la palabra clave ASM y terminada por la palabra clave ENDASM . Ambas , aparecen solas en sus respectivas líneas .

```
@ bsf PORTA, 0

Asm
bsf STATUS,RP0
bcf TRISA, 0
bcf STATUS,RP0
Endasm
```

Las líneas de ensamblador son copiadas exactamente dentro del archivo ensamblador de salida .Esto le permite al programa PBP usar todas las ventajas de PM .Sin embargo ,requiere que el programador esté familiarizado con las librerías PBP . Las convenciones PBP son similares a la de otros compiladores comerciales y no serían problema para un programador experimentado .

Todos los nombres identificatorios definidos en un programa PBP son definidos en forma similar en el ensamblador , pero precedidos por un (_) Esto permite el acceso a variables de usuario , constantes y aún a direcciones etiquetadas en ensamblador .

Cualquier nombre definido en ensamblador que comience con un (_) tiene la posibilidad de conflictuar con un símbolo generado por PBP . ¿ Si se evitan los conflictos , se puede acceder a estos valores marcados por el ensamblador , desde PBP ? . No .Recuerde que los nombres con (_) generados por PBP son solo imágenes de la información actual definida en el compilador .Como el ensamblado en línea es copiado directamente al archivo de salida y no es procesado por el compilador , éste ignora cualquier información de tipo ó valor de los símbolos ensambladores .Si las variables son compartidas por el ensamblador y PBP ,deben ser definidas en PBP .

También pueden tener conflictos , símbolos que no comiencen con (_) .El problema son los identificadores internos de las librerías .Afortunadamente , muchos de ellos contienen un “?” ó hacen referencia a uno de los registros de trabajo (como R0) .Evitar esos nombres es evitar problemas .Si usted tiene un conflicto de nombres ,el compilador reporta las definiciones duplicadas como un error .
En lenguaje ensamblador el designador de comentarios cambia del (´) de PBP a (:) .

```
´ comentario de PBP
: comentario de lenguaje ensamblador
```

8.3. UBICACIÓN DEL ENSAMBLADOR EN LINEA

Las declaraciones PBP se ejecutan según su orden de aparición en el programa fuente .El código se organiza de la siguiente manera : Comenzando en la ubicación 0 , vector de reset ,PBP inserta un código de arranque seguido por un salto a INIT .Después se guardan las librerías de subrutina utilizadas .Al final de las librerías está INIT , dónde se completa una inicialización adicional .Finalmente ,en la etiqueta MAIN , se agrega el código compilado PBP

La primer línea ejecutable que aparece en el fuente PBP es donde se empieza a ejecutar el programa .Esa declaración aparece después del arranque del controlador , las librerías de código y la etiqueta MAIN .

La tendencia de los programadores es a ubicar sus propias librerías usando el ensamblador en línea , antes ó después de su código . Esto puede crear problemas obvios .Si aparecen antes , las rutinas ensambladoras se ejecutan antes que cualquier instrucción PBP (algunos programadores invariablemente aprovechan esta opción) .Si aparecen en la cola del programa la ejecución que “falla cuando está por terminar “ puede encontrarse ejecutando rutinas inesperadas.

Hay un par de factores decisivos acerca de cual puede ser el mejor lugar para insertar subrutinas en lenguaje ensamblador .Si todo el programa entra en 2 K (una página de código) ,coloque sus rutinas ensambladoras después del código PBP .Si necesita terminar su programa , explícitamente coloque un END ó STOP al final de su código , en lugar de dejarlo flotando en el limbo .

Si el programa es mayor a 2 K , tiene más sentido colocar las rutinas en lenguaje ensamblador en el comienzo del programa PBP .Esto las coloca en la primera página y usted sabrá donde hallarlas .Esta es la forma de colocar las rutinas de interrupción en lenguaje ensamblador.

Si las rutinas están ubicadas al principio ,debe incluir un GOTO (ó JMP) rodeando el código de la primer declaración ejecutable PBP .Vea la sección de interrupciones .

El código para las rutinas de lenguaje ensamblador puede ser incluído en su programa ó en un archivo separado .Si una rutina es usada solo por un programa en particular , tiene sentido incluir el código ensamblador dentro del archivo fuente PBP .A esta rutina se puede acceder usando el comando CALL . Si es usada por varios programas ,se puede incluir un archivo separado ,conteniendo las rutinas en lenguaje ensamblador , en el lugar apropiado en el fuente PBP .

Asm

Include “myasm.inc”
Endasm

8.3. OTRO TOPICO ACERCA DEL ENSAMBLADOR

Los registros del microPIC están ordenados en bancos .PBP sabe a que banco de registros está apuntando en todo momento .Sabe que si está apuntando a un registro TRIS ,necesita cambiar los bits de selección de banco antes de acceder a un port .

Sabe también que debe poner en cero los bits de selección de banco antes de efectuar un Call ó un Jump .Lo hace así , porque no puede saber el estado de los bits de selección de banco de la nueva ubicación . Así , cada vez que hay un cambio de ubicación ,ó una etiqueta que puede ser llamada ó a la cual saltar , pone en cero los bits de selección de banco .

También pone en cero los bits de selección de banco antes de cada ASM y atajo @ de ensamblador .Una vez más ,la rutina ensambladora no sabe el estado actual de los bits ,por lo que son colocados en un estado conocido .El código ensamblador debe estar seguro de poner en cero los bits de selección de banco antes de salir , si los ha alterado .

9. INTERRUPCIONES

Las interrupciones pueden ser una forma de hacer que su programa sea realmente difícil de depurar . Las interrupciones son disparadas por eventos de hardware , ya sea un pin de I/O cambiando su estado ó un tiempo terminado ó cualquier otro .Si está habilitada (por defecto no lo está) , la interrupción causa que el procesador detenga lo que está haciendo y salte a una rutina específica en el micro controlador , llamada handler de interrupciones .

Pueden ser dificiles de implementar adecuadamente , pero también pueden proveer funciones muy útiles . Por ejemplo , una interrupción puede ser usada para acumular datos seriales de entrada ,mientras el programa principal está haciendo otra tarea .
(este uso particular requiere un micro controlador con un port serial) .

Hay muchas formas de evitar usar las interrupciones .Un polling rápido de un pin ó un bit de registro hace el mismo trabajo en forma rápida .O se puede verificar el valor de una bandera de interrupción sin tener que habilitarlo .

Sin embargo ,si usted desea hacerlo , le damos algunas ideas de cómo hacerlo .
El compilador PBP tiene dos mecanismos diferentes para manejar interrupciones .La primera es simplemente escribir el handler de interrupción en ensamblador y colocarlo en el frente de un programa PBP .El segundo método es usar la declaración ON INTERRUPT .Cada método será explicado por separado ,después de explicar las interrupciones en general .

9.1. INTERRUPCIONES EN GENERAL

Cuando ocurre una interrupción , el microPIC guarda la dirección de la próxima instrucción que debería ejecutar en el stack (pila) y salta a la dirección 4 .Esto significa que se necesita una dirección extra en el stack de hardware , que solamente tiene 8 .

Las librerías de rutinas de PBP pueden usar hasta 4 direcciones del stack ,ellas solas .Las 4 restantes están reservadas para CALLs y GOSUBs anidados .Debe asegurarse de que sus GOSUB no estén anidados en más de tres niveles , y sin CALL dentro de ellos , para tener una dirección de stack disponible para la dirección de regreso Si su handler de interrupciones usa el stack (haciendo un CALL ó un GOSUB ,p.ej.) necesitará espacio adicional disponible en el stack .

Después , usted necesita habilitar las interrupciones apropiadas .Eso significa dar valores al registro INTCON .Setee los bits de habilitación necesarios junto con el Global Interupt Enable .Por ejemplo ;

INTCON = %10010000

habilita la interrupción para RB0/INT .Dependiendo de la interrupción deseada , puede necesitar setear el registro PIE .

Refiérase a los manuales de Microchip para información adicional acerca de cómo usar las interrupciones. Hay ejemplos del contexto general del procesador y de cómo habilitar una interrupción en particular .

9.2. INTERRUPCIONES EN BASIC

La forma más fácil de escribir un handler de interrupción ,es escribirlo en PBP junto a una declaración ON INTERRUPT .ON INTERRUPT le indica a PBP que active su handler interno de interrupción (el de PBP) y salte tan pronto pueda a su handler de interrupción BASIC (creado por el usuario) después de reicibir una interrupción .

Usando ON INTERRUPT , cuando ocurre una interrupción ,PBP simplemente marca el evento y vuelve a la tarea que estaba realizando . No salta inmediatamente al handler .Como las declaraciones de PBP no son re-entrantes (PBP debe terminar con la declaración en curso antes de ejecutar otra) puede haber considerable demora (latencia) antes de manejar a la interrupción .

Como ejemplo ,digamos que el programa PBP recién comenzó la ejecución de PAUSE 10000 cuando ocurre una interrupción .PBP marca la interrupción y continúa con el PAUSE , pueden transcurrir 10 segundos antes de que se ejecute el handler de interrupción .Si se están acumulando caracteres en un port serial , muchos de ellos pueden perderse .

Para minimizar este problema ,use declaraciones que no tomen mucho tiempo de ejecución .Por ejemplo , en lugar de PAUSE 1000 use PAUSE 1 dentro de un loop FOR...NEXT .Esto permite completar cada declaración más rápidamente y manejar cualquier interrupción pendiente .
Si se necesita un proceso de interrupción más rápido que el provisto por ON INTERRUPT , se deben usar interrupciones en lenguaje ensamblador .

Lo que sucede cuando se usa ON INTERRUPT es lo siguiente :un corto handler de interrupción es colocado en la dirección 4 del microPIC .Este handler de interrupción es simplemente un RETURN . Esto envía el programa de vuelta a lo que estaba haciendo antes de ocurrir la interrupción .No requiere guardar

ningún contexto del procesador .Lo que esto no hace es re-habilitar el Global Interrupts , como hace el Retfie .
Un Call a una pequeña subrutina es colocado después de cada declaración en el programa PBP cuando se encuentra un ON INTERRUPT .Esta pequeña subrutina verifica el estado del bit de Global Interrupt Enable .si está apagado hay una interrupción pendiente ,por lo que apunta al handler de interrupción del usuario ..Si no el programa continúa con ña próxima declaración BASIC , después de lo cual se verifica nuevamente el bit GIE , y así sucesivamente .

Cuando se encuentra una declaración RESUME después del handler de interrupción BASIC , setea el bit GIE para rehabilitar las interrupciones y vuelve donde estaba el programa cuando ocurrió la interrupción .Si se indica en RESUME una etiqueta hacia donde saltar ,la ejecución continuará en esa dirección .En ese caso , se pierden todas las direcciones previas de regreso .
DISABLE detiene PBP insertando un Call al verificador de interrupción después de cada declaración .Esto permite que se ejecuten secciones de código sin la posibilidad de ser interrumpidas .ENABLE permite la inserción para continuar .

DISABLE debe ser colocado antes que el handler de interrupción para que éste no sea arrancado cada vez que se chequee el bit GIE .

Si por alguna razón se desea apagar las interrupciones después que se encuentra un ON INTERRUPT ,no debe apagar el bit GIE .Apagando este bit , se le indica a PBP que ha sucedido una interrupción y esto ejecutará el handler de interrupción por siempre .En su lugar haga :

INTCON = $80

Esto deshabilita todas las interrupciones individuales , pero deja seteado el bit GIE.
Una nota final acerca de las interrupciones en BASIC .Si rl programa usa :

loop: goto loop

y espera ser interrumpido ,eso no sucederá .Recuerde que la bandera de interrupción es chequeada después de cada instrucción .Realmente no hay un lugar donde chequear después de un GOTO . Inmediatamente salta al loop , sin chequear la interrupción .Debe colocarse alguna declaración dentro del loop , para que haya un chequeo de interrupciones .

9.3. INTERRUPCIONES EN ENSAMBLADOR

Las interrupciones en lenguaje ensamblador son un poco más complicadas .
Como usted no tiene idea acerca de la tarea del procesador cuando fue interrumpido ,no sabe el estado del registro W , ni las banderas de STATUS , ni PCLATH , ni siquiera a que página estaba apuntando . Si necesita alterar alguno de ellos , y es probable que así sea , debe guardar los valores encontrados , para restaurarlos antes de hacer que el procesador vuelva a la tarea que estaba cumpliendo antes de ser interrumpido .A esto se le llama guardar y restaurar el contexto del procesador .

Si el contexto del procesador al volver de la interrupción no es exactamente igual al que tenía antes de la misma , pueden suceder todo tipo de fallas menores y mayores .

Esto , por supuesto , significa que usted ni siquiera puede usar en forma segura las variables del compilador para guardar el contexto del procesador .Usted no puede saber cuales son las variables usadas por las librerías en un momento determinado .
Debe crear variables en el programa PBP con el expreso propósito de guardar W , el registro STATUS y cualquier otro registro que pueda alterar el handler de interrupción .Estas variables no pueden ser usadas por el programa BASIC .

Aunque parece simple guardar W en un registro RAM , en realidad es bastante complicado .El problema es que usted no sabe a que banco de registros está apuntando cuando ocurre la interrupción .Si usted reservó un lugar en el banco 0 y el apuntador de registro corriente está apuntando a banco 1 ,por ejemplo, puede sobreescribir una dirección no deseada .Por lo tanto ,debe reservar una ubicación en cada banco de registros RAM, con la misma dirección .

Como ejemplo , elijamos el 16C74(A) .Tiene dos bancos de registros RAM comenzando en $20 y $A0 respectivamente .Para estar seguros ,debemos reservar la misma dirección en ambos bancos .En este caso , elegimos la primera ubicación de cada banco .El comando VAR nos permite hacerlo :

```
w_save var byte $20 system
w_save1 var byte $a0 system
```

Esto le dice al compilador que coloque la variable en una ubicación particular de RAM De esta manera , la grabación de W no va a corromper datos .

La rutina de interrupción debe ser tan corta y rápida como sea posible .Si demora mucho en ejecutarse , puede terminar el tiempo del WatchDog Timer , lo que arruinaría todo .

La rutina debe terminar con una instrucción Retfie para volver desde la interrupción y permitir al procesador retomar el programa PBP donde lo había abandonado .

El mejor lugar para colocar el handler de interrupción en lenguaje ensamblador es probablemente el comienzo exacto del programa PBP .Esto asegura que está ubicado dentro de los primeros 2 K para minimizar problemas de límites .Se debe insertar un GOTO previo , para asegurarse de que no se ejecute al comenzar el programa .Vea el ejemplo siguiente .

Si el microPIC tiene más de 2 K de espacio de código , existirá un problema con el vector de interrupción ,,ubicación 4 .Este código es un simple GOTO al handler de interrupción .El problema puede ocurrir si PCLATH apunta a la página de código incorrecta cuando ocurre una interrupción , cuando hay más de una página de código (más de 2 K) .Usted necesita una porción de código mucho más sofisticada para guardar los registros y setear PCLATH antes del salto al handler de interrupción .También se debe resetear cuando se termina el handler de interrupción.

Además debe informar a PBP que está usando un handler de interrupción en lenguaje ensamblador y donde encontrarlo .Esto se logra con un DEFINE :

```
DEFINE INTHAND Label
```

Label es el comienzo de su rutina de interrupción .PBP coloca un salto a este Label en la ubicación 4 del microPIC .

```
´ ejemplo de interrupción en lenguaje ensamblador

led var PORTB.1

w_temp var byte $20 system
s_temp var byte bank0 system

goto start ´ saltee el handler de interrupción

´ defina el handler de interrupción
define INTHAND myint

´ handler de interrupción en lenguaje ensamblador asm
; guarda W y STATUS
myint movwf w_temp
swapf STATUS,W
clrf STATUS
movwf s_temp

; inserte aquí el código de interrupción
; guarda y restaura PCLATH y FSR si los usó

bsf LED ; enciende LED

; restaura STATUS y W
swapf s_temp,W
movwf STATUS
swapf w_temp,F
swapf w_temp.W
retfie
endasm

´ el programa PBP comienza aquí
start : low LED ´ apaga el Led

´ habilita la interrupción en PORTB.0
INTCON = %10010000

Loop: goto loop ´espera por una interrupción
```

10. DIFERENCIAS ENTRE PBP / PIC BASIC/ STAMP

La compatibilidad es una espada de dos filos .Y además tiene punta . PBP ha hecho algunas concesiones a la facilidad de uso y tamaño de código .Por lo tanto ,lo llamamos similar a BASIC Stamp y no BASIC Stamp compatible .PBP tiene mucho de las instrucciones y la sintaxis de BASIC Stamp I y II .Sin embargo hay algunas diferencias significativas .

Las siguientes secciones discuten detalles de implementación de programas PBP que pueden ocasionar problemas .Deseamos que si encuentra problemas , esto le sirva para solucionarlos .

10.1 VELOCIDAD DE EJECUCION

El mayor problema potencial es la velocidad .Sin necesidad de leer instrucciones desde el EEPROM , muchas instrucciones de PBP se ejecutan cientos de veces más rapidamente que sus equivalentes BASIC Stamp .Aunque en muchos casos es un beneficio , programas cuyo tiempo ha sido determinado empíricamente pueden experimentar problemas .

La solución es simple , los buenos programas no deben depender del tiempo de las declaraciones .como los loop FOR...NEXT .Siempre que sea posible , los programas deben incluir métodos de sincronización no temporal .Si se necesitan demoras ,se debe usar declaraciones especificas (NAP , SLEEP , PAUSE , PAUSEUS).

10.2. I / O DIGITAL

Los programas PBP operan directamente en los registros PORT y TRIS .Aunque esto da ventajas de velocidad y tamaño de RAM / ROM , también tiene sus contras.

Algunos comandos de I / O (TOGGLE , PULSEOUT ,..) efectúan operaciones de leer - modificar - grabar ,directamente en el registro del PORT .Si dos operaciones de este tipo , se realizan en un lapso de tiempo muy corto , y la salida está manejando una carga inductiva ó capacitiva , la operación puede fallar . Suponga que un parlante está conectado a través de un capacitor de 10 uF (lo que sucede con el comando SOUND) .También suponga que el pin esta inicialmente bajo y el programador intenta generar un pulso usando TOGGLE .El primer comando lee el estado bajo del pin y envía su complemento .El driver de salida (que ahora está alto) empieza a cargar el capacitor .Si la segunda operación se efectúa demasiado rápido , lee el nivel bajo del pin aunque el driver de salida esté alto .Así , la segunda operación llevará el pin a alto .

En la práctica , esto no es un problema Estos comandos diseñados para este yipo de interfase (SOUND, POT , etc) tienen protección incluida .Este problema no es específico de los programas PBP , sino común a los programas para microPIC (y otros controladores) y es una de las realidades de programar directamente el hardware .

10.3. INSTRUCCIONES DE BAJA POTENCIA

Cuando el WatchDog Timer despierta al microPIC del modo de sueño (SLEEP) ,la operación recomienza sin modificar el estado de los pins de I/O .Por razones desconocidas , cuando BASIC Stamp recomienza la ejecución después de una instrucción de baja potencia (NAP ó SLEEP) los pins de I / O quedan con disturbios por aproximadamente 18 mseg .Los programas PBP hacen uso de la coherencia de los I / O de los microPIC.
NAP y SLEEP no alteran los pins de I / O.

10.4. PC , LA INTERFASE PERDIDA

Como los programas PBP corren directamente en un microPIC , no se necesitan los pins de interfase Sramp PC (PCO y PCI) . La falta de una interfase PC introduce algunas diferencias .

Sin el Stamp IDE corriendo en una PC ,no hay un lugar donde enviar la información de depuración .La depuración puede lograrse usando una instrucción de salida serial como DEBUG ó SEROUT junto con un programa terminal ejecutado en un PC (HyperTerm)

Sin el PC para despertar al microPIC de un END ó un STOP ,permanece inactivo hasta que el pin /MCLR se haga bajo ,ocurra una interrupción ó la energía se cicle

10.5. SIN VARIABLES AUTOMATICAS

El compilador PBP no crea variables en forma automática , como B0 ó W0 .Deben ser definidas usando VAR . Se entregan dos archivos : BS1DEFS.BAS y BS2DEFS.BAS que definen las variables standard BS1 y BS2 .Sin embargo ,es recomendable que usted asigne sus propias variables .

10.6. OPERADORES MATEMATICOS

Las operaciones matemáticas tienen precedencia de operación .Significa que no son evaluadas estrictamente de izquierda a derecha como están originalmente en BASIC Stamp y en el compilador PBP.Esta precedencia significa que multiplicación y división son realizadas antes que suma y resta , por ejemplo.
Se debe usar paréntesis para agrupar las operaciones en el orden en que deben ser realizadas
La tabla siguiente muestra el orden jerárquico de los operadores :

Mayor precedencia
()
NOT
~
-
SQR ABS DCD NCD COS SIN
*
**
*/
/
//
+
-
<<
>>
MIN
MAX
DIG
REV
&
^
I
& /
/ I
^ /
&& AND
^ ^ XOR

I I OR
Menor precedencia

10.8 [ ] VERSUS ( )

PBP usa corchetes , [ ] , en declaraciones donde se usaron previamente paréntesis ( ) .
Por ejemplo , el SEROUT de BS1 y del compilador PBP original era :

Serout 0,T2400, (B0)

La instrucción SEROUT del compilador PBP es :

Serout 0,T2400, [B0]

Cualquier instrucción que usara previamente paréntesis ,debe ser cambiada usando corchetes .

10.9. DATA,EEPROM,READ y WRITE

Basic Stamp permite que el EEPROM no usado para guardar programa sea usado para guardar datos . Como los programas PBP se ejecutan directamente desde el espacio de ROM del microPIC ,el almacenamiento en EEPROM debe implementarse de otra manera .

El PIC16F84 ,PIC16F83 y el PIC16C84 tienen 64 bytes de EEPROM .Los programas PBP pueden usar esto para operaciones de EEPROM y para soprtar los comandos DATA , EEPROM , READ y WRITE de Stamp.

Para acceder al almacenamiento de datos no volátiles , se usan las instrucciones I2CREAD y I2CWRITE . Estas instrucciones permiten comunicaciones bidireccionales con EEPROMs seriales como el 24LC01B .

10.10.DEPURACION

El depurador DEBUG en PBP no es un caso especial de SEROUT como en Stamp .Tiene su propia rutina que trabaja con un pin fijo y un baud rate .Puede usarse para enviar información de depuración a un terminal ó a un dispositivo serial .

Los signos de interrogación (?) en las declaraciones DEBUG se ignoran..No se debe usar el modificador ASC? .

10.11. GOSUB y RETURN

A través de GOSUB y RETURN se implementan subrutinas .La variable de usuario W6 es usada por BS1 como un stack de 4 nibble .Así ,los programas Stamp pueden hasta 16 GOSUB y las subrutinas pueden estar anidadas hasta cuatro niveles .

Los microPIC tienen instrucciones Call y Return , así como un stack de 8 niveles .Los programas hacen uso de estas instrucciones y pueden usar 4 niveles de este stack , mientras que los otros cuatro niveles están reservados para librerías .

Así , se dispone de W6 ,se puede anidar subrutinas hasta 4 niveles y el número de GOSUB solo está limitado por el espacio de código .

10.12. I2CREAD y I2CWRITE

Los comandos I2CREAD y I2CWRITE difieren de los I2CIN y I2COUT del original compilador PBP .La diferencia más obvia es que los números de pin de datos y clock ahora están especificados como parte del comando .No están fijos en pins específicos .

La otra diferencia es que el formato del byte de control ha cambiado .No se setea el tamaño de la dirección como parte del byte de control .En cambio , el tamaño de la dirección es determinado por el tipo de variable de dirección .Si se usa una variable de byte ,se envía una dirección de 8 bit .Si se envía una variable de word ,se envía una dirección de 16 bit .

10.13. IF...THEN

BASIC Stamp y el compilador PBP original solo permitían especificar un label después de un IF...THEN .PBP permite la construcción de IF...THEN...ELSE...ENDIF ,así como la ejecución de un código como resultado de un IF ó un ELSE .

10.14. MAX y MIN

Las funciones MAX y MIN han sido bastante modificadas respecto a las de Stamp y el compilador original PBP

MAX devuelve el máximo de dos valores .MIN devuelve el mínimo de dos valores .Es más parecido a los demás BASIC y no tiene los problemas de límites en 0 y 65535 del Stamp.
En la mayoría de los casos basta intercambiar MAX y MIN en los programas Stamp para que funcionen en PBP .

10.15. SERIN y SEROUT

SERIN y SEROUT usan sintaxis BS1 SERIN2 y SEROUT2 usan sintaxis BS2 .Se agregó un control de tiempo estilo BS2 a SERIN .

SERIN y SEROUT han sido alterados para funcionar hasta 9600 baud en lugar del límite de 2400 baud de BS1 .Esto se logró reemplazando el baud rate de 600 por el de 9600 .
Ahora se pueden usar Modes T9600 ,N9600 ,OT9600 ,ON9600.
600baud no está disponible y causará un error de compilación si se intenta usar .

10.16. SLEEP

El comando SLEEP está basado en el WatchDog Timer .No se calibra usando el oscilador .Este cambio fue necesario por el efecto que el reset del WatchDog Timer tenía sobre el microPIC .
Cada vez que el microPIC se reseteaba durante la calibración de SLEEP , se alteraban los estados de algunos de los registros internos .

Se decidió ejecutar SLEEP en modo sin calibración , basado solo en la exactitud del WatchDog Timer .Esto asegura la estabilidad de los registros y ports del microPIC .Sin embargo ,como el WatchDog Timer está controlado por un oscilador R/C interno , su período puede variar en forma signifixativa por temperatura y de chip a chip .Si se necesita mayor precisión puede usarse PAUSE , que no es un comando de baja potencia .

APENDICE A

SUMARIO DE INSTRUCCIONES ENSAMBLADOR

ADDLW	k
ADDWF	f,d
ANDLW	k
ANDWF	f,d
BCF	f,b
BSF	f,b
BTFSC	f,b
BTFSS	f,b
CALL	k
CLRF	f
CLRW	
CLRWDT	
COMF	f,d
DECF	f,d
DECFSZ	f,d
GOTO	k
INCF	f,d
INCFSZ	f,d

IORLW	k
IORWF	f,d
MOVF	f,d
MOVLW	k
MOVWF	f
NOP	
RETFIE	
RETLW	k
RETURN	
RLF	f,d
RRF	f,d
SLEEP	
SUBLW	k
SUBWF	f,d
SWAPF	f,d
XORLW	k
XORWF	f,d

b - direccion de bit
d - destino ; 0=w ,1=f
f - direccion del archivo de registro
k - constante literal

ÍNDICE DE CONTENIDO: